«Қостанай жоғары политехникалық колледжі» КМҚК
КГКП «Костанайский политехнический высший колледж»

Учебно методический комплекс

ПМ05 «СОЗДАНИЕ WEB-СТРАНИЦ, САЙТОВ С ПРИМЕНЕНИЕМ WEB-ТЕХНОЛОГИЙ»

СОДЕРЖАНИЕ

1 ТЕОРЕТИЧЕСКИЕ ЗАНЯТИЯ

РАЗДЕЛ 1. ОСНОВЫ ТЕХНОЛОГИЙ ИНТЕРНЕТ-ПРОГРАММИРОВАНИЯ

Лекция 1.1. Организация Web-сайта

Исторический экскурс

Начать наш курс хотелось бы с недавней истории. Для пользователей Интернета 90-х годов Всемирная паутина представлялась набором неподвижных Web-страничек. На этих страницах обычно отображалась табличная информация, например, **курсы акций, расписание движения железнодорожных поездов или результаты очередного эксперимента**. Web-странички тех времен сильно напоминали содержимое толстых телефонных справочников, отпечатанных на дешевой желтой бумаге. Всемирно известный ныне **поисковый сервер Yahoo!** в те времена предоставлял справочную информацию, известную под именем **Yahoo Yellow Pages**.

Web-странички 80-х годов создавались с помощью языка гипертекстовой разметки **(Hyper Text Markup Language, HTML).** Интернет-контенту недоставало динамизма, вследствие чего в **90-е годы** были предприняты попытки "оживить" HTML. Появился динамический DHTML (Dynamic HTML). *Суть идеи заключалась в следующем. Web-странички, как известно, состоят из так называемых тэгов (tags, например,* ***<IMG SRC= "1.JPG" WIDTH=300>).*** *Тэги являются командами языка HTML, которые визуализируются с помощью браузера клиента (например, с помощью MS Internet Explorer). С помощью тэгов формируются элементы управления, образующие интерфейс программы, например, кнопки, меню и так далее.* В случае с динамическим HTML'ом Web-странички начали сопровождать разделом сценариев (**<SCRIPT>**), написанном на одном из скриптовых языков, например, на **VBScript или JavaScript**. В разделе сценариев создавались подпрограммы-обработчики событий, связанные с тем или иным элементом управления, например, с кнопкой. В результате, Web-странички "ожили" и "заиграли" всеми цветами радуги. У такого подхода вскоре выявились серьезные недостатки. Суть проблемы заключалась в том, что при загрузке Web-странички на клиентскую машину закачивались программы, о назначении которых пользователь не имел ни малейшего представления. При этом приходилось уповать лишь на порядочность и квалификацию разработчика данного сайта.

Проблема загружаемого на клиентскую машину ПО не потеряла актуальности и в наши дни. Простенькие сценарии, изменяющие цвет кнопки превратились в изощренные компоненты **ActiveX** фирмы **Microsoft и Java Applet'ы компании Sun Microsystems**. Для решения проблем безопасности **Sun и Microsoft** пошли разными путями. **Microsoft** решила снабжать компоненты **ActiveX** цифровыми сертификатами и уведомлять пользователя об установке таких программ. Sun решила для запуска апплетов использовать небольшую часть оперативной памяти клиентской машины, названную "песочница" с тем, чтобы сбои в работе апплетов не влияли на другие программы.

В настоящее время переживает настоящий бум так называемое spyware или шпионское ПО, которое тайком устанавливается на клиентской машине, после чего начинает передавать своему хозяину конфиденциальные данные пользователя.

Проблема настолько актуальна, что в американском верховном суде рассматривается вопрос о запрете загружаемого ПО, с чем категорически несогласны производители антивирусов, считающие, что их программы каким-то образом должны обновлять свои антивирусные базы…

В девяностых годах Интернет неожиданно приобрел взрывную популярность, людям требовалось все больше и больше доступных данных. Для хранения и поиска информации в настоящее время широко используются реляционные СУБД, такие как MS SQL Server, IBM DB2 или Oracle. Возникла острая необходимость в технологии, позволяющей Интернет-пользователям извлекать информацию из реляционных СУБД.

Вскоре такая технология появилась. Ее назвали "**Активные серверные страницы", ASP.NET (Active Server Pages).** Рассмотрим принцип ее работы на примере Web-сайта некоего автосалона. Предположим, что информация о представленных в салоне автомобилях хранится в базе таблице CAR_TBL базы данных MyCar.mdf. В таблице существуют три столбца - Title, Photo и Description. Некий пользователь решил приобрести автомобиль, предварительно ознакомившись с его техническими характеристиками. Далее, ситуация будет развиваться по следующему сценарию. На Web-сервере автосалона размещена ASP.NET-страничка index.aspx. На этой Web-страничке имеется раскрывающийся список drpCar, отображающий названия автомобилей. У раскрывающего списка есть свойство DataSource - (источник данных), которое можно связать со столбцом таблицы. Иными словами, чтобы заполнить раскрывающийся список drpCar, нужно выполнить SQL-ный запрос:

```
SELECT Title FROM CAR_TBL
```

Пользователь, загрузив страничку index.aspx, видит на своем экране раскрывающийся список с названиями автомобилей. Предположим, что его заинтересовал Mercedes Benz CL 600. Он выбирает данную модель из раскрывающегося списка, для того, чтобы ознакомиться с техническими характеристиками машины. При этом будет выполнен примерно следующий SQL'ный запрос:

```
SELECT Title, Photo, Description FROM CAR_TBL
WHERE Title='Mercedes Benz CL 600'
```

Через несколько секунд пользователь видит у себя на экране название, фотографию и описание машины.

Как же работают ASP.NET-странички? Вначале они получают запрос от клиента по протоколу HTTP. Запросом может быть, например, нажатие на кнопку, выбор элемента из раскрывающегося списка и т.д. С элементами управления программист связывает особые подпрограммы - обработчики событий. Результатом работы обработчиков событий является HTML-код, посылаемый на компьютер клиента и визуализируемый его браузером. Что очень важно, на клиентскую машину не посылается потенциально опасный код, только инструкции языка HTML. Если дело касается извлечения информации из СУБД, обработчики событий генерируют SQL-запрос, отправляемый на сервер баз данных (рис. 1.1). Механизм баз данных обрабатывает поступивший SQL-запрос и возвращает набор строк (обычно по протоколу XML). Далее, обработчик событий ASP.NET-странички преобразует

полученный набор строк в инструкции языка гипертекстовой разметки и отсылает их клиенту. При этом код ASP.NET-странички, отличается от того, что можно увидеть, нажав в браузере кнопки "Вид" \to "Просмотр HTML-кода" как небо и Земля [1] (рис. 1.1). По сути, Нейману удалось обобщить научные разработки и открытия многих других ученых и сформулировать на их основе принципиально новое.

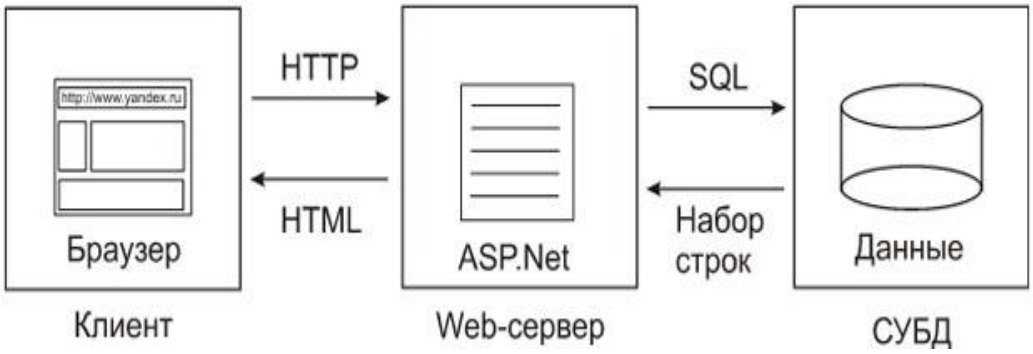


Рис. 1.1

В случаях, когда основная вычислительная нагрузка ложится на web-сервер, говорят, что пользовательское приложение является "тонким клиентом". Если вычисления преобладают на пользовательской машине, мы имеем дело с "толстым клиентом". Как уже упоминалось выше, первым web-страничкам очень недоставало динамизма. В этом они сильно уступали настольным приложениям. В рамках данного курса мы познакомимся со многими инновациями Microsoft, придающими клиентским приложениям интерактивность и зрелищность. Сюда, прежде всего, можно отнести технологию Silverlight, RIA-приложения, средства визуального поиска и многое другое.

Напомним слушателям некоторые положения, которые необходимо знать при создании web-сайтов.

Протокол TCP/IP

Информация в компьютерных сетях передается не непрерывно, подобно воде в трубопроводе, а порциями, или пакетами. Способ упорядочивания информации в пакеты называется сетевой протокол. Самым распространенным сетевым протоколом, используемым и в локальных сетях, и в Интернете, является TCP/IP (Transport control protocol/Internet protocol). В институте открытых стандартов (Open Standards Institute, OSI) была разработана семиуровневая модель передачи данных, применимая, в частности, и для TCP/IP. Для рядовых пользователей представляет интерес самый верхний уровень TCP/IP, или уровень прикладных программ. К этому уровню относятся такие полезные технологии, как протокол передачи гипертекста (Hyper Text Transfer Protocol, http), протокол передачи файлов (File Transfer Protocol, ftp) и протокол передачи простых почтовых сообщений (Simple Mail Transfer Protocol, smtp). С помощью http осуществляется навигация в Интернет, благодаря ftp мы можем скачивать большие файлы, а smtp используется при работе с электронной почтой.

IP-адреса

У всех устройств, подключенных к компьютерной сети, есть некий идентификатор, именуемый ip-адресом. Существует две разновидности ip-адресов: общедоступные и частные. Общедоступными ip-адресами обычно наделяются web-серверы, которые должны быть доступны круглосуточно. Частные ip-адреса

назначаются компьютерам, входящим в состав локальных сетей. По соображениям безопасности они должны быть "невидимы" извне, то есть из Интернета. В момент выхода во Всемирную паутину таким компьютерам назначаются общедоступные ip-адреса.

IP-адреса в четвертой версии протокола IP (IPv4) являются 32-х битными двоичными числами, которые можно представить в виде совокупности четырех групп по три цифры, разделенных точками в диапазоне от 0 до 255, например, 192.168.0.1. Один ip-адрес занимает 4 байта, или 32 бита. У такого подхода есть недостаток - потенциально не очень большая емкость - 2^{32} адресов. Постепенно на смену 4-й версии протокола IP приходит более новая, шестая версия (IPv6), поддерживающая гораздо больший диапазон адресов. В этом случае IP адрес имеет 128-битовое представление. Адреса разделяются двоеточиями (напр. fe80:0:0:0:200:f8ff:fe21:67cf) [2].

Служба DNS

Как уже говорилось выше, web-серверам назначаются двоичные ip-адреса. Людям гораздо удобнее запоминать символьные обозначения ресурсов, например, http://www.mail.ru. Для решения этой проблемы была создана служба разрешения доменных имен (Domain Name Services, DNS). Она работает подобно записной книжке сотового телефона, где каждому телефонному номеру соответствует словесное описание абонента. Если очень упростить ситуацию, можно представить службу DNS в виде таблицы с двумя столбцами. В одном столбце находятся ip-адреса web-серверов, которым соответствуют символьные обозначения во втором столбце. В развитых странах есть организации, поддерживающие базы данных DNS, они называются сетевыми информационными центрами. В нашей стране роль такого сетевого информационного центра играет компания РОСНИИРОС (Российский научно-исследовательский институт развития образовательных сетей). Когда пользователь набирает в адресной строке браузера символьное название нужного ресурса, он вначале соединяется с сетевым информационным центром. Если он набрал адрес с ошибкой, или такого ресурса нет в базе данных DNS, то выдается сообщение об ошибке. Если все нормально, то пользователь перенаправляется на ip-адрес соответствующего web-сервера.

Советы начинающему web-мастеру

Для того чтобы пользователи Всемирной паутины смогли посетить ваш сайт, нужно выполнить следующие действия.

Регистрация домена. Прежде всего, нужно выбрать название будущего сайта, например, http://www.grisha.ru. Для этого нужно зайти на сайт компании РОСНИИРОС (http://www.nic.ru) и узнать, не занято ли данное доменное имя. Предположим, оно оказалось занято. В этом случае можно или подыскать свободный домен, например, недавно появился новый домен .РФ, или изменить название, например, grigory вместо grisha. В конце концов, вы найдете удобный для вас вариант названия сайта. После этого нужно заключить договор и заплатить некоторую сумму. Например, регистрация домена в зоне .ru стоит 600 рублей в год, в зоне .рф - 1200 рублей и так далее.

Делегирование домена. После регистрации домена нужно выбрать провайдера и приобрести у него два ip-адреса: основной и резервный. После этого снова зайти на сайт http://www.nic.ru и указать эти ip-адреса напротив вашего доменного имени.

Хостинг. Под хостингом подразумевается физическое размещение вашего сайта на компьютере, подключенном к Интернет. Существует несколько схем хостинга в зависимости от вашего бюджета и квалификации.

а) Самый бескомпромиссный. Вы приобретаете компьютер, подключаете его к Интернету, устанавливаете серверную операционную систему, настраиваете службу, запускающую web-сервер, размещаете свой web-сайт, настраиваете сетевые подключения с провайдером и обслуживаете свое детище.

б) Вы приобретаете компьютер, устанавливаете на него необходимое ПО и приносите его к провайдеру. Специалисты провайдера устанавливают ваш компьютер в серверную стойку и подключают его к интернету. Они же и обслуживают ваш сайт, хотя это можно делать и дистанционно. С вас будет взиматься небольшая ежемесячная плата.

в) Вы записываете свой сайт на какой-нибудь носитель и приходите с ним к провайдеру. Специалисты провайдера разместят ваш сайт на своем web-сервере. С вас будет взиматься небольшая плата за аренду и обслуживание.

г) Бесплатный вариант. Вы можете воспользоваться одним из многочисленных серверов, предоставляющих услуги бесплатного хостинга, например, http://www.narod.ru. К недостаткам можно отнести более длинное доменное имя, уже не http://www.grisha.ru, а http://www.grisha.narod.ru и некоторые технические ограничения, например, объем дискового пространства, возможность использовать скрипты, необходимость мириться с наличием чужой рекламы на вашем сайте и так далее.

