БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Механико-математический факультет

Кафедра веб-технологий и компьютерного моделирования

СОГЛАСОВАНО
Председатель учебно-методической комиссии механико-математического факультета
В. Г. Кротов
17 мая 2012 года

СОГЛАСОВАНО
Декан механико-математического факультета
Д. Г. Медведев
17 мая 2012 года

Регистрационный № УД 8854

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

Web-программирование

Для специальностей
1-31 03 01 – Математика (по направлениям)
1-31 03 01-05 – Математика (информационные технологии)

Составители: канд. физ.-мат. наук, доцент Романчик В.С.

Обсуждено и утверждено
Советом ММФ БГУ *15 мая 2012 года*,
протокол № 7

ОГЛАВЛЕНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Учебно-методический комплекс (УМК) преследует *цель* оказать помощь студентам в усвоении учебного и нормативного материала, сориентировать в подборе специальной литературы для подготовки к лабораторным и практическим занятиям, направить на развитие навыков самостоятельного решения практических задач

Содержание тем УМК структурировано с учетом вопросов, рассматриваемых на лекционных занятиях. Для подготовки к лабораторным и практическим занятиям, промежуточным и итоговым формам контроля следует исходить из содержания «Типовой учебной программы по дисциплине для высших учебных заведений по специальности 1-31 03 01 – Математика (по направлениям), 1-31 03 01-05 – Математика (информационные технологии), утвержденной Министерством образования Республики Беларусь 31 декабря 2008 г., рег. № ТД – G172/тип.», вопросов и заданий к практическим занятиям, размещенным в настоящем УМК.

Программа дисциплины «Web-программирование» ориентирована на студентов, обучающихся по специальности 1-31 03 01 – «Математика» по направлению: информационные технологии (1-31 03 01-05). Данная дисциплина изучается студентами указанного направления на младших курсах, что позволяет применять полученные знания в последующем обучении.

В соответствии с образовательными стандартами по указанным специальностям выпускник должен

знать:

- способы представления, поиска, передачи и хранения информации;
- методы решения научно-технических и информационных задач;
- современные информационные технологии;

уметь:

- решать типовые задачи математики и информатики;
- работать на современных вычислительных средствах;
- применять современные информационные технологии и методы реализации решения распределенных web-приложений.

В данном курсе рассмотрены базовые знания по представлению, организации и передаче информации и структуре Web, этапы разработки веб-сайта от планирования до рекламы, принципы гипертекста, основы языка HTML и спецификации CSS (Каскадных таблиц стилей), популярные принципы верстки сайтов, программы Dreamweaver, CorelDRAW, Adobe Photoshop, Flash. Скрипты на клиенте (JavaScript) позволяют сделать сайт динамическим, скрипты на сервере (PHP) позволяют реализовать распределенное web-приложение типа клиент-сервер.

Комплексная программа позволит слушателю стать квалифицированным разработчиком Web-сайтов, познакомит с технологиями проектирования сайтов и их оформления.

Объем курса – 204 часа. Из них аудиторных – 136 часов, в том числе: лекции – 68 часов, лабораторные занятия – 36 часов, практические занятия – 32 часа, самостоятельная работа – 68 часов.

КОНСПЕКТЫ ЛЕКЦИЙ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

Конспекты лекций соответствуют содержанию учебного материала и учебно-методической карте.

Содержание учебного материала

№ разделов	Наименование разделов	Аудиторные часы			
		Всего	Лекции	Лабораторные занятия	Практические занятия
1.	Введение в WWW.	2	2	0	0
2.	Язык разметки гипертекста HTML.	18	8	6	4
3.	Каскадные таблицы стилей. Введение в CSS	8	2	2	4
4.	Графический пакет CorelDRAW.	18	6	6	6
5.	Графический пакет PhotoShop	6	4	2	0
6.	Html-редактор Dreamweaver	4	2	2	0
7.	Adobe Flash. Adobe Flex. ActionScript	20	10	5	5
8.	Скрипты на клиентской странице. Язык JavaScript	16	8	3	5
9.	Создание серверных приложений. Язык PHP	28	10	10	8
10.	Введение в Базы Данных	2	2	0	0
11.	Технологии Java	2	2	0	0
12.	Технологии ASP.NET	2	2	0	0
13.	XHTML и XML	2	2	0	0
14.	Технологии Web 2.0	2	2	0	0
15.	Веб-сервисы	2	2	0	0
16.	Администрирование и безопасность	2	2	0	0
17.	Продвижение и оптимизация сайтов	2	2	0	0

∘ ∘∘∘∘	∘∘∘	∘∘	∘∘	∘∘
Итого	136	68	36	32

ГЛАВА 1. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА САЙТОВ

Создание сайта можно разделить на следующие основные этапы: проектирование и разработка дизайна главной и вторичных страниц, включая разработку клиентской части, создание скриптов, выполняемых на стороне сервера, создание баз данных, тестирование и продвижение.

Виды сайтов

Сайт — это набор из нескольких десятков, (сотен или тысяч) веб-страниц, связанных вместе единой темой, общим оформлением, взаимными гипертекстовыми ссылками и, близким размещением (обычно в пределах одного домена). Условно выделяют следующие виды сайтов:

1. *Личные страницы* (home pages) – до недавнего времени самая массовая категория веб-сайтов. На этих сайтах может быть все что угодно: информация о себе, о детях, о кошках, цветах, путешествиях. В настоящее время значительная часть такой информации перенесена в социальные сети и блоги.

2. *Сайт визитка* – Краткий и компактный сайт, содержащий общую информацию о компании её товарах и услугах, а также необходимые контактные данные. Сайт-визитка — небольшой сайт с максимально простой и удобной системой навигации. Сайт-визитка идеально подходит для небольших компаний, которые хотят разместить информацию о себе и своих услугах в интернете, и впервые столкнулись с задачей создания сайта. Следует иметь ввиду, что спустя какое-то время, из такого сайта возможно надо будет сделать сложную структуру, с возможностью решать дополнительные задачи.

Сайт-визитка имеет короткие сроки разработки и невысокую стоимость. Если сайт-визитка больше не удовлетворяет потребностям расширяющегося бизнеса, можно изменить его структуру до корпоративного сайта, Интернет-магазина или веб-портала.

3. *Некоммерческие* (академические) сайты — это сайты, принадлежащие всевозможным добровольным объединениям, временным проектам, международным или благотворительным организациям. К этой категории можно отнести и страницы научных центров, официальные сайты некоммерческих программных проектов а также организаций. Каждая страница некоммерческого сайта знает, что она хочет сказать своим читателям, и знает, зачем им это нужно. Результатом является логичное и последовательное оформление, сознательное использование академического стиля.

4. *Образовательные сайты*. К этой же категории можно отнести многочисленные страницы учебных заведений и университетов. Такие сайты содержат большие объемы образовательной информации.

5. *Информационные сайты* – сайт, который содержит исчерпывающую информацию по некоторой предметной области. Аналогично новостной сайт содержит информацию о последних событиях.

6. *Контент-сайты*. Контент-сайты являются, основными поставщиками информации и генераторами траффика. К этой категории относятся новостные сайты и колонки обозревателей, развлекательные и образовательные ресурсы.

Для сайтов этого типа содержание превосходит по важности оформление, так что средний уровень их дизайна заметно ниже, чем у сайтов компаний. Рассчитанные на многократные регулярные визиты, такие сайты стремятся свести оформительские элементы к минимуму, чтобы читатели не тратили драгоценное время на перекачивание графики и чтобы ничто не отвлекало их от главного, ради чего они пришли на сайт, — информации, содержания, контента.

7. *Промо-сайт* – это сайт, являющийся прямой рекламой товара или услуги.

8. *Интернет магазин* – сайт, предназначен для продажи товаров или услуг через

Интернет. Сайт содержит каталог продукции, прайс-листы, систему заказов. Интернет-магазин позволяет избежать множества дополнительных расходов на аренде помещений и складов. Продавец получает возможность взаимодействовать с пользователем в любое время суток, получая высокую рентабельность собственного бизнеса. Интернет-магазин — идеальное решение для малого бизнеса, работа которого может быть организована одним или несколькими сотрудниками. Магазин в Интернете максимально ориентирован на целевую аудиторию. Управление Интернет-магазином осуществляется с помощью удобной CMS-системы, что позволит оперативно самостоятельно менять содержимое каталога товаров несколькими щелчками мыши. Удобная система приёма, обработки заказов и удалённого взаимодействия с покупателями. Возможность отслеживания действий пользователя на сайте, контроля их деятельности и анализ эффективности работы Интернет-магазина.

Следует позаботится о том, чтобы каталог товаров содержал подробное описание и фотосъёмки товаров.

9. *Портал* – это очень большой веб-ресурс, который объединяет большое число различных Интернет-ресурсов в информационном пространстве компании, а также различные интерактивные службы.

10. *Корпоративный сайт*. Корпоративный сайт предоставляет клиентам полную информацию о компании: здесь размещаются сведения о предлагаемых товарах или услугах, описание политики компании, её истории, целей, особенностей, контактных сведений. Схемы работы корпоративного сайта следующая: Предоставление информации о компании на сайте > укрепление имиджа фирмы > увеличение продаж и прибыли. Большинство корпоративных сайтов содержит информацию не только о самой фирме, но и о ее продуктах.

11. *Сайты самих дизайнерских компаний* всегда интересны. Такой сайт обязан нравится потенциальным заказчикам и дизайнерам, отвечать жестким критериям, выработанным современной дизайн-культурой. Дизайнерам приходится делать выбор между «попсой» и «высоким искусством». Сайты веб-дизайнеров обычно немногословны - дизайн должен говорить сам за себя. На его первой странице часто не содержится вообще никакого текста, кроме названия фирмы и надписей на кнопках навигации (если же текст и есть, то чаще всего он нарочито загадочен, многозначителен и ни в коем случае не прямолинейно-рекламен). Вместо текста основное пространство страницы либо заполнено тщательно подготовленным, «ударным» визуалом или логотипом фирмы, либо просто оставлено пустым. Для дизайнерских сайтов всегда характерно активное использование пустоты, подчеркивающей их эстетский, антипрагматический стиль.

12. *Рекламные сайты*. В последнее время рекламный бизнес становится в Интернете все более прибыльным, и живущие за счет рекламы контент-сайты растут и множатся.

13. *Развлекательные сайты*. Содержат музыку, видео, игры.

14. *Сайты книг, фильмов, музыкальных альбомов и групп, спортивных команд* не несут в себе почти ничего коммерческого. Если принадлежат они не самим издателям, киностудиям или музыкантам, а независимым поклонникам то, уступая чисто рекламным сайтам в уровне дизайна, нередко выигрывают в полноте и качестве информации.

Этапы проектирования и разработки сайта

Этап 1. Определение основных целей создания сайта.

От правильно поставленных целей зависит успех при создании сайта.

Этап 2. Создание технического задания.

Разработка веб-сайта начинается с составления технического задания. Сначала может быть заполнен бриф, в котором заказчик излагает свои пожелания относительно

визуального представления и структуры сайта. Затем составляется техническое задание, в котором должны быть оговорены следующие вопросы:

- Тип сайта (портал, промо, визитка, корпоративный, магазин и т.д.)
- Функционал сайта (поиск, каталог, лента новостей и т.д.)
- Стиль дизайна (строгий, веселый, мрачный и т.д.)
- Структура сайта (какие страницы должны быть на сайте)
- Структура страниц (какие блоки должны быть на страницах)

Этап заканчивается после утверждения технического задания заказчиком.

Этап 3. Разработка дизайн – проекта

На этом этапе рассматриваются вопросы проектирования интерфейсов, юзабилити и многое другое. Этап делится на несколько подэтапов:

1. Набор идей дизайна предоставляется в виде эскизов с текстовыми пояснениями.
2. Разработка макета дизайна главной страницы.
3. Исправление замечаний заказчика, доработка макета.
4. Разработка внутренних страниц по аналогичному стилю.

Дизайнер создает в графическом редакторе отдельно дизайн главной страницы, и дизайны типовых страниц (новости, каталог продукции). Дизайн страницы представляет собой графический файл, слоеный рисунок, состоящий из мелких картинок-слоев элементов общего рисунка. Этап также заканчивается утверждением эскиза заказчиком.

Этап 4. *HTML-верстка*. Утверждённый дизайн передаётся html-верстальщику, который «нарезает» графическую картинку на отдельные рисунки, из которых впоследствии складывает html-страницу. В результате создаётся код, который можно просматривать с помощью браузера. А типовые страницы впоследствии будут использоваться как шаблоны.

Этап 5. – разработка программной части

Для начала нужно определиться с системой управления контентом (CMS), которая используется, когда разработка веб-сайта закончена и нужно его обслуживать. Разработка веб-сайта, в котором задействована CMS предоставит в дальнейшем простой способ добавления или редактирования информации на сайте. Можно использовать бесплатные CMS, платные программные разработки сторонних организаций или собственные разработки. Как минимум создаются две части сайта – клиентская и серверная.

Этап 6. – тестирование сайта

Основной целью тестирования является оценка и гарантия качества программного продукта. Тестирование направлено на исправление ошибок и неточностей, а также на повышение безопасности и удобства использования сайта. Веб-дизайн сайта должен адекватно выглядеть в различных браузерах, Internet Explorer, Firefox, Safari, Chrome и Opera.

Эта 7. Публикация сайта в Интернет

Сначала производится выбор и регистрация доменного имени. Затем осуществляется физическое размещение сайта на сервере провайдера - хостинг. После размещения производят нужные настройки сайта.

Этап 8. Заполнение сайта материалами

Сайт наполняют контентом — текстами, изображениями, файлами для скачивания и т. д. Тексты составляются либо специалистом студии, либо стороной заказчика.

Этап 9. Продвижение сайта и реклама в интернете

Раскрутка сайта это отдельный процесс, стоимость которого может превышать расходы на создание сайта. Для «раскрутки» сайта можно воспользоваться контекстной или баннерной рекламой, SEO оптимизацией. Внутренняя SEO-оптимизация связана с изменениями самого сайта. Начинается она с определения семантического ядра. Здесь

определяются такие ключевые слова, которые привлекут посетителей (например “поиск кладов”). Тексты, ссылки, другие теги адаптируются так, чтобы поисковые системы могли их успешно находить по ключевым словам. Внешняя SEO-оптимизация сводится, как правило, к построению структуры входящих ссылок.

Этап 10. Сдача проекта.

Заказчик рассматривает готовый проект и в случае, если все устраивает, то подписывает документы о сдаче проекта. На этом этапе производится также обучение представителя заказчика навыкам работы в администраторской зоне сайта.

Модели проектирования и управление проектами

Управление проектами - это скорей философия управления, а не инструмент или техника. Для любого проекта важно рассчитать ресурсы и затраты на разработку, качественно выполнить проект и в срок сдать заказчику. Для разработки и создания сайта существует несколько моделей проектирования, которые позволяют поэтапно реализовать проект от идеи до ее воплощения. Практически все модели содержат этапы:

1. Концептуальное проектирование.
2. Логическое проектирование.
3. Физическое проектирование.

Концептуальное проектирование служит для определения целей, задач сайта, а также аудитории, на которую он рассчитан. Универсальный критерий, который характеризует эффективность сайта - это достижение разработчиками сайта поставленных перед ними целей.

На этом этапе следует описать следующее:

Основные и второстепенные цели, состав пользователей, критерии достижения цели. С учетом поставленных целей, в итоге получается список сервисов и разделов, которые будут располагаться на сайте.

Логическое проектирование включает организацию информации на сайте, построение его структуры и навигации по разделам. На данном этапе следует решить вопрос, каким образом будет упорядочена информация: по времени, разделам, в алфавитном порядке, определенным группам или другим критериям. Одновременное использование различных способов охватывает большую аудиторию и позволяет быстрее найти нужную информацию на сайте.

На этом этапе следует описать следующее:

структура сайта (линейная, иерархическая, контекстная, другая);

названия и содержание разделов;

организация и связь разделов между собой;

какая информация будет размещена на определенных страницах сайта.

Результат логического проектирования оформляется в виде блок-схем, структурных диаграмм или другими способами.

Физическое проектирование – это этап, связанный, по большей части, с технической реализацией сайта. На этом этапе следует описать следующее:

Технологии, которые будут применяться на сайте.

Используемое программное обеспечение.

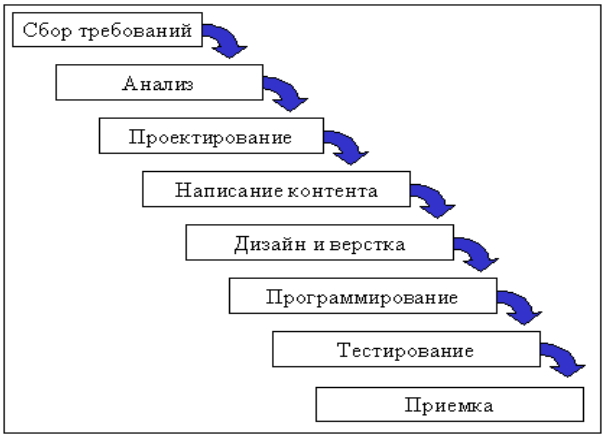
Возможные проблемы и способы их устранения.

Как будет обновляться информация.

Затем следует реализация программного обеспечения сайта.

Модель водопада управления проектами

В 1970 году в своей статье Ройс описал в виде концепции то, что сейчас принято называть «модель водопада», и обсуждал недостатки этой модели. В модели водопада следующие фазы шли в таком порядке:



Достоинством модели является ее простота и доступность. Все этапы идут последовательно, и каждый последующий не начинается, пока не закончится предыдущий.

К недостаткам относится следующее: модель предполагает точное знание того, что следует реализовать на сайте. На практике часто не представляется возможным сразу сформулировать цели, которые следует выполнить. Методику «модель водопада» часто критикуют за недостаточную гибкость и объявление самоцелью формальное управление проектом в ущерб срокам, стоимости и качеству. Однако, при управлении большими проектами формализация часто является большой ценностью, так как может снизить многие риски проекта и сделать его более прозрачным. Поэтому даже в PMBOK (Project Management Body of Knowledge) формально была закреплена только методика «каскадной модели» и не были предложены альтернативные варианты, известные как итеративное ведение проектов. Начиная с PMBOK 4-ой версии удалось достичь компромисса между методологами, приверженными формальному и поступательному управлению проектом, с методологами, делающими ставку на гибкие итеративные Agile-методологии (Scrum и Kanban).

Модель водопада (WaterFall) можно отнести к детерминированным процессам. На языке Водопада идет начальное обсуждение проекта на уровне заказчика, аналитика, CEO, финансового менеджера. Они должны понять сколько денег и времени нужно потратить на проект. Идет предварительный сбор требований, добавляются риск-факторы и получаются: X денег и Y дней.

Итерационная (спиральная) модель

Гибкая (Agile) методология представляет процесс, который применяет итерации разработки, командную работу, вовлечение владельцев, объективные метрики и эффективное управление.

Работа с проектом начинается с этапа «Планирование и анализ» и по часовой стрелке переходит к этапам выполнения, тестирования полученных результатов и

оценки. На следующей итерации все повторяется, но уже с учетом выявленных недочетов проекта. Таким образом, пройдя несколько итераций и повторив все этапы несколько раз, проект избавляется от недостатков, обрастая дополнительными возможностями и преимуществами.

Достоинством является возможность разрабатывать проект за несколько итераций и постепенно его улучшать, реализуя различные идеи.

Рассмотрим подробнее модель Скрам (Scrum). Скрам — это набор принципов, на которых строится процесс разработки, позволяющий в жёстко фиксированные и небольшие по времени итерации, называемые спринтами (sprints), предоставлять конечному пользователю работающее ПО с новыми возможностями, для которых определён наибольший приоритет. Возможности ПО к реализации в очередном спринте определяются в начале спринта на этапе планирования и не могут изменяться на всём его протяжении. При этом строго фиксированная небольшая длительность спринта придаёт процессу разработки предсказуемость и гибкость.

Спринт— одна итерация в скрам, в ходе которой создаётся функциональный рост программного обеспечения. Длительность одного спринта от 2 до 4 недель. На протяжении спринта никто не имеет права менять список требований к работе, внесенном в резерв проекта.

Резерв проекта— это список требований к функциональности, упорядоченный по их степени важности, подлежащих реализации. Резерв проекта открыт для редактирования для всех участников скрам процесса.

Резерв спринта — содержит функциональность, выбранную владельцем проекта из резерва проекта. Все функции разбиты по задачам, каждая из которых оценивается скрам-командой. Каждый день команда оценивает объем работы, который нужно проделать для завершения спринта.

Диаграмма сгорания задач (Burndown chart) отображает завершенный спринт. Показывает оставшиеся нерешенные задачи и трудозатраты, необходимые для их завершения.

Диаграмма, показывающая количество сделанной и оставшейся работы обновляется ежедневно, чтобы показать подвижки в работе над спринтом.

Диаграмма сгорания работ для выпуска проекта — показывает, сколько уже задач сделано и сколько ещё остаётся сделать до выпуска продукта (обычно строится на базе нескольких спринтов).

Требуемую функциональность, которую добавляют в резерв называют историей спринта.

Остановка спринта может быть произведена раньше срока его планового окончания в исключительных ситуациях. Спринт может остановить команда, если понимает, что не может достичь цели спринта в отведенное время. Спринт может остановить владелец проекта, если исчезает необходимость в реализации цели спринта.

Задачи истории спринта (Sprint Story Tasks)

Добавляются к историям спринта. Выполнение каждой задачи оценивается в часах. Каждая задача не должна превышать 12 часов.

Скорость команды - общее количество очков набранных командой за предыдущий спринт. Данная метрика помогает команде понять, сколько историй она может сделать за один спринт.

Роли в скрам-процессе:

Скрам-мастер (ScrumMaster) — проводит совещания (Scrum meetings), следит за соблюдением принципов скрам, разрешает противоречия и защищает команду от отвлекающих факторов.

Владелец проекта (Product Owner) — представляет интересы конечных пользователей.

Скрам-команда (Scrum Team) — кросс-функциональная команда разработчиков проекта, состоящая из специалистов разных профилей: тестировщиков, архитекторов, аналитиков, программистов и т. д. Размер команды в идеале составляет 7±2 человека. Команда является единственным полностью вовлечённым участником разработки и отвечает за результат как единое целое. Никто кроме команды не может вмешиваться в процесс разработки на протяжении спринта.

Планирование спринта (Sprint Planning Meeting) происходит в начале новой итерации Спринта.

Еще одна технология управления проектами это Канбан. Канбан имеет множество значений: карточка, бирка, плакат, доска объявлений. В основе Kanban лежит простая идея. Количество незавершённой работы должно быть ограничено, и что-либо новое может начинаться только тогда, когда какой-то существующий кусок работы поставляется или (в терминах lean) вытягивается следующим элементом конвейера. Kanban (или сигнальная карточка) подразумевает, что производится визуальное оповещение о том, что можно "вытягивать" новую работу.

Для ведения проектов широко используются канбан-доски. Канбан-доска имеет вид таблицы со следующими полями.

«Сделать»- здесь размещаются карточки с задачами по всем проектам.

«В процессе» – здесь число карточек с задачами ограничено; сюда помещаются те процессы, над которыми в данный момент ведется работа, они поступают из списка «Сделать» в порядке приоритетности

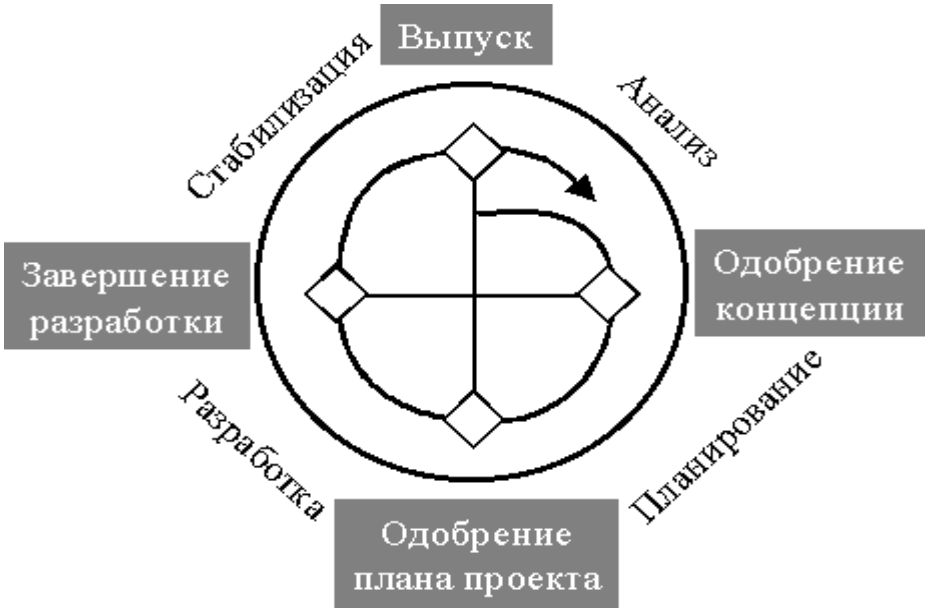
«Релиз» – карточки с задачами, которые уже завершены, и требуют проверки.

«Сделано» – задачи, которые полностью проверенные и завершены.

По ходу работы над карточкой, она перемещается в соответствующие колонки. Такие разделы может использовать любая фирма, но, в зависимости от специфики ее деятельности, доска может быть видоизменена, как потребуется.

Согласно публикациям по обследованию 80 фирм в Германии на предмет применения системы Канбан, был сделан вывод итог: производственные запасы с внедрением Канбан уменьшаются, в среднем, в два раза, а производительность труда увеличивается до 50%.

Microsoft Solutions Framework (MSF)



Программные продукты Microsoft создаются по этой методологии.

Программные средства управления проектами

Управление проектами (англ. project management) — область деятельности, в ходе которой определяются и достигаются поставленные цели при балансировании между объемом работ, ресурсами (время, деньги, труд, материалы), качеством и рисками в рамках разрабатываемых проектов. Существует большое число компьютерных приложений для автоматизации управления проектами.

Microsoft Project. Microsoft Project предоставляет многофункциональное решение, которое позволяет контролировать проекты, выводить на экран таблицы и графики. Microsoft Project интегрируется с Outlook, MS Office и Internet Explorer. Основной недостаток – сложность в использовании.

Ниже рассмотрены несколько более простых и дешевых программ, полезных в управлении проектами и ориентированных на вебразработку и вебдизайн.

Basecamp. Basecamp признаётся лучшей платформой для организации управления проектами и совместной работы над ними. Basecamp задумывался как средство управления проектами в небольших компаниях. Здесь есть смена цветовой гаммы и логотипа системы, просмотр общей информации о клиентах и проектах на одном экране, назначение и отслеживание задач, загрузка, категоризация и отслеживание версий файлов, форумы для обсуждения задач и проектов, ведение расписания и управление ключевыми точками проекта, отслеживание потраченного времени, получение основной информации о проекте на одном экране, добавление сообщений и комментариев. Сервис платный, но дешевый. Часто используемая программа Springloops – это SVN браузер с возможностью добавления функционала для управления проектами посредством интеграции с BaseCamp.

Lighthouse. Lighthouse – это приложение для отслеживания багов и запросов на изменение. Создание проекта занимает несколько минут – для этого требуется лишь ввести его название и описание. Lighthouse хорошо подходит для небольших групп веб-разработчиков. В комбинации с SVN Lighthouse представляет собой полнофункциональную систему по управлению проектами. В самом простом случае сервис бесплатен. SVN(Subversion) это система управления версиями, применяемая в Unix –сістемах.

Jumpchart – это приложение, предназначенное для планирования навигации по веб-сайту посредством создания и перетаскивания страниц. В созданные странички можно добавить текст и форматирование, а по завершению работы экспортировать CSS файлы и sitemap. Кроме функционала планирования страничек здесь присутствуют и опции для отслеживания задач. Бесплатный набор возможностей простейшего варианта небольшой, поэтому для больших проектов следует ориентироваться на платные планы.

Trac Project. Trac Project – это бесплатная система для управления проектами, которая кроме базовой функциональности вики-приложений, обладает SVN браузером. Отличительная черта Trac это наличие большого количества плагинов для него, например плагинов для web administration, аутентификации, управления документацией кода, тикетами, тестированием, пользователями и контроля версий.

Pbwiki. Pbwiki – одно из наиболее простых вики-приложений, которое позволяет совместно использовать файлы вместе с другими пользователями, разграничивать доступ к отдельным файлам и папкам, добавлять пользователей, отслеживать изменения версий файлов. Установка программы очень быстрая, как и освоение принципов работы самой программы. Создание папок и страниц, а также их редактирование осуществляется максимально простым путём. К каждой странице можно добавить комментарий, а также одним кликом получить её версию для печати. Ещё более упрощают использование приложения набор темплейтов для стандартных страниц. Кроме того предусмотрены и различные темы для дизайна самой Pbwiki. Бесплатный план сервиса предполагает использование приложения максимум тремя пользователями.

JIRA. JIRA гораздо сложнее и обладает большим количеством возможностей по отслеживанию багов и всевозможных запросов на изменение. JIRA обладает продвинутыми возможностями по репортингу, отображению процесса разработки (маппингу) и организации отслеживания изменений и запросов. Кроме того, JIRA предлагает пользователям большое количество плагинов, посредством которых можно добавить дополнительный функционал по управлению проектом, управлению временем, календарь, интеграцию с Bamboo и т.д. Главная проблема JIRA – это её цена.

Перечислим несколько других программных средств для управления веб-проектами:

Easy Projects .NET — система для управления проектами, написанная на .NET, eGroupWare бесплатное ПО для управления проектами, Kommandcore - платный многопользовательский веб-сервис по управлению проектами, предназначенный в первую очередь для руководителей проектами. OpenProj — бесплатная, открытая альтернатива Microsoft Project.

Project Kaiser — Веб-ориентированная система управления проектами и задачами с поддержкой wiki и развитыми средствами взаимодействия пользователей

Redmine - бесплатный многопользовательский веб-сервис, ориентированный на специфику IT-проектов и разработчиков.

TrackStudio Enterprise — система управления задачами. Есть экспорт в MS Project.

Google Code запустил сервис хостинга проектов, в рамках которого предоставляет свободным проектам систему управления версиями (Subversion, Git или Mercurial), а также систему отслеживания ошибок, вики-систему для документации и файловый архив (с ограничением в 100 Мб на размер одного файла). Сервис доступен и бесплатен для использования.

Веб – дизайн и разработка сайтов

Под термином веб-дизайн понимают проектирование структуры веб-ресурса и обеспечение клиенту удобства пользования ресурсом. К Веб-дизайну относятся: логика построения сайта и предоставления информации на экране, создание привлекательного и функционального интерфейса сайта, наиболее удобные решения подачи информации.

Стили сайтов

В настоящее время выделяют некоторые общие направления стилей: классический, хай-тек, табличный, пиктографический, минимализм, журнальный, газетный, текстовый, фирменный стиль, стиль Web 2.0.

Классический стиль поддерживает строгие лаконичные формы и неброские цвета. Таким стилем хорошо сообщать публике прогнозы рынка forex, экономические новости и биржевые сводки.

Сайты, имеющие много информации, придерживаются текстового стиля, где важна четкая определенная структура, а графика имеет второстепенное значение.

Газетный стиль подходит для тех сайтов, которые имеют большое количество перекрестной информации, поскольку в этом стиле выделена табличная верстка, на всех страницах присутствуют ссылки на темы рубрик.

К художественному стилю часто прибегают крупные корпорации и студии дизайна.

Техно-стиль или хай-тек это стиль, где за основу берутся каркасы моделей, трехмерные конструкции, роботы и части металлических изделий.

Минимализм в веб-дизайне означает использование простоты классических геометрических форм и минимального количества цвета и красок. Многие известные дизайнеры вернулись к старому заключению «все гениальное — просто». Белый фон, черный или серый шрифт, минимум графики, максимум «юзабилити» и

информативности. Одной из известных работ в таком стиле стал портал американской газеты New York Times.

Фирменный стиль для компании означает несколько вещей. В первую очередь, это логотип. Логотип — это герб бизнеса, успешный выбор которого сделает компанию запоминающейся для клиентов. Во-вторых, уникальное оформление визиток, фирменных бланков и сайта обеспечивает уникальность рекламной и корпоративной продукции компании. Фирменный стиль станет гарантией доверия и узнаваемости компании среди потребителей, способствуя росту её положительной репутации на рынке.

Одним из наиболее популярных сейчас является стиль Web 2.0. Он прост в исполнении, имеет небольшое число колонок, выравнивание по центру, удобную навигацию. Для пользователей он привлекателен крупным текстом, яркими цветами и объемными эффектами, оригинальными иконками.

Логическое проектирование дизайна сайта

Логическое проектирование включает организацию информации на сайте, построение его структуры и навигации по разделам. На этом этапе следует описать следующее:

1. Тип структуры сайта (линейная, иерархическая, контекстная, другая).
2. Названия разделов.
3. Что будет содержать в себе каждый раздел.
4. Организация и связь разделов между собой.
5. Какая информация будет размещена на определенных страницах сайта.

При логическом проектировании информационного наполнения сайта и средств навигации следует придерживаться четырех базовых принципов, которые основаны на восприятии информации человеком.

1. *Использование обозначений.*

Применяйте слова и термины, которые являются устоявшимися и понятными большинству посетителей сайта. В этом смысле не очень удачным является применение пиктограмм, которые могут быть поняты посетителями сайта неправильно.

2. *Уместность*

Разделы сайта должны содержать информацию и элементы, которые соответствуют данной части или фрагменту сайта. Элементы, которые не отвечают данному принципу, должны быть перенесены в другое место или удалены.

3. *Единообразие*

Использование единой навигации по и единого стиля оформления. Так, одинаковые элементы на страницах должны иметь один и тот же размер и находиться в одном и том же месте. Если страницы, по какой-либо причине имеют отличия, пользователь должен четко понимать, чем они обусловлены.

4. Разделение на части

Человек не в состоянии воспринимать одновременно большой объем информации. Как правило, люди способны воспринимать информацию, содержащую от четырех до шести различных элементов. Поэтому, посетители сайта лучше ориентируются и быстрее находят нужные им материалы, когда они разделены на группы, содержащие от четырех до шести элементов.

Организация данных в виде упорядоченной структуры должна сообщать посетителю, какую информацию он может обнаружить на сайте и где ее искать. Структура сайта напоминает оглавление книги; если оно хорошо написано, становится понятно, что ожидать от книги, еще не прочтя ее. Информация должна быть организована таким образом, чтобы посетитель знал, что его ждет на следующей странице уже по названию ссылки, на которую он нажимает.

Организация информации на сайте может быть:

Линейная структура

Наиболее простой способ организации данных. Представляет собой набор следующих друг за другом веб-страниц.

Иерархическая структура

Самый распространенный вариант размещения информации. Предусматривает применение главной страницы, на которой размещается меню со ссылками на разделы сайта, расположенные на следующих страницах. Разделы могут содержать в себе подразделы и другую детальную информацию.

Контекстно-зависимая структура

В подобной структуре ссылки на другие разделы сайта формируются в зависимости от определенных действий пользователя. К примерам такой структуры относятся элементы интернет-магазинов, поиск по ключевым словам.

Комбинированная структура

Представляет собой сочетание нескольких предыдущих структур. Например, на сайте может быть предусмотрена иерархическая структура, которая в некотором месте может содержать выполнение пошаговых действий (линейную структуру).

Навигация характеризует удобство перемещения между разделами сайта, возможность быстро переходить на нужную страницу. Продуманная навигация сайта должна отвечать на вопросы посетителя: где я нахожусь, какие страницы я уже посетил, какие разделы еще могу посетить.

Необходимо показать посетителю, в каком месте он находится относительно самого сайта. HTML позволяет использовать разные цвета для посещенных и не посещенных ссылок, чтобы разница между ними была заметна. Ссылки на разделы сайта должны быть на каждой его странице, включая и повторяющуюся от страницы к странице навигацию.

Главная страница сайта

Главная страница является самой важной страницей любого веб-сайта. Большинство пользователей приходят на нее. Главная страница должна содержать приветствие, из которого посетитель может понять что это за сайт, и что на нем есть. На главной странице должны быть ссылки на разделы сайта.

Главная страница дает представление обо всем сайте - о его содержимом, дизайне, навигационной системе, структуре. Поэтому ее дизайн должен быть эргономичным. Например, назначение начальной страницы коммерческого или корпоративного сайта заключается в том, чтобы дать представление пользователям о данной компании, о товарах и услугах, предлагаемых компанией о преимуществах перед другими компаниями.

На главной странице сайта могут быть разделы: *Статьи*. В разделе можно почитать материалы по теме. *Новости*. Несут ту же функцию, что и статьи, только постоянно обновляются. *Магазин или раздел с описанием услуг*. Рядом со статьями или новостями могут отображаться товары и услуги. *Блог*. Блог можно сделать как альтернативу новостному разделу. *Форум*. Нужен для оказания технической поддержки и для общения посетителей. *FAQ*. Часто задаваемые вопросы и ответы. В этом разделе можно описать вопросы и ответы в поддержку товаров. Эти страницы генерируют также поисковый трафик. *Файловый архив*. Это документация, программное обеспечение, фотографии. *Портфолио*. Нужно для студий Веб-дизайна, фрилансеров, разработчиков ПО. Своего рода витрина, на которой представлены лучшие образцы работ. *Обратная связь*. Форма обратной связи или данные для связи с менеджером, администратором должна быть на любом сайте. *Рассылка*. Она необходима, чтобы иметь контакт с посетителями. *Поиск*. Если у Вас есть все, что описано выше, то поиск просто необходим. *Информеры:* погода, курсы валют, календарь. *Регистрация/Авторизация* на

сайте проводится в случае, если предполагается предоставление некоторых дополнительных услуг

Оформление главной страницы должно отличать ее от остальных страниц сайта. Для этого применяется либо слегка измененный дизайн, который согласуется с оформлением и стилем всего сайта. К проблемам дизайна главной страницы отнесем следующие вопросы и разделы:

Назначение сайта: Название или логотип компании должны иметь подходящий размер и располагаться в заметном месте страницы. Обычно это верхний левый угол. Нельзя перегружать эту область визуальными эффектами. Количество ключевых задач должно быть небольшим, а область страницы вокруг нее - незаполненной.

Информация о компании: Кроме названия и назначения компании к странице часто добавляется рекламный лозунг(слоган), который бы четко характеризовал деятельность компании. Лозунг должен быть коротким и метким. Так, например, лозунг компании Яндекс "Найдется все" (еще один - “Мы обуем всю Россию”). От лозунгов можно отказаться, если в имени компании раскрывается ее назначение или если компания известна как Microsoft.

Стиль содержимого сайта:

Выбирая слова, следует ориентироваться на язык пользователей, а не компании. Строго следите за применением прописных букв и соблюдением текстового стиля. Формулируя задания для пользователей сайта, используйте повелительное наклонение, например "Введите название города". Избегайте восклицательных знаков.

Использование примеров для краткого обзора содержимого сайта: Каждый пример должен сопровождаться ссылкой на подробное описание данного продукта и фотографии.

Ссылки: Ссылки подчеркиваются и выделяются голубым цветом. Цвета просмотренных и не просмотренных ссылок должны различаться. Если функция ссылки отличается от обыкновенного перехода на другую страницу и состоит в загрузке PDF-файла, отправке электронного сообщения, запуске аудио- или видеопроигрывателя об этом должно быть сказано явно.

Навигация:

Главная панель навигации должна находиться в заметном месте страницы, предпочтительно рядом с ее основной частью. Не следует размещать верхнюю горизонтальную панель навигации над графическими элементами наподобие горизонтальных границ или баннеров. Страница не должна содержать активной ссылки на саму себя.

Поиск: Главная страница должна содержать непосредственно поле для ввода поисковых запросов.

Ссылки на службу сайта: Не добавляйте на сайт службы, не имеющие отношения к его основной тематике. Например, не обязательно добавлять ссылку на прогноз погоды, если никакого отношения к погоде этот сайт не имеет.

Графика и анимация: Позвольте пользователю самому решать, хочет ли он видеть анимированную заставку к сайту - не запускайте ее по умолчанию или обеспечьте легкий и понятный способ ее отключения.

Графическое оформление:

Постарайтесь обойтись без горизонтальной полосы прокрутки. Все главные элементы главной страницы по возможности должны находиться (выше линии сгиба) (в первом экране страницы, доступном без вертикальной прокрутки) для наиболее распространенного разрешения экрана. Используйте гибкую структуру главной страницы, чтобы ее размер мог автоматически приспосабливаться к различным разрешениям экрана.

Раскрывающиеся окна и промежуточные страницы: Пользователи, набравшие в адресной строке обозревателя главный URL Вашего сайта либо пришедшие на него по ссылке, должны попадать сразу на настоящую главную страницу. Откажитесь от раскрывающихся окон.

Рекламные объявления: Рекламные баннеры других компаний следует вынести на периферию страницы.

Большинство сайтов применяют стандартную компоновку главной страницы. Вверху слева помещается название и логотип компании. Затем может идти раздел "Новости", потом - раздел с названием "О компании", содержащий историю фирмы. Далее идет раздел "Услуги". На главной странице торгового сайта можно также разместить раздел "Статьи", содержащий статьи о товарах и услугах и, например, "Полезное". В последнем можно предоставить ссылки на бесплатное скачивание и руководство по настройке и использованию программ, используемых в Интернет. Последним разделом, который обязательно должен присутствовать на главной странице Интернет-магазина и не только, является раздел с названием "Контакты". Здесь нужно разместить не только телефон, и Skype компании, но и фактический адрес. В этом разделе может быть представлена точная схема проезда. Естественно, на главной странице торгового сайта обязательно должны быть приведены картинки и фото предлагаемого товара, ведь благодаря им посетитель сможет быстрее и лучше понять суть вашего бизнеса.

Самые лучшие главные страницы не только информативны, просты и понятны. Они еще должны быть адресованными лично посетителю. Персонализация главной страницы очень важна.

Приведем несколько рекомендаций для улучшения дизайна сайта.

1. Посетитель должен сразу понять, куда он попал и что он может найти на сайте. На главной странице сайта должно быть 2-3 абзаца с описанием основных услуг компании.
2. Основные услуги должны иметь ссылки с главной страницы сайта.
3. Навигация – интуитивно понятна, любой раздел должен быть достижим за 2-3 клика мышью.
4. Желательно иметь диалоговую форму для отправки заявок
5. На сайте с разветвленной структурой должны быть были дополнительные навигационные элементы (навигационная строка «хлебные крошки») и дополнительные дублирующие меню.
6. Контакты должны быть на видном месте. Это телефон или адрес электронной почты.
7. Дизайн главной страницы может отличаться от дизайна внутренних страниц сайта, но все страницы должны быть выполнены в едином стиле.
9. *Люди не читают веб-страницы, а просматривают.* Они задерживаются на словах и выражениях, которые привлекают внимание.
10. *Пользователи избегают медленно загружающихся сайтов.*

Внутренние страницы сайта

Если главная страница содержит крупные изображения по теме сайта, пользователь вынужден загружать определенный объем, что увеличивает время загрузки. Пользователя заставляют посмотреть рекламную вступительную часть сайта. Это, как правило, соответствует цели выделить сайт на фоне других. Нет смысла преследовать те же цели на внутренних страницах. Внутренние страницы сайта должны в первую очередь предоставлять запрашиваемую информацию и только затем следовать общему стилю и художественным изыскам. Исходя из всех этих соображений, стараются не перегружать внутренние страницы новыми графическими объектами, и по возможности использовать уже примененные на главной странице. Это позволяет осуществлять более быструю загрузку страницы, т.к. эти объекты уже загружались из сети и они теперь есть в кэш-

директории браузера. В этих же целях названия разделов целесообразнее делать с использованием системных шрифтов. При этом теряется неповторимость их написания, но зато все пользователи быстро и правильно увидят навигацию и доступ к другим разделам.

Примерно та же ситуация и с самой текстовой информацией раздела. Для пользователя будет наиболее удобно и универсально, если это будет системный шрифт (что бы не пришлось что-то дополнительно устанавливать). Размер шрифта, по возможности, должен остаться не жестко фиксированным, что бы была возможность увеличить или уменьшить шрифт в зависимости от размера и разрешения экрана. Так же удобнее будет, если страница будет масштабируемой, что позволит пользователю настроить параметры восприятия страницы так, как ему удобнее и как это позволяет его компьютер и монитор.

Информация на внутренних страницах сайта может быть следующей:

Название или логотип сайта (может быть уменьшенный), навигация по всем разделам первого уровня, название раздела, в котором находится пользователь, собственно информация - тексты, иллюстрации и т.п., возврат на главную.

Глава 3. Краткий обзор основных технологий разработки Веб приложений

Перечислим средства, необходимые при создании сайта.

HTML, XML. Языки разметки гипертекстовых документов, которые передаются через Internet и отображаются браузером.

CSS. Язык описания стилей, определяющий вид отображения гипертекстового документа. Описываются типы шрифтов и другие дизайнерские атрибуты.

JavaScript, Flash. Предоставляют возможность для написания скриптов, выполняемых на стороне клиента.

Эти средства включены в состав любого браузера, лучшим из которых для разработки является Mozilla Firefox.

PHP, Perl, Java. Используются для написания скриптов, выполняемых на стороне сервера.

Сервер баз данных. Программа, организующая работу с базами данных на сервере. Вместе с PHP наиболее часто используется сервер БД MySql.

WEB-сервер. Программа, обрабатывающая запросы браузеров

Язык разметки гипертекста XML

Язык XML привлекает к себе внимание со стороны разработчиков и пользователей Интернет. Появляются новые языки, созданные на основе XML, возникают Web-сервера, использующие эту технологию для организации хранящейся на них информации. Годом рождения XML можно считать 1998 год, когда спецификация языка была утверждена. Хотя понятие гипертекста было введено В. Бушем в 1945 году и, начиная с 60-х годов, стали появляться приложения, использующие гипертекстовые данные, всплеск активности вокруг этой технологии начался тогда, когда возникла реальная необходимость в механизме объединения множества информационных ресурсов. В 1986 году появился язык SGML, с помощью которого можно описывать структурированные данные, организовывать информацию, содержащуюся в документах, представлять эту информацию в некотором стандартизованном формате. Контроль за правильностью использования дескрипторов было предложено осуществлять при помощи специального набора правил, называемых DTD- описаниями, которые используются программой клиента при разборе документа. Ввиду своей сложности, SGML использовался, в основном, для описания синтаксиса других языков, наиболее известным из которых является HTML.

HTML является упрощенной версией языка разметки SGML. Инструкции HTML, в первую очередь определяют способ представления документа, но не его структуру. Игнорирование структуры документа приводит к тому, что поиск или анализ информации внутри него не будет отличаться от работы со сплошным, не разбитым на элементы текстовым файлом. Другим существенным недостатком HTML можно назвать ограниченность набора его тэгов. У разработчика нет возможности вводить собственные, специальные тэги.

На смену HTML был предложен язык гипертекстовой разметки XML (Extensible Markup Language). Этот язык может использоваться для описания грамматики других языков и для контроля правильности составления документов. XML определяет порядок создания тегов, предназначенных для разметки и позволяет легко их расширить. Таким образом появляется возможность определять собственные теги, для работы с данными, содержащимися в документе. При создании структуры документа, строятся связи между элементами и разметка, необходимая для выполнения операций просмотра, поиска, анализа документа. XML можно использовать в качестве универсального языка запросов к хранилищам информации. Таким образом, XML-документы могут выступать в качестве уникального способа хранения данных, включающего в себя средства для разбора информации и ее представления.

Сильные и слабые стороны

XML —позволяет стандартизировать вид файлов-данных, используемых компьютерными программами, в виде текста, понятного человеку; XML поддерживает Юникод; в формате XML могут быть описаны такие структуры данных как записи, списки и деревья; XML — это самодокументируемый формат, который описывает структуру и имена полей, так же как и значения полей; XML имеет строго определённый синтаксис и требования к анализу. XML — формат, основанный на международных стандартах; Иерархическая структура XML подходит для описания практически любых типов документов, кроме аудио и видео мультимедийных потоков; XML представляет собой простой текст, свободный от лицензирования и каких-либо ограничений; XML не зависит от платформы и поддерживается на низком аппаратном и микропрограммном уровнях.

Фактически XML - это способ разметки документов, предназначенный для формирования в документах какой-либо структуры и определения отношений между различными элементами этой структуры. Для создания такой разметки служат специальные инструкции, называемые тегами. Теги располагают между символами < и >. Благодаря наличию тегов становится возможной унифицированная автоматическая обработка и форматирование XML-документов. Благодаря XML удается также контролировать правильность данных, хранящихся в документах, а также установить единый стандарт на структуру документов, в которых могут содержаться произвольные данные.

Пример типичного XML-документа

```
<?xml version="1.0" encoding="windows-1251" ?>
<!DOCTYPE InfoPacket SYSTEM "http://xml.prime-tass.ru/dtd/UIF.dtd">
<InfoPacket>
  <Source type="string">ПРАЙМ-ТАСС</Source>
  <Time type="datetime">08.05.2001 13:53:17</Time>
  <InfoItem>
    <Time type="datetime">17.04.2001 10:07:47</Time>
    <Title type="string">  ……    </Title>
    <Text>
      ……..
    </Text>
```

```
   <Text>
……….……
   </Text>
     </InfoItem>
</InfoPacket>
```

Первые две строчки в этом примере должны присутствовать. Первая строчка означает, что данный документ является XML-документом, а вторая указывает на специальный ресурс http://xml.prime-tass.ru/dtd/UIF.dtd, в котором содержатся правила оформления данного XML-документа). Информацию можно получить на сайте http://www.w3.org.

Остальные строчки имеют иерархическую структуру, в которой есть один корневой тег **<InfoPacket>**, а остальные теги вложены в него. Все эти теги встречаются парами, например, **<InfoPacket>** и **</InfoPacket>**, или **<Text>** и **</Text>**. Первые - обозначают открывающийся тег, вторые - парный ему закрывающийся. В XML-документе каждый открывающийся тег обязан иметь парный ему закрывающийся.

Тег **<InfoPacket>** является самым главным тегом - корнем иерархической структуры и предназначен для хранения вложенных в него остальных тегов. Таким образом, он представляет собой своеобразный "конверт" или "обертку" - информационный пакет. В него вложены теги **<Source>**, **<Time>** и **<InfoItem>**. Кроме тега **<Time>** в **<InfoItem>** вложен тег **<Title>**, в котором написан заголовок документа или "письма", если продолжить нашу аналогию. И, наконец, остался тег **<Text>**, который несколько раз встречается внутри тега **<InfoItem>**. Этот тег **<Text>** обозначает просто абзацы документа.

Осталось только сказать, что означают и для чего нужны записи, типа **type="datetime"** или **type="string"** внутри названий тегов **<Source>**, **<Time>** и **<Title>** и почему их нет в тегах **<Text>**. Такого рода записи в XML называются атрибутами тегов. В данном случае мы имеем один атрибут **type**, который принимает значения **"datetime"** и **"string"**. Значения атрибутов в XML всегда заключаются в кавычки. Атрибут **type** нужен для того, чтобы точно указать как следует интерпретировать последовательность символов, заключенную внутри тега с этим атрибутом. Значение **"datetime"** означает, что значение тега нужно интерпретировать как дату и время, а значение **"string"** - как обычную строку. У тега **<Text>** атрибут **type** отсутствует, т.к. по смыслу этого тега и так ясно, что в этом теге содержится текстовая строка.

Сам по себе XML не содержит никаких тэгов, предназначенных для разметки, он определяет порядок их создания. Таким образом, если, например, мы считаем, что для обозначения элемента *rose* в документе необходимо использовать тэг <*flower*>;, то XML позволяет свободно использовать определяемый нами тэг и мы можем включать в документ фрагменты, подобные следующему:

```
<flower>rose</flower>
```

Набор тэгов может быть легко расширен. Если мы хотим указать, что описание цветка должно по смыслу идти внутри описания оранжереи, в которой он цветет, то просто задаем новые тэги и выбираем порядок их следования:

```
<conservatory>
<flower>rose</flower>
</conservatory>
```

Если мы хотим посадить туда еще несколько цветочков, то должны внести следующие изменения:

```
<conservatory>
<flower>rose</flower>
<flower>tulip</flower>
```

<flower>cactus</flower>
</conservatory>

Как видно, сам процесс создания XML документа очень прост и требует от нас лишь базовых знаний HTML и понимания тех задач, которые мы хотим выполнить, используя XML в качестве языка разметки. Таким образом, у разработчиков появляется уникальная возможность определять собственные команды, позволяющие им наиболее эффективно определять данные, содержащиеся в документе. Автор документа создает его структуру, строит необходимые связи между элементами, используя те команды, которые удовлетворяют его требованиям и добивается такого типа разметки, которое необходимо ему для выполнения операций просмотра, поиска, анализа документа.

XML позволяет осуществлять контроль за корректностью данных, хранящихся в документах, производить проверки иерархических соотношений внутри документа и устанавливать единый стандарт на структуру документов, содержимым которых могут быть самые различные данные. Это означает, что его можно использовать при построении сложных информационных систем, в которых очень важным является вопрос обмена информацией между различными приложениями, работающими в одной системе. Создавая структуру механизма обмена информации в самом начале работы над проектом, менеджер может избавить себя в будущем от многих проблем, связанных с несовместимостью используемых различными компонентами системы форматов данных.

На основе XML созданы такие известные специализированные языки разметки, как SMIL, CDF, MathML, XSL, их список пополняется.

Правильно построенные и действительные документы XML

Стандартом определены два уровня правильности документа XML:

- *Правильно построенный* документ соответствует всем правилам синтаксиса XML. Если, например, начальный тег не имеет конечного тега, то это *неправильно построенный* документ XML. XML-процессор (парсер) не должен обрабатывать его обычным образом и обязан классифицировать ситуацию как фатальная ошибка.
- *Действительный* (англ. *valid*). Действительный документ дополнительно соответствует некоторым семантическим правилам. Эти правила могут быть разработаны как самим пользователем, так и разработчиками словарей или стандартов обмена данными. Обычно такие правила хранятся в специальных файлах — схемах, где самым подробным образом описана структура документа, все допустимые названия элементов, атрибутов и многое другое. И если документ, например, содержит не определённое заранее в схемах название элемента, то XML-документ считается *недействительным*; проверяющий XML-процессор (валидатор) при проверке на соответствие правилам и схемам обязан (по выбору пользователя) сообщить об ошибке.

Объявление XML

Первая строка XML-документа называется *объявление XML* (англ. *XML declaration*) — это строка, указывающая версию XML. В версии 1.0 объявление XML может быть опущено, в версии 1.1 оно обязательно. Также здесь может быть указана кодировка символов и наличие внешних зависимостей.

<?xml version="1.0" encoding="UTF-8"?>

Спецификация требует, чтобы процессоры XML обязательно поддерживали Юникод-кодировки UTF-8 и UTF-16. Допустимы другие кодировки, например, русские Windows-1251, KOI-8. Часто в тегах принципиально не используют не-латинские буквы, в этом случае UTF-8 является очень удобной кодировкой: декодирование может быть выполнено как для всего документа, так и для конкретных атрибутов и текстов; весь документ не содержит запрещённых символов при попытке разбора с неправильной кодировкой.

Корневой элемент

Документ имеет только один *корневой элемент* (англ. *root element*). Это означает, что текст или другие данные всего документа должны быть расположены между единственным начальным корневым тегом и соответствующим ему конечным тегом.

Комментарий

XML-комментарии размещаются внутри специального тега, начинающегося с символов `<!--` и заканчивающегося символами `-->`.

<!-- Это комментарий. -->

Теги

Остальная часть этого XML-документа состоит из вложенных *элементов*, которые имеют *атрибуты* и *содержимое*. *Элемент* обычно состоит из открывающего и закрывающего тегов, обрамляющих текст и другие элементы. *Открывающий тег* состоит из *имени элемента* в угловых скобках, например, `<step>`, а *закрывающий тег* состоит из того же имени в угловых скобках, но перед именем ещё добавляется косая черта, например, `</step>`. Имена элементов могут быть на любом языке, поддерживаемом кодировкой XML-документа. Имя может начинаться с буквы, подчёркивания, двоеточия. *Содержимым элемента* (англ. *content*) называется всё, что расположено между открывающим и закрывающим тегами, включая текст и другие (вложенные) элементы. Ниже приведён пример XML-элемента, который содержит открывающий тег, закрывающий тег и содержимое элемента:

<step>Замесить ещё раз, положить на противень и поставить в духовку.</step>

Кроме содержания у элемента могут быть *атрибуты* — пары имя-значение, добавляемые в открывающий тег после названия элемента. Значения атрибутов всегда заключаются в кавычки (одинарные или двойные), одно и то же имя атрибута не может встречаться дважды в одном элементе. Не рекомендуется использовать разные типы кавычек для значений атрибутов одного тега.

<ingredient amount="3" unit="стакан">Мука</ingredient>

В приведённом примере у элемента «`ingredient`» есть два атрибута: «`amount`», имеющий значение «3», и «`unit`», имеющий значение «стакан». С точки зрения XML-разметки, приведённые атрибуты не несут никакого смысла, а являются просто набором символов.

Кроме текста, элемент может содержать другие элементы:

```
<instructions>
 <step>Смешать все ингредиенты и тщательно замесить.</step>
 <step>Закрыть тканью и оставить на один час в тёплом помещении.</step>
 <step>Замесить ещё раз, положить на противень и поставить в духовку.</step>
</instructions>
```

В данном случае элемент «`instructions`» содержит три элемента «`step`».

XML не допускает перекрывающихся элементов. Например, приведённый ниже фрагмент некорректен, так как элементы «`em`» и «`strong`» перекрываются.

```
<!-- ВНИМАНИЕ! Некорректный XML! -->
<p>Обычный <em>акцентированный <strong>выделенный и акцентированный</em> выделенный</strong></p>
```

Для обозначения элемента без содержания, называемого *пустым элементом*, необходимо применять особую форму записи, состоящую из одного тега, в котором после имени элемента ставится косая черта. Если в DTD элемент не объявлен пустым, но в документе он не имеет содержания, для него допускается применять следующие (три) формы записи. Например:

```
<foo></foo>
<foo />
<foo/>
```

Спецсимволы

Отображение XML документа

Наиболее распространены три способа преобразования XML-документа в отображаемый пользователю вид:

1. Применение стилей CSS;
2. Применение XSL;
3. Написание на каком-либо языке программирования обработчика XML-документа.

Без использования CSS или XSL XML-документ отображается как простой текст в большинстве веб-браузеров. Некоторые браузеры отображают структуру документа в виде дерева, позволяя сворачивать и разворачивать узлы с помощью нажатий клавиши мыши.

Применение стилей CSS

Процесс аналогичен применению CSS к HTML-документу для отображения.

Для применения CSS при отображении в браузере, XML-документ должен содержать специальную ссылку на таблицу стилей. Например:

<?xml-stylesheet type="text/css" href="myStyleSheet.css"?>

Это отличается от подхода HTML, где используется элемент <link>.

Применение XSL

XSL является семейством рекомендаций, описывающих языки преобразования и визуализации XML-документов. Документ трансформируется в формат, подходящий для отображения в браузере. Браузер — это наиболее частое использование XSL, но не стоит забывать, что с помощью XSL можно трансформировать XML в любой формат, например VRML, PDF, текст.

Для задания XSL трансформации (XSLT) на стороне клиента требуется наличие в XML инструкции следующего вида:

<?xml-stylesheet type="text/xsl" href="transform.xsl"?>

Adobe Flash

Разработчики страниц пытаются придать страницам привлекательный вид, используя графику и анимацию. Замечательным инструментом создания графики и анимации является программное средство Adobe Flash. Flash обладает такими возможностями как:

- Применение векторной графики, представляющей собой ряд формул, описывающих изображения. Это позволяет уменьшать размеры графических файлов.

- Свойство потоковой передачи графики позволяет отображать сайт в Web-браузере до полной его загрузки.

- Интерактивность во Flash позволяет вводить кнопки, меню или фрагменты анимации, с помощью которых пользователь перемещается по сайту. Задавая события с помощью языка создания сценариев ActionScript можно создавать интерактивные анимации.

Adobe Flash - мультимедийная платформа компании Adobe для создания веб-приложений и мультимедийных презентаций. Широко используется для создания рекламных баннеров, анимации и игр. Flash часто предпочитают другим средствам разработки приложений для web из-за скорости разработки, простоты изучения языка программирования (ActionScript) и возможности легко создавать сложные и красивые эффекты. Adobe Flash позволяет работать с векторной, растровой и ограниченно с трёхмерной графикой, а также поддерживает двунаправленную потоковую трансляцию аудио и видео.

Платформа Adobe Flash включает в себя ряд средств разработки, прежде всего Adobe Flash Professional и Adobe Flash Builder, а также программу для воспроизведения flash-контента — Adobe Flash Player.

Одним из последних новшеств Flash является возможность упаковать приложение в мобильные форматы для iPhone и Android. Для КПК и других мобильных устройств выпущена специальная «облегчённая» версия платформы Flash Lite, функциональность которой ограничена в расчёте на возможности мобильных устройств и их операционных систем.

Стандартным расширением для скомпилированных flash-файлов (анимации, игр и интерактивных приложений) является .SWF. Расширение FLA соответствует формату рабочих файлов в среде разработки.

Вставка флэш в страницу

При вставке .swf ролика в страницу необходимо помнить, что Flash Player часто по разному вставляется в IE, и в другие браузеры. Необходимо проверять наличие необходимой версии Flash Player у пользователя

Пример кода для вставки в страницу .swf ролика:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=5
,0,0,0" width="300" height="150">
<param name=movie value="адрес флэшки">
<param name=quality value=high>
<embed src="адрес флэшки" quality=high
pluginspage="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version
=ShockwaveFlash" type="application/x-shockwave-flash" width="300"
height="150"></embed></object>
```

Для вставки .swf роликов в страницу рекомендуется пользоваться проектом swfobject (http://code.google.com/p/swfobject/). Swfobject позволяет избавиться от типичных ошибок при вставке .swf файлов в страницу и значительно упрощает процесс интеграции flash ролика в страницу.

Язык ActionScript

ActionScript выполняется виртуальной машиной AVM, которая является частью проигрывателя Flash Player и пакета AIR. Код ActionScript обычно преобразуется компилятором в формат байт-кода. Байт-код встроен в SWF-файлы, которые исполняет проигрыватель Flash Player и пакет AIR. Среди примеров компиляторов — компилятор, встроенный в Adobe Flash Professional, компилятор, встроенный в Adobe Flash Builder, а также компилятор, доступный в компоненте Adobe Flex SDK.

Пример hello world на языке ActionScript 3.0

```
package {
import flash.display.Sprite;
import flash.text.TextField;
public class HelloWorld extends Sprite {
   public function HelloWorld() {
      var txtHello:TextField = new TextField();
      txtHello.text = "Hello, world";
      addChild(txtHello);
   }
}
}
```

Видео-аудио проигрыватели

Одним из самых частых способов использования технологии flash в интернете является проигрывание видео и аудио треков. В большинстве случаев для интеграции видео в веб страничку достаточно уже готовых интернет проигрывателей. Рассмотрим, например, один из популярных и свободно распространяемых проигрывателей: JW FLV Player (http://www.longtailvideo.com/players/jw-flv-player/)

Для интеграции JW FLV Player на сайт достаточно проследовать 4-м шагам:

1. Скачать архив с плеером с сайта проигрывателя

2. Загрузить файлы jwplayer.js и player.swf из архива на ваш сервер.

3. Подключите jwplayer.js на странице на которой вы собираетесь его использовать.

Например можно сделать так:

<script type="text/javascript" src="/jwplayer/jwplayer.js"></script>

4. Проинициализируйте плеер где-либо в коде вашей страницы. Сделать это можно следующим образом

Loading the player ...
<script type="text/javascript"> jwplayer("container").setup({});
flashplayer: "/jwplayer/player.swf", file: "/uploads/video.mp4", height: 270,width: 480
</script>

Более детальные инструкции можно найти в документации по плееру.

Flex

Adobe Flex — технология для легкого и очень быстрого создания Rich Internet Applications, использующая описание интерфейса приложения с помощью диалекта XML — MXML. Flex приложение может компилироваться на сервере, а может — из IDE как во Flash, результатом является swf файл, исполняемый с помощью Flash Player.

Flex SDK - это большой набор классов, расширяющих возможности Flash. Flex-framework включает возможности локализации, стилизации приложения, разработки модульного приложения, встроенные валидаторы и форматоры текстовых полей — все те инструменты, которые нужны разработчикам приложений, работающих online.

Adobe предоставляет бесплатную интегрированную среду разработки на Flex: Flash Builder. Adobe Flash Builder создана на свободно распространяемой платформе разработки Eclipse, которую многие разработчики уже используют при программировании на Java.

Flex, помимо скорости разработки, предоставляет полные мультимедийные возможности Flash платформы: включая потоковое мультимедиа, возможность получить доступ к веб-камере и микрофону пользователя, бинарные сокеты, обширные возможности сетевых коммуникаций (HTTP-запросы, веб-сервисы, встроенный формат сериализации AMF), оперирование координатами трехмерного пространства,

возможности использования встроенных фильтров (таких как расфокусировка, падающая тень и др.), и написания собственных.

Если сильно обобщить, то Flex – это сложные приложения для веб и декстопов, в то время как Flash – это анимации, баннеры и простые игры.

Язык клиентских скриптов JavaScript

Язык JavaScript будет рассмотрен ниже. Приведем здесь несколько характеристик. JavaScript - язык управления сценариями просмотра гипертекстовых страниц. JavaScript позволяет получить доступ к библиотекам объектов браузеров, динамически формировать Web-страницы. JavaScript позволяет выполнять на стороне клиента многие функции, которые ранее выполнялись на стороне сервера. Примером является проверка заполнения форм до того, как пользователь передаст информацию Web-серверу. JavaScript позволяет читать и записывать файл cookie, который содержит информацию о пользователе, посылаемую веб-сервером на компьютер пользователя, для сохранения и использования в следующем сеансе связи.

К возможностям JavaScript можно, отнести следующее: отображение таких изменяющихся данных, как текущее время или дата; программирование переменного содержания в зависимости от текущей даты, версии браузера пользователя или других условий; изменение внешнего вида элементов страницы при возникновении события; выполнение вычислений на клиентской странице.

JavaScript является скриптовым языком, программы обходятся без компиляции. JavaScript вместе с XML и библиотекой JQuery образует основу популярной технологии AJAX.

Технология «клиент-сервер»

Приложения WWW работают по технологии «клиент-сервер», в которой все программное обеспечение разделяется на клиентскую и серверную части. Взаимодействие клиента и сервера происходит по принципу «запрос-ответ» . Клиент посылает запрос, сервер обрабатывает его и посылает ответ. Рассмотрим, например, этапы соединения по протоколу http.

- Запрос клиента. Браузер формирует запрос на основе данных из URL пользователя, либо из данных формы.
- Установка соединения клиента с сервером
- Посылка запроса клиента и ожидание ответа от сервера.
- Обработка запроса сервером. Генерация ответа.
- Прием ответа клиентом.
- Разрыв соединения.

Пока нет обращений от клиентов, сам HTTP-сервер просто «спит», установив прослушку заданного порта (80). Когда клиент устанавливает соединение, сервер «просыпается» и, приняв данные запроса, приступает к их обработке. Результат всех манипуляций - это выдача ответа, которого ожидает клиент. После того как сервер выдал ответ, он разрывает соединение и вновь «погружается в сон». Естественно отметить, что в случае возникновения ошибки HTTP-транзакция может закончиться на любом из этих этапов.

Большое количество Web-приложений основано на использовании внешних программ, управляемых Web-сервером. Использование этих программ позволяет строить Web-приложения с динамически обновляемой информацией, хранящейся в базах данных или генерирующейся в зависимости от бизнес-правил решаемых задач. Программа-шлюз запускается WWW-сервером, который обеспечивает передачу запроса пользователя

шлюзу. Он в свою очередь, используя средства прикладной системы, возвращает результат обработки запроса клиенту.

Программирование для серверов

Программирование на стороне сервера позволяет получать и обрабатывать на сервере запросы, введенные на клиенте при помощи формы или с помощью URI. В список других задач входят: обеспечивать динамический доступ к серверным базам данных и другим ресурсам, хранимым на сервере; динамически создавать web-документы в качестве ответа клиенту; использовать серверные компоненты, предназначенные для решения типовых задач; осуществлять аутентификацию пользователя и получать информацию о браузере клиента.

Язык Java на клиентской и серверной странице

Язык Java зародился как часть проекта создания программного обеспечения для различных бытовых приборов, телевизоров, автомобилей. Для этих целей был необходим платформенно-независимый язык программирования, позволяющий создавать программы для различных устройств и различных операционных сред. Способность Java исполнять свой код на любой из поддерживаемых платформ достигается тем, что ее программы транслируются в некое представление, называемое байт-кодом. Байт-код может интерпретироваться в любой системе, в которой есть среда выполнения Java.

Язык Java предоставил на клиентскую страницу апплеты - небольшие, динамичные, не зависящие от платформы сетевые приложения. Однако, в настоящее время язык Java ушел с клиентских страниц и перебрался на серверные страницы. Его место занял JavaScript. Наряду с этим широкое распространение получило еще одно быстро развивающееся направление использования Java – мобильная телефония.

Для работы на клиентской странице развивается направление Java FX.

Глава 4. Информационной обмен, HTML и CSS

Интернет является средством коммуникации, средством обмена информацией, средством хранения и источником информации. В сети используются все основные формы представления информации: аудио, видео, текстовая информация. При обработке информации обычно выделяют три уровня: моделирование, передача, интерпретация информации.

На этапе моделирования создается информационный объект, содержащий данные. На этом этапе выделяются следующие проблемы:

- Анализ предметной области. При этом ищется ответ на вопрос: «Какую именно информацию мы хотим передать».
- Создание правил формализации. «Какая формальная схема, какой способ позволяет лучше всего передать намерения, смысл? Например, музыка, живопись или текст»
- Кодирование в терминах правил. «Как эффективнее выразить информацию в выбранном формализме? »

Второй этап - это этап передачи информации по указанному адресу. Передача может осуществляться с помощью голоса, визуально, явной формализации в тексте. Нас интересует передача информации с использованием протоколов передачи данных Интернет. При этом нужно обеспечить передачу данных полностью и, желательно, без искажений.

Последний этап является уровнем интерпретации информации. На этом уровне адресат производит анализ данных и выделяет заложенный в них смысл с помощью специальных правил, тем самым, реконструируя исходные знания. На уровне интерпретации выполняется раскодирование пакета. При этом выбираются формальные

правила для извлечения знаний и смысла из пакета. Интерпретация подразумевает интерпретирование извлеченного смысла, реконструкция модели знаний. Дальнейшие действия с информацией предполагают обогащение данных и их последующая передача.

Интерпретация основана на заданных или предполагаемых правилах и подбирается в каждом случае. Например, часто сложно впервые прочитать текст, написанный незнакомым подчерком, понять иностранную речь с выраженным акцентом.

Каждый из уровней важен при передаче информации. Гипертекстовая информация, передаваемая в Веб, предусматривает описание и логическую разметку документа на основе языков HTML и XHTML. В отображении документа важную роль играет язык каскадных стилей CSS.

Язык разметки гипертекста HTML

Язык HTML не является языком программирования, это язык разметки гипертекстовых документов. HTML определяет логическую структуру документа. Вид отображения документов устанавливается средствами CSS.

Стандарт HTML разработан под руководством консорциума World Wide Web (W3C, http://www.3w.org). Действующая до последнего времени редакция языка - HTML 4.01 вышла в 1999 году. Эту версию часто называют динамический HTML (DHTML). DHTML использует объектную модель документа (DOM), которая описывает способ организации всего документа как дерева объектов, а также определяет, какие свойства могут быть изменены и какие значения они могут принимать. Событийная модель описывает способ передачи управления сценариям на языке JavaScript. С использованием динамического HTML сценарии могут вставлять, удалять и заменять узлы и блоки HTML. Браузер автоматически отображает полученный новый HTML-код.

В настоящее время широкое распространение получила новая версия HTML 5.0. Эта значительно улучшенная версия HTML поддерживается всеми браузерами.

Язык HTML используется для создания самых разных гипертекстовых документов с гиперссылками и элементами мультимедиа — Web-страниц, интерфейсов, презентаций, электронных книг и учебных пособий. HTML-документ создается с помощью HTML-редактора и сохраняется в виде текстового файла со стандартным расширением .html или .htm. Для просмотра HTML-документов используются Web-браузеры, интерпретирующие эти документы.

Разметка гипертекста средствами HTML происходит путем вставки в текст специальных элементов, которые называют тегами или дескрипторами. Это специальные кодовые слова, определяющие элементы форматирования, например: <b>, <a>, <img>. Рассмотрим пример документа:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Моя первая веб-страница</title>
</head>
<body>
<h1>Заголовок страницы</h1>
<p>Основной текст.</p>
</body>
</html>
```

В HTML документе используются теги в скобках <>, которые могут быть контейнерные (парные) и неконтейнерные (одинарные). Контейнерный тег состоит из

открывающего (старт-тега) и закрывающего (стоп-тега), между ними находится форматируемый текст, например:

<html> текст html документа </html>

Закрывающий тег отличается от открывающего наличием косой черты. Контейнерные теги бывают вложенными, например:

 курсив жирный курсив курсив

При использовании вложенных тегов нужно следить за их последовательностью: тег, открытый первым, закрывается последним.

Одинарный тег состоит только из открывающего и не требует закрывающего тега. Встретив, например, тег <hr>, браузер выведет на экран горизонтальную разделительную линию. По правилам XHTML присутствие закрывающего тега обязательно, например, в конце одинарного тега можно поставить />.

DOCTYPE.

В начале html – документа помещается тег DOCTYPE, содержащий информацию о типе документа и используемой версии HTML. Такое указание позволяет браузеру проверить структуру документа на соответствие правилам и правильно его отобразить. В HTML 4.01 существуют три варианта правил (dtd – описаний, загружаемых с сайта w3.org), устанавливаемых в DOCTYPE: Strict, Transitional и Frameset.

Строгий (Strict) - документ не содержит элементов, помеченных как «устаревшие» или «не одобряемые» (deprecated), например: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

Переходный (Transitional) –может содержать устаревшие теги в целях совместимости для старых версий HTML: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

С фреймами (Frameset) – может содержать теги для создания наборов фреймов:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">

Для актуальной последней версии HTML5 элемент DOCTYPE выглядит наиболее просто:

<!DOCTYPE html>

Теги, атрибуты, значения

Вслед за названием тега через пробел могут помещаться атрибуты (параметры) тега и их значения.

<тег атрибут1=”значение” атрибут2=”значение”> Текст </тег>

Атрибуты тега модификацируют его действие. Атрибуты бывают обязательные и необязательные. Правила записи атрибутов и значений:

- атрибуты отделяют друг от друга пробелами
- порядок следования атрибутов произволен
- значения атрибутов записывают в ординарных или двойных кавычках после знака равенства. Спецификация html позволяет опускать кавычки, если значение атрибута состоит из одного слова из латинских букв и цифр. Если же значение атрибута содержит несколько слов, разделенных запятыми или пробелами, то кавычки обязательны.

В качестве примера приведем атрибут src=”img” который может использоваться для вставки изображения.

В HTML - документе можно использовать и верхний, и нижний регистры для написания тегов. В XML и XHTML используются только строчные буквы и значения атрибутов в кавычках.

Структура документа

Любой html-документ заключается между тегами <html> и </html. Между ними располагается заголовочная часть документа <head> и его тело <body>.

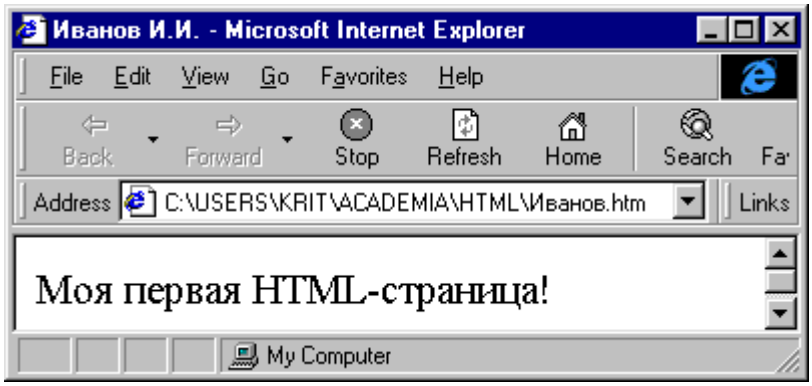
```
<!DOCTYPE html>
<html>
<!—Это заголовок документа, комментарий-->
<head>
<title>Название </title>
</head>
<!—Это тело документа-->
<body>
<!--содержание документа-->
</body>
</html>
```

Технически стартовые и завершающие теги <html>, <head> и <body> необязательны. Однако настоятельно рекомендуется их использовать, поскольку использование данных тегов позволяет Web-браузеру разделить заголовочную и смысловую часть документа.

Заголовочная часть документа

Тег заголовочной части документа <head> используется сразу после тега <html>. В заголовочной части описываются общие правила отображения документа в браузере, содержится вспомогательная информация о документе, а также информация для индексирования документа поисковыми машинами. В частности, тег meta содержит служебную информацию для браузера, не отображаемую в окне браузера. Заголовок документа, ограниченный тегами <title> и </title>, отображается вверху экрана. Рассмотрим пример:

```
<!DOCTYPE html>
<html>
<!—Это заголовок документа-->
<head>
<title> Иванов И.И. </title>
<meta name=”author” content=”Pivovarov”>
 </head>
<!—Это тело документа-->
<body>
Моя первая HTML-страница!
</body>
</html>
```



Тело документа

Тело документа служит контейнером для контента сайта, а также параметров отображения сайта и размещается между тегами <body> и </body>. Это та часть документа, которая отображается как текстовая и графическая информация документа.

Тег <body> может иметь атрибуты, которые в настоящее время являются устаревшими. Вместо них рекомендуется использовать таблицы стилей CSS.

АТРИБУТЫ ТЕГА <BODY>:

Атрибуты	Описание
background	задает изображение, которое как черепица заполнит фон документа: <body background="(url)(путь) имя файла">
bgcolor	цвет фона. <body bgcolor="#ff0000"> или <body bgcolor="red">
text	цвет текста. По умолчанию черный <body text=" цвет ">
link	цвет гиперссылки (темно синий). <body link="цвет" >
alink	цвет активной гиперссылки. Синтаксис: <body alink="цвет" >
vlink	задает цвет посещенной гиперссылки: <body vlink="цвет" >
topmargin	задает верхнюю границу: <body topmargin=”число”>
bottommargin	задает нижнюю границу страницы : <body bottommargin=”число”>
leftmargin	задает границу страницы слева: <body leftmargin=”число”>
rightmargin	задает границу страницы справа: <body rightmargin=”число”>

```
<!DOCTYPE html>
<html>
<head>
<title>pr2 - Атрибуты тела Body</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<!—Это тело документа-->
<body background="ice.png" bgcolor="yellow" text="red"  link="green" >
```

Задает изображение, которое как черепица заполнит фон документа. Устанавливает цвет фона, цвет текста,

цвет гиперссылки (по умолчанию темно синий).

```
<h1>Заголовок страницы</h1>
<p align=center>Основной текст.</p>
</body>
</html>
```

Использование элемента style

Тег <style> используется для определения стиля документа HTML. В элементе style указывается, как элементы HTML должны представляться в браузере. Рассмотрим пример стилей для <body>, <h1> и <p>

```
<!DOCTYPE html>
</html>
<html>
<head>
<meta charset="utf-8" />
<style type="text/css">
body { background-color:#d0e4fe; }
```

```
h1 { color:orange; text-align:center; }
p {font-family:"Times New Roman"; font-size:20px;}
</style>
</head>
<body>
<h1>Пример CSS!</h1>
<p>Это параграф.</p>
</body>
</html>
```

В HTML5 введен новый атрибут, он позволяет определить стили для определенного раздела, а не всего документа. Если атрибут "scoped" указан, стили применяются только к дочерним элементам родителя элемента style.

Структура документа HTML 5

Отличия HTML 5 начинаются уже в теге <!DOCTYPE html >. В HTML5 введены новые теги для улучшения структуры документа:

header: в этом теге размещают заголовок секции;

footer: футер(подвал)- размещается обратная ссылка;

nav: отдельный тег для навигации;

section: тег относится к описанию части документа, и может использоваться с тегами h1-h6, а также быть частью статьи;

article: описывают запись или новость;

aside: описывают данные, не связанные с основным контентом;

dialog:тегом может быть отмечен разговор или интервью;

figure: в теге заключают видео, графику или аудио.

```
<!DOCTYPE html>
<!--Html-5 документ, comment -->
<html>
<head>
  <meta charset="utf-8">
  <title> Html-5 документ, заголовок</title>
</head>
<body>
 <header>...</header>
 <nav>Навигация</nav>
 <article>
   <section>
    Секция
   </section>
 </article>
 <aside>...</aside>
 <footer>Подвал</footer>
</body>
```

Чтобы новые элементы HTML5 отображались правильно как элементы уровня блоков, можно назначить им стиль:

header, footer, article, section, nav, menu, hgroup {
display: block;
}

HTML 5 вводит несколько новых элементов и атрибутов. Некоторые из них являются эквивалентами скобок HTML и ›, но имеют своё семантическое значение, например <nav> и <footer>. Другие элементы предоставляют новую функциональность, как <audio> и <video>. Пятая версия чётко прописывает правила лексического разбора, чтобы различные браузеры отображали один и тот же результат в случае некорректного синтаксиса. Некоторые устаревшие элементы HTML 4, такие как <font> и , были удалены из HTML 5. Пример страницы HTML 5:

```
<!DOCTYPE html>
<!--Html-5 документ, comment -->
<html>
<head>
  <meta charset="utf-8">
  <title> Html-5 документ, заголовок</title>
</head>
<body>
 <header>
<a href="/"><img src=img/logo.gif alt="home"></a>
<a href="/"><img src=img/webconf09.gif alt="home"></a>
<hgroup>
<h3>Header - Демонстрация</h3> <h6>Демонстрационный сайт</h6>
</hgroup>
</header>
 <nav>
<ul>
<lh>Навигация</lh>
<li> Раз</li>
<li>Два </li>
<li> Три</li>
</ul></nav>
 <article> <h1>Article</h1>
  <section>
   <h1>Секция-session</h1>
  </section>
 </article>
 <aside><h1>Aside</h1></aside>
```

```
 <footer><h1>Подвал-footer</h1>
<small>&copy;2012 Romanchik Valery<small>
</footer>
</body>
```

Результат:

Форматирование текста в HTML

Документ HTML отображается браузером с некоторыми особенностями.