Краткие итоги

Web-технологии за последние 15 лет достигли своей зрелости. Они приняты бизнесом и проникли в каждый дом. Основным способом хранения информации во Всемирной паутине являются web-страницы. Первые web-страницы создавались с помощью статического HTML и напоминали страницы из телефонного справочника. Пользователям требовался более динамичный и красочный контент. В середине 90-х годов прошлого века появились скриптовые языки, сделавшие web-страницы интерактивными. В это же время обострилась война браузеров, победителем из которой вышел Microsoft Internet Explorer. Для навигации в Интернет используется протокол TCP/IP. У всех устройств, подключенных к компьютерной сети есть ip-адреса. Различают общедоступные и частные ip-адреса. При регистрации web-серверов используется база данных DNS. Служба разрешения доменных имен помогает, также, пользователям соединяться с нужным ресурсом в Сети. Существует несколько вариантов хостинга, отличающихся стоимостью и предоставляемыми сервисами.

Лекция 1.2 Технологии создания гипертекстовых документов

В 1945 г. Ваневар Буш - научный советник президента США Г. Трумена, проанализировал способы представления информации в **виде отчетов, докладов, проектов, графиков, планов** и осознав неэффективность такого представления, предложил способ размещения информации **по принципу ассоциативного мышления**. На основе этого принципа была разработана **модель гипотетической машины "МЕМЕКС"** - *машины, которая не только хранила бы информацию, но и связывала между собой имеющие друг к другу отношение текст и картинки*. "МЕМЕКС" так и остался в проекте, но **через 20 лет Теодор Нельсон** реализовал этот принцип на ЭВМ и назвал **его гипертекстом**. Под влиянием идей Буша Теодор Нельсон создал компьютерный язык, который давал возможность пользователю переходить от одного источника информации к другому через электронные ссылки.

Гипертекст обладает нелинейной сетевой формой организации материала, разделенного на фрагменты, для каждого из которых указан переход к другим фрагментам по определенным типам связей. При установлении связей можно опираться на разные основания (ключи), но в любом случае речь идет о смысловой, семантической близости связываемых фрагментов. Следуя по ключу, можно получить более подробные или сжатые сведения об изучаемом объекте, можно читать весь текст или осваивать материал, пропуская известные подробности. Текст теряет свою замкнутость, становится принципиально открытым, в него можно вставлять новые фрагменты, указывая для них связи с имеющимися фрагментами.

Фактически гипертекст - *это технология работы с текстовыми данными, позволяющая устанавливать ассоциативные связи типа гиперсвязей или гиперссылок между фрагментами, статьями и графикой в текстовых массивах.*

Благодаря этому становится доступной не только последовательная, линейная работа с текстом, как при обычном чтении, но и произвольный ассоциативный просмотр в соответствии с установленной структурой связей, а также с учетом личного опыта, интересов и настроения пользователей. Гипертекстовый документ таким образом получает дополнительные измерения. С одной стороны, он подобен обычному текстовому документу, имеющему фиксированное начало и конец. С другой стороны, гипертекст одновременно организован по тематическим линиям, по индексам и библиографическим **указателям.**

Структурно гипертекст состоит из следующих элементов, представленных на рис. 2.1

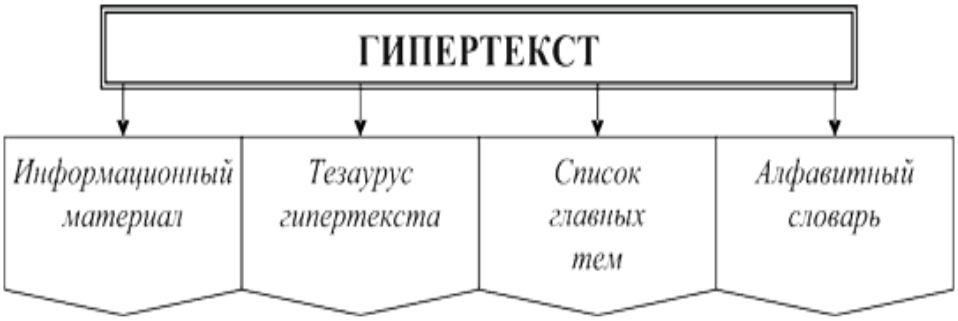


Рис. 2.1

Информационный материал подразделяется на информационные статьи, состоящие из заголовка статьи и текста. Информационная статья может представлять собой файл, закладку в тексте, web-страницу. **Заголовок -** *это название темы или наименование описываемого в информационной статье понятия*. Текст информационной статьи содержит традиционные определения и понятия, т. е. описание темы. Текст, включаемый в информационную статью, может сопровождаться пояснениями, числовыми и табличными примерами, графиками, документами и видеоизображениями объектов реального мира.

В тексте информационной статьи выделяют ключи или гиперссылки, являющиеся заголовками связанных статей, в которых может быть дано определение, разъяснение или обобщение выделенного понятия. Гиперссылкой может служить слово или предложение. Гиперссылки визуально отличаются от остального текста путем подсветки, выделения, оформления другим шрифтом или цветом и т. д. Они обеспечивают ассоциативную, семантическую, смысловую связь или отношения между информационными статьями.

Все гиперссылки можно разделить на две категории:
локальные гиперссылки;
глобальные гиперссылки.

Локальные гиперссылки - это ссылки на другие части того самого документа, откуда они осуществляются.

Примерами локальных гиперссылок являются:

- **ссылки из содержания на главы текста;**
- **ссылки из одной главы текста на другую главу;**
- **ссылки от какого-либо термина на его определение, расположенное в словаре терминов данного текста и т. п.**
-

Пример локальной гиперссылки приведен на рис. 2.2

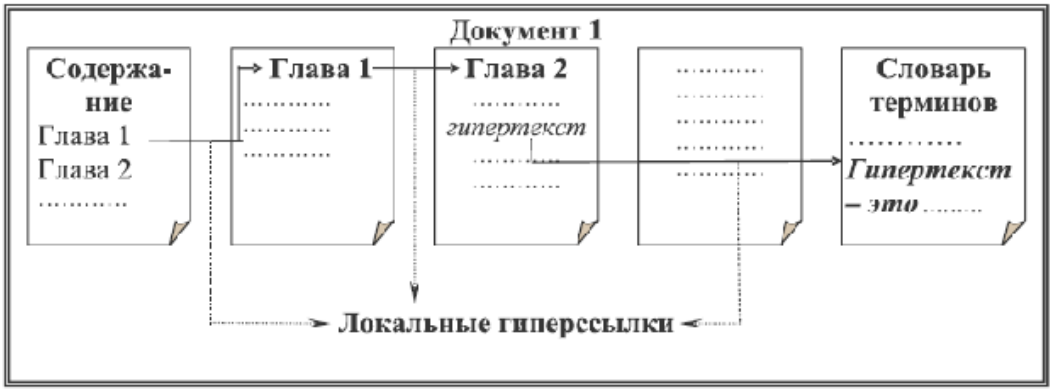


Рис. 2.2

Локальные гиперссылки практически всегда выполнимы, т. е. выполнение данной ссылки приводит к появлению той части документа, куда должен осуществляться переход по ссылке.

Глобальные гиперссылки - ***это ссылки на другие документы, в общем случае на какие-либо ресурсы, расположенные вне данного документа.***

Примерами глобальных ссылок являются:

- **ссылки на другой файл, логически не связанный с документом и существующий независимо от него;**
- **ссылки на страницу удаленного Web-сервера.**
-

Примеры глобальных гиперссылок приведены на рис. 2.3

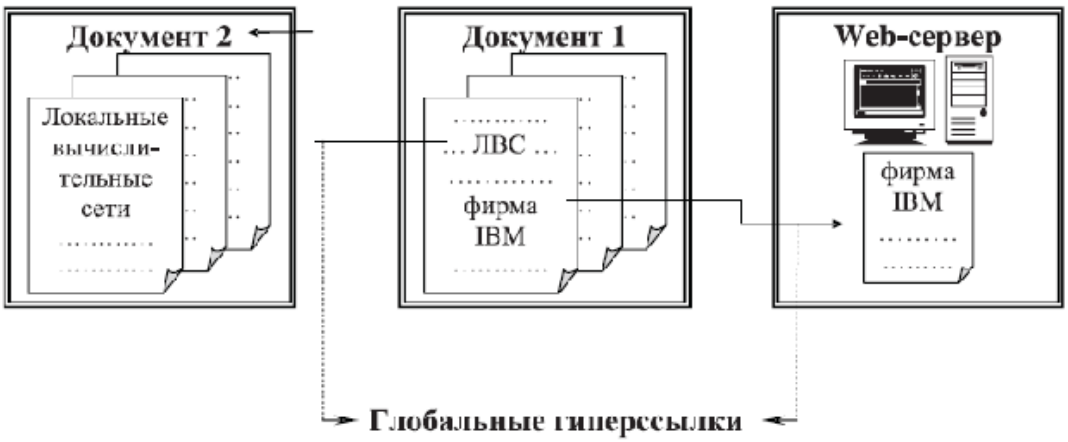


Рис. 2.3

Для глобальных гиперссылок возможны случаи, когда требуемый ресурс, на который производится ссылка, по тем или иным причинам отсутствует. Например, файл, на который следует перейти по ссылке, удален или устарела страница Web-сервера**.**

Тезаурус гипертекста - *это автоматизированный словарь, отображающий семантические отношения между информационными статьями и предназначенный для поиска слов по их смысловому содержанию***.** Термин **"тезаурус"** был введен в XIII в. флорентийцем Брунетто Лотики для названия энциклопедии. С греческого языка этот термин переводится как **"сокровище, запас, богатство".**

Тезаурус гипертекста состоит из тезаурусных статей, каждая из которых имеет заголовок и список заголовков родственных тезаурусных статей. Заголовок тезаурусной статьи совпадает с заголовком информационной статьи и содержит данные о типах отношений с другими информационными статьями. Тип отношений определяет наличие или отсутствие смысловой связи**.**

Существует два типа отношений информационных статей:

- **референтные отношения;**
- **организационные отношения.**

Референтные отношения указывают на смысловую, семантическую, ассоциативную связь двух информационных статей. В информационной статье, на которую сделана ссылка, может быть дано определение, разъяснение, понятие, обобщение, детализация понятия, выделенного в качестве гиперссылки. **Референтные отношения** образуют связь типа**: род - вид, вид - род, целое - часть, часть - целое.**

Пользователь получает более общую информацию по родовому типу связи, а по видовому - более детальную информацию без повторения общих сведений из родовых тем. Примеры референтных отношений информационных статей приведены на рис. 2.4

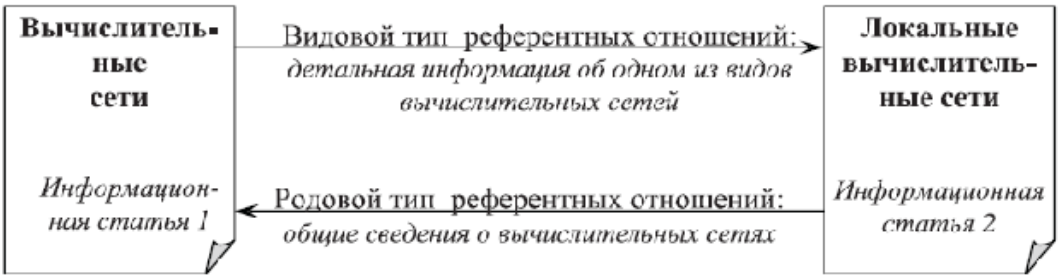


Рис. 2.4

К организационным отношениям относятся те, для которых нет ссылок с отношениями род - вид, целое - часть, т. е. между информационными отношениями нет смысловых связей. Они позволяют создать список главных тем, оглавление, меню, алфавитный словарь.

Пример организационных отношений приведен на рис. 2.5

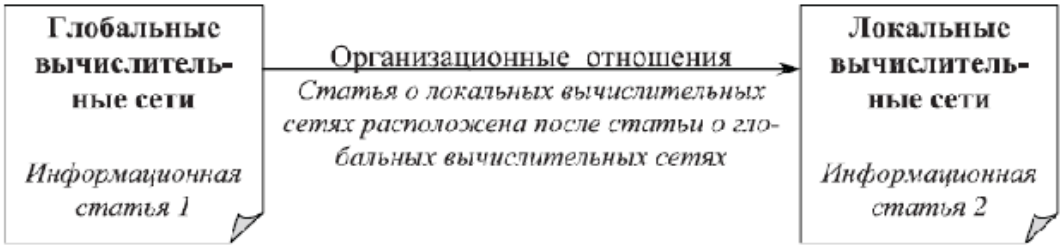


Рис. 2.5

Список главных тем содержит **Алфавитный словарь** содержит заголовки информационных статей сперечень наименований всех организационными отношениями. Обычно онинформационных статей в алфавитном представляет собой меню, содержание книги,порядке. Он также реализует отчета или информационного материала. организационные отношения.

Изучая информацию, представленную в виде гипертекста, пользователь может знакомиться с информационными фрагментами гипертекста в произвольном порядке. Процесс перемещения пользователя по информационным фрагментам называется навигацией.