Любое количество пробелов идущих подряд, отображается как один пробел. Лишние пробелы не изменят вид документа в браузере. (Исключением является тег <pre> </pre>, внутри которого любое число пробелов отображается так, как оно указано в коде.) Это же правило относится к и переносам. HTML не поддерживает расстановку переносов в словах. Короткие строки для выравнивания могут растягиваться за счет автоматического добавления пробелов между словами. Текст занимает ширину окна браузера. Длинная строка будет отформатирована, чтобы текст поместился по ширине в окно. Переносы текста будут добавлены автоматически в местах пробела или дефиса.

Абзац <p>

Поскольку символы перехода на новую строку, используемые для разбиения текста на абзацы, воспринимаются как обычные пробелы, для создания абзацев в HTML-страницах используются теги параграфа— <p>. С помощью тегов <p> текст разбивается на абзацы (параграфы), согласно традициям американской полиграфии — без красной строки и с увеличенным отступом между абзацами. Это означает, что тег добавляет пустой отступ перед строкой.

Для выравнивания абзаца — по левому краю, по правому краю, по центру и по ширине — используется атрибут дескриптора <p> — align, принимающий значение left, right, center и justify, соответственно. Так, для выравнивания абзаца по центру используется следующий код:

```
<html>
<head>
<title>Применение абзацев</title>
</head>
<body>
 <p align = “center”> Абзац, выровненный по центру. </p>
<p align = “left”> Абзац, выровненный по левому краю. </p>
<p align = “ justify ”> Абзац, выровненный по ширине. </p> </body>
 </html>
```

Следует отметить, что, несмотря на то, что дескриптор <p> — контейнерный, указывать для него закрывающий дескриптор не обязательно, хотя и желательно.

Разрыв строки

Тег
 извещает браузер о разрыве строки и не добавляет пустой отступ перед строкой. Атрибут тега
 clear позволяет при обтекании текстом изображения, выполнить обтекание слева (left), справа (right) или обеих (all) границ окна браузера. Например, если рядом с текстом слева встречается рисунок, то можно использовать тег <br clear=”left”> для смещения текста ниже рисунка.

Если требуется, чтобы браузер автоматически не переносил какую – то определенную строку, можно обозначить ее тегами <nobr> и </nobr>. В этом случае браузер не будет переносить строку, даже если она выходит за границы экрана, а позволит горизонтально прокручивать окно. Если вы хотите все же позволить разбиение данной строки на две, но в строго запланированном месте, то вставьте тег <wbr> в это

место. Например: <nobr>. Данная строка является самой длинной строкой документа <wbr>, которая не допускает какого-либо разбиения где бы то ни было </nobr>.

К другим способам форматирования текста относят использование символа неразрывного пробела &nbsr. Неразрывные пробелы записываются вместо обычных и позволяют выводить текст в одной строке, независимо от ширины окна браузера.

Устаревший тег был предназначен для выравнивания текста. Вместо него для выравнивания блока может использоваться тег

 блок или style для css.

Предварительное форматирование

Как уже отмечалось, дополнительные пробелы, символы табуляции и возврата каретки в тексте HTML-документа будут проигнорированы браузером при интерпретации документа. HTML-документ может включать такие элементы, если они помещены внутрь тегов <pre> и </pre>. Эти теги используются, чтобы текст выглядел так, как был набран.

Заголовок <hn>

Для выделения заголовков более крупным и жирным шрифтом в HTML используются теги <hn>, где n — цифра от 1 до 6. Заголовок 1 уровня выводится самым крупным шрифтом, заголовок 6 уровня — самым мелким. Заголовки и следующий за ними текст всегда начинается с новой строки. Для форматирования заголовков используется атрибут align, для которого предусмотрены три значения — left, right и center. Теги заголовков, в отличие от тегов <p>, обязательно нужно закрывать. Иначе весь текст до конца страницы будет считаться заголовком.

Пример:

```
<html>
 <head>
 <title>Применение абзацев</title>
</head>
<body>
<h1>Заголовок первого уровня</h1>
<h2>Заголовок второго уровня</h2>
<h3>Заголовок третьего уровня</h3>
<h4>Заголовок четвертого уровня</h4>
<h5>Заголовок пятого уровня</h5>
<h6>Заголовок шестого уровня</h6>
</body>
</html>
```

Комментарии

Для удобства чтения и правки кода, используются комментарии, не изменяющие отображение страницы в браузере. Комментарий в HTML-коде заключается между символами <!-- и -->: <!-- комментарий -->

Физическая и логическая разметка документа

Все теги разметки можно условно разделить на две категории "физические" и "логические", т.е. такие, которые определяют только внешний вид документа, и такие, которые несут информацию о его логической структуре.

«Физические» теги:

Тег	Значение
...	Курсив (Italic)
...	Жирное начертание шрифта (Bold)
<big>...</big>	Увеличенный размер шрифта

...	Уменьшенный размер шрифта
_{...}	Подстрочные символы
^{...}	Надстрочные символы

Пример:

```
<html>
<head>
<title> Создание жирного текста</title>
<body>
<b><small>Текст маленький и жирный < /small></ b>
<br><!—разрыв строки -->
<p>Надстрочные <sup> символы </sup></p>
</body>
</html>
```

Используемые ранее для форматирования текста теги <u>, <s>,<tt> являются устаревшими и не рекомендуемыми к использованию в настоящее время.

«Логические» теги:

Тег	Значение
<ем>...</ем>	Типографское усиление
...	Усиление
<code>...</code>	примеры кода,например, "коды программ"
<samp>...</samp>	Последовательность литералов
<квd>...</квd>	Пример ввода символов с клавиатуры
<var>...</var>	Переменная
<dfn>...</dfn>	Определение
<q>...</q>	Текст, заключенный в двойные кавычки

Несмотря на то, что многие "логические" теги дублируют друг друга в смысле внешних эффектов, они полезны: благодаря им, во-первых, упрощается анализ страницы поисковыми системами, а во-вторых, достигается единообразие оформления страниц и их соответствие полиграфическим правилам и традициям.

Тег

Теги , <basefont> , center относятся к отображению документов и являются устаревшими. Их использование вместо CSS крайне нежелательно. Здесь они приведены в историческом плане для случаев рассмотрения старых версий документов.

Текст красного цвета, шрифт на 3 единицы крупнее предыдущего

Специальные символы

Символы, которые не могут быть введены в текст документа непосредственно через клавиатуру, называются специальными символами. Символы – знак меньше <, знак больше >, амперсант & и двойные кавычки “” имеют специальное значение внутри HTML и, следовательно, не могут быть использованы в тексте в своем обычном значении. Для таких символов существуют особые теги.

Специальный символ	
<	<
>	>
&	&
"	"

Торговая марка TM	®
Copyright	©
Не разделяющий пробел	

Линии

Тег <hr> проводит контурную горизонтальную линию (опция shade по умолчанию). Например:

<hr noshade> – горизонтальная линия с тенью;

Тег <hr> имеет атрибут align, который задает выравнивание по левому краю - left, центру - center или по правому краю – right.

Атрибут size устанавливает толщину линии в n пикселей, где n от 1 до 175 (по умолчанию n=2).

Пример:

<hr width=75% align=”right” size=”30px”>
<!-- ширина линии 75%, выравнивание вправо, толщина 30px -->

Пример:

```
<!--пример: линии с разным выравниванием-->
<html>
<head><title>примеры горизонтальных линий </title></head>
<body>
<b> Стандартная линия, задаваемая тегом &lthr&gt: </b>
<hr>
<p>
<b> Линия, заданная тегом &lthr&gt с параметром noshade: </b>
<hr noshade>
<b> Линия шириной 50% и с выравниванием по левому краю: </b>
<hr width=”50%” align=”left”><p>
<b> Линия шириной 25% и с выравниванием по центру: </b>
<hr width=”25%” align=”center”>
<p>
<b>Линия шириной 75% с выравниванием по правому краю:</b>
<hr width=”75%” align=”right”>
</body>
</html>
```

Графика

Для размещения графических элементов на Web-странице используется следующая технология: изображение сохраняется в отдельном файле, а в HTML-код вставляется ссылающийся на него тег <img>. Имя файла, в котором находится изображение, и путь к нему определяются атрибутом src.

<img src=”fract.gif”>

Задавая путь, нужно руководствоваться правилами, принятыми для записи URL. Эти правила отличаются от правил записи пути к файлу в ОС Windows. В частности, имена папок и файлов разделяются не обратными, а прямыми слешами. Строчные и прописные буквы различаются. Кроме того, в именах файлов, передаваемых по интернет, не рекомендуется использовать пробелы и кириллицу. Пример: загрузка изображения из каталога уровнем выше:

<img src=../fract.gif>

Атрибутами тега img являются ширина и высота изображения width и height задаваемые в пикселях. Эти атрибуты можно использовать как для изменения параметров изображения, так и для того, чтобы определить их заранее. Второй способ применяется достаточно широко, так как позволяет сразу, не дожидаясь окончательной загрузки изображений, разместить элементы Web-страницы в соответствии с замыслом дизайнера.

<img src=”fract.gif” width=”240” height=”260”>

Часто изображения снабжаются комментирующими надписями, указанными в качестве значения параметра alt. Эти надписи появляются рядом с указателем мыши, наведенным на изображение. Атрибут alt настоятельно рекомендуется использовать во всех случаях

<img src=”fract.gif” alt="Фрактал" width=”240” height=”260”>

Для определения способа обтекания графики текстом используется параметр align. Этот параметр описывает выравнивание не только по горизонтали — по левому краю (значение left) или по правому краю (значение right), — но и по вертикали относительно той строки, в которой размещено изображение. Значение absbottom соответствует выравниванию по нижним выступающим элементам букв в строке, baseline и bottom — по нижнему краю строки, absmiddle и middle — по середине, texttop — по верхнему краю и top — по верхним выступающим элементам. Для отмены обтекания изображения текстом используется тег
 с параметром clear, который принимает значения right (отмена обтекания справа), left (отмена обтекания слева) и all (отмена всех обтеканий).

<img src=fract.gif alt="Фрактал" width=”240” height=”260” align=”left”>

Для отделения изображения от текста используются два средства — рамки и отступы. Атрибут border определяет толщину рамки в пикселях. Цвет такой рамки всегда черный. Для создания более сложных рамок следует использовать графический редактор. Горизонтальный и вертикальный отступы между изображением и текстом задаются в пикселях с помощью параметров hspace и vspace.

<img src=fract.gif alt="Фрактал" width=”240” height=”260” align=”left” border=”1px”>

Изображения для интернета обычно хранятся в файлах форматов GIF и JPEG. В формате JPEG сохраняются полноцветные изображения, в частности качественные фотографии. В формате GIF хранятся изображения с ограниченным количеством цветов, а также анимированные изображения и изображения с прозрачным фоном. Кроме того, в Web-дизайне широко применяется GIF-изображение, представляющее собой 1 пиксель прозрачного "цвета". С его помощью можно, в частности, регулировать расстояние между строками, а также создавать отступы заданной величины.

Ключевое слово lowsrc позволяет сначала загрузить файл с низким разрешением, затем больший файл с высоким разрешением:

<img src="highcar.gif" lowsrc="lowcar.gif">

В примере сначала загружается файл lowcar.gif, а затем highcar.gif.

HTML5 предлагает способ связать текст с элементом изображения с введением элемента <figure>. При комбинировании с элементом <figcaption>, можно семантически связать надписи с их графическими аналогами:

```
<figure>
<img src="path/to/image" alt="About image" />
<figcaption>
<p>This is an interesting image</p>
</figcaption>
</figure>
```

Ссылки

Гиперссылки — важное средство языка HTML, благодаря которому возможны переходы между различными документами, объектами и веб - сайтами, размещенными в интернет. Гиперссылки реализуются с помощью тега <a>. Первым атрибутом тега является href=”адрес ресурса”. Еще один атрибут name= “ссылка в документе” позволяет установить навигацию внутри документа. Атрибут target содержит имя окна, куда будет загружаться документ: target=_blank/_top – загрузка в новое окно; target=_parent – загрузка в родительское окно; target=self – загрузка в текущее окно; target=_search – загрузка в окно поиска. Атрибут title="Пояснительная информация” содержит всплывающую подсказку при наведении курсора на гиперссылку. Атрибут charset=”кодировка” указывает с помощью какой кодировки следует открыть сетевой ресурс.

В следующем примере атрибут href, определяет ссылку на другой HTML-документ, хранящийся в файле:

```
<a href="Minsk.html">Minsk</a>
```

Если файл, на который указывает ссылка, может быть открыт в браузере, он открывается. В противном случае выполняется загрузка этого файла с сохранением его на диске. Если документ, формирующий ссылку, находится в другой директории, то подобная ссылка называется относительной. Например:

```
<a href="Belarus/Minsk.html">Minsk</a>
```

Ссылки можно формировать на основе URL, например:

```
<a href="http://www.w3.org/tr/rec-html40"> документ html </a>
```

Отдельного внимания заслуживает применение гиперссылок для отправки писем по электронной почте. Тег <a> позволяет составить шаблон электронного письма, которое затем посетитель страницы может отправить по адресу, указанному в параметре href.

```
<!--пример: создание ссылок на html-файлы-->
<html>
<head><title>Ссылки на домашней странице</title></head>
<body>
<h1>Внутренние ссылки на части документа</h1>
<hr> Если вас интересует подробная информация, пишите по адресу
<a href="mailto:rom@bsu.by">rom@bsu.by</a>
<hr>
<h2>вы можете:</h2>
<ul>
<li>Посмотреть <a href="pr11.htm">простейший пример</a></li>
<li>Посмотреть <a href="pr12.htm">второй пример</a></li>
<li>Посмотреть <a href="pr13.htm">разбиение текста</a></li>
<li>Узнать <a href="pr14.htm">о линиях</a></li>
</ul>
</body>
</html>
```

В следующем примере рассматривается элемент address который несет в себе контактную информацию для всего сайта или его части. Элемент address не используют для всех e-mail адресов; сюда помещают только контактную информацию людей, связанных с документом.

```
<p>Над сайтом работали:</p>
<address>
<a href="http://alexander.com/">Александр</a>,
```

Вася,
Тузик,
</address>

Для того чтобы сослаться не просто на страницу, а на ее определенное место, используются закладки или якоря. С помощью якоря - тега <a> с атрибутом name, помечается место куда может быть осуществлен переход. Чтобы осуществить переход на якорь используется параметр href = “имя закладки”. Имя закладки содержит URL документа, за которым следует имя якоря. Разделителем между URL и именем якоря служит символ #. Если якорь находится на текущем документе, то URL не указывается.

```
<!--пример: Ссылки на якоря-->
<html>
<head><title>якоря на домашней странице</title></head>
<body>
<!-- создание ссылок на якоря -->
<ul>
<lh>Содержание</lh>
<li><a href="#section1">введение</a></li>
<li><a href="#section2">обзор</a></li>
<li><a href="#section3">подробное рассмотрение</a></li>
</ul>
<br>
<hr>
...тело документа...
<hr>
<h2><a name="section1">Введение</a></h2><hr>
...section 1...
<hr><!-- установка якорей -->
<h2><a name="section2">Обзор</a></h2><hr>
...section 2...
<hr>
<h3><a name="section3">Подробное рассмотрение</a></h3><hr>
...section 3...
<hr>
 <a href="mailto:romanchik@bsu.by">
<address>романчик - e-mail:rom@bsu.by</address></a>
</body>
</html>
```

Такой же эффект можно получить, используя заглавия вместо якоря:

```
<!--пример: заглавия вместо якоря-->
<html>
<head><title>ссылки на заголовки</title></head>
<body>
<h1>table of contents</h1>
<a href="#section1">introduction</a><br>
<a href="#section2">some background</a><br>
<a href="#section3">the first part</a><br>
...the rest of the table of contents...
```

<h2 id="section1">introduction</h2> <hr>
...section 1...<hr>
<h2 id="section2">some background</h2><hr>
...section 2...<hr>
<h3 id="section3">the first part</h3><hr>
...section 3...<hr>
...продолжение документа...
</body></html>

Кроме описанных свойств, тег <a> с помощью атрибута target позволяет выбрать окно, в котором будет открыт новый объект.

Вместо имени гиперссылки можно указать изображение в виде:
<img src="image/img.gif" height="100" width="100">. Щелкнув по изображению можно перейти на указанный адрес. Пример:

```
<html>
<head>
<title> Создание рисунка-ссылки</title>
</head>
<body>
<a href="sample.html"><img src="image/img.gif"
border="0" height="100" width="100"></a>
</body>
</html>
```

Гиперссылкой может служить не только целое изображение, но и его фрагменты. Для этого создаются так называемые карты изображений. Карта изображения — это конструкция языка HTML, образованная с помощью тегов <map> и <area>. Тег <map> определяет имя карты, по которому она впоследствии связывается с самим изображением.

В параметре usemap тега <img> указывается имя, присвоенное с помощью параметра name дескриптора <map>, предваряемое символом #.

Внутри конструкции <map>... </map> помещаются теги <area>. Каждый такой тег описывает параметры определенной области внутри изображения, которой поставлена в соответствие та или иная гиперссылка. Этими параметрами являются форма, координаты, адрес объекта, на который указывает ссылка. Кроме того, здесь, как в теге <a>, действует параметр target и, как в теге <img>, — параметр alt.

Форма области определяется параметром shape, который принимает одно из четырех значений: rect (по умолчанию), circ, poly и default. Эти значения соответствуют прямоугольнику, кругу, многоугольнику и всей области изображения. Для каждой из первых трех форм предусмотрена своя система задания координат. Координаты области определяются параметром coords. Значением этого параметра служит текстовая строка, где через запятую перечислены значения координат (в пикселях), однозначно определяющие размер и положение данной области. Для круга это координаты центра и радиус, для прямоугольника — координаты левой, верхней, правой и нижней сторон, а для многоугольника — координаты вершин.

Пример:

```
<html>
<head>
<title> Создание карты-изображения</title>
</head>
<body>
```

<img src=”map.gif” width=300 height=”100” border=”0” usemap=”#map” alt="Навигация по сайту">
<map name=”map”>
<area shape=” rect” coords="0,0, 100,100" href=linkl.html>
<area shape=” rect” coords="100,0, 200,100" href=link2.html>
<area shape=” rect” coords="200,0, 300,100" href=link3.html>
</map>
</body>
</html>

Списки

В HTML используются маркированные и нумерованные списки. Каждый список размечается с помощью тегов двух типов: первый определяет параметры всего списка, второй — параметры каждого элемента.

Маркированные списки описываются дескриптором <ul>. Его атрибут type определяет вид маркера — квадратики (square), кружки (disc) и "пустые" окружности (circle). По умолчанию параметру type присваивается значение disc.

Нумерованные списки описываются дескриптором <ol>. Этот дескриптор имеет два параметра: type, определяющий способ нумерации, и start, определяющий, с какой буквы или цифры она будет начинаться. Параметр type дескриптора <ol> принимает значения 1, I, i, A или a, что соответствует нумерации арабскими, большими и малыми римскими цифрами, а также большими или малыми латинскими буквами.

Каждый злемент нумерованных или маркированных списков размечаются затем с помощью дескрипторов <li>. Этот тег имеет те же атрибуты что и тег всего списка: если список нумерованный, то это type и start, а если маркированный, то только type. Параметры дескриптора <li> имеют более высокий приоритет, чем параметры всего списка и, таким образом, позволяют изменить порядок нумерации или вид маркера.

Для организации многоуровневых списков со смешанной нумерацией используются вложенные дескрипторы <ul> и <ol>: вместо очередного блока <li>... ставится соответствующий вложенный список.

<!--пример: Ненумерованные и нумерованные списки-->
<html>
<head>
<title>Использование списков </title>
</head>
<body>
<h3 align=”center”>Домашняя страница </h3>
<h4>Маркированный (ненумерованный) список - мои интересы:</h4>

<lh>занятия в свободное время:</lh>
 компьютеры
 чтение книг
 бассейн
 отдых на природе
 <hr>
<h4> Нумерованный (упорядоченный) список:</h4>
<ol type=”1”>
<lh>страны, которые я посетил:</lh>
 польша
 чехия

```
<li> германия</li>
</ol> <hr>
<ol type=”a”>
<lh><b>страны, которые я хочу посетить:</b></lh>
<li> Америка</li>
<li> Китай</li>
<li> Франция</li>
</ol>
</body>
</html>
```

Тег <lh> используется для задания заголовка списка.

Кроме маркированных и нумерованных списков, в HTML предусмотрена конструкция, образующая список определений. Каждый элемент такого списка состоит из некоего термина и его определения. Термины и определения находятся в отдельных абзацах, причем последние выводятся с увеличенным горизонтальным отступом относительно остального текста. Разметка списка определений осуществляется с помощью трех дескрипторов — <dl>, <dt> и <dd>. Дескриптор <dl> описывает весь список в целом, <dt> — определяемый термин, а <dd> — определение.

```
<!--пример: списки определений-->
<html>
<head><title>использование списков</title></head>
<body>
<h2 align=”center”>Толковый словарь</h2>
<dl>
<lh><big><b> список терминов</b></big></lh>
        <dt><b>"Anchor"</b></dt>
<dd><i>то, что образовывает гипертекстовую ссылку</i></dd>
        <dt><b>"Lamer"</b></dt>
<dd><i> юзер-идиот</i></dd>
        <dt><b>"Cookies "</b></dt>
<dd><i>технология, позволяющая сохранять индивидуальную информацию о пользователе сети</i></dd>
</dl>
</body>
</html>
```

Таблицы

Теги <table> ... </table> служат контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк, которые задаются с помощью тегов <tr>. Как в контейнеры <tr> помещаются столбцы <td>. Тег <th> определяет заголовок для каждого столбца.

Ячейки таблицы описываются построчно, слева направо. Для слияния горизонтальных и вертикальных смежных ячеек применяются параметры тега <td> — colspan и rowspan.

Атрибуты тега <table> :

Align - выравнивание таблицы; background- фоновый рисунок в таблице; bgcolor- цвет фона таблицы; border - толщина рамки в пикселах; bordercolor - цвет рамки; cellpadding - отступ от рамки до содержимого ячейки; cellspacing - расстояние между ячейками; cols -Число колонок в таблице; height - Высота таблицы; width - Ширина таблицы.

Таблицы с невидимой границей долгое время использовались для табличной верстки web-страниц, позволяя разделять файл на модульные блоки. Ему на смену не пришел более современный способ верстки с помощью слоев и блоков.

Пример . Использование тега <TABLE>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/HTML4/loose.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
  <title>Ter TABLE</title>
 </head>
 <body>
  <table width="100%" border="0" cellspacing="0" cellpadding="4">
   <tr align="center" bgcolor="#999999">
    <td colspan="3" style="font-size: 160%; font-family: sans-serif">Lorem
     ipsum dolor sit amet</td>
   </tr>
   <tr>
    <td><img src="/images/logo.gif" alt="" width="150" height="70"
     border="0" style="background: #CC9"></td>
    <td align="right"><img src="/images/title.gif" alt="" width="70"
     height="70" style="background: #CF9"></td>
    <td>
     <a href="One.html">One</a><Br>
     <a href="Two.html">Two</a><Br>
     <a href="Three.html">Three</a>
    </td>
   </tr>
   <tr>
    <td colspan="3">
     <table  width="100%" border="0" cellpadding="0"
      cellspacing="0" bgcolor="#333333">
      <tr><td> </td></tr>
     </table>
    </td>
   </tr>
  </table>
 </body>
</html>
```

Нужно помнить, что результат применения атрибутов align или bgcolor в разных тегах может отличаться. Так, например, атрибут align в теге <table> означает выравнивание таблицы относительно окна броузера, а в тегах <tr> и <td> — выравнивание текста относительно ячейки.

Внутри <TABLE> допустимо также использовать элементы: <CAPTION>, <COL>, <COLGROUP>, <TBODY>, <TFOOT>, <THEAD>.

Для определения заголовка и "шапки" таблицы используются дескрипторы <caption> и <th>. Первый из них помещается внутри конструкции <table>... </table> и содержит заголовок всей таблицы. Второй используется вместо дескриптора <td> для ячеек, которые нужно выделить.

```
<!--пример: простая таблица-->
<html>
```

```
<head><title>использование таблиц</title></head>
<body>
<table border=”1”>
<caption align=”top”>Лучшие нападающие года</caption>
        <tr>
                <th>Имя</th>
                <th>Команда</th>
                <th>Очки</th>
        </tr>
        <tr>
                <td>small</td>
                <td> барселона</td>
                <td>5</td>
        </tr>
        <tr>
                <td>superman</td>
                <td>dinamo</td>
                <td>10</td>
        </tr>
        <tr>
                <td>bigman</td>
                <td>батэ</td>
                <td>7</td>
        </tr>
</table>
</body></html>
```

Для задания отступов текста от границ ячеек и расстояния между ячейками используются два атрибута тега <table> — cellpadding и cellspacing. Отступы задаются в пикселях. Для частичного отображения рамок используются атрибуты frame и rules. Для слияния горизонтальных и вертикальных смежных ячеек применяются параметры тега <td> — colspan и rowspan.

```
<!--пример: изменение цвета-->
<html>
<head><title>Таблицы </title></head>
<body bgcolor="white">
<font size=”6”>Примеры таблиц</font>
<br>
<hr color="blue">
<br>
<table border=”4” cellspacing=”3”>
<caption><b>Стандартная таблица</b> </caption>
<tr>
<th bgcolor="yellow">Заголовок 1</th>
        <th bgcolor="yellow">Заголовок 2</th>
</tr>
<tr>
<td>ячейка 1</td>
        <td>ячейка 2</td>
```

```
</tr>
<tr>
<td>ячейка 3</td>
 <td>ячейка 4</td>
</table>
<br>
<hr color="blue">
<br>
<table border="4" cellspacing=”3” background="fon01.gif">
<caption>Таблица с объединенными ячейками и фоновым рисунком</caption> <tr>
<td height="35" bgcolor="#FFCC33" colspan="2"> <center>1x1</center> </td>
<td width="50" bgcolor="#336699"> <center>1x2</center> </td> </tr> <tr> <td
height="35" width="50" bgcolor="#336699"> <center>2x1</center> </td> <td
width="50" bgcolor="#FFCC33"> <center>2x2</center> </td> <td width="50"
bgcolor="#336699"> <center>2x3</center> </td> </tr> </table>
<br>
<hr color="blue">
<br>
<table> <tr> <td height="35" bgcolor="#FFCC33"> <center>1x1</center> </td>
<td width="50" bgcolor="#336699"> <center>1x2</center> </td> <td
width="50" bgcolor="#336699" rowspan="2"> <center>1x3</center> </td> </tr> <tr> <td
height="35" width="50" bgcolor="#336699"> <center>2x1</center> </td> <td
width="50" bgcolor="#FFCC33"> <center>2x2</center> </td> </tr> </table>
<body></html>
```

Фреймы и <iframe>

Фреймы — средство HTML, позволяющее разделить пространство окна браузера на самостоятельные информационные зоны. Фреймовая система является устаревшей и в HTML 5 запрещена. Нескокько предложений приведены здесь в историческом плане.

Фреймовая система описывается в HTML-коде с помощью дескриптора <frameset>. В нем помещаются дескрипторы <frame>, описывающие параметры отдельных фреймов. HTML-коды содержимого этих фреймов находятся в отдельных файлах, имена которых указаны в дескрипторах <frame> с помощью параметров src. Деление окна броузера на части осуществляется с помощью параметров дескриптора <frameset>. Для деления по вертикали используется параметр cols, а по горизонтали — параметр rows.

Для создания внутри окна области с собственной рамкой и самостоятельной прокруткой используется дескриптор <iframe>. Он имеет те же параметры, что и <frame>, но не нуждается в специальной фреймовой структуре, описываемой дескриптором <frameset>, и может располагаться внутри HTML-кода. Дескриптор <iframe> разрешен в HTML5

Формы HTML

Форма — эффективное средство, благодаря которому HTML-страница позволяет принять информацию от пользователя и передать ее для обработки серверу. Способ обработки и передачи данных определяется тегами <form> и </form>, внутри которых и заключается код формы. Средство обработки полей формы определяется параметром action, метод передачи — параметром method, а тип кодирования — параметром enctype. Внутреннее содержание формы можно разделить на две части: активное и пассивное. Пассивными элементами формы являются все комментирующие и декоративные элементы.

Активные элементы формы предназначены для ввода данных. Это строки ввода, поля ввода, списки и кнопки. У каждого активного элемента формы есть параметры — name и value. Первый определяет имя поля, по которому его можно отличить от других элементов формы, второй — значение, которое передается через этот элемент. Большинство активных элементов формы описывается дескриптором <input>, а их вид определяется значением параметра type. Так, значение type=text соответствует строке ввода, type=file — строке выбора файла, password — строке ввода пароля; а значения checkbox и radio определяют список вариантов и список-переключатель, соответственно.

Значения submit, reset и button определяют кнопки различных видов.

Еще один дескриптор <textarea> позволяет создавать поля ввода массивов строк — прямоугольные окна с собственными средствами прокрутки, в которые можно вводить произвольный текст (без форматирования). Дескриптор <textarea> — парный. Внутри него помещается текст, который отображается в поле ввода.

Наконец, еще один распространенный элемент электронных форм — раскрывающийся список — создается с помощью конструкции HTML, образуемой тегами <select> и <option>. Первый заключает в себе весь список, вторые предназначены для создания отдельных пунктов. Списки, созданные таким образом, могут состоять из любого количества строк (если список состоит из одной строки, то он является "раскрывающимся"), а также, в зависимости от параметра multiple, позволяют выбрать один или несколько элементов.

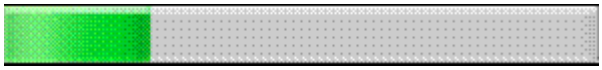
Данные, вводимые посредством формы могут передаваться по электронной почте или непосредственно программе-обработчику. Для обработки таких данных могут использоваться сценарии на языке JavaScript или PHP.

```
<!--пример: простая форма и элементы checkbox и radio-->
<html>
<head><title>Простая форма, checkbox и radio </title></head>
<body>
<form>
<h2>Простая форма</h2>
my street:<input name="street"><br>
city: <input name="city" size="20" maxlength="20" value="minsk"> <br>
zip: <input name="zip" size="5" maxlength="5" value="99999"><br>
</form>
<hr>
<h2>Ваша любимая команда</h2>
<form><!--выбор одной или нескольких команд -->
<input type="checkbox" name="team" value="шахтеры">шахтеры<br>
<input type="checkbox" name="team" value="ковбои">ковбои<br>
 <input type="checkbox" name="team" value="викинги">викинги<br>
</form>
<hr>
<h2>Какая из команд самая любимая?</h2>
<form><!--выбор только одной из нескольких команд -->
<input type="radio" name="team" value="шахтеры">шахтеры <br>
<input type="radio" name="team" value=" ковбои"> ковбои <br>
<input type="radio" name="team" value="викинги">викинги <br>
</form>
<hr>
<h2>Какая из команд наиболее любимая?</h2>
```

```
<form>
<select name=" team ">
<option> шахтеры</option>
<option> ковбои</option>
<option selected> викинги</option>
</select>
</form>
</body>
</html>
```

Новые элементы форм HTML5

HTML5 добавляет 5 новых элементов форм.
Форма progress показывает процесс загрузки. Например
<progress max="100" value="25">25%</progress>



Форма meter показывает шкалу измерения. Например:
<meter value="50" min="0" low="20" optimum="50" high="80" max="100"></meter>

Форма keygen используется для создания пары ключей, один из которых передается через форму на сервер.
<keygen name="key" keytype="rsa" challenge="challenge" />

Форма datalist используется для ввода элемента с возможными значениями в datalist описанными с помощью элементов option. Элемент input связывается с элементом datalist с помощью атрибута list.

```
<input list="company" />
<datalist id="company">
  <option value="BMW">
  <option value="Ford">
  <option value="Volvo">   </datalist>
```

Элемент Output определяет область в которую выводится информация с помощью скриптов. Рассмотрим пример

```
<DOCTYPE html>
<!--пример:  форма -->
<html>
<head><title>Новые формы</title></head>
<body>
<hr>
PROGRESS
<form>
<progress max="100" value="75">75%</progress>
</form>
 <hr>
METER
<form>
<meter value="50" min="0" low="20" optimum="50" high="80" max="200"></meter>
 </form>
<hr>
<form>
```

KEYGEN
<keygen name="key" keytype="rsa" challenge="goodby" />
</form>
<hr>
DATALIST
Элемент ввода input с возможными значениями в datalist с помощью атрибута list.
<form>
<input list="company" />
<datalist id="company">
<option value="BMW">
<option value="Ford">
<option value="Volvo"> </datalist>
</form>
<hr>
OUTPUT
Определяет область в которую выводится информация.
<form action="" oninput="out.value=range.value">

<input type="range" name="range" min="0" max="100" value="25" />

<output name="out">20</output>

</form>
</body>
</html>

PROGRESS

METER

KEYGEN 1536

DATALIST Элемент ввода input с возможными значениями в datalist с помощью атрибута list.

OUTPUT Определяет область в которую выводится информация.

20

Новые типы ввода элемента <input>

Элемент ввода имеет вид:
<input type="Value">
Рассмотрим пример:
<form action="/my.php" method="get">
Имя: <input type="text" name="name" />
Емейл: <input type="email" name="email" />
<input type="submit" value="Go" />
</form>

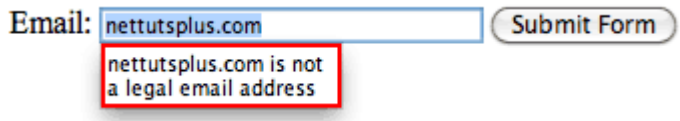
Значением атрибута type тега <input> в HTML4 может быть: type= text - определяет поле ввода в одну строку; type=file - определяет поле ввода и кнопку "Browse...", для загрузки файлов; type=hidden - скрытое поле ввода; type=image - изображение как кнопка отправки; type=button- интерактивные кнопки для активирования скрипта JavaScript; type=password - поле пароля; type=radio - определяет кнопку радио; type=checkbox - флажок; type=reset - определяет кнопку сброса данных; type=submit -определяет кнопку "Отправить" для отправки данных формы на сервер

HTML5 предоставляет 13 новых типов ввода, например
type=email - текстовое поле для адресов электронной почты;
type=number - числовое поле со счетчиком управления; type=range - управление числом с ползунком; type=search - текстовое поле для поиска; type=tel - текстовое поле для телефонных номеров; type=url -текстовое поле для URL-адресов; type=color - выбор цвета; type=date - поле даты (с управлением календарем) с указанием года, месяца и даты (без часового пояса); type=datetime -поле даты (с управлением календарем и временем); type=datetime-local - поле даты (с управлением календарем и временем (местное время); type=month - поле даты (с управлением календарем) с указанием месяца; type=week

Определяет поле даты (с управлением календарем) с указанием недели; type=time

Определяет поле даты (с управлением календарем) с указанием часа, минуты, секунды, и доли секунды (без часового пояса)

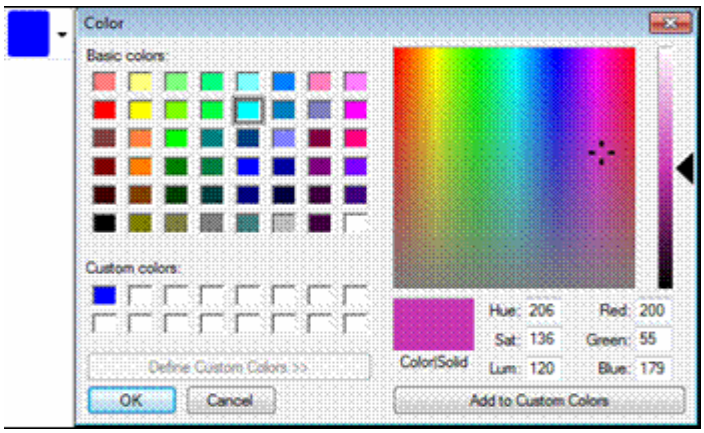
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>untitled</title>
</head>
<body>
<form action="" method="get">
<label for="email">Email:</label>
<input id="email" name="email" type="email" />
<button type="submit"> Submit Form </button>
</form>
</body>
</html>
```



<input type="**number**" min max step />
новый тип input – range служит для ввода значения из диапазона
<input type="**range**" min max step />
Пример создания бегунков с range input.

```
<form method="post">
<h1> Total Recall Awesomness Gauge </h1>
<input type="range" name="range" min="0" max="10" step="1" value="">
<output name="result"> </output>
</form>
```

Если step установлен как 1, будет 10 значений для выбора. Мы также используем в своих интересах новый элемент output



Новые атрибуты ввода HTML5

Атрибут **AUTOFOCUS** -при загрузке страницы автоматически передает фокус элементу. Например:

<input type="text" autofocus />

Атрибут FORM связывает элемент с формой.

<form action="" id="form">
<input type="text" name="login" />
<button type="submit"></button>
</form>
<input type="email" name="email" form="form" />

Атрибут PLACEHOLDER - устанавливает подсказывающий текст в поле ввода. Например:

<input type="**password**" placeholder="**enter your password**"/>

Состояние *off* можно использовать когда вводимые данные строго конфиденциальны, либо когда эти данные никогда не будут использоваться повторно.

<input type="**text**" autocomplete="**off**"/>

Атрибуты **formaction**, **formenctype**, **formmethod**, **formtarget** добавляются к элементам, отвечающим за отправку форм, и переопределяют соответствующие им атрибуты форм Эти атрибуты поддерживаются элементами ввода и кнопками

<input type="**submit**" formmethod ="**post**" />

formaction атрибут для переопределения действия элемента формы.

formenctype атрибут Для переопределения enctype элемента формы.

formmethod - атрибут для переопределения метода элемента формы.

formnovalidate атрибут для переопределения NOVALIDATE элемента формы. formtarget - главный целевой атрибут.

Атрибут required устанавливает , что поле не пройдет валидацию, если его значение пусто, либо не выбрано для *checkbox, radio*. Например

<input type="text" name="Bob" placeholder="Enter your name" required>

Атрибут pattern позволяет создавать регулярное выражение, которому должны соответствовать вводимые данные.

Атрибут SET CUSTOM VALIDITY Позволяет задать собственно сообщение валидации.

<form action="">
<input type="**email**" name="**email**" required/>
<input type="**password**" name="**pass1**" required/>
<input type="**password**" name="**pass2**" required/>
<input type="**submit**" value="**Register**"/>
</form>

Атрибут **autocomplete** уточняет, что поле должно /не должно автозаполнятся или быть предварительно заполнена браузером, на основании прошлых записей пользователя. Это может быть, например, номер кредитной карты или одноразовый пароль. По умолчанию автозаполнение включено, если вы хотите отключить, установите его в положение OFF.

Валидация формы

Традиционно валидация формы использует код JavaScript , чтобы сделать проверку входных данных или обеспечить заполнение необходимых полей перед отправкой формы. В HTML введен атрибут `required.` Если атрибут `required` присутствует, то поле непустое при отправке формы. Следующий пример ввода гарантирует, что поле имеет значение и что это значение есть адрес электронной почты.

<input type="email" id="email_addr" name="email_addr" required />

Для валидации можно использовать атрибут pattern, который содержит регулярное выражение для проверки поля ввода. Для примера, предположим, что номер состоит из трех прописных букв, за которыми следуют четыре цифры.

<input type="text" id="part" name="part" required pattern="[A-Z]{3}[0-9]{4}" />

Атрибут **Formnovalidate** может применяться для `input` or `button` элементов. Если оно есть, то проверка данных формы отключено. Вот пример:

<input type="submit" formnovalidate value="Save">

Вот пример, который отображает пользовательские ошибки, если значения в двух полях не совпадают.

```
<label>Email:</label>
<input type="email" id="email_addr" name="email_addr">
<label>Repeat Email Address:</label>
<input type="email" id="email_addr_repeat" name="email_addr_repeat" oninput="check(this)">
<script>
function check(input) {
  if (input.value != document. getElementById('email_addr').value) {
    input.setCustomValidity('The two email addresses must match.');
  } else {
    // input is valid -- reset the error message
    input.setCustomValidity('');
  }
}
</script>
```

Атрибут Placeholders позволяет создать указатели заполнения полей (подсказки, отображаемые внутри текстовых полей)

<input name="email" type="email" placeholder ="doug@givethesepeopleair.com" />

Атрибут required определяет, обязательно ли заполнение данного поля и назначается двумя способами:

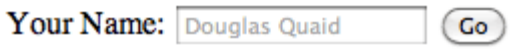
<input type="text" name="someInput" required>

<input type="text" name="someInput" required="required">

Форма не может быть подтверждена, если поле "someInput" пусто.

```
<form method="post" action="">
<label for="someInput"> Your Name: </label>
<input type="text" id="someInput" name="someInput" placeholder="Douglas Quaid" required>
<button type="submit">Go</button>
```

</form>



Если поле оставят пустым и будет нажата кнопка подтверждения формы, текстовое поле будет подсвечено.

Атрибут Autofocus используется, если определённое поле должно быть "выбрано" или сфокусировано, по умолчанию

<input type="text" name="someInput" placeholder="gogo" required autofocus>

Атрибут pattern, позволяет вставлять регулярные выражения для проверки определённого текстового поля прямо в разметку.

<form action="" method="post">
<label for="username">Create a Username: </label>
<input type="text" name="username" id="username" placeholder="4 <> 10"
pattern="[A-Za-z]{4,10}" autofocus required>
<button type="submit">Go </button>
</form>

Шаблон [A-Za-z]{4,10} вводит только заглавные и строчные буквы. Каждая строка должна содержать не менее 4 и не более 10 символов.

Для определения поддержки атрибутов создадим элемент input и определим, распознан ли браузером атрибут pattern. Если так, браузер поддерживает атрибуты HTML5, иначе – нет.

<script>
if (!'pattern' in document.createElement('input')) {
// …
}
</script>

Теги div и span

 Блок Важный тег используемый для форматирования больших блоков текста HTML документа. Устаревший атрибут align отвечает за выравнивание содержимого внутри блока: left - по левому краю; right - по правому краю; center - по центру; justify - выравнивается по содержимому. Атрибут Title позволяет создавать всплывающий текст над надписью. Style стандартный набор атрибутов стилей. Пример

По левому краю</DIV>

< div ALIGN="JUSTIFY">По содержимому краю</DIV>

Тег div является одним из основных элементов блочной верстки. В сочетании с таблицами стилей используется для разметки страницы, разбивки страницы на независимые секции.

Примеры:

 Блок красного цвета

Выравниваем текст по центру

Чтобы не описывать стиль внутри тега, можно выделить стиль во внешнюю таблицу стилей. Затем создается класс с именем селектора, а для тега добавляется параметр class или id.

текст

Стандартные атрибуты тега div:

 Определяет имя класса для элемента
<div dir= rtl , lt> Определяет направление текста в элементе
 Определяет уникальный id для элемента ;

Определяет код языка в элементе ;
Определяет инлайновый стиль элемента
 Определяет дополнительную информацию

Есть ли необходимость использовать div если есть доступ к header, article, section, и footer? Смысл использовать есть , если нужно обернуть блок кода в элемент, чтобы позиционировать контент. Однако, лучше использовать элементы <article> и <nav> соответственно.

Тег предназначен для определения встроенных элементов документа. В отличие от блочных элементов, таких как <table>, <p> или , с помощью тега можно выделить часть информации внутри других тегов и установить для нее свой стиль. Например, внутри абзаца <p> можно изменить цвет и размер первой буквы, если добавить начальный и конечный тег и определить для него стиль текста. Чтобы не описывать каждый раз стиль внутри тега, можно выделить стиль во внешнюю таблицу стилей, а для тега добавить атрибут class или id с именем селектора.

Синтаксис ...

Метатеги

Информация, расположенная в заголовочной части Web-страницы, предназначена главным образом для поисковых систем. Ее назначение —сжатое описание страницы, которое позволило бы правильно позиционировать страницу в каталоге поискового сервера и в списке ссылок, выданных пользователю в ответ на запрос. Среди главных элементов заголовочной части следует отметить название страницы, помещаемое между дескрипторами <title> и </title>, а также аннотацию и список ключевых слов, вводимые с помощью мета-записей keywords и description.

Любой метатег размещается в заголовке HTML-документа между тегами <head> и </head> и состоит из следующих атрибутов:

<META HTTP-EQUIV="имя" CONTENT="содержимое">

<META NAME="имя" CONTENT="содержимое">

Метатеги с атрибутом HTTP-EQUIV управляют действиями браузеров. В качестве параметра “имя” могут быть использованы следующие аргументы:

Expires – дата устаревания: если указанная дата прошла, то запрос этого документа вызывает повторный сетевой запрос, а не подгрузку документа из кэша. Дата со значением "0" заставляет браузер каждый раз при запросе проверять – изменялся ли этот документ. Например:

<meta http-equiv="expires" content="sun, 3 april 2013 9 05:45:15 gmt">

Pragma – контроль кэширования. Значением должно быть “no-cache”.

Content-Type – указание типа документа. Может быть расширено указанием браузеру кодировки страницы (charset). Например:

<meta http-equiv="content-type" content="text/html"; charset="utf-8"/>

Content-language – указание языка документа. Комбинация поля Accept-Language (посылаемого браузером) с содержимым Content-language может быть условием выбора сервером того или иного языка.

<meta http-equiv="content-language" content="en-gb">

Refresh – определение задержки в секундах, после которой браузер автоматически обновляет документ. Дополнительная возможность – автоматическая загрузка другого документа.

<meta http-equiv="refresh" content="3” url=http://www.bsu.iba.by ">

Window-target – определяет окно текущей страницы; может быть использован для прекращения появления новых окон браузера при применении фреймовых структур.

<meta http-equiv="window-target" content="_top">

Ext-cache – определяет имя альтернативного кэша

<meta http-equiv="ext-cache" content= "name=/some/path/index.db; istructions=user nstructions">

Управление индексацией страницы для поисковых роботов осуществляется с использованием атрибута name.

<meta name="robots" content="noindex">

Возможные значения: all, none, index, noindex, follow, nofollow.

Description – краткая аннотация содержания документа. Используется поисковыми системами для описания документа.

<meta name="description" content= "документ содержит словарь метатегов">

Keywords – используется поисковыми системами для индексирования документа. Обычно здесь указываются синонимы к словам в заголовке title или альтернативный заголовок.

<meta name="keywords" content="теги, метаданные, список">

Document-state – управление индексацией страницы для поисковых роботов. Определяет частоту индексации – или один раз индексировать, или реиндексировать документ регулярно.

<meta name="document-state" content="static">

Возможные значения: static, dynamic.

URL – управление индексацией страницы для поисковых роботов. Определяет частоту индексации – или один раз индексировать, или реиндексировать документ регулярно.

<meta name="url" content="absolute_url">

Author – обычно имя автора, формат произвольный.

Generator – обычно название и версия редактора, с помощью которого создана эта страница.

Copyright – обычно описание авторских прав на документ.

Resource-type – текущее состояние данного файла. Важен для поисковых систем: если его значение document, то поисковая система приступает к индексированию.

Новые возможности HTML5

В HTML5 добавлены новые элементы:

- canvas для рисования
- video и audio для мультимедия
- storage-для хранения данных
- article, footer, header, nav, section-новые элементы для контента
- calendar, date, time, email, url, search-новые формы

Первая задача HTML 5 - правильно интегрировать мультимедийный контент. В HTML 4.01 для отображения мультимедия используется Adobe Flash Player, в HTML 5 предполагается использоваться теги video и audio без использования дополнительных плагинов.

<video src="video.mp4" controls></video>

<audio src="The Imperial March.mp3" controls></audio>

Поддержка аудио

```
<audio autoplay="autoplay" controls="controls">
<source src="file.ogg" />
<source src="file.mp3" />
<a href="file.mp3">Download this file.</a>
</audio>
```

Поддержка видео

```
<source src="cohagenPhoneCall.ogg" type="video/ogg; codecs='vorbis, theora'" />
```

```
    <source     src="cohagenPhoneCall.mp4"     type="video/mp4;
'codecs='avc1.42E01E, mp4a.40.2'" />
    <p>       Your       browser       is       old.       <a
href="cohagenPhoneCall.mp4">Download   this   video   instead.</a>
</p>
    </video>
```

Не все браузеры понимают HTML5 видео. Ниже исходного элемента мы можем дать ссылку на загрузку файла или внедрить Flash-версию видео.

Видео может быть предварительно загружено при помощи preload="preload", или просто добавлением preload.

```
<video preload>
```

Атрибут Controls

Чтобы отобразить элементы управления, мы должны определить атрибут controls внутри элемента video.

```
<video preload controls>
```

Тэг <canvas> обеспечивает динамическую перерисовку изображения в зависимости от действий пользователя или изменяемых данных.Он будет описывать размеченную на веб-сайте область, а движок браузера будет отображать в реальном времени графический контент - чертежи, графики, небольшие игры и даже 3D. Для этого разрабатывается стандарт WebGL. Для того, чтобы скрипты "Canvas" не тормозили браузер, предусматривается поддержка многопоточности. Эта опция носит название "Web Workes", она выполняет скрипты и веб-приложения параллельно.

Меняется способ хранения информации на клиенте. Сейчас ее можно сохранить только в маленькие файлы cookies. А по новой технологии WebStorage на стороне клиента будут храниться до 10 Мбайт данных в специальной базе данных. С её помощью можно даже хранить специальные веб-приложения и работать с ними без подключения к интернету.

HTML 5 обеспечивает безопасность компонентов. Самая большая угроза в сети исходит из тегов iFrame (в этой области отображается содержимое стороннего сайта). Если в этой области содержится вирус, то он может проникнуть на компьютер. В новом стандарте в теги iFrame добавлен фильтр Sandbox, который будет ограничивать действие скриптов с внешних сайтов.

Ещё одна новинка - технология Web Forms 2.0. Она более эффективно выполняет обработку введенных пользователем данных, что также обеспечивает более высокую скорость. Новые формы: like calendar, date, time, email, url, search

Таблица тегов HTML 5

Теги	Описание
`<!-- -->`	Определяют комментарии
`<!DOCTYPE>`	Определяет тип документа
`<a> </a>`	Определяют гиперссылку
`<abbr> </abbr>`	Определяют аббревиатуру
`<acronym> </acronym>`	Не поддерживаются в HTML 5
`<address> </address>`	Отображают текст в формате адреса
`<applet> </applet>`	Не поддерживаются в HTML 5
`<area />`	Определяет активную область навигационной карты
`<article> </article>`	Новые теги – определяют внешний

	контент
<aside> </aside>	Новые теги – дополнительный контент
<audio> </audio>	Новые теги – определяют фоновый звук
	Отображают часть текста полужирным шрифтом
<basefont>	Не поддерживается в HTML 5
<bdo> </bdo>	Определяют направление текста
<big> </big>	Не поддерживаются в HTML 5
<blockquote></blockquote>	Определяют блочную цитату
<body> </body>	Определяют тело документа
 	Осуществляет перенос строки
<button> </button>	Создают кнопку
<caption> </caption>	Определяют надпись над таблицей
	Не поддерживаются в HTML 5
<cite> </cite>	Преобразуют текст в курсивный
<code> </code>	Преобразуют текст в моноширинный
<col />	Определяет свойства колонок таблицы
<colgroup> </colgroup>	Группируют колонки таблицы
<command> </command>	Новые теги – добавляют команду к кнопке
<datalist> </datalist>	Новые HTML теги – определяют допустимые значения
<dd> </dd>	Определение списка определений
<del> </del>	Отображают перечеркнутый текст
<details> </details>	Новые теги – определяют детали документа
<dialog> </dialog>	Новые теги – определяют диалог
<dfn> </dfn>	Преобразуют шрифт в наклонный
<dir> </dir>	Не поддерживаются в HTML 5
	Определяют секцию документа
<dl> </dl>	Создают список определений
<dt> </dt>	Определяют определяемый термин
	Преобразуют текст в курсивный
<embed />	Новый тег – внедряет интерактивный объект
<fieldset> </fieldset>	Объединяют элементы формы
<figure> </figure>	Новые HTML теги – группируют элементы страницы
 	Не поддерживаются в HTML 5
<footer> </footer>	Новые теги – нижняя часть документа

<form> </form>	Определяют HTML форму
<frame />	Не поддерживается в HTML 5
<frameset> </frameset>	Не поддерживаются в HTML 5
–	Определяют заголовки
<head> </head>	Содержат информацию о документе, инструкции
<header> </header>	Новые теги – верхняя секция документа
<hgroup> </hgroup>	Новые теги – определяют группу заголовков
<hr />	Создает горизонтальную линию
<html> </html>	Определяют HTML документ
	Преобразуют текст в курсивный
<iframe> </iframe>	Создают документ внутри документа
<img />	Определяет изображение
<input />	Создает поля для ввода данных, кнопки
<ins> </ins>	Преобразуют текст в подчеркнутый
<kbd> </kbd>	Преобразуют текст в моноширинный
<label> </label>	Определяют лейбу для тега <input />
<legend> </legend>	Заголовок для тегов <fieldset> </fieldset>
	Определяют элемент (пункт) списка
<link />	Определяет ссылку на внешний источник
<map> </map>	Определяют навигационную карту
<mark> </mark>	Новые теги – определяют важную часть текста
<menu> </menu>	Определяют список-меню
<meta />	Содержит информацию о документе
<nav> </nav>	Новые теги – определяют группу ссылок
<noframes> </noframes>	Не поддерживаются в HTML 5
<noscript> </noscript>	Предупредят если браузер не читает скрипты
<object />	Внедряет объекты в web-страницу
	Определяют упорядоченный (нумерованный) список
<optgroup> </optgroup>	Определяют группу элементов <option> </option>
<option> </option>	Определяют элемент выпадающего списка
	Определяют параграф
<param />	Определяет проигрыватель
<pre> </pre>	Определяют отформатированный текст

<q> </q>	Определяют короткую цитату
<s> </s>	Не поддерживаются в HTML 5
<samp> </samp>	Преобразуют текст в моноширинный
<script> </script>	Определяют скрипт
<section> </section>	Новые HTML теги – определяют секцию документа
<select> </select>	Определяют выпадающий список
	Преобразуют текст в более мелкий
	Определяют линейную секцию в документе
<strike> </strike>	Не поддерживаются в HTML 5
	Преобразуют шрифт в полужирный
<style> </style>	Определяют стилевые описания
	Преобразуют обычный текст в текст в нижнем индексе
	Преобразуют текст к верхнему индексу
<table> </table>	Определяют таблицу
<tbody> </tbody>	Определяют тело таблицы
<td> </td>	Определяют ячейку таблицы
<textarea> </textarea>	Определяют текстовое поле
<tfoot> </tfoot>	Определяют нижнюю часть таблицы
<th> </th>	Определяют заголовок таблицы
<thead> </thead>	Определяют верхнюю часть таблицы
<time> </time>	Новые теги – определяют дату/время
<title> </title>	Определяют основной заголовок документа
<tr> </tr>	Определяют табличный ряд
<u> </u>	Не поддерживаются в HTML 5
	Определяют ненумерованный список
<var> </var>	Определяют переменную
<video> </video>	Новые теги – внедряют видео в страницу

В HTML5 удалены теги: basefont, big, center, font, s, strike, tt, u, frame, frameset, noframes, acronym, applet, isindex, dir.

Добавлены новые теги:

<!DOCTYPE html> section, article, aside, header, footer, nav, figure, audio, video, source, embed, meter, time, canvas, output, datagrid, details, datalist, datatemplate, progress.

Отметим, что по правилам XHTML все теги и атрибуты должны быть набраны в нижнем регистре, кроме тега <!DOCTYPE html >. По этим же правилам необходимо закрывать любые теги, в том числе одиночные, кроме <!DOCTYPE>. Вместо закрывающего тега может быть использована запись:

, <hr />, <img />, <input />, <link />, <meta />.

Валидация документов

Валидацией называется проверка документа на соответствие стандартам. Код веб-страницы должен подчиняться определенным правилам (www.w3c.org). Валидация широко применяется для проверки XML – документов, однако может быть использована и для XHTML - документов.

Способы проверки веб-страниц на наличие ошибок делятся на онлайновые и локальные. Онлайновые предназначены для проверки страниц с через Интернет, а локальные используются для проверки документов на текущем компьютере.

По адресу http://validator.w3.org располагается распространенный инструмент для проверки отдельных страниц на валидность. Этот сайт предлагает три способа проверки: по адресу, проверку локального файла, проверку введенного в форму кода. Если сайт уже опубликован в Интернете, то любую страницу можно проверить, вводя в текстовое поле ее адрес. Так, вводя http://htmlbook.ru в форме «Validate by URI» и нажав кнопку Check, получим сообщение о том, валидный документ или нет. Хотя в текстовом поле вводится адрес сайта, проверяется не сайт целиком, а только главная страница.

Документы, еще не выставленные в Интернете, можно проверить с помощью формы, озаглавленной «Validate by File Upload» (валидация загруженных файлов. Вначале следует указать путь к HTML-файлу, после чего нажать кнопку Check . Файл будет загружен на сервер и проверен на ошибки.

В некоторых случаях требуется проверить код без сохранения его в отдельный файл. В этом случае пригодится форма для прямого набора текста и отправки его на сервер для валидации.

Для браузера Firefox может использоваться расширение HTML Validator. Эта программа построена по той же технологии, что и валидатор W3C, но не требует подключения к Интернету и работает прямо «на лету». Скачать можно по адресу http://users.skynet.be/mgueury/mozilla/

После скачивания файла установить расширение можно несколькими способами.

1. Через менеджер расширений. Запустите Firefox и откройте меню Инструменты > Расширения . Перетащите мышью загруженный файл (он имеет расширение xpi) в открывшееся окно. Далее расширение будет установлено автоматически.

2. С помощью открытия файла. Выберите в меню Firefox пункт Файл > Открыть файл... и укажите путь к файлу с расширением.

3. Копирование файла в папку extension. Откройте папку на диске, где установлен Firefox (к примеру c:\Program Files\Mozilla Firefox) и найдите в ней подпапку extension, в которую скопируйте расширение. После запуска браузера дальнейшая установка пройдет самостоятельно.

Каскадные таблицы стилей CSS

CSS (Cascading Style Sheets - Каскадные таблицы стилей) – это технология описания внешнего вида документа, написанного языком разметки. Основная идея CSS состоит в том, чтобы отделить описание логической структуры документа (HTML) от его внешнего вида(CSS).

Таблицы стилей CSS позволяют уменьшить размер HTML кода, улучшить его читаемость, сократить объем работы разработчиков, менять внешний вид документа без изменения HTML кода. Одна таблица стилей может применяться к нескольким документам. CSS используется как средство оформления веб-страниц в формате HTML, XHTML и XML.

Использование CSS позволяет ускорить работу web-приложений за счет сокращения загружаемого кода и за счет кэширования файлов (однократная загрузка файла с последующим его сохранением.

Способы включения каскадных таблиц стилей

Существует несколько способов включения CSS в HTML-документ:

Встраивание стиля (inline styles) с помощью атрибута style в один из тегов HTML – документа. Например:

текст

Красный текст на желтом фоне, маленькими заглавными буквами.

Стили часто встраиваются в теги p, h, body, div, span. Способ встраивания стиля нарушает идеологию CSS, и не рекомендуется.

Внедрение таблицы стилей в заголовок HTML документа между тегами <head> и </head> с помощью тега style. При этом CSS-стили располагаются между открывающим и закрывающим тегами style и содержат определения для всего HTML-документа:

<head>

<style type="text/css">

<!—описание стиля-->

</style>

</head>

Приведенные два способа не являются наилучшими, так как размещение стилей непосредственно в разметке страницы, не позволяет браузеру кэшировать стили.

Связывание с внешней таблицей стилей. Если предполагается использовать один стиль для нескольких страниц документа или нескольких документов, следует описать стиль в отдельном файле с расширением .CSS, например style.css. Для подключения затем стилевого файла используется тег <link>, расположенный внутри тега <head>.

<head>

<link rel="stylesheet" type="text/css" href="style.css" />

</head>

Первые два свойства элемента link указывают браузеру, что на странице используется CSS. Значение последнего свойства - имя файла, содержащего стиль страницы.

Импортирование - добавление внешней таблицы стилей при помощи правила @import. В отличие от HTML-тега <link> правило @import является языковой конструкцией CSS и добавляется в элемент style:

<head>

<style type=”text/css”>

@import "style.css";

</style>

</head>

Свойство @import таблицы стилей следует задавать в начале стилевого блока или связываемой таблицы стилей перед заданием остальных правил. Значением свойства @import является URL файла, содержащего таблицы стилей. Импортирование позволяет встраивать в документ таблицу стилей, расположенную на сервере.

Таблицы стилей не чувствительны к регистру. Текстовые комментарии в таблицах стилей оформляются так же, как и в языке Си:

h1{ color: red } /* color is red */

Стили

Стиль – это правило, описывающее форматирование фрагмента документа. Каждое правило состоит из селектора и определения. Селектор определяет, на какую часть документа распространяется правило, а определение задает значения его свойств в фигурных скобках.

Selector{

Property1: value1 value2;

Property2: value3 value4;

Property3: value5 value6;
}

Селекторы правил CSS могут быть селекторами элементов (тегов HTML), селекторами классов, селекторами псевдоклассов, селекторами идентификаторов, селекторами потомков и др.

В фигурных скобках после селектора задаются значения свойств. Если у свойства несколько значений, то эти значения отделяются друг от друга пробелами. Описания свойств отделяются друг от друга точкой с запятой. Точка с запятой ставится всегда, даже если свойство одно.

Селектором может быть любой элемент HTML, например

```
html {color: black;}
p {color: red;}
h2 {font-size: 110 %;}
```

В таблицу стилей можно ставить любое количество пробелов и переносов строк, браузер оставит столько, сколько нужно. Значения свойств, являющихся адресами устанавливается так: url(адрес).

Ниже приводится пример кода CSS, встраиваемого в теги HTML.

```
<DOCTYPE html>
<html>
<head>
<title>pr1css-Пример css, встраиваемого в HTML страницу</title>
<style type="text/css">
<!-- body { color: gold; background: blue; font: bold 14px comic sans ms; text-align: justify; margin: 10px; }  -->
html {color: black; }
p {color: red;}
h1{font-family: arial; font-weight: bold; font-size: 14pt; color:green;}
h2{font-family: arial; font-size: 110 %; color:black;}
h3 {font-weight:bold; color: red;}
</style>
Текст из раздела Head
</head>
<body>
<h1>Примеры CSS - это заголовок H1</h1>
<p>
пример кода CSS, встраиваемого в HTML страницу. <br>
Селектором является тег body
</p>
<h2>Примеры CSS - это заголовок H2</h2>
  <h3>h3-Жирный заголовок красного цвета </h3>
```

Этот текст имеет вид, определенный таблицей стилей: золотые буквы на голубом фоне,
жирный шрифт Comic Sans MS размером 14 пикселей, выровненный по обоим краям с
 отступами со всех сторон 10 пикселей.

```
</body>
</html>
```

Здесь приведен устаревший способ включения стиля CSS в тег body. Для совместимости со старыми браузерами, не понимающими CSS, содержимое элемента <style> ранее заключали в HTML комментарий. Новые браузеры поймут, что в комментариях заключена таблица стилей и подключат ее.

Группирование. Если двум селекторам присваивается одинаковое определение, то их можно записывать через запятую:

```
h1, h2 {
font-family: arial
}
```

Если селектор имеет несколько определений:

```
h1{font-weight:bold} h1 {font-size: 14pt}
```

то они записываются через точку с запятой:

```
h1 {font-weight: bold; font-size: 14pt;}
```

Наследование. При определении стиля элемента, вложенные элементы наследуют свойства головного элемента. Например, если в параграфе <p> задается красный цвет шрифта, то выделенный с помощью <b> текст также будет красным:

```
<p style=”color: red;”>Внутри параграфа с красным цветом текста, <b>выделенный текст</b> наследует цвет головного тега.</p>
```

Наследование удобно использовать для описания свойств элементов по умолчанию. Для этого достаточно описать стиль тега <body>, порождающего остальные элементы документа.

При сложении стилей приоритет имеют атрибуты, определенные в более конкретном стиле. Стили в порядке убывания конкретности: <body>, остальные теги html, CSS – классы, встроенные стили.

CSS - классы

Чтобы иметь возможность отображать одни и те же элементы HTML в разных местах по-разному, необходимо использовать CSS-классы. Это позволяет для одного тега использовать несколько классов и, соответственно, стилей. Свойства класса присваиваются определенному тегу с помощью атрибута class: class="classname". Имя класса и его свойства перед этим должны быть определены в виде селектора класса: .classname{свойства}

Это можно продемонстрировать на следующем примере:

```
<html>
<head>
<title>Использование классов</title>
<style type="text/css">
html{color:green;}
.font1 {
color: yellow;
background: red;
}
 .font2 {
color: blue;
background: yellow;
}
</style>
</head>
<body>
<span class="font1">Желтый текст на красном фоне</span>
<span class="font2">Синий текст на желтом фоне</span>
<table class="font2" border="2">
<tr>
<td>Это ячейка1 таблицы</td>
<td>это ячейка2 таблицы</td>
</tr>
```

```
</table>
</body>
</html>
```

Два класса «font1» и «font2» задают цвет фона и изображения. Эти классы были применены к тегу и таблице.

Если, например, необходимо поменять стиль выделения текста с помощью класса во всем документе, то достаточно поменять значение цвета в определении этого класса. Если необходимо определить класс не для любого элемента HTML, а только для конкретного тега, то используется конструкция тегового класса:

имя_тега.имя_класса {свойства}

Теговый класс сработает только в том теге, для которого он предназначен, а для всех остальных тегов будет проигнорирован. Рассмотрим, например, класс select для тега :

```
<html>
<head>
<title>pr2css-Использование классов</title>
<style type="text/css">
.clasname{ font-size: 9pt; color:green }
span.select {color: red;}
</style>
</head>
<body>
<p class=clasname>Стиль работает</p>
<p class=select>Стиль не работает</p>
Здесь мы выделили <span class="select">часть текста красным.</span>
</body>
</html>
```

Значением class может быть множественный класс, состоящий из нескольких слов, разделенных пробелами. Например:

```
<p class=”urgent warning”>Опасность! Предупреждение!</p>
.warning.urgent {font-style: italic;}
```

Объединяя 2 селектора класса, можно выбрать только те элементы, которые имеют оба имени класса, стоящие в любом порядке.

В каких тегах лучше определять стили посредством класса? Чаще всего для этого используется одна из следующих конструкций:

```
<p class="big">Что-то</p>
<td class="big">Что-то</td>
<div class="big">Что-то</div>
<span class="big">Что-то</span>
```

ID-селекторы

ID-селекторы используются для определения уникальных частей веб-страницы типа <header>, <footer>. Идентификаторы используются, когда необходимо определить разделы страницы, которые встречаются один раз. Для стиля, используемого неоднократно, применяются классы.

При объявлении стиля перед ID-селектором ставится #:

```
#green {color: green;}
```

Затем HTML элементу, подлежащему форматированию, присваивается соответствующее значение атрибута id:

```
<footer id=”green”>Text</p>.
```

Символы “#” и ”.” используются только при описании стилей. При вставке имени идентификатора или класса в теги их указывать не нужно!

Теги div , span и link

Элементы и используются для структурирования документа совместно с атрибутами class и id. В сочетании с CSS, может использоваться для визуальных эффектов применимо к отдельным блокам текста. Пример

```
<p>Если ты умный, почему не богатый </p>
 <p> Если ты очень умный, то <span class="benefit">
здоровый</span>, и<span class="benefit">богатый</span> </p>
 span.benefit {
 color:red;
 }
```

Тег , в отличие от , делает до и после себя отбивку (как и <p>). В то время как используется в элементах уровня блока, применяется для группирования блок-элементов. Тег является универсальным контейнером для элементов разметки, удобным для форматирования и стилизации.

Тег link используется для установки связи между HTML документом и внешней таблицей стилей из CSS файла, например:

```
<link rel="stylesheet" href="/s.css" type="text/css" media="all" />
```

Свойства шрифтов (фонтов).

Свойство шрифтов font позволяет задать характеристики шрифта: font-family| font-style | font-variant | font-weight| font-size

Свойство	**Описание значения**
font-family	Указывается до трех шрифтов, через запятую: serif|san-serif |cursive |fantasy | monospace.
font-style	normal (по умолчанию), italic (курсив), oblique (наклонный).
font-variant	варианты начертания: normal (по умолчанию), small-caps (все буквы заглавные уменьшенного размера).
font-weight	жирность шрифта: normal , bold, bolder(жирный), lighter(бледный).
font-size	размер в абсолютных: (константы xx-small | x-small | small | medium | large | x-large | xx-large | {Абсолютный размер} или в относительных единицах или процентах: larger | smaller | {Отн. размер}%;

Пример:

```
<html>
<head>
<title>pr3css-Установка шрифта с помощью стилей</title>
<style type="text/css">
body {
font-family: “Times New Roman”, times, serif;/* шрифты*/
font-size: 100%; /* Размер шрифта для основного текста */
font-style: oblique;}
th {
font-family: arial, sans-serif; /* Семейства шрифтов*/
font-size: 90%; /* Размер шрифта для заголовка таблицы */
font-weight: bold /* Полужирная насыщенность*/
}
```

```
h1, h2, h3 {
font-family: verdana, tahoma, arial, sans-serif /*шрифты*/
}
#cursive { font-style: italic} /* Курсивный текст */
</style>
</head>
<body>
<div id=”cursive”>Курсивный текст</div>
<h1>Заголовок1</h1>
<h2>Заголовок2</h2>
<h3>Заголовок3</h3>
<table>
<th>Таблица</th>
</table>
</html>
```

Результат:

Курсивный текст

ЗАГОЛОВОК1

ЗАГОЛОВОК2

Заголовок3

Порядок свойств, установленных в font следующий: font-style|font-variant| font-weight|font-size|font-family. Используя сокращенную запись font, можно указывать все свойства шрифта в одном стилевом правиле:

```
p {font: italic normal bold 30px arial, sans-serif;}
```

В свойстве font-family обычно указывается несколько шрифтов, через запятые, чтобы выбрать из них тот шрифт, который установлен на компьютере клиента. Кроме названия может указываться гарнитура (семейство) шрифтов: serif, sans – serif, monospace.

Еще одно свойство @font-face - указывает список семейств или названий шрифтов, а также электронный адрес, по которому будут загружаться шрифты, если их нет на компьютере пользователя. Пример:

```
body {
@font-face: Myfont;
src: url(“http://www.atlant.ru/Myfont.eot”);
}
```

Стили текста.

Свойство	**Описание значения**
text-decoration	none (по умолчанию), underline (подчеркивание), line-through (зачеркивание), overline (надчеркивание).
text-transform	регистр: none, capitalize (Первая Буква Слова Преобразуется В Заглавную), uppercase (ВСЕ В ЗАГЛАВНЫЕ), lowercase (в строчные).
text-align	Горизонтальное выравнивание текста: left (по умолчанию), right (по правому краю), center центрирование, justify (по ширине колонки).
vertical-align	Вертикальное положение базисной линии элемента: baseline, middle, sub – элемент подстрочный, super – элемент надстрочный, text-top – выравнивает верх элемента по верху шрифта родительского элемента , text-bottom – выравнивает низ по низу шрифта родительского эле-

	мента.
text-indent	*{Отступ}|{Отступ}%;* Устанавливает величину отступа в первой строке параграфа абсолютной величиной или в процентах.
line-height	вертикальное расстояние между строками текста: ***normal|{Y}|{Y}%***; та Задано абсолютной величиной или процентом.
word-spacing	расстояние между словами: ***normal|{Величина}***; Значение задано либо абсолютной величиной, либо предопределенным значением ***normal***.
letter-spacing	расстояние между символами в тексте: ***normal|{Величина}***;
word-wrap	***normal|break-word;*** normal – запрещает переносить строки по словам, break-word – разрешает.
white-space	пробелы между словами: normal, nowrap – пробелы не учитываются; pre – текст с учетом пробелов и переносов; pre-line – пробелы и переносы не учитываются; pre-wrap – в тексте сохраняются все пробелы и переносы; inherit – наследует значение родителя.

Рассмотрим пример. Свойство text-indent - определяет величину отступа первой строки абзаца текста в единицах длины или процентах. По умолчанию отступ первой строки равен 0.

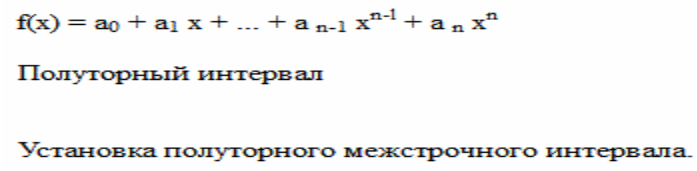
```
<html>
<head>
<style type="text/css">
p { text-indent: 3em ;}
div { text-align: center;}
</style>
</head>
<body>
<p>Создание отступа с помощью параметра text-indent.</p>
<div>Выравнивание текста с помощью свойства text-align.</div>
</body>
</html>
```

В следующем примере рассматривается свойство vertical-align для создания нижних и верхних индексов

```
<html><head>
<title> Использование стилей для управления индексами </title>
<style type="text/css">
.sup {
vertical-align: super; /* Сдвигаем текст вверх */
font-size: 70% ; /* Уменьшаем размер шрифта */
}
```

```
.sub {
vertical-align: sub; /* Сдвигаем текст вниз */
font-size: 70%;
}
<p style="line-height: 1.5">
</style>
</head>
<body>
<div>
f(x) = a<span class="sub">0</span> + a<span class="sub">1</span> x + ... + a
<span class="sub">n-1</span> x<span class="sup">n-l</span> + a
<span class="sub">n</span> x<span class="sup">n</span>
</div>
<p>Полуторный интервал</p><br>
Установка полуторного межстрочного интервала.
</body>
</html>
```

Результат:



Свойство text-decoration - определяет оформление текстового элемента в виде подчеркивания (underline), надчеркивания (overline) или перечеркивания текста (line_through). По умолчанию никакого оформления текста не производится (none). Пример:

```
<html>
<head>
<style type="text/css">
a {
text-decoration: none /* Отменяем подчеркивание у ссылки */
}
</style>
</head>
<body>
<a href=link.html>Ссылка без подчеркивания</a>
</body>
</html>
```

Пример. Свойство letter-spacing - определяет расстояние между текстовыми символами. Значениями свойства могут быть: normal – стандартный интервал для текущего шрифта; величина, на которую увеличивается межсимвольное расстояние в дополнение к стандартному.

Пример Свойство word-spacing - указывает расстояние между словами. Значениями свойства могут быть: normal – стандартный интервал для текущего шрифта; величина, на которую увеличивается расстояние между словами в дополнение к заданному по умолчанию.

```
<html>
```

```
<head>
<style type="text/css">
h1{ word-spacing: 1 em }
</style>
</head>
<body>
<p>Интервал, установленный по умолчанию</p>
<p styles ="letter-spacing: 0.3em">Очень большой интервал</p>
<p style = "letter-spacing: 0.2em"> Большой интервал </p>
<h1>Изменение пробелов с помощью свойства word-spacing</h1>
</body>
</html>
```

Свойство text-transform - управляет преобразованием регистра букв. По умолчанию регистр букв не преобразуется. Пример

```
<html>
<head>
<title> Использование свойства text-transform </title>
<style type="text/css">
.upper { text-transform: uppercase }
.capital { text-transform: capitalize }
</style>
</head>
<body>
<div class=”upper”> Все буквы преобразуются в прописные </div>
<div class=”capital”> Первая буква каждого слова преобразуется в прописную
</div>
</body> </html>
```

Цвет и фон.

Свойство	**Описание**
color	Определяет цвет элемента.
background	Свойства фона: *[{background-color}] [{background-image}] [{background-repeat}] [{background-attachment}] [{background-position}];*
background-color	фоновый цвет страницы или ее элемента: *{Цвет}*|*transparent;* значение *transparent* задает "прозрачный" фон.
background-image	фоновый рисунок страницы или ее элемента. *url({Интернет-адрес файла рисунка})*|*none*; значение *none* отключает фоновый рисунок.
background-attachment	*Fixed* -позволяет "зафиксировать" фоновый рисунок, чтобы он не прокручивался вместе с содержимым тега <BODY>.
background-repeat	Устанавливает порядок повторения фонового рисунка: *repeat*|*no-repeat*|*repeat-x*|*repeat-y*; repeat – размножает изображение во всех направлениях; repeat-x/ repeat-y – размножает изображение по горизонтали/вертикали;
background-position	Задает местонахождение фонового рисунка и заменяет атрибуты *[{background-position-x}] [{background-position-y}];*
background-position-x	Задает горизонтальную координату фонового рисунка. *{X}*|*{X}%*|*left*|*center*|*right*;
background-position-y	Задает вертикальную координату фонового рисунка. *{Y}*|*{Y}%*|*top*|*center*|*bottom*;

Пример color - указывает цвет текста элемента.

```
<html>
<head>
<title> Изменение цвета символов</title>
</head>
<body>
<p><span style="color: blue">Первое</span> слово в строке - синее.</p>
<p style="color: rgb(49, 151, 116)"><span style="color:#fe0">Желтое</span> слово в строке зеленого цвета.</p>
</body>
</html>
```

Пример background-color - определяет цвет фона элемента.

```
<html>
<head>
<title> Изменение цвета фона</title>
<style type="text/css">
. inverse {
font-family: Verdana; /* Шрифт Verdana */
font-weight: bold; /* Жирное начертание */
background-color: green; /* Зеленый фон */
color: white; /* Символы белого цвета */
}
</style>
```

```
</head>
<body>
<div class=”inverse”>Белые буквы на зеленом фоне</div>
</body>
</html>
```

background - одновременно устанавливает свойства background-color, background-image, background-repeat, background-attachment и background-position.

Пример:

```
<html>
<head>
<style type="text/css">
body {
background:
white /* Цвет фона */
url(image.gif) /* Путь к файлу с рисунком фона */
left top /* Положение в левом верхнем углу */
no-repeat /* Не повторять рисунок */
fixed /* Зафиксировать фон */
}
</style> .
</head>
<body>
```

Свойства списков

list-style-Задает параметры маркера или номера позиции списка: [{list-style-image}] [{list-style-position}] [{list-style-type}];

list-style-type задает тип маркера или номер позиции списка: disc|circle|square|decimal|lower|roman|upper-roman|lower-alpha|upper-alpha|none; disc – (по умолчанию) сплошной кружок; circle – окруж-ность; square – сплошной квадрат; decimal – нумерует арабскими циф-рами; lower-roman – малыми римскими; upper-roman – большими рим-скими; lower-alpha –малыми латинскими буквами; upper-alpha – большими латинскими; none – убирает маркер или нумерацию.

list-style-image-Задает графическое изображение, отображаемое в качестве маркера позиции списка.

: none|url({Интернет-адрес файла изображения});

list-style-position Задает местонахождение маркера позиции списка: в тексте позиции или вне: outside|inside; Значение outside (по умолчанию) задает отображение маркера позиции списка вне текста позиции. Значение inside заставляет Web-браузер отобразить маркер позиции в ее тексте в качестве первого символа.

list-style-type - определяет вид маркера элемента списка, если не задано изображение в качестве маркера, либо оно не доступно.

Пример:

```
ol {list-style-type: lower-roman}
```

list-style-image - определяет графическое изображение в качестве маркера элемента списка.

Пример:

```
<html>
<head>
<style type="text/css">
a.outer {
```

```
list-style-image: url(image.gif);/*Путь к файлу с рисунком */
}
</style>
</head>
<body>
<ol>
<li class=”outer”>С маркером</li>
<li>Без маркера</li>
</ol>
</body>
</html>
```

list-style-position - определяет положение маркера в списке. По умолчанию маркеры находятся вне пространства, отведенного под список

Пример:

```
<html>
<head>
<style type="text/css">
ul { list-style-position: outside }
</style>
</head>
<body>
<ul>
<1i>Изменение положения маркеров </li>
<1i>Маркеры размещаются за пределами текстового блока</li>
</ul>
</body>
</html>
```

list-style - указывает значения сразу трех свойств: list-style-type, list-style-image и list-style-position.

Свойства таблиц

Свойство	**Описание**
border-collapse	*separate*|*collapse*; separate - разделяет границу таблицы и границы ее ячеек (по умолчанию); collapse - объединяет их.
border-spacing	Задает расстояние между границами ячеек в таблице: *значение1 [значение2*]. Одно значение устанавливает одновременно расстояние по вертикали и горизонтали между границами ячеек. Если значений два, то первое определяет горизонтальное расстояние, а второе – вертикальное.
caption-side	Определяет положение заголовка таблицы, который задается с помощью тега <caption> относительно самой таблицы*: top* | *bottom.*top – располагает заголовок по верхнему краю таблицы. bottom – заголовок располагается под таблицей. right – заголовок размещается справа от таблицы. left – заголовок размещается слева от таблицы.
empty-cells	Задает отображение границ и фона в ячейке, если она пустая. При одновременном добавлении к таблице свойства border-collapse со значением collapse, свойство empty-cells игнорируется. empty-cells: *show* | *hide.* show – отображает границу вокруг ячейки и фон в ней.

	hide – граница и фон в пустых ячейках не отображается. Если все ячейки в строке пустые, то строка прячется целиком.
table-layout	Определяет, как браузер должен вычислять ширину ячеек таблицы: auto | *fixed* | *inherit:*auto – браузер загружает таблицу, анализирует ее для определения размеров ячеек и отображает.fixed – ширина колонок определяется либо с помощью тега <col>, либо вычисляется на основе первой строки. inherit – наследует значение родителя.

Псевдоклассы

Для управления цветом ссылок используются псевдоклассы. Действие псевдокласса распространяется не на весь текст, к которому применен данный стиль, а лишь на его часть и в определенном состоянии.

Общий синтаксис для ссылок: a: псевдокласс {параметр: значение }

Рассмотрим эффект, при котором ссылки подчеркиваются лишь при наведении на них курсора:

a { text-decoration: none; }

a:hover{ text-decoration: underline; }

Верхняя строчка - это переопределение стандартного тега <a>, которое запрещает подчеркивать ссылки, а нижняя - это определение стиля для псевдокласса hover, который описывает стиль ссылки в момент, когда курсор находится над ней.

Псевдокласс	**Описание**
:active	Применяется к активным гиперссылкам; Пример: *a:active {color: lime;}*
:first-child	Применяет стилевое оформление к первому дочернему элементу элемент:first-child { ... }
:focus	определяет стиль элемента, получающего фокус. Например, им может быть текстовое поле формы, элемент:focus { ... }
:hover	Применяется к гиперссылкам, над которыми пользователь помещает курсор мыши: Пример:*a:hover {color: lime; text-decoration: none;}*
:lang	Определяет язык, который используется в документе или его фрагменте. элемент:lang(<язык>) { ... } В качестве языка могут выступать следующие: ru – русский; en – английский ; de – немецкий;
:link	Применяется к не посещенным еще гиперссылкам: link {Определение стиля};Пример:*a:link {color: black;}*
:visited	Применяется к посещенным ссылкам:; Пример:*a:link {color: indigo;}*

Пример:

```
<html>
<head>
<title> Псевдоклассы для ссылок</title>
<style type="text/css">
a:link {color : lime}
a:visited {color: indigo }
a:hover {color: red }
a:active {color : #fe0 }
</style>
```

```
</head>
<body>
<a href=linkl.html>Ссылка 1</a>
<a href=link2.html>Ссылка 2</a>
<a href=link3.html> Ссылка 3</a>
</body>
</html>
```

Если необходимо в конце странички указать копирайт, чтобы копирайт также был ссылкой достаточно локально переопределить цвет ссылки:

```
<a href="#"><span style="color: #0;">Copyright (C) 2012-2015 MyDesign </span> </a>
```

Псевдостили текста

Псевдостили применяются к некоторым элементам текстовых абзацев, например, к первой строке абзаца или первой букве первой строки.

first-letter - применяется к первой букве первой строки абзаца:

```
p:first-letter { font-size: 200%; font-weight: bold; }
```

Рассмотрим пример:

```
<html>
<head>
<style type="text/css">
p:first-letter {
color: red;
font-size: 12px
}
</style>
</head>
<body>
<p>Использование псевдокласса first-lette.</p>
</body>
</html>
```

first-line - применяется к первой строке абзаца.

{Задание стиля абзаца}:first-line{Определение стиля};

Пример:

```
p:first-line {text-decoration: underline;}
```

Псевдоэлементы

Свойство	**Описание**
:after	Псевдоэлемент, который используется для вывода желаемого контента после элемента, к которому он добавляется. элемент:after { content: "текст" }
:before	Псевдоэлемент :before применяется для отображения желаемого контента до элемента, к которому он добавляется. Работает совместно со свойством *content*:before { content: "текст" }
:first-letter	Применяется к первой букве первой строки абзаца. Может использоваться для создания буквиц:first-letter {Определение стиля}; Пример: *st:first-letter {font-size: 16pt;}*
:first-line	Применяется к первой строке абзаца:first-line{Определение стиля}; Пример: *st:first-line {text-decoration: underline;}*

Форматирование псевдоклассов и псевдоэлементов

Псевдоклассы это - ссылки, first-child и lang:

a: link - свойства обычной ссылки; a: active - свойства активной ссылки; a: visited - свойства посещенной ссылки; a: hover - свойства ссылки при наведении на неё мыши; a: focus - свойство ссылки при фокусе. first-child - выделяет первый элемент среди последующих. lang - форматирование элементов в зависимости от применения языков на странице.

В каскадных листах стилей при форматировании ссылок применяются те же значения, что и к обычному тексту. При описании других псевдоклассов в CSS перед их значениями указывается символ ">".

К псевдоэлементам в стилях относятся свойства: first-letter - форматирует первый символ для первой строки; first-line - назначается отдельное форматирование первой строки блока текста; after - назначает месторасположения объекта после текущего элемента; before - назначает месторасположения объекта до текущего элемента.

Для задания размеров в CSS используется несколько способов: относительный размер в процентах, относительный размер при помощи словесного описания (larger, smaller, xx-small, x-small, small, medium, large, x-large, xx-large), абсолютный размер в типографских единицах - размер может задаваться в пунктах (pt), пиках (pc), пикселях (px), средней шириной буквы "m" (em), средней шириной буквы "x" (ex) , абсолютный размер в стандартных единицах длины - размер может задаваться в сантиметрах (cm), миллиметрах (mm), дюймах (in), абсолютный в пикселях (px)

Цвет может быть определен одним из трех способов: при помощи названия цвета: yellow, red, green, grey,.. ; шестнадцатеричным заданием цвета в формате #RRGGBB: #ff0000, #883490, #ffffff,..; десятичным заданием составляющих цвета в формате rgb(red, green, blue): rgb(255,0,0), rgb(100,23,78),..

Приведем несколько примеров описания таблицы стилей:

```
.epigraph {
   font-size: 12pt;
   font-style: italic;
   text-align: right;
   color: rgb(127,127,0);
}
p.big {
   font-size: 16px;
   font-weight: bold;
   color: #ff0000;
}
.menu {
   font-weight: bold;
   font-size: 9pt;
   font-family: arial, helvetica, sans-serif;
}
a:hover {
   color: #b63a3a;
   text-decoration: none;
}
```

Различные свойства

Свойство	**Описание**
content	позволяет вставлять генерируемое содержание в текст веб-страницы. Применяется совместно с псевдоэлементами after и before.content: *строка* | ***attr(параметр)*** | ***open-quote*** | ***close-quote*** | ***no-open-quote*** | ***no-close-quote*** | ***url*** | ***counter*** | ***normal*** | ***none*** | ***inherit;*** строка – текст, который добавляется на веб-страницу, строка должна браться в двойные или одинарные кавычки. attr(параметр) – возвращает строку, которая является значением параметра тега указанного в скобках. Например, ***img:after {content:attr(href)}*** добавит после изображения его адрес, т.е. значение параметра href. open-quote – вставляет открывающую кавычку, close-quote – вставляет закрывающую кавычку. url –абсолютный или относительный адрес вставляемого объекта. Если указанный файл браузер не может отобразить, то значение игнорируется. counter – выводит значение счетчика, заданного свойством counter-reset. normal - задается как none для псевдоэлементов before и after. inherit - наследует значение родителя.
orphans	Свойство orphans задает минимальное число строк текста, которое остается на предыдущей странице при печати документа. Это свойство работает в том случае, если весь текст размещается на двух и более печатных страницах. orphans: *число* | ***inherit***
page-break-after	Добавляет разрыв страницы при печати документа после заданного элемента. page-break-after: ***always*** | ***auto*** | ***avoid*** | ***left*** | ***right*** | ***inherit*** always – всегда добавляет разрыв страницы после элемента. auto – вставляет разрыв страницы при необходимости. avoid – запрещает разрыв страницы после элемента. left – пропускает одну или две страницы после элемента, чтобы следующая страница при печати была четной. right – пропускает одну или две страницы после элемента, чтобы следующая страница при печати была нечетной. inherit – наследует значение родителя.
page-break-before	Добавляет разрыв страницы при печати документа перед заданным элементом. page-break-before: ***always*** | ***auto*** | ***avoid*** | ***left*** | ***right*** | ***inherit*** always – всегда добавляет разрыв страницы перед элементом. auto – вставляет разрыв страницы при необходимости. avoid – запрещает разрыв страницы перед элементом. left – пропускает одну или две страницы перед элементом, чтобы следующая страница при печати была четной. right – пропускает одну или две страницы перед элементом, чтобы следующая страница при печати была нечетной. inherit – наследует значение родителя.
page-break-inside	Разрешает или запрещает разрыв страницы внутри элемента при печати: ***auto*** | ***avoid*** | ***inherit*** auto – вставляет разрыв страницы, avoid – запрещает разрыв страницы внутри элемента, inherit – наследует значение родителя.

windows	задает минимальное число строк текста, которое располагается на следующей странице при печати документа. widows: *число* | ***inherit***

Курсор

Определяет форму курсора мыши, которую он принимает при наведении на элемент.

scrollbar-arrow-color	Задает цвет стрелок на кнопках полосы прокрутки. scrollbar-arrow-color:*{Цвет};*
scrollbar-base-color	Задает цвет бегунка и кнопок-стрелок полосы прокрутки. scrollbar-base-color:{*Цвет*};
scrollbar-highlight-color	Задает цвет "освещенной" части бегунка и кнопок прокрутки полосы прокрутки (цвет левых и верхних их граней). scrollbar-highlight-color:{*Цвет*};
scrollbar-shadow-color	Задает цвет "неосвещенной" части бегунка и кнопок прокрутки полосы прокрутки (цвет правых и нижних их граней).
scrollbar-track-color	Задает цвет рабочей части полосы прокрутки, т.е. той ее части, по которой перемещается бегунок. scrollbar-track-color: {*Цвет*};

cursor: ***auto|crosshair|default|hand|move|text|wait|help***;

auto-указывает браузеру самому определять нужную форму курсора;

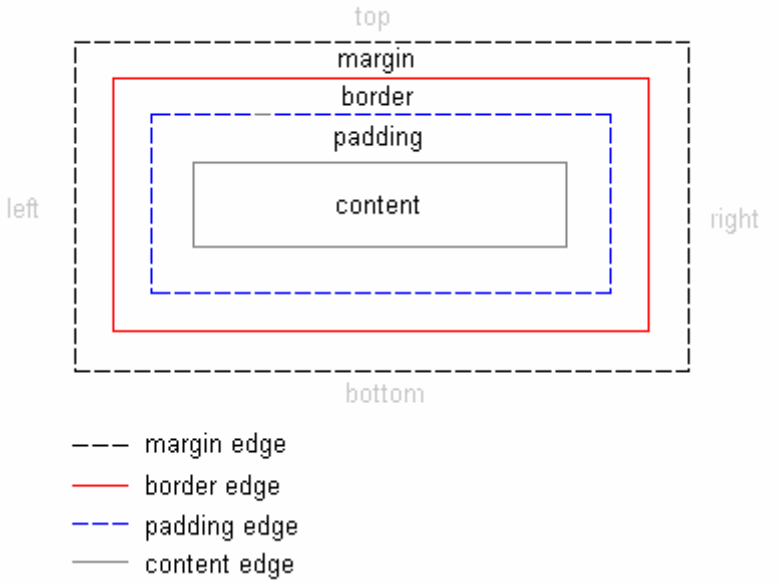
crosshair - крестообразный курсор; default - курсор по умолчанию, стрелка; hand - "указывающий перст"; move - стрелка, указывающая "на все четыре стороны"; text - текстовой курсор; wait - "песочные часы", курсор ожидания; help - стрелка с вопросительным знаком. Пример:

```
<html>
<head>
<title> Курсор для разных областей веб-страницы</title>
<style type="text/css">
.cross { cursor: crosshair }
. hand { cursor: hand }
</style>
</head>
<body>
<div class="cross"> курсор примет вид перекрестья.</div>
<a href="ref.htm" class=" hand ">Ссылка</a>
</body>
</html>
```

Представление документа в виде блоков

В модели форматирования каскадных таблиц стилей элементы HTML представлены в виде блоков. Каждый блок состоит из вложенных прямоугольников, в самом внутреннем из которых находится содержимого элемента. Прямоугольник содержимого окружает блок отступа (padding) между содержимым элемента и его границей. Следующим располагается блок границы (border). Самым внешним прямоугольником является прозрачный блок поля (margin). Для выделения блоков часто

используется тег . Внутреннюю часть блока может определять тег



Отступы, поля, позиционирование

Свойство	**Описание**
margin	Задает поля между элементами страницы:*{margin-top} [{margin-right}] [{margin-bottom}] [{margin-left}]*; Если задано одно значение, оно применяется ко всем четырем полям. Если заданы два значения, первое относится к верхнему и нижнему полю, а второе - к левому и правому. Если задано три значения, то первое применяется к верхнему полю, второе - к левому и правому, третье - к нижнему.
margin-top	Задает поле сверху как абсолютной величиной, так и процентом от высоты родителя: *auto|{Y}|{Y}%*;.
margin-right	Задает поле справа: *auto|{X}|{X}%*;
margin-bottom	Задает поле снизу: *auto|{Y}|{Y}%*;
margin-left	Задает поле слева: *auto|{X}|{X}%*;
padding	Задает отступ между элементом страницы и различными границами. : *{X}|{X}%*;
padding-top	Задает отступ до верхней границы: *{Y}|{Y}%*; Значение по умолчанию 0, для тега <td> 1.
padding-right	Задает отступ до правой границы: pading-right: *{X}|{X}%*;
padding-bottom	Задает отступ до нижней границы: pading-bottom: *{Y}|{Y}%*;
padding-left	Задает Отступ до левой границы: pading-left: *{X}|{X}%*;
width	Задает ширину свободно позиционированного элемента: *auto|{X}|{X}%*;
height	Задает высоту свободно позиционированного элемента: *auto|{X}|{X}%*;

position	static – блок позиционируется в соответствии с основным потоком форматирования; relative – положение блока вычисляется. absolute - положение блока указывается с помощью свойств left, top, right и bottom, относительно контейнера данного элемента. fixed - положение блока рассчитывается по алгоритму, используемому для схемы absolute, но после позиционирования блок привязывается либо к области просмотра, либо к странице
top	Задает вертикальную позицию верхней границы свободно позиционированного элемента относительно родителя: *auto|{Y}|{Y}%*;
bottom	Задает вертикальную позицию нижней границы свободно позиционированного элемента относительно родителя. bottom: *auto|{Y}|{Y}%*;
left	Задает горизонтальную позицию левой границы свободно позиционированного элемента относительно родителя. left: *auto|{X}|{X}%*;
right	Задает горизонтальную позицию правой границы свободно позиционированного элемента относительно родителя. right: *auto|{X}|{X}%*;
float	Определяет обтекание элемента другими слева или справа вместо помещения под ним. float: *none|left|right*;
clear	Устанавливает, с какой стороны элемента запрещено его обтекание другими элементами. Если задано обтекание элемента с помощью свойства float, то clear отменяет его действие для указанных сторон. clear: *none* | *left* | *right* | *both* | *inherit*
clip	*rect(Y1, X1, Y2, X2)* | *auto* | *inherit* определяет прямоугольник для позиционированного элемента, в котором будет показано его содержимое. Все, что не помещается в эту область, будет обрезано. clip работает только для абсолютно позиционированных элементов:
display	определяет, как элемент должен быть показан в документе: none|*block* | *inline* | *inline-table* | *list-item* | *none* | *run-in* | *table* | *table-caption* | *table-cell* | *table-column-group* | *table-footer-group* | *table-header-group* | *table-row* | *table-row-group*. none – скрыть блок, освободив место; block – начать и закончить блок с новой строки, inline – считать блок одним из символов текстовой строки; inline-block – с виду inline, а внутри — block; list-item – элемент списка; run-in – начинается как block, заканчивается как inline; inline-table – таблица с внешними свойствами как inline; table – позиционируется как таблица;
max-height	Устанавливает максимальную высоту элемента: *значение* | *проценты* | *none* | *inherit*
max-width	Устанавливает максимальную ширину элемента: *значение* | *проценты* | *none* | *inherit*
min-height	Задает минимальную высоту элемента: *значение* | *проценты* | *inherit*

min-width	Устанавливает минимальную ширину элемента: *значение* | *проценты* | ***inherit***
overflow	Управляет отображением содержания блока, если оно выходит за область заданных размеров: ***auto*** | ***hidden*** | ***scroll*** | ***visible*** | ***inherit.*** visible – отображается все содержание, даже за пределами области. hidden – отображается только область внутри элемента. scroll – добавляются полосы прокрутки. auto – полосы прокрутки добавляются только при необходимости. inherit – наследует значение родителя.
visibility	Предназначен для отображения или скрытия элемента, включая рамку вокруг него и фон. При скрытии элемента, место, которое элемент занимает, остается за ним: ***visible*** | ***hidden*** | ***collapse*** | ***inherit***
z-index	Любые позиционированные элементы на веб-странице могут накладываться друг на друга в определенном порядке, их размещением по z-оси и управляет z-index. Это свойство работает только для элементов, у которых значение position задано как absolute, fixed или relative. Значение: *число* | ***auto*** | ***inherit.*** Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше.

Рассмотрим некоторые свойства. margin - определяет размеры всех полей блока. По умолчанию значением является 0. Пример:

```
<html>
<head>
h1 {margin: 10px}
<style type="text/css">
p {
margin-top: 0px /* Отступ сверху */
margin-bottom: 0px; /* Отступ снизу */
}
</style>
</head>
<body>
<p> Убираем интервалы у параграфа.</p>
</body>
</html>
```

padding - задает отступы контента до границы блока. Если значение свойства одно, то значение применяется ко всем отступам блока. В случае двух значений первое задает верхний и нижний отступы, а второе – левый и правый отступы. При указании четырех значений они применяются к верхнему, правому, нижнему и левому отступам. По умолчанию значением данного свойства является 0.

Пример:

```
<html>
<head>
<style type="text/css">
#img {
padding: 10px; /* Поля вокруг изображения */
border: lpx solid black; /* Параметры рамки */
```

```
width: 100px; /* Ширина */
float: left; /* Выравнивание по левому краю */
}
</style>
</head>
<body>
<div>
<img src=”image.gif” id=” img ”>
Использование стилей для создания подписи под рисунком
</body>
</html>
```

Пример:

```
<html>
<head>
<style type="text/css">
p {
text-indent: 20px; /* Выступ первой строки */
padding-left: 20px /* Отступ всего текста слева */
padding-right: 20px /* Отступ всего текста слева */
}
</style>
</head>
<body>
<p>Использование свойства padding-left и padding-right для создание отступов текста слева и справа.</p>
</body>
</html>
```

Границы элементов

Свойство	**Описание**
border	Задает свойства границ элемента страницы: [{border-color}] [{border-style}] [{border-width}]; Значение по умолчанию ***medium none***.
border-color	Задает цвет: {***border-top-color***} [{***border-right-color***}] [{***border-bottom-color***}] [{***border-left-color***}];Может быть задано от одного до четырех значений. Если задано одно значение, оно применяется ко всем четырем границам. Если задано два значения, первое относится к верхней и нижней границам, а второе - к левой и правой. Если задано три значения, то первое применяется к верхней границе, второе - к левой и правой, третье - к нижней.
border-top-color	Задает цвет верхней границы элемента страницы: *{Цвет}*;
border-bottom-color	Задает цвет нижней границы элемента страницы: *{Цвет}*;
border-left-color	Задаёт цвет левой границы элемента страницы: *{Цвет}*;

border-right-color	Задает цвет правой границы элемента страницы. border-right-color: *{Цвет}*;
border-style	Задает тип сразу всех границ элемента страницы. border-style: *none*|*dotted*|*dashed*|*solid*|*double*|*groove*|*ridge*|*inset*|*outset;* none – запрещает рисование границы; dotted – рисует точечную линию; dashed –штриховую линию; solid – сплошную линию; double – двойную сплошную линию; groove – трехмерную вдавленную границу; ridge – трехмерную выпуклую границу; inset – трехмерную "ступеньку вверх"; outset – рисует трехмерную "ступеньку вниз".
border-top-style	Задает тип верхней границы элемента страницы: *none*|*dotted*|*dashed*|*solid*|*double*|*groove*|*ridge*|*inset*|*outset*;
border-bottom-style	Задает тип нижней границы элемента страницы: *none*|*dotted*|*dashed*|*solid*|*double*|*groove*|*ridge*|*inset*|*outset*;
border-left-style	Задает тип левой границы элемента страницы: *none*|*dotted*|*dashed*|*solid*|*double*|*groove*|*ridge*|*inset*|*outset*;
border-right-style	Задает тип правой границы элемента страницы: *none*|*dotted*|*dashed*|*solid*|*double*|*groove*|*ridge*|*inset*|*outset*;
border-width	Задает толщину границ элемента страницы: {*border-top-width*} [{*border-right-width*}] [{*border-bottom-width*}] [{*border-left-width*}]; Если задано одно значение, оно применяется ко всем четырем границам. Если заданы два значения, первое относится к верхней и нижней границам, а второе - к левой и правой. Если задано три значения, первое применяется к верхней границе, второе - к левой и правой, третье - к нижней. Толщина может быть задана числом или предопределенным значением : *thin*, *medium*, *thick*. Значение по умолчанию *medium.*
border-top-width	Задает толщину верхней границы элемента страницы. border-top-width: *medium*|*thin*|*thick*|*{Толщина}*;
border-bottom-width	Задает толщину нижней границы элемента страницы. border-bottom-width: *medium*|*thin*|*thick*|*{Толщина}*;
border-left-width	Задает толщину левой границы элемента страницы. border-left-width: *medium*|*thin*|*thick*|*{Толщина}*;
border-right-width	Задает толщину правой границы элемента страницы. border-right-width: *medium*|*thin*|*thick*|*{Толщина}*;
border-top	Задает все свойства верхней границы элемента страницы за один прием: [{*border-top-color*}] [{*border-top-style*}] [{*border-top-width*}]; Значение по умолчанию *medium none.*
border-bottom	Задает все свойства нижней границы элемента страницы: [{*border-bottom-color*}] [{*border-bottom-style*}] [{*border-bottom-width*}];
border-left	Задает все свойства левой границы элемента: [{*border-left-color*}] [{*border-left-style*}] [{*border-left-width*}];

border-right	Задает все свойства правой границы: [{*border-right-color*}] [{*border-right-style*}] [{*border-right-width*}];

Пример:

h1 {border-top-width: thin}

h1 {border-width: 5px}

border-top-color - указывает цвет верхней границы блока.

Пример:

p {border-top-color: black}

border-top-style - указывает стиль верхней границы блока.

Пример:

p {border-top-style: solid}

border-style - определяет стили всех границ блока. Если значение свойства одно, то значение применяется ко всем границам блока. В случае двух значений первое задает стили верхней и нижней границы, а второе – стили левой и правой границ. При указании четырех значений они применяются к верхнему, правому, нижнему и левому границам.

Пример:

```
<html>
<head>
<title> Использование стилей для отмены рамки изображений</title>
<style type="text/css">
img { border: none }
</style>
</head>
<body>
<img src=”image.gif”>
</body>
</html>
```

Визуальное форматирование

Свойства правил визуального форматирования позволяют правильно расположить фрагменты Web-страницы в окне браузера.

position: static - определяет способ позиционирования блока на странице. Блок позиционируется в соответствии с естественным потоком отображения элементов. Это значение задается по умолчанию.

position:relative – определяет смещение блока относительно его естественного положения в потоке отображения элементов.

position:absolute –размещает блок произвольным образом.

top - указывает смещение верхнего края блока относительно верхнего края родительского элемента.

bottom - задает смещение нижнего края блока относительно нижнего края родительского элемента.

position:left/ position:right - определяет смещение левого/правого края блока относительно левого/ правого края родительского элемента.

Пример:

```
<img src=”img.gif” style=”position: absolute; top: 30px;
bottom: 30px; left: 30px; right: 30px;”>
```

float - изменяет положение блока. Значениями свойства могут быть: left – блок смещается влево, а его содержимое отображается вдоль правой стороны блока; right – блок перемещается вправо, а его содержимое выводится вдоль левой стороны блока; none – блок не смещается.

Пример:

```
<html>
<head>
<title> Использование стилей для создания буквицы</title>
<style type="text/css">
. letter {
font-size: 150%; /* Размер шрифта буквицы */
float: left; /* Выравнивание по левому краю */
color: green; /* Цвет буквицы */
padding: 3px /* Отступ между буквицей и текстом */
}
</style>
</head>
<body>
<span class=”letter”>Б</span>уквица
</body>
</html>
```

clear - указывает стороны блока, где плавающие элементы не ставятся. По умолчанию плавающие элементы устанавливаются на всех сторонах. Значениями свойства могут быть:

left – все плавающие элементы на левой стороне блока будут опущены вниз.

right – все плавающие элементы на правой стороне блока будут опущены вниз.

none – плавающие элементы на всех сторонах блока.

Пример:

```
h1 {float: left }
```

width - указывает ширину блока содержимого элемента. По умолчанию ширина блока вычисляется браузером автоматически.

Пример:

```
p {width: 100px}
```

height -определяет высоту блока содержимого элемента. По умолчанию высота блока вычисляется браузером автоматически.

Пример:

```
p {height: 100px}
```

Визуальные эффекты

Свойство overflow - управляет поведением элемента в случае, когда его размеры не соответствуют размерам блока отображения. Значениями свойства overflow могут быть:

scroll – добавляет полосы прокрутки к блоку отображения; hidden – обрезает элемент в соответствии с размерами блока; auto – добавляет полосы прокрутки к блоку отображения в случае, если размеры содержимого элемента превосходят размеры блока отображения; visible – принуждает элемент сжаться или увеличиться, чтобы полностью отобразиться в заданном блоке для рисунка или увеличивает размеры блока отображения в случае текста. Пример:

```
<html>
```

```
<style type="text/css">
body { overflow: hidden }
</style>
<body>
<p>Запрет полосы прокрутки на веб-странице</p>
</body>
</html>
```

clip - обрезает видимое изображение элемента. По умолчанию усечение не производится. Значениями свойства clip могут быть:

границы видимого прямоугольного изображения элемента в виде rect(<top>, <right>, <bottom>, <left>), где параметры <top>, <right>, <bottom>, <left> определяют верхнюю, правую, нижнюю, левую границы видимого изображения.

auto – усечение изображения элемента не производится.

Свойство visibility - определяет, будет ли отображаться элемент в окне браузера. Значениями свойства могут быть: visible – элемент будет отображаться; hidden – элемент не будет выводиться на экран.

Новое в CSS3

border-radius, box-shadow, text-shadow, opacity, border-image, rgba(x,y,z,a), backgroun-size, background:url(), url(), column-count, transition, transform, @font-face, background-clip, animation, @animation-keyframes.

```
Закругление углов: .radius_border {     border-radius: 25px;
                                        -moz-border-radius: 25px;
                                         -webkit-border-radius: 25px;
                                                }
Тень от блока:box_shadow { box-shadow: 5px 5px 2px #000;
                                        -moz-box-shadow: 5px 5px 2px #000;
                                                 -webkit-box-shadow: 5px 5px 2px #000;
                                        }
Тень от текста:         TEXT.text_shadow { text-shadow: 2px 2px 2px #000;
                                                   -moz-text-shadow: 2px 2px 2px #000;
                                                   -webkit-text-shadow: 2px 2px 2px #000;
                                        }
Transition: .pic { margin-left: 80%;
                  -webkit-transform: rotate(-30deg) scale(1.5);
                          -webkit-transition: all 2s ease-in;
                 }
```

Глава 5. Язык JavaScript

Введение

Язык программирования JavaScript был разработан в 1995 году компанией Netscape Communication Corporation, известной также как создатель Web-браузера Netscape Navigator. JavaScript — это язык скриптов, исполняемых в первую очередь на стороне клиента и позволяющих улучшить внешний вид и управление Web-страницами. JavaScript — это интерпретируемый язык программирования. Это означает, что браузер выполняет каждую строку скрипта последовательно, строка за строкой. Все проведенные изменения вступают в силу сразу после загрузки документа в окне браузера. Интерпретируемые языки программирования позволяют легко переносить приложения на различные платформы.

JavaScript принес на клиентскую страницу динамику и интерактивность. Ближайшей альтернативой JavaScript является технология Flash, содержащая средства

для работы с мультимедиа. К стандартным проблемам, решаемым на JavaScript можно, например, отнести следующие:

Улучшение пользовательского интерфейса. С помощью Javascript реализуются анимационные рисунки, часы, календари, бегущие строки, средства перемещения по документу и отображения информации. Одно из применений состоит в изменении рисунка, используемого в качестве гиперссылки при наведении на него указателя мыши. JavaScript позволяет последовательно отображать рисунки, хотя анимация и не входит в число преимуществ JavaScript.

Использование строки состояния. Строка состояния — это серая панель в нижней части окна браузера. Она позволяет отобразить описание команд меню и кнопок панели инструментов, например, с помощью бегущих строк.

Средства перемещения по документу. JavaScript позволяет создать меню и раскрывающиеся списки для перемещения между различными страницами Web-узла.

Окна с сообщениями и другие элементы. JavaScript позволяет создать окна сообщений с предупреждением, напоминанием или подтверждением. С помощью JavaScript можно создаеть новое окно браузера

Проверка и изменение форм. Формы позволяют проводить самые различные операции - от заказа товаров в электронных магазинах до получения сведений о популярности узла. JavaScript часто используется для проверки правильности заполнения полей формы.

Определение версии браузера. В JavaScript предусмотрена возможность определения типа браузера и выполнения команд, поддерживаемых только им.

Внедряемые модули. Надстройки браузера, позволяют отображать в документах HTML разные типы данных, например для отображения видео, и аудио информации.

Другие проблемы, решаемые с помощью JavaScript – это отображение изменяющихся данных, таких как текущее время или дата; программирование переменного содержания в зависимости от браузера, имени пользователя, текущей даты, или других условий; изменение внешнего вида элементов страницы при возникновении события, например щелчка мышью; выполнение вычислений на клиентской странице.

Структура JavaScript

JavaScript можно представить в виде объединения трёх частей:

- ядро (ECMAScript),
- объектная модель браузера (Browser Object Model или BOM),
- объектная модель документа (Document Object Model или DOM).

Спецификация ECMAScript описывает типы данных, инструкции, ключевые слова, операторы, объекты, регулярные выражения.

BOM представляется объектами window, navigator, location, history, frames, screen, а также функциями setTimeout() и setInterval(). Помимо управления окнами, в рамках объектной модели браузера обеспечивается: задержка в исполнении кода, системные диалоги, управление адресом открытой страницы, управление информацией о браузере, управление информацией о параметрах монитора, управление историей просмотра страниц, поддержка работы с HTTP cookie.

Согласно объектной модели документа DOM, документу можно поставить в соответствие дерево объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции: нахождение узлов, изменение узлов, удаление узлов.

Включение скриптов JavaScript в HTML-код

Чтобы добавить сценарий JavaScript на Web-страницу, используется пара дескрипторов <script> и </script>. Дескриптор <script> указывает браузеру рассматривать

дальнейший текст как скрипт. Обнаружив дескриптор </script>, браузер возвращается к выполнению HTML.

Традиционное включение скрипта в HTML- документ в недавнем прошлом имело вид:

```
<!-- пример #pr1: шаблон HTML для скрипта -->
<html> <head>
<title> Шаблон HTML </title>
<script language="javascript">
<!--Маскируемся, начало JavaScript
 //…код скрипта
// снятие маскировки; конец JavaScript
-->
</script>
</head>
<body >
</body></html>
```

В настоящее время использование атрибута language="JavaScript" считается устаревшим (deprecated) и отсутствует в DTD. Вместо него используется атрибут type, значением которого является "text/javascript", принимаемое также по умолчанию:

```
<script type="text/javascript">
//JavaScript код
</script>
```

Рассмотрим пример.

```
<!-- пример pr2 -->
<html>
<head>
<title>Наша начальная страница</title>
</head>
<body>
<p>Добро пожаловать на первую скриптовую web-страницу.</p>
<script type="text/javascript">
document.write("Hello");
</script>
</body>
</html>
```

Метод write() объекта document позволяет отображать результат сценария на Web-странице. Атрибуты “language” и “type” тега <script> можно вообще не указывать, Они будут установлены по умолчанию.

В следующем примере скрипт выбирается из файла myscript.js и помещается в заголовок HTML документа

```
<DOCTYPE HTML >
<html>
<head>
<title> заголовок страницы идет здесь </ title>
<script src="myscript.js"> </ script>
<link rel="stylesheet" href="code.css" type="text/css">
</ head>
<body>
```

```
<p>содержание страницы идет здесь, включая все HTML-теги.</ p>
</ body>
</ html>
```

Код скрипта выполняется незамедлительно, как только встретится, если только этот код не является функцией. Скрипты могут размещаться в различных частях документа HTML.

1. В разделе body. В этом случае результат сценария отображается на Web-странице при ее загрузке в браузере.

2. В заголовке программы между <head> и </head>. Это гарантирует выполнение скрипта до начала загрузки HTML-кода страницы. В заголовке размещаются функции, которые не выполняются при загрузке страницы, а вызываются по имени из различных мест скрипта.

3. В дескрипторе(теге) HTML. Такая конструкция называется обработчиком событий. Обработчик событий представляет собой отдельный тип сценария, который даже не требует использования дескриптора <script> для его обозначения. Рассмотрим пример:

```
<!-- Использование кнопки и события -->
<html> <body>
<form>
<input type="button" value="click me" onclick="window.alert(' Hello!')">
</form>
<a href="delete.php" onclick="return confirm('Вы уверены?'); ">
Удалить</a>
</body>
</html>
```

В приведённом примере при нажатии на ссылку, функция confirm('Вы уверены?') вызывает модальное окно с надписью «Вы уверены?», а return false – блокирует переход по ссылке.

4. В отдельном файле с расширением js. В этом случае включаемый файл, содержащий скрипт, объявляется в HTML-коде с помощью тега <script> с атрибутом src, в котором прописывается путь к файлу. Например: <script type="text/javascript" src="/jspr/pr.js"></script>

В директории /jspr/ должен находиться файл pr.js, который содержит код JavaScript без тегов <script> и </script>.

Отметим, что символы нижнего и верхнего регистров в JavaScript различаются, как и в C++. Операторы должны отделяться друг от друга точкой с запятой, только если они находятся в одной строке. Операторы записываются на нижнем регистре.

Создание простых сценариев

Пример: Сценарий определения текущего времени. Для получения текущей даты и времени в соответствии с временным поясом места жительства пользователя используется объект Date.

```
<!-- пример pr3 -->
<html> <head>
<title>Отображение даты</title>
</head>
<body>
<h1>Текущее время и дата</h1>
<p>
```

```
<script type="text/javascript">
var now = new Date();
var localtime=now.toString();
var hours=now.getHours();
var mins=now.getMinutes();
var secs=now.getSeconds();
document.write("<b>Текущее время: </b>" + localtime + "<br>");
document.write("<font size='+5'>");
document.write (hours + ":" + mins + ":" + secs);
document.write("</font>");
</script>
</body> </html>
```

Результат:
pr3-Текущее время и дата
Текущее время: Tue Oct 09 2012 15:24:56 GMT+0400
15:24:56

Пример: Бегущая строка в окне статуса состояния снизу

```
<!-- пример pr4 -->
<html>
<head><title>Пример бегущей строки</title>
<script type="text/javascript">
var msg = "Это пример бегущей строки.";
var spacer = "... ...";
var pos = 0;
function ScrollMessage()
 {window.status = msg.substring(pos, msg.length) + spacer + msg.substring(0,pos);
  pos++;
  if (pos > msg.length) pos = 0;
  window.setTimeout("ScrollMessage()",200);
//установка времени прерывания
 }
ScrollMessage();//вызов функции
</script>
</head>
<body>
<h1>Пример бегущей строки</h1>
Взгляните на строку состояния в нижней части окна браузера - окне состояния.
</body></html>
```

Пример: Смена изображения

```
<!—пример pr5: Использование события для смены изображения и перехода по
гиперссылке на bsu.by-->
<html> <body>
<h1>Событие onmouse </h1>
<a href="http://bsu.by">
<img src="s1.jpg" name="ris" width="100px" hight="100px"
onmouseover="ris.src='s2.jpg'; "onmouseout="ris.src='s3.jpg';">
```

```
</a>
</body>
</html>
```

Комментарии. Скрытие сценариев от браузеров

Как уже отмечалось, для просмотра документов ранее использовались дескрипторы комментариев HTML, чтобы приказать старым браузерам не исполнять, а игнорировать программу скрипта. Например:

```
<script type="text/javascript">
<!--
document.write("браузер поддерживает JavaScript"); //может игнорироваться
-->
</script>
```

В современных браузерах сами такие комментарии игнорируются.

Синтаксис самого JavaScript использует другие комментарии, подобные на комментарии C++:

```
//это однострочный комментарий
/*Этот сценарий содержит различные команды и операторы, а также
многострочные комментарии */
```

Отладка скриптов. Ввод и вывод данных

Распространенный способ отладки заключается в многократном вызове метода alert() объекта window, который выводит стандартное диалоговое окно с текстом и кнопкой OK или методов, выводящих диалоговые окна confirm() и prompt():

1. alert («сообщение»). Метод позволяет выводить диалоговое окно с заданным сообщением и кнопкой OK. Окно, создаваемое посредством alert(), является модальным и останавливает все последующие действия пользователя и программ, если его не убрать, щелкнув на кнопке OK.

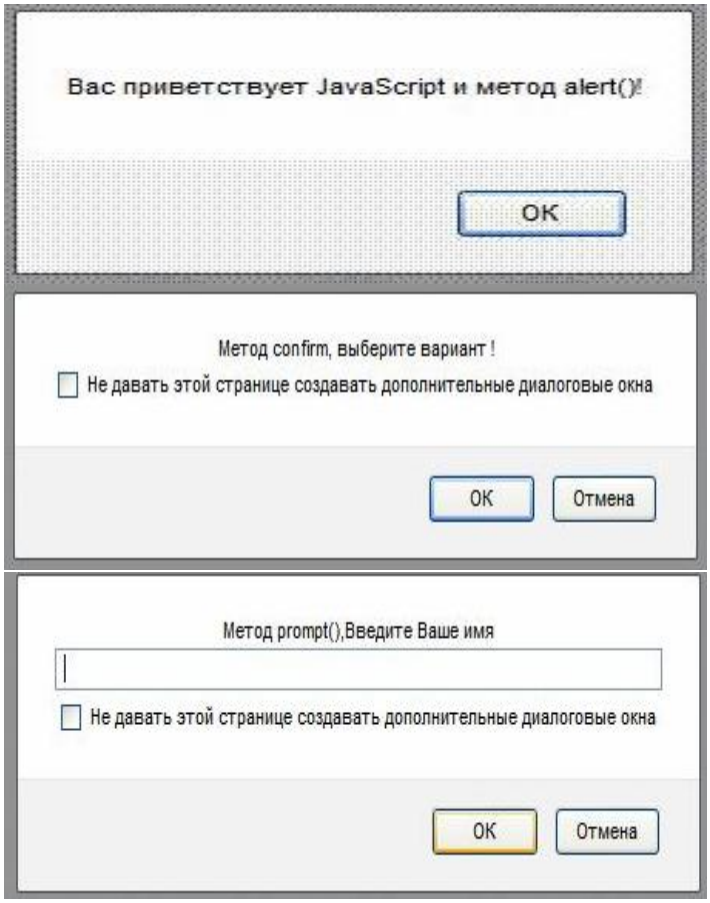
2. confirm («сообщение»). Окно, создаваемое посредством confirm(), также является модальным. Если пользователь щелкнет на кнопке «OK», то метод вернет логическое значение true, а если он щелкнет на кнопке «Отмена» (Cancel), то возвращается значение false**.**

3. prompt(«сообщение», «значение_поля_ввода_данных»);

Метод prompt выводит на экран диалоговое окно с сообщением, и с текстовым полем, в которое пользователь может ввести данные. Кроме этого, в окне предусмотрены две кнопки: OK и Отмена (Cancel). Если пользователь щелкнет на кнопке OK, то метод вернет содержимое поля ввода данных, а если он щелкнет на кнопке Отмена, то возвращается логическое значение false. Объект window, из которого происходит вызов методов, может не указываться при вызове.

```
<!-- пример pr6: Вывод текста в окно -->
<html> <head>
<title> вывод окна</title>
<script type="text/javascript">
alert("Вас приветствует JavaScript и метод alert()!");
confirm("Метод confirm, выберите вариант !");
prompt("Метод prompt(),Введите Ваше имя");
</script>
</head>
<body >
<p>
```

```
<h1 style="color:blue">Вывод окон: alert(), confirm() и prompt() </h1>
<hr><p><p>Страница документа </center>
</body>
</html>
```



Помимо этого, существует более продвинутые способы отладки и специальные отладчики, позволяющие производить отладку скриптов.

Одним из лучших редакторов для разработки кода JavaScript является редактор “Adobe DreamWeaver CS5” с возможными расширениями (JQuery API) и отладчиком FireBug. Программисты Java могут использовать для работы с JavaScript среду разработки Eclipse и плагины к Eclipse JSEclipse и Aptana Studio.

Среди известных JavaScript библиотек можно отметить Adobe life, Dojo Toolkit, Extjs, jQuery, Mootools, Prototype, Qooxdoo.

Описание языка

Типы данных

JavaScript поддерживает базовые типы данных и объекты. Имеется пять базовых типов данных: числа; строки; булев (логический) тип; undefined – неопределенный; null - пустой. Фактический интерес с точки зрения хранения данных представляют три: числа, логические значения и строки. Числам, строкам и логическим данным соответствуют объектные типы Number, String и Boolean (классы оболочки), которые включают большое число методов. Таким объектам могут присваиваться значения соответствующих базовых типов. Хотя строковые литералы относят к базовым типам, фактически переменные типа строка представляют собой ссылки на объекты.

Переменные и объекты в JavaScript объявляются с помощью ключевого слова var, и представляют собой ссылки, например:

var x;//undefined -переменная

После этого можно задать значение переменной:

x = "Sample string";

Язык JavaScript позволяет создавать переменные без их объявления:

y = "Second string";

При этом переменная y попадет в свойства глобального объекта window и становится глобальной. В языке JavaScript переменные не имеют строго закрепленного типа, тип переменной определяется данными, которые она хранит. Фактически переменная представляет собой ссылку, которую можно установить на различные значения несколько раз. Пример:

```
var man1 =10;
man1 = "Superman";
```

Объектные переменные создаются через оператор new или с помощью инициализации:

```
var man2 = new String("Superman");
```

В Javascript - все переменные представляют объекты, кроме базовых типов. Везде, где необходимо, базовые типы автоматически преобразуются в объекты. Так можно, например, определять длину литерала:

```
alert("Superman".length);
```

Преобразование типа

Преобразование типа может производиться неявно в выражениях или с помощью оператора присваивания. Чтобы преобразовать число в строку, достаточно сложить его с пустой строкой. Преобразование типа можно сделать явным через вызов конструктора объекта-оболочки:

```
var a="15"+2 //152 – складываются как строки
var a="15"-2 //13 – вычитания у строк нет, только у чисел
var a=true+false // 1
var test = Boolean("Stroka") // true
var x = Number("1234")
var z=String(x);
```

Для преобразования цифровой строки в число может быть использованы функции parseInt(“строка”) и parseFloat(“строка”). Еще один способ состоит в использовании методов объектов. Метод toString () возвращает строку, соответствующую вызывающему его объекту.

```
var x=Number.toString()
var y=number.toString()
```

Число number преобразуется к объекту Number, затем вызовется метод toString(). Преобразовать число в строку с указанным числом знаков можно с помощью методов toFixed и toExponential().

```
<!-- пример pr7: Преобразования типов-->
<html> <head>
<title> вывод окна</title>
</head>
<body >
<P> <CENTER>
<HR><P><P>Страница документа </center>
<script type="text/javascript">
 var x=parseInt("35.55");
```

```
 alert(2+2+ " "+25);//425
confirm(" x= " + x);//35
x=Number("25.5")+0.5; //26
 prompt(" x= "+x);//26
</script>
</body>
</html>
```

Числа

В JavaScript численный тип данных включает целые и вещественные числа. Целые числа могут принимать значения от –255 до 255, вещественные могут принимать большие значения в пределах ±10e308 или быть точными в пределах ±2.2250e-308. Числа также могут записываться в экспоненциальной форме, например 1.25e-05.

В языке JavaScript также имеется возможность записи чисел в восьмеричной системе счисления: вначале ставится 0, затем цифры от 0 до 7.

Для записи целого шестнадцатеричного числа вначале ставится ноль, затем буква x, затем само число, которое может содержать цифры от 0 до 9 и буквы от A до F. Числа в шестнадцатеричной системе счисления могут использоваться для битовых операций, а также для хранения цветов .

Если результат математической операции выходит за допустимые пределы, переменная принимает специальное значение Infinity – бесконечность. При сравнении положительная бесконечность всегда больше любого действительного числа, и наоборот, отрицательная бесконечность всегда меньше любого действительного числа.

Еще одним специальным значением является NaN (not a number). Типичным примером операции, которая возвращает NaN, является деление на ноль. Для определения, является ли значение переменной NaN, используется функция isNaN(value), которая возвращает true, если value является действительным числом(включая бесконечность), и false, если значение переменной равно NaN.

К специальным числам относятся: Number.MAX_VALUE-максимальное значение числа, Number.MIN_VALUE-минимальное значение числа, Number.NaN - не число, Number.POSITIVE_INFINITY - положительная бесконечность, Number.NEGATIVE_INFINITY - отрицательная бесконечность

Строки

Строковый литерал – это последовательность символов ограниченная двойными или одинарными кавычками. Строка представляет собой объект, который имеет свойство length (длину строки) и ряд методов, из которых приведем несколько часто употребляемых:

charAt(index : Number) : String – возвращает символ, находящийся на определенной позиции;

concat([string1 : String [, ... [, stringN : String]]]]) : String – соединяет строки (аналогично оператору «+»);

substr(start : Number [, length : Number]) : String – возвращает подстроку, которая начинается с определенной позиции и имеет определенную длину;

substring(start : Number, end : Number) : String – возвращает подстроку, которая начинается и заканчивается в указанных позициях.

Строки в JavaScript являются неизменяемыми, метод charAt() есть , а метода SetcharAt() нет. Строки сравниваются по значению (s1==s2), а не по ссылке как объекты.

Одиночный символ фактически рассматривается как строка. В JavaScript используются специальные символы(управляющие последовательности): \n, \r, \t, \’, \\, \xcode.

Булев тип

Переменные булевого типа могут принимать одно из двух значения: true – истина; false – ложь. Переменные булевого типа часто используются в условном операторе if. Пример:

```
var d= true;
if (d) { alert(“Hello, World!”); }
```

Переменные типа Undefined и Null

Тип undefined используется для несуществующих переменных или переменных, значения которых еще не определены. В следующем примере переменная x будет иметь значение undefined.

```
var x;// undefined
```

Тип null означает пустое значение. Пример объявления:

```
var x = null;
```

Массивы

В JavaScript массив – это упорядоченная коллекция различных данных. По сути это объект, такой же, как Object, который определяется также как ассоциированный массив(хеш). Первый способ объявления:

```
var myArr = new Array();
```

Все элементы массива пронумерованы, первый элемент массива имеет нулевой индекс. Доступ к элементу массива осуществляется с помощью квадратных скобок и номера этого элемента. Пример:

```
myArr[3] = “Hello!”;
var x = myArr[3];
```

Второй способ создания массива через присваивание значений. Имя массива в любом случае является ссылкой.

```
var emptyArr = [];
Var x=[1,2,3];
Var y=x;
x[0]=100;
alert(y);// выедет 100,2,3
```

Массивы могут быть многомерными. Объявление и обращение к многомерному массиву выглядит так:

```
var myArr = [[1,2],[3,4],[5,6]]; // массив размером 3x2
alert(myArr[2][1]);//выводится элемент, равный 6.
```

Операторы и выражения

Выражения на одной строке программы JavaScript разделяются точкой с запятой; если выражения находятся на разных строках, то точка с запятой может и не ставиться. В этом отличие от языков C++ и Java, в которых точкой с запятой используется для завершения операторов. Точка с запятой является пустым оператором.

Несколько выражений могут объединяться в блок с помощью фигурных скобок { }, однако блоки не определяют область видимости и время жизни переменных, как в C++ .

Оператор присваивания

Значение присваивается одной переменной или сразу нескольким переменным, поскольку "=" является пустым оператором:

```
var n = j = k = 2;
```

Арифметические операторы

В языке JavaScript поддерживаются арифметические операторы: «+», «-», «*», «/», «%», <<+=>>, << -=>>, ...

Операция деления выполняется следующим образом:

```
var y=6/4; //1.5
var z=6%4;//остаток от деления - 2
```

Так как оператор сложения "+" является перегружаемым, при работе со строками он означает конкатенацию последних s = "str1" + "str2".

Остальные арифметические операции выполняются в числовом контексте, в котором строки автоматически преобразуются в числа, например

```
var w="12"+3;//123
var x="5"*"20";//100
var v="12"-3;//9
alert(""+x+v+w);
```

Операция инкремента «++», служит для прибавления 1 к операнду, соответственно декремент «--» – используется для вычитания 1 от операнда.

Операторы сравнения

В языке JavaScript поддерживаются следующие операторы сравнения:

«<», «<=», «>», «>=», «!=», «==» – равно; «===» – равно и операнды одинакового типа (тождественно равно, строгое сравнение); «!==» – не равно или операнды разных типов.

Для иллюстрации оператора строгого сравнения приведем пример:

```
var x = 0;
var y = false;
alert(x==y);
//true из-за приведения операндов в выражении к одному типу
alert(x===y);
//выдаст на экран false, так как операнды разных типов.
```

Логические операторы

В языке JavaScript поддерживаются следующие логические операторы:

«&&» – логическое И; «||» – логическое ИЛИ; «!» – логическое НЕ.

Оператор «?»

Оператор «?» возвращает значение первого выражения, если условие истинно, и второго выражения, если условие ложно при синтаксисе:

```
условие ? выражение1 : выражение2;
```

Пример использования:

```
var x = 6; var y = 9;
var res = x < y ? "x меньше y" : "x больше или равно y";
alert(res);
//выведет экран фразу «x меньше y»
```

Оператор typeof

Оператор typeof *операнд* возвращает строковое значение, которое определяет тип операнда: "number", "string", "boolean", "object", "function" и "undefined".

Условный оператор if

Оператор имеет следующий вид:

```
if (условие)
 // выражение или блок выражений
else
 //выражение или блок выражений
```

Ветка else может отсутствовать.

Операторы организации циклов

Ниже приведен синтаксис для организации циклов. Цикл выполняется, пока значение логического выражения равно true

```
1. while (условие){
```

```
                    // выражение или блок выражений
}
2. do
        // выражение или блок выражений
while (условие);
3.Цикл for имеет синтаксис:
for (нач_значения; условие; изменение_нач_значений){
 //выражение или блок выражений
}
Пример цикла for показан ниже:
<!-- пример pr8 -->
<html>
<body>
<script type="text/javascript">
for (var i = 0; i < 4; i++) {
 alert(i );
}
//на экран последовательно будут выведены числа:0 1 2 3
document.write("i=",i);// i=4
</script>
</body>
</html>
```

4. Оператор цикла for…in служит для просмотра всех свойств в объекте и присваивания их значений переменной:

```
for (имя_переменной in объект)
 выражение или блок выражений
```

Примером может служить перебор всех стилей какого-либо элемента:

```
<!-- пример pr9 -->
<html>
<head>
<style type=text/css>
        #myP{color:green;
        font-style: italic;
        font-variant:normal;
        font-weight: bold;
        font-size: 30px;
        font-family: arial, sans-serif;}
</style>
</head>
<body>
<p id="myP">test</p>
<script type="text/javascript">
var obj;
for (obj in document.getElementById("myP").style) {
alert(obj+'='+document.getElementById("myP").style[obj]);
}
</script>
</body>
</html>
```

Цикл выведет все свойства объекта style элемента myP.

Операторы continue и break могут применяться во всех циклах, первый служит для перехода к следующей итерации в цикле, второй для выхода из цикла.

Оператор with

Оператор with присоединяет имя объекта к имени свойства объекта. С помощью оператора with можно обращаться со свойствам объекта в сокращенном виде:

```
with (object) {
 //свойства
}
```

Оператор switch

```
switch (переменная) {
 case условие1: выражение; break;
 case условие2: выражение;  break;
// ...
 case условие N: выражение;  break;
 default: выражение
}
```

Ветка default выполняется, если ни одно из предыдущих условий не выполняется и может отсутствовать.

Функции

Стандартные функции.

В JavaScript используются стандартные функции:

eval() : позволяет выполнить строку, содержащую выражение, как javascript-код. Это полезно когда код формируется в ходе выполнения скрипта. Пример реализации простейшего калькулятора приведен ниже:

```
<!-- пример pr11 -->
<html>
<body>
<input type="text" id="calc" />//вводится текст выражения
<input type="button" value="Calculate!"
onclick="alert(eval(document.getElementById('calc').value))" />
//вычисляется выражение
</body>
</html>
```

Функции parseInt(), parseFloat() предназначены для того, чтобы превращать строку в число. Функции encodeURI(), decodeURI(), encodeURIComponent(), decodeURIComponent() предназначены для работы с URI (Uniform Resource Identifier). isNaN() - если числовая операция не может вернуть число, то используется специальное нечисловое значение NaN, и указанная функция. isFinite() - проверяет Infinite (бесконечность).

Функции пользователя

В языке JavaScript определение функций имеет синтаксис:

```
function name(список_параметров) {
 //тело функции-список выражений;
 return (значение)
}
```

Параметры в списке разделяются запятыми и могут отсутствовать. Оператор return используется для выхода из функции и возврата значения в вызвавшее функцию

выражение. Возвращаемое значение и сам оператор return могут отсутствовать. Например:

```
function hello() {
 alert(“Hello World!”)
}
hello(); //выведет фразу «Hello World!»
```

Как и в C++ вызов функции производится следующим образом: name(arglist);

Если в описании функции определено несколько параметров, а при вызове эти параметры функции не передаются, то неопределенным параметрам присваивается значение undefined. Пример:

```
<!-- пример pr12 -->
<html>
<body>
<script  type="text/javascript">
function sum(arg1, arg2, arg3) {
 var res = arg1 + arg2;
 if (arg3){ res = res + arg3;}
    else alert(arg3);//Выводит значение undefined
 return res;
}
var x = 1; var y = 2; var z = 3;
alert(sum(x,y,z)); //выведет на экран 6
alert(sum(x,y));//выведет 3, прибавление arg3 к res //не  производится, так как arg3=undefined.
alert(sum(x));//Выводит значение NaN
</script>
</body>
</html>
```

Передача параметров по значению и по ссылке

Базовые типы данных (числа, строки, булевы переменные) передаются в функцию в качестве параметров по значению, которое не изменится во время выполнения функции. Данные объектных типов, например массивы, передаются по ссылке(адресу) и могут изменяться во время выполнения функции. Пример, иллюстрирующий данное поведение, приведен ниже:

```
<!-- пример pr13 -->
<html>
<body>
<script type="text/javascript">
function passval(arg1) {
 var arg=5; arg1=arg++;
 return arg1;
}
var x = 1; var y = 2;
alert(passval(x)); //выведет в окно 5
alert(x); //выведет 1, значение x не изменится
function passRef(arr1) {
        arr1[0] = "NEWfirst";
}
 var y = ["first", "second"];//массив
```

```
passRef(y);
alert(y);//выведет в окно "NEWfirst", second
</script>
</body>
</html>
```

При вызове в функцию передается ссылка на объект. И любые изменения этого объекта не исчезнут после завершения функции. Учитывая такую разницу в поведении типов, необходимо определять, с каким типом приходится работать. И для этого в JavaScript существуют два оператора – typeof и instanceof.

Глобальные и локальные переменные

Существует два вида переменных: глобальные, которые объявлены перед функцией и видны во всем документе во время выполнения скрипта и локальные, которые объявляются при помощи оператора var и видны только внутри функций. Если мы используем необъявленную внутри функции переменную, то это будет либо глобальная переменная, либо свойство объекта window. Если объявлена локальная переменная, то доступ к глобальной переменной с тем же именем закрывается – все действия переменных будут происходить с локальной переменной. Аргументы функции автоматически становятся локальными переменными.

Если перед именем переменной ключевое слово var не ставится, то переменная также считается глобальной и принадлежат объекту window как свойство.

```
<!-- пример pr14 -->
<html>
<body>
<script type="text/javascript">
var x="global2";
function myFunc() {
        var x = "local";//переопределение глобал_переменной
        y = "global1";
var z=5;
}
myFunc();
alert(window.y); //вернет global1
alert(x); //вернет слово global2
alert(z); //ошибка
</script>
</body>
</html>
```

Функции в JavaScript также считаются объектами(call objekt). Аргументы и локальные переменные являются свойствами такого объекта.

Всегда можно использовать объект arguments, при помощи которого и можно было бы добраться до параметров. Могут также использоваться функциональные литералы вида:

```
var f=function(x){return x*x}
```

Модель событий

Модель событий (Event Model) – это реализация способности JavaScript реагировать на изменение состояния документа в браузере. При возникновении события, например, при нажатии на ссылку или при отправке заполненной формы вызывается код JavaScript или функция - обработчик события. Событие представляет собой указатель на эту функцию и записывается в виде: on+”событие”. Ниже приведены основные события:

onfocus – элемент получает фокус, onblur – теряет фокус; onchange –элемент формы теряет фокус, а его значение изменилось; onclick/ ondblclick – происходит при нажатии мышкой на любой визуальный элемент; onkeydown/onkeyup – пользователь нажимает/отпускает клавишу на клавиатуре; onkeypress – пользователь нажимает и отпускает клавишу на клавиатуре; onload/onunload – происходит, когда документ загружен/покидается; onmousedown/onmouseup – пользователь нажимает/отпускает клавишу мыши; onmousemove – пользователь двигает курсором мыши над элементом; onmouseover/onmouseout – указатель мыши попадает/покидает область элемента; onreset – происходит, когда значения элементов формы сбрасываются; onselect – происходит, когда пользователь выделяет текст в элементе формы; onsubmit – происходит, когда пользователь отсылает форму;

В первых версиях браузеров события определялись как дополнительные атрибуты в дескрипторах HTML. Чтобы обработать какое-либо событие, необходимо добавить в тэг атрибут, соответствующий одному из вышеперечисленных событий. После имени атрибута - события записывается знак равенства, после него указывается значение атрибута. Значением атрибута является обработчик события - функция или группа команд JavaScript.

В следующем примере событие onclick вызывается при нажатии кнопки "Browser", чтобы получить имя и версию браузера

```
<!-- пример pr15: получение типа и версии браузера -->
<html> <head>
<title>Test of Browser name</title>
</head>
<body>
<h1 align=center>Проверка типа браузера</h1>
<P><hr>
<form name=fr>
<input type=button name=browser value=Browser onClick=
"alert(window.navigator.appName+navigator.appVersion)">
</form>
<a href="pr3.htm" onclick="alert('click')"> click me now</a>
<p id="test" onclick="alert('Тест')">Тест</p>
</body> </html>
```

Имя и версия браузера здесь возвращаются через свойства объекта navigator.appName и navigator.appVersion.

Вместо кода JavaScript после onclick можно написать функцию - обработчик события.

```
<html>
<head>
<title> pr16-Функция будет исполняться при событии</title>
<script type="text/javascript">
function myalert(){alert('Тест')}
</script>
<body>
<p id="test" onclick="myalert();">Тест</p>
</body>
</html>
```

События могут рассматриваться не только как атрибуты, но и как свойства объектов JavaScript. Вот пример установки обработчика события click на элемент с id="button":

```
<html> <head>
<title> prr-Функция будет исполнятся при событии</title>
```

```
<script type="text/javascript">
function doMy() {
 confirm('OK!')
}
</script>
</head>
<body>
<input  type="button" id='button'  value="Нажми" >
<script>
document.getElementById('button').onclick = function() {
 alert('Click')
}
document.getElementById('button').onclick = doMy
</script>
</body> </html>
```

Здесь onclick свойство, а не атрибут. Обработчиком является анонимная функция. Анонимную функцию можно и не создавать, а использовать обычную:

```
function doMy() {
 alert('OK!')
}
document.getElementById('button').onclick = doMy
```

Свойству объекта здесь присваивается функция-обработчик doMy без скобок.

Ключевое слово this

Если обработчику события необходимо передать ссылку на элемент, вызвавший это событие, это осуществляется с помощью указателя this на текущий объект. Пример:

```
<DOCTYPE html>
<html> <head>
<title>Test of Browser name</title>
</head>
<body>
<!-- пример pr16: -->
<a href="pr3.htm" onclick="show(this);"> click me</a>
<script  type="text/javascript">
function show(_obj) {
 alert(_obj.innerHTML);
//нажатие на ссылку выводит в диалоговое окно «click me»
}
</script>
</body> </html>
```

После нажатия на ссылку происходит переход на другую страницу. Это действие можно отменить, если обработчик события вернет значение false. Например:

```
<a href="sample.html" onclick="return showInfo(this);">click me</a>
<script type="text/javascript">
function showInfo(_obj) {
 return confirm("Do you want go to another page?");
/*при нажатии на ссылку будет выведен диалог с кнопками OK и Cancel; если затем будет нажата Cancel, то браузер не перейдет по адресу, на который указывает ссылка*/
}
```

```
</script>
```

Модель событий JavaScript 1.2 и объект Event.

В язык JavaScript 1.2 добавлен объект Event. Он содержит свойства, описывающие событие. Каждый раз, когда происходит какое-либо событие, объект Event передается соответствующей программе обработки. Сами события Abort, Blur, Clicки др., аналогичны вышеперечисленным.

Ниже обрабатывается событие нажатия кнопки мыши, и определяется, какая именно из них была нажата:

```
<html><head>
<title>pr17-Test of Event</title>
<script >
function whichButton(event)
{if (event.button == 2)
  {  alert("Вы щелкнули правой кнопкой мыши!");  }
else  {  alert("Вы щелкнули левой кнопкой мыши!");  }
}
</script>
</head>
<body onmousedown="whichButton(event)">
<p>Щелкните кнопкой мыши в любом месте документа(надписи)</p>
</body>
</html>
```

В следующем примере на экран выводится изображение. Если щелкнуть над ним клавишей мыши, появится окошко сообщений, где будут показаны координаты той точки, где в этот момент находилась мышь

```
<html><head>
<title>pr18-Test of Event</title>
</head>
<body >
<a href="#" onClick="alert('x:'+event.x+'y:'+event.y); return false;">
<img src="avtor.jpg" width=249 height=264 border=0></a>
</body> </html>
```

Здесь используются свойства объекта Event event.x и event.y, чтобы узнать координаты мыши. Инструкция return false; используется здесь для того, чтобы браузер обрабатывал далее данную ссылку.

Объект Event имеет свойства:

currentTarget идентифицирует текущий элемент, в котором в данный момент обрабатывается событие. data - содержит данные для обработчика событий; LayerX/LayerY - горизонтальное/ вертикальное положение курсора относительно слоя. Синоним x, y.

pageX / pageY- Горизонтальное/ Вертикальное положение курсора (в пикселах) относительно окна браузера. screenX/ screenY - горизонтальное/ вертикальное положение курсора относительно экрана. target – идентифицирует DOM элемент, инициировавший событие (узел, в котором произошло событие). timeStamp содержит число миллисекунд с 1 января 1970 года до момента срабатывания события. result содержит последнее возвращенное обработчиком события значение; type - тип события; which - значение нажатой клавиши или клавиши мыши. Методы объекта Event: isDefaultPrevented()- позволяет определить, был ли вызван метод preventDefault(); event. isImmediatePropagationStopped() - позволяет определить, был ли вызван метод event.stopImmediatePropagation(); isPropagationStopped() -позволяет определить, был ли

вызван метод stopPropagation(); preventDefault()- отменяет семантическое действие по умолчанию для события; stopImmediatePropagation()-предотвращает всплытие объекта события вверх по DOM-дереву и запуск последующих обработчиков события, связанных с текущим элементом.

Перехват события.

С помощью обработки событий можно добиться того, чтобы объект, соответствующий окну, документу или слою, перехватывал и обрабатывал событие еще до того, как для этой цели объектом указанной кнопки будет вызван обработчик событий. Рассмотрим следующий пример:

```
<html>
<head>
<script language="JavaScript">
window.captureEvents(Event.CLICK);
window.onclick= handle;
function handle(e) {
alert("Объект window перехватывает это событие!");
return true; // т.е. проследить ссылку
}
</script>
</head>
<body>
<a href="test.htm">Click on this link</a>
</body>
</html>
```

Как видно, мы не указываем программы обработки событий в тэге <a>. Вместо этого пишем window.capture Events (Event.CLICK);с тем, чтобы перехватить событие Click объектом window. Если же Вы хотите перехватывать несколько событий, то Вам следует отделить их друг от друга символами |. Например:

```
window.captureEvents(Event.CLICK | Event.MOVE);
```

Помимо этого в функции handle(), назначенной на роль обработчика событий, мы пользуемся инструкцией return true;. В действительности это означает, что браузер должен обработать и саму ссылку, после того, как завершится выполнение функции handle(). Если же Вы напишете вместо этого return false;, то на этом все и закончится.

Если теперь в тэге <a> Вы зададите программу обработки события onClick, то поймете, что данная программа при возникновении данного события вызвана уже не будет. И это не удивительно, поскольку объект window перехватывает сигнал о событии еще до того, как он достигает объекта link. Если же Вы определите функцию handle() как

```
function handle(e) {
alert("The window object captured this event!");
window.routeEvent(e);
return true;
}
```

то компьютер будет проверять, определены ли другие программы обработки событий для данного объекта. Переменная e - это наш объект Event, передаваемый функции обработки событий в виде аргумента.

Кроме того, Вы можете непосредственно послать сигнал о событии какому-либо объекту. Для этого Вы можете воспользоваться методом handleEvent(). Это выглядит следующим образом:

```
<html>
```

```
<script >
window.captureEvents(Event.CLICK);
window.onclick= handle;
function handle(e) {
document.links[1].handleEvent(e);
}
</script>
<a href="test.htm">"Кликните" по этой ссылке</a><br>
<a href="test.htm"
onClick="alert('Обработчик событий для второй ссылки!');">Вторая ссылка</a>
</html>
```

Все сигналы о событиях Click, посылаются на обработку по второй ссылке - даже если Вы вовсе и не щелкнули ни по одной из ссылок!

Следующий скрипт демонстрирует, как скрипт может реагировать на сигналы о нажатии клавиш.

```
<html>
<script >
window.captureEvents(Event.KEYPRESS);
window.onkeypress= pressed;
function pressed(e) {
                        alert("Key pressed! ASCII-value: " + e.which);
}
</script>
</html>
```

Методы addEventListener, removeEventListener, attachEvent.

Еще одна возможность обработки событий состоит в использовании методов addEventListener (событие, обработчик события, этап события) и removeEventListener (событие, обработчик события, этап события). С помощью этих методов обработчик события можно задать или отменить для любого DOM-элемента в самом скрипте без назначение атрибута on + "событие". Здесь 1-й параметр – это событие без префикса "on" ("click", "mousemove", "blur", "load" ...), 2-й параметр – обработчик события, функция которая будет выполняться при события указанной в первом параметре, 3-й параметр – этап события. Если false, то обработчик события будет выполняться на этапе всплытия события (если существуют вложении элементы с тем же обработчиком события), если true - то на этапе перехвата события.

```
<p id="test">Тест</p>
<script type="text/javascript">
function alerting(){alert('Тест')};
var test=document.getElementById('test');
test.addEventListener('click',alerting,false);
</script>
```

IE не поддерживает addEventListener () и removeEventListener (), у него есть аналогичные: attachEvent (on + событие, обработчик события) и detachEvent (on + событие, обработчик события).

Для кроссбраузерной привязки обработчиков события надо делать перебор методов, используя операторы if...else:

```
<script type="text/javascript">
var test=document.getElementById('test');
function alerting(){alert('Тест')}
```

```
if(test.addEventListener){//код для всех браузеров
test.addEventListener("click",alerting,false)
}
else{//код для IE
test.attachEvent("onclick",alerting)
}
</script>
```

Следует заметить, что скрипт с методом addEventListener() оптимально прописывать в конце Html-документа или уже после загрузки всех необходимых для выполнения скрипта DOM-элементов. Можно также использовать метод загрузки скрипта после происшествия window.onload

Исключения: throw/catch/finally

Работа с исключениями в JavaScript организована, как и в C++ или Java:

```
try {
 //...
 throw {message: "err!"}
 //..
}
catch (e) {
 alert("Ошибка!"+e)
}
```

Блок try{} указывает выполнить код внутри блока. В случае ошибки осуществляется выход из этого блока и переход на блок обработки исключений: catch (e) {}

Оператор throw {} генерирует объект – исключение, после чего осуществляется выход из блока try. Обычно при этом генерируются потомки встроенного класса Error:

```
throw new Error("server timeout")
```

Часто при перехвате исключений надо перехватить определенный класс исключений. Оператор catch такого не умеет, поэтому полный код обработки будет выглядеть так:

```
try {
 // код ...
} catch(e) {
 // ловим нужное исключение
 if (e instanceof ConnectionError) {
 // обрабатываем его
} else {
 // пробрасываем незнакомое исключение дальше
 throw e
 }
 } finally {
 //finally выполняется вне зависимости - было исключение или нет
 }
```

В этом примере присутствует блок finally, взятый в javascript из java. В стандартной схеме try..catch..finally, код в finally выполнится при любом результате работы try/catch, и туда удобно ставить чистки, уведомления о конце процесса.

```
<!-- пример pr18 -->
<html>
<head>
```

```
<title>Наша начальная страница</title>
</head>
<body>
<script type="text/javascript">
function showErrorInfo(e) {
 document.write(e, ":<BR>");
 document.write("Источник ошибки: ", (e.number >> 16) & 0x1FFF, "<BR>");
 document.write("Номер ошибки: ", e.number & 0xFFFF, "<BR>");
 document.write("Описание ошибки: ", e.description);
}
var x;
try {
 x = y; // Ошибка: переменная y не определена
}
catch (e) { // Создает локальный объект e класса Error
 showErrorInfo(e);
}
</script>
</body></html>
```

Будет выведено:

[objectError]: Источник_ошибки:10 Номер_ошибки:5009 Описание ошибки: 'y' - определение отсутствует

Объектная модель

В JavaScript используются следующие виды объектов:

- пользовательские объекты, которые создаются пользователями с помощью конструктора и объекта Object или с помощью инициализации;
- встроенные объекты языка JavaScript: String — строка; Array — массив; Date — дата и время; Math — математические функции;
- объекты браузера, которые создаются автоматически при загрузке документа: window — объект верхнего уровня в иерархии объектов; location — содержит свойства, описывающие местонахождение текущего документа; history — содержит информацию обо всех ресурсах, к которым пользователь обращался во время текущего сеанса; navigator — содержит информацию о версии браузера;

Кроме этого, имеются объекты frames, screen а также методы setTimeout() и setInterval().

- объекты, связанные с тэгами HTML и стилями CSS — в соответствии с моделью DOM большинство тэгов HTML и стилей CSS соответствуют свойствам объекта document. document — содержит свойства innerText, innerHTML, textContent, которые относятся к текущему HTML-документу и сами также являются объектами.

Методы: getElementById(), getElementByTopName.

Пользовательские объекты

Объектно-ориентированные языки (например, Java и C++) основаны на базовых понятиях классов объектов и экземпляров (instances) объектов. JavaScript основан на прототипах и в нем есть только объекты. Вверху иерархии находится объект Object с методами toString() и свойством prototype. П*рототип объекта* определяет начальный набор свойств объекта. В процессе работы объект может получать новые свойства через прототип и может сам выступать в качестве прототипа при создании новых объектов.

Существует два основных способа создания новых объектов в JavaScript:

1. Использование конструктора объектов.

2. Использование инициализатора объекта.

При создании объектов в JavaScript используют специальные методы, называемые *конструкторами*. Имя конструктора определяет имя класса. Декларация конструктора совпадает с декларацией класса. Иными словами, мы определяем конструктор как функцию, которая создает объекты с заданным начальным набором свойств и их значений. Затем создаем объекты вызовом операции *new имя_конструктора()*. Например:

```
<DOCTYPE html>
<!-- пример #pr1: шаблон HTML для скрипта -->
<html>
<head>
<title>  Объект HTML </title>
</head>
<body >
<script language="javascript"  type="text/javascript">
alert("Begin");
function Book() {//конструктор
 this.paper = true;
}
var myBook = new Book(); //Объект
alert(myBook.paper);                          //выведет  true
alert("MyBooktype  "+typeof(myBook)); //object
alert(Book.prototype);                        // [object Object]
alert(Object.getPrototypeOf(Book));       //function(){}
alert(Object.getPrototypeOf(myBook))   //[object Object]
for (var i in myBook)
  alert('myBook'+i  )          //myBook.paper
alert("End");
</script>
</body></html>
```

Конструктор вызывается с помощью оператора new. Оператор new создает пустой обект и передает ссылку this на него конструктору. Конструктор отвечает за выполнение соответствующих действий для нового объекта. Конструктор может обращаться к создаваемому объекту, используя указатель this, таким образом, мы можем добавить свойство создаваемому объекту. В конструктор можно передавать параметры, чтобы задать начальные свойства создаваемого объекта. Новое свойство объекта создается просто присваиванием ему значения. Имеется только одна копия прототипа, которую используют все объекты, созданные с помощью одного конструктора. Свойства, созданные с помощью прототипа, будут иметь одинаковые значения для всех объектов данного конструктора.

```
<DOCTYPE html>
<html>
<head>
<title> Объект HTML </title>
</head>
<body >
<script language="javascript"  type="text/javascript">
function Book(isPaper, name, version) {//конструктор с параметрами
 if (isPaper) this.paper = true;
 else this.paper = false;
```

```
        this.name = name;      this.version = version;
    this.aboutBook = function() {
       document.write("Book: " + this.name + " " + this.version);   }
    }
    var myBook = new Book(true, "Programming","3");
    myBook.price = 100;//добавление свойства объекту
    Book.prototype.benifit = 20;// добавление свойства прототипу объект
    alert(myBook.aboutBook); //выведет  function(){…..}
    alert(myBook.price); //100
    alert(myBook.benifit)//20
    alert(Object.getPrototypeOf(Book));// function(){}
    alert(Object.getPrototypeOf(myBook));//[object Object]
    var myBook1 = new Book(false, "Progr","5");
    alert(myBook1.price); //undefined
    alert(myBook1.benifit)//20
    for (var i in myBook1)//отсутствует price
      alert('myBook1'+i  )//paper, name,version, aboutbook, benifit
    </script>
    </body></html>
```

Свойство price добавлено только для объекта myBook, а свойство benefit для всего прототипа. Свойство, являющееся функцией, называется методом.

Объект JavaScript представляет набор свойств, образующих ассоциированный массив. Для доступа к свойству объекта используется синтаксис: имя_объекта.имя_свойства. Если название свойства задано текстовой строкой, то доступ к свойству возможен и так: *имя_объекта*[*"имя_свойства"*]

Мы можем удалить свойство или ранее созданный объект с помощью операции `delete`, например: `delete myBook;`

Прототипы

Каждый объект имеет свойство prototype, которое определяет его структуру. Прототип относится к порождающему объекту и немного похож на указатель super в языке Java. Свойство prototype является свойством конструктора, прототипированные свойства относятся ко всем объектам (аналогично static в Java). Пример прототипирования объектов приведен ниже:

```
    Book.prototype.paper = false;
    Book.prototype.isPaperBook = function() {
     if (this.paper) alert("This is a paper book");
     else alert("This is not a paper book");
    }
    function Book(isPaper) {
     if (isPaper) this.paper = true;
    }
    var myBook = new Book(true);
    myBook.isPaperBook(); //выведет фразу «This is a paper book»
```

В этом примере показано создание метода объекта и свойства.

JavaScript позволяет нам задать новый прототип для класса пользовательских объектов (прототипы встроенных объектов доступны только для чтения). Рассмотрим такой пример:

```
function Circle(radius) {//констуктор1
  this.radius = radius;
}
Circle.prototype.dl = function() {
```

```
  return Math.PI * 2*this.radius;
}
function OCircle(x, y, radius) {
  this.x = x;
  this.y = y;
  this.radius = radius;
}
OCircle.prototype = Circle.prototype;
var myCircle = new OCircle(0, 0, 1);
document.write(myCircle.dl());
```

В этом примере сначала определяется класс объектов Circle со свойством radius и методом dl(). Затем определяется класс OCircle, конструктор которого дополнительно содержит координаты центра окружности. Затем указывается, что он наследует прототип класса Circle. После этого создается объект myCircle и вызывается метод dl(), унаследованный от прототипа класса Circle. JavaScript поддерживает наследование, основанное на прототипах.

С каждым конструктором связан соответствующий прототип объекта, и каждый объект, созданный конструктором, содержит неявную ссылку на этот прототип. Прототип, в свою очередь, может содержать ссылку на свой прототип. Так образуется *цепочка прототипов*. Ссылка на свойство объекта — это ссылка на первый прототип в цепочке прототипов объекта, который содержит свойство с данным именем.

Хеш-таблицы в JavaScript

Объект Object в JavaScript находится в вершине иерархии и в то же время представляет собой обычный ассоциативный массив или "хэш". Он хранит любые соответствия "ключ => значение" и имеет несколько методов. Следующие два варианта создания объекта эквивалентны:

```
var o = new Object()
var o = {}
```

Есть два способа добавления свойств в объект. Первый - точка, второй - квадратные скобки:

```
o.test = 6// эквивалентные записи
o["test"] = 6
var name = 'test'
o[name] = 6
```

Имя свойства "test" является ключом в ассоциативном массиве, по которому лежит значение 5. Еще один пример:

```
var obj = new Object();
obj.property = 10;
alert(obj['property']);// то же самое, что и obj.property.
```

Удаление свойств осуществляется с помощью оператора delete:

```
alert(delete obj['property']); //выведет true
alert(obj['property']); //выведет undefined
```

Последний пример – перебор всех свойств объекта:

```
var obj = new Object();
var prop = '';
for(var i in h)
 prop += i + ' : ' + h[i] + '\n';
alert(props);
```

Переменная i внутри цикла содержит имя свойства, а не значение.

Создание объектов с помощью инициализатора

Инициализатор объекта имеет вид:

{*свойство*:*значение* [,*свойство*:*значение*]}
Здесь *свойство* — идентификатор, задающий имя свойства, а *значение* — выражение, задающее значение этого свойства.
Например, объект myBook может быть создан так:
```
var myBook = {name: "Programming", version: "6"};
```
Добавим еще одно свойство options, , которое является объектом:
```
var myBook = {name: " Programming", version: "6",
  options: {enableJava: true, enableCookies: false}};
```

Встроенные объекты String, Array, Date, Math

Объект String

Экземпляр объекта String можно объявить двумя способами:

var имя_переменной = new String (" строковое_ значение ")

var имя_переменной = "строковое_значение"

Свойства String:

length — количество символов (включая пробелы) в строке;

prototype — свойство, позволяющее добавить новые свойства и методы ко всем создаваемым строковым объектам, если существующих недостаточно. Пример: создаем новый метод для всех строковых объектов. Содержание метода определяется пользовательской функцией myFunc().

```
<!-- пример pr17 -->
<html>
<head>
<title>Наша начальная страница</title>
</head>
<body>
<SCRIPT type="text/javaScript">
function myFunc() {
return "Шура";
}
String.prototype.myName= myFunc;
var mystring=new String("Балаганов")
mystring += " Автор этой книги - " + mystring.myName();
document.write (mystring);//Балаганов Автор этой книги - Шура
</script>
</body>
</html>
```

Методы объекта String:

big(), small() – помещает текст строки внутрь тега <big> или <small>

bold()/italics() – помещает текст строки внутрь тега <bold>/<i>

fixed () – помещает текст строки внутрь парного тега <tt>

fontcolor (цвет), fontsize (размер) – помещает текст строки внутрь парного тега <font> с установленным атрибутом цвета, размера,

link (интернет-адрес) – преобразует строку в гиперссылку, указывающую на адрес, переданный в качестве параметра

charCodeAt (номер символа) – возвращает код символа, номер которого передан в качестве параметра, в формате Unicode,

fromCharCode (список кодов символов Unicode, разделенных запятыми) – возвращает строку, созданную из символов, Unicode-коды которых переданы в качестве параметров. Текущая строка не изменяется.

concat (список строковых значений, разделенных запятыми) – объединяет текущую строку со всеми строками, переданными в качестве аргументов

charAt (номер символа) – возвращает символ, номер которого передан в качестве параметра

indexOf (подстрока, начало поиска) – возвращает номер позиции подстроки в текущей строке. Второй параметр задает номер символа, с которого начинается поиск, если этот параметр пропущен, то поиск начинается с начала строки.

lastIndexOf (подстрока, начало поиска) – то же самое, что и indexOf, но поиск ведется до конца строки.

slice (начало фрагмента, конец фрагмента) – возвращает фрагмент строки в виде объекта. Если второй параметр пропущен, выбираются все символы до конца строки.

split (разделитель, лимит) – возвращает массив, строк, полученных в результате разделения текущей строки. Символ-разделитель передается первым параметром. Второй параметр, если он присутствует, задает лимит количества элементов в результирующем массиве.

substr (начало фрагмента, конец фрагмента) – возвращает фрагмент строки, заданной длины. Если второй параметр опущен, то выбираются все символы до конца строки,

substring (начало фрагмента, конец фрагмента) - возвращает фрагмент строки, последний символ во фрагмент не включается

toLowerCase ()/toUpperCase () – конвертирует все символы строки в нижний регистр/в верхний регистр

toString () – возвращает значение строки.