В зависимости от признака классификации можно выделить следующие виды навигации, представленные на рис. 2.6:

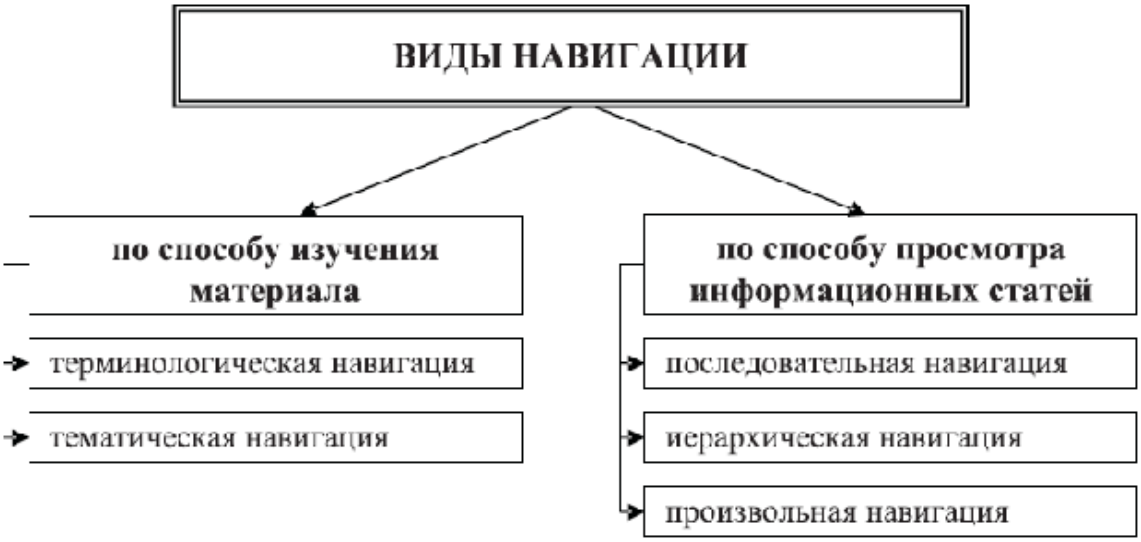


Рис. 2.6

По способу изучения материала выделяют:

- **терминологическую навигацию** - последовательное движение пользователя по терминам, вытекающим друг из друга;
- **тематическую навигацию** - получение пользователем всех статей, необходимых для изучения выбранной темы.

По способу просмотра информационных статей различают:

- **последовательную навигацию** - просмотр информации в порядке расположения ее в гипертекстовом документе, т. е. в естественном порядке;
- **иерархическую навигацию** - просмотр информационных статей, характеризующих общие понятия по выбранной теме, затем переход к информационным статьям, детализирующим общие понятия и т. д. ;
- **произвольную навигацию** - произвольное перемещение по ссылкам гипертекстового документа, порядок которого определяется личным опытом, интересами и настроением пользователя.

Переход пользователя от одной информационной статьи к другой может быть постоянным или временным.

Постоянный переход. Пользователь имеет возможность ознакомиться с новымимеет возможность ознакомиться с информационным фрагментом, а затемпримечанием, пояснением, толкованием выбрать следующую информационнуютермина, а затем обязательно должен статью для перехода без возврата квернуться к первоначальному первоначальному фрагменту информационному фрагменту

Временный переход. Пользователь

Гипертекстовые технологии реализуют следующие сервисные инструменты пользователя, представленные в табл. 2.1.

Название инструмента	Описание возможностей
Откат	Возврат к ранее рассмотренным фрагментам. Возможны два варианта реализации этого инструмента: быстрый переход к последнему в цепочке просмотренных фрагменту; предоставление пользователю заголовков всех просмотренных им фрагментов с возможностью выбора того уровня, к просмотру которого необходимо вернуться
Список всех фрагментов гипертекстового документа	Список организован в виде иерархической системы, на верхнем уровне которой содержатся только общие формулировки тем фрагментов, а при снижении от уровня к уровню в существующей иерархии достигается все большая детализация и конкретизация
Поисковый механизм	Позволяет искать в сети гипертекстового издания фрагменты (статьи), содержащие необходимую информацию по ключевым словам и (или) атрибутам фрагментов. В качестве атрибутов могут выступать, например, название, имя автора, стоимость фрагмента, дата его последней модификации и т. п.
Книжная закладка	Предоставляет пользователю возможность отмечать интересующий его информационный фрагмент
Записная книжка	Обеспечивает пользователю реализацию функции сохранения произвольных комментариев к просмотренным фрагментам и возможность их последующего изменения
Средства сохранения состояния системы	Возможность продолжения работы с гипертекстовым документом после вынужденного или запланированного перерыва, начиная с того места, на котором работа с изданием была прервана

Лекция 1.3 Понятие сервера

Чтобы ответить на этот вопрос, нужно понять, откуда происходит слово «сервер». В его основе — английский глагол to serve, который переводится как «служить». Итак, server, говоря простыми словами — это специальный компьютер, служащий тому, чтобы та или иная информационная сеть (например, локальная) могла полноценно функционировать.

С термином «сервер» неразрывно связан другой — «клиент». Так называется персональный компьютер, мобильное или другое устройство, находящееся в одной сети с сервером, направляющее на него те или иные запросы и получающее от него необходимую информацию. В существовании сервера не было бы никакого смысла без связанных с ним клиентских устройств.

Для чего предназначен сервер

Эта машина в зависимости от её типа и назначения может выполнять разные полезные функции:

хранить информацию одного или нескольких сайтов. Так работают серверы интернет-провайдеров, оказывающих услуги хостинга;

координировать взаимодействие множества компьютеров, находящихся в одной сети. Классический пример — игровые серверы;

использоваться для хранения корпоративных данных и доступа сотрудников к ним. Один из возможных примеров — сервер в центральном офисе компании, на котором хранится и регулярно обновляется бухгалтерская база данных.

Это — лишь некоторые из распространённых способов использования серверов. Ниже мы подробно рассмотрим существующие виды этих машин и более детально раскроем вопрос их применения.

Из чего состоит сервер

Любой сервер вне зависимости от его типа и назначения представляет собой компьютер. Именно поэтому он имеет на борту классические компьютерные комплектующие:

материнскую плату, играющую роль основы системы;

один или несколько центральных процессоров;

определённый объём оперативной памяти;

систему хранения данных, состоящую из накопителей того или иного типа.

«Железо» сервера при принципиальной схожести с комплектующими обычного персонального компьютера имеет и свою специфику. Она связана с тем, что основные задачи серверного оборудования — хранение, интенсивная обработка и быстрая передача больших объёмов данных. Чтобы они успешно выполнялись, серверы оснащают:

высокопроизводительными многоядерными процессорами;

большим объёмом оперативной памяти с контролем ошибок;

ёмкими и скоростными жёсткими дисками и твердотельными накопителями.

Человек взаимодействует с сервером минимальное количество времени — лишь на этапах его настройки, обслуживания и ремонта. Отпадает необходимость вывода на монитор качественной 3D-графики, и с этим связана ещё одна аппаратная особенность серверов — как правило, в них используются видеокарты начального уровня.

Интенсивная обработка данных влечёт сильный нагрев серверных комплектующих. Для нейтрализации этого негативного эффекта в состав серверов включают мощные системы охлаждения.

Прямые обязанности сервера — приём и передача данных. Учитывая это, производители и сборщики оснащают серверные машины высокопроизводительными сетевыми картами. Если речь идёт о сервере с доступом через всемирную сеть, в компаниях и дата-центрах обеспечивают широкополосное интернет-подключение оборудования.

В прошлом серверы часто собирались в вертикальных корпусах, похожих на те, что используются для обычных компьютеров. Позже общепринятым стандартом стало стоечное исполнение. Современные серверные комплектующие монтируются в широких плоских блоках, которые, в свою очередь, в горизонтальном положении устанавливаются в серверный шкаф. Такое исполнение позволяет максимально компактно разместить оборудование в ограниченном объёме пространства.

Чтобы вся сеть функционировала стабильно, нужно обеспечить непрерывное электроснабжение сервера. Для этого в большинстве случаев используются источники бесперебойного питания

Важно защитить серверы от больших перепадов температуры и влажности, затопления, пожара и других форс-мажорных ситуаций. Для этого в дата-центрах и крупных компаниях оборудуют специальные серверные комнаты, которые заслуживают отдельного обзора.

Особое внимание стоит уделить серверному программному обеспечению, под управлением которого работает перечисленное выше «железо». Обычные операционные системы — та же Windows, известная, пожалуй, каждому — в данном случае подходят плохо или не подходят вовсе. Используются серверные версии ОС — как разработанные Microsoft, так и входящие в семейство Linux. Последние, к слову, получили максимально широкое распространение, поскольку обеспечивают стабильность работы серверов, безопасность данных и гибкость конфигурирования. Работа с серверной ОС семейства Linux (например, Ubuntu Server) требует специальных познаний, поэтому дата-центры и компании нанимают на работу квалифицированных IT-специалистов.

Если речь идёт о небольшом сервере, от которого не требуется особой функциональности и вычислительной мощности, может подойти и обычная операционная система. Так, возможностей Windows более чем достаточно для того, чтобы настроить FTP и организовать работу файлового сервера, доступного ограниченному кругу лиц.

Виды и типы серверов

Многообразие задач, которые ставятся перед серверами, обусловило большое количество видов и типов этого оборудования. IT-специалисту стоит ориентироваться в существующих разновидностях и понимать назначение и особенности каждой из них.

Web-сервер

Это — пожалуй, наиболее распространённая разновидность, с которой имеет дело каждый пользователь Интернета. На web-серверах хранится текстовый, графический, видео- и другой контент, из которого состоят интернет-сайты. Посетитель отправляет запрос, используя для этого браузер персонального компьютера или мобильного устройства, играющего роль клиента. Web-сервер даёт ответ в формате HTTP и отправляет клиенту данные. В результате посетитель видит на экране интересующий его сайт, переходит по страницам, отправляет через формы данные — словом, взаимодействует с web-сервером.

Игровой сервер

Эта машина обеспечивает взаимодействие игроков, запускающих одну и ту же игру в режиме мультиплеера и одновременно находящихся в виртуальном мире. Геймерам хорошо известны такие названия, как World of Tanks, Counter Strike, DotA, World of Warcraft и многие другие. Во всех случаях речь идёт именно об игровых серверах, причём в случае с известными играми они являются весьма и весьма мощными, ведь им приходится выдерживать немалые нагрузки.

Видеосервер

Как легко понять по его названию, он используется для хранения видеороликов — фильмов, клипов и многих других. Пользователь, обращаясь к видеосерверу со своего устройства, получает возможность смотреть видео, не скачивая его и не расходуя собственное дисковое пространство. При упоминании видеосерверов в первую очередь вспоминается крупнейший ресурс, известный, пожалуй, каждому — YouTube.

Сервер локальной сети

Так называется машина, к которой организован ограниченный доступ — например, внутри корпоративной сети, развёрнутой на предприятии. Благодаря её наличию сотрудники, находясь на разных (и нередко весьма удалённых) рабочих местах, могут одновременно использовать информацию — например, бухгалтерскую базу данных. Такой сервер позволяет общаться по рабочим вопросам, отслеживать выполнение поручений, решать многие другие задачи.

Почтовый сервер

Применяется для хранения электронной почты, пересылки писем, фильтрации спама, сортировки электронных писем по категориям, решения других задач, связанных с использованием e-mail. В числе наиболее известных сервисов, позволяющих воспользоваться почтовыми серверами — Mail, Yandex, Gmail, ряд других. Такую возможность дают и хостинг-провайдеры, которые создают электронные ящики на персональных доменах пользователей.

FTP-сервер

Используется для хранения файлов и удалённого доступа к ним по FTP — File Transfer Protocol. В зависимости от назначения и масштаба сервера доступ может быть реализован как через Интернет, так и по локальной сети. Если в случае с web-сервером пользователи работают в браузерах, то в данном случае удобнее всего применять специализированные программы для передачи файлов — например, Filezilla.

DNS-сервер

Что такое адрес сервера в сети, то есть его IP-адрес? Это — набор из нескольких групп цифр, разделённых точками. Зная его, можно открыть сайт, скачать файлы, решить другие задачи, связанные с доступом к серверу. Есть проблемы: набор цифр гораздо сложнее запомнить, к тому же он изменится при переезде сайта на другой хостинг. Они решаются с помощью доменных имён — проще говоря, привычных каждому адресов веб-сайтов. Связь между IP-адресами и доменными именами обеспечивают DNS-серверы. Они нужны для автоматического определения упомянутых выше наборов цифр при вводе пользователями адресов сайтов.

VPN-сервер

Это оборудование обеспечивает работу виртуальной сети, которая позволяет зашифровать и защитить персональные данные пользователей. Последние могут пользоваться общедоступным каналом связи, то есть Интернетом, однако благодаря VPN-серверу оставаться при этом внутри защищённой частной сети.

Прокси-сервер

Одна из функций этого серверного оборудования — кэширование (сохранение на локальном диске) информации, полученной из Интернета. При повторном обращении прокси-сервер отдаёт сохранённые данные пользователю, делая ненужным очередной выход во всемирную сеть и экономя трафик.

Лекция 1.4 Установка сервера

Установка

Open Server является портативным программным комплексом и не требует установки. Сборку можно разместить на внешнем жёстком диске, это позволит использовать Open Server на любом компьютере, который отвечает системным требованиям.

Не стоит размещать Open Server на USB-флеш накопителе ввиду крайне медленной работы флеш-памяти при параллельных запросах на чтение/запись и её быстрого износа.

Дистрибутив представляет собой самораспаковывающийся архив в формате RAR (расширение .exe). Запустите исполняемый файл дистрибутива и выберите путь для распаковки файлов.

Системные требования

Поддерживаемые версии ОС: 64-бит Windows 7 SP1 или новее (32-битные системы не поддерживаются);

Минимальные аппаратные требования: 500 МБ свободной RAM и 4 ГБ свободного места на HDD;

Требуется наличие Microsoft Visual C++ 2005-2008-2010-2012-2013-2015-2019 Redistributable Package;

Первый запуск

Перед началом использования Open Server выполните [Меню → Дополнительно → Первый запуск]. Наличие установленного набора библиотек от Microsoft является обязательным системным требованием, без их наличия Open Server работать не будет.

Установку/переустановку необходимо выполнить даже в том вслучае, если вы уже делали это ранее для предыдущей версии. Каждый раз библиотеки обновляются, в то время как установленные ранее библиотеки могут быть несовместимы с новыми версиями модулей. Делайте это для каждой новой версии Open Server.

Запуск

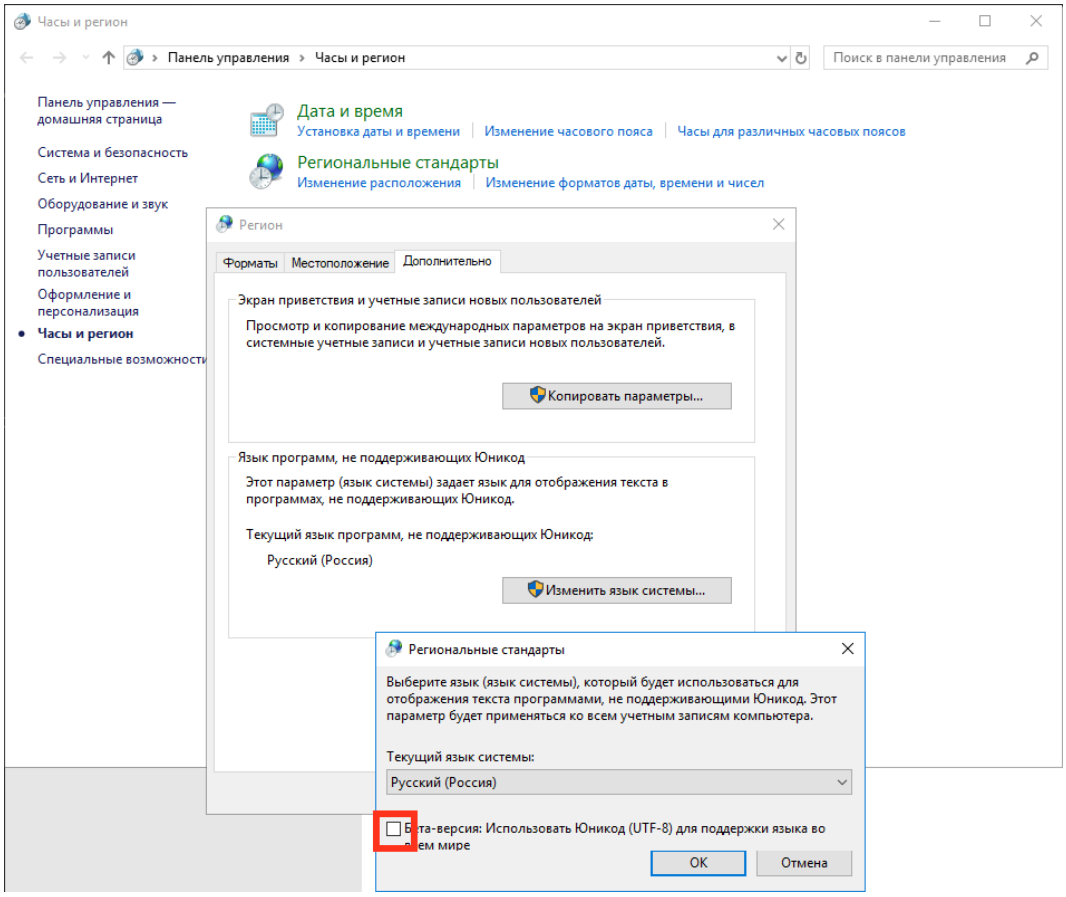
Для запуска Open Server используйте файл Open Server.exe. После старта программы вы увидите красный флажок в области уведомлений Windows (область возле системных часов). Чтобы включить непосредственно сам веб-сервер и сопутствующие модули нажмите на флажок, далее выполните [Меню → Запустить].

Если сервер не запускается перейдите к разделу Решение проблем данного справочного руководства.

Настройка Windows 10

При запуске Open Server в системах семейства Windows 10 необходимо предварительно снять галочку, показанную на картинке, и перезагрузить систему.

Без выполнения данного условия запуск Open Server на Windows 10 будет невозможен.



Установка обновлений

Open Server это достаточно сложный программный комплекс с постоянно совершенствующейся архитектурой. Как таковой процедуры обновления не предусмотрено. При выходе новой версии Open Server необходимо заново выполнить все настройки, скопировать папки ваших сайтов и выполнить перенос баз данных.

Не распаковывайте файлы дистрибутива поверх существующей версии, а так же не пытайтесь скопировать файлы конфигурации и профилей из старой версии программы в новую!

Если есть возможность обновиться простым копированием файлов (довольно редкое явление), то ссылка на патч всегда публикуется в новостях на сайте вместе с аннотацией к новой версии.

Подключение

Ниже представлены установленные в Open Server начальные настройки (значения по умолчанию) для подключения к различным модулям. Вы всегда можете самостоятельно изменить эти настройки по своему усмотрению.

Подключение к MySQL

Адрес: домен вашего сайта*
Порт: 3306
Пользователь: mysql
Пароль: mysql

ROOT подключение к MySQL

Пользователь: root

Пароль: root

Информация

Кодировка, установленная по умолчанию в настройках MySQL сервера, не действует на пользователя ROOT. Кодировку нужно будет явно указывать в скриптах подключения к БД, потому использовать пользователя ROOT не рекомендуется.

Предупреждение

В целях безопасности доступ с удалённых хостов (%) по умолчанию отключён и возможен только с локальной машины 127.0.0.1 (localhost). Выполните настройку удалённого доступа до того, как вы запустите Open Server на публичном IP (если планируете).

Подключение к PostgreSQL

Адрес: домен вашего сайта*
Порт: 5432
Пользователь: postgres
Пароль: (пусто)

Подключение к MongoDB

Адрес: домен вашего сайта*
Порт: 27017
Пользователь: (пусто)
Пароль: (пусто)

Подключение к Memcached

Адрес: домен вашего сайта*
Порт: 11211

Макс. размер памяти используемой сервером Memcache по умолчанию равен 64 Мб. Данный параметр можно изменить выполнив [Меню → Настройки → Разное].

Подключение к Redis

Адрес: домен вашего сайта*
Порт: 6379

Подключение к DNS

Адрес: домен вашего сайта*
Порт: 53

Подключение к FTP

Адрес: домен вашего сайта*
Порт: 21 (990 для FTPS)
Пользователь: ftp
Пароль: ftp

Внимание

Модуль FTP-сервера экспериментальный. Он имеет известные уязвимости и проблемы с быстродействием. Используйте его только в целях локальной разработки. Не активируйте модуль в условиях, когда доступ к серверу открыт во внешнюю сеть!

Домен localhost

Если вы хотите использовать привычный адрес localhost для подключения к MySQL, PostgreSQL, FTP или Memcache серверу, то достаточно не удалять стандартный домен localhost или создать одноимённый алиас.

Автоматизация подключения

При работе с локальными копиями действующих веб-проектов часто возникают трудности с постоянным редактированием файлов конфигурации, в основном это касается настроек подключения к базе данных. Чтобы этого избежать рекомендуется:

Локально создать пользователя базы данных с теми же именем, паролем и привилегиями, что используются на удалённом сервере.

Создать алиас с тем же именем, что используется в качестве хоста базы данных на удалённом сервере.

* Например, если ваш скрипт размещен по адресу testserver.loc/mysql.php, то хостом (адресом) для подключения к MySQL, Redis и другим модулям будет домен: testserver.loc

Домены и алиасы

Режимы управления доменами

В Open Server существует три режима управления списком доменов: автопоиск, ручное управление и ручное+автопоиск. По умолчанию используется первый режим автоматического подключения папок из корневой директории указанной в настройках.

Как работает автопоиск

Программа сканирует заданную веб-директорию на наличие папок с доменами, после чего в каждой найденной папке производится поиск подпапок (корневой папки домена) которые указаны в настройках для автосканирования. Если ни одна из предполагаемых корневых подпапок не найдена, то корнем домена становится сама папка с доменом.

Как работает совмещённое управление (ручное + автопоиск)

При использовании совмещенного режима управления доменами программа сначала подключает домены созданные вручную, после чего производится автоматическое сканирование по процедуре описанной выше.

Создание домена в автоматическом режиме

Чтобы создать домен или поддомен откройте [Меню → Папка с сайтами] и создайте папку с именем будущего домена. После создания домена перезапустите сервер.

Создание домена в ручном режиме

Чтобы создать домен или поддомен перейдите в раздел [Меню → Настройки → Домены] и создайте запись вида: домен => папка. В качестве папки домена можно выбрать уже существующую папку на диске или создать её непосредственно в окне выбора каталога. После создания домена сохраните настройки.

Создание кириллического домена

Open Server поддерживает кириллические домены, однако будьте внимательны, папку с доменом нужно называть его реальным именем, а не псевдо названием на кириллице. Для пиво.рф реальным названием (punycode формат) будет xn--b1altb.xn--

p1ai и создав такой домен вы получите доступ к http://пиво.рф. Для конвертации доменных имён в punycode формат и обратно используйте [Меню → Дополнительно → IDN конвертер].

Создание поддомена

Процесс создания поддомена аналогичен процедуре создания обычного домена. При создании только поддомена доступность основного домена существующего в сети Интернет не теряется, т.е. вы сможете работать с локальным поддоменом имея при этом доступ к рабочему домену в сети Интернет.

Создание алиаса

Чтобы создать алиас перейдите в раздел [Меню → Настройки → Алиасы] и создайте запись вида: исходный домен => конечный домен. После создания алиаса сохраните настройки.

Обратите внимание - создание алиаса вида *.xxx.xx не имеет смысла в Windows и не означает то, что вам станут доступны любые поддомены вида test.xxx.xx, mail.xxx.xx и т.д. Необходимо создать конкретный алиас или домен чтобы он стал доступен, это особенность операционной системы Windows.

Иконка сайта в меню доменов

При наличии корректного файла favicon.ico в корневой папке домена иконка сайта будет отображаться в меню программы.

Ограниченный режим работы

В некоторых случаях управление доменами и алиасами недоступно (см. Ограниченный режим).

Работа с MySQL

Создание пользователя MySQL

Откройте [Меню → Дополнительно → PHPMyAdmin]
Введите имя пользователя root и пароль root (по умолчанию)
В PHPMyAdmin откройте раздел [Привилегии]
Нажмите ссылку [Добавить нового пользователя]
Заполните форму и нажмите кнопку [Создать пользователя]

Информация

В том случае, если помимо прочих привилегий для пользователя будет отмечена привилегия SUPER, то кодировка, установленная по умолчанию в настройках MySQL сервера, не будет на него действовать. Кодировку нужно будет указывать в ваших скриптах персонально для каждого подключения к MySQL, потому отмечать привилегию SUPER не рекомендуется.

Создание базы данных MySQL

Откройте [Меню → Дополнительно → PHPMyAdmin]
Введите имя пользователя root и пароль root (по умолчанию)
В PHPMyAdmin откройте раздел [Базы данных]
Введите название новой базы данных и выберите её кодировку
Нажмите кнопку [Создать]

Переключение модулей баз данных

Будьте внимательны при смене активного модуля базы данных. Каждый модуль имеет свое отдельное хранилище баз и настроек, они никак не связаны друг с другом, поэтому вы не увидите созданные вами базы данных при переключении на другой модуль.

Предупреждение

Перед началом использования Open Server выполняется процедура первого запуска [Меню → Дополнительно → Первый запуск], во время которой IPv4 трафик получает приоритет над IPv6 при разрешении доменных имён. Если не выполнять данную процедуру, то процесс подключения в MySQL серверу может выполняться очень долго (более 1 сек).

Чтобы проверить корректность настройки приоритетов откройте консоль и введите ping localhost. Если в ответе вы получаете адрес ::1 вместо 127.0.0.1, значит приоритеты настроены неверно.

Переменная Path

Для добавления собственных путей в переменную окружения PATH можно использовать файл ./userdata/config/path.txt

Пути необходимо добавлять по одному вписывая каждый с новой строки, например:

C:\Windows
D:\My Programs
%realprogdir%\data\dll
C:\Windows\System32

Информация

По умолчанию файл path.txt не подключается. См. [Меню → Настройки → Сервер].

Использование переменных в качестве подстановок

Переменная	Описание переменной
%realprogdir%	Реальный путь до папки с Open Server (обратный слеш "\")
%progdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (обратный слеш "\")
%sprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (слеш "/")
%dprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (двойной обратный слеш "\\")
%dsprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (двойной слеш "//")
%realsitedir%	Реальный путь до корневой папки доменов (обратный слеш "\")
%sitedir%	Генерируемый путь до корневой папки доменов с учетом виртуального диска (обратный слеш "\")
%ssitedir%	Генерируемый путь до корневой папки доменов с учетом виртуального диска (слеш "/")

Переменная	Описание переменной
%httpport%	Порт HTTP сервера
%httpsport%	Порт HTTPS сервера
%postgresqlport%	Порт PostgreSQL сервера
%mysqlport%	Порт MySQL сервера
%mongodbport%	Порт MongoDB сервера
%memcacheport%	Порт Memcache сервера
%ftpport%	Порт FTP сервера
%httpdriver%	Название модуля HTTP
%phpdriver%	Название модуля PHP
%mysql_driver%	Название модуля MySQL/MariaDB
%pg_driver%	Название модуля PostgreSQL
%mongo_driver%	Название модуля MongoDB
%memcachedriver%	Название модуля Memchached
%redisdriver%	Название модуля Redis
%dnsdriver%	Название модуля DNS
%ip%	IP адрес сервера
%disk%	Буква диска из генерируемого пути до папки с Open Server с учетом виртуального диска (только буква)
%osdisk%	Буква диска из реального пути до папки с Open Server (только буква)
%sysdisk%	Системный диск Windows (только буква)

Настройка DNS

Встроенный DNS сервер предназначен для использования в локальных сетях или для отладки веб-приложений. Для детальной настройки доступна общая конфигурация сервера, а так же конфигурация доменов.

Для использования встроенного DNS сервера необходимо выполнить настройку сетевого интерфейса на каждом компьютере ДО запуска самого сервера. Выполнить настройку необходимо как на локальной машине, так и на других компьютерах в локальной сети, которые хотят получить доступ к вашим доменам. Без указания локального DNS сервера в настройках сетевого подключения запуск сервера будет невозможен (если модуль DNS активирован в настройках Open Server).

По умолчанию параметр TTL установлен в значение 60 (секунд), вы можете изменить это значение в файле ./userdata/init.ini однако следует иметь ввиду, что бездумное изменения TTL может спровоцировать кэширование неактуальных записей другими компьютерами в вашей сети. Изменяйте этот параметр только в том случае, если вы действительно понимаете его предназначение.

Предупреждение

Перед началом использования DNS-модуля убедитесь в том, что у вас отсутствуют домены без точки в своём названии. Если таковые имеются, то переименуйте их добавив .virtual или .loc к имени домена.

Внимание

Если ваш браузер производит разрешение доменных имён через прокси-сервер или использует функцию DNS over HTTPS от стороннего DNS-провайдера (например Google или CloudFlare), то локальные домены, созданные вами в Open Server, не будут доступны в этом браузере.

Рекомендации по настройке

В качестве IP адреса сервера выберите в настройках Open Server IP адрес вашего компьютера в локальной сети или сети Интернет (не выбирайте параметр Все доступные IP).