Объект Array

Свойства Array: length — количество элементов в массиве; prototype — свойство, позволяющее добавить новые свойства и методы ко всем созданным объектам.

Методы Array:

concat (список добавляемых элементов) – возвращает массив, получившийся в результате объединения текущего массива и элементов, перечисленных в списке. join (разделитель) – возвращает строку, получившуюся в результате слияния значений всех элементов, разделенных разделителем; pop () – удаляет последний элемент массива и возвращает его. Если массив пуст, возвращает undefined;

shift () – удаляет первый элемент массива и возвращает его; unshift (список добавляемых элементов) – возвращает массив, получившийся в результате объединения текущего массива и элементов, перечисленных в списке, причем элементы вставляются в начало текущего массива;

push (список добавляемых элементов) – добавляет в массив элементы, перечисленные в списке, и возвращает новую длину массива. Список может содержать другие массивы.

reverse () – возвращает массив, порядок элементов которого изменен на противоположный;

sort (функция сортировки) – возвращает массив, заполненный отсортированными элементами текущего массива. Функция сортировки должна принимать два параметра и возвращать одно из следующих значений: 1, если первый больше второго, -1, если второй больше первого, и 0, если они одинаковы. Если функция сортировки опущена, то выполняется символьная сортировка.

slice (индекс первого элемента, индекс последнего элемента) – возвращает массив, образованный их элементов текущего массива, от первого элемента включительно до последнего исключительно. Если индекс последнего элемента пропущен, выбираются все элементы до конца массива. splice (индекс первого элемента, количество удаляемых элементов, список добавляемых элементов, разделенных запятыми)

Объект Number (Число)

Свойства

MAX_VALUE — константа, значение которой равно 1.7976931348623157e+308.

MIN_VALUE — константа, значение которой равно 5e-324.

NEGATIVE_INFINITY - число, меньшее, чем Number.MIN_VALUE.

POSITIVE_INFINITY - число, большее, чем Number.MAX_VALUE.

NaN — константа, имеющая значение NaN (Not a Number).

prototype — свойство, играющее такую же роль, что и в случае объекта String.

Методы

toExponential(число) - число в экспоненциальной форме;

toFixed (количество) - представляет число в форме с фиксированным количеством цифр после точки;

toString (основание) - возвращает строковое представление числа в системе счисления с указанным основанием.

Объект Math (Математика)

Свойства Math (Math.свойство)

E - постоянная Эйлера,

LN10 - значение натурального логарифма числа 10,

LN2 - значение натурального логарифма числа 2,

LOG10E - значение десятичного логарифма экспоненты (числа в),

LOG2E - значение двоичного логарифма экспоненты,

PI - значение постоянной п,

SORT1_2 - значение квадратного корня от 1/2,

SORT2 - значение квадратного корня из 2.

Методы Math

Math .метод (параметры)

abs (число) — возвращает модуль (абсолютное значение) числа;

acos (число), asin (число) — возвращает арккосинус или арксинус;

atan (число) — возвращает арктангенс числа;

atan2(x, y) — возвращает угол в радианах между горизонтальной осью и прямой, проведенной через начало координат и точку с координатами x и y;

cos (число), sin(число) — возвращает косинус числа или синус;

tan (число) — возвращает тангенс числа.

round (число) — округляет число до ближайшего целого;

ceil (число) — округляет число до ближайшего целого – большего или равного;

floor (число) — округляет число до ближайшего целого – меньшего или равного;

exp (число) — возвращает число e в степени число;

log (число) — возвращает натуральный логарифм числа;

max (a1,a2), min (a1,a2) — возвращает большее или меньшее из чисел a1, a2;

pow (число1,число2) — возвращает число1 в степени число2; Например, Math.pow(10,3) выведет 1000

sqrt(число) — возвращает квадратный корень из числа;

random() — возвращает случайное число между 0 и 1;

```
<!--пример pr18: функция вычисления площади круга -->
<html>
<head>
<title>Функция вычисления площади круга </title>
</head>
<body>
<script type="text/javascript">
```

```
function sq(r) {
document.write("Задали r= ", r ," для функции.","<BR>")
return Math.PI*r * r;
}
var r=2;
document.write("Площадь круга равна ",sq(r),".")
</script>
</body> </html>
```

Объект Date (Дата)

Объект Date предоставляет набор методов для работы с датой и временем. Текущее время в объекте Date берется из операционной системы. Дата и время хранятся в виде числа, показывающее количество миллисекунд с 1 января 1970 года. Нумерация дней недели начинается с нуля, нулю соответствует воскресенье.

Экземпляр объекта Date объявляется следующим образом:

имяОбъекта = new Date([год, месяц, число, часы, минуты, секунды, миллисекунды])

Методы объекта Date

getFullYear(), getYear() – год (с 1970-..)

getMonth() – месяц(0-11), getDate() – число(1-31), getDay() – день недели (0-6)

getHours() – часы (0-23), getMinutes() – минуты (0-59), getSeconds() – секунды (0-59), getTime(), getMilliseconds() - миллисекунды

getUTCFullYear() – год UTC (1970- …), getUTCMonth() – месяц UTC

getUTCDate() – число UTC, getUTCDay() – день недели UTC

getUTCHours() – часы UTC, getUTCMinutes() – минуты UTC

getUTCSeconds() – секунды UTC, getUTCMilliseconds() – миллисекунды UTC

setYear(год), setFullYear(год) – установка года

setMonth(месяц) - установка месяца, setDate(число) - установка числа, setDay(день) - установка дня недели (от 0-6)

setHours(часы) - установка часов, setMinutes(число) - установка минут, setSeconds(число) - установка секунд

setMilliseconds(число), setTime(число) - установка миллисекунд

setUTCFullYear(число) – установка года UTC

setUTCMonth(число), setUTCDate(число), setUTCDay(число), setUTCHours(число), setUTCMinutes(число), setUTCSeconds(число), setUTCMilliseconds(число)

getTimezoneOffset () – разница в минутах по отношению к UTC

toDateString() - Строка с датой (без времени) в формате браузера

toGMTString() - Строка с датой и временем в глобальном формате

toLocaleDateString() - Строка с датой без времени в локализованном формате

toString() - Строка с датой и временем в формате браузера

```
<!--пример pr19:часы.Использование методов объекта Date-->
<html> <head>
<title>Clock</title>
<script type="text/JavaScript">
function clockform()
  {d=new Date();
 time=d.getHours()+":"+d.getMinutes()+":"+d.getSeconds();
          document.formcl.fclock.value=time;
  setTimeout("clockform()",100); }
</script>
```

```
</head>
<body onLoad="clockform()">
<center>
<form name=formcl metod="get">
<input name=fclock maxlength=8 size=8>
</form> </center>
</body></html>
```

Объект Function

Функция в JavaScript также является объектом и создается следующим образом: name= new Function (arguments)

{тело функции}

Свойства:

arguments — массив значений параметров, переданных функции. Индексация элементов массива производится с 0. length — количество параметров, указанных в определении функции

caller — содержит ссылку на функцию, из которой была вызвана данная функция; если функция не вызывалась из другой функции, то значение свойства равно null

Методы :

toString() — возвращает определение функции в виде строки.

Объект RegExp **и регулярные выражения**

В JavaScript регулярные выражения представлены объектом RegExp. Свойства RegExp принимают значения литералов регулярных выражений, заключенных в скобки /…/. Регулярные выражения из нескольких символов в стиле Perl записываются в косых скобках. Рассмотрим простой пример: /stud/. Регулярное выражение ищет все строки, содержащие подстроку stud. Выражение /^stud/ означает, что stud должно находиться в начале слова (student), а /stud$/ - в конце (restud).

Символы могут объединяться в группы путем помещения в квадратные скобки. Квадратные скобки /[абв]/ означают “любой символ а,б,в из перечисленных в скобках”. Символ ^ в группе соответствует отрицанию, то есть указывает символы, не соответствующие шаблону. /[^абв]/ означает “любой символ отличный от а,б,в ” .

Для создания регулярных выражений могут быть использованы интервалы:

/[0-9]/ - совпадает с любой десятичной цифрой от 0 до 9;

/[a-z]/ - совпадает с любым символом нижнего регистра от a до z;

/[a-zA-Z0-9]/ - совпадает с любым символом или цифрой из набора Latin.

Конструкция [^a-zA-Z] совпадает с любым символом, не входящим в указанные интервалы (a-z и A-Z).

Специальные символы записываются с обратным слешем \ - признаком особой интерпретации следующего за ним символа:

\d - обозначает любую десятичную цифру;

\D - обозначает любой символ кроме десятичной цифры;

\w - алфавитно-цифровой символ /[a-zA-Z0-9_]/

\W - символ, не являющийся алфавитно-цифровым /[^a-zA-Z0-9_]/

\s - пробельный символ

\S - символ не являющийся пробельным

\n – Символ перехода на следующую строку. Строка\nЕще строка

\r – Символ возврата на начало строки. Текст\r

\t – Символ табуляции из нескольких пробелов. \tКрасная строка

\\ – Обратный слеш

\b забой, совпадает с границами слов: /sa\b/. Противоположный символ, \b, совпадает с чем угодно, кроме границы слова: /sa\b/.

Служебный символ . (точка) означает любой символ поэтому для поиска точки ее надо экранировать символом обратный слеш \. Шаблону соответствует знак точки в строке.

Символы +, *, ? и {...}, обозначающие количество повторений отдельного символа или конструкции, заключенной в квадратные скобки, называют квантификаторами. Принцип их действия проще всего пояснить на примерах:

[p]+ означает один или несколько символов p, стоящих подряд;

[p]* означает ноль и более символов p, стоящих подряд;

[p]? означает ноль или один символ p;

[p]{2} означает ровно два символа p, стоящих подряд;

[p]{2,3} означает от двух до трех символов p, стоящих подряд;

[p]{2,} означает два и более символов p, стоящих подряд.

Например, шаблон /stu+/ совпадает с последовательностью stu, за которой могут следовать один или несколько дополнительных символов u. Шаблон /st{2,4}/ совпадает с символом s, за которым следуют от 2 до 4 экземпляров символа t. /\w{3}\d?/ - три буквы и необязательная цифра. /\s+Java\s+/ - несколько пробелов до слова или после него. Выражение /[\d]+/ используется для поиска цифровой подстроки, выражение /([\d]+)000/ может использоваться при поиске денежных сумм.

Шаблон /<([\w]+)>/ совпадает с конструкциями, заключенными в угловые скобки, - например, тегами HTML.

Подстроки в регулярных выражениях можно группировать при помощи круглых скобок: /домен – (by|ru|uk|com)/ соответствует строке домен – by или другой.

Оператор | (или) проверяет совпадение одной из нескольких альтернатив. ( ) – Логическая группировка выражений, которая (может)+ повторяться.

В регулярных выражениях могут использоваться флаги. Флаг i указывает , что поиск по шаблону должен быть нечувствительным к регистру символов, а флаг g- поиск должен быть глобальным. Пример:

/\bJava\b/gi

Методы класса String, поддерживающие регулярные выражения:

search(рег_выраж); возвращает позицию символа в первой найденной подстроке. Например “JavaScript”.search(/script/i) возвращает 4.

replace(рег_выраж, строка замены); - строка замены заменяет все строки с регулярным выражением если указан флаг g, иначе заменяется только первое найденное вхождение. Text.replace(/Javascript/gi,”JavaScript”). Метод match(рег_выраж) возвращает массив найденных по шаблону подстрок. Метод split(рег_выраж) – разбивает строку на массив строк.

Конструкция

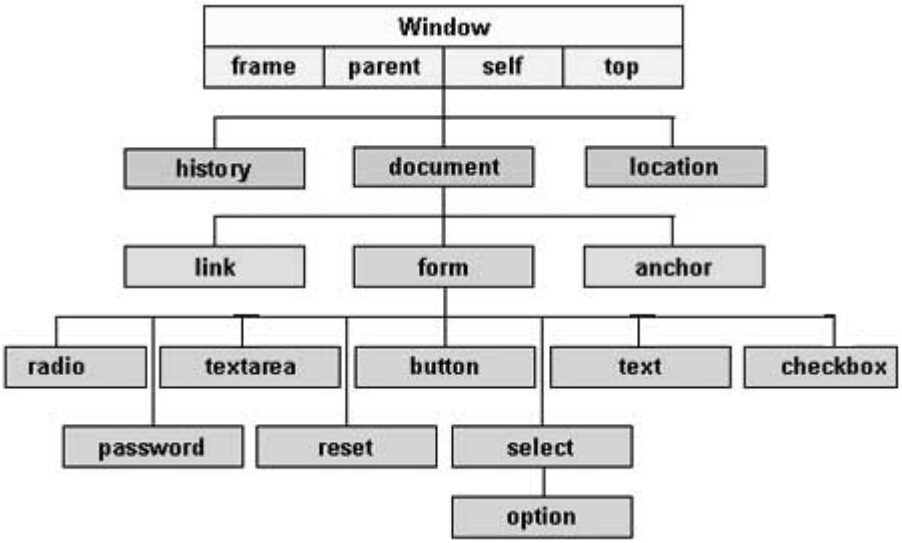
```
/MSIE (5\.5|6).+Win/.test(navigator.userAgent)
```

читается так: "проверить, соответствует ли строка, содержащаяся в свойстве navigator.userAgent, следующиему шаблону: Строка MSIE, после которой стоит пробел, затем наборы символов "5.5" или "6", после которых один или более символов, за которыми следует набор символов "Win"".

Регулярные выражения - очень мощное средство, но не лишенное недостатков. И главный из них - производительность. Разбор, компиляция и поиск в тексте по шаблону значительно более затратная операция, нежели простой поиск на точное соответствие. Поэтому не стоит впадать в крайности и использовать регулярные выражения для точного поиска.

Document Object Model (DOM)

Объектная модель JavaScript предоставляет возможность работы с объектами, зависящими от браузера. Ниже представлена схема объектов, которая позволяет манипулировать свойствами и структурой документа.



Принятые в 2000-2001 годах стандарты объектной модели документа называются DOM Level 1 DOM Level 2, и DOM Level 3. Модель DOM основана на использовании объектов браузера document, а также navigator, window, images, forms и JavaScript.

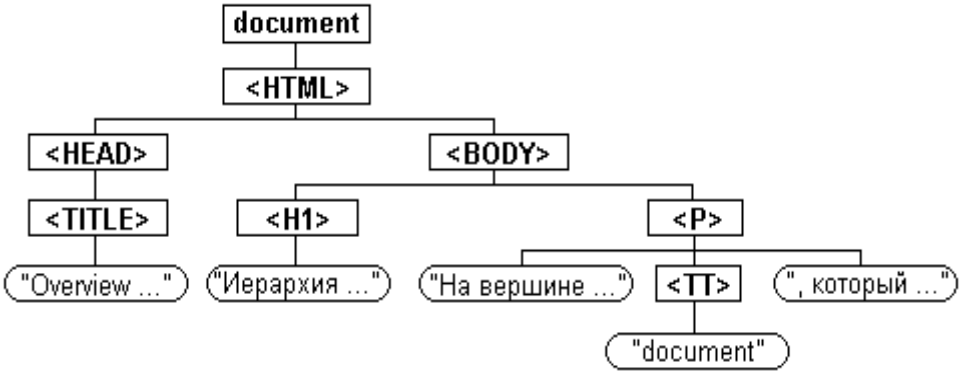
Объект браузера window.document содержит отображение загруженного HTML-документа в виде дерева узлов (node). Тег<html> становится «корнем» дерева. Дочерними узлами <html> являются узлы <head> и <body>, у которых, в свою очередь, есть собственные дочерние узлы. Рассмотрим фрагмент документа:

<!-- пример pr27: -->
<HTML>
<HEAD>
<TITLE> Overview of the DOM </TITLE>
</HEAD>
<BODY>
<H1> Иерархия узлов </H1>
<P>

На вершине иерархии находится узел document, который представляет в DOM сам документ и узел HTML.

</P>
</BODY>
</HTML>

На следующей диаграмме показаны узлы этого документа.



Элемент <html>доступен как свойство объекта `document. documentElement` и имеет тип Element. Кроме типа Element имеются также типы для атрибутов и текста.

Свойство nodeName возвращает имя HTML тэга, которому соответствует данный узел. Для атрибутов nodeName возвращает название атрибута, а для тестовых узлов возвращает #text.

Свойство nodeType возвращает 1, 2 или 3 для узлов, соответствующих тегу, атрибуту или тексту, соответственно.

Свойство: nodeValue хранит содержание текстового узла. Для элементов оно возвращает null, а для атрибутов - значение атрибута.

```
<!DOCTYPE HTML SYSTEM>
<html><head>
<title>Dom</title>
<script>window.onload = function() { alert(document.body.childNodes.length) }//5
</script>
</head>
<body>
<script>
 function go() {
alert(document.documentElement.nodeName+""+document.documentElement.
nodeType+""+ document.documentElement.nodeValueType)//1
alert(document.body.lastChild.nodeName+""+document.body.lastChild.nodeType+""+d
ocument.body.lastChild.nodeValue)    //3
                    alert(document.documentElement);//[object HTMLHTMLElement]
        alert(document.body);//[object HTMLBodyElement]

                }
    </script>
    <input type="button" onclick="go()" value="Go"/>
  </body> </html>
```

Коллекции

Объект document предоставляет доступ к отдельным элементам Web-страницы. Если элемент страницы имеет уникальное имя, заданноечерез id, то можно использовать обращение как к отдельному объекту:

image1.innerHTML или image1.outerHTML;

В остальных случаях обращение происходит через коллекции. Коллекция – это объект, который используется для хранения других объектов и отличается от ассоциативного массива наличием свойств и методов. Все коллекции имеют свойство length - количество элементов. Коллекция document.all содержит все используемые в документе элементы. Рассмотрим пример:

```
<!DOCTYPE HTML SYSTEM>
<html>  <head>
  <title>Документ</title>
        <script type=text/javascript>
function find()
{
var tag, tags="На странице теги:";
var n=document.all.length;
for(i=0;i<n;i++){
tag=document.all(i).tagName;
tags=tags+"\r"+tag;
}
alert(tags);}
</script>
   </head>
   <body onload = "find()">
<h1>DOM1</h1>
 <div id="footer">My production &copy;</div>
   </body> </html>
```

К элементам коллекции можно обращаться по номеру в HTML-коде или по имени, например document.all(8) или document.all("imge1"). Элементы коллекции нумеруются с нуля. Коллекция all имеет дополнительный метод tags, позволяющий фильтровать элементы коллекции по их тегу. Следующее выражение вернет ссылку на коллекцию, содержащую только заголовки первого уровня document.all.tags("H1")

Рассмотрим коллекцию images, которую включает объект document. Получить доступ к ее элементам можно по порядковому номеру или ключу.

```
document.images.item(1);
document.images.item("iamge1") ;
```

Индекс элемента коллекции помещается не в квадратных, а в круглых скобках, потому что он является аргументом метода item(), поддерживаемого всеми коллекциями. Имя этого метода можно опускать.

```
document.images(1);
document.images("image1");
<script type="text/javascript">
var someImage = document.images("image1");
</script>
```

Метод getElementByTagName возвращает коллекцию элементов, созданных посредством тега, переданного в качестве параметра. Пример:

```
document.getElementByName("someImage")
document.getElementByTagName("H1")
```

Свойства элемента innerHTML и outerHTML

Свойство innerHTML содержит HTML-код между открывающим и закрывающим тегами. С помощью этого свойства можно работать с кодом внутри тега, как со строкой. Отличие свойства outerHTML в том, что это свойство включает в себя не только HTML-код между открывающим и закрывающим тегом, а также открывающий и закрывающий теги. Свойство outerHTML доступно для записи только после того, как весь документ загружен (произойдет событие window.onload). Ниже приведен пример использования свойства outerHTML. Когда пользователь щелкает по кнопке передачи формы на сервер, вместо кнопки появляется сообщение с благодарностью:

```
<INPUT TYPE="submit" VALUE="Отправить" onClick= "this.outerHTML
```

```
='Благодарим Вас за участие в нашем опросе.'">
  Пример использования свойств innerHTML и outerHTML:
  <!-- пример pr31: использование свойств -->
  <html><head>
  <script type="text/javascript">
  function transformBody() {
        var myPar = document.getElementById("myP");
        myP.innerHTML = "<i>Hello, World!</i>";
        myP.outerHTML = "<strong>" + myP.innerHTML
  + "</strong>";
  }
  </script>
  </head>
  <body onload="transformBody();">
  <p id="myP">sample text</p>
  </body> </html>
  <!-- после выполнения функции структура элемента body будет:
  <BODY><STRONG><I>Hello, World!</I></STRONG></BODY>
   -->
```

Навигация по дереву документа

Для поиска узлов дерева можно применить обход дерева, а также функции поиска. Первый вариант требует наличия уже известного узла, например, document.documentElement, document.body (для доступа к тегу <body>), а также списки элементов document.forms или document.images.

Любой элемент имеет следующие ссылки на другие элементы:

- element.previousSibling, element.nextSibling: указатели на предыдущий и следующий соседние элементы. Если таких нет, в данных полях содержится null
- element.firstChild, element.lastChild: указатели на первого и последнего потомков. Если таких элементов нет, в данных полях содержится null
- element.parentNode: указатель на родителя. Так как предок отсутствует только у корня дерева, то этот указатель равен null только для document.documentElement.
- element.childNodes: массив указателей на всех потомков.
- Несколько других свойств узлов: attributes – список атрибутов; ownerDocument – указатель на объект document, которому принадлежит текущий узел.

Второй способ доступа к элементам,– это использование функций: element.getElementById(), element. getElementByTagName(), element. getElementsByname(). Поскольку эти функции определены не для всего документа, а элемента, то, поиск элементов выполняется среди потомков элемента, для которого функция вызывается.

Навигацию по дереву документа можно начинать с любого узла, для которого мы знаем идентификатор, присваиваемый ему в качестве значения атрибута ID. Ссылку на такой узел можно получить с помощью метода getElementById().

Приведем пример кода:

```
<DOCTYPE HTML SYSTEM>
<html><head><title></title>
<body>
<!-- пример pr28 -->
<p id="myP">Hello, World!</p>
<script type="text/javascript">
alert(document.getElementById("myP").innerHTML);
//выведет на экран фразу Hello World!
```

```
var str = "";
str += document.getElementById("myP").nodeName + '\n';
str += document.getElementById("myP").nodeValue + '\n';
str += document.getElementById("myP").nodeType + '\n';
alert(str);//выведет «P null 1»
 alert(document.getElementById("myP").childNodes[0].nodeValue);//Hello, World!
</script>
</body></head></html>
```

На экран вывелось nodeValue, равное null, потому что на самом деле внутри узла <p> есть еще один текстовый узел, в котором содержится искомый текст «Hello, World!». Для того чтобы получить доступ к этому значению, используется следующая строка:

```
document.getElementById("myP").childNodes[0].nodeValue;
```

Стартуя с некоторого узла, можно бродить по дереву, используя свойства узлов. Узлы-элементы и текстовые узлы имеют свойство parentNode, которое возвращает ссылку на родительский узел.

```
var oParent = oList.parentNode
```

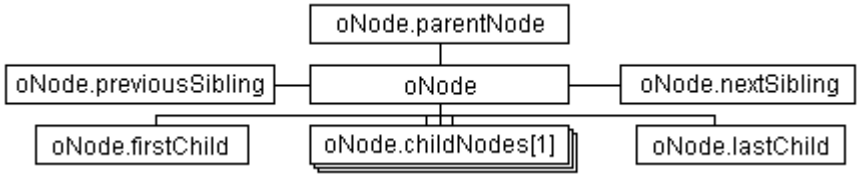
Узлы, являющиеся потомками, входят в состав коллекции childNodes этого узла. (Узлы-атрибуты составляют отдельную коллекцию attributes) К каждому из них можно обращаться по индексу массива. Например, строка кода

```
var oItem1 = oList.childNodes[1]
```

присваивает переменнойoItem1ссылку на элемент<LI>CSS</LI> нашего списка.

Ссылка на более удаленные узлы, как по горизонтали, так и по вертикали дерева формируется путем слияния ссылок на ближайших родственников по стандартным правилам объектно-ориентированного программирования. Так, oList.childNodes[1].firstChild является ссылкой на текст "CSS" элемента <LI>CSS</LI> нашего списка.

На следующей диаграмме приведены имена всех ближайших родственников некоторого узла oNode.



Заметим, что все описанные выше свойства узлов (parentNode, firstChild, lastChild, nextSibling и previousSibling), необходимые для навигации по дереву документа, являются свойствами только для чтения. Помимо них, узлы имеют еще ряд свойств, которые мы сейчас опишем.

Чтобы проиллюстрировать иерархию узлов DOM, приведем пример:

```
<DOCTYPE HTML>
<html><head><title></title>
</head>
<body>  <!-- пример 29: иерархия узлов DOM-->
<table>
<tr id="firstRow">
   <td id="firstCell"></td>
   <td id="currentNode" width="10">
<span id="spanNode">
text</span>
<p id="pNode">text</p>
</td>
```

```
    <td id="lastCell"></td>
  </tr>
  </table>
  <script type="text/javascript">
  alert(document.getElementById("currentNode").parentNode.id);
  //выведет на экран firstRow
  alert(document.getElementById(currentNode").childNodes[0].id);
  //выведет на экран spanNode
  alert(document.getElementById("currentNode").firstChild.id);
  //выведет на экран spanNode
  alert(document.getElementById("currentNode").lastChild.id);
  //выведет на экран pNode
  alert(document.getElementById("currentNode").previousSibling.id);
  //выведет на экран firstCell
  alert(document.getElementById("currentNode").nextSibling.id);
  //выведет на экран lastCell
  alert(document.getElementById("currentNode").attributes["width"].
  value);
  //выведет на экран 10
  alert(document.getElementById("currentNode").ownerDocument.
  nodeName);
  //выведет на экран #document
  </script>
  </body></html>
```

Приведем пример кода внутри тега BODY:

```
<!-- пример pr28 -->
<p id="myP">Hello, World!</p>
<script type="text/javascript">
alert(document.getElementById("myP").innerHTML);
//выведет на экран фразу Hello World!
var str = "";
str += document.getElementById("myP").nodeName + '\n';
str += document.getElementById("myP").nodeValue + '\n';
str += document.getElementById("myP").nodeType + '\n';
alert(str);//выведет «P null 1»
</script>
```

На экран вывелось nodevalue, равное null, потому что на самом деле внутри узла <p> есть еще один текстовый узел, в котором содержится искомый текст «Hello, World!». Для того чтобы получить доступ к этому значению, используется следующая строка:

```
document.getElementById("myP").childNodes[0].nodeValue;
```

Чтобы проиллюстрировать иерархию узлов DOM, приведем пример:

```
<!-- пример 29: иерархия узлов DOM-->
<table>
<tr id="firstRow">
        <td id="firstCell"></td>
        <td id="currentNode" width="10">
<span id="spanNode">
text</span>
<p id="pNode">text</p>
```

```
        </td>
                <td id="lastCell"></td>
        </tr>
        </table>
        <script type="text/javascript">
alert(document.getElementById("currentNode").parentNode.id);
        //выведет на экран firstRow
alert(document.getElementById(currentNode").childNodes[0].id);
        //выведет на экран spanNode
alert(document.getElementById("currentNode").firstChild.id);
        //выведет на экран spanNode
alert(document.getElementById("currentNode").lastChild.id);
        //выведет на экран pNode
alert(document.getElementById("currentNode").previousSibling.id);
        //выведет на экран firstCell
alert(document.getElementById("currentNode").nextSibling.id);
        //выведет на экран lastCell
alert(document.getElementById("currentNode").attributes["width"].
value);
        //выведет на экран 10
alert(document.getElementById("currentNode").ownerDocument.
nodeName);
        //выведет на экран #document
        </script>
```

Помимо метода getElementById(), существует несколько других, облегчающих доступ к необходимым элементам документа. Метод getElementsByName() возвращает коллекцию элементов с определенным атрибутом name, а метод getElementsByTagName() возвращает коллекцию элементов (тегов) с одинаковым именем. Оба метода принадлежат объекту document.

Создание новых узлов

DOM поддерживает следующие методы, связанные с созданием новых узлов:

createElement(tagName) – создает узел (тег) с именем, переданным в параметре;

createTextNode(string) – создает текстовый узел с содержанием, переданным в параметре.

createAttribute(name) – создает атрибут с именем, переданным в параметре;

createComment(string) – создает HTML-комментарий в виде <!--string -->, текст комментария передается в параметре;

createDocumentFragment() – создает новый документ для хранения новых узлов;

Создание новых элементов обсудим на примере. Предположим, что нужно добавить к существующему списку элемент: <LI>XML</LI>. Этому элементу в DOM соответствуют два узла: узел-элемент<LI>и текстовый узел"XML". Следовательно, нужно создать два новых узла с помощью методов createElement()и createTextNode() объекта document.

```
var oItem =     document.createElement("LI")
var oText =     document.createTextNode("XML")
```

Чтобы узел стал частью документа, его надо добавить к существующим узлам с помощью методов insertBefore(), appendChild() или replaceNode().

Метод appendChild(newChild) добавляет новый узел newChild после узла, который его активизировал. Например, строка кода:

oItem.appendChild(oText) добавляет узел oText к узлу oItem. Для добавления узлов в документ используются также метод:

insertBefore(newChild, referenceChild);

Теперь имеем в памяти элемент (веточку из двух узлов) <LI>XML</LI>

Пора вставлять эту веточку в текущий документ. Если мы хотим вставить ее в конец списка, то надо использовать метод appendChild():

oList.appendChild(oItem)

Поскольку узел oList, к которому мы присоединили узел oItem, является частью текущего документа, созданный нами элемент списка также становится частью документа. Теперь наш список выглядит так:

```
<UL ID="components">
<LI>HTML</LI>
<LI>CSS</LI>
<LI>Javascript</LI>
<LI>XML</LI>
</UL>
```

Обсудим теперь как можно вставить созданный нами элемент списка не в конец, а, скажем, после элемента списка<LI>HTML</LI>. Сделать это можно с помощью метода insertBefore(), который добавляет новый узел перед дочерним узлом, указанным в параметре referenceChild. В отличие от метода appendChild(), метод insertBefore() позволяет указать, в какое место коллекции childNodes будущего родительского узла будет вставлен новый узел. В качестве первого параметра метод insertBefore() принимает ссылку на узел, который мы хотим добавить, а в качестве второго параметра - ссылку на узел, перед которым будет вставлен новый узел. Второй параметр метода не является необязательным, если родительский узел не имеет деток. Итак, код

var oBrother = oList.firstChild.nextSiblingoList.insertBefore(oItem, oBrother)

добавляет в коллекцию childNodes узла oList узел oItem сразу после узла childNodes[0].

Теперь наш первоначальный список выглядит так:

```
<UL ID="components">
<LI>HTML</LI>
<LI>XML</LI>
<LI>CSS</LI>
<LI>Javascript</LI>
</UL>
```

Пример:

```
<!-- пример pr30: -->
<html>
 <head>
 <title>A Simple Page</title>
 <script type="text/javascript">
 function modify() {
  var newElem = document.createElement("p");
  newElem.id = "newP";
  var newText = document.createTextNode("This is the second paragraph.");
  newElem.appendChild(newText);
  document.body.appendChild(newElem);
 document.getElementById("emphasis1").childNodes[0].nodeValue = "first ";
 }
 </script>
```

```
</head>
<body>
<button onClick="modify()">Add/Replace Text</button>
<p id="paragraph1">This is the <em id="emphasis1">one and only</em> paragraph on the page.</p>
</body>
</html>
```

Удаление и замена узлов в документе

Метод replaceChild() позволяет у узла, который его активизирует, заменить одного из его деток на нового. Ссылку на новый и на заменяемый узлы метод принимает в качестве первого и второго параметров, соответственно. Так, следующий фрагмент сценария

```
var oItem =  document.createElement("LI")
oItem.appendChild(document.createTextNode("JScript"))
oList.replaceChild(oItem, oList.lastChild)
```

создает сначала элемент списка с текстом "JScript", а затем заменяет им последний элемент нашего списка. Метод возвращает ссылку на вставленный в документ узел.

Теперь наш список выглядит так:

```
<UL ID="components">
<LI>HTML</LI>
<LI>CSS</LI>
<LI>JScript</LI>
</UL>
```

Конечно, описанный выше пример надо рассматривать только как иллюстративный, поскольку тот же результат можно получить гораздо проще:

```
oList.lastChild.firstChild.nodeValue= "JScript"
```

Удаление: методы removeChild() и removeNode()

Метод removeChild() позволяет у узла удалить одного из его потомков. Ссылку на удаляемый узел метод принимает в качестве параметра. Например, строка кода

```
var oRemovedItem = oList.removeChild(oList.lastChild)
```

удаляет из списка последний элемент. Метод возвращает ссылку на удаляемый им узел. Поскольку удаленный из документа узел остается в памяти, мы можем в дальнейшем работать с ним, используя эту ссылку. Еще один метод заменяет узел oldChild на newChild.

```
currentNode.replaceChild(newChild, oldChild)
```

Если мы хотим удалить некоторый узел из документа, то надо воспользоваться методом removeNode(). Если значение параметра равно false, то удаляется только тот узел, который активизировал метод. При этом идущая от него ветвь дерева присоединяется к его родительскому узлу. Если параметр метода равен true, то узел удаляется вместе со своими потомками. Например, строка кода удаляет из документа весь список.

```
var oRemovedList = oList.removeNode(true)
```

Метод removeNode() возвращает ссылку на удаляемый узел. Поскольку удаленный из документа узел остается в памяти, мы можем в дальнейшем работать с ним.

Объект window

Объект window является контейнером других объектов JavaScript. Окно создается при запуске браузера или в коде JavaScript:

var winobj=window.open([параметры]);

Свойства объекта window.

Closed= true, если текущее окно закрыто; свойства defaultStatus, status – сообщение отображается в строке состояния окна; history – возвращает ссылку на объект истории браузера; innerHeight/innerWidth – возвращает высоту/ширину клиентской области окна; name – возвращает имя окна или фрейма; opener – возвращает ссылку на окно, которое открыло текущее окно, методом open; outerHeight/outerWidth – возвращает полную высоту / ширину окна в пикселах; pageXOffset/pageYOffset – расстояние по горизонтали/ по вертикали между позицией окна и левой границей документа

Свойства окна, передаваемые методу open()

alwaysLowered=yes|no. Если yes, то создаваемое окно будет всегда находиться под другими окнами.

alwaysRaised=yes|no. Если yes, то создаваемое окно будет всегда находиться над другими окнами фокуса.

dependent=yes|no. Если yes, то создаваемое окно будет дочерним по отношению к создавшему, т.е. при закрытии создающего окна будет закрываться и создаваемое.

fullscreen=yes|no. Если yes, то создаваемое окно займет весь экран (так называемый "режим киоска").

innerHeight={Высота}/innerWidth={Ширина}. Задает высоту/ширину клиентской области создаваемого окна в пикселах.

left={X}. Задает горизонтальную координату левого верхнего угла создаваемого окна.

location=yes|no. Включает или отключает отображение панели адреса, включающего строку ввода адреса, у создаваемого окна.

menubar=yes|no. Включает или отключает отображение строки меню.

outerHeight={Высота}. Задает полную (с рамками, строкой меню, полосами инструментов) высоту создаваемого окна в пикселах.

outerWidth={Ширина}. Задает полную ширину создаваемого окна в пикселах.

replace=yes|no. Если yes, то адрес документа, размещаемого в создаваемом окне, заменит в списке истории адрес документа, находящегося в создающем окне.

resizable=yes|no. Включает или отключает возможность изменения размера создаваемого окна.

screenX={X}/screenY={Y}. То же, что и left/ top

scrollbars=yes|no. Включает отображение полос прокрутки у создаваемого окна.

titlebar=yes|no. Включает или отключает отображение заголовка у создаваемого окна.

toolbar=yes|no. Включает/отключает отображение панели инструментов у создаваемого окна.

Вместо значений yes и no можно использовать 1 и 0.

Методы объекта window

Ранее были рассмотрены три метода объекта window для отображения диалоговых окон: alert({Текст})-окно предупреждения; confirm({Текст}) - окно выбора Ok/Cancel; prompt({Приглашение}, [{Значение по умолчанию}]) - диалоговое окно с полем ввода.

Ниже приведены другие методы объекта window: back()-возвращается к предыдущему документу; focus()/ blur()-переносит/ удаляет фокус на текущее окно; clearInterval({Таймер})- останавливает таймер, установленный setInterval(); clearTimeout({Таймер})-останавливает таймер, установленный setTimeout(); close()-закрывает текущее окно, открытое методом open(); execScript({Выражение},"JavaScript")-вычисляет выражение; find([{Строка поиска}[, true|false]) - второй аргумент равен true, если поиск с учетом регистра символов, иначе false; forward()-переходит к следующему

документу в списке истории; home() - переходит на "домашнюю" страницу, заданную в настройках браузера.
moveBy({X},{Y})-перемещает окно на X пикселов вправо и на Y пикселов вниз; moveTo({X},{Y})-перемещает окно в точку экрана, заданную координатами X и Y; navigate({Адрес}) - загружает Web-страницу; open({Адрес}, {Имя окна}, [{Список свойств окна}]) -открывает новое окно, загружает в него документ, адрес которого указан, и присваивает окну переданное имя; print()-печатает содержимое окна.
resizeBy({X},{Y})-увеличивает/уменьшает окно на X пикселов по горизонтали и на Y пикселов по вертикали; resizeTo({X},{Y})-увеличивает или уменьшает окно до размера, заданного X и Y; scrollBy({X},{Y}) - прокручивает содержимое окна на X пикселов вправо и Y пикселов вниз; scrollTo({X},{Y}) - прокручивает содержимое окна в точку, заданную значениями X и Y; setHotKEys(true|false) - разрешает (true) или запрещает (false) "горячие" клавиши в окне, не имеющем строки меню; setInterval({Функции или выражение},{Интервал}, [{Список аргументов функции, разделенных запятыми}]) - вычисляет значение выражения каждый раз по истечении заданного интервала (в миллисекундах). Возвращает указатель на объект таймер, который можно использовать в методе clearInterval. Функция setResizeeable(true|false) разрешает (true) или запрещает (false) пользователю изменять размеры окна. setTimeout({Функции или выражение},{Интервал}, [{Список аргументов функции, разделенных запятыми}]) - вычисляет значение выражения по истечении заданного интервала (в миллисекундах), если до этого не был вызван метод clearTimeout. Возвращает указатель на объект, который можно использовать в методе clearTimeout для уничтожения таймера. stop() - останавливает загрузку текущей страницы.

Все методы вызываются одним из следующих способов:
window. метод(параметры);
self. метод(параметры);
winobj. метод(параметры);

В следующем примере рассмотрим использование метода window.open() для открытия минимизированного окна.

```
<DOCTYPE html>
<!-- пример pr20: минимизация окна -->
<html>
 <head>
 <title>window.closed Property</title>
 <script type="text/javascript">
  var newWind; // новое окно
 function newWindow() {
 newWind = window.open("","subwindow","height=100,width=50");
  setTimeout("fNewWindow()", 100);
 }
 function fNewWindow() {
var str = "";
str += "<html><body><h1>Порожденное окно</h1>";
str +="</body></html>";
  newWind.document.write(str);
  newWind.document.close();
 }
  function closeWindow() { // close subwindow
  if (newWind && !newWind.closed) {
  newWind.close();
```

```
 }
 }
 </script>
 </head>
 <body>
 <form>
<input type="button" value="ОткрытьПорожденноеокно" onclick="newWindow()" />
 <br />
<input type="button" value="ЗакрытьПорожденноеокно" onclick="closeWindow()" />
 </form>
 </body> </html>

 <!-- пример pr21: открытие документа в новом окне -->
 <html>
 <head>
 <title> Окно#1 HTML </title>
 <script type="text/javascript">
{//Первая функция открывает в новом окне существующий документ
 function openStaticWin()
window.open("test21.htm", "_blank",
"height=500, width=600,status=yes,location=no, resizable=yes ");
 document.bgColor ='ff0000'}
 function openDynamicWin() { // задает HTML-код нового документа динамически.
 var newWin = window.open();
 newWin.document.open();
 newWin.document.write("<html><head></head><body>"
 + new Date() + "</body></html>");
 newWin.document.close();
 }
 </script>
 </head>
 <body>
 <form name="form2">
 <input type="button" name="button1" value="openStaticWin"
 onclick="openStaticWin();">
 <input type="button" name="button2" value="openDynamicWin"
 onclick="openDynamicWin();">
 </form>
 </body></html>

 <!-- пример test # 21: шаблон HTML для скрипта -->
 <html>
 <head>
 <title> Test Window Окно#2 </title>
 <form name=form1>
 <input type="button" name="button3" value="close"
 onclick="window.close();">
 </form>
```

```
</head>
<body >
</body></html>

<!-- пример pr22: адрес каталога из которого загружен документ -->
<html>
<head>
<title>Extract pathname</title>
<script type="text/javascript">
// function to extract URL of current directory
function getDirPath(URL) {
var result=unecape(URL.substring(0,(URL.lastIndexOf("/"))+ 1));
 return result;
}
// passing work onto general purpose function
function showDirPath(URL) {
 alert(getDirPath(URL));
}
</script>
</head>
<body>
<form>
 <input type="button" value="View directory URL"
 onclick="showDirPath(window.location.href)" />
</form>
</body>
</html>
```

Свойства и методы объекта navigator

Объект window.navigator служит для доступа к Web-браузеру. Свойство appName - возвращает имя, appVersion -возвращает версию, browserLanguage - возвращает код браузера. Свойство cookieEnabled возвращает true, если браузеру разрешен прием cookie. cpuClass - возвращает класс процессора. Language - возвращает код языка браузера. onLine - возвращает true, если клиент находится в режиме on-line, иначе false. Platform - Возвращает название клиентской платформы. userAgent -возвращает строку, идентифицирующую клиента. appCodeName-возвращает имя кода браузера.

```
<html> <head>
      <title>JavaScript </title>
<meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
      <script type="text/javascript">
         function display() {
          window.onerror=null;
          var name = navigator.appName;
          var ver= navigator.appVersion;
          var codname = navigator.appCodeName;
          var userag= navigator.userAgent;
          var platform = navigator.platform;
alert("name "+name+" ver "+ver+" codnam "+codname+" userag "
+ userag+" platf "+platform);
          }
```

```
    </script>
  </head>
  <body >
<form>
   <input type=button name="again" style="width: 20%" value=
"Получить данные" onClick="display()">
      </form>
  </body> </html>
```

Результат:

name Netscape ver 5.0 (Windows) codnam Mozilla userag Mozilla/5.0 (Windows NT 5.1; rv:19.0) Gecko/20100101 Firefox/19.0 platf Win32

Свойства и методы объекта history

Объект window.history представляет интерфейс к списку всех Web-страниц, просмотренных пользователем в течение времени, указанного в настройках.

Свойство current - возвращает интернет-адрес документа, загруженного в настоящее время; length - возвращает размер списка истории; next - возвращает интернет-адрес следующего в списке истории документа; previous - возвращает интернет-адрес предыдущего в списке истории документа.

Метод back() - загружает в окно браузера предыдущий документ из списка истории; forward() - загружает в окно браузера следующий документ из списка истории; go({Адрес}) - загружает в окно браузера следующий документ из списка истории, адрес которого наиболее близок к переданному в качестве параметра; go({Позиция}) - перемещается в списке истории на указанную позицию.

```
<!-- пример pr23: -->
<html>
 <head>
 <title>History </title>
 <script type="text/javascript">
 function showCount() {
  var histCount = window.history.length;
 alert("You have been " + histCount + " Web pages this session.");
  var hist= window.history. previous;
  alert("Предыдущий документ"+hist);
 }
 </script>
 </head>
 <body>
 <form>
  <input type="button" name="activity" value="My Activity"  onclick="showCount()"
/>
 </form>
 </body>
</html>
```

Свойства и методы объекта location

Объект location содержит интернет-адрес текущего документа. Его также можно использовать для перехода на другой документ и перезагрузки текущего документа.

Свойство	Описание
Hash	Имя "якоря" в интернет-адресе документа, если оно есть.

Host	Имя компьютера в сети интернет вместе с номером порта.
hostname	Имя компьютера в сети Интернет.
Href	Полный интернет-адрес документа.
pathname	Путь и имя файла, если они есть.
Port	Номер порта. Если не указан, возвращает номер 80 - стандартный.
protocol	Идентификатор протокола. Если не указан, возвращается "http:".
search	Строка параметров, если она есть.

Метод	Описание
assign({Адрес})	Загружает документ, адрес которого передан в качестве параметра.
reload([true|false])	Перезагружает документ с Web-сервера.
replace({Адрес})	Загружает документ, адрес которого передан в качестве параметра, и заменяет в списке истории Web-обозревателя адрес предыдущего документа адресом нового.

Пользуясь объектом location, можно загрузить другой документ на место текущего. Для этого надо присвоить значение нового интернет-адреса свойству href: document.location.href = "http://bsu.by";

Если надо полностью заменить текущий документ, чтобы даже адрес его не появлялся в списке истории, то:

document.location.replace("http:// bsu.by");

Использование каскадных таблиц стилей и объекта style

Каждый узел DOM имеет объект style, который описывает стили. Можно изменить, например, цвет шрифта элемента "simpl", document.getElementById("simpl").style.color = red.

Объект style поддерживает ряд свойств, которые можно разделить на две группы: задающие стиль документа и относящиеся к самому объекту style. Свойства первой группы аналогичны соответствующим атрибутам стиля и имеют похожие имена. Символы "-" в именах убираются, как не соответстующие соглашению JavaScript. Первые буквы второго, третьего слов, имени атрибута, делаются прописными. Примеры: background-attachment-> backgroundAttachment;border-bottom-color -> borderBottomColor; font-family ->fontFamily; z-index ->zIndex.

По аналогии можно преобразовать все атрибуты стилей в свойства объекта style. Все не относящееся к стилю свойства объекта style перечислены в таблице.

Свойство	Описание
cssText	текстовое представление стиля (параметр атрибута STYLE).
pixelHeight	Высота элемента в пикселах.
pixelLeft	Смещение левого края элемента в пикселах.
pixelTop	Смещение верхнего края элемента в пикселах.
pixelWidth	Ширина элемента в пикселах.
posHeight	Высота элемента в единицах, в которых она была установлена в определении стиля.
posLeft	Смещение левого края.
posTop	Смещение верхнего края.

posWidth	Ширина элемента.

Объект style позволяет изменить геометрические размеры элемента и его месторасположение, например:

```
paragraph1.style.fontSize = 7;
image1.style.height = "100mm";
image1.style.width = "120mm";
iamge1.style.top = "200px";
image1.style.left = "50px";
```

Можно использовать также свойства pos*, принимающие числовые значения в тех единицах, в которых они были заданы в определении стиля.

```
<IMG src="img1.gif" id="img1" style="height: 100mm; width: 100">
nheight = img1.style.posHeight; // Значение в миллиметрах
nwidth = img1.style.posWidth; // Значение в пикселах
```

JavaScript и AJAX

Технология AJAX (Asynchronous JavaScript And XML) позволяет создавать интерактивные веб-приложения, в которых пересылаемая сервером клиенту страница может полностью не перегружаться, а перегружается лишь часть, содержащая изменившиеся данные. Все новые сервисы Google, в том числе GMail, Orkut, Google Groups, Google Maps, Google Suggest, Google Finance, являются AJAX-приложениями.

Базисом технологии являются: использование модели DOM для обновления содержимого, обмен и обработка данных в виде XML, асинхронные запросы к серверу через интерфейс XMLHttpRequest, использование JavaScript.

Рассмотрим отличия классической модели веб-приложения от AJAX.

Классическая модель веб-приложения:

Пользователь загружает в браузер веб-страницу и вызывает событие.
Браузер отправляет HTTP запрос серверу.
Сервер генерирует страницу - ответ и возвращает ее браузеру.
В браузере отображается новая страница.

Модель AJAX:

Между загруженной в браузер страницей и сервером появляется еще одна прослойка - уровень AJAX.

Пользователь загружает web-страницу и генерирует событие.

Скрипт определяет, какая информация необходима для обновления страницы и передает ее уровню AJAX.

AJAX отправляет запрос на сервер.

Сервер возвращает уровню AJAX только ту часть документа, на которую пришел запрос или только данные в формате XML.

Уровень AJAX вызывает код на языке JavaScript, который вносит изменения на страницу с использованием методов DOM без полной перезагрузки. Для передачи данных обычно используется XML файл, который формируется динамически.

Класс XMLHttpRequest

Для запроса к серверу в модели Ajax используется класс XMLHttpRequest, который впервые был реализован компанией Microsoft в браузере IE 5.0 в виде объекта ActiveX, доступного через JScript. Программисты проекта Mozilla разработали свою версию класса XMLHttpRequest, в дальнейшем также реализованную компаниями Apple, Opera и другими. Класс XMLHttpRequest входит в набор API (XMLHTTP), используемый в JavaScript, для пересылки по HTTP-протоколу документов XML, данных формы или текстовых строк. Объект-запрос XMLHttpRequest выполняет функцию HTTP запроса.

Свойства класса

Свойство onreadystatechange - обработчик события при смене состояния объекта.

readyState - возвращает состояние объекта (0 — не инициализирован, 1 — открыт, 2 — отправка, 3 — получение данных, 4 — данные загружены).

responseText - текст ответа на запрос;

responseXML - текст ответа на запрос в виде XML, который затем может быть распарсен посредством DOM;

status - возвращает HTTP-статус в виде числа (404 — «Not Found», 200 — «OK»);

statusText - возвращает статус в виде строки («Not Found», «OK»)

Методы класса

open(method, URL, async, userName, password) – открыть соединение. Параметры: GET/POST method, URL, имя и пароль запроса; async=true/false - определяет, происходит ли работа в асинхронном режиме;

abort() - отменяет текущий запрос;

send(content) - отправляет запрос на сервер;

getAllResponseHeaders() - возвращает список всех HTTP-заголовков в виде строки; getResponseHeader(headerName) - возвращает значение указанного заголовка; setRequestHeader(label, value) - добавляет HTTP-заголовок к запросу.

Создание экземпляра объекта

```
if (window.XMLHttpRequest) {
req = new XMLHttpRequest();//создание экземпляра
req.onreadystatechange = processReqChange;//обработчик событий
 req.open("GET", url, true);// открытие соединения
 req.send(null); }// Отправка запросов.
```

В методе open() параметр async – флаг асинхронного выполнения. Если async= true, метод send() сразу вернет управление сценарию. В этом случае пользователь сможет продолжить работу с документом, пока запрос будет пересылаться к серверу и обратно. При получении ответа от сервера, управление перейдет к функции обратного вызова, задаваемую свойством onreadystatechange. С помощью свойства readyState проверяется состояние процесса обработки запроса (readyState = 4 окончание), данные извлекаются из свойств responseText или responseXML.

```
function processReqChange() {
        try { // Важно!
if (req.readyState == 4) {  // действия для статуса "OK"
 if (req.status == 200) {// здесь действия с полученным ответом
}
else {
  alert("Не удалось получить данные:\n" +
  req.statusText);  }
          }
 }
 catch( e ) { }
}
```

После определения всех параметров запроса его отправляют функцией send(). При отправке GET-запроса вызывают метод req.send(null) в котором указывают параметр null. Чтобы передать на сервер POST-данные, их надо передать в качестве параметра для этой функции. POST-данные должны быть свернуты в URL-escaped строку в кодировка UTF-8 и добавлен заголовок req.setRequestHeader("Content-Type","application/x-www-form-urlencoded");.

После этого начинает работать вышеуказанный обработчик событий. Он — фактически основная часть программы. В обработчике обычно происходит перехват всех возможных кодов состояния запроса и вызов соответствующих действий, а также перехват ошибок. Пример кода:

Приведем код всего AJAX-приложения:

```
//JavaScript’овая часть:
var req;
var reqTimeout;
function loadXMLDoc(url) {
 req = null;
 if (window.XMLHttpRequest) {
 try {
  req = new XMLHttpRequest();
 } catch (e){}
 }
 }
  if (req) {
 req.onreadystatechange = processReqChange;
 req.open("GET", url, true);
 req.send(null);
 reqTimeout = setTimeout("req.abort();", 5000);
 } else {
 alert("Браузер не поддерживает AJAX");
 }
}
function processReqChange() {
 document.form1.state.value = stat(req.readyState);
  if (req.readyState == 4) {
 clearTimeout(reqTimeout);
  document.form1.statusnum.value = req.status;
 document.form1.status.value = req.statusText;
  // only if "OK"
 if (req.status == 200) {
  document.form1.response.value=req.responseText;
 } else {
  alert("Не удалось получить данные:\n" + req.statusText);
 }
 }
}
function stat(n){
 switch (n) {
 case 0: return "не инициализирован"; break;
 case 1:  return "загрузка..."; break;
 case 2:  return "загружено"; break;
 case 3:  return "в процессе..."; break;
 case 4:  return "готово"; break;
 default: return "неизвестное состояние";  }
}
```

```
function requestdata(params)
{
 loadXMLDoc('examples/httpreq.php?'+params);
}
```

HTML-форма:

```
<form name=form1>
<table width=100% style="font-size: 100%">
<tr>
<td width=30% valign=top>Состояние запроса</td>
<td width=70%><input size=25 disabled type=text name=state value=""></td>
</tr>
<tr>
<td valign=top>Код статуса</td>
<td><input disabled size=2 type=text name=statusnum value="">
<input disabled size=19 type=text name=status value=""></td>
</tr>
<tr>
<td valign=top>Данные от сервера</td>
<td><textarea rows=6 name=response></textarea></td>
</tr>
<tr>
<td>Строка GET-запроса<td>
<td></td>
</tr>
</table>
<input type=text name=getparams value="?">
<input type=button onclick="requestdata(getparams.value);" value="GET">
</form>
```

PHP-файл:

```
<?php
header("Content-type: text/plain; charset=windows-1251");
header("Cache-Control: no-store, no-cache, must-revalidate");
header("Cache-Control: post-check=0, pre-check=0", false);
echo "Hello world!\n\n";
if (isset($a))
{
 for ($i=1; $i < 10000; $i++)
 {
 echo 'Это тестовая строка. ';
 if (($i % 1000) == 0) flush();
 }
}
if (count($_GET) > 0)
{
 echo "\n\nПередано GET\n"; print_r($_GET);
}
?>
```

Использование DOM

Полученные новые данные нужно включить в состав документа, уже отображающегося на экране. Для этого надо воспользоваться структурой объектов DOM. От свойства responseText лучше отказаться, так как его применение потребует дополнительного синтаксического разбора. Гораздо полезнее свойство responseXML, в котором информация изначально представляется в совместимом с DOM виде. В структуре текущего документа необходимо выбрать узел, в состав которого следует включить возвращенные данные, и вызвать метод replaceChild(), заменив старое поддерево новым, прочитанным из responseXML объекта XMLHttpRequest. После этого осталось задать внешний вид полученных данных посредством каскадных таблиц стилей.

Выполнив упомянутые выше действия, мы получим Ajax-приложение.

Важно, чтобы цикл связи Ajax не прерывался. Серверный скрипт должен возвращать корректный XML документ. Если серверный скрипт завершится аварийно, сообщение об ошибке будет утеряно и приложение упадет.

При создании клиентских скриптов нельзя забывать о пользователях, у которых скрипты отключены. Это можно сделать с помощью использования «захвата» ссылки или формы.

<form action="traditional_action.php" method="post"
onsubmit="return perform_ajax_action();">

Если все пройдет хорошо, будет выполнен код Ajax, а подтверждение формы пропущено. При наличии проблем с поддержкой Ajax, форма будет подтверждена, и пользователь сможет продолжить работу обычным образом.

Часто выбор класса для объекта req оформляется в виде функции. Рассмотрим пример:

```
//пишем функцию, использующую этот объект
var req;
function loadXMLDoc(url) {
 // branch for native XMLHttpRequest object
 if (window.XMLHttpRequest) {
 req = new XMLHttpRequest();
 req.onreadystatechange = processReqChange;
 req.open("GET", url, true);
 req.send(null); }
if (req) {
  req.onreadystatechange = processReqChange;
  req.open("GET", url, true); //открыть запрос
  req.send();  } //послать запрос
}
}
//В теле HTML файла пишем скрипт:
function checkName(input, response)
{
 if (response != ''){
 // Response mode
 message = document.getElementById('nameCheckFailed');
 if (response == '1'){
 message.className = 'error';
 }else{
 message.className = 'hidden';
```

```
}
}else{
// Input mode
url = 'http://localhost/xml/checkUserName.php?q=' \\
+ input;
loadXMLDoc(url);
}
}
```

В данном примере в файле localhost/xml/checkUserName.php обрабатываются данные, полученные из командной строки в переменной q. А результат сохраняется в XML структуре, которая хранится в этом же файле. Так можно получить из БД и обработать данные, которые необходимо обновить.

Объект `FormData`

`FormData` позволяет создать форму HTML на лету, используя JavaScript, и затем переслать ее, используя `XMLHttpRequest.send()`. Пример:

```
var formData = new FormData();
formData.append("part_num", "123ABC");
formData.append("part_price", 7.95);
formData.append("part_image", somefile)
var xhr = new XMLHttpRequest();
xhr.open("POST", "http://some.url/");
xhr.send(formData);
```

You can also use `FormData` to add additional data to an existing form before submitting it.

```
var formElement = document.getElementById("someFormElement");
var formData = new FormData(formElement);
formData.append("part_description", "The best part ever!");
var xhr = new XMLHttpRequest();
xhr.open("POST", "http://some.url/");
xhr.send(formData);
```

JQuery

jQuery — это библиотека (фреймворк) JavaScript, облегчающая работу с DOM, CSS, AJAX и визуальными приложениями. Это бесплатное и открытое ПО, созданное Джоном Резигом (John Resig) в 2006 году. Функциональность обеспечивается файлом JQuery, который можно загрузить с сайта jQuery.com. В скачивании файла в свой каталог js/ и, затем, в подключении состоит первый способ загрузки:

```
<script type="text/javascript" src="js/jquery.js"></script>
```

Во втором способе загрузка происходит в онлайн режиме. В этом случае файл можно не скачивать, а подгрузить с сайта jQuery.com по сервису загрузки контента CDN

```
<script type="text/javascript"  src="http://code.jquery.com/jquery.js"></script>
```

Рассмотрим пример. Допустим, мы хотим изменить цвет заголовка h1 на сайте

```
<!doctype html>
<html>
  <head>
```

```
        <meta charset="utf-8">
        <title>prjq1 -Demo</title>
    <script type="text/javascript" src="js/jquery-1.8.3.js">
</script>
    </head>
      <body>
        <a href="http://jquery.com/">jQuery</a>
        <h1> Использование jQuery</h1>
        <script>
         jQuery(document).ready(function(){
    jQuery("h1").css("color", "green");//цвет h1
    });//меняет цвет H1 на зеленый
        </script>
      </body>
    </html>
```

Основная функция jQuery() библиотеки находит ссылку на элемент по строке описания элемента. Функция `$()` позволяет обращаться к элементам DOM по их естественным именам и является синонимом `JQuery()`. Вызов функции `$(document).ready()` означает, что функция будет дожидаться окончания загрузки страницы клиента (onload). Затем она выполнит анонимную функцию, содержащую обращение к jQuery. Обращаем внимание на то, что аргументы функций в jQuery() берутся в ковычки. Это позволяет осуществлять поиск элементов по правилам регулярных выражений.

В следующем примере находятся элементы <p>подчеркиваются, текст с id=’idname’ выводится в окно, изменяется цвет шрифта в<body>:

```
<!DOCTYPE html>
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>prjq1 - Jquery</title>
<script type="text/javascript" src="js/jquery-1.8.3.js"> </script>
<style type="text/css">
body{color: green;}
p{text-align:center;}
h1{text-align:center;}
</style>
<script type="text/javascript">
 jQuery(document).ready(function(){// ждём загрузки документа,
jQuery("h1").css("color", "yellow");
$("p").after("<hr/>") ;
alert($("p"));
alert($('#idname').text());
});
</script >
 </head>
<body > <!--Это тело документа-->
  <h1>Заголовок страницы</h1>
<p >Основной текст.</p>
<div id='idname'>Это  div  </div>
 <a href="http://jquery.com">познавательно</a>
<a href="http://jsapi.com">познавательно тоже</a>
```

</body> </html>

Чтобы код сработал верно, нужно поместить его в низ страницы после <h2> либо вызывать после загрузки документа. С этой целью используется функция jQuery(document).ready() для ожидания конца загрузки документа (событие«onload»).

Сокращением для последней функции является функция Query();

Доступ к элементам DOM

Одной из основных областей применения JQuery является (DOM). Функция JQuery() (сокращение $()) возвращает массив элементов, выбранных из документа на основе параметров. Рассмотрим несколько вариантов: $('tagname') — доступ к тегу. Например $('span') возвратит все ссылки на теги ; $('*') — доступ ко всем элементам документа. Функция JQuery позволяет также выбирать элементы по «id» либо «className»: $('#idname') — доступ к элементу с id='idname'; $('.classname') — доступ к элементу по имени класса class=' classname'; Например $(".bebe") - выбираем элементы с class=bebe, $("div.wrap") - выбираем div с class=wrap. Выбираемые элементы можно перечислить через запятую: $('img, .myclass') получит все элементы <img>, а также элементы с классом myclass. Однако $('img#myPort') получит элемент . Кроме этого можно указывать: `$('element:first')`—первый элемент, `$('element:last')`—последний элемент. `$('element:even')`— чётные элементы, `$('element:odd')`— нечётные элементы. `$("div.myclass")- div` имеющий `class="myclass", $("#my")-элемент с id="my", $("input:text")-элементы input с type="text".$("input:checkbox:checked")-все чекбоксы с checked=true.`

Другие методы библиотеки jQuery, вызываемые через точку, имеют приклеивающийся функционал, т.е. применяются к объекту, после которого были вызваны и возвращают новый или модифицированный массив. Рассмотрим пример:

```
$("div") //вернул все div
.not('.red') //отсек div с классом red и вернул остальные
.add('span.green') //добавил к выборке span'ы с классом green
.addClass('mycl')//добавил класс mycl каждому элементу.
```

$("div").filter(".red, :first") – все дивы, из них первый с классом red.

Методы выполняются в порядке вызова слева направо. Рассмотрим пример перемещения по DOM дереву, соответствующему таблице:

```
<!DOCTYPE html>
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>prjq3 - "Элементы JQuery</title>
<script type="text/javascript" src="js/jquery-1.8.3.js"> </script>
<style type="text/css">
body{color: green;}
h1{text-align:center;}
</style>
</head>
<body > <!--Это тело документа-->
 <h1>Заголовок страницы</h1>
<div id='idname'>Это второй div </div>
<table width="75%" border="1" >
<tr>
<td>здесь текст</td>
```

```
<td>этот текст будет удален</td>
<td>а этот нет</td>
<td><a id="myid">del</a></td>
</tr>
</table>
<script>
$("#myid").click(function (){
$(this).parent().prev().prev().empty();
});
</script>
</body></html>
```

В примере использовались функции для родственных связей: $("p").parent() - выбор всех прямых предков элементов p, $("p").parents("div") - выбор всех предков элемента p, которые есть div. В поиске элементов в документе заключается большая часть работы с jQuery.

Управление атрибутами

Ещё одна полезная возможность JQuery — управление атрибутами тегов, такими как alt, src, href, width, style, title, class и др. Найденному элементу можно задать, удалить, получить класс, или любой атрибут. Методы, которые позволяют получить информацию об элементе, называют getters, а методы, позволяющие установить свойства - setters. Аргументы методов берутся в скобки.

Функция `.attr(имя_атрибута)` — возвращает значение атрибута (getters),
`.attr(имя_атрибута, значение_атрибута)` — устанавливает новое значение атрибута (setters).
`.removeAttr(имя_атрибута)` — удаляет атрибут
`.addClass(имя_класса)` — добавляет класс
`.removeClass(имя_класса)` — удаляет
`.html()` — возвращает HTML-код. Например, $('h1').html() - getters
`.html(код)` — устанавливает HTML-код. $('h1').html('hello world') - setters
`.val()` — возвращает значение элемента, `.val(значение)` — устанавливает значение элемента.

Рассмотрим пример поиска по документу: выбираем все теги h2 которые являются непосредственными потомками тега article // выбор всех потомков элементов p

```
<!DOCTYPE html>
<html ><head>
<meta charset="UTF-8"/>
<title>prjq4 - "Элементы JQuery</title>
<script type="text/javascript" src="js/jquery-1.8.3.js">  </script>
<style type="text/css">
body{color: green;}
p{text-align:center;}
h1{text-align:center;}
</style>
 </head>
<body  >
 <div id='idname'>Это второй div  </div>
<div id="content" class="wrapper box">
<hgroup><h1>Page Title</h1> <h2>Page Description</h2> </hgroup>
<article id="stick"> <h2>Article Title</h2>
<p>Заголовок статьи</p> </article>
```

```
<article> <h2>Article Description</h2>
<p>Описание текста</p> </article>
</div>
<script>
var a=$("article").find("h2").html(); // // выбираем теги h2 - потомки тега article
alert("h2.html=" +a);
$("div").find("article").find("h2") //
$("article > h2").html("NEWHTML");
 var a= $("article").children().css('fontsize', '100px'); //
alert(a);
$("#stick").prev() // выбор предыдущего элемента от найденного
$("#stick").next() // выбор следующего элемента от найденного
</script>
<footer> <p>&copy;copyright 2013</p> </footer>
</body> </html>
```

Кроме атрибутов, у элементов имеются также свойства, такие как selectedlndex, tagName, nodeName, nodeType, ownerDocument, defaultChecked и defaultSelected. Для работы со свойствами используем функции prop():prop(propName) — получение значения свойства, prop(propName, propValue) — установка значения свойства. removeProp(propName) — удаление свойства. Для отключения элементов формы и для изменения состояния чекбоксов используется функция pnop().

Манипуляции с DOM

Метод after(content) — вставляет контент после каждого элемента из выборки, например $("p").after("<hr/>") - после каждого параграфа вставит линию.

insertafter(etement) — вставляет элементы из выборки после каждого элемента переданного в качестве аргумента. Строка $("<hr/>").insertafter("p") - означает «линия будет вставлена после каждого параграфа». before(content) — вставляет контент перед каждым элементом из выборки. insertBefore(etement) — вставляет элементы из выборки перед каждым элементом переданным в качестве аргумента. append(content) — вставляет контент в конец каждого элемента из выборки, т.е. строку кода $("p").append("<hr/>"), следует читать как «в конец каждого параграфа будет добавлена линия»

appendTo(element) — вставляет выбранный контент в конец каждого элемента переданного в качестве аргумента: $("<hr/>").appendTo("p") — «линия будет добавлена в конец каждого параграфа»

prepend(content) — вставляет контент в начало каждого элемента из выборки

prependTo(element) — вставляет выбранный контент в начало каждого элемента переданного в качестве аргумента

replaceWith(content) - заменяет найденные элементы новым содержимым, replaceAll(target) - вставляет контент в замен найденному.

wrap(element) - оборачиваем каждый найденный элемент новым элементом, wrapAll(element) - оборачивает найденные элементы новым элементом.wraplnner(element) - оборачивает контент каждого найденного элемента новым элементом,

unwrap() - удаляет родительский элемент у найденных элементов

clone(withDataAndEvents) - клонирует выбранные элементы, для дальнейшей вставки копий назад в DOM, позволяет так же копировать и обработчики событий

detach() - удаляет элемент из DOM, но при этом сохраняет все данные о нём в jQuery,

empty() - удаляет текст и дочерние DOM элементы, remove() - удаляет элемент из DOM, насовсем.

text() - вернёт текст заданного элемента, text(newText) - заменит текст внутри выбранных элементов.

```
<!DOCTYPE html>
<html ><head>
<meta charset="UTF-8"/>
<title>prjq41 - "Элементы JQuery</title>
<script type="text/javascript" src="js/jquery-1.8.3.js">  </script>
<style type="text/css">
body{color: green;}
h1{text-align:center;}
</style>
 </head>
<body  >
 <div id='idname'><a href="http://jquery.com">познавательно</a>
<a href="http://jsapi.com">познавательно тоже</a></div>
<div id="content" class="wrapper box">
<hgroup><h1>Page Title</h1> <h2>Page Description</h2> </hgroup>
<article id="stick"> <h2>Article Title</h2>
<p>Заголовок статьи</p> </article>
<article> <h2>Article Description</h2>
<p>Описание текста</p> </article>
</div>
<div id="cont"><h3 >Здесь контент</h3> </div>
<script>
alert("Hello1" );
var $myNewElement = $('<p>New element</p>');
$myNewElement.appendTo('#content');
$('#idname a:first').attr('href', 'prjq1.htm');
// Manipulating multiple attributes
$('#idname a:last').attr({
href : 'prjq2.htm' , rel : 'super-special'})
</script>
</body> </html>
```

Атрибуты и CSS

Начнём с изучения метода css(): css(property) — получение значения CSS свойства; css(property, value) — установка значения CSS свойства; css({key: value, key:value}) — установка нескольких значений; css(property, function(index, value) { return value }) ;— для установки значения используется функция обратного вызова; index порядковый номер элемента в выборке; value — старое значение свойства (callback-функция)

Метод css() возвращает текущее значение, а не прописанное в CSS файле. Примеры использования:

```
$("#my").css('color')  // получаем значение цвета шрифта
$("#my").css('color', 'red') // устанавливаем значение цвета шрифта
$("#my").css({
'color':'red', 'font-size':'I4px', 'margin-left':'l0px'}) // установка нескольких значений
// используя функцию обратного вызова
$("#my").css('height', function(i, value){ return parseFloat(value) * 1.2;
})
```

Рассмотрим некоторые манипуляции с классами:

addClass(className) — добавление класса, addClass(function(index, currentdass){ return className }) — добавление класса используя функцию обратного вызова.

hasClass(className) — проверка на причастность к определённому классу

removeClass(className) — удаление класса

removeClass(function(index, currentdass){ return className }) — удаление класса используя функцию обратного вызова

toggleClass(className) — переключение класса, toggleClass(className, switch) — переключение класса по флагу switch, toggleClass(function(index, currentdass, switch){ return className }, switch) — переключение класса используя функцию обратного вызова

```
<!DOCTYPE html>
<html ><head>
<title>prjq5 - "Элементы JQuery</title>
<script type="text/javascript" src="js/jquery-1.8.3.js"> </script>
 <style type="text/css">
body{color: green;}
p{text-align:center;}
.big{font-size:200%;color:black;}
.test {color:black; font-weight: bold;}
</style>
</head> <body >
<script>alert('Hello1')</script>
 <div>Это первый div  </div>
<div id='idname'>Это второй div  </div>
<div id="content" class="wrapper box">
<h1>Заголовок h1</h1>
<article id="stick"> <h2>Article Title</h2>
<p>Заголовок статьи</p> </article>
<article> <h2>Article Description</h2>
<p>Описание текста</p> </article>
<p >Основной текст.</p>
</div>
<a href=prjq5.htm>Это Ссылка</a>
<script>
alert('Hello2');
var x=$("article").find("h2").html();  // выбираем теги h2 - Article Title
alert(x);
$("div").find("article").find("h2") //
$("article > h2").html("NEWHTML");
 var x= $("article").children().css('fontsize', '200px'); //
 alert('Hello3');
 $ ("a").addClass ("test");//Все  элементы теперь будут выделены
//$ ("a").removeClass ("test");//Для удаления class , используйте removeClass
var x=$('h1').css("fontSize"); alert(x); // returns a string  "32px"
$('h1').css({ "fontSize" : '100px', "color" : "red "}); // setting multiple properties
        var $p = $('p');
        $p.addClass("big");
alert('Hello4');
//$p.toggleClass('big');
if ($p.hasClass('big')) {  $p.removeClass('big'); }
</script>
</body> </html>
```

В приведённых функциях в качестве className может быть строка содержащая список классов через пробел, например $("#my").addClass('active notice'), удаляем несколько классов за раз $("#my").removeClass('active notice')

У DOM элементов бывают атрибуты отличные от класса, и мы их тоже можем изменять, для этого нам потребуются следующие методы: removeAttr(attrName) — удаление атрибута.

Блочная верстка

Рассмотрим методы, которые работают с размерами, и координатами элементов:

offset() - вернёт позицию DOM элемента относительно document, данные будут получены в виде объекта: { top: 10, left: 30 }. offset(top: 10, Left: 30 }) - устанавливаем расположение DOM элемента по указанным координатам

position() - вернёт позицию DOM элемента относительно родительского элемента

height() - возвращает высоту элемента за вычетом отступов и границ; если у нас несколько элементов в выборке, вернётся первый; height(height) - устанавливает высоту всех элементов в выборке, если значение высоты передано без указания единиц измерения, то это будут px

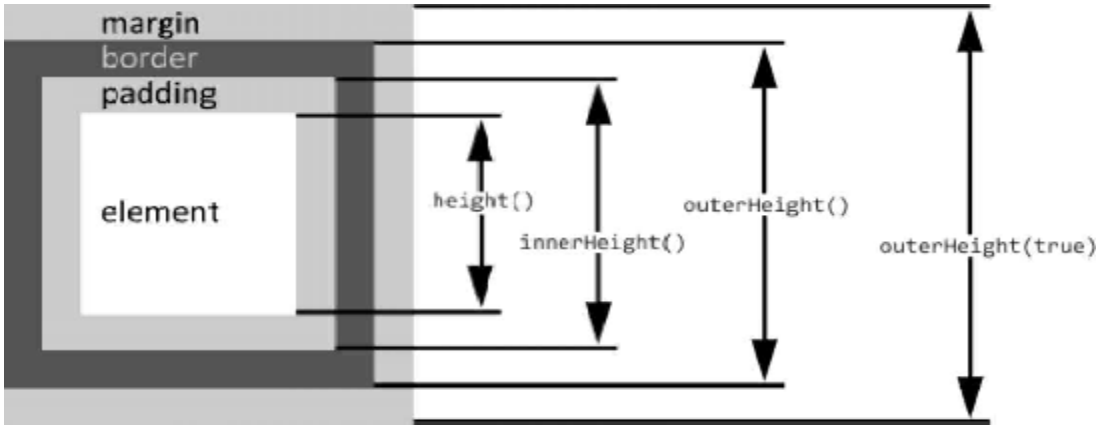
width() и width(width) - ведут себя аналогично методу height(), но работают с шириной элемента

Методы height() и width() не изменяют своего поведения в зависимости от выбранной блочной модели, т.е. они всегда возвращают параметры области внутри margin, padding и border'a элемента.

innerHeight() и innerWidth() - вернут соответственно высоту и ширину элемента, включая padding

outerHeight() и outerWidth() - вернут высоту и ширину элемента, включая padding и border

outerHeight(true) и outerWidth(true) - высота и ширина, включая padding, border и margin




```
<!DOCTYPE html>
<html ><head>
<title>prjq6 - "Элементы JQuery</title>
<script type="text/javascript" src="js/jquery-1.8.3.js"> </script>
 <style type="text/css">
body{color: green;}
p{text-align:center;}
.big{font-size:200%;color:black;}
.test {color:black;
font-weight: bold;}
</style>
</head>
<body >
<script>alert('Hello1')</script>
 <div>Это первый div  </div>
```

```
<div id='idname'>Это второй div  </div>
<div id="content" class="wrapper box">
<h1>Заголовок h1</h1>
<article id="stick"> <h2>Article Title</h2>
<p>Заголовок статьи</p> </article>
<article> <h2>Article Description</h2>
<p>Описание текста</p> </article>
<p >Основной текст.</p>
</div>
<script>
var $h1=$('h1');
var x=$h1.width();// gets the width of the first H1
var y=$h1.height();//gets the height of the first H1
alert("X="+x+"   Y="+y);// x=931, y=36
$h1.width('50px'); // sets the width of all H1 elements
var x=$h1.width();
$h1.height('50px'); // sets the height of all H1 elements
var y=$h1.height(); //всегда возвращают параметры области внутри margin, padding
и border'a элемента.
alert("X="+x+"   Y="+y);//x=50, y=50
alert($('h1').position());//returns an object containing position  information for the first
H1
$("div").offset();
$("article").offset( {top: 10, left: 30 });
 var x= $("article").offset();
 alert('Hello2');
 var $p = $('p');
y=$p.innerHeight();// вернут соответственно высоту и ширину элемента,
x=$p.innerWidth();// включая padding
alert("X="+x+"   Y="+y);//x=931, y=19
x=$p.outerWidth();//outerHeight() и outerWidth() - вернут высоту и ширину
элемента,
y=$p.outerHeight();//включая padding и border
alert("X="+x+"   Y="+y);//x=931, y=19
x=$p.outerWidth(true); //
y=$p.outerHeight(true); //outerHeight(true) и outerWidth(true) - высота и ширина,
//включая padding, border и margin
alert("X="+x+"   Y="+y);//x=931, y=51
</script>
</body> </html>
```

Обработка событий

Обработчик события в JavaScript можно добавить следующим образом:

```
document.getElementById('id').click = function1;.
```

В JQuery это можно сделать так:

```
$('#id').click(function(){/*код обработки по id*/});
```

Код пишется, как обработка события ready() документа. Простой пример:

```
<img src="img/avtor.jpg" id="myfoto" />
<script type="text/javascript">
$(document).ready(function(){
```

```
        $('#myfoto').click(function(){
        alert('This is my photo!');
        });
});
</script>
```

Основные события в JQuery: mousedown — нажата кнопка мыши; mouseup — отпущена кнопка мыши; mouseover — курсор поместился над элементом; mouseout — курсор ушёл с элемента; mousemove — курсор переместился по элементу; keypress — нажата клавиша; submit — отправлены данные (применимо к форме и её элементам); focus — получен фокус; change — изменено; click — клик.

```
<script>
     $(document).ready(function(){
       $("a").click(function(event){
         alert("As you can see, the link no longer took you
to jquery.com");
         event.preventDefault();
       });
     });
   </script>
```

Опробовать события можно на примере с событиями мышки и элементами формы.

```
<!DOCTYPE html>
<html ><head>
<title>prjq6 - "Элементы JQuery</title>
<script type="text/javascript" src="js/jquery-1.8.3.js"> </script>
 <style type="text/css">
body{color: green;}
p{text-align:center;}
.big{font-size:200%;color:black;}
.test {color:black; font-weight: bold;}
</style>
</head>
<body >
 <div>Это первый div  </div>
<h1>Заголовок h1</h1>
<article id="stick"> <h2>Article Title</h2>
<p>Заголовок статьи</p> </article>
<article> <h2>Article Description</h2>
<p>Основные события в JQuery: mousedown — нажата кнопка мыши;
mouseup — отпущена кнопка мыши; mouseover — курсор  над элементом;
mouseout — курсор ушёл; mousemove — курсор переместился по элементу;
keypress — нажата клавиша; submit — отправлены данные (применимо к форме);
focus — получен фокус; change — изменено; click — клик.</p> </article>
<a href=prjq6.htm id='idname'>Это Ссылка</a>
<img src="img/avtor.jpg" id="myfoto" />
<script type="text/javascript">
// сообщение  не при щелчке на картинке, а при перемещении курсора над ней:
        $('#myfoto'). mousemove(function(){
                alert('This is my photo too!');
        });
```

```
$('#idname').mouseover(function(){//событие помещения курсора над ссылкой
                alert('This is my href!');
        });
alert('Hello1');
   // Returns a function that will always run in the provided scope.
  var myFunction = function() {console.log(this);  alert('SOS');};
  var myObject = { foo : 'bar' };
   myFunction(); // logs window object
   var myProxyFunction = $.proxy(myFunction, myObject);
  myProxyFunction(); // logs myObject object
    $('#foo').click(myObject.myFn); // logs DOM element #foo
    $('#foo').click($.proxy(myObject, 'myFn')); // logs myObject
</script>
</body> </html>
```

События для работы с формами

События с которыми чаще всего придётся работать: change — изменение значения элемента, submit — отправка формы. Отслеживание change позволяет обрабатывать такие события как изменение selectbox, или radiobutton. Отслеживание submit потребуется для проверки правильности заполнения формы, а так же для отправки. Для работы с формами используются методы, которые реализуют также события: .blur(); .change(); .focus(); .select() ;.serialize(); .serializeArray(); .submit(); .val()

Рассмотрим простую форму:

```
<form action="/save/">
<input type="text" name="name" value="lvan"/> <select name="role">
<option>User</option> <option>Admin</option> </select>
<input type="submit"/> </form>
```

А примеры будут идти в обратном порядке, вот отправка формы AJAX'ом по ссылке из action:

```
$('form').submit( unctioi(){
// чуть позже расскажу подробнее о AJAX  $.post(
$(this).attr('action'), // ссылка куда отправляем данные
$(this).serialize()        // данные формы
// отключаем действие по умолчанию return false;
});
```

Вот и первый метод - serialize() - он в ответе за «сбор» данных с формы в удобном для передачи данных формате:

```
name=lvan&role=Admin
```

Так же есть метод serializeArray() -он собранные данные представляет в виде объекта: [

```
{
name:"name", value:"Ivan"
}, {
name:"role", value:"Admin"
},
]
```

Теперь стоит добавить в данный код немного проверки данных:

```
$('form').submit( unctioi(){
if ($(this).find('input[name=user]').val() == '') { alert('Введите имя пользователя');
return false;
```

}
// кусок кода с отправкой // ...
});
Вот еще один метод, который нам будет частенько нужен:
val() - получение значения первого элемента формы из выборки val(value) - установка значение всем элементам формы из выборки
Данный метод отлично работает практически со всеми элементами формы, вот только с ^юЬ^^п^ми установить значение таким образом не получится, тут потребуется небольшой workaround:
$('input[type=radio][name=choose][value=2]').prop('checked', true)
Можно конечно же использовать и метод click() дабы эмулировать выбор необходимо пункта, но это вызовет обработчики clicks, что не желательно
С checbox'ами чуть-чуть попроще:
$('input[name=check] ,).prop('checked', true)
Проверяем «чекнутость» простым скриптом:
$('input[name=check] ').prop('checked') // или чуть более наглядным способом $('input[name=check] ').is(':checked')
Проверять и отправлять форму AJAX'ом теперь умеем, теперь осталось решить вопрос с динамическим изменением формы, и для этого у нас уже есть все необходимые знания, вот, к примеру, добавление выпадающего списка:
$('form').append('<select name="some"></select>');
А если потребуется изменить список? Есть на все случаи жизни:
// возьмём список заранее, поберегу чернила var $select = $('form select[name=Role]'); // добавить новый элемент в выпадающий список $select.append('<option>Manager</option>'); // выбрать необходимый элемент $select.val('Value 1'); // или по порядковому номеру, начиная с 0 $select.find('option:eq(2)').prop('selected', true); // очищаем список
$select.remove('option'); // преобразуем в multiple
// не забываем, что имя такого селекта, должно быть с [], т.е. // myselect[], иначе сервер получит, лишь одно значение $('select').attr('size',
$('select option').length
)
$('select').attr('multiple', true)
Хорошо, работать с формой теперь можем, осталось прикрутить более вменяемый вывод ошибок (да-да, за alert() да по рукам):
if ($(this).find('input[name=user]').val() == '') { $(this).find("input[name=user]')
.before(' Введите имя ,); return false;
}
При повторной отправки формы не забудьте убрать сообщения оставшиеся от предыдущей проверки:
$(this).find('.error').remove()
Теперь можно объединить кусочки кода и получить следующий вариант:
$('form').submit(unction(){ // чистим ошибки
$(this).find('.error').remove(); // проверяем поля формы
if ($(this).find('input[type=name]').val() == '') { $(this).find("input[name=user]')
.before(' Введите имя '); return false;
}
// всё хорошо - отправляем запрос на сервер $.post(
$(this).attr('action'), // ссылка куда отправляем данные
$(this).serialize()
);

return false;

Теперь стоит вернуться к списку событий формы, и перечислить недостающие:

focus — фокус на элементе, blur — фокус ушёл с элемента + метод blur(), для работы с ним; пригодится при валидации формы по мере заполнения полей

select — выбор текста в textarea и input[type=text] + метод select(); submit — отправка формы + метод submit();

Примеры работы с элементами формы доступны на странице form.html

// данные формы

:button Selects <button> elements and elements with type="button"

:checkbox Selects inputs with type="checkbox"

:checked Selects checked inputs

:disabled Selects disabled form elements

:enabled Selects enabled form elements

:file Selects inputs with type="file"

:image Selects inputs with type="image"

:input Selects <input> , <textarea> , and <select> elements

:password Selects inputs with type="password"

:radio Selects inputs with type="radio"

:reset Selects inputs with type="reset"

:selected Selects options that are selected

:submit Selects inputs with type="submit"

:text Selects inputs with type="text"

Пример $('#myForm :input'); // get all elements that accept input

Для работы с формами используются методы, которые реализуют также события:

.blur(); .change(); .focus(); .select() ;.serialize(); .serializeArray(); .submit(); .val()

Эффекты и анимация

Список JQuery-методов, с помощью которых можно «оживить» сайт. Первым параметром (необязательным) к каждому из методов идёт скорость в милисекундах.

`Show()` — показать элемент. В зависимости от параметров, либо просто прописывает элементу `display: block;` без анимации, либо медленно сворачивает элемент. `hide()` — скрыть элемент. Аналогично предыдущему.

`toggle` — показать/скрыть работает как переключатель hide / show или show / hide.

Если скрыт — показывает, и наоборот.

`slideDown` — свёртывание, `slideUp` — развёртывание

`slideToggle` — свёртывание/развёртывание, в зависимости от текущего состояния

`fadeIn` — появление, `fadeOut` — затухание

`fadeTo` — появление-затухание до нужной прозрачности. Вторым параметром идёт прозрачность, до которой необходимо затухнуть

// скроем все картинки

$('img').hide();

// теперь вернём их на место

$('img').show();

Данные вызовы оперируют лишь CSS атрибутом display и переключают его из текущего состояния в none и обратно. В качестве первого параметра можно задать скорость анимации, для этого можно использовать одно из зарезервированных слов «slow» или «fast», либо же указывать скорость в миллисекундах (1000 мс = 1 сек):

$('img').hide('slow');

$('img').show(400);

вторым параметром в приведенных методах может быть callback-функция - она будет выполнена по окончанию анимации элементов:

// скрываем все картинки $('img').hide('slow', function (){alert("lmages was hidden");
});

```
$("a").click(function(event){
   event.preventDefault();
   $(this).hide("slow");
 });
```

JQuery и AJAX

С появлением jQuery реализация Ajax упростилась. Рассмотрим два метода load() и ajax(). Начнем с рассмотрения загрузки HTML кода в элемент DOM с помощью метода load() с параметрами: url - запрашиваемой страницы, data - данные (необязательный параметр), callback – функция, вызываемая при завершении запроса к серверу (необязательный параметр). Например: $("#content").load("/mypage.htm") вставит в элемент с id=content указанный код HTML.

// передаем данные на сервер
$("#content").load("/mypage.htm", {id:l8});
// обрабатываем полученные данные
$("#content").load("/mypage.htm", function (){ alert("Ничего строка");
});

Второй пример может выручить, когда надо реализовать загрузку AJAX'ом, а доступа к сервер-сайту нет.

Метод ajax() главный. Метод принимает параметры и URL куда стучаться:

$.ajax({
url: "/mypage.html", // указываем URL и тип загружаемых данных
success: function (data) {
// вешаем свой обработчик события success
$("#content").html(data)
}
});

Тут мы обрабатывали HTML ответ от сервера, но данные лучше передавать в «правильном» формате XML. Проще выбрать JSON:

{
"note": {
"time":"2012.09.21 13:11:15", "text":"Рассказать про JSONP"
}
}

Фактически это и есть JavaScript код. Для загрузки JSON существует быстрая функция-синоним - jQuery.getDSON(url [,data] [,success(data, textStatus, jqXHR)]) обязательный параметр лишь ссылка, куда стучимся. Опционально можно указать данные, для передачи на сервер и функцию обратного вызова

Обработчики AJAX событий

AJAX запросы имеют несколько событий:

ajaxStart — событие возникает момент, когда послан первый AJAX запрос, а других активных AJAX запросов нет. beforeSend — локальное событие возникает до отправки запроса и позволяет редактировать XMLHttpRequest. ajaxSend — возникает до отправки запроса, аналогично beforeSend. success — локальное событие возникает по возвращению ответа, когда нет ошибок ни сервера, ни вернувшихся данных.

ajaxSuccess — возникает по возвращению ответа, аналогично success error — возникает в случае ошибки, локальное событие ajaxError — возникает в случае ошибки

complete — локальное событие всегда возникает по завершению текущего AJAX запроса (с ошибкой или без), ajaxComplete — глобальное событие, аналогичное complete

ajaxStop —событие возникает в случае, когда больше нет активных запросов

jQuery позволяет обрабатывать эти события для каждого AJAX запроса локально либо глобально. Пример для отображения элемента с id="loading" во время выполнения любого AJAX запроса (глобальное событие):

```
$("#loading").bind("ajaxSend", function(){
$(this).show(); // показываем элемент }).bind("ajaxComplete", function(){
$(this).hide(); // скрываем элемент
});
```

Можно глобальные обработчики отключить принудительно, используя флаг global= false, и писать функционал в обход событий ajaxStart и ajaxStop.

Для локальных событий - вносим изменения в опции метода ajax():

```
$.ajax({
beforeSend: function(){// обработчик будет вызван  перед отправкой запроса
},
success: function(){//будет вызван  при удачном завершении
},
error: function(){//будет вызван при возникновении ошибки
},
complete: function(){//будет вызван по завершению запроса
}
});
```

Рассмотрим примеры

```
$.ajax({//url, к которому обращаемся
url:     '/ajax.php?act=jquery_test',
type:    'GET', //тип: GET или POST
success: function(response){ //вызывается, когда //запрос
//прошёл успешно и данные (data) получены
alert('Сервер вернул ответ: ' + response);
  }
});
```

Это простейший пример использования ajax в JQuery. Теперь посмотрим, как можно отдать сереверу данные методом POST:

```
$.ajax({
url:    '/ajax.php?act=ajax_jquery_post',
type:   'POST',
contentType: 'application/x-www-form-urlencoded',
 //Тип передаваемых данных
data:   'text='+$('#text').val()+'&id=282&c=w',
//а это, собственно, данные (произвольные)
success: function(response){
alert('Данные отправлены! Сервер вернул ответ: ' +
response);
  }
});
```

Это был пример отправки данных на сервер с помощью JQuery AJAX.

Рассмотрим пример отправки комментария.

```
<form id="comment">
<input type="hidden" id="contentid" value="номер статьи
(вставляется с помощью PHP)" />
```

```
<input type="text" name="nick"><br />
<textarea name="comment"></textarea><br />
<button id="sendcomment">Отправить</button>
</form>
<script type="text/javascript">
$(document).ready(function(){
$('#sendcomment').click(function(){
  $.ajax({
  url: '/some-url/content_id='+$('#contentid').val(),
//Указываем УРЛ
  type: 'POST', //Указываем метод:
  data: 'nick='+$('input[name=nick]').val()+'&comment='+
$('textarea[name=comment]').val(),
//Указываем данные, отправляемые POSTом
  success: function(data){ //будет выполняться при успехе
  alert('Ваш комментарий отправлен!');
  },
  error: function(){ // будет выполняться при неудаче
  alert('Ошибка! Попробуйте ещё раз!');
        }
        });
  });
});
</script>
```

Теперь немного о других форматах:

```
$.ajax({
url:      'ajax.php?act=ajax_jquery',
type:     'POST',
dataType: 'JSON',//форматы могут быть:JSON, XML, HTML, text
data:     someData,
success:  function(response){
   //какие-нибудь действия
  }
});
```

Кроме того, можно создать обработчик события "успех" (success) и других событий:

Событие **success** вызывается, когда запрос завершился успешно. Его обработчику передаются параметры: возвращаемые сервером данные, строка с кодом ответа сервера, и объект xmlHttpRequest. Аналогично **error** вызывается, когда запрос завершился неудачей. Передаются: xmlHttpRequest и строка, указывающая тип ошибки

Событие **complete** вызывается, когда запрос завершился, независимо от того, удачей или нет. Передаются: xmlHttpRequest и строка, указывающая код успеха или ошибки.

dataFilter вызывается перед вызовом success. Ему передаются полученные от сервера данные. В ней данные обрабатываются и возвращаются **в success. beforeSend** вызывается **до** отправки запроса на сервер

Теперь еще один пример:

```
$.ajax({
url:     'ajax.php?act=something',
type:    'GET', //что-нибудь получим
success: function(response, code){
 //пропустили последний параметр — он не нужен
```

```
        if (code==200){
          alert('Сервер вернул: ' + response);
        }else{
          alert('Сервер вернул непонятный код ответа: ' +
code);
        }
    },
    error:  function(xhr, str){
        alert('Возникла ошибка: ' + xhr.responseCode);
    }
    complete:  function(){
    //тут ничего из предложенных параметров не берем :)
        $('#something').hide(); //например, спрятали какую-то
кнопочку, которая вызывала запрос
    }
    });
```

Это уже пример посерьёзней,— с обработкой ошибок.

С помощью **ajaxSetup()** можно глобально во всём скрипте задать все необходимые опции. Пример:

```
    $(document).ready(function(){
    $.ajaxSetup({
      url:   'ajax.php',
      type:  'POST',
      success: function(data){
        $('#somefield').val(data);
      }
            error:   function(){
        $('#somebutton').addClass('error');
      }
    });
    $('#somebutton1').click(function(){
      $.ajax({data: 'act=1'});
      });
      $('#some2').click(function(){
            $.ajax({data: 'act=2'});
    });
    });
```

Опции $.ajax

- **async** (true или false). По умолчанию true. Включает или выключает асинхронные запросы как в компоненте xmlHttpRequest.
- **cache** (true или false). Включает или выключает кеширование браузером
- **contentType** (строка). Тип содержания, передаваемого на сервер. При сабмите форм используйте application/x-www-form-urlencoded
- **data** (строка). Данные, отправляемые на сервер.
- **dataType** (строка). Тип ожидаемых от сервера данных: xml, json, script, html. Если не указан, JQuery попытается определить результат, основанный на MIME-типе ответа.

ifModified (true или false (по умолчанию)). Если установлено в true, то запрос будет успешным только тогда, когда ответ изменился с момента прошлого запроса (достигается путём проверки заголовка Last-Modified)

- **timeout** (в милисекундах). Период, после которого, соединение с сервером будет обрываться.
- **type** (запрос: GET или POST).

- **url** (строка). Страница сервера, к которой будет сделан запрос.

Для того, чтобы примеры работали корректно, необходимо: 1. Все файлы должны быть записаны в кодировке UTF-8. 2. Скрипты должны выполняться на веб-сервере, а не запускаться в браузере, как файл.

Создадим простую программу, отображающую текущее время, обновляемое по таймеру. Особенностью данной программы будет получение информации о текущем времени из другого внешнего файла.

Содержимое файла index.html.

```
<!-- Пример 1. Динамическое обновление контента по таймеру-->
<html><head>
<meta http-equiv="Content-Type" content="text/html; Charset=UTF-8">
<script type="text/javascript" src="jquery.js"></script>
</head>
<body>
	<div id="content"></div>
		<script>
		function show()
		{
			$.ajax({
				url: "time.php",
				cache: false,
				success: function(html){
					$("#content").html(html);
				}
			});
		}
			$(document).ready(function(){
			show();
			setInterval('show()',1000);
		});
	</script>
	</body> </html>
```

Файл time.php

```
<?php echo date("H:i:s"); ?>
```

```
<!-- Пример 2. Динамическое обновление контента по выбору пользователя
-->
<html><head>
<meta http-equiv="Content-Type" content="text/html; Charset=UTF-8">
<script type="text/javascript" src="jquery.js"></script>
</head>
body>
		<p>Какую страницу желаете открыть?</p>
	<form>
		<input id="btn1" type="button" value="Страница 1"> <input id="btn2"
type="button" value="Страница 2">
	</form>
	<div id="content"></div>
		<script>
		$(document).ready(function(){
				$('#btn1').click(function(){
```

```
                $.ajax({
                        url: "page1.html",
                        cache: false,
                        success: function(html){
                                $("#content").html(html);
                        }
                });
        });
                                $('#btn2').click(function(){
                $.ajax({
                        url: "page2.html",
                        cache: false,
                        success: function(html){
                                $("#content").html(html);
                        }
                });
        });

    });
</script>
</body> </html>
```

Page2.htm

```
<h1>Страница 2</h1>
<p>Текст второй страницы</p>
<p>Продолжение</p>
<p> Продолжение</p>
```

Page1.htm

```
<h1>Страница 1</h1>
<p>Первая страница</p>
<p>Продолжение </p>
```

```
<!-- Пример 3. Отправка данных на сервер в фоновом режиме и получение
контента-->
<html><head>
<meta http-equiv="Content-Type" content="text/html; Charset=UTF-8">
<script type="text/javascript" src="jquery.js"></script>
</head>
<body>
        <form id="myForm">
        Введите имя:<br/>
                <input id="username" type="text" size="20"><br/><br/>
                <input type="submit" value="Отправить">
        </form>
                <div id="content"></div>
                <script>
                $(document).ready(function(){
                                $('#myForm').submit(function(){
                $.ajax({
```

```
                                type: "POST",
                                url: "greetings.php",
                                data: "username="+$("#username").val(),
                                success: function(html){
                                        $("#content").html(html);
                                }
                        });
                        return false;
                });

        });
</script>
</body></html>
```

Файл greetings.php

```
<?php
        echo "Приветствую, <b>".$_REQUEST['username']."</b>!<br>";
        echo "В вашем имени букв: ".strlen($_REQUEST['username']).".";
 ?>
```

Cookies

Cookie является решением одной из проблем HTTP протокола. Известно, что транзакция завершается после того, как браузер сделал запрос, а сервер выдал соответствующий ответ. После этого сервер "забывает" о пользователе и каждый следующий запрос считает новым пользователем. Используя cookie, на первом запросе можно запомнить его значение, а при каждом последующем запросе это значение читается из переменной окружения HTTP_COOKIE и обрабатывается. Например, при каждом последующем входе из браузера пользователя на странице появляется либо именное приветствие (если есть установленное значение cookie), либо первоначальная форма с запросом имени пользователя (если значение cookie не установлено).

Cookie - это небольшая порция текстовой информации о предыдущем сеансе клиента, которую сервер передает браузеру для хранения. Браузер будет передавать ее серверу с каждым запросом как часть HTTP заголовка. Значения cookie, установленные на некоторый период времени, записываются в файл. 'cookies.txt'. Сервер может считывать содержащуюся в cookies информацию и на основании ее анализа совершать те или иные действия. Например, в случае авторизованного доступа к чему-либо через WWW в cookies сохраняется login и password в течение сессии, что позволяет пользователю не вводить их снова при запросах каждого документа, защищенного паролем. Еще одна распространенная область использования cookies - при настройке индивидуального профиля каждого зарегистрированного пользователя. И, наконец, самая последняя область - использование механизма cookie в рекламном бизнесе на Интернет. Cookie используются для определения целевой аудитории по географическому положению пользователей, отслеживания интересов пользователей, учета количества показов и проходов сквозь баннеры.

Работа с cookie

Cookie является частью HTTP заголовка. Полное описание поля Set-Cookie HTTP заголовка:

Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME;

NAME=VALUE - NAME-имя cookie, VALUE - значение. expires=DATE - время до которого хранится cookie в формате "expires=Monday, DD-Mon-YYYY HH:MM:SS

GMT". Если атрибут не указан, то cookie хранится в течение одного сеанса, до закрытия браузера.

domain=DOMAIN_NAME - домен, для которого значение cookie действительно. Если этот атрибут опущен, то по умолчанию используется доменное имя сервера, на котором было задано значение cookie.

path=PATH - этот атрибут устанавливает подмножество документов, для которых действительно значение cookie. Например, указание "path=/win" приведет к тому, что значение cookie будет действительно для множества документов в директории /win/, в директории /wings/ и файлов в текущей директории с именами типа wind.html и windows.shtml. Для того, чтобы cookie отсылались при каждом запросе к серверу, необходимо указать корневой каталог сервера, например, "path=/".

Если этот атрибут не указан, то значение cookie распространяется только на документы в той же директории, что и документ, в котором было установлено значение cookie.

Синтаксис HTTP заголовка для поля Cookie

Когда запрашивается документ с HTTP сервера, браузер проверяет свои cookie на предмет соответствия домену сервера. Если найдены удовлетворяющие всем условиям значения cookie, браузер посылает их в серверу в виде пары имя=значение:
Cookie: NAME1=OPAQUE_STRING1;NAME2=OPAQUE_STRING2 ...

Одновременно можно задавать несколько значений cookie.

В случае, если cookie принимает новое значение при имеющемся уже в браузере cookie с совпадающими параметрами NAME, domain и path, то старое значение заменяется новым. В остальных случаях новые значения cookie добавляются к старым.

Клиент (браузер) имеет следующие ограничения для cookies:

- всего может храниться до 300 значений cookies
- каждый cookie не может превышать 4Кбайт
- с одного сервера или домена может храниться до 20 значений cookie

Если ограничение 300 или 20 превышается, то удаляется первая по времени запись. При превышении лимита объема отрезается кусок записи (с начала этой записи) равный превышению объема.

В случае кеширования документов, например, proxy-сервером, поле Set-cookie HTTP заголовка не кэшируется.

Ниже приведено несколько примеров, иллюстрирующих использование cookies

Пример 1. Управление подмножеством документов, для которых действительны значения cookie, и их сроком годности

Браузер запрашивает документ и принимает от сервера в ответ:

Set-Cookie: CUSTOMER=WILE_E_COYOTE; path=/; expires=Wednesday, 09-Nov-99 23:12:40 GMT

Когда браузер запрашивает URL с путем "/" на этом сервере, он посылает серверу:
Cookie: CUSTOMER=WILE_E_COYOTE

Браузер запрашивает документ и принимает от сервера в ответ: Set-Cookie: PART_NUMBER=ROCKET_LAUNCHER_0001; path=/

Когда браузер запрашивает URL с путем "/" на этом сервере, он посылает серверу уже два значения cookie: Cookie: `CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001`

Сервер установил еще одно значение cookie, на этот раз с другой областью действия: Set-Cookie: SHIPPING=FEDEX; path=/foo

Теперь браузер, запрашивая URL с путем "/" на этом сервере, посылает лишь два значения cookie: Cookie: `CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001`

и лишь при запросе браузером документов с путем "/foo" на этом сервере посылаются все три значения cookie: Cookie: `CUSTOMER=WILE_E_COYOTE; PART_NUMBER=ROCKET_LAUNCHER_0001; SHIPPING=FEDEX`

Комментарий: после закрытия браузера в файле 'cookies.txt' останется только одно значение cookie:

CUSTOMER=WILE_E_COYOTE

поскольку только для него установлен срок годности - 9 ноября 1999 года. Все остальные значения не будут сохранены.

Способы задания значений cookie

Есть несколько способов задания, наиболее часто используются три - через META-теги языка HTML, JavaScript и серверные-скрипты.

1. Задание cookie с помощью META-тегов

В общем случае это выглядит следующим образом:

<META HTTP-EQUIV="Set-Cookie" CONTENT="NAME=value; EXPIRES=date; DOMAIN=domain_name; PATH=path; SECURE">

2. Задание cookie с помощью JavaScript. Такие примеры можно посмотреть по адресу http://www.citforum.ru/internet/javascript/exorg.shtml

Пример. Функция установки значения cookie

```
// name - имя cookie
// value - значение cookie
// [expires] - дата окончания действия cookie (по умолчанию - до конца сессии)
// [path] - путь, для которого cookie действительно (по умолчанию - документ, в котором значение было установлено)
// [domain] - домен, для которого cookie действительно (по умолчанию - домен, в котором значение было установлено)
// [secure] - логическое значение, показывающее требуется ли защищенная передача значения cookie
function setCookie(name, value, expires, path, domain, secure) {
 var curCookie = name + "=" + escape(value) +
  ((expires) ? "; expires=" + expires.toGMTString() : "") +
  ((path) ? "; path=" + path : "") +
  ((domain) ? "; domain=" + domain : "") +
  ((secure) ? "; secure" : "")
 if (!caution || (name + "=" + escape(value)).length <= 4000)
  document.cookie = curCookie
 else
  if (confirm("Cookie превышает 4KB и будет вырезан !"))
   document.cookie = curCookie
}
```

Пример. Функция чтения значения cookie. Возвращает установленное значение или пустую строку, если cookie не существует.

```
// name - имя считываемого cookie
function getCookie(name) {
 var prefix = name + "="
 var cookieStartIndex = document.cookie.indexOf(prefix)
 if (cookieStartIndex == -1)
  return null
 var cookieEndIndex = document.cookie.indexOf
(";", cookieStartIndex + prefix.length)
```

```
 if (cookieEndIndex == -1)
  cookieEndIndex = document.cookie.length
 return unescape(document.cookie.substring(cookieStartIndex + prefix.length,
cookieEndIndex))
}
```

Пример. Функция удаления значения cookie. Принцип работы этой функции заключается в том, что cookie устанавливается с заведомо устаревшим параметром expires, в данном случае 1 января 1970 года.

```
// name - имя cookie
// [path] - путь, для которого cookie действительно
// [domain] - домен, для которого cookie действительно
function deleteCookie(name, path, domain) {
 if (getCookie(name)) {
  document.cookie = name + "=" +
  ((path) ? "; path=" + path : "") +
  ((domain) ? "; domain=" + domain : "") +
  "; expires=Thu, 01-Jan-70 00:00:01 GMT"
 }
```

Глава 6. Язык серверных скриптов PHP

Язык PHP является простым и мощным скриптовым языком, предназначенным для разработки серверной части сайта. Язык был разработан Расмусом Лердорфом в 1994 году. Считается, что название PHP произошло от словосочетания Personal Home Page Tools. Более современная аббревиатура PHP - “Препроцессор гипертекста”. Скрипты PHP обрабатывают на сервере данные из передаваемого с помощью браузера запроса клиента, и возвращают клиенту результат в виде гипертекстового документа.

Использование PHP – скриптов позволяет добавить на сайт такие сэрвисы как формы для обратной связи, счетчики посещений, гостевые книги, форумы, голосования, регистрацию, поиск информации, службы новостей.

На PHP можно генерировать динамические страницы, пересылать файлы cookies, содержащие индивидуальную информацию о клиенте. PHP обеспечивает поддержку различных баз данных. Широкое распространение получила связка PHP с БД MySQL. PHP понимает и умеет работать с почтовыми протоколами POP3 и SMTP, с протоколами TCP/IP, HTTP и другими, позволяет работать с файлами и сокетами.

Синтаксис PHP подобен синтаксису C++, хотя объектная модель ближе к модели Java. Программы, написанные на PHP, просты для понимания и обеспечивают высокую скорость работы. PHP имеет открытый исходный код.

Инструменты для разработки

Хорошим редактором для разработки кода PHP является редактор “Adobe DreamWeaver CS5” с возможными расширениями (JQuery API) или “PHP Storm”. Программисты Java могут использовать для разработки PHP- скриптов среду разработки Eclipse и плогины к Eclipse JSEclipse и Aptana Studio.

Для выполнения PHP необходим web-сервер Apache, препроцессор PHP и база данных MySql, которые устанавливаются на сервере. Для работы с файлами полезной будет программа Total Comander, которая позволяет также разместить PHP – приложение на удаленном сервере. Для этого необходим хостинг сайта и DNS - адрес. Отладка PHP – сайта возможна на локальном (localhost) компьютере. Одним из лучших браузеров, используемых при разработке, является Firefox. Для быстрой установки сервера Apache и БД MySql можно использовать пакет Denver. В этот пакет входит еще и SMTP сервис для работы с почтой.

Как PHP работает

Как и для JavaScript, PHP – код включается внутрь HTML - документа. Расширение файла, содержащего код, может быть PHP, HTM, HTML или PHTML. В конфигурационном файле WEB-Сервера Apache необходимо добавить директиву обработки этого расширения файлов: AddType. Препроцессор PHP должен быть установлен на сервере в одном каталоге вместе с Apache.

Перед началом работы с PHP- скриптами необходимо загрузить Веб-сервер Apache. Обычно при этом загружается и сервер MySql. Затем надо загрузить браузер и в окне набрать адрес www.localhost/name.php. Поскольку код PHP перед отправкой пользователю должен обрабатываться сервером, то наличие сервера Apache обязательно, иначе страница будет просто отсылаться пользователю в необработанном виде.

Код PHP может быть оформлен следующими способами:

```
<?php //стиль xml, основной
//php инструкции
?>
```

В таком виде код вставляется в HTML – документ. После этого скрипт может сохраняться и исполняться в виде файла - документа с расширением .php или .htm. В сокращенном варианте символы php после вопросительного знака отсутствуют, каждый скрипт открывается тегом <? и закрывается ?>. Пример простого файла с PHP:

```
 <!—pr0.php -->
 <html>
 <head> </head>
<?php
 echo "А это внутри PHP - основной способ вставки кода PHP<BR><HR>";
phpinfo(); //возвращает много информации о версии
// и настройках php
?>
 <body>
 Вставка кода PHP внутрь документа Html<BR>
 <?
 print "Hello, to you - второй способ вставки кода PHP<BR>";
 ?>
 <script language="php">
 print "Hello,world-третий способ вставки кода PHP<BR>";
 </script>
 </body></html>
```

Вывод:

А это внутри PHP – основной способ вставки кода PHP
Вставка кода PHP внутрь документа Html
Hello, to you - второй способ вставки кода PHP
Hello,world-третий способ вставки кода PHP

Как видно из примера, скрипт PHP, содержащий команду echo, внедрен внутрь HTML документа. Оператор echo (string arg1, string arg2, string argn) выводит все параметры - строки. Для вывода могут использоваться также операторы print, аналогичные echo(). С операторами print() и echo() скобки не обязательны в отличие от оператора форматированного вывода printf(“format”, arglist), аналогичного C++.

PHP позволяет использовать такие структуры:

```
<?php //pr1.php
$exp=true;
if ($exp) {
    ?>
```

```
        <strong>Это истина.</strong>
        <?php
    }
    else {
        ?>
        <strong>Это ложь.</strong>
        <?php
    }
    printf("<br> %s", "Выход из режима интерпретации позволяет
вывести большой текст");
    ?>
    Вывод:
    Это истина.
```

Выход из режима интерпретации позволяет вывести большой текст.

Когда PHP встречает закрывающие теги ?>, он просто выводит все, что он находит до следующего открывающего тега. Приведенный в примере выход из режима интерпретации PHP для вывода больших блоков текста более эффективен, чем отправка текста через echo() или print() .

В любом случае в результате работы PHP – кода получим гипертекстовый документ, готовый к отображению браузером.

Описание языка

Типы данных

PHP поддерживает 4 скалярных типа данных: boolean (значения true и false), integer (значения целые числа, которые могут быть десятичными, восьмеричными и шестнадцатеричными), float (длинные вещественные числа с плавающей точкой, соответствующие типу double), string(строки в одинарных и двойных апострофах) и два составных типа: array и object. Имеется два специальных типа: resource и NULL. Перед целыми восьмеричными значениями записывается нуль, перед шестнадцатеричными – 0x. Рассмотрим пример:

```
<?php //pr01.php
$a = 10;
$b = 020;//восьмеричное значение
$c = 0x41;//шестнадцатеричное значение
$d=3.62e+2;
printf("%d %o %c  %f",$a,$b,$c,$d);
?>
```

Вывод:

10 20 A 362.000000

Здесь %d – формат для десятичного целого, %o – для восьмеричного, %c – для вывода символа с указанным кодом, %f - для вывода вещественных значений.

Если необходимо проверить тип и значение определенного выражения, можно использовать функцию var_dump(). Если же для отладки необходимо просто представление типа, используйте gettype(). Чтобы проверить выражение на определенный тип, применяйте функцию is_*type()*.

Имена всех переменных начинаются знаком $. PHP не требует явного определения типа при объявлении, тип переменной определяется по присвоенному ей значению. То есть, если присвоить переменной $p строковое значение, $p становится строкой. Если затем присвоить переменной $p значение целого числа, то $p станет целым.

Для инициализации переменной в PHP ей присваивается значение. Кроме переменных могут быть использованы поименованные константы, объявленные в операторе define() без знака $. Рассмотрим пример:

```
<?php //pr2.php
define("d",5,true);
$a = "Добро пожаловать ";
$b = "в PHP";
$c = 4;
$c++;
echo "$a$b$c"; //получим - Добро пожаловать в PHP5
print d;//Вывод константы 5
$p=NULL;
print  var_dump($p);
print  gettype($a);
?>
```

Вывод:

Добро пожаловать в PHP55NULL string

Комментарии в PHP используются, как и в C++:

```
/*многострочный комментарий */ или
 // однострочный комментарий или
 #еще один однострочный комментарий.
```

Строки

Строки являются последовательностями символов, каждый символ соответствует байту. Это означает, что PHP не имеет встроенной поддержки Unicode. Некоторую поддержку Unicode обеспечивают функции utf8_encode() и utf8_decode(). Переход на кодировку Unicode запланирован в версии PHP6.

В PHP не существует ограничения на размер строк. Строковый литерал можно представлять следующими способами: строки в одиночных кавычках и строки в двойных кавычках. Строка, заключенная в двойные кавычки может содержать подстроку в одинарных кавычках и наоборот. Строка, заключенная в двойные кавычки, означает, что встречающиеся в этой строке переменные будут заменены их значениями. Если строка заключена в одинарные кавычки, такая замена не производится, так как такая строка не разбирается в PHP.

```
<?php //pr3.php
$name = 'Domine?';
$name_1 = "Quo vadis, $name";
$name_2 = ' Quo vadis, $name';
echo "$name_1"; // выведет - Quo vadis, Domine?
echo "$name_2"; // выведет - Quo vadis, $name
?>
```

Для строк в двойных кавычках PHP поддерживает так называемые экранированные символы, которые могут встречаться среди символов строки: \n - перенос строки, \r- возврат каретки , \t- горизонтальная табуляция, \\- обратная косая черта, \$- знак доллара, \\"- двойная кавычка, \код – код символа. Например:

```
$s="\x41- это символ ‘a’ ";
$st=" можно использовать \" управляющий \" символ";
```

Самым важным свойством строк в двойных кавычках является обработка переменных, находящихся в строках.

Третий способ определения строк - это использование heredoc-синтаксиса ("<<<"). После <<< указывается идентификатор, затем идет строка, а потом этот же идентификатор, закрывающий строку. Закрывающий идентификатор должен начинаться в первом столбце строки. Строка с закрывающим идентификатором не содержит других символов, за исключением, точки с запятой (;). Это означает, что идентификатор не должен вводиться с отступом и не может быть никаких пробелов или знаков табуляции до или после точки с запятой. Первым символом перед закрывающим идентификатором должен быть символ новой строки. Например, \r. Heredoc-текст ведет себя так же, как и строка в двойных кавычках, при этом их не имея. Это означает, что нет необходимости экранировать кавычки в heredoc, но по-прежнему можно использовать управляющие последовательности. Пример определения heredoc-строки:

```
<?php
  $name = "Max";
  $str = <<<DEMO
Hello $name! <br/>
This is a demo message
with heredoc.
DEMO;
   echo $str;
?>
```

Вывод:

Hello Max! This is a demo message with heredoc.

Если строка определяется в двойных кавычках, либо при помощи heredoc, переменные внутри нее обрабатываются. Существует два типа синтаксиса: простой и сложный. В простом синтаксисе если интерпретатор встречает знак ($), он захватывает столько символов, сколько можно, чтобы сформировать правильное имя переменной. Чтобы точно определить конец имени, заключайте имя переменной в фигурные скобки.

```
<?php
$beer = 'Речицкое';
echo "$beer's taste is great"; /* работает, 's - неверный символ для имени переменной */
echo "He drank some $beers";   /* не работает, 's' это верный символ для имени переменной*/
echo "He drank some ${beer}s"; // работает
echo "He drank some {$beer}s"; // работает
?>
```

Сложный (фигурный) синтаксис позволяет использовать сложные выражения.

Фактически, вы можете включить любое значение, находящееся в пространстве имени в строке с этим синтаксисом. Вы просто записываете выражение таким же образом, как и вне строки, а затем заключаете его в { и }. Поскольку вы не можете экранировать '{', этот синтаксис будет распознаваться только когда $ следует непосредственно за {. Пример:

```
<?php
// Давайте покажем все ошибки
error_reporting(E_ALL);
$great = 'fantastic';
// Не работает, выведет: This is { fantastic}
echo "This is { $great}";
// Работает, выведет: This is fantastic
echo "This is {$great}";
echo "This is ${great}";
```

```
// Работает
echo "Этот квадрат шириной {$square->width}00 сантиметров.";
// Работает
echo "Это работает: {$arr[4][3]}";
// Это неверно по той же причине, что и $foo[bar] неверно вне
// строки. Другими словами, это по-прежнему будет работать,
// но поскольку PHP сначала ищет константу foo, это вызовет
// ошибку уровня E_NOTICE (неопределенная константа).
echo "Это неправильно: {$arr[foo][3]}";
// Работает. При использовании многомерных массивов, внутри
// строк всегда используйте фигурные скобки
echo "Это работает: {$arr['foo'][3]}";
// Работает.
echo "Это работает: " . $arr['foo'][3];
echo "Вы даже можете записать {$obj->values[3]->name}";
echo "Это значение переменной по имени $name: {${$name}}";
?>
```

В PHP существуют, так называемые, обратные кавычки (‘), унаследованные из языка Perl. Заключенные в них строки воспринимаются и выполняются как команды операционной системы. Например

```
<?php
echo ‘dir’; ?>
```

Массивы и инициализация массивов

Система PHP поддерживает массивы с числовыми индексами и ассоциативные массивы, в качестве индексов которых используется символьная строка. Индексы массивов задаются в квадратных скобках и изменяются начиная с 0:

```
$massiv[0] = 'name';
$massiv[1] = 'phone';
```

Ассоциированные массивы (хеши) используют в качестве индексов строковые значения-ключи.

```
$new_massiv['name'] = 'Valera';
$new_massiv['email'] = 'rvs@bsu.by';
```

Массив может инициализироваться одним из двух способов: последовательным присвоением значений, или посредством конструкции array(). При последовательном добавлении значений в массив, просто записываются значения элементов массива, используя пустой индекс. При этом каждое следующее значение будет добавляться в качестве последнего элемента массива.

```
$massiv[] = "третий"; //$massiv[2] = " третий "
$massiv[] = "четвертый";//$massiv[3] = " четвертый "
```

Второй способ создания массива реализуется путем вызова функции array():

```
$massiv = array('one','two','three');
```

Для ассоциированных массивов такой вызов будет иметь вид:

```
$new_massiv = array('name' => 'nobody', 'email' => 'mail@bsu.by');
```

Между индексом и значением помещается здесь знак =>. Рассмотрим пример:

```
<?php //pr45
$massiv = array('one','two','three');
$massiv[5]="5";
$massiv[6]="6";
$massiv[]="7";
print_r($massiv);
```

```
$new_massiv = array('name' => 'nobody', 'email' => 'mail@bsu.by');
print_r($new_massiv);
print $new_massiv['name']
?>
```

Результат:

```
Array ( [0] => one [1] => two [2] => three
[5] => 5 [6] => 6 [7] => 7 )
Array ( [name] => nobody [email] => mail@bsu.by )
nobody
```

Функция print_r() используется для вывода всего массива.

PHP включает также ряд предопределенных или глобальных массивов. Их называют также суперглобальными переменными, так как они всегда присутствуют и доступны для всех блоков сценария PHP. Ниже показаны обычно используемые суперглобальные переменные PHP:

```
$_GET[], $_POST[], $_REQUEST[], $_COOKIE[], $_FILES[],
$_SERVER[], $_ENV[], $_SESSION[]
```

Массив $_ENV содержит переменные среды окружения, как и в C++. Другие суперглобальные переменные PHP будут описаны в дальнейшем. Следующий пример выполняет вывод значений элементов суперглобального массива $_SERVER

```
<?php //pr46
//print_r($_GET);
//print_r($_POST);
//print_r($_REQUEST);
//print_r($_COOKIE );
//print_r($_FILES);
print_r($_SERVER);
//print_r($_ENV);
//print_r($_SESSION);
?>
```

Результаты:

```
Array
(
 [COMSPEC] => C:\WINDOWS\system32\cmd.exe
 [DOCUMENT_ROOT] => z:/home/localhost/www
 [HTTP_ACCEPT] => image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel, application/msword,
application/vnd.ms-powerpoint, */*
    [HTTP_ACCEPT_ENCODING] => gzip, deflate
    [HTTP_ACCEPT_LANGUAGE] => ru
    [HTTP_CONNECTION] => Keep-Alive
    [HTTP_HOST] => localhost
    [HTTP_USER_AGENT] => Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 2.0.50727)
    [PATH] => \usr\local\ImageMagick;\usr\local\php5;
C:\PROGRA~1\Borland\CBUILD~1\Bin; C:\PROGRA~1\Borland\CBUILD~1\Projects\Bpl;
C:\WINDOWS\system32; C:\WINDOWS; C:\WINDOWS\System32\Wbem
    [REMOTE_ADDR] => 127.0.0.1
    [REMOTE_PORT] => 1071
    [SCRIPT_FILENAME] => z:/home/localhost/www/myprimers/pr44.php
    [SERVER_ADDR] => 127.0.0.1
      [SERVER_ADMIN] => webmaster@localhost
```

```
        [SERVER_NAME] => localhost
        [SERVER_PORT] => 80
        [SERVER_SIGNATURE] => <ADDRESS>Apache/1.3.33 Server at <A
HREF="mailto:webmaster@localhost">localhost</A> Port 80</ADDRESS>
        [SERVER_SOFTWARE] => Apache/1.3.33 (Win32) PHP/5.1.6
        [SystemRoot] => C:\WINDOWS
        [WINDIR] => C:\WINDOWS
        [GATEWAY_INTERFACE] => CGI/1.1
        [SERVER_PROTOCOL] => HTTP/1.1
        [REQUEST_METHOD] => GET
        [QUERY_STRING] =>
        [REQUEST_URI] => /MyPrimers/pr44.php
        [SCRIPT_NAME] => /MyPrimers/pr44.php
        [PATH_TRANSLATED] => z:/home/localhost/www/myprimers/pr44.php
        [PHP_SELF] => /MyPrimers/pr44.php
        [REQUEST_TIME] => 1196893515
        [argv] => Array
        (
        )

        [argc] => 0
       )
```

Операции и выражения

Таблица Арифметические операции

Пример	Результат
$a + $b	Сумма $a и $b.
$a - $b	Разность $a и $b.
$a * $b	Произведение $a и $b.
$a / $b	Частное от деления $a на $b.
$a % $b	Целочисленный остаток от деления $a на $b.

Операция деления ("/") всегда возвращает число с плавающей точкой, даже если операнды являются целыми числами (или строками, которые конвертируются в целые числа).

Битовые операции

Битовые операции дают возможность устанавливать значение битов для целых чисел. Если операнды являются строками, битовая операция выполняется над символами строки.

Таблица. Битовые операции

Пример	Имя	Результат
$a & $b	And	Устанавливаются биты, которые установлены и в $a, и в $b.
$a | $b	Or	Устанавливаются биты, которые установлены в $a или в $b.
$a ^ $b	Xor	Устанавливаются биты, которые установлены в $a

		или $b, но не в обоих.
~ $a	Not	Устанавливаются биты, которые в $a не установлены, и наоборот.
$a << $b	Сдвиг влево	Сдвигает биты переменной $a на $b шагов влево.
$a >> $b	Сдвиг вправо	Сдвигает биты переменной $a на $b шагов вправо.

Операции сравнения

Таблица. Операции сравнения

Пример	**Название**	**Результат**
$a == $b	равно	**TRUE**, если $a равно $b.
$a === $b	идентично	**TRUE**, если $a равно $b и они одного типа.
$a != $b	не равно	**TRUE**, если $a не равно $b.
$a <> $b	не равно	**TRUE**, если $a не равно $b.
$a !== $b	не идентично	**TRUE**, если $a не равно $b или они разных типов.
$a < $b	меньше	**TRUE**, если $a строго меньше $b.
$a > $b	больше	**TRUE**, если $a строго больше $b.
$a <= $b	меньше или равно	**TRUE**, если $a меньше или равно $b.
$a >= $b	больше или равно	**TRUE**, если $a больше или равно $b.

Условной операцией является операция ? формата: (expr1) ? (expr2) : (expr3). При этом значение выражения равно expr2, если expr1=TRUE, и expr3, если expr1= FALSE.

Логические операции

Таблица. Логические операции

Пример	**Имя**	**Результат**
$a and $b	And	**TRUE**, если и $a, и $b **TRUE**.
$a or $b	Or	**TRUE**, если $a или $b **TRUE**.
$a xor $b	Xor	**TRUE**, если $a или $b **TRUE**, но не оба.
! $a	Not	**TRUE**, если $a не **TRUE**.
$a && $b	And	**TRUE**, если и $a, и $b **TRUE**.
$a || $b	Or	**TRUE**, если $a или $b **TRUE**.

Операции and и or выполняются также, как && и || однако они имеют более высокий приоритет.

В качестве false принимается 0, в качестве true не 0. Обычно операции и функции возвращают в качестве true значение 1. Рассмотрим пример:

```
<?php //pr7.php
$b=('2' & '9');
 echo "b=", $b, "<br>";// выдаёт '0'
$b=(2 && 9);
 echo "b=", $b, "<br>";// выдаёт '1'
```

```
$a=(3 | 9);
 echo "a=", $a, "<br>";// выдаёт  11
$c= $a and $b;
echo "c=", $c, "<br>";//выдаёт '11'
echo "c=", $c=$a && $b,"<br>";// выдаёт '1'
echo "c=", $c=3||4, "<br>";//выдаёт '1'
echo "c=", $c=3 or 4, "<br>";//1
?>
```

Строковые операции

Операция “+” используется только для сложения чисел, являющихся значениями строк и не является операцией конкатенации (слияния строк). Операцией конкатенации является операции “.” И ”.=”.

```
<?php //pr4.php
$one = '1';
$two= "2";
echo $one, $two;
print $one+$two; // выведет - 1 2 3
echo $one.$two; // выведет - 12
$one = (int)$one; $two= (float)$two;
print $one+$two;//3
  $a = "Hello ";
  $a .= "World!"; // теперь $a содержит "Hello World!"
?>
```

Вывод:

123123

Переменные $one и $two являются строковыми, при сложении происходит неявное преобразование их значений к числовым. Может быть использовано явное преобразование: $one = (int)$one; $two= (float)$two;

Рассмотрим операторы сравнения строк. В PHP операнды сравниваются, как строки, только в случае, если оба они - строки. В противном случае они сравниваются как числа. При этом любая строка, которую PHP не удается перевести в число, в том числе и пустая, будет восприниматься как 0. Для сравнения строк не рекомендуется использовать операторы сравнения == и !=, поскольку они требуют преобразования типов. Пример:

```
<?php
$x=0;
$y=1;
if ($x == "") echo "<p>x - пустая строка</p>";
if ($y == "") echo "<p>y - пустая строка</p>";
// Выводит:
// x - пустая строка
?>
```

Данный скрипт сообщает нам, что $x - пустая строка. Это связано с тем, что пустая строка ("") трактуется здесь как 0. Чтобы избежать путаницы и преобразования типов, при сравнении строк рекомендуется пользоваться оператором эквивалентности. Оператор эквивалентности сравнивает величины и по значению, и по типу:

```
<?php
$x="string";
$y="string";
$z="Строка";
if ($x === $z) echo "<p>Строка X равна строке Z</p>";
```

```
if ($x === $y) echo "<p>Строка X равна строке Y</p>";
if ($x !== $z) echo "<p>Строка X НЕ равна строке Z</p>";
// Выводит:
// Строка X равна строке Y
// Строка X НЕ равна строке Z
?>
```

Операторы управления

PHP содержит все основные конструкции языка C++: условные операторы if…else (elseif), switch, четыре вида операторов цикла и др.

Оператор выбора **if** имеет следующий синтаксис:

if (boolexp) { */*операторы, выполняемые при boolexp=true*/*} *//1*

else { */*операторы, выполняемые при boolexp=false */* } *//2*

Если выражение **boolexp** принимает значение **true**, то выполняется группа операторов **1**, иначе – группа операторов **2**. При отсутствии оператора **else** операторы, расположенные после окончания оператора **if** (строка 2), выполняются вне зависимости от значения булевского выражения оператора **if**. Допустимо также использование конструкции-лесенки **if {} else if {}**.

В следующем примере показана возможность отключения PHP, чтобы вывести без интерпретации часть html – документа.

```
<?php  //pr8.php
$boolexp=true;
if ($boolexp) :  ?>  //отключение php
<img src="img/A_A.jpg" >
<?php endif;  //включение php
$c=5;
echo "c=", $c=$c or 4, "<br>";//1
?>
```

Оператор if() при этом надо закончить оператором endif (аналогично endwhile, endfor). Создаваемый после PHP-интерпретации HTML- документ, пересылаемый браузеру, будет выглядеть следующим образом:

```
 //отключение php
<img src="img/A_A.jpg" >
c=1<br>
```

Другой вариант состоит в использовании фигурных скобок вместо двоеточия:

```
<?php  //pr81.php
$bexp=true;
if ($bexp) {  ?>  //отключение php
<img src="img/A_A.jpg" >
<?php }  //включение php
$c=5;
echo "c=", $c=$c or 4, "<br>";//1
?>
```

В PHP существует четыре вида циклов, первые три из них аналогичны соответствующим циклам в языке C++:

Цикл с предусловием:

while (boolexp) { */*операторы, выполняемые при boolexp=true */* }

Цикл с постусловием:

do { */*операторы, выполняемые при boolexp=true */* }
while (boolexp);

for(exp1; boolexp; exp3){ */*операторы*/* } *// цикл с параметрами*

Здесь по традиции **exp1** – начальное выражение, **boolexp** – условие выполнения цикла, **exp3** – выражение выполняемое в конце итерации цикла (как правило, это изменение начального значения). Циклы выполняются, пока булевское выражение **boolexp** равно **true**.

Еще один цикл, упрощающий доступ к массивам:

foreach($array as[$key=>] $value){ */*операторы*/* }

При проходе каждого элемента массива в переменную $key помещается индекс данного элемента, а в переменную value – значение элемента. Индекс $key может отсутствовать.

Рассмотрим пример вывода элементов массива:

```
<?php //pr9.php
$massiv = array('one','two','three');
foreach($massiv as $value)echo $value;//onetwothree
?>
```

Аналогично C++ используется оператор выбора варианта switch:

```
switch(exp) {
        case exp1:{/*операторы*/}
                break;
        case expN:{/*операторы*/}
                break;
        default: {/*операторы*/}
}
```

При совпадении значения exp с одним из значений, указанным в case, выполняется соответствующий вариант и далее подряд все блоки операторов до тех пор, пока не встретится оператор break, Значения exp1,…, expN могут быть константами не только целого, но и вещественного или стрингового типа в отличие от C++.

Для выхода из циклов используются оператор прерывания цикла break и оператора прерывания итерации цикла continue.

```
<html>
<head></head>
<body>
<p>в прямоугольной матрице переставить столбцы в порядке возрастания суммы их элементов</p>
<?php //pr10.php
$matrix = array();
$n = 3;
$m = 5;
for($i=0;$i<$n;$i++)
        for($j=0;$j<$m;$j++){
                $matrix[$i][$j] = rand(0,9);
        }
```

Функции

Функция может быть определена с использованием синтаксиса:

```
function fname ($arg_1, $arg_2, ..., $arg_n)
{
 echo "Пример \n";
 return $val; //возвращаемое значение
}
```

Необязательный оператор return возвращает значение любого типа, в том числе список и объект. Как и во многих скриптовых языках в определении функции не нужно указывать тип возвращаемого значения.

```
<?php //pr11.php-найти все натуральные числа, не /превосходящие m
//и содержащие хотя бы одну девятку в десятичном представлении
  $m=rand(10,150); //случайное значение m
  print "m=$m<br>";
  for($i=1;$i<$m;$i++)
  {$k=f($i);
  if($k!=0)print "$k<br>";
  }
  function f($n)
  {$l=$n;
  do
  {
  if(($l%10)==9)return $n;}
  while(($l/=10)!=0);
  return 0;
  }
  ?>
```

Пример. Вычисление площади треугольника по формуле Герона.

```
<html>
<head><title>Square</title></head>
<body>
Стороны треугольника:<br>
<?php //pr12
$a=rand(1,100);
$b=rand(1,100);
$c=rand(1,100);
print "$a<br>$b<br>$c<br>";
Proverka($a,$b,$c);
function Proverka($a,$b,$c)
{if(($a+$b<=$c)||($a+$c<=$b)||($b+$c<=$a))
echo "Треугольника с указанными сторонами не существует!!!!<br>Перезагрузите страницу!<br>";
 else{
echo "Всё правильно! Треугольник с указанными сторонами действительно существует!!!!<br>";
 $s=Geron($a,$b,$c);
echo "Его площадь равна<br> ";
print "$s";
 }
}
function Geron($a,$b,$c){
$p=($a+$b+$c)/2;
 return sqrt($p*($p-$a)*($p-$b)*($p-$c));
}
?>
</body>
</html>
```

Несколько значений можно возвратить путём возвращения списка и функции list().

```
function numbers()
{
 return array (0, 1, 2);
}
list ($zero, $one, $two) = numbers();
```

Операция ссылки & в объявлении функции и в присвоении возвращаемого значения позволяет возвратить ссылку:

```
function &reference()
{
 return $ref;
}
$newref =&reference();
```

Рекурсивные функции

```
<?
// Написать функцию вычисления  x^n
function Evaluate($x,$n)
{
        if ($n==1) return $x;
        else return Evaluate($x,$n-1)*$x;
}
echo("x=5; n=3; reset=");
echo(Evaluate(5,3));
echo("<BR>");
echo("x=0.2; n=3; reset=");
echo(Evaluate(0.2,3));
?>
```

Аргументы функции

Список аргументов представляет набор переменных или констант, разделённых запятыми. PHP поддерживает разбор аргументов функций по значению, разбор по ссылке и значения по умолчанию.

В PHP, как и в C++, аргументы передаются в функцию по значению. Если необходимо дать функции возможность модифицировать аргументы, передавать их надо по ссылке &:

```
function f(&$strike)
{
$strike .= 'and extra.';
}
$strike = 'This is a string, ';
f($strike);
echo $strike;// выводит 'This is a string, and extra.'
```

По ссылке в функцию передается не значение переменной $strike, а ее адрес. После этого изменение переменной $strike в функции влечет изменение аргумента $strike в вызывающей функцию программе.

Функция может определить значения по умолчанию (значение аргументу присваивается в случае, если аргумент не был передан в функцию) в стиле C++ для скалярных аргументов:

```
function f($a = "0")
{
return $a.;
}
```

```
echo f(); // 0.
echo f("1");// 1.
```

При вызове такой функции без аргумента, в качестве аргумента будет взято значение по умолчанию. Значение по умолчанию обязано быть константным выражением, но не переменной. Эти значения должны находиться справа от любых значений не по умолчанию.

Область действия и время жизни переменных

Переменные, объявленные вне функций имеют глобальную область действия. Любая переменная, определенная внутри функции, ограничена локальной областью функции и уничтожается при выходе из функции. Например, такая конструкция работать не будет, т.к. переменная находится в глобальной области, а ссылка на неё находится в теле функции.

```
$perem = 1; /* глобальная область */
function Test ()
 {
 echo $perem;
 }
Test ();
```

А такая конструкция выдаст правильный результат:

```
function Test ()
 {
 $perem = 1; /* локальная область функции */
 echo $perem;
 }
Test ();
```

Чтобы глобальная переменная была доступна внутри функции, её необходимо внутри этой функции задекларировать как глобальную переменную.

```
<?php  //pr18.php
/* глобальная область */
$a = 2;
$b = 3;
function Sum ()
 {
 global $a, $b; /* декларируем переменные $a и $b как глобальные */
 $c = $a + $b; /* и теперь функция имеет к ним доступ */
 echo $c;//5
global $myglobal;
$myglobal=0;
 }
echo ($myglobal) ;//неопределенная переменная $myglobal
Sum ();
echo $myglobal; //0
?>
```

Для доступа к переменным глобальной области _GET[], $_POST[], $_REQUEST[], $_COOKIE[], _FILES[], $_SERVER[], $_ENV[], $_SESSION[], которые называются суперглобальными и доступны постоянно во время выполнения сценария, в PHP используется специальный массив $GLOBALS.

Время жизни локальной переменной – это время работы функции, в которой объявлена переменная. Для удлинения времени жизни локальной переменной используется ее объявление статической переменной. Статическая переменная

существует только в локальной области функции, но не теряет своего значения после выхода из функции. Например:

```
Function Count ()
 {
static $counter = 0; /* декларируем переменную $counter как статическую, при обращении к функции она не будет каждый раз обнуляться */
echo $counter;
 $counter++; }
Count(); //количество обращений к функции
```

Изменяемые (динамические) переменные

Иногда бывает удобно давать переменным изменяемые имена. Такие имена могут изменяться динамически. Например:

```
$a = "hi";
```

Изменяемая переменная берет некое значение и обрабатывает его как имя переменной. В приведенном выше примере значение hi может быть использовано как имя переменной, посредством применения двух записанных подряд знаков доллара, т.е.:

```
$$a="PHP"; /* фактически получается что $hi="PHP" */
print "$a ${$a}"; /* получаем на выходе: hi PHP */
print "$a $hi"; /* или так, что одно и то-же */
```

Внешние библиотечные функции

В этом разделе описываются некоторые из наиболее часто используемых функций для работы с массивами. Более обширный список доступен на Web-сайте PHP.net.

Функции для работы с массивами

Функция count(), sizeof() – используется для подсчета числа элементов в массиве; sort() – сортировка элементов массива; shuffle() – для случайного перемешивания элементов; array_slice($array_name,offset, length) – используется для извлечения части существующего массива. $array_name является именем разрезаемого массива, offset указывает позицию, где будет начинаться разрез, length указывает число элементов, которое будет вырезано. array_merge($arname1, $arname2) – используется для объединения или слияния двух или более массивов. Пример:

```
<?php //pr48
$massiv = array( 'В пятницу вечером после работы
 мама свои начинает заботы',
 'папа залег на диван, как медведь',
 'многое должен ребенок суметь',
 'много проблем сам решить, все успеть');
foreach($massiv as $value) echo $value;//Вывод
sort($massiv);//Сортировка
foreach($massiv as $key=>$value)
 echo ("<li>$key=>$value</li>");//
rsort($massiv);// Сортировка в обратном порядке
for($i=0;$i<count($massiv);$i++){
echo("<br>$massiv[$i]");
}
shuffle($massiv);// Перемешивание
for($i=0;$i<count($massiv);$i++){
echo("<br>$i-$massiv[$i]");
}
$massiv1=array_slice($massiv,2,3); //вырезать три члена, начиная со второго
// и присвоить их massiv1
print_r($massiv1);
```

```
?>
```

Функции для работы со строками.

Приведем несколько распространенных функций: strlen($strike) – возвращает длину строки; ltrim($strike) – удаляет символы разделители типа пробела или табуляции в начале строки; rtrim($strike) – удаляет символы разделители в конце строки; trim($strike) - вырезает пустое пространство в начале и в конце строки. Рассмотрим пример:

```
<?php    //str1.php
//Удаление лишних пробелов по-левому  боку текста:<br />
$strike="    –Текст с лишними пробелами по бокам.-      “;
$strike=ltrim($strike);  //удаление  пробелов слева<br />
echo "$strike", strlen($strike)," <br />";
  $strike=rtrim($strike); //Удаление  пробелов справа<br />
  echo "$strike"," <br />";
$strike="      Еще  Текст  с  пробелами  по  бокам                                  ".
$strike=trim($strike);//Удаление пробелов  слева и справа <br />
  echo "$strike";
?>
```

Вывод:

-Текст с лишними пробелами по бокам.- 89

-Текст с лишними пробелами по бокам.-

Еще Текст с лишними пробелами по бокам -Текст с лишними пробелами по бокам.-

Функция strpos(string $where, string $what[,int $from =0])пытается найти в строке $where подстроку $what и возвращает позицию этой подстроки в строке. Необязательный параметр $from можно задавать, если поиск нужно вести не с начала строки $from, а с другой позиции. Если подстроку найти не удалось, функция возвращает false.

string substr (string $strike, int $start [, int $length]) - возвращает часть строки $strike, начиная с позиции $start и длиной $length. Если $length не задана, то возвращается подстрока от позиции $start до конца строки $strike. Если в $start отрицательное число, то это число является индексом подстроки, отсчитываемым от конца $strike.

string strstr (string $strike, string $needl) - находит в строке $strike первое вхождение строки $needl. Возвращает часть строки $strike от первого вхождения $needle до конца. Если needl не найден, возвращает FALSE. Функция чувствительна к регистру символов. Для поиска без учёта регистра используется функция stristr(). Рассмотрим пример:

```
<?php //str2.php
echo strpos("Hello","el"),"<br/>"; // Выводит 1
echo strstr("Hello","el"),"<br/>";
// чтобы избежать проблем с определением типов  используйте
//операторы тождественных сравнений (===) (!==)
if (strpos("Norway","rwa") !== false)
echo "Строка rwa есть в Norway","<br/>";
//Пример использования substr()
echo $rest = substr("abcdef", 1),"<br/>"; //возвращает "bcdef"
echo $rest = substr("abcdef", 0, 4),"<br/>";//возвращает "abcd"
 $strike = "Programmer";
 echo substr($str,0,2),"<br/>"; // Выводит Pr
 echo substr($str,-3,3),"<br/>"; // Выводит mer
//Пример strstr()
$email = 'user@example.com';
$domain = strstr($email, '@');
```

```
print "$domain, <br/>";  // печатает @example.com
?>
```

Вывод:

```
1
Ello
Строка rwa есть в Norway
Bcdef
Abcd
Pr
Mer
@example.com,
```

Функция strcmp(string $str1, string $str2)

сравнивает две строки посимвольно и возвращает: 0, если строки совпадают; -1, если строка $str1 лексикографически меньше $str2; и 1, если, наоборот, $str1 "больше" $str2. Для функции strcmp() учитывается регистр символов, для strcasecmp(string $str1, string $str2) регистр не учитывается.

Функция strspn( ) возвращает длину первого сегмента строки1, содержащего символы, присутствующие в строке2. Синтаксис функции strspn( ):

int strspn (string строка1, string строка2)

Следующий фрагмент показывает, как функция strspn( ) используется для проверки пароля. Здесь же сравниваются две одинаковые строки:

```
<?  //Str13.php
$string1 = "butter";
$string2 = "Butter";
if ((strcmp($string1, $string2)) == 0)
print "Strings are equivalent!";
else
print "Strings are not equivalent!";
if ((strcasecmp($string1, $string2)) == 0)
print  "Строки совпадают с точностью до регистра!";
$password = "12345";
if (strspn($password, "1234567890") ==strlen($password))
print "Password cannot consist solely of numbers!";
?>
```

Вывод:

```
Strings are not equivalent!
Строки совпадают с точностью до регистра!
Password cannot consist solely of numbers!
```

Функция strcspn( ) возвращает длину первого сегмента строки1, содержащего символы, отсутствующие в строке2. Синтаксис функции:

int strcspn (string строка1, string строка2);

str_replace(string $from, string $to, string $strike)

Заменяет в строке $strike все вхождения подстроки $from (с учетом регистра) на $to и возвращает результат. Исходная строка, переданная третьим параметром, при этом не меняется. Например, вот так мы можем заместить все символы перевода строки на их HTML эквивалент — тэг
:

```
$st=str_replace("\n","<br>\n",$str)
```

string WordWrap(string $str, int $width=75, string $break="\n")

Функция разбивает блок текста $strike на несколько строк, завершаемых символами $break, так, чтобы на одной строке было не более $width букв. Разбиение

происходит по границе слова. Возвращается получившаяся строка с символами перевода строки, заданными в $break. Пример:

```
<?php //str3.php
$strike = "Это текст электронного письма, которое нужно отправить адресату ";
// Разбиваем текст по 20 символов
$strike = WordWrap ($strike, 30, "<br>");
echo $strike;
?>
```

Вывод: Это текст электронного письма,
которое нужно отправить
адресату

string stripcslashes (string strike) - возвращает строку с вырезанными обратными слэшами. Распознаёт С-подобные \n, \r.

strip_tags (string $strike [, string $allowable_tags])

Функция удаляет из строки все тэги и возвращает результат. В параметре $allowable_tags перечисляться вплотную друг к другу тэги, которые не следует удалять из строки. Примеры:

```
<? //str4.php
$strike="<b>жирный текст</b>";
echo "$strike,<br/>";
$strike=strip_tags($strike);
echo "НЕ $strike<br/>";
//Удаление всех тэгов, кроме <b> и <i>:
$strike="<h1>большой текст</h1> <b><i>жирный текст</i></b>";
echo "$strike <br/>";
$strike=strip_tags($strike,"<b><i>");
echo "$strike,<br/>";
$strip = strip_tags ($strike); // Удаляет все - теги из строки
echo "$strip<br/>";
?>
```

Вывод:

жирный текст,
НЕ жирный текст

большой текст
жирный текст
большой текст ***жирный текст***,
большой текст жирный текст

Следующие функции предназначены для быстрого URL-кодирования и декодирования при передаче данных через интернет.

UrlEncode(string $strike)

Функция URL-кодирует строку $strike и возвращает результат. Например:

```
echo "<a href=/script.php?param=".UrlEncode($UserData);
```

UrlDecode(string $st)

Производит URL-декодирование строки. Используется реже, чем UrlEncode(), потому что PHP умеет автоматически декодировать данные.

HtmlSpecialChars(string $strike)

Функция обычно используется в комбинации с echo. Основное ее назначение - гарантировать, что в выводимой строке ни один участок не будет воспринят как тэг.

Заменяет в строке некоторые символы (такие как амперсант, кавычки и знаки "больше" и "меньше") на их HTML-эквиваленты, так, чтобы они выглядели на странице "самими собой". Самое типичное применение этой функции — формирование параметра value в различных элементах формы, чтобы не было никаких проблем с кавычками, или же вывод сообщения в гостевой книге, если вставлять тэги пользователю запрещено.

StripSlashes(string $strike)

Заменяет в строке $strike некоторые предваренные слэшем символы на их однокодовые эквиваленты. Это относится к следующим символам: ", ', \ и никаким другим.

AddSlashes(string $strike)

Вставляет слэши только перед следующими символами: ', " и \. Функцию очень удобно использовать при вызове eval() (эта функция выполняет строку, переданную ей в параметрах, так, как будто имеет дело с небольшой PHP-программой.

Функция strpbrk(string, char) – ищет в строке символ char и возвращает false или строку, начинающуюся с найденного символа; strtoupper(string) – преобразует строку в верхний регистр; strtolower(string) – преобразует строку в нижний регистр; strrev(string) – возвращает строку string в обратном порядке. Следующий блок кода демонстрирует, как использовать строковые функции PHP.

```
<?php//str6.php
$string = "Hello World";
$another_string = "Welcome to PHP";
echo strlen($string);
echo strtoupper($another_string);
echo strrev($another_string);
echo strpbrk($string, "W");
?>
```

Вывод:

```
11
WELCOME TO PHP
PHP ot emocleW
World
```

Первая строка выводит длину строки "Hello World", равную 11. Затем строка "Welcome to PHP" преобразуется в верхний регистр и выводится. Эта строка используется также с функцией strrev для изменения порядка символов строки на обратный. Наконец, функция strpbrk () производит поиск символа "W". Так как первое появление символа происходит в тексте "World", выводится эта строка.

Функции форматных преобразований строк

Язык PHP поддерживает ряд форматных функций вывода.

string sprintf (string $format [, mixed args]) – возвращает строку, созданную из аргументов в соответствии со строкой форматирования $format. Строка $format может включать в себя команды форматирования, предваренные символом %. Все остальные символы копируются в выходную строку. Каждый спецификатор формата соответствует одному параметру, указанному после параметра $format.

```
<? // Пример str7.php Использование sprintf()
$num=5;
$location="tree";
$format = "There are %d monkeys in the %s";
printf($format,$num,$location); // выведет: "There are 5 //monkeys in the tree".
//sprintf(): целые числа с заполнением нулями
$year=3;
$month=10;
```

```
$day=11;
$dat = sprintf(" %03d-%03d-%02d ", $year, $month, $day);
echo $dat;
//sprintf(): форматирование валюты
$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
echo $money;// will output "123.1";
$formatted = sprintf(" %01.2f", $money);
echo $formatted; //выдаст "123.10"
?>
```

Вывод:

```
There are 5 monkeys in the tree 003-010-11 123.1 123.10
```

Функция printf(string $format [, mixed args, ...]) делает то же самое, что и sprintf(), только результирующая строка не возвращается, а направляется в браузер пользователя.

В PHP существует еще несколько функций для форматных преобразований, среди них - sscanf() и fscanf(), которые часто применяются в Си.

mixed sscanf (string strike, string format [, string var1]) - это input-аналог printf(). sscanf() читает из строки strike и интерпретирует её в соответствии с форматом format. Любой пробел в строке формата совпадает с любым пробелом в строке ввода.

```
<? // str8.php Использование sscanf()
$serial = sscanf("SN/23501","SN/%s",&$ser);
// получение серийного номера и даты изготовления
echo $serial,"<br>";
$mandate = "January 01 2011";
list($month, $day, $year) = sscanf($mandate,"%s %d %d");
echo "Item $ser was manufactured on: $year-" .substr($month,0,3)."-$day\n";
// функция возвращает количество присвоенных значений.
?>
```

Вывод:

```
1
Item 23501 was manufactured on: 2011-Jan-1
```

Использование форматных функций в Веб – приложениях ограничено из-за низкой скорости: для разбора строк оказывается выгоднее привлечь регулярные выражения или функцию explode().

Функция array explode (string разделитель, string $strike [, int n]) делит строку на элементы и возвращает эти элементы в виде массива. Разбиение $strike происходит по разделителям, количество фрагментов может ограничиваться необязательным параметром n.

Ее двойник - функция string implode (string разделитель, array фрагменты) - объединяет массив в строку. Формирование массива из строки и строки из массива продемонстрировано в следующем примере:

```
<? // str9.php
$info = "Minsk | baseball | indians";
$user = explode("|", $info);
//$user[0]="Minsk";$user[1]="baseball";$user[2]="Indians";
print_r($user);
$cities=array("Colum","Youngstown","Cleveland","Cincinnati");
$city_string = implode("l", $cities);
//$city_string="Colum1Youngstown1Cleveland1Cincinnati";
echo  "<br>",$city_string;
```

```
?>
```
Вывод:
```
Array ( [0] => Minsk [1] => baseball [2] => indians )
 ColumlYoungstownlClevelandlCincinnati
```

Функция string strtok (string $arg1, string $token) – выполняет лексемизацию строки, в результате которой строка arg1делится на слова (лексемы/tokens), где каждое слово отделено символом из $token. В примере строка "This is an example string", лексемизируется на отдельные слова с применением пробела как разделителя.

```
<? // str10.php
$string = "This is\tan example\nstring";
/* Использовать символы tab и newline как лексемизирующие символы */
$tok = strtok($string," \n\t");
while ($tok) {
 echo "Word=$tok<br>";
 $tok = strtok(" \n\t");
}
?>
```
Вывод:
```
Word=This
Word=is
Word=an
Word=example
Word=string
```

В параметр token можно поместить несколько лексем.

void parse_str (string $strike) - Разбирает strike так, как если бы она была строкой запроса, переданной через URL, и устанавливает переменные.

```
<? // Пример str11.php  Использование parse_str()
$strike = "first=value &second[]=this+works &second[]=another";
parse_str($strike);
echo $first; /* печатает "value" */
echo $second[0]; /* печатает "this works" */
echo $second[1]; /* печатает "another" */
?>
```
Вывод:
```
value this works another
```

getHostByName(domain_name) - преобразует переданное имя домена в IP адрес в формате nnn.nnn.nnn.nnn.

getHostByAddr(ip_address) - Преобразует данный IP адрес в формате nnn.nnn.nnn.nnn в полное имя домена.

Пример. Определяем в какой стране живёт посетитель:

```
<?  //Str12.php
$host = gethostbyaddr($_SERVER['REMOTE_ADDR']);
$countrys=array( ru => Россия, ua => Украина, by => Беларусь );
$array=array_reverse(explode(".",$host));
if(!empty($countrys[$array[0]]))
echo "Ваша страна: ".$countrys[$array[0]];
else echo "Откуда Вы пришли - я не знаю :(";
$ip = gethostbyname('localhost');
echo "<br>",$ip;
```

```
?>
```

Вывод:

```
Откуда Вы пришли - я не знаю :(
127.0.0.1
```

Преобразование строк и файлов к формату HTML и наоборот

Функция nl2br( ) заменяет все символы новой строки (\n) эквивалентными конструкциями HTML
. Синтаксис функции: string nl2br (string $strike)

Функция htmlentities( ) преобразует символы в эквивалентные конструкции HTML. Синтаксис функции: string htmlentities (string $strike)

Функция string htmlspecialchars (string $strike) заменяет некоторые символы, имеющие особый смысл в контексте HTML, эквивалентными конструкциями HTML. Функция htmlspecialchars( ) преобразует следующие символы: & преобразуется в & " " преобразуется в " < преобразуется в < > преобразуется в >.

В частности, эта функция позволяет предотвратить ввод пользователями опасных символов в форумах. Следующий пример демонстрирует удаление потенциально опасных символов функцией htmlspeclalchars( ):

```
<?php  //str14.php
$text= " Мишка косолапый
По лесу идет ";
print $text;
// Преобразовать символы новой строки "\n" в <br>
 $htinl = nl2br($text);
print $htinl;
$input = "The cookbook, entitled Cafe Francaise' costs < $42.25.";
echo  "input:", $input;
$conv = htmlentities($input);
// $conv = "The cookbook, entitled 'Caf&egrave;
// Frac&ccediliaise' costs &lt; 42.25.";
echo "<br> conv:",$conv;
$strike = "A 'quote' is <b>bold</b>";
echo "<br>strike:", $strike,"<br>";
// Outputs: A 'quote' is &lt;b&gt;bold&lt;/b&gt;
echo htmlentities($strike),"<br>";
// Outputs: A &#039;quote&#039; is &lt;b&gt;bold&lt;/b&gt;
echo htmlentities($strike, ENT_QUOTES);
//Функция htmlspecialchars( ) преобразует следующие символы:
//& преобразуется в &amp; " "  в &quot; < преобразуется в &lt; >  в &gt;.
// удаление потенциально опасных символов функцией htmlspeclalchars( ):
$input = "I just can't get of PHP & those fabulous cooking recipes!";
echo  "<br>input:", $input;
$conv = htmlspecialchars($input);
// $conv_input = "I just can't &lt;&lt;enough&gt;&gt; of PHP &amp those fabulous cooking recipes!"
echo "<br> conv:",$conv;
?>
```

Если функция htmlspecialchars( ) используется в сочетании с nl2br( ), то последнюю следует вызывать после htmlspecialchars( ). В противном случае конструкции
, сгенерированные при вызове nl2br( ), преобразуются в видимые символы.

Функция string get_html_translation_table (int таблица) обеспечивает удобные средства преобразования текста в эквиваленты HTML. Функция возвращает одну из двух таблиц преобразования, определяемых параметром таблица, и используемых в работе

стандартных функций htmlspecialchars( ) и htmlentities( ). Параметр таблица принимает одно из двух значений: HTML_ENTITIES или HTML_SPECIALCHARS.

В примере функция используется при преобразовании текста в код HTML:

```
$string = "La pasta e il piatto piu amato in Italia";
$translate = get_html_translation_table(HTML_ENTITIES);
print strtr($string, $translate);
// Специальные символы преобразуются в конструкции HTML
// и правильно отображаются в браузере.
```

Преобразование HTML в простой текст

Функция string strip_tags (string строка [, string разрешенные_теги]), удаляет из строки все теги HTML и PHP. Используется для удаления опасных символов из сообщений на форумах или в гостевых книгах.

Функция get_meta_tags( ) предназначена для поиска в файле HTML тегов META. Синтаксис функции:

```
array get_meta_tags (string имя_файла/URL [, int включение_пути])
```

Применение тегов META:

```
<html>
<head>
<title>PHP Recipes</title>
<meta name="keywords" content="gourmet. PHP, food. code, recipes, chef, programming, web">
<meta name="description" content="PHP Recipes provides savvy readers with the latest in PHP programming and gourmet cuisine!">
<meta name="author" content="WJ Gilmore">
</head>
```

Функция get_meta_tags( ) ищет в заголовке документа теги, начинающиеся словом META, и сохраняет имена тегов и их содержимое в ассоциативном массиве. Ниже продемонстрировано применение этой функции к файлу example.html.

```
<?php
$meta_tags = get_meta_tags("example.html");
print_r $meta_tags;
?>
// Переменная $meta_tags содержит массив со следующей информацией:
// $meta_tags["keywords"] = "gourmet. PHP. food. code, recipes, chef, programming. //Web":
// $meta_tags["description"] = "PHP Recipes provides savvy readers with the latest in PHP
programming and gourmet cuisine";
// $meta_tags["author"] = "WJ Gilmore";
```

Данные тегов META можно извлекать не только из файлов, находящихся на сервере, но и из других URL.

Преобразование строки к верхнему и нижнему регистру

В PHP существует четыре функции, предназначенных для изменения регистра строки: strtolower( ); strtoupper( ); ucfirst( ); ucwords( ).

Функция string strtolower(string $strike) преобразует все алфавитные символы строки к нижнему регистру.

```
$sentence = "COOKING and PROGRAMMING PHP are my TWO favorite pastimes!";
$sentence = strtolower($sentence);
// $sentence= "cooking and programming php are my two favorite pastimes!"
$sentence = "cooking and programming PHP are my two favorite pastimes!";
$sentence = strtoupper($sentence);
// $sentence= "COOKING AND PROGRAMMING PHP ARE MY TWO FAVORITE PASTIMES!"
```

Преобразование к верхнему регистру выполняется функцией string strtoupper ($strike)

Функция ucfirst(string $strike) преобразует к верхнему регистру первый символ.

Функция ucwords(string $strike) преобразует к верхнему регистру первую букву каждого слова в строке.

Хэш-функции

md5(string $strike)

Возвращает хэш-код строки $strike, основанный на алгоритме под названием "MD5 Message-Digest Algorithm". Хэш-код — это просто строка, практически уникальная для каждой из строк $strike. То есть вероятность того, что две разные строки, переданные в $strike, дадут нам одинаковый хэш-код, стремится к нулю.

Если длина строки $strike может достигать нескольких тысяч символов, то ее MD5-код занимает максимум 32 символа.

Для чего нужен хэш-код и, в частности, алгоритм MD5? Например, для проверки паролей на истинность.

Пусть, к примеру, есть система со многими пользователями, каждый из которых имеет свой пароль. Можно, хранить эти пароли в обычном или зашифрованном виде в файле, но это небезопасно. В файле паролей будем хранить не сами пароли, а их (MD5) хэш-коды. При попытке пользователя войти в систему вычислим хэш-код введенного им пароля и сравним его с тем, который записан у нас в файле или базе данных. Если коды совпадут, значит, все в порядке.

Конечно, при вычислении хэш-кода какая-то часть информации о строке $strike безвозвратно теряется. И именно это позволяет нам не опасаться, что злоумышленник, получивший файл паролей, сможет его когда-нибудь расшифровать. Ведь в нем нет самих паролей, нет даже их каких-то связных частей!

Пример использования алгоритма хеширования MD5:

```
<?php
$pass_a = "MySecret";
$pass_b = "MySecret";
//Выводим хеш-код строки MySecret($pass_a) - исходный пароль
echo "<b>Хеш-код исходного пароля '$pass_a':</b><b
style=\"color:green\">".md5($pass_a)."</b><br>";
//Выводим хеш-код строки MySecret($pass_b)-верифицируемый пароль
echo "<b>Хеш-код верифицируемого пароля '$pass_b':</b><b
style=\"color:green\">".md5($pass_b)."</b><br>";
// Сравниваем хеш-коды MD5 исходного и верифицируемого пароля
echo "<h3>Проверяем истинность введенного пароля:</h3>";
if(md5($pass_a)===md5($pass_b))
 echo "<h3 style=\"color:green\">Пароль верный! (Хеш-коды совпадают)</h3>";
else echo "<h3 style=\"color:red\">Пароль неверный! (Хеш-коды не
совпадают)</h3>"
// выводит: Пароль верный! (Хеш-коды совпадают)
// Попробуйте изменить значение строки $pass_b :
?>
```

Функция crc32() вычисляет 32-битную контрольную сумму строки $strike. Результат ее работы — 32 битное (4-байтовое) целое число. Эта функция работает гораздо быстрее md5(), но в то же время выдает гораздо менее надежные "хэш-коды" для строки.

Установка локальных настроек

Локалью будем называть совокупность таких настроек системы, как формат даты и времени, язык, кодировка. Для установки локали используется функция SetLocale():

SetLocale(string $category, string $locale)

Параметр $category может принимать следующие строковые значения:

LC_CTYPE — активизирует указанную локаль для функций перевода в верхний/нижний регистры; LC_NUMERIC — активизирует локаль для функций форматирования дробных чисел, задает разделитель целой и дробной части в числах; LC_TIME — задает формат вывода даты и времени; LC_ALL — устанавливает все вышеперечисленные режимы.

Параметр $locale задает локаль, по установленному в системе уникальному имени, по которому к ней можно обратиться. Именно оно и фиксируется в этом параметре. Если величина $locale равна пустой строке "", то устанавливается та локаль, которая указана в глобальной переменной окружения, если в этом параметре передается 0, то новая локаль не устанавливается, а просто возвращается имя текущей локали для указанного режима.

Часто встречается ситуация, когда нам требуется преобразовать строку из одной кодировки кириллицы в другую. Например, мы в программе сменили локаль: была кодировка windows, а стала — KOI8-R. Но строки-то остались по-прежнему в кодировке WIN-1251, а значит, для правильной работы с ними нам нужно их перекодировать в KOI8-R. Для этого и служит функция преобразования кодировок.

convert_cyr_string(string $strike, char $from, char $to);

Функция переводит строку $strike из кодировки $from в кодировку $to. Конечно, это имеет смысл только для строк, содержащих "русские" буквы, т. к. латиница во всех кодировках выглядит одинаково. Разумеется, кодировка $from должна совпадать с истинной кодировкой строки, иначе результат получится неверным. Значения $from и $to — один символ, определяющий кодировку:

k — koi8-r; w — windows-1251; i — iso8859-5; a — x-cp866; d — x-cp866;

Регулярные выражения

Одним из способов поиска данных являются механизмы поиска по шаблону. Базовым средством для поиска по шаблону являются регулярные выражения - последовательность символов, применяемых для поиска искомого текста.