Откройте свойства нужного сетевого подключения:

Центр управления сетями → Подключение xxx → Свойства → Протокол Интернета версии 4 → Свойства → Общие → Использовать следующие адреса DNS-серверов.

Пропишите следующие адреса NS серверов:

xxx.xxx.xxx.xxx
8.8.8.8 (или любой другой реальный резервный DNS)

Вместо xxx.xxx.xxx.xxx впишите IP адрес вашего компьютера в локальной сети или сети Интернет.

Повторите процедуру настройки на других компьютерах в локальной сети.

Не выполняйте настройку алиасов из инструкции раздела Внешний доступ данного руководства. Подобная настройка не требуется, поскольку удалённые компьютеры будут напрямую работать с вашим DNS сервером и получат доступ ко всем доменам.

Сохраните настройки и выполните запуск сервера Open Server.

После правильной настройки все компьютеры в вашей локальной сети смогут получить доступ к доменам Open Server.

Совпадение имён локальных и реальных доменов

Если включён DNS-модуль, то при каждом запуске сервера и при каждой его остановке Open Server выполняет очистку DNS-кэша Windows

Таким образом, если имя локального и реального домена совпадают и если ваш браузер не имеет собственного DNS-кэша, то браузер будет получать контент с реального домена пока сервер не запущен и с локального домена когда сервер запущен. Если ваш браузер имеет встроенный DNS-кэш, то вам необходимо установить расширение для возможности его ручной очистки чтобы иметь удобный доступ к двум доменам попеременно.

Так же вы можете просто иметь второй браузер, который настроен на использование стороннего DNS-провайдера (например Google или CloudFlare) и потому он будет видеть только реальный домен независимо от того, запущен локальный сервер или нет.

Планировщик заданий

Использование планировщика заданий (Cron)

Значения времени вводятся в виде цифр, комбинации цифр или *.

Вы можете задать моменты времени, используя запятые как разделители:

Например: 1,2,3

Результат: задание выполняется 3 раза, в 1-ю,2-ю и 3-ю минуты часа.

Вы можете указать диапазон, используя дефис:

Например: 5-7

Результат: задание выполняется 3 раза, в 5-ю,6-ю и 7-ю минуты часа.

Вы можете задать периодичность выполнения используя звездочку (*) и слэш (/):

Например: */2

Результат: задание выполняется каждые 2 минуты.

Вы можете комбинировать способы для создания точного расписания:

Например: 1,5,11-15,30-59/2

Результат: задание выполняется в 1,5, с 11 по 15 и каждые 2 минуты с 30 по 59.

Вышеприведённая схема указания времени актуальная для всех временных периодов (минута, час, число, месяц, день недели). Для запроса URL в планировщике нужно использовать вспомогательную утилиту Wget (см. примеры). В строке команд можно использовать подстановки.

Использование переменных в качестве подстановок

Переменная	Описание переменной
%realprogdir%	Реальный путь до папки с Open Server (обратный слеш "\")
%progdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (обратный слеш "\")
%sprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (слеш "/")
%dprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (двойной обратный слеш "\\")
%dsprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (двойной слеш "//")
%realsitedir%	Реальный путь до корневой папки доменов (обратный слеш "\")
%sitedir%	Генерируемый путь до корневой папки доменов с учетом виртуального диска (обратный слеш "\")
%ssitedir%	Генерируемый путь до корневой папки доменов с учетом виртуального диска (слеш "/")
%httpport%	Порт HTTP сервера

Переменная	Описание переменной
%httpsport%	Порт HTTPS сервера
%postgresqlport%	Порт PostgreSQL сервера
%mysqlport%	Порт MySQL сервера
%mongodbport%	Порт MongoDB сервера
%memcacheport%	Порт Memcache сервера
%ftpport%	Порт FTP сервера
%httpdriver%	Название модуля HTTP
%phpdriver%	Название модуля PHP
%mysql_driver%	Название модуля MySQL / MariaDB
%pg_driver%	Название модуля PostgreSQL
%mongo_driver%	Название модуля MongoDB
%memcachedriver%	Название модуля Memchache
%redisdriver%	Название модуля Redis
%dnsdriver%	Название модуля DNS
%ip%	IP адрес сервера
%disk%	Буква диска из генерируемого пути до папки с Open Server с учетом виртуального диска (только буква)
%osdisk%	Буква диска из реального пути до папки с Open Server (только буква)
%sysdisk%	Системный диск Windows (только буква)

Примеры заданий планировщика

Запрос по протоколу HTTP каждые 30 минут:

*/30 * * * *

"%progdir%\modules\wget\bin\wget.exe" -q --no-cache http://xxx.ru/cron.php -O nul

Запрос по протоколу HTTPS каждый час в 10 минут, в 20 минут и в 40 минут:

10,20,40 * * * *

"%progdir%\modules\wget\bin\wget.exe" --secure-protocol=TLSv1_2 --no-check-certificate -q --no-cache https://xxx.ru/cron.php -O nul

Выполнение файла cron.php интерпретатором PHP ежеминутно:

*/1 * * * *

"%progdir%\modules\php\%phpdriver%\php-win.exe" -c "%progdir%\modules\php\%phpdriver%\php.ini" -q -f "%sitedir%\xxx.ru\cron.php"

Информация

Конструкция -O nul в аналогична -O /dev/null в Linux и используется в примере для того, чтобы ответ, полученный wget от сервера, не сохранялся на диске.

Исполняемый файл php-win.exe используется вместо обычной CLI-версии PHP потому,

что он ничего не выводит и, соответственно, не открывает консоль (окошко dos не появляется на экране).

Предупреждение

При составлении заданий для планировщика заключайте любые указания путей файловой системы в кавычки. В противном случае задание не сможет выполняться при наличии пробела(ов) в пути к исполняемому файлу или в его аргументах (при указании папок и файлов).

Предопределенные переменные планировщика

Переменная	Описание переменной
%hh%	Текущий час (00-23)
%mm%	Текущая минута (00-59)
%ss%	Текущая секунда (00-59)
%MM%	Текущий месяц (01-12)
%MMM%	Текущий месяц (Янв-Дек)
%DD%	Текущий день (01-31)
%WW%	Текущий день недели (Пн-Вс)
%WD%	Текущий день недели (1-7, 1-понедельник, 7-воскресенье)
%YYYY%	Текущий год (4 цифры)
%YY%	Текущий год (последние 2 цифры)
%QUOTE%	Знак кавычек
%PERCENT%	Знак процента

В таблице показаны не все переменные. Полный список смотрите здесь.

При составлении заданий вы можете использовать как переменные для подстановки значений из Open Server, так и предопределённые переменные самого планировщика. Кроме того, в качестве переменных планировщика можно использовать любые переменные среды окружения (environment variables), например %COMSPEC%.

Примеры заданий с использованием переменных

Очистка папки c:\temp\:

%COMSPEC% /c del /S /Q c:\temp*

Задание будет развёрнуто планировщиком так:

C:\Windows\system32\cmd.exe /c del /S /Q c:\temp*

А теперь пример создания резервной копии баз данных:

"%realprogdir%\modules\database\%mysql_driver%\bin\mysqldump.exe" -A --add-drop-database -E -i -Q -R --opt --user=root --password=root --result-file="%realprogdir%\%DD%_%MM%_%YYYY%_%hh%_%mm%_dump.sql"

Задание будет развёрнуто планировщиком так:

"C:\OSPanel\modules\database\MariaDB-10.3\bin\mysqldump.exe" -A --add-drop-database -E -i -Q -R --opt --user=root --password=root --result-file="C:\OSPanel\09_01_2020_14_25_dump.sql"

Информация

Следует принимать во внимание тот факт, что выполнение заданий происходит без переключения в рабочую папку той программы или скрипта, которые указаны в задании. Некоторые программы и PHP скрипты могут работать неправильно, если определение корневой папки реализовано в них некорректно.

Меню закладок

Для быстрого доступа к нужным страницам на сайтах и рабочим папкам, для запуска и выполнения любых программ и команд в Open Server существует возможность создавать закладки.

Виды закладок
Ссылка
Папка
Программа (возможен запуск с параметрами)
Команда в формате командной строки Windows

Использование переменных в качестве подстановок

Переменная	Описание переменной
%realprogdir%	Реальный путь до папки с Open Server (обратный слеш "\")
%progdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (обратный слеш "\")
%sprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (слеш "/")
%dprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (двойной обратный слеш "\\")
%dsprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (двойной слеш "//")
%realsitedir%	Реальный путь до корневой папки доменов (обратный слеш "\")
%sitedir%	Генерируемый путь до корневой папки доменов с учетом виртуального диска (обратный слеш "\")
%ssitedir%	Генерируемый путь до корневой папки доменов с учетом виртуального диска (слеш "/")
%httpport%	Порт HTTP сервера
%httpsport%	Порт HTTPS сервера
%postgresqlport%	Порт PostgreSQL сервера
%mysqlport%	Порт MySQL сервера
%mongodbport%	Порт MongoDB сервера
%memcacheport%	Порт Memcache сервера
%ftpport%	Порт FTP сервера

Переменная	Описание переменной
%httpdriver%	Название модуля HTTP
%phpdriver%	Название модуля PHP
%mysql_driver%	Название модуля MySQL / MariaDB
%pg_driver%	Название модуля PostgreSQL
%mongo_driver%	Название модуля MongoDB
%memcachedriver%	Название модуля Memchache
%redisdriver%	Название модуля Redis
%dnsdriver%	Название модуля DNS
%ip%	IP адрес сервера
%disk%	Буква диска из генерируемого пути до папки с Open Server с учетом виртуального диска (только буква)
%osdisk%	Буква диска из реального пути до папки с Open Server (только буква)
%sysdisk%	Системный диск Windows (только буква)

Внимание

При создании закладок на запуск программ с параметрами разделителем пути до программы и параметрами запуска является знак #, иначе закладка будет выполнена через командную строку как обычная команда!

Примеры создания закладки на запуск программы с параметрами:

%realprogdir%\modules\heidisql\heidisql.exe#-h=127.0.0.1 -u=root -p=root -P=%mysqlport%

Меню программ

Добавление портативной программы

В Open Server существует возможность добавить в меню свою портативную программу. Для этого необходимо выполнить следующие действия:

Откройте папку ./progs/ в каталоге с Open Server или создайте такую папку если её не существует;

Создайте в папке ./progs/ подпапку с именем категории программ, например ./progs/Офисные программы, или используйте имя Default (программы из подпапки Default отображаются в корне меню, без категории);

Скопируйте каталог с вашей портативной программой (например CintaNotes) в созданную вами папку ./progs/Офисные программы, в итоге у вас должен получиться такой путь: ./progs/Офисные программы/CintaNotes;

Создайте в папке с портативной программой файл osinit.txt содержащий единственную строку с текстом, которая будет использована в меню Open Server вместо отображения оригинального названия EXE файла программы;

Перезапустите управляющую программу Open Server;

Внимание

Имя портативной программы (без расширения) и имя папки, в которой она находится, должны быть идентичны! Например, если главный файл портативной программы называется CintaNotes.exe, то путь к нему должен выглядеть так: ./progs/Офисные программы/CintaNotes/CintaNotes.exe

Добавление категории программ

Чтобы добавить собственную категорию в меню программ следует создать одноимённую папку в директории ./progs/. Программы добавленные в папку Default отображаются в корне меню программ, без категории.

Добавление ярлыка программы

Помимо портативного софта в меню программ можно добавить ярлыки на локально установленные программы. Такой указатель на программу будет отображаться в меню вместе с другими программами и будет работать как самый обычный ярлычок. Чтобы добавить ярлык на локально установленную программу просто скопируйте его в папку ./progs/нужная_вам_категория_программ/ или ./progs/Default/.

Информация

Не пытайтесь скопировать в папку портативных программ ярлыки на папки, сайты и другие объекты не являющиеся программами. Такие ярлыки не будут отображаться в меню и соответственно не будут работать. Для создания ярлыков к сайтам, папкам и т.д. используйте [Меню → Настройки → Закладки] (см. Меню закладок).

Старт/стоп скрипты

Дополнение сценариев запуска и остановки сервера

В Open Server существует возможность дополнять сценарии запуска и остановки сервера своими .bat файлами (батниками).

Чтобы дополнить какой-либо сценарий необходимо создать файл-шаблон .tpl.bat в папке ./userdata/ с определённым именем (см. список ниже). Вы можете записать в такой файл произвольные последовательности команд, предназначенных для исполнения командным интерпретатором Windows.

Нужно понимать, что файлы с расширением .tpl.bat являются только шаблонами для итоговых .bat файлов. В процессе запуска или остановки сервера файлы-шаблоны будут преобразованы в исполняемые .bat файлы в той же папке с заменой переменных-подстановок и уже эти временные .bat файлы будут отправлены на выполнение.

Перед запуском

Используйте шаблон с именем /userdata/pre_start.tpl.bat для указания команд выполняемых ДО запуска всех модулей.

После запуска

Используйте шаблон с именем /userdata/start.tpl.bat для указания команд выполняемых ПОСЛЕ запуска всех модулей, но до того, как флаг состояния (в области уведомлений Windows) станет зеленым.

Перед остановкой

Используйте шаблон с именем /userdata/stop.tpl.bat для указания команд выполняемых ДО остановки модулей.

После остановки

Используйте шаблон с именем /userdata/post_stop.tpl.bat для указания команд выполняемых ПОСЛЕ остановки модулей, но до того, как флаг состояния (в области уведомлений Windows) станет красным.

Предупреждение

После начала процедуры завершения работы Windows запуск любых новых процессов невозможен, поэтому СТОП-скрипты (stop.bat и post_stop.bat) не выполняются. Если вам требуется обязательное выполнение скриптов остановки, то необходимо выключать Open Server вручную непосредственно перед тем, как вы хотите выключить компьютер.

Внимание

Каждый созданный вами батник (для любого сценария из перечисленных выше) должен корректно отработать и завершиться, иначе сервер будет ожидать окончания его выполнения бесконечно.