Простейшее регулярное выражение совпадает с одним или несколькими символами строки. Квадратные скобки ([ ]) означают “любой символ из перечисленных в скобках”. Ниже перечислены некоторые интервалы, с помощью знака -. Конструкция [^a-zA-Z] совпадает с любым символом, не входящим в указанные интервалы (a-z и A-Z).

Служебный символ . (точка) означает <любой символ>. Например, выражение p.p совпадает с символом p, за которым следует произвольный символ, после чего опять следует символ p.

( ) – Логическая группировка выражений, которая (может)+ повторяться.

Иногда требуется найти служебные символы в строках вместо того, чтобы использовать их в описанном специальном контексте. В этом случае служебные символы экранируются обратной косой чертой (\).

Perl считается одним из лучших языков обработки текстов. Разработчики PHP сделали синтаксис регулярных выражений Perl доступным для пользователей PHP. Регулярные выражения из нескольких символов в стиле Perl записываются в косых скобках. Рассмотрим простой пример: /stud/. Если записать /stud/i, поиск осуществляется без учета регистра. /^stud/i означает, что stud должно находиться в начале слова (student), а /stud$/ - в конце (restud). Регулярное выражение /^$/i соответствует пустой строке.

Квадратные скобки /[абвг]/ означают “любой один символ а,б,в или г из перечисленных в скобках”. Для создания регулярных выражений могут быть использованы интервалы. Ниже перечислены некоторые интервалы, задаваемые с помощью знака -:

/[0-9]/ - совпадает с любой десятичной цифрой от 0 до 9;

/[a-z]/ - совпадает с любым символом нижнего регистра от a до z;

/[A-Z]/ - совпадает с любым символом верхнего регистра от A до Z;

/[a -Z]/ - совпадает с любым символом нижнего или верхнего регистра от a до Z.

Специальные символы представляет собой алфавитный символ с префиксом \ - признаком особой интерпретации следующего символа:

\d - обозначает любую десятичную цифру;

\D - обозначает любой символ кроме десятичной цифры;

\w - алфавитно-цифровой символ

\W - символ, не являющийся алфавитно-цифровым

\s - пробельный символ

\S - символ не являющийся пробельным

\ . – Шаблону соответствует знак точки в строке.

\n – Символ перевода строки. Строка\n Еще строка

\r – Символ возврата каретки. Текст\r

\t – Символ табуляции. \tКрасная строка

\\ – Обратный слеш

Еще один символ \b, совпадает с границами слов: /sa\b/. Противоположный символ, \B, совпадает с чем угодно, кроме границы слова: /sa\B/

Символы +, * и {...}, обозначающие количество повторений отдельного символа или конструкции, заключенной в квадратные скобки, называются квантификаторами.

Шаблон /stu+/ совпадает с последовательностью stu, за которой могут следовать дополнительные символы u. Рассмотрим другой пример использования квантификатора: /st{2,4}/. Этот шаблон совпадает с символом s, за которым следуют от 2 до 4 экземпляров символа t.

Конструкция [^a-zA-Z] совпадает с любым символом, не входящим в указаные интервалы (a-z и A-Z).

Служебный символ . (точка) означает любой символ поэтому для поиска точки ее надо экранировать символом \.

Например, выражение /[\d]+/ используется для поиска цифровой подстроки.

Шаблон /<([\w]+)>/ совпадает с конструкциями, заключенными в угловые скобки, - например, тегами HTML.

Подстроки в регулярных выражениях можно группировать при помощи круглых скобок: /домен – (by|ru|uk|com)/ соответствует строке домен – by или другой.

| – подобен оператору || (OR) в логическом выражении. Оператор | (или) проверяет совпадение одной из нескольких альтернатив. ( ) – Логическая группировка выражений, которая (может)+ повторяться.

Perl-совместимые функции для работы с регулярными выражениями

В PHP существует пять функций поиска по шаблону с использованием Perl-совместимых регулярных выражений: preg_match( ); preg_match_all( ); preg_replace( ); preg_split( ); preg_grep( ).

preg_match(pattern, $strike, [regs]) – ищет в строке $strike соответствия с регулярным выражением pattern , и сохраняет их в массиве regs (если указано). Пример:

```
<?php
$sub = "abcdef";
$pattern = '/^def/';
preg_match($pattern, $sub, $matches);
print_r($matches);
echo"<br/>";
$pattern = '/def$/';
preg_match($pattern, $sub, $matches);
```

```
print_r($matches);
?>
```

Вывод:

```
Array()
Array ( [0] => def )
```

Функция preg_replace( ) может использовать регулярные выражения в обоих параметрах, шаблон и замена. Синтаксис функции preg_replace( ):

mixed preg_replace (mixed шаблон, mixed замена, mixed строка [, int порог])

Необязательный параметр порог определяет максимальное количество замен в строке. Параметры шаблон и замена могут представлять собой масивы. Функция preg_replace( ) перебирает элементы обоих массивов и выполняет замену по мере их нахождения.

```
<?php
$strike="Mersedes is a good car ever created!";
// Выполнить поиск слова  без учета регистра :
if (preg_match("/mers/i", $strike, $match)!=0)
{
print_r($match);
}
$strike = "регулярное выражение"; // просто строка
$preg = preg_replace("/p.+e/i", "<i>[вырезано]</i>", $strike);
echo "<br/>".$preg;
echo"<br/>";
$string = 'April 15, 2010';
$pattern = '/(\w+) (\d+), (\d+)/i';
$replacement = '${1}1,$3';
echo preg_replace($pattern, $replacement, $string);
?>
```

Результат выполнения:

```
Array([0]=>Mers)
[вырезано]
April1,2010
```

Функция preg_match_all( ) находит все совпадения шаблона в заданной строке. Синтаксис функции :

int preg_match_all (string шаблон, string строка, array совпадения [, int порядок])

Порядок сохранения в массиве совпадения текста, совпавшего с подвыражениями, определяется необязательным параметром порядок. Этот параметр может принимать два значения:

* PREG_PATTERN_ORDER - используется по умолчанию, если параметр порядок не указан. Порядок, определяемый значением PREG_PATTERN_ORDER, на первый взгляд выглядит не совсем логично: первый элемент (с индексом 0) содержит массив совпадений для всего регулярного выражения, второй элемент (с индексом 1) содержит массив всех совпадений для первого подвыражения в круглых скобках и т. д.;

* PREG_SET_ORDER - порядок сортировки массива несколько отличается от принятого по умолчанию. Первый элемент (с индексом 0) содержит массив с текстом, совпавшим со всеми подвыражениями в круглых скобках для первого найденного совпадения. Второй элемент (с индексом 1) содержит аналогичный массив для второго найденного совпадения и т. д.

Следующий пример показывает, как при помощи функции preg_match_all( ) найти весь текст, заключенный между тегами HTML:

```
<?php
```

```
preg_match_all("/\(?  (\d{3})?  \)?  (?(1)  [\-\s] ) \d{3}-\d{4}/x",  "Call 555-1212 or 1-800-555-1212", $phones);
print_r ($phones);
echo "<br/>";
$html = "<b>bold text</b><a href=howdy.html>click me</a>";
preg_match_all("/(<([\w]+)[^>]*>)(.*?)(<\/\\2>)/", $html, $matches, PREG_SET_ORDER);
foreach ($matches as $val) {
   echo "matched: " . $val[0] . "\n";
   echo "part 1: " . $val[1] . "\n";
   echo "part 2: " . $val[2] . "\n";
   echo "part 3: " . $val[3] . "\n";
   echo "part 4: " . $val[4] . "\n\n";
}
echo "<br/>";
$userinfo = "Name: <b>Romanchik Valery</b> <br> Title: <b>PHP Teacher</b>";
preg_match_all ("/<b>(.*)<\/b>/U", $userinfo, $pat_array);
print_r ($pat_array);
print "<br/>".$pat_array[0][0]." <br> ".$pat_array[0][1]."\n";
?>
```

Результат:

```
Array ( [0] => Array ( [0] => 555-1212 [1] => 800-555-1212 ) [1] => Array ( [0] => [1] => 800 ) )
matched: bold text part 1: part 2: b part 3: bold text part 4:
matched: click me part 1: part 2: apart3:click me part4:
Array ( [0] => Array ( [0] => Romanchik Valery [1] => PHP Teacher ) [1] => Array ( [0] => RomanchikValery[1]=>PHPTeacher))
RomanchikValery
PHP Teacher
```

Функция array preg_split (string шаблон, string $strike[, int порог [, int флаги]]) разбивает строку $strike в массив посредством регулярного выражения. Необязательный параметр порог определяет максимальное количество элементов, на которые делится строка. В следующем примере функция preg_split( ) используется для выборки информации из переменной.

```
<?php
$inf="+Romanchik+++VS+++++@gmail.com+++++bsu.by";
$fields = preg_split("/[\+]{1,}/", $inf);
print_r ($fields);
$i=0;
while ($i < sizeof($fields)){
print $fields[$i]. "<br/>";
echo $i++."<br/>";
}
?>
```

Результат:

```
Array([0]=>[1]=>Romanchik[2]=>VS[3]=>@gmail.com[4]=>bsu.by)
0
Romanchik
1
VS
2
```

@gmail.com
3
bsu.by
4

Функция array preg_grep (string $pattern, array $mas) перебирает все элементы заданного массива и возвращает в виде массива все элементы, в которых совпадает заданное регулярное выражение. Пример использования функции для поиска в массиве слов, начинающихся на p:

```
<?php
$foods = array ("pasta", "steak", "fish", "potatoes");
// Поиск элементов, начинающихся с символа "p".
// за которым следует один или несколько символов
$pfoods = preg_grep("/p(\w+)/", $foods);
print_r( $pfoods);
echo "<br/>";
$i = 0;
while ($i < sizeof($foods)) {
print  $pfoods[$i]. "<br>";
$i++;}
?>
```

Результат:

Array([0]=>pasta[3]=>potatoes)
pasta
Potatoes

Регулярные выражения - мощный инструмент для поиска и организации данных, как, например, проверка адреса электронной почты на корректность или поиск URL. Но в то же время регулярные выражения работают медленнее других функций PHP. Например, для перевода целой строки в заглавные буквы можно написать следующее:

```
<?
$data = ereg_replace ("[a-z]", "[A-Z]", $data);
print $data;
$data = strtoupper ($data);
?>
```

Медленная функция ereg_replace() потратит много времени на задачу, с которой функция strtoupper() справилась бы быстрее. Всегда следует искать более "лёгкую" замену регулярным выражениям, поскольку скорость выполнения вашего скрипта в таких случаях резко возрастает. Еще один способ замедлить выполнение Веб – приложения состоит в использовании функций форматированного ввода – вывода типа printf(), sscanf().

Функции даты и времени

Функции позволяют форматировать отметку времени для применения в запросах базы данных или браузера. PHP включает следующие функции даты и времени:

date(format) – возвращает текущее время сервера, форматированное согласно заданному множеству параметров format.

checkdate(month, day, year) – проверяет заданную дату. Успешная проверка означает, что год year находится между 0 и 32767, месяц month – между 1 и 12, и правильное количество дней каждого месяца.

time() – возвращает текущее время сервера, измеренное в секундах начиная с 1 января 1970 г.

Следующая страница использует функцию date() для определения и вывода текущего времени сервера и даты:

```
<?php
echo"<span style='font:10pt arial'>Today is date('lFjY') </span>";
echo "<br/>";
echo "<span style='font:10pt arial'>The current time is: date('g:i:s a')</span>";
?>
```

Формат даты/времени, выводимый с помощью функции date(), зависит от типов параметров формата, подставленных в функцию. Параметры функции date() можно объединять, разделяя запятой, двоеточием или другими знаками пунктуации, в зависимости от желаемого формата вывода. Все параметры, однако, должны быть заключены в одиночные кавычки. В примере выше время выводится с помощью параметров формата времени g, i, s, и a. Двоеточия и пробелы также вставляют для разделения часов, минут, секунд и признаков am/pm. Функции checkdate() и time() обычно используются в процессах принятия решений.

Математические функции

- abs() — Модуль числа
- acos() — Arccos, asin() — Arcsin, atan() — Arctan
- ceil() — Округляет дробь в большую сторону
- cos() — Cos
- exp() — Exp
- floor() — Округляет дробь в меньшую сторону
- fmod() — Возвращает дробный остаток от деления
- log10() — десятичный логарифм, log() — натуральный логарифм
- max() — наибольшее значение, min() — наименьшее значение
- pi() — Возвращает число Пи
- pow() — степенное выражение
- rand() — Генерирует случайное число
- round()— Округляет число типа float
- sin() — Sin
- sqrt() — квадратный корень
- srand() — Изменяет начальное число генератора псевдослучайных чисел
- tan() — Тангенс

Объектно-ориентированное программирование в PHP

Основные понятия ООП

ООП – методология программирования, основанная на представлении программ в виде совокупности объектов, каждый из которых является экземпляром конкретного класса.

Объект – обладающий именем набор данных (полей объекта), физически находящихся в памяти компьютера, и методов, имеющих доступ к ним и выполняющих операции над ними. Имя объекта используется для доступа к полям данных и методам, составляющим объект. Объект является экземпляром определенного класса. В классе дается обобщенное описание некоторого набора реально существующих объектов. Объектно-ориентированное программирование основано на принципах:

- абстрагирования данных;
- инкапсуляции;
- наследования;
- полиморфизма;
- «позднего связывания».

Инкапсуляция (encapsulation) – принцип, объединяющий данные и код, манипулирующий этими данными, а также защищающий данные от прямого внешнего доступа и неправильного использования. Другими словами, доступ к данным класса возможен только посредством методов этого же класса.

Наследование (inheritance) – это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства объекта - предка и добавлять к ним свойства и методы, характерные только для него. Наследование бывает двух видов:

линейное – класс имеет один и только один суперкласс (предок);

множественное – класс может иметь любое количество предков.

Полиморфизм (polymorphism) – механизм, использующий одно и то же имя метода для решения двух или более похожих, но несколько отличающихся задач.

Целью полиморфизма, применительно к ООП, является использование одного имени для задания общих для класса действий. В более общем смысле, концепцией полиморфизма является идея "один интерфейс, множество методов".

Механизм «позднего связывания» в процессе выполнения программы определяет принадлежность объекта конкретному классу и производит вызов метода, относящегося к классу, объект которого был использован.

Классы и Объекты

Модель объектов PHP больше похожа на модель объектов Java, чем на модель C++. Класс представляет объединение переменных и функций-методов, работающих с этими переменными. Переменные класса определяются с ключевым словом var как в PHP4 или со спецификатором доступа private, public или protected как в PHP5. Методы определяются в классе с ключевым словом function и спецификатором доступа. Рассмотрим пример:

```
<?php
class Mycart
{//var $items=0; в PHP4
 private $items=0; //переменные класса в PHP5
 public function add ($n) // Добавить $n артикулов
 {
 $this->items += $n;
 }
public function remove ( $n) // Изъять $n артикулов
 {
 if ($this->items > $n) {
 $this->items -= $n;
 return true; }
else { return false; }
 }
public function show()
{
echo '$items=', $this->items ;
}
}//конец класса
$instance=new Mycart();
$instance->add(5);
$instance->remove(2);
$instance->show();
echo $instance->items ;//ошибка
?>
```

Вывод:

$items=3

Fatal error: Cannot access private property Mycart::$items in x:\home\localhost\www\my\pr41.php on line 25 – нет доступа к private переменной items

Здесь $instance представляет ссылку на объект класса, сам объект создается операцией new. Каждый объект получает собственный числовой идентификатор (handler), который используется при обращении к объекту.

Чтобы иметь возможность доступа к переменным и функциям внутри класса, можно использовать псевдопеременную $this, которая может читаться как 'ссылка на текущий объект'.

В PHP 5 введены спецификаторы доступа public, protected и private, которые позволяют указать степень доступа к свойствам и методам класса.

К общедоступным (public) свойствам и методам можно получить доступ без каких либо ограничений. Защищенные (protected) элементы класса доступны внутри класса, в котором они объявлены, и в производных (наследуемых) от него классах.

Частные (private) элементы доступны только методам в классе, в котором они объявлены.

Если не указывать ни один из спецификаторов, то по умолчанию элемент будет иметь уровень доступа public. Такой же уровень доступа по умолчанию получают свойства, для объявления которых использовалось устаревшее и не рекомендуемое к использованию в PHP 5 ключевое слово var, а спецификатор доступа не используется. Это сделано для переемственности с PHP4.

Конструкторы и деструкторы

В PHP допустимы только константные инициализаторы для переменных класса. Для инициализации переменных класса используется функция инициализации, которая вызывается автоматически при создании объекта класса. Такая функция называется конструктором. Метод-конструктор вызывается автоматически при каждом создании объекта. В PHP4 имя этой функции совпадает с именем класса. В PHP 5 конструктором класса является метод __construct(). При уничтожении объекта вызывается специальный метод __destruct() – деструктор класса.

```
<?php //pr21
 class MyClass {
private $property;
function _construct() {
 echo "Запущен конструктор";
 }
 function __destruct() {
 echo "Запущен деструктор";
 }
function  MyClass($n=1) {
echo "Запущен другой конструктор";
$this->property=$n;
}
}
 $obj = new MyClass(); // Выводит "Запущен конструктор"
 unset($obj); // Выводит "Запущен деструктор"
$obj1 = new MyClass(10);
?>
```

Запущен конструктор Запущен деструктор Запущен конструктор Запущен деструктор

Конструкторы PHP4, имена которых совпадают с именами классов, будут работать с PHP5 без изменений кода. Для совместимости с предыдущей версией PHP 5 поступает следующем образом: если при создании объекта в классе не найдется конструктора __construct(), то PHP пытается выполнить метод, имя которого совпадает с именем класса.

В приведенном выше примере конструктор не перегружается, а вызывается первый конструктор.

В случае наследования классов конструктор базового класса при создании порожденного класса неявно не вызывается. Если необходимо вызвать конструктор или деструктор базового класса из порожденного класса, это нужно сделать явно, через указатель parent.

```
<?php //pr22
 class MyClass {
 function __construct() {
 echo "конструктор базового класса";
 }
 function __destruct() {
 echo " деструктор базового класса";
 }
 }
 class DerivedClass extends MyClass {
 function __construct() {
echo "конструктор порожденного класса";
 parent::__construct();
 }
 function __destruct() {
 echo " деструктор порожденного класса";
 parent::__destruct();
 }
 }
 $obj = new DerivedClass();
unset($obj);
?>
```

Вывод:

конструктор порожденного класса конструктор базового класса
деструктор порожденного класса деструктор базового класса

Рассмотрим еще один пример вывода даты/времени:

```
<?php
class Dateclass {
var $month=array( "Январ", "Феврал", "Март", "Апрел", "Ма", "Июн",
"Июл","Август","Сентябр", "Октябр", "Декабр", "Январ");
 var $day=array ( "Воскресенье", "Понедельник", "Вторник","Среда", "Четверг" ,
"Пятница" , "Суббота");
var $dnum; var $mnum; // Переменные класса
 var $daym; var $year;
function Dateclass() { //Конструктор: инициализация переменных
$this->dnum = date("w");
$this->mnum = date("n");
$this->daym = date("d");
$this->year = date("Y");
}
```

```
function show(){// метод
$dnum =$this->dnum;
$mnum = $this->mnum;
$daym =$this->daym;
$year = $this->year;
$textday =$this->day[$dnum];
$monthm =$this->month[$mnum-1];
if ($mnum==3||$mnum==8)
{ $k="а"; }
else {$k="я";}
echo "Сегодня:  $textday, $daym $monthm$k $year г.";
}
}
$obj=new Dateclass();// Создание объекта
$obj->show();//Вызов метода из объекта
?>
```

Результат:

Сегодня: Воскресенье, 05 Апреля 2009 г.

В PHP объекты представляют объектные ссылки, передача объекта в качестве параметра функции происходит по ссылке, а не по значению. Если необходимо провести копирование объекта, то используется клонирование объекта со всеми методами, свойствами и значениями:

```
<?php //pr24
 class MyClass{
 public $property=1;
 }
 $obj1 = new MyClass;
 $obj2=$obj1;//ссылка на тот же объект
 echo $obj1->property; // Выводит 1
 $obj2->property = 2;
 echo $obj1->property; // Выводит 2
 $obj3 = clone $obj1;//создание нового объекта
echo $obj3->property; // Выводит 2
$obj3->property = 3;
echo $obj1->property; // Выводит 2
?>
```

Для копирования объекта используется ключевое слово clone, которое вызывает метод __clone() и к которому нельзя обратиться непосредственно. Метод __clone() необязательно описывать в классе, однако его перегрузка, позволит изменить значения свойств копируемого объекта:

```
<?php
 class MyClass{
        var $property;
        function __clone() {
        $this->property = 2;
        }
        }
        $obj1 = new MyClass;
        $obj1->property = 1;
        $obj2 = clone $obj1;
```

```
echo $obj1->property; // Выводит 1
echo $obj2->property; // Выводит 2
?>
```

Метод __clone() не может принимать никакие аргументы, однако позволяет обратиться к исходному объекту через указатель $this и получаемому в результате копирования объекту через указатель $that.

Наследование классов и интерфейсов

Класс может быть наследником только одного суперкласса и множества интерфейсов. Рассмотрим пример

```
<?php
class MyClass extend MySuperClass implements I1,I2,I3{
//реализация класса
}
?>
```

В вершине иерархии наследования обычно размещают абстрактные базовые классы, содержащие абстрактные методы и объявленные как абстрактные. Абстрактные методы имеют только объявление и не имеют реализации.

```
<?php
abstract class MySuperClass {
    abstract public function abstrFunc();
    }
    class MyClass extends MySuperClass {
    public function abstrFunc() {
    echo “Be-Be-Be”;
    }
    }
    $obj = new MyClass;
    $obj->abstrFunc(); // Выводит Be-Be-Be
?>
```

Невозможно создать объект абстрактного класса, можно только определять производные классы от базового абстрактного класса и создавать объекты уже от производных классов. Стоит отметить, что абстрактные классы также могут содержать и обычные (не абстрактные) методы. Класс не может быть порожден от нескольких классов, в том числе и абстрактных, но зато может быть создан на основе любого числа интерфейсов.

Интерфейсами (interface) являются абстрактные классы, содержащие только абстрактные методы и вообще не имеющие никаких свойств. При этом в интерфейсе методы объявляются ключевым словом function без указания каких-либо спецификаторов, в том числе и abstract и нет тела функции.

```
<?php
interface Int1 {
    function func1();
    }
    interface Int2 {
    function func2();
    }
    class MyClass implements Int1, Int2 {
    public function func1() {
    echo 1;
    }
    public function func2() {
```

```
echo 2;
}
}
$obj = new MyClass;
$obj->func1(); // Выводит 1
$obj->func2(); // Выводит 2
?>
```

Таким образом, хотя множественное наследование классов и не поддерживается, однако можно создавать классы на основе нескольких интерфейсов.

В PHP 5 введена возможность определять методы класса и сами классы как финальные (final). Метод, при определении которого используется ключевое слово final, не может быть переопределен в классах, производных от данного класса.

```
<?php
class MyClass {
final public function func() {
// Код метода
}
}
class MyClass1 extends MyClass {
// Следующий код вызывает ошибку
// переопределения финального метода
// базового класса MyClass
public function func() {
// Код метода
}
}
?>
```

Кроме этого, если final используется при определении самого класса, то порождение от него других классов становится невозможным.

```
<?php
final class MyClass {
// Код описания класса
}
// Следующий код вызывает ошибку
// порождения от финального класса
class MyClass1 extends MyClass {
// Код описания класса
}
?>
```

Если класс определен как final, то и все методы данного класса автоматически становятся финальными, таким образом, определять их явно как final уже нет необходимости.

В отличие от Java, определять свойства класса как финальные константы в PHP недопустимо. В PHP введен новый элемент класса – константа класса.

```
<?php
class MyClass {
const CONSTANT = "константа класса";
}
echo MyClass::CONSTANT; // Выводит "константа класса"
?>
```

В PHP 5 возможно объявление статических свойств класса.

```
<?php
class MyClass {
 static $static = 1;
 }
 echo MyClass::$static; // Выводит 1
?>
```

В PHP используются статические свойства и методы классов. Статические свойства едины для всего класса и не могут принадлежать ни одному из объектов класса. Изменение такого свойства в одном из методов любого объекта приводит к его изменению для всех остальных объектов данного класса. Кроме этого, становится возможным обратиться к такому свойству из класса вне контекста объекта.

Статические методы классов, также как и статические свойства, принадлежат всему классу в целом. Это позволяет использовать такой метод без создания объекта такого класса.

```
<?php
 class MyClass {
 static function statFunc() {
 echo "статический метод";
 }
 }
 MyClass::statFunc(); // Выводит "статический метод"
?>
```

Однако в статическом методе становится невозможным использовать указатель $this, так как при вызове статического метода неизвестно в контексте какого объекта он вызывается, или такого объекта может и не существовать.

Специальное ключевое слово instanceof в PHP 5 позволяет определять является ли объект экземпляром определенного класса, или же экземпляром класса производного от какого-либо класса.

```
<?php
 class MyClass { }
 $obj1 = new MyClass();
 if ($obj1 instanceof MyClass) {
 echo "\$obj1 - объект класса MyClass";
 }
  class MyClass1 extends MyClass { }
 $obj2 = new MyClass1();
 if ($obj2 instanceof MyClass) {
 echo "\$obj2 - объект класса, производного от MyClass";
 }
  interface Int { }
 class MyClass2 implements Int { }
 $obj3 = new MyClass2();
 if ($obj3 instanceof Int) {
 echo "\$obj3 - объект класса, созданного на основе интерфейса Int";
 }
?>
```

Также с помощью instanceof можно определить является ли объект экземпляром класса, созданного на основе определенного интерфейса.

В PHP 5 введена возможность разыменования (dereferencing) объектов, которые возвращаются функциями.

```
<?php
```

```
class MyClass1 {
public function showClassName() {
echo "объект класса MyClass1";
}
}
class MyClass2 {
public function showClassName() {
echo "объект класса MyClass2";
}
}
function deref($obj) {
switch ($obj) {
case "MyClass1":
return new MyClass1();
case "MyClass2":
return new MyClass2();
}
}
deref("MyClass1")->showClassName(); // Выводит "объект
// класса MyClass1"
deref("MyClass2")->showClassName(); // Выводит "объект
// класса MyClass2"
?>
```

Данный механизм позволяет вызывать методы объектов, имена классов которых возвращаются пользовательскими функциями.

В PHP 5 имеется возможность производить уточнения типов классов (class type hints), которые передается методам в качестве параметров.

```
<?php
interface Int1 {
function func1(Int1 $int1);
}
interface Int2 {
function func2(Int2 $int2);
}
class MyClass implements Int1, Int2 {
public function func1(Int1 $int1) {
// Код метода
}
public function func2(Int2 $int2) {
// Код метода
}
}
$obj1 = new MyClass;
$obj2 = new MyClass;
$obj1->func1($obj2);
$obj1->func2($obj2);
?>
```

При этом уточнение типов классов производится не при компиляции, а только на этапе исполнения.

Магические методы

Функции, *__construct, __destruct, __call, __callStatic, __get, __set, __isset, __unset, __sleep, __wakeup, __toString, __invoke, __set_state* , *__clone* , имена которых начинаются с символа “_“ , являются магическими в PHP классах и имеют особую функциональность.

Методы доступа к свойствам объектов __get($name) и __set($name,$value) позволяют легко проводить динамическое назначение свойств объектам. В качестве параметров этим методам передаются имена свойств класса.

Метод __set() также получает и значение, которое устанавливается для свойства. Методы __get() и __set() вызываются только в том случае, если требуемого свойства вообще нет в классе.

```
<?php
 class MyClass {
 private $properties;
 function __set($name, $value) {
 echo "задание нового свойства $name = $value";
 $this->properties[$name]=$value;
 }
 function __get($name) {
 echo "чтение значения свойства ", $name;
 return $this->properties[$name];
 }
 }
 $obj = new MyClass;
 $obj->property = 1; // Выводит "задание нового свойства property=1"
 $a = $obj->property; // Выводит "чтение значения свойства property"
 echo $a; // выводит 1;
?>
```

При вызове в PHP 5 несуществующего метода объекта автоматически вызывается специальный метод __call().

```
<?php
 class MyClass {
 function __call($name, $params) {
 echo "Вызван метод $name с параметром $params[0]";
 }
 }
 $obj = new MyClass;
 echo $obj->method(1); // Выводит "Вызван метод method
 // с параметром 1"
?>
```

В качестве параметров __call() принимает имя вызываемого метода и передаваемые этому методу параметры.

В PHP 5 псевдо-константа __METHOD__ возвращает имя класса и вызываемый метод.

```
<?php
 class MyClass {
 public function myMethod() {
 echo "вызов метода ", __METHOD__;
 }
 }
 $obj = new MyClass;
```

```
$obj->myMethod();//Выводит "вызов метода MyClass::myMethod"
function myFunction() {
echo "вызов функции ", __METHOD__;
}
myFunction(); // Выводит "вызов функции myFunction"
?>
```

При обращении к функции вне класса __METHOD__ возвращает только имя функции.

В PHP 5 введен еще один специальный метод класса - __toString().

```
<?php
class MyClass {
function __toString() {
return "вызван метод __toString()";
}
}
$obj = new MyClass;
echo $obj; // Выводит "вызван метод __toString()"
?>
```

Метод класса __toString() позволяет выполнить перегрузку преобразования объекта в строку.

Обработка ошибок

Начиная с PHP 5 введена современная схема обработки *исключений*. Конструкция **try/catch/throw** позволит весь код обработки ошибок локализовать в одном месте сценария. Рассмотрим пример:

```
<?php
try {
$fp = @fopen("file.txt", "w");
        if (!$fp) throw new Exception("Невозможно открыть файл!");
        // Запись данных в файл
        fclose($fp);
}
        catch (Exception $exception) {
        echo "Ошибка в строке ", $exception->getLine();
        echo $exception->getMessage(); // Выводит "Невозможно
        // открыть файл"
        }
        ?>
```

В конструкции можно использовать *несколько блоков* ***catch***. Также возможно создание собственных классов исключений, производных от встроенного класса Exception.

Ошибки PHP и журнал error_reporting

Библиотека PHP может генерировать ряд ошибок, не относящихся к уровню исключений – ошибок ООР.

В PHP обрабатываются три уровня ошибок: информационные ошибки, ошибки с возможностью действия (предупреждения) и неустранимые (фатальные) ошибки. Информационные ошибки: E_NOTICE – компилятор столкнулся с логической проблемой, которая может привести к ошибке выполнения. Аналогично E_USER_NOTICE генерируемое пользователем уведомление

Предупреждения означают, что возможно придется завершить работу с выдачей сообщения из-за того, что нельзя открыть файл и т.д. Ошибка - E_WARNING

предупреждение времени выполнения связанное с возможностью выполнения функции или блока. Аналогично E_USER_WARNING генерируемое пользователем предупреждение. E_COMPILE_WARNING предупреждения времени компиляции.

Неустранимые (фатальные) ошибки возникают тогда, когда выполнение или загрузка программы невозможны. E_ERROR(1) фатальные ошибки времени выполнения. E_PARSE(4) ошибки разбора времени компиляции. E_COMPILE_ERROR(64) фатальные ошибки времени компиляции. E_CORE_ERROR(16) фатальные ошибки при начальном старте PHP. E_ALL всё вышеуказанное.

Уровень сообщений об ошибках устанавливается как значение директивы error_reporting. В PHP установлено значение по умолчанию error_reporting = E_ALL & ~E_NOTICE, что означает отображение всех ошибок и предупреждений, за исключением логических ошибок, которые имеют уровень E_NOTICE. Начальное значение error_reporting может быть изменено в php.ini-файле директивой error_reporting, в Apache httpd.conf-файле директивой php_error_reporting и, наконец, оно может быть установлено на этапе прогона скрипта функцией error_reporting().

Параметр error_reporting в php.ini файле устанавливает уровень строгости ошибок E_ERRORLEVEL, которые будут обрабатываться внутренним обработчиком ошибок. Параметр display_error устанавливает вывод сообщений об ошибках на экран. Параметр log_errors регистрацию ошибок в журнальном файле, указанном в параметре error_log.

В природе существует большое количество скриптов не справляющихся с пользовательскими ошибками. До начала разработки необходимо правильно распланировать будущий проект и определить места возможных ошибок. Причём этим следует заняться до того, как скрипт был написан. Недоработки подобного рода приводят к сбоям программы, что чревато не только получением некорректных результатов, но и падением системы!

Любой скрипт может "свалиться" при наступлении "критичных" условий. Всегда нужно: проверять результаты вызова функций; проверять результаты системных вызовов; в файле php.ini устанавливать уровень error_reporting на E_ALL

При вызове функции, результаты которой подвергаются дальнейшей обработке, обязательно убедитесь, что возвращаемые данные находятся в интервале допустимых значений.

В приведённом ниже примере на шестом витке цикла возникает ошибка "деление на ноль", поскольку $i наращивается на 1, а $j уменьшается на 1. На шестом проходе $i=$j=1.

```
<?php
mt_srand((double)microtime() * 10000000);
function do_math ($a, $b) {
   return (($a - $b) * 2) / mt_rand();
}
for ($i = 5, $j = -5; $i > -5; $i--, $j++){
   print $j / do_math ($i, $j) . "\n";
}
?>
```

Проверка результатов системных вызовов

При обращении к внешним файлам или процессам всегда проверяйте, всё ли работает корректно. Хороший пример - проверка ответа системы при вызове функции sql_connect(). Стоит проверить этот ответ и убедиться, что подключение к БД действительно имело место. Если этого не сделать, то все запросы к БД могут не состояться, а некоторые данные могут быть утеряны.

```
<?php
$conn = @sql_connect ($host, $user, $pass);
```

```
if (!$conn) {
   die (sprintf ("Ошибка [%d]: %s", sql_errno (), sql_error ()));
}
?>
```

Установка уровня error_reporting в файле php.ini на E_ALL

Убедитесь, что PHP правильно сконфигурирован, то есть уровень error_reporting (отображение сообщений об ошибках) выставлено на наивысший уровень. При другой конфигурации, по крайней мере, на время отладки скриптов, многие ошибки типа "неверное регулярное выражение", "недопустимое значение" ускользнут от вашего внимания.

Обратимся ещё раз к примеру, приведённому в части "Проверка результатов вызова функций". Предположим, что error_reporting выставлен не на максимум, а, скажем, на E_ERROR.

Обратите внимание на то, как скрипт выполняет функцию do_math, но не сообщает об ошибке "деление на ноль", которая, однако, имела место (при $i=$j=0 вывода результата просто не было).

```
<?php
error_reporting(E_ERROR);
mt_srand ((double)microtime() * 1000000);
function do_math ($a, $b) {
   return (($a - $b) * 2) / mt_rand();
}
for ($i = 5, $j = -5; $i > -5; $i--, $j++){
   print $j / do_math ($i, $j) . "\n";
}
?>
```

Результат работы скрипта:

```
-5148.25
-5271
-323.75
-4931
-7713.5
-4702.5
-488.5
-928.5
-1394.75
```

Свои обработчики ошибок

Как правило, PHP выдаёт сообщения об ошибках непосредственно в браузер и не позволяет разработчику подавить или перехватить их. Однако в PHP имеется возможность перехвата таких сообщений с помощью функции set_error_handler().

В следующем примере set_error_handler() назначает обработчиком по умолчанию функцию error_handler(). В случае возникновения ошибки вызывается error_handler(), и встроенная функция error_log() регистрирует сбой в файле лога error_file.

Если происходит ошибка класса E_ERROR, работа скрипта прекращается и выводится сообщение об ошибке.

```
<?php
   // void error_handler(string type, string message, string file, int line)
   // Индивидуальный обработчик ошибок, определён функцией
   // set_error_handler()
   function error_handler ($type, $message, $file = __FILE__, $line = __LINE__) {
      error_log("$message, $file, $line", 3, 'error_file');
```

```
    if ($type & E_ERROR) {
        print 'Произошла ошибка, зарегистрирована.';
        exit;
    }
}
set_error_handler('error_handler');
?>
```

Новая модель обработки ошибок try{} …catch{} является прогрессивной.

Файлы

Файлы могут быть текстовыми, содержащими символьные строки переменной длины и бинарными, представляющими последовательность байт.

Для того, чтобы вставить другие файлы в текущий текстовый файл PHP, может использоваться директива include "filename". Строка filename содержит имя включаемого файла.

При работе с файлами в PHP используются три действия: 1) открытие файла для чтения или записи, 2) чтения или записи, 3) закрытие файла.

Открытие файла:

fopen ($filename, $mode [, use_include_path]). Функция осуществляет открытие файла с именем, указанным в $filename, и возвращает его дескриптор (номер). Режим открытия файла mode, может принимать следующие значения:

"r" - файл открывается только для чтения. "r+" - открывается на чтение и на запись. Текущий указатель файла устанавливается в его начало.

"w" - Файл открывается только для перезаписи. Указатель файла устанавливается в его начало. Всё старое содержимое файла теряется, счётчик длины файла устанавливается равным 0. Если файл с указанным именем не существует, функция пытается его создать. "w+" - Файл открывается на чтение и на запись. Указатель файла устанавливается в его начало. Всё старое содержимое файла теряется.

"a" - Файл открывается на добавление (запись). Указатель файла помещается в конец файла. Если файл с указанным именем не существует, функция пытается его создать. "a+" - Файл открывается на чтение и на запись. Вот несколько примеров открытия файлов с помощью fopen()

```
$fp = fopen ("/home/a/bases.dat", "r");
$fp = fopen ("/home_dir/client/count.txt", "w+");
$fp = fopen ("http://www.yahoo.com/pp.txt", "r");
```

В случае успешного завершения функция fopen() возвращает "ссылку" на открытый файл, а в случае ошибки - булевское значение false.

В PHP имеется возможность удаленного открытия файлов путем указания URL-адреса в качестве параметра $filename функции fopen (). Осуществлять запись в такие файлы, невозможно, но зато можно их читать.

URL, содержащие недопустимые символы (например, пробельные символы в имени файла), необходимо кодировать перед их использованием с помощью функции urlencode (). Функция urlencode () принимает единственный параметр (URL, подлежащий кодированию) и возвращает закодированный URL. Использование функции fopen()

```
<?php
/* Открыть файл для чтения */
$fr = fopen("myfile.txt",  'r');
/* Открыть бинарный файл для чтения/добавления */
$fr = fopen("myfile.dat",  'ba+');
/* Открыть файл для чтения/записи (искать файл в пути,
заданном директивой include_path)*/
```

```
$fr = fopen("code.php",  'w+', true);
/* Открыть файл index.php, для чтения по протоколу HTTP */
$fr = fopen("http://www.php.net/index.php",  'r');
/* Открыть файл index.php, для чтения по протоколу FTP */
$fr = fopen("ftp://ftp.php.net/index.php", 'r');
/*Закодировать URL, затем открыть  для чтения по протоколу HTTP */
$url = "http://www.php.net/this is my invalid URL.php";
$url = urlencode($url);
$fr = fopen($url, 'r');
 ?>
```

Закрытие файла: bool fclose ($fp).

Для работы с текстовыми файлами наиболее часто используются функции: fgets (), которая извлекает строку из файла, и fputs (), которая записывает строку в файл.

Функция считывания строки string fgets ($fp, $length) возвращает строку длиной до length-1 байт, считанную из файла $fp. Операция чтения завершается после загрузки length-1 символов, либо после обнаружения символа конца строки, либо при обнаружении признака конца файла. Вот пример построчного вывода на экран, содержимого файла:

```
<? //pr415
$fp = fopen ("pr415.php", "r");
while (!feof ($fp)) {
$stroka = fgets ($fp, 80);
echo $stroka."<BR>";
}
fclose($fp);
?>
```

Загрузка строки с пропуском HTML-тегов выполняется функцией: string fgetss ($fp, $length [, string allowable_tags]). Функция работает идентично fgets(), однако считанной строки удаляются все HTML-теги, которые в ней присутствуют. Есть возможность использовать необязательный третий аргумент, для указания разрешенных тегов, которые удалены не будут.

Альтернативой fgets () является функция fscanf (). Синтаксис функции fscanf () показан ниже:

```
fscanf ($filename, $format , $var_one [, $var_two ...]]])
```

где $filename — входной поток, $format задает шаблон для чтения, а $var_one, $var_two представляют собой переменные, в которых сохраняются разобранные фрагменты данных (эти параметры необходимо передавать по ссылке). В случае успешного завершения ввода fscanf () возвращает количество разобранных элементов, а случае ошибки возвращает значение false.

Допустимые символы форматирования функции fscanf ():

%b-Двоичное число. %c-Одиночный символ. %d-Десятичное число со знаком. %u-Десятичное число без знака. %f- Число с плавающей запятой. %o-Восьмеричное число. %s-Строка. %x-Шестнадцатеричное число.

Функция fputs () служит для записи строки (или любых других данных) в указанный поток и имеет следующий синтаксис: fputs($filename, $data [, int $length]), где $filename представляет выходной поток, $data содержит записываемые данные, а необязательный параметр $length задает размер фактически записываемых данных.

Чтение и запись бинарных файлов

Чтение данных из файла: string fread (int $fp, int length). Функция осуществляет чтение до length байт из файла, адресуемого указателем fp. Чтение прекращается, если будет считано указанное количество байт, или будет достигнут конец файла.

Запись данных в файл: int fwrite (int $fp, string strike [, int length])

Функция производит запись содержимого строки string в файл, адресуемый указателем fp. Если задан аргумент length, операция записи прекращается после вывода указанного количества символов, либо при достижении конца строки.

```
<? //pr416
$open=fopen("my_file.txt","w+");
//Очищаем файл и добавляем в него строку,
//если файл не существует, то он создаётся:
fwrite($open,"строка\r\n");
fclose($open);
//Добавляем новую строку в конец файла:
$open=fopen("my_file.txt","a");
fwrite($open,"новая строка\r\n");
fclose($open);
?>
```

В созданном файле my_file.txt будут записаны две строки

строка

новая строка

Загрузка всего файла: array file (string filename [, int use_include_path]). Функция file() записывает запрошенный файл в массив. При этом каждый элемент массива представляет собой одну строку файла. Символ новой строки, является последним символом каждой строки.

```
<? //pr417
//Создаём массив $array, где каждый индекс будет равняться номеру строки в файле:
$array=file("pr417.php");
print_r($array);
$count=count($array); // количество строк в файле
echo $count;
?>
```

Будет выведено:

Array ([0] => //Создаём массив $array, где каждый индекс будет равняться номеру строки в файле: [2] => $array=file("php416.php"); [3] => print_r($array); [4] => //Подсчитываем количество строк в файле: [5] => $count=count($array); [6] => echo $count; [7] => ?>) 8

```
<?php //php418.php - вывод строк из файла
$fi="php418.php"; // имя файла
$array=file($fi);
//Считываем из файла первые 3 строки:
echo "<hr>";
$n=3; //количество считываемых строк
 for($i=0;$i<$n;$i++)
 {print "$i : $array[$i],'\n'"; }

for($i=(count($array)-$n-1);$i<count($array)-1;$i++)
{ echo "$i : $file[$i],'\n'"; }

?>
```

Будет выведено три первых и три последних строки:

```
0 : <?php //php418.php - вывод строк из файла ,
' '1 : $fi="pr418.php"; // имя файла ,
' '2 : $array=file($fi); ,
' '10 : { echo "$i : $array[$i],'\n'"; } ,
' '11 : ,
' '12 : ?> ,' '
```

Объединяем 2 файла в один массив $new_array:

```
<? //pr419
$files=array(
 "php416.php", # первый файл
 "php417.php" # второй файл
);
 for($i=0;$i<count($files);$i++)
 {
 $array[]=file($files[$i]);
 }
 while(list(,$result)=each($array))
 {
 for($i=0;$i<count($result);$i++)
 { $new_array[]=$result[$i]; }
 }
print_r($new_array); #выводим массив
?>
```

Пример. Удаление указанной строки из файла:

```
<?
$line="1"; # строка, которую нужно удалить
$file=file("my_file.txt");
$open=fopen("my_file.txt","w");
 for($i=0;$i<count($file);$i++)
 {
 if(($i+1)!=$line){fwrite($open,$file[$i]);}
 }
fclose($open);
?>
```

В следующем примере массив устанавливается в файл. Затем выполняется замена строки в файле на указанную с помощью функции fwrite():

```
<?
$line="1"; # строка, которую нужно изменить
$replace="du du du"; # на что нужно заменить
$filarr=file("my_file.txt");
$f=fopen("my_file.txt","w");
 for($i=0;$i<count($filarr);$i++)
 {
 if(($i+1)!=$line){fwrite($f,$file[$i]);}
 else{fwrite($f,$replace."\r\n");}
 }
fclose($f);
?>
```

Поиск слова в файле. Ищем в файле file.txt слово привет:

```
<?PHP
$word="привет"; # искомое
$f="file.txt"; # имя файла, в котором будем искать слово
$open=fopen($file,"r");
while(!feof($open)) $search.=fgets($open,1024);
fclose($open);
 if(eregi(strtolower($word),strtolower($search)))
 { echo "Слово ".$word." - найдено"; }
 else
 { echo "Слово ".$word." - не найдено"; }
?>
```

Прямой доступ к указанной позиции бинарного файла осуществляется с помощью функции fseek (): fseek($file, $offset [, $reference]);

где $offset указывает число байт смещения от начала файла ($reference =SEEK_SET или по умолчанию) от текущей позиции файла ($reference=SEEK_CUR), от конца файла ($reference= SEEK_END). Функция fseek () может использоваться только для файлов, расположенных в локальной файловой системе и не работает с файлами, которые открываются удаленно по протоколу HTTP или FTP. Функция fseek () возвращает нуль в случае успешного завершения и -1 в случае, если указатель файла не может быть установлен. Отсчет смещений указателя в fseek () начинается с нуля. Чтобы установить указатель файла в позицию $offset, функции fseek () необходимо передать параметр $offset-l.

Копирование одного файла в другой можно выполнить с помощью функции copy(filefrom,fileto);. Функция unlink(filename) –удаляет файл. Функция rename(oldname, newname)-переименовывает файл.

Работа с каталогами в PHP

PHP предоставляет набор функций для работы с каталогами: для создания, удаления и вывода оглавления каталогов. Для этого в PHP имеются функции opendir() и closedir (), аналогичные функциям fopen () и fclose () для файлов. Функция opendir () имеет следующий синтаксис:

$dir_reference =opendir($dir_path)

Здесь $dir_path представляет путь к открываемому каталогу. Функция opendir() выведет сообщение об ошибке, если указанный каталог не существует. При успешном завершении opendir () возвратит дескриптор каталога. Функция closedir() принимает единственный параметр - дескриптор каталога из вызова opendir ().

После открытия каталога, его элементы можно прочитать с помощью функции readdir($dir_reference), где $dir_reference — значение, которое возвращает функция opendir (). При успешном завершении функция возвращает строку, содержащую имя одного из файлов каталога. Каждый последующий вызов функции readdir () возвращает очередной файл, пока весь список файлов не будет исчерпан. Если файлов в каталоге больше нет, или произошла ошибка, readdir () вернет значение false. Выведем содержимое папки c:/windows

```
<?
$dir="c:/windows"; # папка, которую нужно прочитать
if($OpenDir=opendir($dir))
{
 while(($file=readdir($OpenDir)) !== false)
 if($file != "." && $file != "..")
 echo $file."<br>";
```

```
}
else echo "нет прав";
?>
```

Для создания каталога можно использовать функцию mkdir(name), для удаления – rmdir(), для изменения – chdir(). Чтение каталога:

```
<?php
$dr = @opendir(' /tmp/') ;
if(!$dr)  {
echo "Ошибка при открытии каталога /tmp/!<BR>"; exit;
1
while (($filesf] = readdir($dr))  !== false);
print_r(Sfiles);
?>
```

Поскольку функция readdir() возвращает каждый раз новое имя файла, то каждый файл каталога можно просмотреть только однажды. Если необходимо повторно просмотреть содержимое каталога, PHP предоставляет функцию, которая позволяет "перемотать" оглавление каталога в исходное состояние. Эта функция, называемая rewinddir (), имеет следующий синтаксис: rewinddir($dir_reference)

Полезным является альтернативный метод получения списка файлов, удовлетворяющих определенному критерию (или шаблону). Этой цели служит функция:

```
glob($filemask [, flags])
```

Здесь $filemask — это строка, содержащая шаблон поиска (например, *.txt), a flags представляет одну или несколько констант, перечисленных ниже. Значения flags: GL0B_MARK -Добавлять слэш к именам, которые являются каталогами; GL0B_N0S0RT -Не сортировать возвращаемый список файлов; GL0B_N0CHECK -Если нет файлов, совпадающих с шаблоном, вернуть шаблон вместо пустого массива; GL0B_0NLYDIR - Вернуть только каталоги, совпадающие с шаблоном.

Использование функции glob ():

```
<?php
Sdirectories = glob("/tmp/*", GL0B_0NLYDIR);
$complete = glob("/tmp/*");
Sfiles = array_diff(Sdirectories, $complete);
echo "Каталоги в /tmp/<BR>";
foreach(Sdirectories as $val) ( echo "$val<BR>\n";)
echo "<BR><&aunbi в /tmp/<BR>";
foreach($files as Sval)  ( echo "$val<BR>\n";)
?>
```

PHP **и** MySQL

MySQL – это многопоточный быстрый сервер баз данных, разработанный компанией TcX. MySQL является идеальным решением для малых и средних приложений, созданных на PHP. MySQL поддерживает стандарт языка запросов SQL и имеет множество расширений к стандарту. MySql сервер загружается автоматически вместе с web – сервером и постоянно работает на компьютере – сервере. MySQL поддерживает трехуровневую структуру: базы данных, таблицы, записи и может работать сразу с несколькими базами данных:

MySql – сервер: БД1; БД2; БД3; БД4;

Под БД понимается совокупность взаимосвязанных таблиц, объединенных общим управлением. Перед началом работы с MySql надо: спроектировать БД; спроектировать структуру таблиц БД; создать таблицы БД; создать БД.

Каждая таблица состоит из записей. В MySql используются следующие типы полей записей:

Целые числа: INT(-2147483648..2147483647), TINYINT(-128..127), SMALLINT, MEDIUMINT, BIGINT;

Вещественные числа: DOUBLE, REAL(синоним DOUBLE), FLOAT, DECIMAL (числовая строка), NUMERIC;

Строки: VARCHAR (1..255), TINYTEXT (1..255), TEXT(1..65535); MEDIUMTEXT, LONGTEXT;

Блочно-бинарные данные: BLOB(до 65535 символов), TINYBLOB(до 255 символов), MEDIUMBLOB, LONGBLOB: при поиске, в отличие от TEXT, учитывается нижний и верхний регистр символов;

Дата и время: DATE(гггг-мм-дд), TIME(чч:мм:сс), DATETIME(гггг-мм-дд чч:мм:сс), TIMESTAMP.

К полям могут присоединяться модификаторы primary key, auto_increment, Default.

Создание БД и таблиц может быть выполнено вручную, например с помощью приложения phpmyadmin, входящего в состав MySql. Набирается www.localhost/phpmyadmin и выполняются действия по созданию БД и таблиц.

Другой способ создание БД и таблиц состоит в использовании команд языка SQL, являющегося частью MySql.

Язык запросов SQL

Язык запросов SQL – основной язык для работы с реляционными базами данных, построенными в соответствии с правилами реляционной алгебры. Состав языка SQL следующий:

Язык манипулирования данными состоящий из команд: SELECT (выбрать), INSERT (вставить), UPDATE (обновить), DELETE (удалить).

Язык определения данных используемый для создания и изменения структуры БД и ее составных частей – таблиц, индексов, представлений (виртуальных таблиц). Основными его командами являются: CREATE DATABASE (создать базу данных), CREATE TABLE (создать таблицу), CREATE INDEX (создать индекс), ALTER DATABASE (модифицировать базу данных), ALTER TABLE (модифицировать таблицу), ALTER INDEX (модифицировать индекс), DROP DATABASE (удалить базу данных), DROP TABLE (удалить таблицу), DROP INDEX (удалить индекс).

Язык управления данными (управления доступом) состоит из двух основных команд: GRANT (дать права), REVOKE (забрать права).

Наиболее важной и часто используемой командой языка манипулирования данными SQL является команда SELECT. Формат команды SELECT в языке SQL:

SELECT поля FROM таблицы WHERE условие;

Базовыми операциями команды SELECT являются: **выборка, проекция, соединение, объединение**.

Операция выборки позволяет получить все строки либо часть строк таблицы.

SELECT * FROM Student; – Получить все строки таблицы Student

SELECT * FROM Student WHERE Kurs=2; – Получить подмножество строк таблицы, удовлетворяющих условию Kurs=2. Точка с запятой является стандартным признаком конца команды, который вставляется автоматически.

Операция проекции позволяет выделить подмножество столбцов таблицы.

SELECT StudName FROM Student WHERE Kurs=2; – Получить имена студентов второго курса.

Операция соединения позволяет соединять строки из более чем одной таблицы:

SELECT StudName, Exammark FROM Students, Exams WHERE Students.Stud_Id =Exams.Stud_Id – Получить список студентов и экзаменационных оценок.

Операция объединения позволяет объединять результаты отдельных запросов. Предложение UNION объединяет вывод двух или более SQL-запросов.

SELECT name FROM employee WHERE country = "Беларусь" UNION SELECT contact_name, FROM customer WHERE country = "Беларусь"; – Получить список работников и заказчиков, проживающих в Беларуси.

Условия отбора. Директива WHERE содержит условия отбора (предикат). Запрос возвращает только строки, для которых предикат имеет значение true. Типы предикатов, используемых в предложении WHERE:

Сравнение: = (равно); <> (не равно); != (не равно); > (больше); < (меньше); >= (больше или равно); <= (меньше или равно); BETWEEN, IN , LIKE, CONTAINING, IS NULL, EXIST, ANY, ALL.

Предикат BETWEEN задает диапазон значений, для которого истинно значение выражения. Например:

SELECT StudName, Stipend FROM Student WHERE Stipend BETWEEN 120 AND 200 – получить список студентов стипендия которых больше 120 и меньше 200.

Тот же запрос с использованием операторов сравнения будет выглядеть следующим образом:

SELECT StudName, Stipend FROM Student WHERE Stipend>=120000 AND Stipend<=200000

Предикат IN (NOT IN) проверяет, входит ли заданное значение, предшествующее ключевому слову “IN”, в указанный в скобках список. Например:

SELECT name FROM employee WHERE job_code IN ("VP", "Admin", "Finan") – получить список сотрудников, занимающих должности “вице-президент”, “администратор”, “финансовый директор”.

Предикат LIKE проверяет, соответствует ли данное символьное значение строке с указанной маской. В качестве маски используются все разрешенные символы (с учетом верхнего и нижнего регистров), а также специальные символы: % – замещает любое количество символов, _ – замещает только один символ. Например:

SELECT StudName FROM Student WHERE StudName LIKE "Ф%" – получить список студентов, фамилии которых начинаются с буквы ‘Ф’.

Предикат CONTAINING аналогичен предикату LIKE, однако он не чувствителен к регистру букв.

Предикат IS NULL принимает значение true только тогда, когда выражение слева от “IS NULL” имеет значение null (пусто, не определено).

Логические операторы. К логическим операторам относятся NOT, AND, OR. В одном предикате логические операторы выполняются в указанном порядке.

Преобразование типов. В SQL имеется возможность преобразовать значение к другому типу для выполнения операций сравнения. Для этого используется функция CAST.

Изменение порядка выводимых строк. Порядок выводимых строк может быть изменен с помощью предложения ORDER BY в конце SQL-запроса. Это предложение имеет вид: ORDER BY [ASC | DESC]

Способом по умолчанию является упорядочивание “по возрастанию” (ASC). Если указано “DESC”, упорядочивание производится “по убыванию”. Например:

SELECT StudName, Stipend FROM Student ORDER BY StudName – получить список в алфавитном порядке.

Операция соединения.

Внутреннее соединение возвращает только те строки, для которых условие соединения принимает значение true. Рассмотрим пример запроса:

SELECT StudName, ExamMark FROM Student, Exams WHERE Kurs=2 AND ExamMark=5 – получить список студентов 2-го курса, сдававших экзамен на 5.

Внешнее соединение возвращает все строки из одной таблицы и те строки из другой таблицы, для которых условие соединения принимает значение true. Существуют

два вида внешнего соединения: в левом соединении (LEFT JOIN) запрос возвращает все строки из_ таблицы, стоящей *слева* от LEFT JOIN и только те из правой таблицы, которые удовлетворяют условию соединения. Для правого соединения – все наоборот. Например:

SELECT name, department FROM employee e LEFT JOIN department d ON e.dept_no = d.dept_no – получить список сотрудников и название их отделов, включая сотрудников, еще не назначенных ни в какой отдел.

Команды SQL для создания баз данных и таблиц

CREATE DATABASE database_name - создать базу данных,

CREATE TABLE table_name(Имя_поля1 тип1, Имя_поля2 тип2…)-создать таблицу,

DROP DATABASE database_name - удалить базу данных,

DROP TABLE table_name -удалить таблицу,

ALTER DATABASE database_name - модифицировать базу данных,

ALTER TABLE table_name -модифицировать таблицу,

INSERT INTO table_name (Имя_поля1 тип1, Имя_поля2 тип2…) values(‘val1’,’val2’,…) - вставка записи со значениями полей val1, val2, … .

DELETE FROM table_name WHERE выражение -удаление записи, для которой выполнено выражение

UPDATE table_name SET(Имя_поля1 ‘val1’, ‘Имя_поля2 val2’…) where выражение –обновить таблицу.

Для выполнения SQL запроса используется функция mysql_query($sql);. Сначала создается строка, содержащая SQL – запрос. Затем эта строка передается для выполнения. Рассмотрим пример выполнения команд SQL по созданию базы данных из PHP:

```
//соединяемся с сервером
int mysql_connect();
    // создаем базу данных:
    $sql=”create database ‘mydb’”;
    mysql_query($sql);//выполнение директивы SQL
    //Создаем таблицу:
    $sql=”CREATE TABLE userstable
    (name VARCHAR(25), email VARCHAR(25), choise VARCHAR(15))”
    mysql_query($sql); );//выполнение директивы SQL
```

Аналогично порядок работы с MySQL для выборки данных из базы в PHP устанавливается следующий:

Создать соединение с сервером MySQL, расположенным на $hostname:

int mysql_connect([string $hostname],[string $username],[string password]);

или mysql_pconnect().

Здесь hostname - имя хоста (по умолчанию localhost), username - имя пользователя, password - пароль пользователя. Функция возвращает параметр ID соединения (link_identifier), который равен 0, если соединение не прошло успешно.

Выбрать базу данных для работы:

int mysql_select_db($database_name[, int link_identifier]);

Здесь: database_name - имя базы данных, link_identifier – необязательный параметр, содержащий ID соединения из mysql_connect. Если этот параметр отсутствует, выбирается последнее открытое соединение. Функция возвращает значение true или false.

Выполнить запрос к базе данных.

int mysql_query(string $query[, int $link_identifier]);

Здесь $query - строка, содержащая SQL запрос, link_identifier –ID соединения. Функция возвращает ID результата или 0, если произошла ошибка.

Закрыть соединение с MySQL.

int mysql_close(int link_identifier); Параметры: link_identifier - ID соединения. Функция возвращает значение true или false.

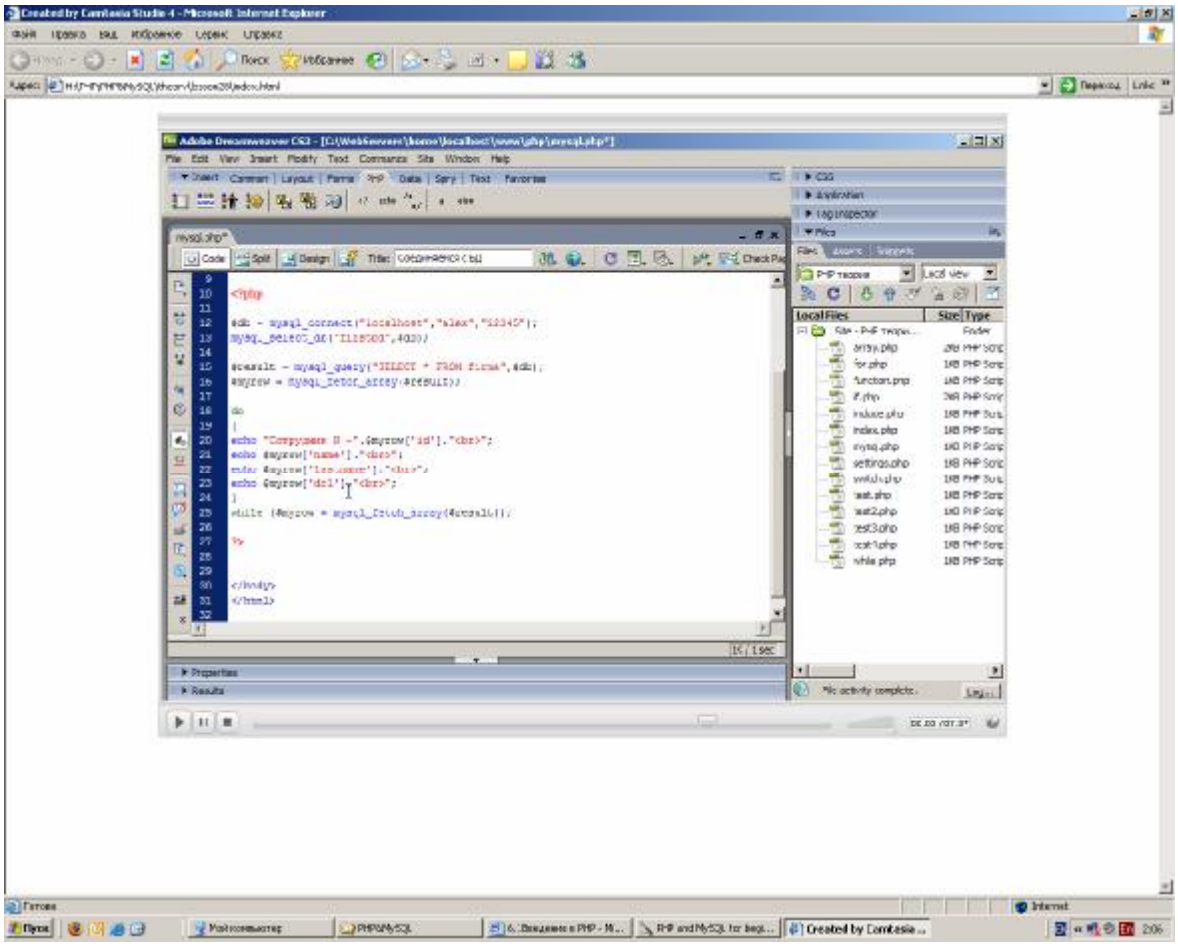
Пример: Создание базы данных и таблицы.

```
<html>
<head>
<title>Запрос информации</title>
<body>
<center> Введите Ваши данные <p>
<table width = 400><tr><td align = right>
<form action="pr1BD.php" method="post">
Ваше имя:<BR>
<input type="text" name="name" size="20" maxlength="30">
<P>
Ваш email:<BR>
<input type="text" name="email" size="20" maxlength="30">
<p>
<input type="submit" value="Отправить запрос!">
</form>
</td></tr></table></center>
</body>
</html>

<?php  // pr1bd.php - скрипт создает БД и таблицу
$dbName ='mydb';
/* создать соединение */
$link = mysql_connect( "localhost" , "root"  )  or exit("Could not connect");;
/* уничтожить старую БД */
$sql= sprintf("DROP DATABASE %s ",$dbName);
if (mysql_query($sql, $link)) {
   echo "Database drop successfully\n";
} else {     echo 'Error drop database: ' . mysql_error() . "\n"; }
/* создать новую БД */
$sql="create database $dbName";
//выполнение директивы SQL
if (mysql_query($sql, $link)) {
   echo "Database created successfully\n";
} else {     echo 'Error creating database: ' . mysql_error() . "\n"; }
//Создаем таблицу: mydb.clients
$tablename = 'clients' ;
$sql=sprintf("CREATE TABLE %s.%s (
`name` VARCHAR( 25 ) NOT NULL , `email` VARCHAR( 25 ) NOT NULL )
ENGINE = MYISAM ",$dbName,$tablename);
if (mysql_query($sql, $link)) {
   echo "Table created successfully\n";
} else {     echo 'Error creating Table: ' . mysql_error() . "\n"; }
$name=$_POST['name'];
$email=$_POST['email'];
// вставка записи со значениями полей val1, val2,
```

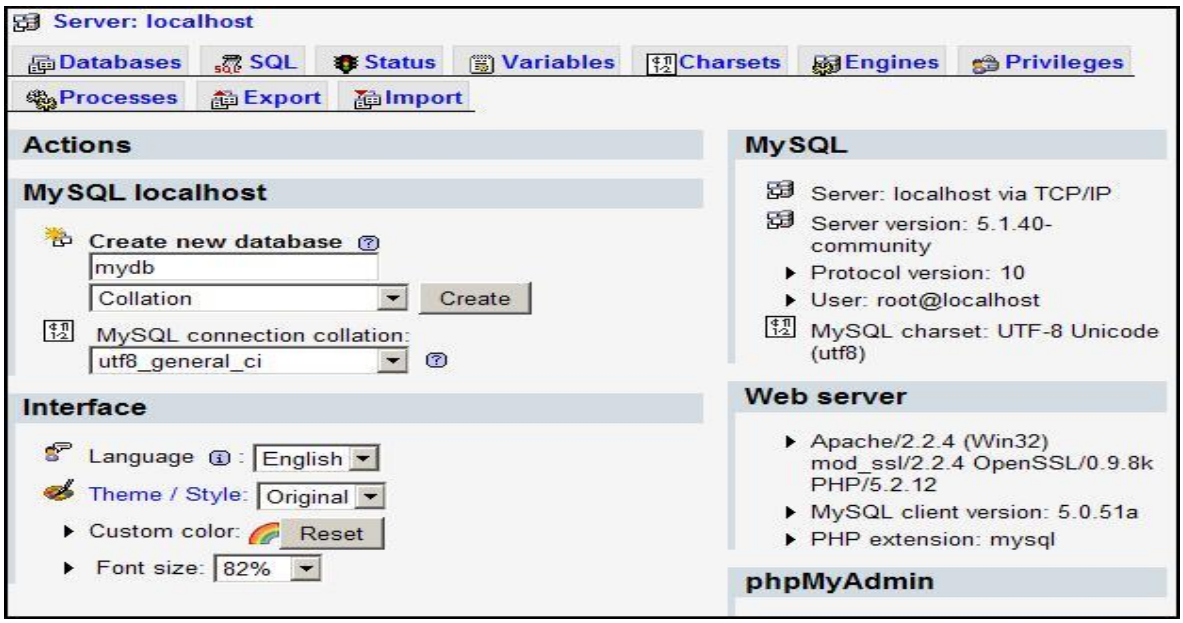
```
$sql=sprintf("INSERT INTO %s.%s  values('$name', '$email')", $dbName, $tablename);
if (mysql_query($sql, $link)) {
   echo "Table insert successfully\n";
} else {     echo 'Error insert Table: ' . mysql_error() . "\n"; }
mysql_close($link); /* Закрыть соединение */

$link = mysql_connect( "localhost" , "root"  )or die("Не могу создать соединение ");;
//Выбрать БД
$result =mysql_select_db("$dbName",$link) or die("Не могу выбрать базу данных ");
//создать SQL - запрос
$sql=sprintf("Select *  from %s", $tablename);
$result = mysql_query($sql,$link);
if (!$result) {     die('Invalid query: ' . mysql_error()); }
$myrow=mysql_fetch_array($result); //установка первой записи
echo"<br>", $myrow['name'];//вывод поля “name” первой записи
echo $myrow['email'];
mysql_close($link);/* Закрыть соединение */
?>
```



PHPMYADMIN

Программное средство phpmyadmin представляет удобные средства и инструменты для создания баз данных, для проектирования и создания таблиц. Вызвать утилиту возможно следующим образом: http://www.localhost/phpmyadmin



Часто вручную создают базу данных и устанавливают права доступа. Затем создается структура таблиц и сами таблицы.

Выборка данных из таблиц БД

Создать соединение с MySQL.

int mysql_connect(string hostname, string username, string password);

Выбрать базу данных для работы.

int mysql_select_db(string database_name, int link_identifier);

SQL запрос к базе данных на выборку данных.

int mysql_query(string query, int link_identifier);

Функция возвращает ID результата или 0, если произошла ошибка.

Закрытие соединения с MySQL.

int mysql_close(int link_identifier);

Выборка данных из базы производится SQL командой SELECT.

В PHP имеются ряд функций, позволяющих сохранять и обрабатывать данные, на которые указывает дескриптор result $result=mysql_query(“SELECT …”);

int mysql_result(int result, int i, column);

Функция возвращает значение поля в столбце column и в строке i.

int mysql_num_rows(int result);

Функция возвращает количество строк в результате запроса.

mysql_fetch_row($result))

Возвращает массив соответствующий очередной записи таблицы

mysql_fetch_array(result)

Возвращает результаты запроса в виде ассоциированного массива

```
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
$result = mysql_query("SELECT * FROM employees",$db);
$strike= mysql_result($result,0,"first");
printf("First Name: %s<br>\n",$str);
$strike= mysql_result($result,0,"last");
printf("Last Name: %s<br>\n",$str);
$strike= mysql_result($result,0,"address");
printf("Address: %s<br>\n",$str);
```

```
$strike= mysql_result($result,0,"position");
printf("Position: %s<br>\n",$str);
mysql_close($db);
?>
```

Функция mysql_connect() открывает связь с сервером баз данных MySQL. В результате выполнения этой функции получаем значение идентификатора соединения, которое присваиваем переменной $db. С помощью функции mysql_select_db() выбираем базу данных. В качестве параметров передаем название нужной нам базы данных и идентификатор соединения, полученный нами при выполнении предыдущей команды.

В результате выполнения функции mysql_select_db() получаем значение true, если соединение с базой данных произошло успешно иначе false. Наконец, мы обращаемся к базе данных с запросом на языке SQL. Для этого служит функция mysql_query(). Результаты выполнения функции mysql_query() – записи, удовлетворяющие нашему запросу - помещаем в переменную $result.

С помощью функции mysql_result() извлекаем из переменной $result, первую запись (порядковый номер 0), и значение определенного поля (по его имени). Перебирая друг за другом записи, выберем все записи, что хранятся в базе данных.

Рассмотрим более современные, чем mysql_result(), функции выборки данных mysql_fetch_row() и mysql_fetch_array(). Выведем все записи, хранящиеся в базе. Обратимся к базе данных со следующим кодом:

```
<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
$result = mysql_query("SELECT * FROM employees",$db);
echo "<table border=1>\n";
echo "<tr><td>Name</td><td>Position</tr>\n";
while ($myrow = mysql_fetch_row($result))
{
printf("<tr><td>%s %s</td><td>%s</td></tr>\n", $myrow[1], $myrow[2],
$myrow[3]);
}
echo "</table>\n";
?>
</body>
</html>
```

Цикл while() говорит, что пока в переменной $result остается запись для выборки, ее необходимо извлечь с помощью функции mysql_fetch_row и присвоить переменной $myrow. А после этого выполнить код, что расположен внутри фигурных скобок "{}". Функции mysql_fetch_row в качестве параметра подается массив $result. Функция выбирает из него строку, которую мы можем записать в переменную $myrow и автоматически переходит на следующую строку. Вызвав снова mysql_fetch_row, мы выберем следующую строку из массива, и так далее до тех пор, пока не достигнем конца массива. В этом случае mysql_fetch_row вернет значение false, которое послужит сигналом, что все записи выбраны и можно завершить цикл.

Теперь наша задача как-то вывести в теле цикла полученную запись. Выбранный ряд хранится в переменной $myrow. Она также, как и $result, является массивом, только

одномерным. Главное не забывайте, что элементы массивов нумеруются от 0, а не от 1, так как здесь мы имеем дело с компьютерной, а не человеческой логикой. Наш код грешит одним недостатком: если в базе данных не будут найдены записи, удовлетворяющие нашему запросу, мы не получим никакого сообщения об этом. Неплохо было бы, чтобы программа выдавала какое-нибудь сообщение. Сделаем ее более дружественной. Взгляните на следующий код:

Пример: Соединение с БД и выборка записей

```
<?// php428bd.php - скрипт создает БД и таблицу
$dbName ="mydb";
/* создать соединение */
$link = mysql_connect("localhost", "valera", "12345")
 or exit("Could not connect");
mysql_select_db("$dbName",$link) or die("Не могу выбрать базу данных ");
//создать SQL - запрос
$sqlq="Select * from students";
//
$result = mysql_query($sqlq,$link) or die ("Invalid query");
$myrow=mysql_fetch_array($result); //установка первой записи
echo $myrow["name"];//вывод поля “name” первой записи
do//цикл просмотра всех записей
{
echo “сотрудник-”, $myrow[‘id’];
echo $myrow["name"];
}
while($myrow=mysql_fetch_array($result));
MYSQL_CLOSE();/* Закрыть соединение */
?>

<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
$result = mysql_query("SELECT * FROM employees",$db);
if ($myrow = mysql_fetch_array($result))
{
echo "<table border=1>\n";
echo "<tr><td>Name</td><td>Position</td></tr>\n";
do
{
printf("<tr><td>%s %s</td><td>%s</tr>\n", $myrow["first"], $myrow["last"],
$myrow["address"]);
}
while ($myrow = mysql_fetch_array($result));
echo "</table>\n";
}
else
{
echo "Sorry, no records were found!";
}
?>
```

```
</body>
</html>
```

В данном коде вместо функции mysql_fetch_row() мы использовали функцию mysql_fetch_array(). С помощью этой функции мы можем обращаться к каждому полю массива не по номеру, а по имени. Например, если раньше для получения имени нам приходилось писать $myrow[1], то теперь мы можем писать $myrow["first"] ("first" – название столбца в базе данных и в массиве). Кроме этого, в коде использован цикл do/while и условная конструкция if-else. Выражение if-else говорит, что если мы можем присвоить значение $myrow, то надо начать выборку, в противном случае записей нет, переходим к блоку else и выводим соответствующее сообщение.

Рассмотрим параметры запроса. Существует три способа передачи параметров запроса. Первый, использовать метод GET в форме. Второй – набрать параметры прямо в адресной строке браузера. И третий, это вставить параметры в обычную ссылку на странице. То есть сделать ссылку примерно такого вида

```
<a href="http://my_machine/mypage.php3?id=1">
```

Для начала, давайте обратимся к базе данных и выведем список персонала. Взгляните на следующий код.

```
<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
$result = mysql_query("SELECT * FROM employees",$db);
if ($myrow = mysql_fetch_array($result))
{
do
{
printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF,
$myrow["id"],$myrow["first"], $myrow["last"]);
}
while ($myrow = mysql_fetch_array($result));
}
else
{
echo "Sorry, no records were found!";
}
?>
</body>
</html>
```

Рассмотрим функцию printf() поближе. Во-первых, обратите внимание на обратные косые черты. Они говорят PHP-движку, что необходимо вывести символ, следующий за чертой, а не рассматривать его, как служебный символ или как часть кода. В данном случае это касается кавычки, которая нам нужна в тексте ссылки, но для PHP является символом окончания текстовой строки.

Далее, в коде используется интересная переменная $PHP_SELF. В этой переменной всегда хранится имя и URL текущей страницы. В данном случае эта переменная важна для нас потому, что мы хотим через ссылку вызвать страницу из нее самой. То есть вместо того, чтобы делать две страницы, содержащие разные коды для разных действий, мы все действия запихнули в одну страницу. С помощью условий if-else мы будем переводить стрелки с одного кода на другой, гоняя одну и ту же страницу по кругу. Это конечно увеличит размер страницы и время, необходимое на ее обработку,

но в некоторых случая, такой трюк очень удобен. Переменная $PHP_SELF гарантирует нам, что наша страница будет работать даже в том случае, если мы перенесем ее в другой каталог или даже на другую машину. Как мы уже сказали, ссылки, сгенерированные в цикле ссылаются на ту же самую страницу, только к имени самой страницы на лету добавлена некоторая информация: переменные и их значения. Переменные, которые передаются в строке-ссылке, автоматически создаются PHP-движком, и к ним можно обращаться так, как если бы вы их создавали в коде сами. При втором проходе страницы наша программа отреагирует на эти пары name=value и направит ход исполнения на другие рельсы. В данном случае мы проверим, есть ли переменная $id, и в зависимости от результата выполним тот или иной код. Вот как это будет выглядеть:

```
<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
// display individual record
if ($id)
{
$result = mysql_query("SELECT * FROM employees WHERE id=$id",$db);
$myrow = mysql_fetch_array($result);
printf("First name: %s\n<br>", $myrow["first"]);
printf("Last name: %s\n<br>", $myrow["last"]);
printf("Address: %s\n<br>", $myrow["address"]);
printf("Position: %s\n<br>", $myrow["position"]);
}
else
{
// show employee list
$result = mysql_query("SELECT * FROM employees",$db);
if ($myrow = mysql_fetch_array($result))
{
// display list if there are records to display
do
{
printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF, $myrow["id"],
$myrow["first"], $myrow["last"]);
}
while ($myrow = mysql_fetch_array($result));
}
else
{
// no records to display
echo "Sorry, no records were found!";
}
}
?>
</body>
</html>
```

Работа с MySQL (**сохранение данных в базе данных**).

Мы научились извлекать данные из базы и выводить их на странице. Теперь давай попробуем осуществить обратное действие. С PHP это не составит большого труда. Создадим простую форму:

```
<html>
<body>
<form method="post" action="<?php echo $PHP_SELF?>">
First name:<input type="Text" name="first"><br>
Last name:<input type="Text" name="last"><br>
Address:<input type="Text" name="address"><br>
Position:<input type="Text" name="position"><br>
<input type="Submit" name="submit" value="Enter information">
</form>
</body>
</html>
```

Мы опять используем переменную $PHP_SELF. PHP-код можно как угодно перемешивать с обычным HTML. Также обратите внимание, что название каждого элемента формы совпадает с названием поля в базе данных. Это не обязательно, но удобно, чтобы не запутаться в том, какая переменная какому полю в базе данных соответствует.

Помимо этого мы присвоили имя кнопке Submit. Это сделано для того, чтобы в коде затем проверить, есть ли переменная $submit. Таким образом, когда страница будет вызываться, мы будем узнавать, вызывается ли она в первый или во второй раз.

Итак, добавим код, который будет проверять, введены ли в форму данные. Пока это будет лишь простая проверка, при которой все переменные, передаваемые странице, будут выводиться на экран с помощью переменной $HTTP_POST_VARS. Эта переменная удобна в случае отладки. Если вы хотите вывести на экран вообще все переменные, используемые в странице, вызовите переменную $GLOBALS.

```
<html>
<body>
<?php
if ($submit) {
// process form
while (list($name, $value) = each($HTTP_POST_VARS)) {
echo "$name = $value<br>\n";
}
} else{
// display form
?>
<form method="post" action="<?php echo $PHP_SELF?>">
First name:<input type="Text" name="first"><br>
Last name:<input type="Text" name="last"><br>
Address:<input type="Text" name="address"><br>
Position:<input type="Text" name="position"><br>
<input type="Submit" name="submit" value="Enter information">
</form>
<?php
} // end if
?>
</body>
</html>
```

Теперь возьмем подданную через форму информацию и внесем ее в базу данных.

```
<html>
<body>
<?php
if ($submit) {
// process form
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
$sql = "INSERT INTO employees (first,last,address,position) VALUES
('$first','$last','$address','$position')";
$result = mysql_query($sql);
echo "Thank you! Information entered.\n";
} else{
// display form
?>
<form method="post" action="<?php echo $PHP_SELF?>">
First name:<input type="Text" name="first"><br>
Last name:<input type="Text" name="last"><br>
Address:<input type="Text" name="address"><br>
Position:<input type="Text" name="position"><br>
<input type="Submit" name="submit" value="Enter information">
</form>
<?php
}
// end if
?>
</body>
</html>
```

Мы внесли данные в базу. Что случится, если при заполнении формы кто-то оставит пустые поля или введет текст в поле, в которое надо ввести число? Мы записывали SQL-выражение в переменную ($sql), прежде чем передать запрос в базу данных через функцию mysql_query(). Это делается на случай отладки. Если что-то пойдет не так, мы всегда сможем вывести интересующее нас SQL-выражение на экран и проверить, нет ли в нем ошибок.