Использование переменных в качестве подстановок

В файлах-шаблонах с расширением .tpl.bat можно использовать подстановки. Доступные варианты подстановки программных переменных:

Переменная	Описание переменной
%realprogdir%	Реальный путь до папки с Open Server (обратный слеш "\")
%progdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (обратный слеш "\")
%sprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (слеш "/")
%dprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (двойной обратный слеш "\\")
%dsprogdir%	Генерируемый путь до папки с Open Server с учетом виртуального диска (двойной слеш "//")
%realsitedir%	Реальный путь до корневой папки доменов (обратный слеш "\")
%sitedir%	Генерируемый путь до корневой папки доменов с учетом виртуального диска (обратный слеш "\")
%ssitedir%	Генерируемый путь до корневой папки доменов с учетом виртуального диска (слеш "/")
%httpport%	Порт HTTP сервера
%httpsport%	Порт HTTPS сервера
%postgresqlport%	Порт PostgreSQL сервера
%mysqlport%	Порт MySQL сервера
%mongodbport%	Порт MongoDB сервера
%memcacheport%	Порт Memcache сервера
%ftpport%	Порт FTP сервера

Переменная	Описание переменной
%httpdriver%	Название модуля HTTP
%phpdriver%	Название модуля PHP
%mysql_driver%	Название модуля MySQL / MariaDB
%pg_driver%	Название модуля PostgreSQL
%mongo_driver%	Название модуля MongoDB
%memcachedriver%	Название модуля Memchache
%redisdriver%	Название модуля Redis
%dnsdriver%	Название модуля DNS
%ip%	IP адрес сервера
%disk%	Буква диска из генерируемого пути до папки с Open Server с учетом виртуального диска (только буква)
%osdisk%	Буква диска из реального пути до папки с Open Server (только буква)
%sysdisk%	Системный диск Windows (только буква)

Кроме написания команд перечисленные выше переменные можно использовать для указания реальных переменных окружения, например:

setx DIR_PHP "%realprogdir%\modules\php\%phpdriver%"

Ограниченный режим

При недоступном на запись HOSTS файле программа переходит в ограниченный режим работы с урезанной функциональностью.

В ограниченном режиме вам будет недоступна следующая функциональность:

Использование своих алиасов и доменов (кроме домена localhost);

Указание IP адреса сервера (кроме адресов 127.0.0.1 и *);

Таким образом в ограниченном режиме вам будет доступен один из двух IP адресов: 127.0.0.1 или *, и только один домен localhost. Другие созданные вами алиасы и домены не будут обработаны программой.

Работать в ограниченном режиме уместно только при полном отсутствии прав администратора, например в условиях организации. В любом другом случае крайне рекомендуется правильно настроить права доступа к HOST файлу для возможности полноценной работы с разными доменами. Разрешить запись в HOSTS файл для всех пользователей можно выполнив через консоль (запускать от имени Администратора) следующую команду:

attrib -s -r -h -a C:\Windows\system32\drivers\etc\hosts

Если в логах запуска вы видите сообщение о том, что Hosts файл недоступен для записи, то возможно что доступ к этому файлу блокируется антивирусом/файрволом, либо действуют ограничения прав доступа Windows.

Добавьте Open Server, а так же все остальные компоненты, о которых будет спрашивать антивирус/файрвол, в доверенные программы. Отключите защиту HOSTS файла (или системных файлов) в настройках вашего антивируса/файрвола, если такая

защита присутствует. Попробуйте вручную удалить файл C:\Windows\System32\Drivers\etc\hosts и заново создать со следующим содержимым:

127.0.0.1 localhost

В случае работы без прав администратора, но с доступным на запись HOSTS файлом, программа работает в нормальном режиме без каких-либо ограничений.

Предупреждение

При включённой службе контроля учётных записей пользователей (UAC) и запуске без прав администратора Open Server не будет иметь доступа к HOSTS файлу и автоматически перейдёт в ограниченный режим работы.

Запуск без внесения записей в HOSTS файл

В Open Server реализована возможность полноценного запуска без внесения записей в HOSTS файл. Эта возможность будет полезна пользователям офисных сетей и терминалов, где доступ к HOSTS файлу имеет только старший администратор. Если опция [Не вносить изменения в HOSTS файл] включена, то запуск сервера происходит без редактирования HOSTS файла будто все домены в нём уже прописаны, т.е. доступ к этому файлу не требуется вовсе.

Следует не забывать обращаться к администратору вашей сети после каждого создания домена, администратор должен внести нужные записи в HOSTS файл, иначе вы не сможете их использовать.

Формат внесения записей в HOSTS файл стандартный - *ip пробел домен*, например:

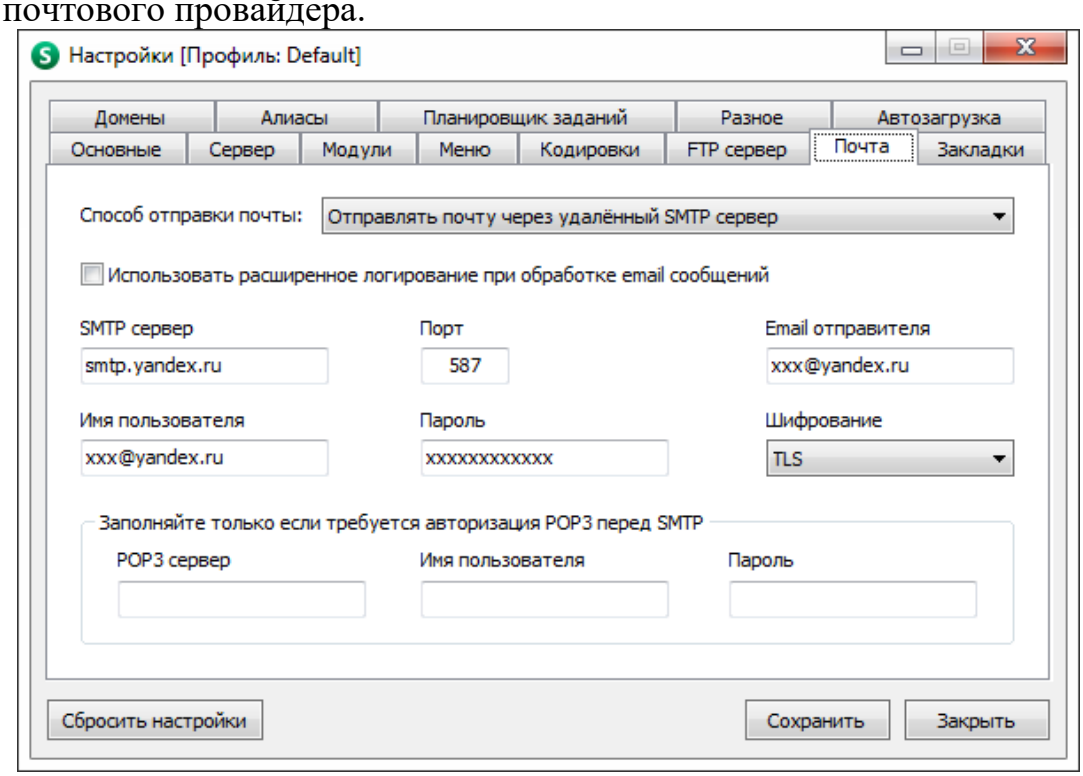
192.168.5.10 rhino.acme.com

192.168.5.10 x.acme.com

Отправка почты через SMTP

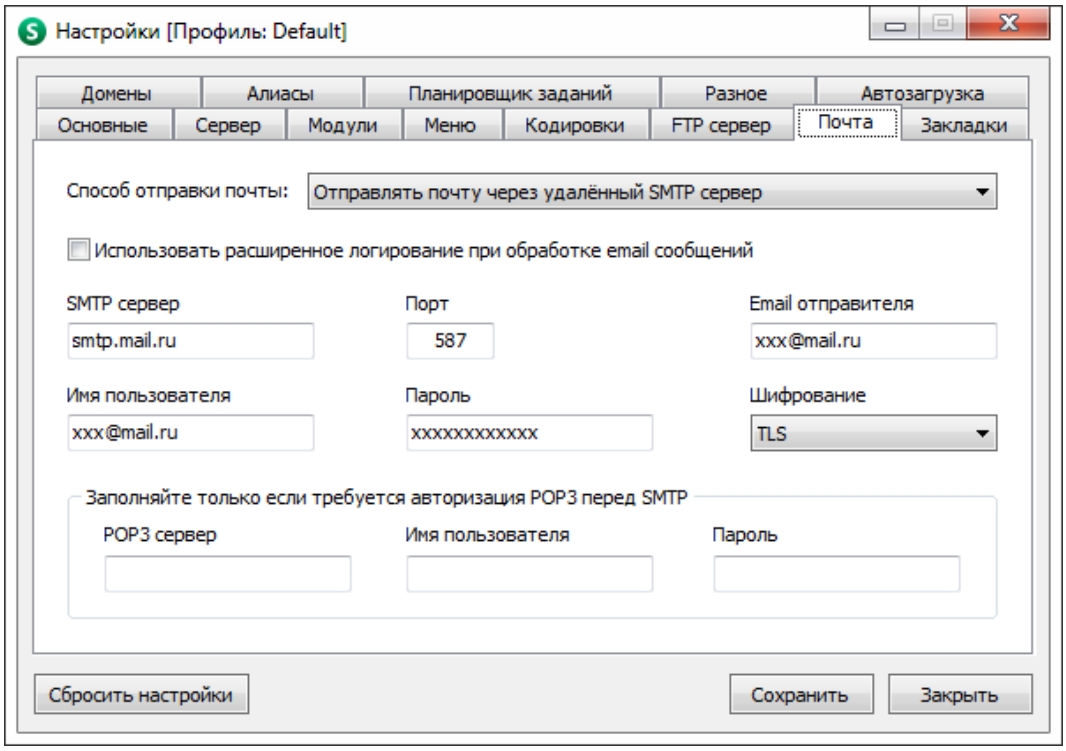
Яндекс почта

Ниже показаны типичные настройки для отправки почты через SMTP сервер выбранного почтового провайдера.



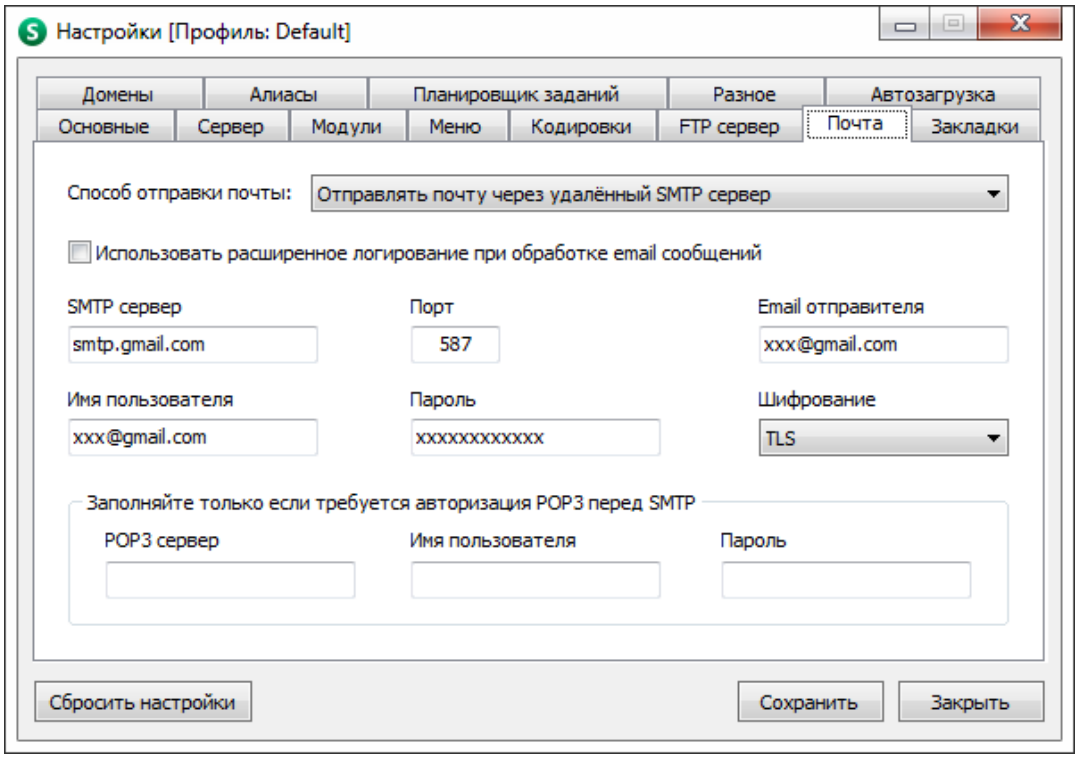
Mail.ru почта

Ниже показаны типичные настройки для отправки почты через SMTP сервер выбранного почтового провайдера.



Gmail почта

Ниже показаны типичные настройки для отправки почты через SMTP сервер выбранного почтового провайдера.

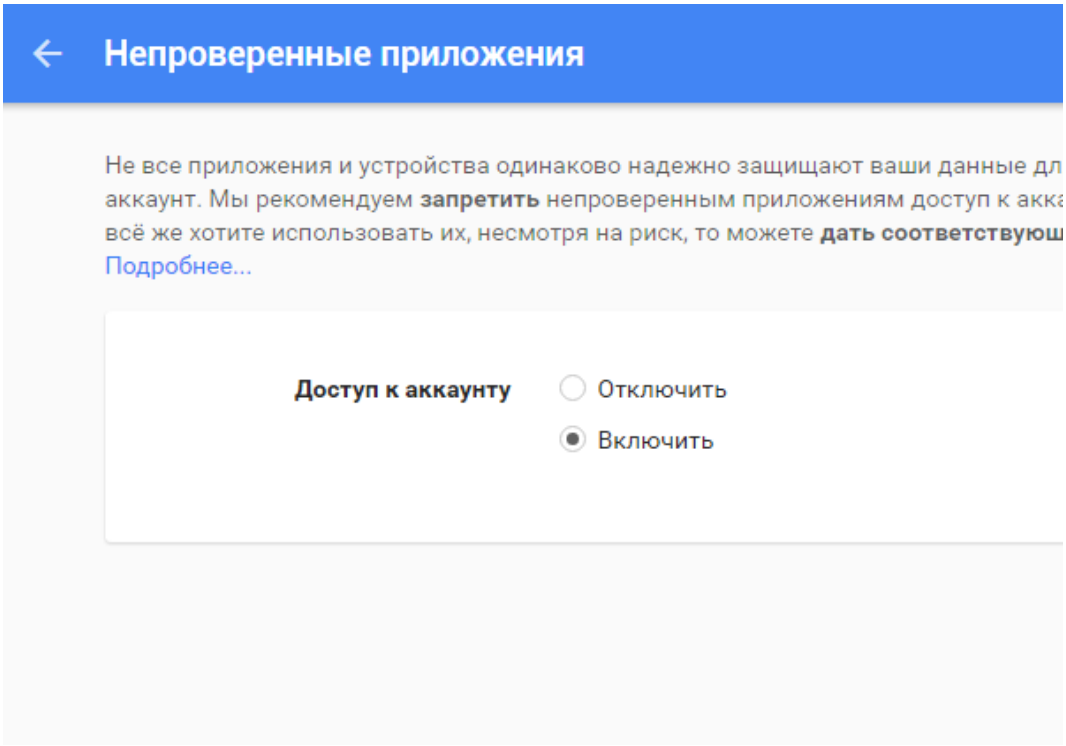


По умолчанию Gmail не даёт возможности использовать SMTP, поэтому доступ нужно активировать отдельно. Вам необходимо выполнить следующие шаги по активации:

1. Авторизоваться в своём аккаунте Gmail

2. 2. В той же вкладке браузера перейти по адресу https://www.google.com/settings/security/lesssecureapps
   3. Активировать доступ к аккаунту для "непроверенных приложений" (см. картинку ниже)
3. 4. Выполнить тестовую отправку письма через ваш скрипт на сервере Open Server
4. 5. Обновить страницу https://www.google.com/settings/security/lesssecureapps и снова активировать доступ к аккаунту для "непроверенных приложений" если он не активировался (пришедшее письмо о попытке доступа к аккаунту можно проигнорировать)

Теперь попробуйте отправить письмо еще раз, на этот раз отправка должна заработать.