Мы уже знаем, как вставлять данные в базу. Теперь давайте научимся менять записи, которые уже находятся в таблице. Редактирование данных сочетает в себе два кода. Которые мы уже проходили: извлечение данных из базы с выводом их на экран, и внесение данных через форму обратно в базу. Тем не менее программа правки данных немного отличается тем, что мы в форме должны вывести некую конкретную запись. Для начала давайте воспользуемся кодом из предыдущей главы, для вывода списка служащих на экран. Однако теперь информацию о служащих мы будет отображать в форме. Код страницы будет выглядеть так:

```
<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
if ($id) {
// query the DB
$sql = "SELECT * FROM employees WHERE id=$id";
$result = mysql_query($sql);
```

```
$myrow = mysql_fetch_array($result);
?>
<form method="post" action="<?php echo $PHP_SELF?>">
<input type=hidden name="id" value="<?php echo $myrow["id"] ?>">
First name:<input type="Text" name="first" value="<?php echo $myrow["first"]
?>"><br>
Last name:<input type="Text" name="last" value="<?php echo $myrow["last"]
?>"><br>
Address:<input type="Text" name="address" value="<?php echo
$myrow["address"] ?>"><br>
Position:<input type="Text" name="position" value="<?php echo
$myrow["position"] ?>"><br>
<input type="Submit" name="submit" value="Enter information">
</form>
<?php
} else {
// display list of employees
$result = mysql_query("SELECT * FROM employees",$db);
while ($myrow = mysql_fetch_array($result)) {
printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF, $myrow["id"],
$myrow["first"],
$myrow["last"]);
}
}
?>
</body>
</html>
```

В этой странице мы просто вывели в каждое поле формы соответствующее значение из базы данных, что было достаточно несложно. Теперь мы усложним программу. Добавим к ней возможность внесения отредактированных данных назад в базу. Опять таки мы прибегаем к помощи кнопки Submit, которой присваиваем имя, чтобы при втором проходе страницы проверить, какую часть кода нам надо выполнять. Также мы здесь используем слегка измененное SQL-выражение.

```
<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
if ($id) {
if ($submit) {
$sql = "UPDATE employees SET
first='$first',last='$last',address='$address',position='$position'
WHERE id=$id";
$result = mysql_query($sql);
echo "Thank you! Information updated.\n";
} else {
// query the DB
$sql = "SELECT * FROM employees WHERE id=$id";
$result = mysql_query($sql);
$myrow = mysql_fetch_array($result);
?>
```

```
<form method="post" action="<?php echo $PHP_SELF?>">
<input type=hidden name="id" value="<?php echo $myrow["id"] ?>">
First name:<input type="Text" name="first" value="<?php echo $myrow["first"]
?>"><br>
Last name:<input type="Text" name="last" value="<?php echo $myrow["last"]
?>"><br>
Address:<input type="Text" name="address" value="<?php echo
$myrow["address"] ?>"><br>
Position:<input type="Text" name="position" value="<?php echo
$myrow["position"] ?>"><br>
<input type="Submit" name="submit" value="Enter information">
</form>
<?php
}
} else {
// display list of employees
$result = mysql_query("SELECT * FROM employees",$db);
while ($myrow = mysql_fetch_array($result)) {
printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF, $myrow["id"],
$myrow["first"],
$myrow["last"]);
}
}
?>
</body>
</html>
```

Теперь пришло время свести все вместе .

```
<html>
<body>
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
if ($submit) {
// here if no ID then adding else we're editing
if ($id) {
$sql = "UPDATE employees SET
first='$first',last='$last',address='$address',position='$position'
WHERE id=$id";
} else {
$sql = "INSERT INTO employees (first,last,address,position) VALUES
('$first','$last','$address','$position')";
}
// run SQL against the DB
$result = mysql_query($sql);
echo "Record updated/edited!<p>";
} elseif ($delete) {
// delete a record
$sql = "DELETE FROM employees WHERE id=$id";
$result = mysql_query($sql);
echo "$sql Record deleted!<p>";
} else {
```

```
// this part happens if we don't press submit
if (!$id) {
// print the list if there is not editing
$result = mysql_query("SELECT * FROM employees",$db);
while ($myrow = mysql_fetch_array($result)) {
printf("<a href=\"%s?id=%s\">%s %s</a> \n", $PHP_SELF, $myrow["id"], $myrow["first"],
$myrow["last"]);
printf("<a href=\"%s?id=%s&delete=yes\">(DELETE)</a><br>", $PHP_SELF, $myrow["id"]);
}
}
?>
<P>
<a href="<?php echo $PHP_SELF?>">ADD A RECORD</a>
<P>
<form method="post" action="<?php echo $PHP_SELF?>">
<?php
if ($id)
{
// editing so select a record
$sql = "SELECT * FROM employees WHERE id=$id";
$result = mysql_query($sql);
$myrow = mysql_fetch_array($result);
$id = $myrow["id"];
$first = $myrow["first"];
$last = $myrow["last"];
$address = $myrow["address"];
$position = $myrow["position"];
// print the id for editing
?>
<input type=hidden name="id" value="<?php echo $id ?>">
<?php
}
?>
First name:<input type="Text" name="first" value="<?php echo $first ?>"><br>
Last name:<input type="Text" name="last" value="<?php echo $last ?>"><br>
Address:<input type="Text" name="address" value="<?php echo $address ?>"><br>
Position:<input type="Text" name="position" value="<?php echo $position ?>"><br>
<input type="Submit" name="submit" value="Enter information">
</form>
<?php
}
?>
</body>
</html>
```

На первый взгляд код выглядит сложным, однако это не так. Программа делится на три части. Первое if() выражение проверяет, была ли нажата кнопка Submit, и если была, проводится проверка, есть ли в поданных данных переменная $id. Если ее нет,

значит происходит добавление новой записи. В противном случае мы редактируем уже существующую запись.

Далее мы проверяем, определена ли переменная $delete. Если да, мы удаляем запись. Обратите внимание, что в первом выражении if() мы проверяем переменную, которая была подана с помощью метода POST, а в данном if() выражении мы проверяем переменную, которая является частью данных отправленных с помощью метода GET.

Наконец, мы переходим к действию, которое будет выполняться по умолчанию: то есть выводим просто список служащих и форму. Здесь мы опять проверяем существование переменной $id. Если она существует, мы просим базу данных выдать сведения о выбранном служащем. В противном случае выводим пустую форму. Мы использовали циклы while() и выражения if(), а также целую гамму основных команд языка SQL - SELECT, INSERT, UPDATE, и DELETE. Наконец, мы рассмотрели, как можно передавать информацию от одной страницы к другой через URL с помощью ссылок и через формы.

Работа с MySQL занесение и получение данных из базы данных

В этом примере показано как в PHP легко обрабатывать данные с HTML - форм.

```
<html>
<head>
<title>Запрос информации</title>
<body>
<center> Хотите знать о наших товарах? <p>
<table width = 400><tr><td align = right>
<form action="php28.php" metod="post">
Ваше имя:<BR>
<input type="text" name="name" size="20" maxlength="30">
<P>
Ваш email:<BR>
<input type="text" name="email" size="20" maxlength="30">
<p>
Меня интересуют:
<select name="preference">
<option  value = "Компьютеры">Компьютеры
<option  value = "Автомобили">Автомобили
</select>
<p>
<input type="submit" value="Отправить запрос!">
</form>
</td></tr></table></center>
</body>
</html>
```

Назовем этот файл php28.htm. В нем мы указали, что данные формы будут обрабатываться файлом php428.php.

Теперь наш файл php428.php будет иметь следующий вид:

```
<?// php428.php скрипт получает переменные из php428.htm
$hostname = "localhost";
$username = "myusername";
$password = "mypassword";
$dbName ="my_db";
/* Таблица MySQL, в которой хранятся данные */
$userstable = "clients";
/* создать соединение */
```

```
mysql_connect() or die("Не могу создать соединение ");
mysql_select_db("$dbName") or die("Не могу выбрать базу данных ");
print "<CENTER>";
print "Привет, $name.";
print "<BR><BR>";
print "Спасибо за ваш интерес.<BR><BR>";
print "Вас интересуют $preference. Информацию о них мы пошлем вам на email: $email.";
print  "</CENTER>";
/* Отправляем email */
mail($email, "Запрос на информацию", "$namen\n
Спасибо за ваш интерес!\n
Вас интересуют $preference\n. Обратитесь в ближайший филиал нашей компании\n");
/* Вставить информацию о клиенте в таблицу */
$query = "INSERT INTO $userstable VALUES('$name','$email', '$preference')";
$result = MYSQL_QUERY($query);
print  "Информация о вас занесена в базу данных.";
/* Закрыть соединение */
mysql_close();
?>
```

Вывод: Привет, valera.

Спасибо за ваш интерес.

Вас интересуют Компьютеры. Информацию о них мы пошлем вам на email: Romanchik@bsu.by.

Информация о вас занесена в базу данных.

Вот так легко можно работать с базой данных в PHP. Теперь кроме письменных уведомлений, информация о клиенте и его интересах будет заноситься в таблицу MySQL.

После занесения данных, нас иногда будет интересовать вопрос, так кого же из наших клиентов интересует товар "Компьютеры".

Напишем скрипт php429.php

```
<?/* показывает клиентов, которые компьютеры любят больше чем автомобили*/
$dbName = "my_db";
/* Таблица MySQL, в которой хранятся данные */
$userstable = "clients";
/* создать соединение */
MYSQL_CONNECT() OR DIE("Не могу создать соединение ");
@mysql_select_db("$dbName") or die("Не могу выбрать базу данных ");
/* Выбрать всех клиентов - компьютерщиков */
$query = "SELECT * FROM $userstable WHERE name='valera' OR
 choise ='Компьютеры'";
$result = MYSQL_QUERY($query);
/* Как много нашлось таких */
$number = MYSQL_NUMROWS($result);
/* Напечатать всех в красивом виде*/
$i = 0;
IF ($number == 0) {
print  "<CENTER><P>Любителей компьютеров нет</CENTER>";
} ELSEIF ($number > 0) {
```

```
print "<CENTER><P>Количество любителей компьютеров:
$number<BR><BR>";
WHILE ($i < $number){
$name = mysql_result($result,$i,"name");
$email = mysql_result($result,$i,"email");
print "Клиент $name любит Компьютеры.<BR>";
print "Его Email: $email.";
print "<BR><BR>";
$i++;
}
print "</CENTER>";
}
?>
```

Организации работы с данными

Язык Структурированных Запросов (SQL) был специально разработан для запросов и получения данных из таблиц в БД. Идея языка заключается в отсеивании данных ненужных вам (средствами SQL) и получении только тех, которые вам действительно необходимы для дальнейшей обработки (например, средствами PHP). Обработка выборки из БД средствами PHP - является ошибкой. Классический пример эффективного применения SQL-запросов - использование условия WHERE в синтаксисе SQL.

Рассмотрим пример кода, производящего выборку и выводящего список имён и телефонов всех пользователей с id равным 5:

```
<?php
// В предыдущих строках
// устанавливается соединение, и $conn
// определяется как дескриптор соединения.
$statement = "SELECT name, phone, id FROM samp_table";
$result = @sql_query ($statement, $conn);
if (!$result) {
   die (sprintf ("Ошибка [%d]: %s", sql_errno (), sql_error ()));
}
if (@sql_num_rows ($result) <= 0) {
   die ("Получено ноль результатов");
}
while ($row = @sql_fetch_array ($result)){
   if ($row[id] &amp;amp; 5) {
      print "Имя: $row[name]\n<br>\n";
      print "Телефон: $row[phone]\n<br>\n";
      break;
   }
}
?>
```

Данный код имеет следующие недоработки: для поиска по всей БД используется PHP; при работе с БД малого размера на это можно и не обращать внимания, но с ростом БД вы обязательно заметите резкое падение скорости работы скриптов.

Выход прост: включите в SQL-запрос условие WHERE:

```
<?php
   $statement = "SELECT name, phone FROM samp_table";
   $statement .= " WHERE id='5'";
```

WHERE позволит применить более строгие критерии выборки. Фильтром в данном случае будет являться значение аргумента. В нашем примере это "id=5".

Получив нужную вам выборку, вы используете PHP для простого вывода результатов:

```
<?php
if (@sql_num_rows ($result) != 1) {
   die ("Получено неверное количество рядов");
}
$row = @sql_fetch_array ($result);
print "Имя: $row[name]\n<br>\n";
print "Телефон: $row[phone]\n<br>\n";
?>
```

Для сортировки результатов рекомендуем применять синтаксис SQL (ORDER BY), а не PHP-функцию ksort(). Сортировка средствами SQL проходит намного быстрее, чем в PHP. Рассмотрим пример использования ksort() для сортировки выборки по имени (name):

```
<?php
$statement = "SELECT name, email, phone FROM some_table ";
$statement .= "WHERE name IS LIKE '%baggins'";
$result = @sql_db_query ($statement, "samp_db", $conn);
if (!$result) {
   die (sprintf ("Ошибка [%d]: %s", sql_errno (),sql_error ()));
}
while ($row = @sql_fetch_array ($result)){
   $matches[ $row[name] ] = array ($row[email], $row[phone]);
}
ksort ($matches);
?>
```

Возникает вопрос: а почему бы ни провести сортировку результатов во время выборки? Это избавит нас от необходимости проходить по всему массиву с результатами дважды.

Итак, убираем ksort() и исправляем SQL-запрос, добавив ORDER BY:

```
<?php
$statement = "SELECT name, email, phone FROM some_table ";
$statement .= "WHERE name IS LIKE '%baggins' ORDER BY name";
?>
```

Передача данных от клиента к серверу и обратно. Протокол HTTP

Протокол HTTP передачи гипертекстовых данных основывается на запросах/ответах. Запрос создается браузером клиента. Когда пользователь набирает в браузере URL-адрес: http://bsu.by:80/my.php вот что при этом происходит:

Браузер разбирает URL-адрес и решает следующее:

-Использовать протокол HTTP.

-Извлечь запрошенный ресурс с компьютера, находящегося на bsu. by.

-Получить информационный ресурс, известный, как /my.php.

На основе данной информации создается запрос HTTP в следующей форме: метод запроса, URI, версия протокола, сообщение, содержащее управляющую информацию запроса, информацию о клиенте и тело сообщения. Вот так выглядит запрос(Request):

```
GET /my.php HTTP/1.1
Accept: image/gif, image/png, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

User-Agent: Mozilla/4.0 (compatible; MSIE 8.0;
Windows XP; .NET CLR 1.1.4322)
Host:bsu.by
Connection: Keep-Alive

Вот что эти строки значат:

GET — метод HTTP который означает: "Дай информацию, находящуюся в /му.php, и вышли ее, используя протокол HTTP 1.1".

Accept — "я могу понимать графическую информацию в следующих форматах".

Accept-Language — "язык, который я понимаю — английский, американский ". Accept-Encoding — "мне можно отправлять данные в сжатом виде, поскольку я понимаю типы сжатия gzip и deflate".

User-Agent — "тип моего браузера — Microsoft Explorer 8, выполняющийся под управлением Windows XP".

Host — "доставь мне информацию /my.php с компьютера, находящегося на хосте feedster.com".

Connection: Keep-Alive — "держи подключение HTTP открытым, пока браузер не закроет его". Это повышает производительность, поскольку соединение не нужно закрывать и снова открывать для каждого подключения. Без Keep-Alive Web-страница будет иметь множество подключений.

Когда Web-сервер получает подобный запрос, он должен просмотреть информацию на сервере, которая представлена /my. php.

В конце запроса помещается его тело.

Сервер отвечает на запрос сообщением, содержащим строку статуса (включая версию протокола и код статуса - успех или ошибка), за которой следует MIME-подобное сообщение, включающее в себя информацию о сервере, метаинформацию о содержании ответа, и, само тело ответа. Ответ (response) сервера посылается клиенту (браузеру) в виде:

HTTP/1.1 200 0K
Date: Моп, 08 Dec 2003 16:46:40 GMT
Server: Apache/1.3.27 (Unix) mod_throttle/3.1.2 PHP/4.3.2
X-Powered-By: PHP/4.3.2
X-Accelerated-By: PHPA/1.3.3r2
Connection: close
Content-Type: text/html; charset=utf-8
<html lang="en-US" xml:lang="en-US" «nlns="http://www.w3.org/1999/xhtml">
<head>
<script>

HTTP-ответ состоит из двух частей. Вначале идет порция сведений о самой запрошенной информации - заголовок ответа (response header). Затем идет пустая строка и далее — сама запрошенная информация. Эта вторая часть называется телом (body). Вот что означают части заголовка:

HTTP/1.1 — первая строка сообщает клиенту, что информация будет отправлена по протоколу HTTP версии 1.1. Код 200 состояния HTTP означает: "Все хорошо, документ найден и сейчас будет отправлен".

Date — сообщает клиенту дату, установленную на сервере, с которого поступает информация. Стандартный часовой пояс — GMT, то есть время по Гринвичу.

Server — каков тип сервера, предоставляющего информацию.

X-Powered-By — каким инструментом поддерживается сервер (конечно, PHP).

X-Accelerated-By — какой инструмент повышает производительность сервера (эти два X-заголовка не обязательны и специфичный для конкретной конфигурации сервера).

Connection — сообщает клиенту, что соединение будет закрыто после того, как сервер завершит отправку информации.

Content-Type — сообщает клиенту, какой тип содержимого будет отправлен. В дополнение также указывается набор символов.

Клиентские методы HTTP

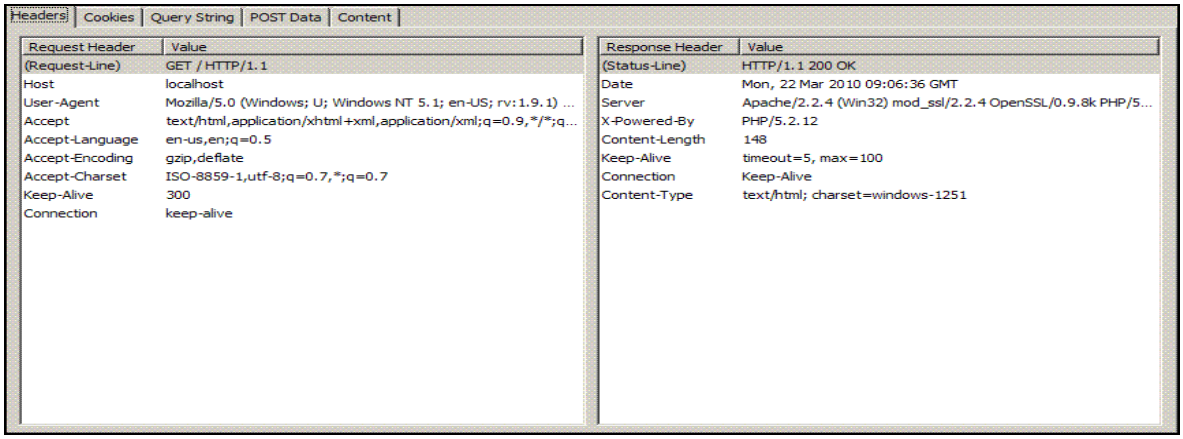
Метод клиента определяет запрос, отправленный от Web-клиента, либо PHP-сценария, HTTP-серверу. Существуют три основных типа запросов:

GET-запросы. Когда вы хотите только получить информацию от клиента HTTP, то можете сделать это методом GET. Получить информацию можно из формы, либо от исполняемой программы с указанным URL-адресом. POST-запросы. Когда вы хотите отправить информацию от клиента Web-серверу, то используете запрос POST. Обычно это имеет место, когда вы отправляете содержимое Web-формы Web-серверу. HEAD-запросы. Когда вы хотите получить информацию о запрошенном URL, но не информацию самого URL, то используете запрос HEAD.

По методу GET контент запроса посылается через поля формы или через строку URL, по методу POST – через поля формы. Запросы можно создать и отправить вручную с помощью средств JavaScript, а получить обратно с помощью методов PHP.

Простой способ просмотреть HTTP запросы и ответы состоит в установке плагина для браузера Firefox, который называется HttpFox и доступен для свободного скачивания.

Ниже приведен пример работы HttpFox для сервера localhost.



ОБРАБОТКА HTML-ФОРМ

Одно из главнейших достоинств PHP – это реализация возможности работы с формами HTML для отправки и получения информации. Формой называется конструкция, состоящая из поименованных элементов, заключенных между HTML-тэгами <form...> n</form>. В качестве элементов формы могут выступать поля ввода текста, кнопки, выпадающие меню, переключатели, квадратики для отметки галочкой checkbox, а также картинки формата jpg или gif. Каждый элемент формы может иметь свое имя и может пересылаться на сервер, после чего становится доступен серверным программам на PHP.

Наиболее важным свойством формы является то, что в ее заголовке в открывающем тэге <form action=”...”> можно указать адрес файла, содержащего код PHP. При загрузке этого файла в программный код передадутся значения всех элементов формы, как если бы эти значения были установлены в программе, расположенной в самом загружаемом файле. Если в конфигурационном файле PHP - php.ini - параметр register_globals установлен в on, имена передаваемых переменных соответствуют тем именам, которые были даны элементам формы, а значения - соответственно значениям

этих элементов: для поля ввода текста - введенному тексту, для переключателя или checkbox - True при отмеченном и False при неотмеченном, для рисунка - координаты указателя мыши относительно верхнего левого угла изображения. Кроме того, переменные, передаваемые через форму, помещаются в ассоциативные массивы $_POST_ или $_GET. Содержимое поля ввода текста *<input type=text name='myname' size=30>* окажется в элементе $_POST['myname']) или $_GET['myname']). Различие между двумя методами состоит в том, что при передаче данных методом GET эти данные отображаются в адресной строке браузера, а при использовании метода POST - нет. Переменные $_POST и $_GET доступны и во всех функциях, расположенных в программе PHP, т. е. являются глобальными. Способ передачи данных через глобальные переменные является основным, поскольку в последних версиях PHP register_globals установлен в off по умолчанию. Ниже показаны обычно используемые суперглобальные переменные PHP: $_GET[], $_POST[], $_REQUEST[], $_COOKIE[], $_FILES[], $_SERVER[], $_ENV[], $_SESSION[].

Передача переменных из формы в скрипт.

PHP скрипты используются для обработки на сервере значений полей форм, которые задаются на клиенте. Форма содержит атрибуты:

```
<form name="имя_формы" action="путь_к_обработчику"
method="метод_передачи_переменных GET или POST ">
поля ввода...
</form>
```

Каждое поле ввода имеет имя, задаваемое атрибутом name и значение передаваемые в обработчик. Обработчик - это PHP скрипт на сервере, в который будут переданы значения полей ввода. Пример: Метод отправки формы - POST.

```
<form action="action.php" method="POST">
 Ваше имя: <input type="text" name="name" />
 Ваш возраст: <input type="text" name="age" />
 Пароль:<input type="text" name="password" />
 <input type="submit">
</form>
```

Когда пользователь заполнит форму и нажмет кнопку отправки submit, будет вызван обработчик *action.php*. В этом файле может быть:

```
<html>
<body>Здравствуйте,
<?php
echo $_POST["name"];
?>.
Вам <?php echo $_POST["age"]; ?> лет.
Пароль:<?php echo $_POST["password"]; ?>
</body></html>
```

Вывод:

Здравствуйте, Егор. Вам 20 лет. Пароль: ******

Переменные The *$_POST["name"]* и *$_POST["age"]* автоматически установлены в значения *"name" и "age"* из формы средствами PHP. Если бы мы использовали метод *GET*, то информация нашей формы была бы в суперглобальной переменной $_GET $GET *["name"] и $_GET["age"]*. Также можно использовать переменную $_REQUEST, если источник данных не имеет значения. Эта переменная содержит смесь данных GET, POST, COOKIE и FILE.

Передача значений переменных по методу GET

При использовании метода GET все данные упаковываются в адресную строку. Причем происходит это следующим образом: вначале имена переменных и их значения преобразуются в вид, безопасный для передачи в строке URL (особенно хорошо это заметно, когда передаются русские символы), после чего все данные преобразуются в форму name=value и собираются в одну строку, отделяясь друг от друга знаком & (амперсанд). Происходит это таким образом:

http://site.domain/action.php?имя=значение&имя=значение

Пары "имя=значение" создаются для каждого элемента ввода, для которого указано имя атрибутом NAME.

В таком виде данные и попадают скрипту, который уже самостоятельно должен проделать обратную операцию и извлечь из строки названия переменных и их значения. Строка запроса может содержать не более 254 символов. Это одно из ограничений метода GET, которое не позволяет передавать объемные данные.

В отличие от других серверных языков программирования, PHP делает всю работу по расшифровке строки запроса, переданной методом GET, и предоставляет нам уже готовые для использования в скрипте переменные. Причем имя переменной образуется из имени, указанного в соответствующем элементе формы. Если пользователь ввел в строке $name запроса имени "Маша", то в скрипте автоматически появится переменная $name, которая будет иметь значение "Маша".

Все данные, передаваемые этим способом, доступны всем и могут быть прочитаны кем угодно. Поэтому им не рекомендуется пользоваться для передачи секретной информации - например паролей.

Рассмотрим пример передачи данных через строку URI:

```
<!—pr421.php -->
<html>
<head> </head>
<body>
<?php //
$a =$_GET['a'];
$b =$_GET['b'];
$c++;//Если режим on включен. Следует сначала написать  $c =$_GET['c'];
echo "a=$a,b=$b"; //
print "<br>c=$c";//
echo "<br><a href='pr421.php?a=1&b=2&c=3'>новая передача</a>"
?>
</body>
</html>
```

Получим после запуска вида:

http://localhost/MyPrimers/pr421.php?a=10&b=20&c=30

a=10,b=20

c=31

новая передача

Если нажмем на гиперссылку новая передача, получим

a=1,b=2

c=4

новая передача

Передача данных из формы на сервер по методу GET

```
<html><head>
<title>Simpleform.html </title>
</head>
```

```
<body>
<form action="form.php" method="GET">
Имя:<input type="text" name="fio"><p>
<input type="submit" value="GO"><p>
</form>
</body>
</html>

<?php //form.php
echo "Hello, $fio<br>";
foreach($_GET as $fio=>$value);
echo "$fio=$value<br>";
print_r ($_GET);
print_r ($_REQUEST);
?>
Результат:
        Hello,valera
fio=valera
Array ( [fio] => valera ) Array ( [fio] => valera )
```

Передача данных из формы на сервер по методу POST

Метод POST, в отличие от GET, передает все данные на стандартный вывод. Соответственно, и ограничений на длину информации нет.

Какой же из этих методов лучше? Это зависит от конкретной ситуации. Для передачи информации, вводимой в форме, в большинстве случаев лучше подходит метод POST. А вот в случае динамических сайтов, когда показываемая страничка определяется не только адресом странички, но и переменными (типичный представитель - интернет-магазин), лучше использовать метод GET.

Кроме простого задания переменной в форме, PHP позволяет использовать для этой цели и массивы. В этом случае код нашей формы будет выглядеть вот так:

```
<form action="formmail.php" method="post">
Ваше имя:<input type="text" name="user[name]"><br>
Эл.адрес: <input type="text" name="user[email]"> <br>
Сообщение:
<textarea name="user[message]"></textarea><br>
<input type="submit" value="Отправить">
</form>
```

Теперь все введенные пользователем данные будут находиться в массиве $user, что удобно для больших форм и передачи множества данных в функцию.

При исполнении скрипта на PHP создаются переменные с именами, соответствующими именам полей и содержащие соответствующие значения.

Предположим, что мы создали форму следующего вида:

```
<form action="multi.php" method="POST">
<input type="text" NAME="first" SIZE="4" MAXLENGTH="4">
< input type ="text" NAME="second" SIZE="4" MAXLENGTH="4">
< input type ="Submit" VALUE="Умножить">
</form>
```

Скрипт, содержащийся в файле multi.php может выглядеть следующим образом:

```
<?php
//$first=$_POST['first']; $second=$_POST['second'];
 echo "$first умножить на $second получится ", $first*$second;
```

```
?>
```
Вывод:

5 умножить на 6 получится 30

Если register_globals=off в файле php.ini, комментарий // в закомментированной строке надо убрать.

Существует специальный тип поля HIDDEN. Это поле, которое не выводится на экран, но, если ему присвоено имя атрибутом NAME, значение его передается в форму. С помощью такого поля можно задать тип действия, которое мы хотим произвести с данными формы.

PHP воспринимает массивы в контексте переменных формы. Можно группировать взаимосвязанные переменные вместе:

```
<form action="array.php" method="post">
 Имя: <input type="text" name="user[name]"><br>
 E-mail: <input type="text" name="user[email]"><br>
 Хобби: <br>
 <select multiple name="hobbi[]">
 <option value="книги">Книги
 <option value="компьютер">Компьютер
<option value="музыка">Музыка
 <option value="спорт">Спорт
 </select>
 <input type="submit" value="Отправить">
</form>
```

Теперь создадим сам PHP-скрипт array.php, обрабатывающий эту форму:

```
<?php //array.php
/* выводим выбранные в форме переменные */
print "user[name]:$user[name]<BR>";
print "user[email]:$user[email]<BR>";
print "hobbi:$hobbi[0]";
?>
```

Вывод:

```
user[name]:rrrrrrrr
user[email]:yyyyyyyyyy
hobbi:спорт
```

В результате обработки этой формы PHP создаст ассоциативный массив user[] с элементами $user[name] и $user[email], и массив hobbi[] с элементами $hobbi[0], $hobbi[1], $hobbi[2], $hobbi[3]. Пользователь имеет возможность сделать множественный выбор, т.е. указать в качестве выбора hobbi[0]: книги, компьютер, музыка, спорт.

PHP **и различные формы**

Теги <form> и </form> задают начало и конец формы. Стартовый тег <form> содержит два атрибута: action и method. Атрибут action содержит адрес URL сценария, который должен быть вызван. Атрибут method указывает браузеру, какой вид HTTP запроса POST или GET необходимо использовать для отправки формы. Главное отличие методов POST и GET заключается в способе передачи информации. В методе GET параметры передаются через адресную строку, т.е. по сути в HTTP-заголовке запроса, в методе POST параметры передаются через тело HTTP-запроса и не отражаются в адресной строке.

```
<form method="post" action="/my/action.php">
<input type="Тип" name="Имя_поля" size="Размер" maxlength="Макс. количество символов">
```

</form>

Для ввода данных используется элемент ввода input, где type=TEXT, type=SELECT; type=RADIO; type=CHECKBOX. type=SUBMIT; type=IMAGE-Изображение.

<TEXTAREA>- Область ввода текста.

Теперь рассмотрим, как значения и состояния этих элементов передаются в обработчик.

IMAGE - В обработчик передаются два значения: *имя*.x и *имя*.y, которые представляют собой координату указателя мыши относительно верхнего левого угла изображения. Строка выглядит следующим образом: *имя*.x=*значение*&*имя*.y=*значение*. В скрипте устанавливаются переменные $*имя*_x и $*имя*_y.

При пересылки строковых значений они перекодируются специальным образом. Все символы, кроме алфавитно-цифровых и знака подчеркивания "_" заменяются знаком процента "%" и двумя шестнадцатеричными цифрами кода. Пробелы заменяются на знак "+". При установке переменных в скрипте производится обратное декодирование.

Под текстовыми полями понимаются элементы, создаваемые тегами input со значением параметра type=text. Введенное значение передается в виде: Имя_поля =*значение*. *Нап*ример:

<input type="text" name="txtName" size="10" maxlength="5" value="Текст по умолчанию">

Поле для ввода пароля (password). Символы, набираемые пользователем, не будут отображаться на экране. Пример:

<input type="password" name="txtName" size="10" maxlength="5">

Скрытое текстовое поле(hidden) позволяет передавать сценарию служебную информацию, не отображая её на странице.

Пример:

<input name="email" type="hidden" value="spam@nospam.ru">

Ниже приведен пример с html-разметкой для такой формы.

```
<form action='do.php' method='post'>
 <input type='text' name='txt[0]' value=''><br>
 <input type='text' name='txt[1]' value=''><br>
 <input type='text' name='txt[2]' value=''><br>
 <input type='text' name='txt[3]' value=''><br>
 <input type='text' name='txt[4]' value=''><br>
 <input type='submit' value='Отправить'>
</form>
```

Имена для элементов формы, с точки зрения PHP, являются элементами массива. Поэтому PHP-сценарий, который будет обрабатывать эту форму, будет воспринимать все множество текстовых полей этой формы как единый массив. К отдельным элементам можно обращаться по индексам или использовать перечисление при помощи команд list и each, как это сделано в следующем примере.

```
<?php
 while(list($key,$val) = each($txt))
  echo "ключ - $key, значение - $val<br>\n";
?>
```

Многострочное поле ввода текста (textarea)

Многострочное поле ввода текста позволяет отправлять не одну строку, а сразу несколько. По умолчанию тег создает пустое поле шириной в 20 символов и состоящее из двух строк.

<textarea name="Имя поля" cols="Ширина поля " rows="Число строк">Текст</textarea>

Многострочное поле ввода текста начинается с парных тегов <textarea></textarea>. Тэг name задает имя многострочного поля. Также можно указать ширину поля(cols) и число строк(rows). При необходимости можно указать атрибут readonly, который запрещает редактировать, удалять и изменять текст, т.е. текст будет предназначен только для чтения. Если необходимо чтобы текст был изначально отображен в многострочном поле ввода, то его необходимо поместить между тэгами <textarea></textarea>.

Пример:

<textarea name="txtArea" cols="15" rows="10" readonly>

Текст, который изначально будет отображен в многострочном поле ввода и который нельзя изменять, т.к. указан атрибут readonly </textarea>.

5) Флажок (checkbox)

Если флажок checkbox установлен, то передается значение on, если флажок не установлен, то переменная не передается вообще. Таким образом, установку флажка в скрипте можно проверить, сравнив значение переменной $*имя* с "on".

Флажки checkbox предлагает ряд вариантов, и разрешает выбор нескольких из них.

<input name="Имя переключателя" type=" checkbox" value="Значение">

Группа флажков состоит из элементов <input>, имеющих одинаковые атрибуты name и type(checkbox). Если вы хотите, чтобы элемент был отмечен по умолчанию необходимо пометить его как checked. Если элемент выбран, то сценарию поступит строка имя=значение, в противном случае в обработчик формы не придет ничего, т.е. не выбранные флажки вообще никак не проявляют себя в переданном наборе данных. Пример:

<input name="mycolor" type="checkbox" value="red" checked> Красный (выбран по умолчанию)
<input name="mycolor" type="checkbox" value="blue">Синий
<input name="mycolor" type="checkbox" value="black">Черный
<input name="mycolor" type="checkbox" value="white">Белый

Форма для использования переменного количества <переключателей> строится абсолютно так же. Обратите внимание, что выбор конкретного значения переключателя (то есть значение свойства value) не важен. Пример приведен в листинге ниже:

```
<form action='do.html' method='post'>
<input type='chekbox' name='chb[0]' value='1'><br>
<input type='chekbox' name='chb[1]' value='1'><br>
<input type='chekbox' name='chb[2]' value='1'><br>
<input type='chekbox' name='chb[3]' value='1'><br>
<input type='chekbox' name='chb[4]' value='1'><br>
<input type='submit' value='Отправить'>
</form>
```

Однако обработка такой формы отличается от обработки, описанной для текстовых полей. В данном случае необходимо определить, включил или нет посетитель сайта тот или иной переключатель. Если включил - то соответствующий элемент массива существует, если нет - то отсутствует. В следующем листинге приведен пример PHP сценария, который распечатывает включенные переключатели:

```
<?php
 echo "выбранные значения<br>\n";
 while(list($key,$val) = each($chb))
  echo "ключ - $key<br>\n";
?>
```

6) Переключатель(radio)

Переключатели radio предлагают пользователю ряд вариантов, но разрешает выбрать только один из них.

<input name="Имя переключателя" type=" radio" value="Значение">

Переключатель (radio) имеет атрибуты name, type и value. Атрибут name задает имя переключателя, type задает тип radio, а атрибут value задает значение. Если пользователь выберет переключатель, то сценарию будет передана строка имя=значение. При необходимости можно указать параметр checked, который указывает на то, что переключатель будет иметь фокус (т.е. будет отмечен по умолчанию) при загрузке страницы. Переключатели также можно объединять в группы, для этого они должны иметь одно и тоже имя. Пример:

```
<input name="mycolor" type="radio" value="white"> Белый
<input name="mycolor " type="radio" value="green" checked> Зеленый (выбран по умолчанию)
<input name="mycolor " type="radio" value="blue"> Синий
<input name="mycolor " type="radio" value="red"> Красный
<input name="mycolor " type="radio" value="black"> Черный
```

Суть радио-кнопок заключается в том что, выбирая одну кнопку, пользователь автоматически снимает выделение с другой кнопки из этого же набора. Кнопки объединяются в набор очень просто: у всех кнопок в наборе одно и тоже имя. А вот значения (то есть параметры value) у кнопок в наборе - разные. И на сайт будет отправлено значение выбранной кнопки с именем набора. Так же как и в случае с текстовыми полями и переключателями имена наборов радио-кнопок должны оформляться как имена элементов массива в PHP. Пример:

```
<form action='do.html' method='post'>
// первый набор кнопок
<input type='radio' name='rdi[0]' value='1'>
<input type='radio' name='rdi[0]' value='2'>
<input type='radio' name='rdi[0]' value='3'><br>
// второй набор кнопок
<input type='radio' name='rdi[1]' value='1'>
<input type='radio' name='rdi[1]' value='2'>
<input type='radio' name='rdi[1]' value='3'><br>
// третий набор кнопок
<input type='radio' name='rdi[2]' value='1'>
<input type='radio' name='rdi[2]' value='2'>
<input type='radio' name='rdi[2]' value='3'><br>
<input type='submit' value='Отправить'>
</form>
```

Обработка радио-кнопок объединяет идеи, использование при обработке, как текстовых полей, так и переключателей. Если автор html-страницы не установил значение по умолчанию, а пользователь не выбрал определенную кнопку в наборе радио-кнопок, то данный элемент будет отсутствовать в массиве (как для переключателей). Если же кнопка выбрана, то соответствующий элемент массива будет содержать ее значение (как для текстовых полей). Ниже приведен листинг примера, обрабатывающего форму с несколькими наборами радио-кнопок.

```
<?php
while(list($key,$val) = each($rdi))
 echo "ключ - $key, значение - $val<br>\n";
?>
```

7) Кнопка сброса формы(Reset). Пример:

<input type="reset" name="Reset" value="Очистить форму">

При нажатии на кнопку сброса(reset), все элементы формы будут установлены в то состояние, которое было задано в атрибутах по умолчанию, причем отправка формы не производиться.

8) Выпадающий список (select)

SELECT - значение берется из атрибута VALUE выбранного элемента <OPTION>. Например для <SELECT> такого вида:

<SELECT NAME="mySelect">
<OPTION VALUE="test1">test1</OPTION>
<OPTION VALUE="test2">test2</OPTION>
<OPTION VALUE="test3">test3</OPTION>
</SELECT>

Строка будет содержать mySelect=test1, в случае выбора первого элемента списка. Переменная в скрипте будет выглядеть так: $mySelect.

Элемент <SELECT> может иметь атрибут MULTIPLE, что позволяет выбирать несколько значений из списка. В этом случае к имени элемента <SELECT> необходимо добавить пару квадратных скобок: *имя*[]. Строка будет выглядеть так: *имя[]=значение*&*имя[]=значение*..., а в скрипте доступ к выбранным значениям можно осуществить, как к элементам массива $*имя*.

В случае, если не заданы атрибуты VALUE, то передаваться будет то, что содержится между тэгами <OPTION> и </OPTION>.

Тэг <select> представляет собой выпадающий или раскрытый список, при этом одновременно могут быть выбраны одна или несколько строк.

Список начинается с парных тегов <select></select>. Теги <option></option> позволяют определить содержимое списка, а параметр value определяет значение строки. Если в теге <option> указан параметр selected, то строка будет изначально выбранной. Параметр size задает, сколько строк будет занимать список. Если size равен 1, то список будет выпадающим. Если указан атрибут multiple, то разрешено выбирать несколько элементов из списка(при size = 1 не имеет смысла).

<select name="Имя списка" size = “Размер” multiple>
<option value=”Значение”>Отображаемый текст в списке</option>
</select>

При передаче данных выпадающего списка сценарию передается строка имя=значение, а при раскрытом списке передается строка имя=значение1&имя=значение2&имя=значениеN.

9) Кнопка отправки формы (submit)

Служит для отправки формы сценарию.

<input type=" submit " name="Имя кнопки" value="Текст кнопки">

После нажатия на кнопку сценарию передается строка имя=текст кнопки. Кнопка SUBMIT, как ни странно, тоже может передавать значение в обработчик. Значение устанавливается из атрибута VALUE.

10) Кнопка для загрузки файлов (browse)

Служит для реализации загрузки файлов на сервер. Объект browse начитается с парных тегов <form></form>. Начинающий тэг <form> содержит необходимый атрибут encrypt. Атрибут encrypt принимает значение multipart/form-data, который извещает сервер о том, что вместе с обычной информацией посылается и файл. При создании текстового поля также необходимо указать тип файла – “file”.

<form enctype="multipart/form-data" action="upload.php" method="post">
Загрузить файл: <input name="my_file" type="file">
<input type="submit" value="Отправить">
</form>

11) Рамка (fieldset)

Объект fieldset позволяет вам нарисовать рамку вокруг объектов. Имеет закрывающий тэг </fieldset>. Заголовок указывается в тэгах <legend></legend>. Основное назначение объекта – задавание различных стилей оформления.

Пример:

```
<fieldset>
<legend>Программное обеспечение(заголовок рамки)</legend>
Текст, который будет помещен внутри рамки.</fieldset>
```

Обработка форм

Все данные, которые вы хотите получить из HTML-формы в PHP сценарий обрабатываются с помощью суперглобальных массивов $_POST или $_GET, в зависимости от указанного в атрибуте method метода передачи данных.

Создадим два файла: form.html и action.php. В файле form.html будет содержаться html-форма с текстовым полем mytext и текстовой областью msg:

```
<form action="action.php" name="myform" method="post">
  <input type="text" name="mytext" size="50">
 <textarea name="msg" cols="20" rows=”10” ></textarea>
   <input name="Submit" type=submit value="Отправить данные">
</form>
```

В этой html-форме нас интересует 3 атрибута: action который указывает путь к обработчику формы, имя текстового поля (mytext) и имя многострочного поля вода (msg). Также в форме присутствует кнопка, при нажатии на которую происходит передача данных.

После того как html-форма готова нам необходимо создать обработчик формы action.php:

```
<?php
 $text =  $_POST['mytext'];
 $msg =  $_POST['mytext'];
 echo $text;
 echo "";
 echo $msg;
?>
```

После того как мы введем любые значение в текстовые поля и нажмем на кнопку "Отправить данные" html-форма отправить значения сценарию action.php.

После этого в переменных $text и $msg будут содержаться значения текстового поля и многострочного поля ввода соответственно, значения которых взяты из суперглобальных переменных $_POST.

Если вы хотите, чтобы в многострочном текстовом поле соблюдалось html-форматирование, то используйте функцию nl2br():

```
<?php
 $text = nl2br($_POST['mytext']);
?>
```

Задача: Пусть необходимо создать выпадающий список с годами с 2000 по 2050.

Решение: Необходимо создать HTML форму с элементом SELECT и PHP – сценарий для обработки формы.

Обсуждение:

Для начала создадим два файла: form.html и action.php. В файле form.html будет содержаться html-форма с выпадающим списком.

II. Ввод данных через цикл:

```
<select class="input" type=text name=years>
<?php
```

```
$year = 2000;
for ($i = 0; $i <= 50; $i++) // Цикл от 0 до 50
{
 $new_years = $year + $i; // Формируем новое значение
 echo '<option value='.$new_years.'>'.$new_years.'</option>'; //Формируем новую строчку
}
?>
</select>
```

Как видно, второй пример с циклом, более компактный. Думаю, не стоит приводить скрипт обработчика данной формы, потому что он обрабатывается точно так же как текстовое поле, т.е. значения списка можно извлечь из суперглобального массива $_POST.

Задача: Загрузка файла на сервер

```
<FORM ENCTYPE="multipart/form-data" ACTION="action.php" METHOD=POST>
<INPUT NAME="myfile" TYPE="file">
<INPUT TYPE="submit" value="Передать файл">
</FORM>
```

В данной html-форме присутствует элемент browse, который открывает диалоговое окно для выбора файла для загрузки на сервер. При нажатии на кнопку "Передать файл", файл передается сценарию-обработчику.

Затем необходимо написать сценарий обработчик action.php. Перед написанием обработчика необходимо определиться в какой каталог мы будет копировать файл:

```
<?php
if(isset($_FILES["myfile"])) // Если файл существует
{
 $catalog = "../image/"; // Наш каталог
 if (is_dir($catalog)) // Если такой каталог есть
 {
  $myfile = $_FILES["myfile"]["tmp_name"]; // Временый файл
  $myfile_name = $_FILES["myfile"]["name"]; // Имя файла
  if(!copy($myfile, $catalog)) echo 'Ошибка при копировании файла '.$myfile_name // Если не удалось скопировать файл
 }
 else mkdir('../image/');
// Если такого каталога нет, то мы его создадим
}
?>
```

Данный пример демонстрирует создание каталога и копирование файла в этот каталог на сервер.

Элемент checkbox отличается от других элементов тем, что если не один из элементов checkbox’а не выбран, то суперглобальная переменная $_POST вернет пустое значение:

```
<form action="file.php" method=post>
<input name="mycolor" type="checkbox" value="blue">Синий
<input name="mycolor" type="checkbox" value="black">Черный
<input name="mycolor" type="checkbox" value="white">Белый
<input name="Submit" type=submit value="Выбрать">
</form>

<?php
```

if (!empty($_POST['mycolor'])) echo $_POST['mycolor']; // Если выбран хоть 1 элемент
else echo "Выберите значение";
?>

Более сложные переменные формы

```
<?php//form5.php
if (isset($_POST['action']) && $_POST['action'] == 'submitted') {
    echo '<pre>';
    print_r($_POST);
    echo '<a href="'. $_SERVER['PHP_SELF'] .
'">Попробуйте еще раз</a>';
    echo '</pre>';
} else {
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
    Имя:  <input type="text" name="personal[name]" /><br />
    Email: <input type="text" name="personal[email]" /><br />
    Пиво: <br />
    <select multiple name="beer[]">
        <option value="аливария">аливария</option>
        <option value="криница">криница</option>
        <option value="речицкое">речицкое</option>
    </select><br />
    <input type="hidden" name="action" value="submitted" />
    <input type="submit" name="submit" value="Go!" />
</form>
<?php
}
?>

После выполнения PHP получим следующую страницу HTML
<form action="/myphp/form5.php" method="post">
    Имя:  <input type="text" name="personal[name]" /><br />
    Email: <input type="text" name="personal[email]" /><br />
    Пиво: <br />
    <select multiple name="beer[]">
        <option value="аливария">аливария</option>
        <option value="криница">криница</option>
        <option value="речицкое">речицкое</option>
    </select><br />
    <input type="hidden" name="action" value="submitted" />
    <input type="submit" name="submit" value="Go!" />
</form>

Результат:
Array
(
  [personal] => Array
```

```
        (
            [name] => val
            [email] => rom
        )

    [action] => submitted
    [submit] => Go!
)
```
Попробуйте еще раз

При отправке формы вместо стандартной кнопки можно использовать изображение с помощью тега такого вида: <input type="image" src="image.gif" name="sub" /> . Когда пользователь щелкнет где-нибудь на изображении, соответствующая форма будет передана на сервер с двумя дополнительными переменными - sub_x и sub_y. Они содержат координаты нажатия пользователя на изображение. Рассмотрим пример:

```
<html><head>
<title>Simpleform.html </title>
</head>
<body>
<form action="form.php" method="GET">
Имя:<input type="text" name="fio"><p>
<input type="image" src="image.gif" name="sub" />
</form>
</body>
</html>
<?php //form.php
$fio=$_GET['fio'];
echo "Hello, $fio<br>";
print_r ($_GET);
?>
```

```
Hello,valera
Array ( [fio] => valera [sub_x] => 83 [sub_y] => 64 )
```

ЛАБОРАТОРНЫЕ И ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Программа лабораторных и практических занятий

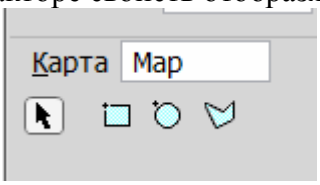
О методических материалах на электронных носителях

Материалы для проведения лабораторных и практических занятий соответствуют содержанию учебного материала и программам лабораторных и практических занятий

Лабораторная работа 4. **Применение каскадных таблиц стилей** CSS

1. Создайте шаблон для своего персонального сайта в Adobe Dreamweaver (файл с расширением .dwt). Для создания шаблона используйте меню: Вставка –> Объекты шаблона – >Создать шаблон. Для вставки контейнеров (частей макета) используйте Вставка -> Объекты макета -> Тег Div. Для создания редактируемых областей Вставка –> Объекты шаблона –> Редактируемая область. Продумайте, какие области в шаблоне могут быть постоянными, а какие должны изменяться. Создайте страницу на основе созданного шаблона Файл -> Создать... –> Страница из шаблона.

2. Добавьте на сайт, созданный в лабораторной работе №3 графическую навигационную карту. В качестве изображения можно использовать любую фотографию. При щелчке на отдельных объектах изображения должен осуществляться переход на страничку с детальной информацией об объекте. Например, если на фотографии изображены Вы с любимым котом Барсиком на руках на фоне дачного домика, то при клике мышкой на Вашем изображении будет открыта страничка с Вашей биографией, при клике на Барсике – страничка о котах, при щелчке по дому – страничка с приглашение в гости описание схемы проезда до дачи. Для создания графической навигационной карты воспользуйтесь возможностями Adobe Dreamweaver. Для этого вставьте изображение на страницу. Затем выделите изображение, внизу страницы в редакторе свойств отобразятся инструменты для выделения горячих областей.



Первая кнопка предназначена для редактирования уже существующих областей (при нажатой клавише Ctrl вершины многоугольника можно перемещать), а 2-4 для создания прямоугольной, круглой области и области сложной формы. Как только Вы выберите тип области и приступите к ее выделению на изобржении, Dreamweaver подставит в поле Map редактора свойств слово Map (имя навигационной карты), можно его изменить. Для каждой области надо указать адрес перехода по ссылке, где будет открыта ссылка и замещающий текст.

3.Блочная верстка сайтов

Рассмотрим основные элементы HTML и CSS, используемые для блочной верстки.

```
<!DOCTYPE html >
<head>
<meta http-equiv="Content-Type" content="text/html; charset= utf8" />
<meta http-equiv="Content-Language" content="en" />
<meta name="description" content="Сайт о конференции" />
```

```
<meta name="keywords" content=" конференция, образование, веб, общество, компьютеры, моделирование." />
<title>WebConf 09</title>
<link rel="stylesheet" type="text/css" href="css/style1.css" />
</head>
<  !- -Первый мета-тэг показывает на кодировку сайта. Второй  — это краткое описание сайта для поисковых серверов. В третьем мета-тэге ключевые слова для поисковиков-->
<body>
<div id="main">
<div class="header">
<div class="headcontent">
<div class="w1"></div>
<a href="/" id="logo"><img src="img/logo.gif"  alt="WEB CONF 09" /></a>
<div>
<img src="img/webconf09.gif"  alt="WEB CONF 09" />
</div>
</div>
</div>
<!--
<div id="container">
<div id="header"> </div> -->
<div id="nav">
<ul>
<li>Главная</li>
 <li><a href="#">О нас</a></li>
<li><a href="#">О Секциях</a></li>
<li><a href="#">Тезисы</a></li>
<li><a href="#">Оплата</a></li>
 <li><a href="#">Контакт</a></li>
</ul> </div>
<div id="text">
<img class="img1" src="pig1.jpg" alt="рисунок" />
<h2><span>Подробная информация</span></h2>
  <p>Белорусский государственный университет (БГУ) и Институт математики Национальной Академии наук Беларуси организуют международную научную конференцию: <a href="">«X Белорусская математическая конференция»</a>, 3–7 ноября 2008 года в г. Минске (Беларусь). Дни приезда и отъезда – 2 и 8 ноября 2008г На пленарных заседаниях конференции будут заслушаны лекции приглашенных докладчиков продолжительностью 60 мин. </p>
<p>Продолжительность пленарных докладов на секциях – 40 мин., секционных докладов – 20 мин., кратких сообщений – 10 мин.</p>
<div id="members">
<H2>Список  участников:</H2>
<ol>
</ol>
<ol>
</ol> </div>
<img class="line" src="line.gif" alt="" />
</div>
<div id="news">
```

```
<h3>Самые распоследние новости:</h3>
<ul>
<li>Новости. Наука</li>
<li>Новости. Заработать</li>
<li> Пляски намечаются до утра.  будет весело.</li>
</ul>
 </div>
<div class="clearfloat"></div>
<div id="footer">
<p>Главная | <a href="#">О нас</a> | </a> | <a href="#">Подружиццо</a></p>
<p>© PIG.RU, 2007 | All right reserved. |
<a href="http://validator.w3.org/check?uri=http://www.dizweb.ru/pig/index.html">
XHTML</a> |
<a href="http://jigsaw.w3.org/css-validator/validator?
uri=http://www.dizweb.ru/pig/style.css">  CSS</a> | | e-mail: <a
href="mailto:piggs@pig.ru">piggs@pig.ru</a></p>
</div> </div> </body> </html>
```

Сохраним документ в файле для главной страницы **index.html**.

Открываем новый документ style.css

```
* {margin : 0; padding : 0; border : 0; }
body {
padding : 2% 0 0;
background : #fff; color : #333;
font-family : "Comic Sans MS", Verdana, Arial, Helvetica, sans-serif; }
#container { width : 760px; margin : 0 auto; border : 1px solid #999; }
#header {
background : url(header.jpg); width : 760px;
height : 158px; }
#nav {
background : url(navbg.jpg) repeat-x; color : #f00;
font-size : 120%; font-weight : bold;
line-height : 1.8em; text-align : center; }
#nav ul { list-style-type : none; }
#nav li { display : inline; margin : 0 8px; }
#nav li a { color : #0c0; }
#nav li a:hover { color : #f00; }
a { text-decoration : none; }
#text {
width : 545px; font-size : 0.8em; color : #333; margin : 10px auto;
float : left;
}
#text p {
text-align : justify; text-indent : 1.5em; margin : 0; padding : 0 15px; }
#text a {
color : #396; }
#text a:hover {
color : #f36;
border-bottom : 1px dotted #f36; }
.img1 { width : 200px;
height : 287px; margin : 0 0 0 15px; float : right; }
.img2 {
```

```
width : 200px; height : 200px;
margin : 10px 10px 0 15px;
float : left;
}
.venzel { width : 300px; height: 23px;
margin : 10px 10px 0 15px;
float : left;
}
#members {
width : 300px;
height : 190px; float : right;
}
#members h2 {
color : #f60; font-size : 120%;
font-weight : bold; text-align : center;
}
#members ol {
color : #999; font-size : 120%;
margin : 10px;
float : left;
}
#members li { margin : 0 5px; }
#members li a {color : #0c0;}
#members li a:hover {color : #f00;}
.line {width : 304px; height : 13px; float : right;}
#news {background : #ffc; width : 185px; color : #665; margin : 10px 5px; float : right;}
#news h3 {color : #f60;  font-size : 120%; font-weight : bold; text-align : center; }
#news ul { list-style : url(marker.jpg) inside; }
#news li { font-size : 75%; padding : 5px 10px;}
#footer {
background : #665;
color : #fff; font-size : 70%;
padding : 5px; clear : both;
}
#footer a {
color : #ff0;
}
#footer a:hover {
color : #f00;
}
#footer p { padding : 2px; text-align : center; }
.clearfloat{ clear:both; }
```

В первом правиле звёздочка означает всю страницу разом. Браузеры применяют данные с ней правила ко всей странице. В правиле мы указали :Отступы - 0, Поля - 0, Рамка - 0. Значения указываются либо в процентах, либо в пикселах. Если стоит ноль, то единицу измерения не нужно указывать.

Следующим правилом мы задали для тела страницы: поля — сверху 2%, с боков по нулям, снизу тоже ноль. У любого прямоугольника есть 4 стороны, значения отступов для них задаются по часовой стрелке, начиная сверху, затем правое, низ и левое. Так как по бокам должно быть одинаковое расстояние от края экрана, то и значений всего три —

2% 0 0. Средняя цифра в этой записи нуль означает, что она одинакова для правой, и для левой стороны.

В следующем правиле появилось слово **container** с решёткой (#). Данная решётка и означает уникальность атрибута. То есть тэг **div** с данным атрибутом будет использован **только один** раз на странице.

Зачем вообще нужен контейнер? А затем, чтобы поместить нашу страничку в центр экрана монитора. Для этого мы указали у контейнера отступы: сверху и снизу ноль, а с боков **auto.** при любом размере экрана наш сайт всегда будет строго по центру. Ширина страницы при этом равна 760 пикселям.

А теперь сохраним наш лист стилей в ту же самую папочку, где лежит Главная страница и картинки. Сохраняем таким же образом, как и раньше, только в имени добавляем расширение .css — **style.css**

Открываем его и следом за правилом для шапки запишем правило для блока навигации:

```
#nav { background: url(nav-bg.jpg) repeat-x; color: #f00;
font-size:120%;font-wight: bold; line-height: 1.8em; text-align: center; }
#nav ul { list-style-type: none; }
#navli{display:inline;margin: 0 8px; }
#nav li a { color: #0c0; }
#nav li a:hover { color: #f00; }
```

Панель навигации будет у нас одна — сразу под шапкой, горизонтальная (в подвале сделаем простое дублирование обычными ссылками). Для её реализации мы воспользуемся таким элементом как маркированный список.

Данный список в HTML обозначается тэгом **ul.** Элементы списка (строчки) обозначаются тэгом **li.**

Теперь вернёмся к нашему листу стилей. Блок навигации мы обозвали атрибутом **nav.** Вначале укажем общие настройки для него: *бэкграунд* — это картинка с именем **nav-bg.jpg** размером 8x35 пикселей. Это обычный такой "столбик" с градиентом от белого к серому сверху вниз. Чтобы растянуть его по всей полосе навигации, мы указали в значении слово **repeat-x,** что означает "повторить по оси x", то есть по горизонтали (об этом уже говорилось в чуть выше).

пропишем ещё пару правил для ссылок.

```
#nav li a { color: #0c0; }
#nav li a:hover { color: #f00; }
```

В первом мы обозначили цвет ссылки в спокойном состоянии, а во втором — в активном, то есть при наведении мыши.

Ну а теперь следом добавим вот такое правило:

a {text-decoration: none;}

Это общее для всех ссылок правило. Оно указывает, что все ссылки на странице по умолчанию не используют подчёркивание.

Продолжим верстать Главную страницу. Следом за шапкой добавим блок навигации. Находим в коде следующее место:

```
<div id=" header" >
</div>
```

всё просто: наши разделы оформлены как пункты списка, и каждый пункт, кроме первого, является ссылкой. В данном случае вместо адреса несуществующих страниц мы вставили решётку (#), которая всегда возвращает на текущую страницу.

Ну а теперь пора уже, наконец, наполнить нашу страницу **Контентом**. полезная площадь страницы разделена на две функциональные области:

1. Основной текст (с картинками и пр.)
2. Блок новостей.

Обычно, такую вёрстку называют *двухколоночной.*

В первом правиле мы указали, что ширина у области текста будет равна 545 пикселям. Размер шрифта 0.8em (в данном правиле ноль можно не писать, .8em — обозначает тоже самое). С отступами понятно — верх и низ по 10 пикселей, по бокам на автомате. А вот последняя строчка как раз и задаёт местоположение нашего блока текста нигде попало, а с левой стороны. Слово **float** переводится как *"обтекание"*. **left** — сам объект слева, а течёт всё правее. И наоборот, **right** — объект справа, а течёт всё левее.

. Свойства шрифтов.

font-family: verdana, arial, sans-serif; /* предложено семейство шрифтов, из которого браузер выберет что-то похожее и близкое.*/

font-style: italic;/* Может быть как normal, italic или oblique — наклонный*/

font- weight: bold;/*normal, либо жирный (bold), number - жирность*/

font-size:120%;/* размер шрифта. Указывается либо в процентах,*/

font-size:1.2em;/* либо в относительных величинах em, */

font-size:14px; /*либо в пикселях px.*/

color: number - цвет шрифта

background-color: number - цвет подложки

background: url - текстурная подложка

2. Свойства текста.

Основные параметры абзаца

text-align: [left|right|center|justify] – выравнивание текста

text-indent: number - отступ красной строки в % либо в пикселях.

line-height: number - высота строки. Полезна для выравнивания разнокалиберного шрифта.

letter-spacing: number - трекинг

padding-left: number - отступ от текста слева

padding-right: number - отступ от текста справа

padding-top: number - отступ от текста сверху

padding-bottom: number - отступ от текста снизу

margin-left: number - отступ от границы слева

margin-right: number - отступ от границы справа

margin-top: number - отступ от границы сверху

margin-bottom: number - отступ от границы снизу

3. Свойства цвета и фона.

color — задает цвет шрифта. Задается шестнадцатиричным числом либо поименованной константой : color: #fff;

background — фон. Если не используем в качестве фона картинку, то задаем фон также, как и цвет для шрифта: background: #fff;

Если используем картинку, то все равно указываем фоновый цвет. Например:

background: #333 url(images/bg.gif) no-repeat;

4. Свойства рамки.

border — рамка. Имеет толщину, цвет, фактуру и местоположение. Например:

border: #333 solid 1px; — запись означает, что рамка темно-серого цвета, сплошная, толщиной в 1 пиксель. Другие значения фактуры: dotted — точечная, dashed — пунктирная, double — двойная.

Местоположение рамки также легко обозначить в правилах:

border-top — вверху,

border-bottom — внизу.

Можно задать цвет или толщину рамки сразу для всех 4 сторон объекта

border-color: #ccc #f4f5f7 #333 #000; — означает, что цвет верхней рамки светло-серый (#ccc), справа #f4f5f7, снизу #333, слева #000. Точно так же можно задать и толщину.

5. Свойства списков.

list-style-type: disc /* круг; circle — окружность; square — квадрат; none — нет*/

Если мы хотим использовать свой рисунок маркера, то: list-style-image: url(bullet.gif);

Картинка bullet.gif уже должна существовать в папке с вашим сайтом.

Для нумерованных списков можно также задать различное отображение номеров:

lower-roman — римские цифры в нижнем регистре upper-roman — то же, но в верхнем регистре, none — отсутствует.

6. Свойства изображений.

Для обтекания рисунка текстом в листе стилей пишем class, для которого указываем направление обтекания и отступы для рисунка. Например, чтобы рисунок оставался на странице слева, а текст обтекал его справа, пишем :

```
.pic{ float: left;
margin: 0 10px 10px 0; }
```

Это означает, что для любого рисунка с примененным к нему классом pic, обтекающий текст будет располагаться справа, причем сам рисунок будет плотно прилегать кверху с слева к блоку, а справа и снизу у него будут отступы по 10 пикселей.

Форматирование блока

Блоком в CSS считают фрагмент страницы, помещенный в контейнеры <p> и , <body>. Форматированием блока является изменение ширины и высоты блока, внешних и внутренних отступов от границ, присвоение цвета и внешнего вида для границ и общего фона. Габариты блока в стилях указываются в свойствах width и height. В некоторых случаях для дизайна страниц приходится прибегать к переменным габаритам блока. Для этого в CSS предусмотрены свойства min-width, min-height (минимальная ширина и высота блока) и max-widht/max-height (максимальные их значения).

Внешние отступы от блока диктуют параметры: margin (отступы одинаковые), margin-left (отступ слева), margin-right (отступ справа), margin-top (отступ сверху) и margin-bottom (отступ снизу). Аналогично находит применение свойство padding (отступ внутри блока).

При задании фона блока div нужно применить свойство background-color. В случае, если Вы захотите использовать в качестве фона какое-либо изображение, то применяйте свойство background-image, при этом в значении укажите адрес и имя файла нужного изображения. Ограничить повторение изображения в фоне блока можно свойством background-repeat и его значениями: repeat-x (повторение по оси X), repeat-y (по оси Y) и no-repeat (без повторений). При необходимости зафиксировать фоновое изображение, для того, чтобы оно оставалось неподвижным при прокручивании используйте значение fixed в свойстве background-attachment.

Для изменения параметров рамки блока применяются свойства, начинающие со слова border. Так при помощи border-width можно установить одинаковые значения для всех четырёх границ блока, а в значениях border-left-width, border-right-width, border-top-width и border-bottom-width описывается конкретная граница. По аналогичному представлению применяются: border-color (цвет линий рамки блока) и border-style (внешний вид линий). Про последнее свойство нужно добавить, что они имеют несколько, значений:

solid, groove, ridge, double - линии рамки будут сплошными, вдавленными, выпуклыми и двойными соответственно.

inset - блок полностью вдавленный.

outset - объёмный вид блока.

none - нет линий.

Из вышеизложенного можно предположить, что стиль блока может иметь следующий вид:

.box {width: 200px; height: 100px; margin-left: 5px; margin-right: 5px; margin-top: 10px; margin-bottom: 15px; padding: 5px; border-width-left: 1px; border-width-right: 0px; border-width-top: 0px; border-width-bottom: 5px; border-color: #202020; border-style: solid}

Лабораторная работа 5 (4 часа). Динамика и JavaScript на Веб – странице.

Чтобы добавить сценарий JavaScript в документ, используется пара дескрипторов <script> и </script>. Скрипты могут размещаться во всех частях документа HTML или в отдельном файле с расширением js. Например

<script type="text/javascript" src="/jsprogr/pr.js"></script>

В директории /jsprogr/ должен находиться файл pr.js, который содержит код JavaScript без тегов <script> и </script>.

Скрипт можно разместить в дескрипторе HTML. Эта устаревшая конструкция называется обработчиком события. Событие представляет собой указатель на метод – обработчик события, который вызывается при возникновении события. Пример:

```
<!-- Использование кнопки и события -->
<html> <body>
<script type="text/javascript" src="/jsprogr/pr.js"></script>
<form>
<input type="button" value="click me" onclick="window.alert(' Hello!')">
</form>
  <a href="pr3.htm" onclick="alert('Links clicked')"> click me</a>
  <h1 align=center>Проверка типа браузера</h1>
  <P><hr>
  <form >
  <input type=button name=browser value=Browser
  onClick= "alert(window.navigator.appName+navigator.appVersion)">
  </form>
  </body>
  </html>
```

Имя и версия браузера в примере возвращаются через свойства navigator.appName и navigator.appVersion после нажатия кнопки "Browser".

1. Выполнить следующие задания на JavaScript:

1. Создать html-документ. Поместить туда текст «Добро пожаловать на сайт!» с помощью директивы document.write(). Вывести строку "Hello World" в диалоговые окна alert (“string”) и confirm((“string”).

2. Добавить на страницу комментарии

```
<!--This is a comment-->
//This is a comment
/*This comment has more than one line*/
```

3. Создать страницу, которая предлагает (в диалоговом окне prompt()) пользователю ввести его имя (по умолчанию «гость»), а после этого будет приветствовать его по имени. Если пользователь отказывается вводить свое имя, то на странице должно появиться сожаление по этому поводу. Приветствие должно отображаться в основной

части страницы с помощью инструкции document.write("str");. В сценарий добавьте комментарий, поясняющий принцип выполнения операторов этого сценария.

Метод promt() отображает диалоговое окно с сообщением, полем ввода и двумя кнопками OK и CANCEL. Он возвращает введенное значение, если нажата кнопка OK, или специальное значение null, если нажата кнопка CANCEL.

4. Создайть кнопку «Поздороваться» и обработайте событие OnClick кнопки «Поздороваться» таким образом, чтобы по щелчку на кнопке выводилось бы окно confirm() с сообщением "Еще раз здравствуйте!".

5. Напишите оператор JavaScript, который отображает приветствие новых посетителей Web-страницы на уровне заголовка <h1 > страницы документа.

6. Напишите в теле документа скрипт формирующий в документе тег параграфа <p>с помещенной в него фразой «Заходи, гостем будешь».

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Добро пожаловать на сайт!</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <script type="text/javascript">
      window['username'] = false;
      function action() {
        confirm('Приветствую Вас!');

      }
      function sayAgain() {
        alert('Здравствуйте еще раз !')
      }
      function writeHello() {
        var str = prompt('Введите свое приветственное сообщение', '');
        if (str) {
          var newEl = document.createElement('h1');
          newEl.innerHTML = str;
          document.getElementById('strings').appendChild(newEl);
        }
      }
      function createPTag() {
        var newEl = document.createElement('p');
          newEl.innerHTML = 'Приветствую Вас, многоуважаемый Гость!!!';
          document.getElementById('strings').appendChild(newEl);
      }
    </script>
  </head>
  <body onload="action()">
    <div id="strings"></div>
    <input type="button" onclick="sayAgain()" value="Поздороваться еще раз" />
    <input type="button" onclick="writeHello()" value="Написать свое приветствие" />
    <input type="button" onclick="createPTag()" value="Сформировать тег параграфа" />
  </body>
</html>
```

7. Создать страницу, которая предлагает (в диалоговом окне) пользователю ввести число, а затем выводит информацию о четности числа на страницу документа. Для выяснения четности воспользоваться

- оператором if … else;
- оператором ? :;
- оператором switch.

Функция parseInt поможет преобразовать строку к целому числу.

8. Создайте страницу, которая предлагает (в диалоговом окне) пользователю ввести число, а затем выводит кубы всех чисел от 1 до введенного числа при условии, что они не превосходят 1000. Воспользоваться

- циклом for;
- циклом while;
- циклом do while
- if()

9. Вывести в заголовок строки: "раз"; "два"; "три"; "четыре"; "пять"; "вышел зайчик погулять".

Пример создания бегущей строки в заголовке браузера.

```
<html>
<head>
<title>JavaScript в примерах</title>
<script language="JavaScript">
var msg = document.title;
var c = 0;
function animateTitle()
{
  document.title = msg.substring(0,c);
  if(c == msg.length)
     {
       c = 0;
       setTimeout("animateTitle()", 2000)
     }
  else
     {
       c++;
       setTimeout("animateTitle()", 200)
     }
}
animateTitle()
</script>
</head>
<body>
<h1>Пример с заголовком браузера</h1>
</body>
</html>
```

Такой же пример

```
<html>
  <head>
    <title>JavaScript в примерах</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<script type="text/javascript">
   var msg = document.title;
   var c = 0;
var strings = new Array("раз", "два", "три", "четыре", "пять", "вышел зайчик погулять");
   function animateTitle()
   {
      document.title = strings[c];
      if(c == strings.length - 1)
      {
         c = 0;
         setTimeout("animateTitle()", 2000)
      }
      else
      {
         c++;
         setTimeout("animateTitle()", 200)
      }
   }
</script>
</head>
<body onload="animateTitle()">
   <div id="strings"></div>
</body>
</html>
```

Полем status bar называют поле нижней части окна, в котором отображается информация о состоянии браузера (загрузка документа, загрузка графики, завершение загрузки и т.п.). Программа на JavaScript имеет возможность работать с этим полем с помощью двух свойств:

window.status — значение поля статуса;

window.defaultStatus — значение поля статуса по умолчанию.

Значение свойства status можно изменить — и оно тут же будет отображено в поле статуса. Свойство defaultStatus тоже можно менять — и сразу по его изменении оно отображается в поле статуса. Пример:

```
window.status = "put your message here"
```

a)Напишите оператор JavaScript, который отображает сообщение в строке состояния, приветствующее новых посетителей Web-страницы.

b)Создайте страницу на которой расположена кнопка «Установить текст в строке состояния» по нажатию на которую в строку состояния выводится приветствие.

c) Расположите на странице ссылку. При наведении мыши на ссылку отобразите в строке состояния текст 'Заходи, не пожалеешь!' (обработка события onMouseOver), при уходе курсора со ссылки отобразите в строке состояния текст 'Напрасно, не зашел' (обработка события onMouseOut).

d) Создать Web-страницу с «бегущим текстом» в строке статуса состояния браузера

e) Вывести текущее время в окно документа и в окно статуса

Пример программы выводящей время в строке статуса.

```
<HTML>
<HEAD>
<SCRIPT>
function time_scroll()
```

```
{
  var d = new Date();
  window.status = d.getHours()
        + ':' + d.getMinutes()
        + ':' + d.getSeconds();
  setTimeout('time_scroll()',1000);
}
</SCRIPT>
</HEAD>
<BODY onLoad="time_scroll()">
<H1>Часы в строке статуса</H1>
</BODY>
</HTML>
```

Метод setTimeout() используется для создания нового потока вычислений, исполнение которого откладывается на время (в миллисекундах), указанное вторым аргументом:

```
idt = setTimeout("JavaScript_код",Time);
```

Если в Mozilla не изменяется строка состояния, то следует выполнить: Инструменты -> Настройки -> Содержимое и там напротив JavaScript -> Дополнительно -> Изменять текст в строке статуса.

10. Вывести на странице регулярно обновляемую информацию:

a) о том, сколько дней осталось до нового года, в зависимости от текущей даты;

b) о том, сколько часов и минут осталось до конца рабочего дня, или до начала нового рабочего дня в зависимости от текущего времени. Рабочее время считать с 9.00 до 17.00. Расширенный вариант: рабочими днями считать понедельник, вторник, среду, четверг, пятницу, а в рабочее время есть обед с 12.00 до 12.30.

11. Непрерывно выводите заголовок первого уровня в стиле печатной машинки. После того, как строка напечатается полностью не стирайте ее побуквенно как в примере, повторяйте вывод сначала. Измените стиль заголовка: шрифт крупнее в 2 раза, красный цвет шрифта, курсив.

Пример вывода строки в стиле печатной машинки

```
<html>
<head>
<title>Пример с печатающими буквами</title>

<script type="text/javascript">

var delay = 50;  // задержка
var pos = 0;
//Строка, которая будет выводиться
var msg = 'Пример с печатающими буквами';
// функция, печатающая слова
function printWords(text, dir)
{
   var typingtext = text.substring(0, pos);
   document.getElementById("type").innerHTML = typingtext;
   pos += dir;  // печатаем или стираем текст
   if (pos > text.length)
      setTimeout('printWords('"+text+"','+(-dir)+')', delay * 50);
   else
```

```
        {
          if(pos < 0)
            {
              dir = -dir;
            }
           setTimeout('printWords('"+text+"','+dir+')', delay);
        }
    }
    </script>
    </head>
    <body onLoad="printWords(msg, 1)">
    <h1>Эффект печатной машинки</h1>
    <div id="type"></div>
    </body>
    </html>
```

12. Сценарий в разделе body вызывает функцию, которая возвращает текст, введенный в диалоговом окне prompt ().

```
<html> <head>
<script type=text/javascript">
function askName() {
var name = prompt ("What is your name, please?11,"")
return name
}
</script>
</head>
<body>
<script type="text/javascript">.
document.write("Welcome to my web page, " + askName() + ".")
</script>
</body>
</html>
```

По событию onclick вывести это сообщение в окно статуса состояния и в другие диалоговые окна.

Упражнения и задачи по JavaScript

1. Создать кнопку Hi, на которой отображается надпись “Привет”, а обработчик события onclick вызывает диалоговое окно предупреждения, содержащее строку Hello to you, too! (Привет и тебе!).

2. Создайте сценарий, в котором использовался бы оператор: document.write("Hello, world.");. Сценарий должен запускаться при загрузке HTML-документа. В сценарий необходимо добавить комментарий, поясняющий принцип выполнения операторов этого сценария и вывести краткое пояснение в диалоговое окно.

3. Загрузка и манипулирование с изображениями на JavaScript.

4. Создайте HTML-документ, который сразу после загрузки страницы будет отображать диалоговое окно с предупреждением. Другое диалоговое окно должно отображаться при щелчке пользователя на кнопке формы.

5. Напишите оператор JavaScript, который отображает сообщение в строке состояния, приветствующее новых посетителей Web-страницы.

6. Напишите оператор JavaScript, который отображает приветствие новых посетителей Web-страницые на уровне заголовка <hl > страницы.

7. Создайте страницу, которая предлагает (в диалоговом окне) пользователю ввести его или ее имя, а после этого будет приветствовать его (ее) по имени. Приветствие должно отображаться в основной части страницы.

8. Создайте страницу, содержащую произвольный текст. Сразу после загрузки она должна автоматически выводить диалоговое окно с адресом URL текущей страницы.

9. Создать страницу, использующую операторы:

window.status = "Welcome to my Web page.";

document.write("<hl>Welcome to my Web page.</hl>")

Наведение курсора на эти строки должно вызывать диалоговые окна, поясняющие работу этих операторов.

10. Манипулирование окнами: изменение размера, цвета и др.

11. Сценарий в разделе body вызывает функцию, которая возвращает текст, введенный в диалоговом окне prompt ().

```
<html> <head>
<script type=text/javascript">
function askName() {
var name = prompt ("What is your name, please?11,"")
return name
}
</script>
</head>
<body>
<script type="text/javascript">.
document.write("Welcome to my web page, " + askName() + ".")
</script>
</body>
</html>
```

По событию onclick вывести это сообщение в окно статуса состояния и в другие диалоговые окна.

12. Адрес URL получен из свойства href объекта location.

```
<html> <head>
<script type="text/javascript">
function showLocation() {
alert("This page is at: " + location.href);
}
</script> </head>
<body onload="showLocation()"> Bu, be, by.
 </body> </html>
```

Создать кнопку и по событию onclick вывести это сообщение в окно статуса состояния и в другие диалоговые окна.

13. Обработчики событий JavaScript.

14. Навигация по сайту

15. JavaScript и таблица стилей

16. Определение типа ОС клиента.

17. Определение типа браузера клиента.

18. Создание контекстного меню.

19. Раскрывающееся окно.

20. Таймер-часы.

21. Приветствие посетителя с учетом времени суток(утро, день, вечер).

22. Календарь.

23. Формы. Проверка информации, вводимой и посредством форм. Проверка наличия во вводимой строке определенных символов.

24. Рисунки на веб-сайте.

25. Слои на веб-сайте.

26. Вычисления на веб-сайте.

27. Создать игру типа: “Угадай задуманное число”.

28. Создать различные меню на JavaScript.

29. Создать объект калькулятор.

30. Создать объект бегущая строка.

31. Перекодировщик русских символов

32. Автоматическая регистрация страниц на поисковых системах. Программа выполняется на стороне клиента.

33. Время в JavaScript (получение и установка даты-времени).

34. Создать Web-страницу с «бегущим текстом» в строке состояния браузера

35. Задание задержек по времени при выполнении функций. Программирование картинок.

36. Управление слоями.

37. Создать Web-страницу со слоем, появляющимся при нажатии на кнопку «Невидимка», и исчезающем при повторном нажатии на кнопку «Невидимка».

38. Программирование форм. Создать Web-страницу с викториной из 7-ми вопросов с 5-ю возможными вариантами ответов.

39. Создание тестирующей программы.

40. Голосование

41. Создание фреймов.

42. Органайзер

43. Авторизация на JavaScript

44. Решение квадратных уравнений -------x2 +------x +-------- =0

45. Проверка доступности cookie.

46. Объекты JavaScript.

47. DOM.

48. DOM и AJAX.

49. Работа с текстовыми строками из формы: объединение, сравнение, нахождение длины, удаление HTML-тегов из строки, удаление пробелов вначале и в конце.

50. Перевод строки в разные кодировки: KOI8-R, WINDOWS-1251, UNICODE.

51. Конвертация даты из одного формата в другой.

52. Проверка данных, вводимых пользователем в форму:

a) Проверка обязательных для ввода полей

b) Проверка допустимости вводимых данных

c) Удаление HTML тегов

d) Удаление обратных слешей

52. Создать прототип для списка личностей(актеров, спортсменов, ученых, политиков, писателей). Личность:

```
{// свойства:
name       //(имя);
birthYear //(год рождения);
country   //(страна проживания, по умолчанию  USA);
alive      //(жив  ли, по умолчанию – true).
}
```

Добавить методы:

setCountry() – присваивает название страны соответствующему свойству и проверяет является ли название страны корректным setBirthYear –присвавает свойству значение года рождения и проверяет являет ли значение корректным.

Список личностей реализуется с помощью объекта “Spisok”:

{ supermans // массив личностей;
length // количество членов в списке.
}

Добавить к списку следующие методы:

Add () (добавить в список), Remove() (удалить из списка), Replace() (заменить в списке). Заполнить список 5-10 элементами.

Написать функцию, которая будет при загрузке страницы создавать таблицу со списком актеров (для создания таблицы используйте функции insertRow и insertCell, описание которых можно посмотреть в MSDN). Таблица должна будет содержать следующие колонки: порядковый номер, имя, год рождения, страна, жив или нет.

Имя в таблице должно быть ссылкой, при нажатии на которую данные актера должны загружаться в форму (которая должна быть на странице), форма должна иметь два кнопки: Сохранить и Отменить, соответственно должны быть написаны функции для очистки формы и для сохранения данных (после сохранения данные в таблице должны обновиться).

Лабораторная работа 6 по JavaScript (4 часа).

Проверка правильности заполнения формы на сайте

Проверка данных, вводимых пользователем в форму:

a) Проверка обязательных для ввода полей
b) Проверка допустимости вводимых данных
c) Удаление HTML тегов
d) Удаление обратных слешей

1. Добавьте на свой сайт, созданный в предыдущих лабораторных работах, страничку для регистрации пользователей. Для этого сформируйте форму регистрации пользователя со следующей информацией:
   - логин;
   - пароль и подтверждение пароля;
   - e-mail;

   Используя JavaScript и регулярные выражения выполните валидацию формы согласно следующих требований.
   - Все поля должны быть не пустыми.
   - Проверьте пароль на совпадение с подтверждением пароля.

- Используя регулярные выражения проверьте на правильность заполнения поле e-mail. Поле E-mail должно соответствовать всем требованиям для адреса электронной почты. Имя пользователя и имя почтового сервера разделяются знаком @, и могут содержать только латинские буквы, или цифры, или дефис, или точку. Имя домена верхнего уровня может содержать только латинские буквы.