Информация

Если вы включили двухфакторную авторизацию для своей почты у любого почтового провайдера, то вам необходимо создать пароль приложения в панели управления своего почтового аккаунта. Используйте этот пароль в настройках программы.

Внешний доступ

Работа с внешними сетями

Open Server может работать в локальных сетях и сети Интернет как веб-сервер. Для работы в сети Интернет необходим статический(белый) IP адрес. Чтобы открыть доступ к серверу извне необходимо указать * в качестве IP адреса в настройках Open Server, в этом случае доступ откроется сразу для всех сетей, к которым подключён ваш компьютер. Если вы хотите открыть доступ только для одной конкретной сети (например локальной), то укажите в настройках программы IP адрес выданный вашему компьютеру в этой сети.

Информация

При работе через роутер или другой шлюз может потребоваться их дополнительная настройка. См. документацию к используемому оборудованию, обратитесь к администратору сети.

Доступ извне к локальными доменам

После того, как вы настроите удалённый доступ к своему серверу, все локальные домены станут доступны удалённо. Но как их открыть? Чтобы открыть нужный сайт с удалённой машины потребуется внести в HOSTS файл такого компьютера запись с адресом вашего локального домена, например: 64.55.96.24 supersite (где 64.55.96.24 это адрес вашего компьютера, а supersite это имя локального домена). После внесения такой записи в HOSTS файл на удалённом компьютере домен supersite будет доступен при обычном наборе в строке браузера.

Обратите внимание - при простом наборе IP адреса вашего компьютера http://64.55.96.24/ или при попытке доступа к несуществующему локальному домену будет открываться пустая страница.

Для того, чтобы при наборе IP адреса вашего компьютера открывался определённый локальный сайт, необходимо создать алиас вида ваш_внешний_ip => ваш_локальный_домен. После этих действий указанный вами локальный домен, для которого вы создали алиас, станет доступен через локальную сеть или сеть Интернет по адресу http://ваш_внешний_ip/ (например http://64.55.96.24/). Если ваш компьютер подключен к нескольким сетям и в настройках сервера установлен IP=*, то такой алиас необходимо создать для каждого IP адреса выданного вашему компьютеру в каждой из сетей.

Внимание

При работе с внешними сетями нет никакой гарантии безопасности вашего компьютера. Сервер часто запускается с правами Администратора, а значит, скрипты, запущенные под его управлением, могут делать на компьютере всё, что угодно. Дыры в безопасности скриптов могут открыть хакерам и вирусам доступ к вашему компьютеру.

Мы не рекомендуем такое использование Open Server! Перед настройкой удалённого доступа к серверу обязательно ознакомьтесь с разделом Защита сервера данного справочного руководства.

Внешнее управление

Open Server может принимать некоторые команды через сеть Интернет, для этого имеется встроенная панель управления которая работает на выделенном порту. Так же программой можно управлять через командную строку.

Управление через Интернет

Чтобы получить доступ к панели управления необходимо набрать адрес http://localhost:1515/ (по умолчанию). Порт, логин и пароль к контрольной панели можно указать непосредственно в настройках Open Server [Меню → Настройки → Разное].

Панель управления доступна на любом IP адресе с которого доступен ваш компьютер, а так же на любом локальном домене из созданных на сервере. Для управления программой через Интернет ваш внешний IP адрес должен быть "белым", т.е. доступным из сети Интернет.

Команды для использования в командной строке

```
"C:\openserver\Open Server.exe"            # запуск программы
"C:\openserver\Open Server.exe" /start     # запуск сервера
"C:\openserver\Open Server.exe" /restart   # перезапуск сервера
"C:\openserver\Open Server.exe" /stop      # остановка сервера
"C:\openserver\Open Server.exe" /exit      # выход из программы
```

Защита сервера

Настройка защиты

Сервер становится крайне уязвимым, когда он открыт для доступа из сети Интернет, особенно с настройками установленными по умолчанию. Множество ботов и вирусов постоянно сканируют ip адреса в сети Интернет на предмет открытых портов и, как правило, незащищенный сервер оказывается взломанным уже через несколько часов после появления в сети.

Несколько шагов по защите веб-сервера от несанкционированного доступа:

Отключите FTP сервер [Меню → Настройки → FTP сервер];

Установите собственные пароли для root (и других) пользователей всех модулей СУБД;

Включите защиту от внешнего доступа в настройках программы [Меню → Настройки → Сервер];

Теперь перезапустите саму управляющую программу (не сервер);

Выполните настройку файрвола закрыв на доступ извне все порты кроме тех, которые планируется использовать (например: 80,443,21,990,53);

Предупреждение

Не допускайте использования уязвимых скриптов, некорректной конфигурации модулей, простых паролей.

Встроенная защита от внешнего доступа

Выбор опции [Меню → Настройки → Сервер → Защитить сервер от внешнего доступа] отключит часть опасных функций PHP, доступ к веб-инструментам извне будет заблокирован, а доступ к диску для PHP скриптов будет ограничен корневой папкой доменов. Данная опция снижает производительность php-скриптов в 2-10 раз (зависит от интенсивности работы с файловой системой)!

Работа с Composer

Composer совместим со всеми версиями PHP начиная с 5.3 и в Open Server он доступен во всех совместимых модулях "из коробки".

Установка пакетов Composer

Выполните [Меню → Дополнительно → Консоль];

Перейдите в папку с тем сайтом, куда планируется установка;

Выполните установку любого нужного вам пакета, например:

```
# Переход в папку с проектом
cd C:\openserver\domains\localhost

# Установка Symfony
composer create-project symfony/framework-standard-edition symfony

# Установка PHPUnit
composer create-project phpunit/phpunit phpunit & echo @php -d output_buffering=0 phpunit\phpunit %*>phpunit.bat

# Установка Laravel
composer create-project laravel/laravel laravel --prefer-dist
```

Установка phpDocumentor

composer create-project phpdocumentor/phpdocumentor phpdocumentor

В будущем, чтобы обновить установленный таким образом проект, достаточно перейти в папку с нужным проектом и выполнить команду composer update:

cd C:\openserver\domains\localhost\phpunit & composer update

Работа в консоли

Для запуска встроенной консоли необходимо запустить сервер и выполнить [Меню → Дополнительно → Консоль]. Среда окружения (англ. Environment) формируется в момент запуска сервера и передаётся всем запускаемым модулям. Любые другие программы, будь то консоль или IDE, так же могут получить правильное окружение, достаточно запускать эти программы из меню Open Server. Для этого можно либо сделать закладку, либо добавить ярлык в меню программ (см. разделы Меню закладок и Меню программ).

Необходимо знать, что если вы запускаете консоль или любую другую программу из стандартного меню Пуск или используя ярлык на рабочем столе Windows, а не из меню Open Server, то они не смогут работать с виртуальным окружением сформированным в Open Server.

Встроенную консоль можно запустить даже если сервер выключен, в Full версии выполните [Меню → Программы → Консоль]. Однако при выключенном сервере среда окружения не сформирована и поэтому работать из консоли с PHP и другими модулями или программами (wget, composer, скриптами и т.д.) будет невозможно.

Внимание

Если вы запустили консоль ДО запуска сервера, то среда окружения НЕ станет доступна в консоли.

Чтобы начать полноценную работу с модулями после запуска сервера нужно закрыть и заново открыть встроенную консоль (саму программу ConEmu, а не только вкладку), поскольку только при запущенном сервере консоль получает правильно сформированное окружение в момент запуска из меню.

Точно так же среда окружения не будет обновлена в консоли если вы переключились на другой модуль PHP или MySQL и перезапустили сервер, но при этом не перезапустили консоль!

Внимание

При запуске стороннего ПО (консоли, программы, IDE, скрипты и проч.) строго придерживайтесь правила: сначала запустить сервер - потом программу.

Если вы изменили настройки Open Server или конфиги модулей: выключить программу - потом запустиь её заново.

Не пытайтесь запускать софт в обход меню Open Server, т.к. в этом случае рабочее окружение не будет доступно в вашей программе.

Фиксы реестра

Все необходимые исправления и оптимизации реестра применяются автоматически при выполнении процедуры первого запуска [Меню → Дополнительно → Первый запуск]. На данный момент предлагается лишь два дополнительных фикса реестра (см. ниже).

Фикс для отключения LPM и DIPM

Этого фикса изначально нет в закладках, т.к. он спицифичен только для драйвера от Intel и только для ноутбуков. Данное исправление позволяет отключить Link Power Management (LPM) и Device Initiated Link Power Management (DIPM).

Использование LPM и DIPM с SSD дисками часто приводит к возможности возникновения BSOD или зависаний в работе операционной системы при использовании драйвера Intel Rapid Storage Technology (iastor), это проблема драйвера, а не вашего SSD. Вы можете найти этот фикс по пути ./userdata/modules/system/iastor.bat.

Не следует применять фикс если в системе не используются SSD диски или если у вас не ноутбук, а обычный домашний или офисный компьютер.

Фикс для оптимизации работы с SSD

Фикс называется ssd.bat и изначально присутствует в закладках меню. Для предотвращения износа SSD данный фикс отключает службы Superfetch и Prefetch, а так же обновление времени последнего доступа к файлам в NTFS.

Предупреждение

После применения фиксов следует обязательно перезагрузить компьютер.

Вопросы и ответы

Ничего не запускается?

Не стоит отчаиваться, загляните в общий лог программы и другие логи компонентов [Меню → Просмотр логов]. В подавляющем большинстве случаев там вы найдете причину неудачного старта. В более сложных случаях включите опцию [Запускать сервер в отладочном режиме], это позволит увидеть отладочную информацию при запуске. Так же добавьте Open Server в доверенные программы вашего файрвола или антивируса, если таковой имеется.

Пишет что порт 80, 3306 и т.д. уже занят!

Включите в настройках Open Server опцию [Запускать сервер в агрессивном режиме], в этом режиме все программы занимающие нужные порты будут принудительно закрыты. Добавьте Open Server в доверенные программы вашего антивируса/файрвола/прокси-сервера или отключите слежение за портами 80/443/21/90xx/3306 если оно есть. В отдельных антивирусах/файрволах возможны свои доп. настройки связанные с перехватом подключений программ к сети.

Пишет что нет прав доступа для работы в этой папке!

Вы пытаетесь запустить Open Server из папки, которая принадлежит другому пользователю, или же файлы Open Server были записаны на компьютер другим пользователем. Если Open Server будет запускаться пользователем Pavel (например), то войдите в систему под учётной записью Pavel и скопируйте Open Server в такую папку, которая принадлежит пользователю Pavel (имеются права на запись), после чего запуск сервера для этого пользователя станет возможным. Вместо копирования можно установить особые права доступа к папке с Open Server, обратитесь к администратору вашего компьютера.

Почему кнопки серые и не нажимаются?!

В любой момент времени активны только те кнопки и разделы меню, которые могут выполнить возложенное на них действие. Например: если сервер не запущен, то пункт меню PHPMyAdmin будет неактивен; если сервер запущен, то кнопки очистки логов будут недоступны и т.д.

Я не могу открыть меню когда флаг желтого цвета!

Желтый флажок говорит о том, что происходит выполнение команды (например запуск, остановка или сохранение настроек). Во время выполнения команд меню программы недоступно.

Создал 3000 доменов и теперь сервер не запускается!

Увеличьте кол-во проверок состояния сервера до 20-30 или выше в использование HTTP модуля Apache если вы используете модуль Nginx.

Хочу протестировать сайт с учётом разных скоростей интернета (мобильные устройства к примеру)!

В Open Server существует возможность ограничения скорости передачи данных для симуляции медленной загрузки сайта. Для ограничения скорости воспользуйтесь меню Open Server: [Меню → Настройки → Разное]. Скорость можно ограничить в пределах 1-40 КБайт в секунду, установка значения в 0 или более 40 отменяет ограничение. Скоростной лимит устанавливается на каждый запрос, поэтому, если клиент одновременно откроет 4 картинки на сайте (4 соединения), то каждая из них будет загружаться с заданной скоростью.

Где редактировать настройки модулей?

Для редактирования настроек модулей пользуйтесь меню Open Server: [Меню → Дополнительно → Конфигурация].

Куда сохраняются мои письма?

По умолчанию все письма сохраняются во временную папку userdata/tmp/email/, во время остановки сервера эта папка не очищается. Можно настроить отправку писем через удалённый SMTP сервер: [Меню → Настройки → Почта].

Я нажимаю в меню программы на нужный сайт, но ничего не открывается!

Для работы многих пунктов меню требуется наличие корректно установленного и работающего браузера по умолчанию. Так же вы можете напрямую указать свой браузер в настройках программы: [Меню → Настройки → Меню].

Я создал домен local_site.loc, но он не работает!

Символа подчеркивания не может быть в имени домена, поэтому такой домен не подключается. Разрешенные символы: [a-z0-9.-].