2. Развернутые результаты валидации (какие поля не заполнены или неправильно заполнены) вывести в:
   - окно предупреждения.
   - на страницу ниже формы;
   - на страницу правее формы. Напротив ошибочно заполненного поля.

3. Развернутый вариант задания. Добавьте на сайт, страничку для регистрации пользователей. Для этого сформируйте форму регистрации пользователя со следующей информацией:

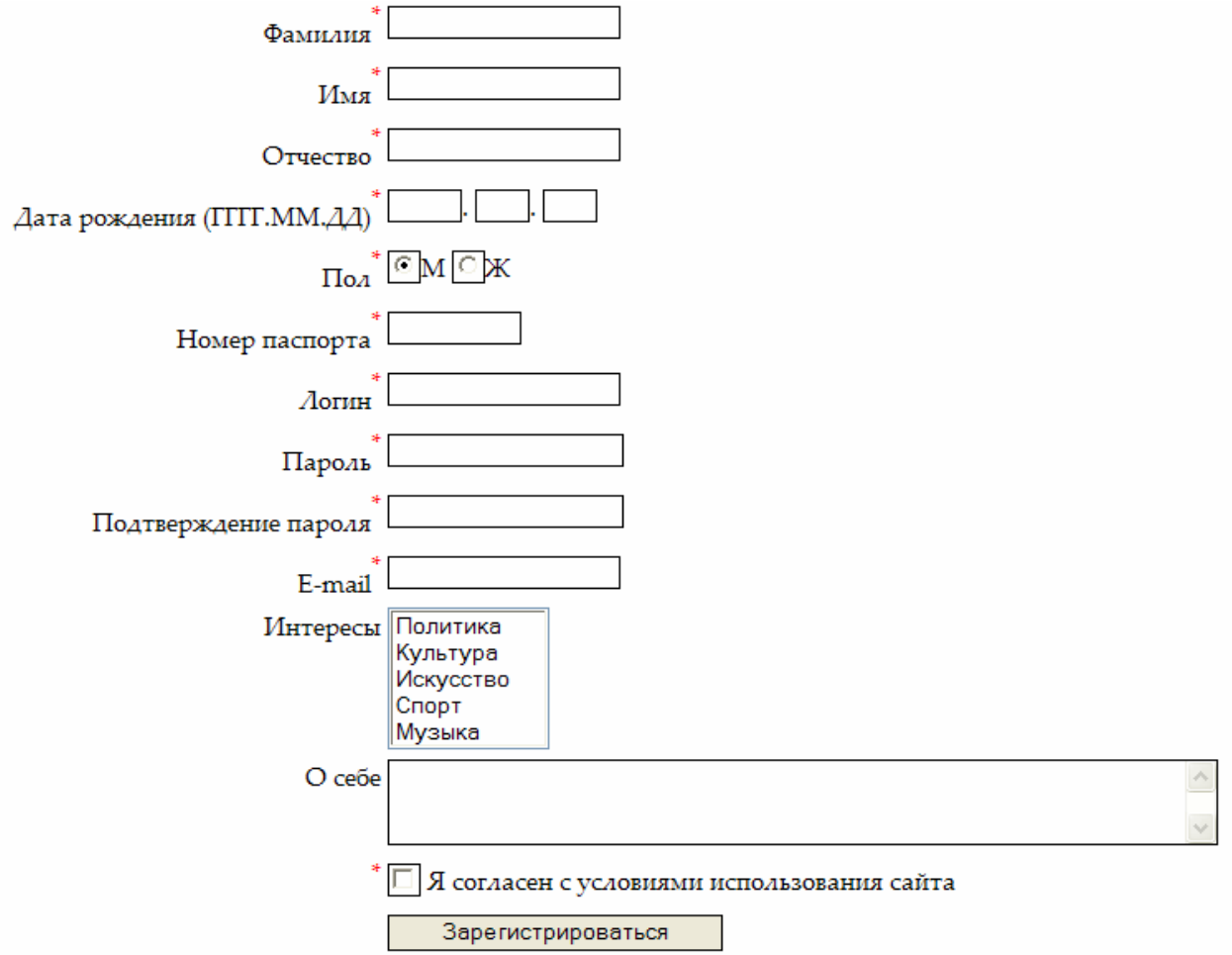
Фамилия,
имя,
отчество;
дата рождения;
пол;
номер паспорта;
логин;
пароль и подтверждение пароля;
e-mail;
список интересов (политика, культура, искусство, спорт, музыка) (список в будущем может быть расширен);
признак согласия с условиями использования сайта;
«о себе».

При этом поля

Фамилия,
имя,
отчество;
дата рождения;
пол;
номер паспорта;
логин;
пароль и подтверждение пароля;
e-mail;
признак согласия с условиями использования сайта

должны быть обязательными для заполнения.

Используйте CSS-стили для оформления формы. Для удобства используйте таблицу для размещения элементов формы и подписей к ним. До подключения стилевых классов разработанных вами в лабораторной работе № 4 форма должна иметь следующий вид.



При необходимости дополните стилевые таблицы новыми классами таким образом, чтобы разработанная форма соответствовала стилю вашего сайта.

4. Используя JavaScript и регулярные выражения выполните валидацию формы согласно следующих требований.

4.1. Все поля, помеченные как обязательные для заполнения, должны быть не пустыми.

4.2. Поля Фамилия, Имя, Отчество могут содержать только символы русского и английского алфавита, точку, дефис и пробел.

4.3. Поле Год может содержать только четыре цифры, и отображать года с 1900 по 2007.

4.4. Поле месяц может содержать только значения месяца в формате ММ: 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11 или 12.

4.5. Поле день может содержать только значения дня в формате ДД. Провести валидацию на недопустимость значений типа 30 или 31 число в феврале, 31 число в апреле, июне, сентябре, ноябре, 29 число в феврале невисокосного года.

4.6. Поле Номер паспорта содержит девять символов из которых два первых – латинские буквы, а семь последующих – цифры.

4.7. Поле Логин может содержать от 6 до 20 символов и состоять только из латинских букв, точки, подчеркивания и дефиса.

4.8. Поле Пароль может содержать от 6 до 20 символов и состоять только из латинских букв, точки, подчеркивания и дефиса и не совпадать с логином.

4.9. Поле Подтверждения пароля должно совпадать с полем Пароль.

4.10. Поле E-mail должно соответствовать всем требованиям для адреса электронной почты. Имя пользователя и имя почтового сервера разделяются знаком

@, и могут содержать только латинские буквы, или цифры, или дефис, или точку. Имя домена верхнего уровня может содержать только латинские буквы.

5. Вывести информацию о заполнении полей для пользователя в соответствии с пунктом задания, указанном преподавателем.

5.1. Развернутые результаты валидации (какие поля не заполнены или неправильно заполнены) вывести в окно предупреждения.

5.2. Развернутые результаты валидации (какие поля не заполнены или неправильно заполнены) вывести на страницу. Например, добавить в таблицу еще одну строку, объединить в ней все ячейки и поместить информацию в нее, или поместить информацию в контейнер вне формы.

5.3. В таблице, в которой вы разместили элементы формы, предусмотрите еще один столбец. В этот столбец поместите все необходимые пояснения для заполнения полей. Например, какие символы могут входить в логин или пароль, каким должен быть формат даты рождения и номера паспорта и т. д. Развернутые результаты валидации (какие поля не заполнены или неправильно заполнены) вывести в этот столбец, ниже пояснений к заполнению. Информация о незаполненном, или неверно заполненном, элементе должна выделяться другим цветом, например красным.

В сценарии JavaScript могут использоваться объекты нескольких видов:

клиентские объекты, входящие в модель BOM: window, location, navigator и т.п.

клиентские объекты, входящие в модель DOM:document

встроенные объекты Array, String, Date, Number, Function, Boolean, Math.

пользовательские объекты

Оператор for(переменная in объект) позволяет "пробежаться" по свойствам объекта. Пример.

```
for(v in document)
 document.write("document."+v+" = <B>"+ document[v]+"</B><BR>");
```

Всю необходимую информацию о запущенном браузере и системе пользователя можно узнать при помощи объекта navigator.

Получите информацию о браузере и системе: имя браузера, версия браузера, кодовое название браузера, платформа на которой работает браузер, доступность на запуск сценариев JavaScript.

b)Укажите корректный JavaScript синтакс для открытия нового окна "window2" ?

window.open("http://www.w3schools.com","window2")
open.new("http://www.w3schools.com","window2")
new.window("http://www.w3schools.com","window2")
new("http://www.w3schools.com","window2")

6. Задание задержек по времени при выполнении функций. Программирование картинок. Загрузка и манипулирование с изображениями на JavaScript.

Следует обратить внимание на использование метода setTimeout(). JavaScript разрабатывался для многопоточных операционных систем, поэтому правильнее будет представлять себе исполнение скриптов следующим образом:

a)скрипт movie() получает управление от обработчика события onLoad;

b)заменяет картинку;

c)порождает новый скрипт movie() и откладывает его исполнение на 500 миллисекунд;

d)текущий скрипт movie() уничтожается JavaScript-интерпретатором.

После окончания срока задержки исполнения все повторяется.

```
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<title>Документ без названия</title>
<SCRIPT>
var i=0;
n=5;
function movie()
{
 document.i.src='n'+i+'.gif';
 i++; if(i>10){i=0;n--;}
 if(n>0)setTimeout('movie();',500);
}
</SCRIPT>
</head>
<body onLoad="movie();"><IMG NAME=i>
</body>
</html>
```

а)Реализовать запуск и остановить мультипликацию по требованию пользователя по нажатию на кнопке.

8. Создайте страницу, содержащую произвольный текст. Сразу после загрузки она должна автоматически выводить диалоговое окно с адресом URL текущей страницы.

9. Создать страницу, использующую операторы:

```
window.status = "Welcome to my Web page.";
document.write("<hl>Welcome to my Web page.</hl>")
```

Наведение курсора на эти строки должно вызывать диалоговые окна, поясняющие работу этих операторов.

10. Манипулирование окнами: изменение размера, цвета и др.
11. Адрес URL получен из свойства href объекта location.

```
<html> <head>
<script type="text/javascript">
function showLocation() {
alert("This page is at: " + location.href);
}
</script> </head>
<body onload="showLocation()"> Bu, be, by.
 </body> </html>
```

Создать кнопку и по событию onclick вывести это сообщение в окно статуса состояния и в другие диалоговые окна.

7. Определите тип браузера на стороне клиента и выведите всю доступную информацию о нем.
8. Поместить на страницу кнопки: "открыть окно" и "закрыть окно". При нажатии кнопки "открыть окно" открывается новое окно, в котором выводятся часы (см. лаб. раб. №4). При нажатии кнопки "закрыть окно", созданное окно закрывается.
9. Пусть имеется ряд изображений. Создать эффект мультипликации на странице, для этого разработать сценарий смены изображений с заданной частотой. 11. Реализовать запуск и остановить мультипликацию по требованию пользователя по нажатию на кнопке. В качестве примера Animation.html.

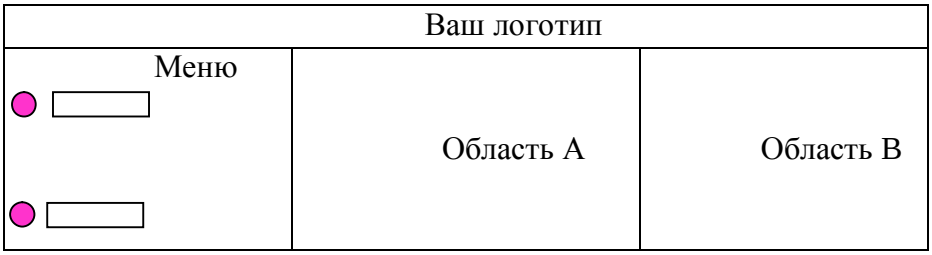
10. На странице задан элемент, содержащий некоторый текст. Разработать сценарий, который при наведении мыши на элемент меняет текст на другой.
11. Задана таблица. Разработать сценарий, который при наведении мыши на ячейку меняет толщину и цвет границы.
12. На странице задан заголовок (1 строка) таблицы

Автор	Название книги

Создать 2 поля для ввода данной информации и кнопку. Разработать сценарий: при нажатии кнопки информация из полей ввода динамически помещается в конец таблицы.
Примечание: Работа с таблицами поясняется в документе Таблица как объект HTML.doc.

13. Организуйте страничку с фотографиями на своем персональном сайте в виде фотографии и двух указателей под ней: "предыдущая" и "следующая". Пример смотри в папке DHTML.

Задание 4. продолжать работу на странице из задания 1).



В области Меню создать несколько ссылок, при этом при прохождении мыши изменяется фон ссылки и увеличивается размер шрифта текста ссылки, при уходе – восстанавливается исходное состояние. При нажатии клавиши мыши - цвет фона изменяется и остается постоянным, кроме того, для одной из ссылок запросить подтверждение перехода по ссылке.

В качестве маркеров в меню использовать небольшие изображения (например, ball.gif в текущей папке).

Задание 5. Пусть имеется K изображений. Разработать сценарий смены изображений с заданной частотой. В качестве примера Animation.html.

Задание 6. Создать сценарий отправки электронной почты.

Задание 7. На странице задан элемент, содержащий некоторый текст. Разработать сценарий, который при наведении мыши на элемент меняет текст на другой.

Задание 8. Задана таблица. Разработать сценарий, который при наведении мыши на ячейку меняет толщину и цвет границы.

Задание 9. На странице задан заголовок (1 строка) таблицы

Автор	Название книги

Создать 2 поля для ввода данной информации и кнопку. Разработать сценарий: при нажатии кнопки информация из полей ввода динамически помещается в конец таблицы.

Примечание: Работа с таблицами поясняется в документе Таблица как объект HTML.doc

1. Ознакомтесь с работой Ajax. Пользуясь Adobe Dreamweaver поместите на сайт не менее одного элемента Spry. Например, панель меню Spry, панели со вкладками Spry, набор вкладок Spry, сворачивающаяся панель Spry, подсказка Spry или др. Внимательно изучите .js и .css файлы для вставленных элементов.

10. Определение типа ОС и типа браузера клиента.
11. Создание контекстного меню.
12. Приветствие посетителя с учетом времени суток (утро, день, вечер).
13. Календарь.
14. Органайзер
15. Слои на веб-сайте.
16. Создать игру типа: “Угадай задуманное число”.
17. Создать различные меню на JavaScript.
18. Создать объект калькулятор.
19. Перекодировщик русских символов
20. Программирование форм. Создать Web-страницу с викториной из 7-ми вопросов с 5-ю возможными вариантами ответов.
21. Голосование
22. Решение квадратных уравнений -------x2 +------x +-------- =0
23. Авторизация на JavaScript
24. Навигация по сайту
25. Работа с текстовыми строками из формы: объединение, сравнение, нахождение длины, удаление HTML-тегов из строки, удаление пробелов вначале и в конце.
26. Перевод строки в разные кодировки: KOI8-R, WINDOWS-1251, UNICODE.
27. Конвертация даты из одного формата в другой.

Лабораторная работа 7. PHP

Задание 1. Массивы и строки

Массив может инициализироваться одним из двух способов: последовательным присвоением значений, или посредством конструкции array().

Для ассоциированных массивов такой вызов будет иметь вид:

$new_massiv = array('name' => 'nobody', 'email' => 'mail@bsu.by');

Выполнить одно из перечисленных ниже упражнений

1.1. В массиве строк проверить начинается ли каждая строка символом “*”. Строки без “* “ перенести в другой массив.

1.2. В массиве из n строк проверить, содержит ли к-я строка символ @. Если не содержит, вставить символ в строку.

1.3. В массиве строк удалить все HTML –теги, заключенные в скобки <>

1.4. В массив случайным образом помещаются строки задающие «совет дня» или «цитата дня». Случайным образом выбрать строку из массива можно использовать функции Shuffle(array arr); или arrayrand(array arr, int num);

1.5. Создать многомерный массив: Факультет, Курс, Группа, Студенты. Вывести список студентов в алфавитном порядке.

1.6. Создать многомерный массив: Факультет, Кафедра, Преподаватели. Вывести список преподавателей в алфавитном порядке.

Задание 2. Функции

Функция может быть определена с использованием синтаксиса:

```
function fname ($arg_1, $arg_2, ..., $arg_n)
{
 echo "Пример \n";
 return $val; //возвращаемое значение
}
```

Необязательный оператор return возвращает значение любого типа, в том числе список и объект.

Выполнить одно из перечисленных ниже упражнений

2.1. Среди n чисел найти наибольшее и наименьшее простые числа.

2.2. Для заданного числа n построить треугольник Паскаля.

2.3. Написать функцию, возвращающую текст приветствия в соответствии с приведенной ниже схемой

```
<?php
if((date(“G”) >=5)AND(date(“G”) <= 11 ))echo “Good Morning!”;
if((date(“G”)>=12)ANDdate(“G”)<=18))echo “Good Afternoon!”;
if((date(“G”) >= 19)AND(date(“G”)<= 4))echo “Good Evening!”;
?>
```

Задание 3. Файлы и строки

Файлы могут быть текстовыми, содержащими строки переменной длины и бинарными, представляющими последовательность байт.

Выполнить одно из перечисленных ниже упражнений

В следующих заданиях использовать файлы, строки и объекты.

1. Текст записан одной длинной строкой. Признаком красной строки служит символ $. Переформатировать текст в 60-символьные строки, формируя абзацы.

2. Текст, сформированный построчно, выровнять по правому краю так, чтобы каждая строка заканчивалась знаком препинания или одним пробелом. Выравнивание осуществить, вставляя дополнительные пробелы между словами (равномерно по всей строке).

3. Данный текст программы на каком-либо алгоритмическом языке и словарь зарезервированных слов этого языка (в английской транскрипции). Преобразовать текст, записав все зарезервированные слова прописными буквами, а остальные конструкции (имена и так далее) – строчными. Русские буквы (имена, литералы) не заменять.

4. Программа, записанная 80-байтовыми строками, в последних 8 байтах содержит номер строки. Строки упорядочены по номерам, но не обязательно с шагом 1. Поступает изменение к программе в таком виде. Вставить новые строки взамен имеющихся старых или между ними.

5. Текст записан 60-символьными строками, содержит следующие знаки корректуры: $ – сделать красную строку; # удалить следующее слово; @ – удалить следующую фразу. Произвести указанную корректировку, переформатируя строки в пределах абзаца.

6. Часто встречающаяся ошибка начинающих наборщиков – дважды записанное слово. Обнаружить и исправить такие ошибки в тексте, записанном 80-символьными строками; переформатировать строки в пределах абзаца.

7. Стихотворный текст (в строке не более 80 символов) имеет четырехстрочную строфу. Записать его «лесенкой» (по одному слову в строке), вставляя пустую строку после каждого четверостишья.

8. Поздравления. По заданному списку фамилий напечатать каждому упомянутому в списке поздравление к определенному празднику. Чтобы избежать

шаблона, перечень желаемых благ выбирать как случайное подмножество из заготовленного списка (например, здоровья, счастья, продвижения по службе, долголетия и т. д.). Можно сделать переменными и название праздника – для универсальности программы.

9. Имеется список членов коллектива с указанием принадлежности каждого к различным общественным организациям (профком, ученый совет, общество книголюбов, федерация пентикса и т. д.). Напечатать приглашение всем членам на очередное заседание указанной организации. Задается только вид организации, место и время сбора.

12. В заданном тексте найти самое длинное слово и самую длинную фразу.

13. Обнаружено, что в тексте пропущены некоторые слова и словосочетания. Эти слова и словосочетания представлены отдельным списком в том порядке, в каком должны быть вставлены. Места вставки отмечены в тексте символом $. Откорректировать текст.

14. Текст не содержит собственных имен и сокращений, набран с использованием прописных и строчных русских букв. Проверить то, что все фразы (и только они) начинаются с прописной буквы; при необходимости откорректировать текст.

15. Текст, не содержащий собственных имен и сокращений, набран полностью прописными русскими буквами. Заменить все прописные буквы, кроме букв, стоящих после точки, строчными буквами.

16. IP-адрес посетителя страницы находится как элемент суперглобального массива S – Server[´Remoute_ADDR´]. Составить список посетителей за текущий день (неделю, месяц).

17. Запрет посещений с определенных IP-адресов. В массиве хранятся IP-адреса, выход с которых на наш сайт нежелателен. Сравнить IP-адрес посетителя с адресами из списка и не пускать нежелательного посетителя.

18. Составить список серверов ($_SERVER[´SERVER_NAME´]) и список страниц ($_SERVER[´PHP_SELF´]), которые мы посетили.

19. Составить список страниц, с которых приходят к нам посетители ($_SERVER[´HTTP_REFERER´]).

20. Составить список браузеров, операционных систем, поисковых роботов, которые посещают нашу страницу: ($_SERVER[´HTTP_USER_AGENT´]), таким же образом определяются менеджеры закачки типа ´DownLoadMaster´, ´FlashGet´ и другие.

21. Выяснить языковые предпочтения посетителя, который браузер отправляет серверу в HTTP-заголовке и помещает в массив $_SERVER[´HTTP_ACCEPT_LANGUAGE´].

Задание 4. Работа с базами данных

Выполнить одно из перечисленных ниже упражнений

Создать распределенную информационную систему. Из пользователей системы обязательно наличие *Администратора* и Зарегистрированного *Пользователя.*

В каждом из заданий необходимо выполнить следующие действия:

- Организацию соединения (пула соединений) с базой данных вынести в отдельный класс, метод которого возвращает соединение;
- Спроектировать БД. Привести таблицы к одной из нормированных форм. Создать БД.
- Создать класс для выполнения запросов на извлечение информации из БД с использованием компилированных запросов;
- Создать класс на добавление информации;
- Создать документ (XHTML, JSP и т.д.) с полями для формирования запроса;

- Результаты выполнения запроса передать клиенту.

1. **Видеотека.** В БД хранится информация о домашней видеотеке – фильмы, актеры, режиссеры.

Для фильмов необходимо хранить:

- название;
- актеров;
- дата выхода;
- страну, в которой выпущен фильм.

Для актеров и режиссеров необходимо хранить:

- ФИО;
- дата рождения.

- Найти все фильмы, вышедшие на экран в текущем и прошлом году.
- Вывести информацию об актерах, снимавшихся в заданном фильме.
- Вывести информацию об актерах, снимавшихся как минимум в 2-х фильмах.
- Вывести информацию об актерах, которые были режиссерами хотя бы одного из фильмов.
- Удалить все фильмы, дата выхода которых была более 2-х лет назад.

2. **Расписание занятий.** В БД хранится информация о преподавателях и проводимых ими занятиях.

Для предметов необходимо хранить:

- название;
- время проведения (день недели);
- аудитории, в которых проводятся занятия.

Для преподавателей необходимо хранить:

- ФИО;
- -предметы, которые он ведет;
- количество пар в неделю по каждому предмету;
- количество студентов занимающихся на каждой паре.

- Вывести информацию о преподавателях, работающих в заданный день недели в заданной аудитории.
- Вывести информацию о преподавателях, которые не ведут занятия в заданный день недели.
- Вывести дни недели, в которых проводится наименьшее количество занятий.
- Вывести дни недели, в которых занято наименьшее количество аудиторий.
- Перенести первые занятия заданных дней недели на последнее место.

3. **Письма.** В БД хранится информация о письмах и отправляющих их людях.

Для людей необходимо хранить:

- ФИО;
- дата рождения.

Для писем необходимо хранить:

- отправителя;
- получателя;
- тема письма;
- текст письма;
- дата отправки.

- Найти пользователя, длина писем которого наименьшая.
- Вывести информацию о пользователях, а также количестве полученных и отправленных ими письмах.

- Вывести информацию о пользователях, которые получили хотя бы одно сообщение с заданной темой.
- Вывести информацию о пользователях, которые не получали сообщение с заданной темой.
- Направить письмо заданного человека с заданной темой всем людям.

4. **Сувениры.** В БД хранится информация о сувенирах и их производителях.
   Для сувениров необходимо хранить:
   - название;
   - производителя;
   - дату выпуска;
   - цену.

   Для производителей необходимо хранить:
   - название;
   - страну.

- Вывести информацию о сувенирах заданного производителя.
- Вывести информацию о сувенирах, произведенных в заданной стране.
- Вывести информацию о производителях, чьи цены на сувениры меньше 1000.
- Вывести информацию о производителях заданного сувенира, произведенных в прошлом году.
- Удалить заданного производителя и его сувениры.

5. **Заказ.** В БД хранится информация о заказах магазина и товарах в них.
   Для заказа необходимо хранить:
   - номер заказа;
   - товары в заказе;
   - дату поступления.

   Для товаров в заказе необходимо хранить:
   - товар;
   - количество.

   Для товара необходимо хранить:
   - название;
   - описание;
   - цену.

- Вывести полную информацию о заданном заказе.
- Вывести номера заказов, сумма которых не превосходит 100 и количество различных товаров равно 1.
- Вывести номера заказов, содержащие товар с заданным заказом.
- Вывести номера заказов, не содержащие товар с заданным названием и поступившие в течение текущего дня.
- Сформировать новый заказ, состоящий из товаров, заказанных в текущий день.
- Удалить все заказы, в которых присутствует заданное количество заданного товара.

6. **Погода.** В БД хранится информация о погоде в различных регионах.
   Для погоды необходимо хранить:
   - регион;
   - дату;
   - температуру;
   - осадки.

   Для регионов необходимо хранить:
   - название;
   - площадь;

- тип жителей.

Для типов жителей необходимо хранить:

- название;
- язык общения.

- Вывести сведения о погоде в заданном регионе.
- Вывести даты, когда в заданном регионе шел снег, и температура была ниже -10.
- Вывести информацию о погоде за прошедшую неделю в регионах, жители которых общаются на заданном языке.
- Вывести среднюю температуру за прошедшую неделю в регионах с площадью более 1000.

7. **Города.** В БД хранится информация о городах и их жителях.

Для городов необходимо хранить:

- название;
- год создания;
- площадь;
- количество населения для каждого типа жителей.

Для типов жителей необходимо хранить:

- город проживания;
- название;
- язык общения.

- Вывести информацию обо всех жителях заданного города, разговаривающих на заданном языке.
- Вывести информацию обо всех городах, в которых проживают жидели выбранного типа.
- Вывести информацию о городе с максимальным количеством населения и всех типах жителей в нем проживающих.
- Вывести информацию самом древнем типе жителей.

8. **Словарь.** В БД хранится англо-русский словарь, в котором для одного английского слова может быть указано несколько его значений и наоборот. Со стороны клиента вводятся последовательно английские (русские) слова. Для каждого из них вывести на консоль все русские (английские) значения слова.

9. **Словари.** В двух различных базах данных хранятся два словаря: русско-белорусский и белорусско-русский. Клиент вводит слово и выбирает язык. Вывести перевод этого слова.

10. **Стихотворения.** В БД хранятся несколько стихотворений с указанием автора и года создания. Для хранения стихотворений использовать объекты типа Blob. Клиент выбирает автора и критерий поиска.

- в каком из стихотворений больше всего восклицательных предложений?
- в каком из стихотворений меньше всего повествовательных предложений?
- есть ли среди стихотворений сонеты и сколько их?

Задание 5. Создание WEB-объектов

Клиентские методы HTTP

В HTTP метод клиента определяет запрос, отправленный от Web-клиента, либо PHP-сценария, HTTP-серверу. Существуют три основных типа запросов:

GET-запросы. Когда вы хотите только получить информацию от источника HTTP, то можете сделать это методом GET. Получить информацию можно из файла, либо от исполняемой программы на Web-сервере с указанным URL-адресом. Привлекательность

HTTP состоит в том, что запрос GET делает выполнение программы таким же простым, как извлечение файла.

POST-запросы. Когда вы хотите отправить информацию от клиента Web-серверу, то используете запрос POST. Обычно это имеет место, когда вы отправляете содержимое Web-формы Web-серверу.

HEAD-запросы. Когда вы хотите получить информацию о запрошенном URL, но не информацию самого URL, то используете запрос HEAD.

По методу GET контент запроса посылается через поля формы или через строку URL, по методу POST – через поля формы. Запросы можно создать и отправить вручную с помощью средств JavaScript, а получить обратно с помощью методов PHP.

Список заданий

1. Загрузка файла на сервер. Просмотр каталога из клиентской формы. Выбор файла и загрузка на сервер

2. Система отправки писем Send Mail. Отправка e-mail. E-mail с вложенными файлами. Интерфейс:

- Имя.
- e-mail.
- Отправить.
- Обратная связь или «контакты».

3. Рекомендация другу. На указанный e-mail оправляется письмо со ссылкой и рекомендациями, подписанные вами.

4. Система рассылки SMS - сообщений. Поддержка мобильной связи.

5. Система рассылки ICQ - сообщений.

6. Получение информации о клиенте и сервере.

7. Регистрация на сайте. Заполнить регистрационную форму: Фамилия, Имя, Город, E-Mail, login.

Система должна сгенерировать уникальный password. После регистрации отослать password по E-Mail.

8. Авторизация посетителя на сайте (логин, пароль, смена пароля и т.д.).

9. Счетчик посещений Counter – текстовый и графический.

10. HTTP Аутентификация и приветствие зарегистрированного клиента.

Интерфейс: Не видели Вас две недели Сэр. Рады видеть Вас снова на нашем сайте.

11. Удаленный web-counter. Необходимо реализовать службу наподобие RAMBLER:TOP100. Интерфейс: пользователь регистрируется и получает идентификатор. Таким образом он связывает свою страницу с сервером- носителем web-couter-a. После этого служба должна вести учет посетителей страницы пользователя.

Необходимо осуществить администраторские функции по включению и исключению произвольного пользователя администратором.

12. Откуда заходят посетители или поисковые машины. Адрес, IP- адрес, государство.

13. Подсчет on-line посетителей на сайте.

14. Администрируемая Гостевая книга.

- Имя:
- e-mail:
- Сайт:
- Город:
- Сообщение:
- Отправить

Интерфейс: пользователь заходит на страницу гостевой книги, на которой он сможет ввести сообщение, а также просмотреть сообщения оставленные другими пользователями. Создать две кнопки “Ввести” и “Просмотреть”.

Должны быть выполнены такие администраторские функции как включение и исключение сообщения в гостевой книге, запрет и управление запретом на включение сообщения содержащее определяемые администатором выражения, запрет и управление запретом на включение сообщения с заданных администатором хостов.

Помимо указанных пунктов можно предусмотреть:

1. Ответ администратора
2. Шаблон вывода.
3. Фильтр слов.
4. Смайлы

15. Голосование.

Интерфейс:

Как жизнь?

.Отлично

.Хорошо

.Средне

.Плохо

.Хуже плохого

Проголосовать

Для реализации голосования продумать детали:

1) Структура базы или файлов.

2) Логику скрипта (вывод результатов, само голосование, администрирование, архив).

3) Интерфейс пользователя.

4) Анти-флуд (для защиты от накрутки).

5) Администрирование - создается как отдельный скрипт, который защищается паролем и содержит формы управления голосованием. Голосование делается на определенный срок, потом оно неактивно. Оно может принудительно закрываться админом.

16. Система оценки материалов, расположенных на сайте.

17. Тесты и викторины.

18. FTP – клиент на PHP. FTP. Список файлов каталога. Загрузка файла с FTP-сервера. Загрузка файла на FTP-сервер.

19. Система отслеживания адресов по которым скачиваются копии текста из данного сайта.

20. Файловый менеджер

– удалять файлы и папки с сайта

– копировать

– переименовать

– создавать новые папки

– загрузка файлов на сайт

21. Электронная почта и список рассылки.

Список рассылки представляет собой систему регистрации удаленных пользователей с целью получения этими пользователями электронных писем с информацией. Интерфейс: произвольный пользователь заходит на страницу списка рассылки, регистрирует там свой E-MAIL, администратор заходит на страницу администрирования, там он набирает письмо, которое затем отправляется зарегистрированным пользователям. Необходимо также реализовать простейшие администраторские функции: исключение пользователя из списка рассылки, включение в список рассылки (подписка и отписка), сохранение отправляемого сообщения(с возможным архивированием), операции связанные с авторизацией (смена пароля и т.д.).

22. Доска объявлений.

23. Знакомства. Служба знакомств.

24. Чаты и общения.

25. Forum & FAQ.

Струтура базы для форума.

Таблица пользователи:

|ID_user|User_pass|User_Name

Таблица форумов:

|ID_forum|Forum_name

Таблица тем

|ID_theme|ID_foruma|ID_Autor|theme_name|theme_text

Посты:

|ID_post|Post_data|Post_autor|Post_text|

26. Навигация на PHP: Различные меню, карта сайта.

27. Cookie. Установка cookie. Чтение cookie. Срок действия cookie. Удаление cookie.

28. Сеансы. Сохранение данных сеансов. Создание счетчика посещений.

Сеансы без cookie. Удаление данных сеансов.

29. Система аутентификации пользователей с помощью сеансов и cookie.

30. Перевод с русского на транслит и обратно «text на translite».

31. Количество кликов по ссылкам на другие сайты.

32. Калькулятор.

33. Фотогалерея.

34. Календари и органайзеры.

35. Словарь – переводчик.

36. Система поиска для web-сервера.

В силу того что поисковые системы общего назначения(такие как Altavista) не позволяют выделить ресурсы отдельного сервера и производят модификацию своих баз данных медленно, возникает необходимость создать свою службу поиска. Поиск производится только для документов хранящихся на сервере-носителе службы. Должен поддерживаться поиск по сложным выражениям, включающим логические операторы и *.

37. Рубрикатор: Система поиска по первой букве (A,B,C…).

38. Использование региональных настроек.

39. Элементы управления и расширенные элементы управления.

40. Баннеры.

41. Быстрый переход на избранные сайты, спроектированный в виде выпадающего меню.

42. Электронные платежные системы

43. Интернет – радио и телевидение

44. Поисковые системы

45. FTP- серверы

46. Служба IRC

47. Написать сервис-редактор изображений. Возможности: конвертирование в различные форматы, изменять размер изображения и сохранять его, реализовать различные фильтры и т. д.

48. Написать скрипт чата. Особенности:

- Страница входа
- Страница с личными настройками
- Общие комнаты
- Комнаты с приватными сообщениями
- Возможность добавить картинку-аватар.

49. Скрипт проверки наличия новых личных сообщений на каком-либо популярном форуме (сайте). Особенности:

- Получение html-страницы при помощи CURL (вход в авторизированный раздел отправкой post-запроса на страницу логина)
- Проверка наличия новых сообщений на странице (NegExps)
- Отправка уведомлений на почту

50. Создать серверную спамилку (спамит по форумам PHPbb и гостевым книгам).

51. Создать синонимайзер (заменить синонимами данный текст).

52. Создать сервис по определению Tug и PageRank сайта.

53. Создать сервис по определению доменов, которые заканчиваются или брошенные. (Работа с сервисом Whois) зона .com; .info.

54. Создать Http-туннель. Создание цепи анонимных http, socks proxy.

55. Нахождение мультов (лиц, зарегистрировавшихся с одного IP несколько раз) и удаление несколько последних, оставляя первого.

56. Использование cookie & session: Пользователю, посетившему раздел музыка, предлагается банер с рекламой магазина музыкальных инструментов.

57. Написать игру типа «Кто хочет стать миллионером?»

58. Написать игру «Пятнашки».

59. Написать игру «Крестики-Нолики».

60. Игра «Морской бой» против компьютера.

61. Написать парсер (Вырезание статей со всех сайтов).

Приложение 1. Курсовые работы

Курсовая работа 1. Создание клиентских приложений

В следующих ниже заданиях следует разработать проект и создать главную страницу сайта, используя блочную верстку и HTML5, CSS и JavaScript. Разместить на страницу собственный логотип. Включить Интернет – сервис в соответствии с заданием. Разместить при возможности на бесплатном хостинге. Выполнить оценку качества.

Предлагаемые для разработки темы сайтов:

1. Создать новостной сайт с подключаемыми с других сайтов информерами: курсы валют, прогноз погоды, гороскопы, спортивные новости, анекдоты,
2. Создать многоязыковый сайт с возможностью перевода страницы на английский, немецкий или китайский языки. (Информер на translate.ru. или другой)
3. Визитка для ученого или учреждения + баннер.
4. Математический сайт. Возможность набора и чтения математического текста + ссылки на ресурсы
5. Игровой сайт.
6. Музыка. Каталог+проигрыватель+мультимедия.
7. Системы активного отображения информации: чаты, блоги, Wiki
8. Системы управления контентом.
9. Информационный сайт.
10. Блоги. Микроблоги(twitter). Live Journal — сервис для ведения блогов.

Создать: Общие комнаты, Комнаты с приватными сообщениями

11. Конференция. Прием заявок, прием тезисов, рассылка сообщений и приглашений. Конференция on-line.
12. Клуб по интересам.
13. Web 2.0. Социальные сети. Создание собственных ресурсов.
14. Web 2.0. Wiki-проекты.
15. Создать сервис для отправки открыток на e-mail. Предоставить пользователю выбор вида открытки и посылаемого текста поздравления или создания собственного текста и открытки. Предусмотреть список рассылки.
16. Создать сервис, который мог бы рандомно генерировать задание для студента, причем задания должны быть разные и зависили бы от уровня сложности, который вводится пользователем.
17. Создать галерею фотографий с возможностью оценивания (голосования). Очередность отображения фотографий зависит от их рейтинга и изменяется.
18. Написать игровой сайт: Игра «Пятнашки», «Кто хочет стать миллионером?», «Крестики-Нолики», «Морской бой» против компьютера или другие.
19. Создать галерею фотографий с возможностью оценивания. Очередность отображения фотографий зависит от рейтинга.
20. Интернет фотоальбом.
21. Разработка интегрированного приложения персонального сайта и Facebook (или других соцсетей).
22. Разработка сайта дошкольного учреждения.
23. Разработка сайта школьного учреждения.
24. Разработка сайта кафедры.
25. Разработка сайта конференции.
26. Разработка сайта интернет - конференции.
27. Разработка сайта научного учреждения БГУ.
28. Разработка сайта факультета.

29. Разработка сайта общества женщин.
30. Разработка сайта общества (клуба) мужчин.
31. Интернет-магазин (по продаже цветов, по продаже компьютерной техники, по продаже аудио/видео CD дисков и др.).
32. Сайт компании по производству (например печек для бань).
33. Сайт рекламирующий определенный товар (например спиннинги).
34. Сайт болельщиков какой-либо спортивной команды.
35. Сайт студенческой группы.
36. Сайт школьного учителя-предметника.
37. Сайт писателя.
38. Сайт поклонников какой-либо знаменитости.
39. Сайт турагенства.
40. Сайт букмекерской конторы.
41. Сайт любителей логических игр.
42. Язык разметки HTML-5. Новые возможности и особенности.
43. Рисунки на веб-сайте.
44. Android и смартфон.
45. Создание сайта тестирования интеллекта или знаний.
46. Использование социальных сетей для образования.
47. Сайт для музыки
48. Учебный сайт. Изучаем английский язык
49. Электронный журнал
50. Бизнес – сайт
51. Корпоративный сайт
52. Книжный магазин
53. Музыкальный магазин
54. Интернет-магазин по продаже косметики.
55. Игровой сайт
56. Служба знакомств
57. Словарь переводчик
58. Электронный учебник
59. Зоопарк
60. Путешествие, отдых
61. Сайт преподавателя университета.
62. Служба знакомств
63. Словарь переводчик
64. Изучаем английский язык.
65. Сайт студенческой группы.
66. Сайт поклонников какой-либо знаменитости.
67. Сайт турагенства.
68. Сайт букмекерской конторы.
69. Сайт любителей логических игр.
70. Сайты, предоставляющие услуги в виде различных сервисов.
71. Сайты компьютерных игр.

Поместить на сайт один или несколько из следующих сервисов:

1. Банеры.
2. Поисковые роботы и “невидимая паутина”.
3. Индексы цитирования веб-сайтов и способы их повышения.
4. Разработка приложений для Facebook.
5. Создание контекстного меню.

6. Таймер-часы.
7. Приветствие посетителя с учетом времени суток(утро, день, вечер).
8. Календарь.
9. Формы. Проверка информации, вводимой и посредством форм. Проверка наличия во вводимой строке определенных символов.
10. Вычисления на веб-сайте.
11. Создать игру типа: “Угадай задуманное число”.
12. Создать различные меню на JavaScript.
13. Создать объект калькулятор.
14. Создать объект бегущая строка.
15. Перекодировщик русских символов
16. Автоматическая регистрация страниц на поисковых системах. Программа выполняется на стороне клиента.
17. Время в JavaScript (получение и установка даты-времени).
18. Создать страницу с «бегущим текстом» в строке состояния браузера
19. Задание задержек по времени при выполнении функций. Программирование картинок.
20. Создать Web-страницу со слоем, появляющимся при нажатии на кнопку «Невидимка», и исчезающем при повторном нажатии на кнопку «Невидимка».
21. Программирование форм. Создать Web-страницу с викториной из n вопросов с 5-ю возможными вариантами ответов.
22. Голосование
23. Мобильные коммуникации и Интернет
25. Органайзер
26. Авторизация на JavaScript
27. Решение квадратных уравнений -------x2 +------x +-------- =0
28. Youtube — видеосервис
29. Платежные системы
30. Перевод сайта на английский / с английского языка.
31. Сайт на белорусском языке.
32. Математический сайт. Встроить MathML в HTML.
33. Информеры. Погода, курсы валют, гороскоп: например, на afn.by

<a href= “http://www.afn.by/”>
<ima stc= “http://www.afn.by/finances/ticket/”>
border=”0”/> <a> – включение инфорера

Курсовая работа №2. Сайты клиент-сервер

Каждая задача состоит из 2 частей: интерфейсная (клиентская) часть и административная часть. Адиновская часть используется для обновления и дополнения материала сайта. Вход в адиновскую часть должен быть закрыт паролем. Все адиновские страницы также должны быть защищены.

1. **Система управления лентой новостей**.

Постановка задачи: Разработать систему управления лентой новостей.

а) Интерфейсная часть представляет собой HTML-страницу, на которой выводятся последние N новостей. Число N задается в системе администрирования. Новости выводятся в кратком виде: *Дата, Анонс новости, Ссылка на подробнее*

Внизу выводится ссылка на архив, при нажатии на которую пользователь попадает на страницу, где выводится архив в следующем виде:

Апрель 2003 (5)
Март 2003 (12)

Февраль 2003 (8)

…

При нажатии на название месяца, выводится страница, содержащая все новости в кратком виде за этот месяц. При нажатии на «подробнее», выводится полный вид новости. На странице «подробнее» (полный вид новости) выводится дата, полный текст новости, автор. Имеется ссылка на страницу новостей.

b) Административная часть.

Вход в административную часть должен быть закрыт паролем. Все административные страницы должны быть защищены, чтобы пользователь не мог получить доступ к администрированию новостей. Реализовать с помощью механизма сессий.

В административной части администратор может:

А) Редактировать существующие новости (изменять, удалять)

Б) Добавлять новые новости

В) Изменять число выводимых на странице новостей

Реализовать хранение новостей в базе данных MySQL.

2. Каталог товаров

Разработать каталог товаров.

На первой странице выводится список разделов товаров. При переходе в раздел, пользователь попадает на страницу товаров из этого раздела, который выводится в табличном виде. Поля для товара: наименование, описание, цена.

Администратор имеет возможность ввести новый раздел, изменить или удалить существующий. Также администратор может добавить, изменить, удалить любой товар в любом разделе.

3. Система вопрос/ответ

Пользователь может просмотреть страницу с вопросами и ответами (FAQ). Внизу в форме он может ввести свой вопрос. Этот вопрос сохраняется в базе и доступен администратору в системе администрирования.

Администратор в системе администрирования, может посмотреть поступившие вопросы и дать на них ответы, после чего они появляются на странице FAQ.

4. **Система** опросов

Система интернет-опросов предназначена для проведения онлайновых опросов и голосований. Интерфейсная часть состоит из HTML-страницы, на которой выводится вопрос и форма, в которой выводятся варианты ответа. Пользователь может выбрать один из вариантов ответа, и ответить на вопрос. После ответа, пользователю выводятся текущие результаты опроса.

Административная часть. Администратор может посмотреть результаты текущего опроса, остановить текущий опрос, добавить новый опрос.

5. Система конкурсов и тестов

В один конкурс входит несколько вопросов. На каждый вопрос несколько вариантов ответа. Администратор при вводе ответов указывает правильные ответы.

Пользователь, отвечая на вопросы, отмечает правильные на его взгляд ответы. После завершения ввода, система проверяет правильность ответов и выводит результат.

6. Форум

Разработать линейную систему форумов. Пример: www.kv.by/forum

7. Простейший Интернет-магазин + база данных+Ajax

– просмотр ассортимента товаров – Витрина

– отметить товары

– принять контактные данные заказчика.

- Корзина заказов. Ожидание заказа

– передать магазину список заказанных товаров и данные покупателя

8. Создать сайт с подключаемыми с других сайтов информерами: курсы валют, прогноз погоды, гороскопы, спортивные новости, анекдоты, переводчик на иностранный язык+проектирование на UML

9. Создать многоязыковый сайт с возможностью перевода страницы на английский, немецкий или французский языки. (Информер на translate.ru. или другой)

10. Визитка для ученого или учреждения + гостевая книга+баннер.

11. Учебный сайт+проектирование на UML.

12. Создать новостной сайт с подпиской по технологии RSS, которая позволяет транслировать информацию с других сайтов. Для подписки необходимо установить программу – агрегатор типа News Reader, в которую помещаются ссылки на транслируемые каналы.

13. Музыка. Каталог+проигрыватель+мультимедия.

14. Системы активного отображения информации:

– чаты, блоги, Wiki

15. Системы управления контентом.

16. Информационный сайт: Поиск телефона по фамилии или по адресу.

17. Библиотека: Просмотр каталога или поиск в каталоге. Заказ книги, если книга свободна. Для электронной книги просмотр или скачивание.

18. Электронный научный журнал. Прием статей от автора. Подписка, рассылка, реклама, скачивание.

19. Интернет – Конференция. Прием заявок, прием тезисов, рассылка сообщений и приглашений. Конференция on-line.

20. Система контроля в учреждении образования. Web – журнал и Web- дневник.

21. Страница приема заданий содержит таблицу со списком студентов и заданий. Студенты могут только посмотреть текущее состояние дел. Преподаватели, после ввода пароля, могут поставить или убрать галочку, означающую факт сдачи задания. Должна быть предусмотрена возможность сортировки списка студентов по имени или по количеству сданных заданий. Возможно использование Java и Java Script.

22. Информационная система отслеживания колебаний котировок акций

23. Интернет аукцион. Создать Интернет-аукцион. Пользователь указывает min цену своего товара и время до конца продажи данного товара. Сделать ботов, которые в случае не достижения min цены будут ее поднимать на 20 секунд до конца продажи данного товара.

24. Создать приложение, позволяющее тестировать качество работы хостеров. Приложение должно посылать запрос на указанный ресурс с заданной периодичностью, проверяя доступен ли сайт в данный момент времени. На основе полученных данных приложение формирует статистику работы.

25. Создать сервис для отправки открыток на e-mail. Предоставить пользователю выбор вида открытки и посылаемого текста поздравления или создания собственного текста и открытки. Предусмотреть список рассылки.

26. Создать сервис, который мог бы рандомно генерировать задание для студента, причем задания должны быть разные и зависили бы от уровня сложности, который вводится пользователем.

27. Создать галерею фотографий с возможностью оценивания (голосования). Очередность отображения фотографий зависит от их рейтинга и изменяется.

28. Автоматический генератор сайтов (по шаблонам).

29. Написать игровой сайт: Игра «Пятнашки», «Кто хочет стать миллионером?», «Крестики-Нолики», «Морской бой» против компьютера или другие.

30. Создать галерею фотографий с возможностью оценивания. Очередность отображения фотографий зависит от рейтинга.

31. Написать сервис-редактор изображений, возможности: конвертирование в различные форматы, изменять размер изображения и сохранять его, реализовать различные фильтры и т. д.

32. Написать скрипт чата. Особенности:

- Страница входа
- Страница с личными настройками
- Общие комнаты
- Комнаты с приватными сообщениями
- Возможность добавить картинку-аватар.

33. Скрипт проверки наличия новых личных сообщений на каком-либо популярном форуме (сайте). Особенности:

- Получение html-страницы при помощи CURL (вход в авторизированный раздел отправкой post-запроса на страницу логина)
- Проверка наличия новых сообщений на странице (NegExps)
- Отправка уведомлений на почту

34. Быстрая разработка сайтов на основеWordPress
35. Быстрая разработка сайтов на основе Joomla!
36. Быстрая разработка сайтов на основе Drupal
37. Бизнес
38. Веб-дизайн
39. Фотосайты
40. Интерьер и мебель
41. Ландшафт
42. Автомобили
43. Спорт
45. Кафе и рестораны
46. Ночные клубы
47. Животные
48. 3D шаблоны
49. Фотогалереи
50. Свадьба
51. Семья
52. Мода
53. Магазин подарков
54. Медицина
55. Отели

Использование сервисов

1. Обеспечение безопасности
2. Защита от автоматического заполнения форм (CAPTCHA)
3. Подтверждение email
4. log авторизаций (фиксация действий пользователей)
5. Уведомления администратору
6. Разделение прав доступа
7. Поддержка
8. Партнерская программа
9. Программа сертификации
10. Встроенная онлайн поддержка
11. Документация по системе
12. Публичный форум

13. Публичная рассылка
14. Сторонние разработчики
15. Перетаскиваемый контент
16. ЧПУ (Дружественный URL)
17. Встроенный редактор изображений
18. Пакетная загрузка файлов
19. Отмена действий
20. Визуальный редактор (WYSIWYG)
21. Закачка архива с распаковкой
22. Восстановление объектов из корзины
23. Коррекция опечаток
24. Быстрое редактирование
25. Копирование объектов
26. Скины
27. Гибкость системы
28. Локализация интерфейса
29. Многосайтовость
30. Многодоменность
31. Поддержка UTF-8
32. Производительность
33. Кеширование страниц
34. Memcached
35. Включенный функционал
36. Статистика посещений
37. Управление стилями и шаблонами
38. Документооборот
39. Управление рекламой
40. SEO
41. Поведенческие технологии
42. Экспорт RSS. Агрегатор News Reader.
43. FAQ
44. Формы обратной связи . Опросы .Голосования. Поиск. Форум
45. Рассылка
46. Новости
47. Каталог
48. Файловый менеджер
49. Внешняя фотогалерея
50. Внутренний фотобанк
51. Управление пользователями
52. Импорт из Excel
53. Экспорт в формат Яндекс.Маркет.
54. поддержка мобильной связи, рассылка SMS сообщений
55. Баннеры. Банерные сети.
56. Персонализированная on-line реклама.
57. Система коллективной разработки приложений assembla.com.

Курсовая работа №3. Технологии разработки Веб – приложений

Проекты сайтов

Создать сайт на основе HTML5+CSS+JavaScript+Jquery+Ajax+PHP средств.

Для каждого Веб – приложения реализовать:

- *интерфейсную (клиентскую) часть.*

-*Административную часть.*

Вход в административную часть и на все административные страницы должен быть защищен паролем. Реализовать с помощью механизма сессий.

В административной части администратор может:

Редактировать существующие новости (изменять, удалять)

Добавлять новые новости

Изменять число выводимых на странице новостей

Реализовать хранение новостей в базе данных MySQL.

- *Для каждого сайта рассмотреть вопросы автоматизации проектирования и оценки качества созданных приложений в сравнении с существующими приложениями*

Перечень заданий

1. Юзабилити. Оценка качества сайта с позиций юзабилити
2. Многоязычный контент. Перевод сайта на другие языки, включая китайский.
3. SEO. Оценка качества сайта с позиций SEO.
4. Электронный научный журнал. Прием статей от автора. Подписка, рассылка, реклама, скачивание.
5. Web – Конференция. Прием заявок, прием тезисов, рассылка сообщений и приглашений. Конференция on-line. Особенности:
   - Страница входа
   - Страница с личными настройками
   - Общие комнаты
   - Комнаты с приватными сообщениями
   - Возможность добавить картинку-аватар.
6. Система контроля в учреждении образования. Web – журнал и Web- дневник. Страница приема заданий.Содержит таблицу со списком студентов и заданий. Студенты могут только посмотреть текущее состояние дел. Преподаватели, после ввода пароля, могут поставить или убрать галочку, означающую факт сдачи задания. Должна быть предусмотрена возможность сортировки списка студентов по имени или по количеству сданных заданий. Возможно использование Java и Java Script.
7. Спамилка серверная (спамит по форумам PHP и гостевым книгам).
8. Web 2.0. Семантические сети. Оценка качества сайта с позиций контента.
9. Интернет аукцион. Создать Интернет-аукцион. Пользователь указывает min цену своего товара и время до конца продажи данного товара. Сделать ботов, которые в случае не достижения min цены будут ее поднимать на 20 секунд до конца продажи данного товара.
10. Создать приложение, позволяющее тестировать качество работы хостеров. Приложение должно посылать запрос на указанный ресурс с заданной периодичностью, проверяя доступен ли сайт в данный момент времени. На основе полученных данных приложение формирует статистику работы.
11. Создать сервис для отправки открыток на e-mail. Предоставить пользователю выбор вида открытки и посылаемого текста поздравления или создания собственного текста и открытки. Предусмотреть список рассылки.

12. Создать сервис, который мог бы рандомно генерировать задание для студента, причем задания должны быть разные и зависили бы от уровня сложности, который вводится пользователем.

13. Автоматический генератор сайтов (по шаблонам).

14. Написать игровой сайт: Игра «Пятнашки», «Кто хочет стать миллионером?», «Крестики-Нолики», «Морской бой» против компьютера или другие.

15. Написать сервис-редактор изображений. Возможности: конвертирование в различные форматы, изменять размер изображения и сохранять его, реализовать различные фильтры и т. д.

16. Скрипт проверки наличия новых личных сообщений на каком-либо популярном форуме (сайте). Особенности:

- Получение html-страницы при помощи CURL (вход в авторизированный раздел отправкой post-запроса на страницу логина)
- Проверка наличия новых сообщений на странице (NegExps)
- Отправка уведомлений на почту

17. Написать парсер (Вырезание статей на данную тему со всех сайтов).

18. Сервис по определению доменов, которые заканчиваются или брошенные. (Работа с сервисом Whois) зона .com; .info.

19. Создать систему оценки учреждения образования по присутствию в Интернет. Подобную Webometrics для университетов.

20. Использование cookie & session: Пользователю, посетившему хотя бы однажды раздел музыка, предлагается банер с рекламой магазина музыкальных инструментов. Аналогично отдел строительных инструментов и др.

21. Http-туннель. Создание цепи анонимных http, socks proxy.

22. Сайт голосования. Нахождение мультов (лиц, зарегистрировавшихся с одного IP несколько раз) и удаляет несколько последних, оставляет первого.

23. Бизнес – сайт+проектирование на UML.

24. Корпоративный сайт+проектирование на UML.

25. Игровой сайт. Определение реакции и оценка степени усталости. +Flash+MultiMedia

26. Системы управления контентом. Сравнение WordPress, Joomla,Drupal на примерах различных сайтов

27. Синхронизация Веб – приложения с 1С

28. Технологии Raby on Rails

29. Google Maps — Google-карты

30. Flickr — онлайн-фотоальбом

31. Netvibes — Персональный десктоп

32. Digg.com — Новостной ресурс

33. Pligg — Веб 2.0 CMS

34. Quintura — Визуальный поисковик с интуитивной картой подсказок

35. MySpace — сайт сетевых сообществ

36. Last.fm — музыкальное сообщество

КОНТРОЛЬ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

ТЕСТИРОВАНИЕ В СОП EUNIVERSITY

Тесты для самоподготовки расположены в электронной библиотеке www.w3schools.com.

Тестирование по модулям осуществляется по адресу http://www.quizful.net в соответствии с планом проведения контролирующих мероприятий:

Тематика модулей:

1 семестр

	Время тестирования
Гафические приложения	ноябрь

2 семестр

	Время тестирования
HTML CSS	Март, апрель

3 семестр

Javascript	Ноябрь

4 семестр

PHP	Апрель

КОНТРОЛИРУЮЩИЕ ЗАДАНИЯ

ВОПРОСЫ

1. Укажите корректный JavaScript синтаксис для вывода строки "Hello World"?
2. Какая из секций подходит для вставки кода JavaScript?
3. Укажите правильный синтаксис для вызова внешнего кода скрипта "xxx.js"?
4. Как вывести строку "Hello World" в окно alert?
5. Как правильно создать и вызвать функцию "myFunction"?
6. Сколько различных видов циклов используется в JavaScript?
7. Как правильно записать массив JavaScript?
   var txt = new Array(1:"tim",2:"kim",3:"jim")
   var txt = new Array="tim","kim","jim"
   var txt = new Array("tim","kim","jim")
   var txt = new Array:1=("tim")2=("kim")3=("jim")
8. Укажите корректный JavaScript синтаксис для корректного открытия нового окна "window2" ?
   window.open("http://www.w3schools.com","window2")
   open.new("http://www.w3schools.com","window2")

new.window("http://www.w3schools.com","window2")
new("http://www.w3schools.com","window2")

9. Как вывести сообщение в окно статуса браузера?
   statusbar = "put your message here"
   status("put your message here")
   window.status("put your message here")
   window.status = "put your message here"
10. Какие из приведенных ниже ссылок являются корректными?
   а) window. document. form [0]
   б) self.entryForm.submit()
   в) document.forms[2].name
   *г) document.getElementById("firstParagraph")
   *д) newWindow.document.write("Howdy")
11. Для каждого из приведенных ниже выражений укажите, чему равно выражение someVal после выполнения следующих операторов JavaScript.
   var someVal = 2;
   someVal = someVal + 2;
   someVal = someVal * 10;
   someVal = someVal + "20"; //4020
   someVal = "Robert";
12. Какое значение выведет на экран следующий код:
   var myArr = new Array();
   alert(typeof(myArr));
13. Какое число выведет на экран следующий код:
   var i = 5;
   var n = 3;
   alert(i-- * ++n);
   (20)
14. В какой строке допущена ошибка при написании приведенного сценария?
   var userName = prompt("Ваше имя? ", "");
   if (userName = "root")
   { document.write(" Доступ разрешен "); }
   else { document.write(" В доступе отказано "); }
   (2) (userName == "root")
16. Какой из участков кода будет выполнен в браузере при работе приведенного JavaScript-сценария?
   var x = 1; var y = "1";
   if (x === y) { // Участок кода 1 } else { // Участок кода 2 }
   (2)
17. Какого рода информация в приведенном ниже HTML-дескрипторе, будет передана обработчиком события? Напишите функцию, отображающую в диалоговом окне предупреждение с предназначенной для передачи информацией.
<input type="text" name="phone" onchange="format(this.value)">

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

	ОСНОВНАЯ ЛИТЕРАТУРА	
1.	Кристиансен Т., Торкингтон Н. Perl: Библиотека программиста :Пер. с англ.- СПб.: Издательство «Питер», 2000. - 736с.: ил.	
2.	Хейл, Бернард Ван. JDBC: Java и базы данных :Пер. с англ. М.,1999.-320с.	
3.	Робин Никсон. «Создаем динамические веб-сайты с помощью PHP, MySQL и JavaScript», – Питер , 2011.	
4.	Люк Веллинг, Лора Томсон. «Разработка Web-приложений с помощью PHP 5 и MySQL 5. PHP5», – Williams, 2008.	
5.	Владимир Дронов. «HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов», – BHV, 2010.	
	ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА	
6.	Романчик В.С. «Web-программирование». Мн., БГУ, 2003.	
7.	Блинов И.Н., Романчик В.С. Объектно-ориентированное программирование на языке С++: учеб.-метод. пособие для студентов мех.-мат. фак. – Мн.: БГУ, 2007.	
8.	Блинов И.Н., Романчик В.С. Java 2. Практическое руководство. – Мн.: Универсалпресс, 2005.	
9.	Блинов И.Н., Романчик В.С. Java. Промышленное программирование: практ. пособие. – Мн: УниверсалПресс, 2007. – 702 с.	
10.	Романчик В.С. Введение в PHP. Электронный учебник. – Мн.: БГУ, 2013, 450с.	
11.	Тодд Томлинсон. «CMS Drupal 7. Руководство по разработке системы управления веб-сайтом». – Williams, 2011.	
12.	Викрам Васвани. «Zend Framework: разработка веб-приложений на PHP», – Питер, 2012.	
13.	Майкл Моррисон. «Изучаем JavaScript», – Питер, 2012.	

ТИПОВАЯ ПРОГРАММА КУРСА

Министерство образования Республики Беларусь
Учебно-методическое объединение вузов Республики Беларусь
по естественнонаучному образованию

УТВЕРЖДАЮ
Первый заместитель Министра
образования Республики Беларусь
_____ А.И. Жук

«_31_» ____12_____2008 г.

Регистрационный № ТД-_G.172_/тип.

Web-программирование

Типовая учебная программа
для высших учебных заведений по специальности
1-31 03 01 – Математика (по направлениям)
(1-31 03 01-05 – Математика (информационные технологии))

СОГЛАСОВАНО

Председатель УМО вузов Республики Беларусь по естественнонаучному образованию

_____ В.В. Самохвал

«___» _____ 2008 г.

СОГЛАСОВАНО

Начальник Управления высшего и среднего специального образования

_____ Ю.И. Миксюк

«_31_» ____12____ 2008 г.

Первый проректор Государственного учреждения образования «Республиканский институт высшей школы»

_____ И.В. Казакова

«_12_» ___12_____ 2008 г.

Эксперт-нормоконтролер

_____ С.М. Артемьева

«_12_» ___12_____ 2008 г.

Минск 2008

СОСТАВИТЕЛИ:

Валерий Станиславович Романчик, заведующий кафедрой численных методов и программирования Белорусского государственного университета, кандидат физико-математических наук, доцент;

Панина Светлана Александровна, старший преподаватель кафедры численных методов и программирования Белорусского государственного университета.

РЕЦЕНЗЕНТЫ:

Кафедра информационных технологий Учреждения образования «Белорусский государственный университет культуры и искусств» (заведующий кафедрой – кандидат физико-математических наук, доцент П.В. Гляков);

А.А. Черняк, профессор кафедры математики Учреждения образования «Белорусский государственный педагогический университет им. М. Танка», доктор физико-математических наук.

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ В КАЧЕСТВЕ ТИПОВОЙ:

Кафедрой численных методов и программирования Белорусского государственного университета

(протокол № 8 от 6 марта 2008 г.);

Научно-методическим советом Белорусского государственного университета

(протокол № 3 от 27 марта 2008 г.);

Научно-методическим советом по математике и механике Учебно-методического объединения вузов Республики Беларусь по естественнонаучному образованию

(протокол № 3 от 10 апреля 2008 г.).

Ответственный за выпуск: Романчик Валерий Станиславович

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Программа дисциплины «Web-программирование» ориентирована на студентов, обучающихся по специальности 1-31 03 01 – «Математика» по направлению: информационные технологии (1-31 03 01-05).

Данная дисциплина изучается студентами указанного направления на младших курсах, что позволяет применять полученные знания в последующем обучении.

В соответствии с образовательными стандартами по указанным специальностям выпускник должен

знать:

- способы представления, поиска, передачи и хранения информации;
- методы решения научно-технических и информационных задач;
- современные информационные технологии;

уметь:

- решать типовые задачи математики и информатики;
- работать на современных вычислительных средствах;
- применять современные информационные технологии и методы реализации решения распределенных web-приложений.

В данном курсе рассмотрены базовые знания по представлению, организации и передаче информации и структуре Web, этапы разработки веб-сайта от планирования до рекламы, принципы гипертекста, основы языка HTML и спецификации CSS (Каскадных таблиц стилей), популярные принципы верстки сайтов, программы Macromedia Dreamweaver, CorelDRAW, Adobe Photoshop, Flash. Скрипты на клиенте (JavaScript) позволяют сделать сайт динамическим, скрипты на сервере (PHP) позволяют реализовать распределенное web-приложение типа клиент-сервер.

Комплексная программа позволит слушателю стать квалифицированным разработчиком Web-сайтов, познакомит с технологиями проектирования сайтов и их оформления.

Всего – 204 часа. Из них аудиторных – 136 часов, в том числе: лекции – 68 часов, лабораторные занятия – 36 часов, практические занятия – 32 часа, самостоятельная работа – 68 часов.

В программе рассматриваются следующие вопросы.

Введение в WWW. В данном разделе рассматриваются основные понятия Web, базовые знания по стилистике и структуре Web, технологии и протоколы, обеспечивающие работу глобальной сети, этапы разработки web-сайта, от планирования до рекламы.

Язык разметки гипертекста HTML. В данном разделе изучаются основы языка HTML, популярные приемы верстки сайтов.

Каскадные таблицы стилей (Cascading Style Sheets, CSS). Изучаются технологии описания внешнего вида документа, написанного языком разметки.

Графический пакет CorelDRAW. В рамках курса предполагается освоение редактора векторной компьютерной графики CorelDRAW.

Графический пакет Photoshop. В данном разделе рассматриваются работа с программой Photoshop, которая является лидером в области коммерческих средств редактирования растровых изображений.

HTML-редактор Dreamweaver – один из наиболее популярных HTML-редакторов.

Adobe Flash. Adobe Flex. ActionScript. Рассматривается язык для разработки интерактивных мультимедийных приложений.

Скрипты на клиентской странице. Язык JavaScript. Рассматривается язык для создания динамических приложений-клиентов.

Создание серверных приложений. Изучаются вопросы использования клиент – серверных технологий CGI, ISAPI. Протокол HTTP. Web – серверы. Технологии PHP.

ПРИМЕРНЫЙ ТЕМАТИЧЕСКИЙ ПЛАН

№ разделов	Наименование разделов	Аудиторные часы			
		Всего	Лекции	Лабораторные занятия	Практические занятия
1.	Введение в WWW.	2	2	0	0
2.	Язык разметки гипертекста HTML.	18	8	6	4
3.	Каскадные таблицы стилей. Введение в CSS	8	2	2	4
4.	Графический пакет CorelDRAW.	18	6	6	6
5.	Графический пакет PhotoShop	6	4	2	0
6.	Html – редактор Dreamweaver	4	2	2	0
7.	Adobe Flash. Adobe Flex. ActionScript	20	10	5	5
8.	Скрипты на клиентской странице. Язык JavaScript	16	8	3	5
9.	Создание серверных приложений. Язык PHP	28	10	10	8
10.	Введение в Базы Данных	2	2	0	0
11.	Технологии Java	2	2	0	0
12.	Технологии ASP.NET	2	2	0	0
13.	XHTML и XML	2	2	0	0
14.	Технологии Web 2.0	2	2	0	0
15.	Веб-сервисы	2	2	0	0
16.	Администрирование и безопасность	2	2	0	0
17.	Продвижение и оптимизация сайтов	2	2	0	0
	Итого	136	68	36	32

СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

Раздел 1. Введение в WWW

Цель, содержание и особенности курса. История Интернета. Технологии и протоколы, обеспечивающие работу глобальной сети. Протоколы TCP/IP. IP – адресация. DNS – адресация. Хостинг. Виды Web –

приложений (сервисы интернета). Браузеры. Web – дизайн и проектирование. Этапы создания сайта. Современные подходы к промышленному созданию программ. Язык программирования программных и информационных систем UML.

Раздел 2. Язык разметки гипертекста HTML

2.1 Введение в HTML. Спецификация HTML. Структура HTML-документа.

2.2 Текст. Форматирование текста: теги логического и физического форматирования. Структурное форматирование: текстовый блок, абзац, заголовок. Специальные символы. Шрифты.

2.3 Списки. Концепция списков в HTML. Нумерованные списки. Маркированные списки. Параметры элемента списка. Многоуровневые списки. Списки определений.

2.4 Графика на web-странице. Характеристика графических стандартов. Вставка графики в HTML-документ. Описание графики в HTML-документе. Параметры тега <img>. Карты-изображения.

2.5 Гипертекстовые ссылки. Создание ссылок. Закладки. Ссылки-изображения. Виртуальная навигация.

2.6 Таблицы. Табличное представление данных. Создание таблиц. Правила описания таблиц. Группировка данных. Вложенные таблицы.

2.7 Фреймы. Области применения фреймов. Правила описания фреймов.

2.8 Формы. Структура пользовательских форм.

2.9 META-теги. DOCTYPE. Создание и типы META-тегов.

Раздел 3. Каскадные таблицы стилей (Cascading Style Sheets, CSS)

3.1 Введение в CSS. Спецификация CSS. Связь стилей с Web-страницей. Организация файлов таблиц стилей. Шрифт, текст, фон, цвет. Расположение элементов. Границы элементов. Свойства таблиц, свойства списков, псевдостили гиперссылок и текста.

3.2 Типы верстok. Табличная верстка. Блочная верстка. Другие виды верстok.

Раздел 4. Графический пакет CorelDRAW.

4.1 Основы композиции и цветоведения.

4.2 Работа с документом. Создание, открытие и сохранение документов. Импорт и экспорт информации. Форматирование документа.

4.3 Интерфейс CorelDRAW. Устройство интерфейса.

4.4 Технологические возможности. Слои. Стили. Символы.

4.5 Обработка векторных объектов. Рисование и обработка линий и фигур. Наложение изображений. Заливка. Обводка. Регулировка прозрачности.

4.6 Объединение объектов. Операция комбинирования, слияния, обрезки, пересечения, упрощения.

4.7 Трансформация объектов. Создание эффектов. Переход, контур, искажение, оболочка, выдавливание, тень, линза, перспектива.

4.8 Обработка растровых изображений. Преобразование цветовых форматов. Регулировка прозрачности. Тоновая коррекция. Создание эффектов.

4.9 Обработка текста. Типы текста. Используемые шрифты. Ввод текста. Операции оформления. Форматирование текста. Редактирование текста.

Раздел 5. Графический пакет Photoshop

5.1 Работа с документом. Импорт и экспорт информации. Форматирование документа.

5.2 Интерфейс Photoshop. Устройство интерфейса.

5.3 Выделение. Выделение слоя. Выделение прямоугольной или эллиптической области. Выделение произвольной формы. Трансформация выделения.

5.4 Компоновка. Перемещение. Копирование. Увеличение резкости и размытие изображений. Использование линеек и направляющих линий.

5.5 Слои. Операции над слоями. Инструменты для работы над слоями. Слияние и объединение слоев.

5.6 Событыя. Использование палитры History. Использование снимков. Восстановление и стирание фрагментов изображения.

5.7 Команды корректировки. Основные сведения о командах корректировки. Корректирующие слои. Команды корректировки.

5.8 Выбор цвета. Основной и фоновый цвет.

5.9 Раскрашивание. Команды Curves и Levels.

5.10 Рисование. Инструмент Brush.

5.11 Градиенты. Применение градиента в качестве слоя заливки. Использование инструмента Gradient.

5.12 Дополнительные сведения о слоях. Непрозрачность слоя. Эффекты слоя. Смешивание слоев. Маски слоя. Группы отсечений. Связывание слоев.

5.13 Контуры и фигуры. Контуры. Операции над контурами. Контуры отсечения слоев. Фигуры.

5.14 Работа с текстом. Создание и редактирование текста. Использование атрибутов абзаца. Специальные эффекты для текста.

5.15 Фильтры.

5.16 Общие установки программы Photoshop. Печать. Работа с Web.

Раздел 6. HTML-редактор Dreamweaver

6.1 Управление сайтом средствами редактора Dreamweaver. Подготовка редактора к работе с web-сервером. Работа с локальной и удаленной версией сайта. Работа с картой сайта.

6.2 Реализация структуры сайта. Соглашение об именах файлов. Создание заглавной страницы. Создание навигационной панели. Подготовка шаблонов. Подготовка библиотеки элементов.

6.3 Форматирование HTML-страниц. Элементы структурной разметки. Визуальное форматирование текста. Создание гипертекстовых ссылок. Создание списков. Создание и форматирование таблиц. Создание и использование стилей CSS. Импорт текста в формат HTML.

6.4 Вставка изображений и других элементов. Использование графических материалов. Создание карты ссылок на изображении. Вставка изменяющегося изображения. Определение интерактивных свойств объектов. Вставка Flash-кнопок. Использование подключаемых модулей.

Раздел 7. Adobe Flash. Adobe Flex. ActionScript

7.1 Среда и инструменты Flash.

7.2 Содержимое фильма: создание и управление. Работа с текстом. Работа с объектами. Объекты многократного использования. Слои.

7.3 Создание анимации в среде Flash. Анимация с использованием временной шкалы. Работа с клипами. Использование сцен при работе с анимационными фильмами.

7.4 Интерактивные фильмы. Базовые действия ActionScript. Создание интерактивных элементов управления.

7.5 Язык сценариев. ActionScript: начальные сведения. Среда разработки ActionScript. Синтаксис ActionScript. Основные понятия.

7.6 Типы данных. Переменные. Определение типа объекта данных. Числа. Математические функции и константы. Строки. Кодировка символов. Логические величины. Тип undefined. Тип null. Объекты. Клипы. Преобразование типов данных. Создание переменных. Особенности типизации переменных. Локальные переменные функций. Глобальные переменные. Особенности операции присваивания.

7.7 Функции, операторы, предложения. Создание функций. Вызов функций. Особенности цепочки областей видимости функций. Объект arguments. Функции как объекты. Виды операторов. Блок предложений. Предложения var и function. Предложения выражений и пустые предложения. Предложение return. Предложение with. Условные предложения. Циклы.

7.8 Массивы, события. Создание массивов. Длина массива. Свойство length. Особенности реализации массивов в ActionScript. Модель событий

Генератор—Листенеры. Событийные методы. Обновление экрана при событиях.

7.9 Объектно-ориентированное программирование. Основные принципы объектно-ориентированного программирования. Класс Object. Объектно-ориентированное программирование в стиле ActionScript.

7.10 Клипы, кнопки, работа с мышью и клавиатурой. Отличия клипов от объектов. Клипы как носители кода. Особенности основной временной диаграммы _root. Создание экземпляров клипов. Виртуальные слои клипов. Импорт внешних фильмов и изображений. Имена экземпляров клипов. Ссылки на клипы. Система координат клипов. Прозрачность и видимость клипа. Перемещаемые клипы. События кнопок. Режим элемента меню. Управление кнопками при помощи клавиатуры. Клипы как кнопки. Работа с мышью. Контекстное меню. Работа с клавиатурой.

7.11 Работа с текстом, время и дата, работа со звуком. Создание текстовых полей. Удаление текстовых полей. Текстовые поля как визуальные объекты. Задание и извлечение текста поля. Настройка текстового поля. Настройка стиля текста. Класс TextFormat. Работа со шрифтами. Событие onChanged. Прокрутка текстовых полей. Работа с фокусом. Форматирование текста при помощи HTML. Форматирование текста с использованием каскадных таблиц стиля (CSS). Работа с выделением. Работа со статичным текстом. Компьютерное время. Класс Date. Основные понятия теории цифрового звука. Событийный (event) и потоковый (stream) звук. Операции со звуком без использования программирования. Создание объектов класса Sound. Динамическое присоединение звука. Импортирование внешних MP3-файлов. Управление воспроизведением звуков.

Раздел 8. Скрипты на клиентской странице. Язык JavaScript

8.1 Создание динамических приложений-клиентов.

8.2 Типы данных. Переменные. Функции. Операторы. Массивы.

8.3 Объекты.

8.4 События. Кнопки. Работа с мышью и клавиатурой. Время и дата.

8.5 Работа с текстом. Работа со звуком. Технология AJAX.

Раздел 9. Создание серверных приложений.

9.1 Клиент – серверные технологии CGI, ISAPI. Протокол HTTP.

9.2 Web – сервер Apache. Администрирование web-серверов.

9.3 Скрипты на сервере.

9.4 Технологии PHP. PHP & AJAX.

9.5 Понятие о технологии PERL.

Раздел 10. Введение в Базы Данных

Основные понятия и классификация БД. Реляционные БД. Методы и средства проектирования БД. Язык SQL. Базы данных на сервере. Базы данных MySql и MSSql Server

Раздел 11. Технологии Java

Сервлеты и JSP. Web-сервер Tomcat.

Раздел 12. Технологии ASP.NET

Web –сервер IIS. Классы основных компонент ASP.NET. Свойства, методы и события. Работа с базами данных в ASP.NET. Технологии ADO.NET. Поддержка работы с пользователями. Контейнеры приложения и сеанса, использование cookies. Использование сторонних и разработка собственных Web-сервисов в ASP.NET. Расширители компонентов ASP.NET.

Раздел 13. XHTML и XML

Введение в расширяемый язык разметки. Обзор технологий, связанных с преобразованием, валидацией, хранением, взаимодействием с языками программирования и XML документами.

Основные уровни и проблемы при обмене информацией. Классификация информационных систем. Принципы дизайна XML документов.

Раздел 14. Технологии Web 2.0

Основные концепции WEB2.0. Синдикация и агрегация данных. Форматы RSS, ATOM. Популярные сервисы манипуляции с данными. Понятие и примеры машапов. Фолксономии и Таксономии как примеры построения семантических классификаторов данных. Примеры популярных WEB2.0 сервисов.

Математические основы построения и функционирования социальных сетей.

Раздел 15. Веб-сервисы

Основные концепции SOA и ROA. Способы организации доступа к сервису. Протоколы коммуникации. Основные принципы проектирования REST сервисов.

Раздел 16. Администрирование и безопасность

Проблема защиты информации в сети Интернет. Криптосистемы с открытым ключом. Цифровая подпись. Методы сжатия информации. Протокол HTTPS.

Раздел 17. Продвижение и оптимизация сайта

Основные способы продвижения сайтов в интернет. Создание веб-контента. Оптимизация сайта с точки зрения юзабилити. Анализ состояния сайта (Website Critique), SEO. Внутренние факторы ранжирования. Методы повышения ТИЦ, PR. Система оценки ссылок и обмен ссылками.

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

Рекомендуемые темы контрольных работ:

1. Создание сайта, с помощью технологий HTML и CSS.
2. Создание Flash-игры с помощью языка Action Script.

ИНФОРМАЦИОННАЯ ЧАСТЬ

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

	ОСНОВНАЯ ЛИТЕРАТУРА	
1.	Кристиансен Т., Торкингтон Н. Perl: Библиотека программиста :Пер. с англ.- СПб.: Издательство «Питер», 2000. - 736с.: ил.	
2.	Хейл, Бернард Ван. JDBC: Java и базы данных :Пер. с англ. М.,1999.-320с.	
3.	Робин Никсон. «Создаем динамические веб-сайты с помощью PHP, MySQL и JavaScript», – Питер , 2011.	
4.	Люк Веллинг, Лора Томсон. «Разработка Web-приложений с помощью PHP 5 и MySQL 5. PHP5», – Williams, 2008.	
5.	Владимир Дронов. «HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов», – BHV, 2010.	
	ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА	
6.	Романчик В.С. «Web-программирование». Мн., БГУ, 2003.	
7.	Блинов И.Н., Романчик В.С. Объектно-ориентированное программирование на языке С++: учеб.-метод. пособие для студентов мех.-мат. фак. – Мн.: БГУ, 2007.	
8.	Блинов И.Н., Романчик В.С. Java 2. Практическое руководство. – Мн.: Универсалпресс, 2005.	
9.	Блинов И.Н., Романчик В.С. Java. Промышленное программирование: практ. пособие. – Мн: УниверсалПресс, 2007. – 702 с.	
10.	Романчик В.С. Введение в PHP. Электронный учебник. – Мн.: БГУ, 2013, 450с.	
11.	Тодд Томлинсон. «CMS Drupal 7. Руководство по разработке системы управления веб-сайтом». – Williams, 2011.	
	Викрам Васвани. «Zend Framework: разработка веб-приложений на PHP», – Питер, 2012.	
	Майкл Моррисон. «Изучаем JavaScript», – Питер, 2012.	

ДОПОЛНИТЕЛЬНАЯ

1. Миронов Д.Ф. Corel Draw12 учебный курс. – СПб.: Питер, 2004. – 442 с.
2. Полонская Е.Л. Язык HTML. Самоучитель. — М.: Издательский дом "Вильяме", 2003. — 320 с.
3. Бадд Э., Молл К., Коллизон С. Мастерская CSS: профессиональное применение Web-стандартов: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 272 с.
4. Мейер Э. CSS-каскадные таблицы стилей. Подробное руководство – Пер. с англ. – СПб: Символ-Плюс, 2008. – 576 с.
5. Колин Мук. ActionScript для Flash MX. Подробное руководство – Пер. с англ. – СПб: Символ-Плюс, 2004. – 1120 с.
6. Бурлаков М. В. CorelDRAW 12. – СПб.: БХВ-Петербург, 2004. – 688 с.
7. Топорков С. С. Adobe Photoshop CS в примерах. — СПб.: БХВ – Петербург, 2005. – 384 с.
8. Мержевич В.В. HTML и CSS на примерах. — СПб.: БХВ – Петербург, 2005. — 448 с.
9. Нильсен Я. Веб-дизайн: книга Якоба Нильсена – Пер. с англ. – СПб: Символ-Плюс, 2003. – 512 с.
10. Ши Д., Хольцшлаг М. Е. Философия CSS-дизайна – Пер. с англ. А.А. Слинкина. – М. : НТ Пресс, 2005. – 312 с.
11. Дунаев В. Самоучитель DreamWeaver MX 2004. – СПб.: Питер, 2005. – 331 с.
12. Блинов И.Н., Романчик В.С. Java. Промышленное программирование: практ. пособие. – Мн: УниверсалПресс, 2007. – 702 с.
13. Романчик В.С., Люлькин А.Е. С++. Лабораторные работы по курсу «Методы программирования»: учеб.-метод. пособие для студентов мех.-мат. фак. – Мн.: БГУ, 2006.