Сервер не стартует с ip 192.168.0.1 и говорит что localhost работает только на ip 127.0.0.1!

Действительно, домен localhost можно использовать только с ip адресом 127.0.0.1, поэтому либо переименуйте этот домен, либо удалите его, если он вам не нужен.

Я хочу работать без виртуального диска!

Выберите в настройках Open Server работу без виртуального диска и скопируйте папку с сервером в такую директорию, путь до которой содержит только латиницу или цифры, в противном случае работа без вирт. диска будет невозможна.

Я не вижу виртуального диска!

Если вы работаете не под учётной записью администратора, но Open Server запущен с правами администратора, то виртуальный диск (если он используется) будет

для вас недоступен. Open Server при этом будет работать нормально. Вы можете отключить использование виртуального диска в настройках, при этом учитывайте требования к абсолютному пути описанные в предыдущем ответе.

Сервер запускается, но браузер говорит что домен не найден!

Если в вашем браузере настроена работа через прокси сервер, то локальные сайты будут ему недоступны. Чтобы исправить ситуацию откажитесь от использования прокси в вашем браузере или добавьте локальные домены и IP адрес сервера в список исключений для прокси в настройках обозревателя. В ряде случаев нужно пересоздать файл HOSTS (см. Решение проблем).

Для чего нужна настройка вывода закладок вместо сайтов?

В таком режиме созданные закладки отображаются вместо списка доменов, это будет полезно при создании демо-сборок с установленными скриптами (см. Создание сборок).

Мне трудно пересоздавать закладки, пользователей FTP и т.д, нужно их просто отредактировать!

Не нужно пересоздавать записи заново, просто сделайте двойной клик на интересующей вас записи в таблице после чего эта запись сразу же удалится из списка и будет вынесена обратно в поля редактирования. Не забудьте добавить отредактированную запись обратно в таблицу!

Подключение к базе данных длится более 1 сек!

Отключите использование протокола IPv6 используя инструкцию из раздела Решение проблем данного руководства.

Я скопировал в папку с доменом свой скрипт, после перезапуска сервер не видит файлов и показывает ошибку!

Скорее всего вы скопировали или создали такую подпапку в папке с доменом, имя которой подпадает под условия автоматического поиска корневой папки домена (см. Домены и алиасы).

Сервер постепенно занял всё место на диске!

Постепенное исчезновение свободного места говорит о том, что вы отключили автоочистку логов при запуске сервера. В результате при активном использовании сервера лог-файлы вырастают до огромных размеров и занимают всё свободное место на диске. Очистите лог-записи в окне просмотра логов и включите автоочистку логов при старте в настройках Open Server. Очистите папку .\userdata\tmp от временных файлов.

Как мне создать собственную конфигурацию хоста для домена?

Используйте шаблон конфигурации виртуального хоста в папке с нужным доменом для создания особой конфигурации домена. При создании/изменении файла требуется перезапуск сервера. Шаблон конфигурации для нужного модуля можно найти в папке ./userdata/config/.

Например, если вы хотите создать собственную конфигурацию домена xxx.ru для модуля Apache-2.4.2, то скопируйте файл ./userdata/config/Apache-2.4.3_vhost.conf в

папку с нужным доменом, отредактируйте этот файл под свои нужды и перезапустите сервер.

Или другой пример, при использовании модуля Apache+Nginx можно скопировать в папку домена оба файла конфигурации для каждого сервера: Apache-2.2.23+Nginx-1.2.4_vhosta.conf и Apache-2.2.23+Nginx-1.2.4_vhostn.conf.

Обратите внимание - при редактировании конфигурации хоста нельзя удалять или заменять системные переменные %...%, вы можете вносить только новые записи дополняющие конфигурацию.

Как и где редактируются конфиги компонентов?

В Open Server используются файлы-шаблоны конфигурации, которые доступны через общее меню программы. Не пытайтесь редактировать временные файлы конфигов, которые создаются в момент запуска сервера, это не имеет смысла. Шаблоны конфигурации индивидуальны для каждого модуля, т.е. если вы выберите какой-либо из модулей в качестве активного, то будут активированы и шаблоны настроек именно этого модуля. Например: вы сконфигурировали шаблон файла php.ini при активном модуле PHP 5.2, после чего выбрали модуль PHP 5.3 в качестве активного. В этом случае вам необходимо еще раз сконфигурировать шаблон файла php.ini уже для модуля PHP 5.3.

Сами шаблоны настроек для всех модулей и инструментов хранятся в папке ./userdata/config/, их нельзя использовать как реальные конфиги и указывать при запуске модулей, консолей и скриптов. В процессе запуска шаблоны конфигурации проходят через парсер, который заменяет все переменные-подстановки на реальные данные и сохраняет уже готовые файлы конфигурации в папку ./userdata/temp/config/.

Как указать особые настройки подключения к SMTP серверу для выбранного домена?

В Open Server существует возможность отправки почты через сторонний SMTP сервер, при этом все домены используют те параметры подключения, которые были указаны пользователем в окне настроек Open Server. Для того чтобы определённый домен мог использовать собственные настройки подключения к SMTP серверу нужно указать их в шаблоне конфигурации этого хоста. Откройте шаблон конфигурации виртуального хоста (как его создать описано в предыдущем ответе) в папке с нужным доменом и добавьте в описание хоста следующее содержимое (одной строкой):

php_admin_value sendmail_path "%mailway% -t --smtp-ssl=none --smtp-server=smtp.xxx.xx --smtp-port=xx --smtp-from=xxx@xxx.xx --smtp-user=xxx@xxx.xx --smtp-pass=xxxxxx --pop3-server=pop3.xxx.xx --pop3-user=xxx@xxx.xx --pop3-pass=xxxxxx"

Например:

```
<VirtualHost *:%httpport%>
  %limit%
  DocumentRoot "%hostdir%"
  ...
  php_admin_value sendmail_path "%mailway% -t --smtp-ssl=none --smtp-server=smtp.xxx.xx --smtp-port=xx --smtp-from=xxx@xxx.xx --smtp-user=xxx@xxx.xx --smtp-pass=xxxxxx --pop3-server=pop3.xxx.xx --pop3-user=xxx@xxx.xx --pop3-pass=xxxxxx"
  ...
```

</VirtualHost>

.

Последнюю часть строки --pop3-server=pop3.xxx.xx --pop3-user=xxx@xxx.xx --pop3-pass=xxxxxx добавляйте только в том случае, если требуется авторизация POP3 перед SMTP. Возможные значения для параметра --smtp-ssl= это none,auto,ssl или tls.

Необходимо помнить, что при указании собственных настроек подключения к SMTP серверу отправка почты для выбранного домена будет работать не зависимо от основных настроек Open Server. Так же обратите внимание на то, что все данные SMTP сервера включая логин и пароль будут доступны в php скриптах и видны в информации phpinfo(), поэтому в целях безопасности не рекомендуется пользоваться описанной возможностью указания настроек отправки почты в конфиге хоста

Решение проблем

Ниже представлен порядок действий необходимых для решения проблем с запуском сервера. После выполнения каждого действия необходимо повторить пробный запуск сервера, а в случае сбоя запуска можно переходить к выполнению следующего пункта.

Выполните [Меню → Дополнительно → Первый запуск]. По желанию можно согласиться на внесение фиксов в реестр.

Убедитесь что IP адрес, заданный в настройках, существует (кроме [*]). Так же убедитесь что все другие настройки программы корректны.

Добавьте Open Server в доверенные программы вашего файрвола/антивируса или настройте их должным образом, если таковые имеются.

Запустите [Меню → Просмотр логов], попробуйте найти причину проблемы и устранить её (если это возможно).

Включите в настройках Open Server опцию [Запускать сервер в отладочном режиме] и попробуйте найти причину проблемы и устранить её (если это возможно).

Выполните через консоль (запускать от имени Администратора) следующую команду: attrib -s -r -h -a C:\Windows\system32\drivers\etc\hosts

Активируйте службу DNS если она отключена и запустите её, отключите все службы в имени которых присутствует IIS.

Отключите глобальные прокси и проксификаторы если они установлены на вашем компьютере.

Информация

Очень распространённая проблема это занятый порт 80, занимает его обычно программа Skype. Для устранения проблемы перейдите в [Настройки Skype → Дополнительно → Соединение] и снимите галочку с пункта [Использовать порты 80 и 443 в качестве альтернативных].

Контроль учётных записей пользователей (UAC)

При включённой службе контроля учётных записей пользователей (UAC) и запуске без прав администратора Open Server не будет иметь доступа к HOSTS файлу и автоматически перейдёт в ограниченный режим работы. Как правило, об ограниченном режиме сигнализирует наличие только одного домена localhost в меню программы, в то время как ранее были созданы и другие домены. Чтобы исправить ситуацию включите опцию [Требовать учётную запись Администратора] в настройках Open Server и перезапустите программу, либо отключите контроль учётный записей (UAC).

Если у вас отсутствует возможность отключения UAC или запуска программы с правами администратора, то существует несколько вариантов решения данной проблемы:

Использование встроенного DNS сервера (настройка согласно Руководства);

Активация опции [Не вносить изменения в HOSTS файл] + ручное редактирование этого файла;

Установить разрешающие права записи в HOSTS файл для всех пользователей Windows;

Отказ от возможности управления доменами и работа с единственным доменом localhost;

Создание сборок

Что это такое и зачем это нужно?

Например у вас есть свой проект (допустим некая CMS), который вы бы хотели сделать доступным на своём сайте в качестве демо-версии для потенциальных клиентов, показать заказчику или попросту сделать демонстрационный пакет на все случаи жизни. Нужно чтобы ваш программный продукт работал одинаково хорошо у всех, в одинаковой конфигурации, уже был установлен и готов для работы или ознакомления. В этом вам как раз и поможет Open Server.

Вы сможете создать готовый и самое главное рабочий мини-сервер с уже установленным скриптом, CMS системой, сайтом, проектом и т.д., а вашим клиентам останется его только скачать и запустить. Ваш продукт будет изначально работоспособен, не потребуется искать для него сервер, не нужно будет выполнять установку, настройку и т.д.

Как создать кастомную сборку

Скачайте дистрибутив Open Server Basic последней версии и распакуйте его во временную папку. Перейдите в папку куда был распакован дистрибутив и запустите программу Open Server.exe.

Удалите ненужные модули в папке ./modules/php/*, ./modules/database/* и ./modules/http/* оставив в папках только по одному модулю, которые будут использоваться для вашего программного продукта. Если вы не планируете использовать модули DNS, Redis и Memcached, то папки с этими модулями так же можно удалить: ./modules/dns/*,./modules/redis/* и ./modules/memcached/* соответственно. Не удаляйте корневые каталоги с подтипом модулей даже если они пусты, это сделает программу неработоспособной.

Нажмите [Меню → Настройки] и выберите для использования те модули, которые вы не удалили. Установите IP адрес сервера в значение 127.0.0.1, отключите [Автоматическую проверку обновлений], не отключайте опцию [Автоматически определять потребность в виртуальном диске]. Установите все другие нужные вам настройки.

Нажмите [Меню → Дополнительно → Конфигурация] и настройте шаблоны конфигурации нужных модулей под ваш продукт (скрипт/сайт/cms), если это необходимо.

Запустите сервер и установите ваш программный продукт (скрипт/сайт/cms) на домен localhost, который присутствует в Open Server по умолчанию. Выполните необходимые действия по настройке скрипта после чего выйдите из всех форм авторизации где вы авторизовались (например: личный аккаунт, администрирование и т.д.).

Остановите сервер. Нажмите [Меню → Настройки] и создайте закладки для основных страниц вашего программного продукта (например: Админ-панель, Личный аккаунт, Главная страница). Включите в настройках Open Server опцию [Показывать закладки вместо сайтов]. Сохраните настройки.

Нажмите [Меню → Выход] и переименуйте файл программы согласно имени вашего продукта, например ./временная папка/Open Server.exe в ./временная папка/MegaCMS Server.exe.

Теперь упакуйте содержимое временной папки в самораспаковывающийся архив. Тип и формат архива зависит лишь от предполагаемой формы распространения сборки.

Готово!

Вышеописанным способом вы можете подготавливать не только кастомные сборки, но и dev-пакеты для разработчиков, offline-презентации сайтов, архивы порталов, интерактивные материалы на CD и т.д.

Лекция 1.5 Знакомство с HTML. Синтаксис языка

Структура HTML-документа

При написании HTML-кода в блокноте, желательно придерживаться одного стиля. Схема стандартного HTML-документа, выглядит следующим образом:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Название страницы<title>
 </head>
 <body>
  <h1>Заголовок статьи</h1>
  <p>Абзац
   статьи</p>
 </body>
</html>
```

Каждый HTML-документ должен начинаться со строки <!DOCTYPE html>, она означает что код в документе будет написан на языке HTML. Затем идёт сам HTML-код, который начинается и заканчивается тегами <html> </html>.

Между тегами <html> </html> располагаются два основных блока, первый блок — это голова HTML-документа, который начинается и заканчивается тегами <head> </head>, второй блок — это тело HTML-документа, который начинается и заканчивается тегами <body> </body>.

В голове HTML-документа содержится различная служебная информация, которую пользователь не видит (кроме тега title), там находятся следующие теги:

<title> </title> — название HTML-страницы,

<meta> — мета-теги, в них содержится служебная информация о странице,

<link> — тег ссылающийся на внешние файлы, например .css, .ico и т.д.,

<script> </script> — теги могут содержать JavaScript-код или ссылаться на внешний файл .js

В теле HTML-документа обычно содержится основная информация, которую мы видим на странице, там могут находиться следующие теги:

<h1> </h1> — заголовок статьи, первого уровня,

<img> — изображение,

  — абзац,

<a> — ссылка,

<table> </table> — таблица,

<form> </form> — форма ввода данных,

и т.д.

Правила написания HTML-кода

Рассмотрим некоторые правила написания HTML-кода. Данные правила нужны для того, чтобы потом удобно было разбираться в собственном коде.

Блочные теги которые находятся внутри других тегов, лучше размещать на одну строку ниже и на один пробел (табуляцию, как вам удобней) правее от тега в котором он размещен. Например таким образом расположены заголовок h1 и абзац p, по отношению к тегу body, в схеме HTML-документа расположенной в начале этой статьи.

Закрывающий и открывающий теги одного элемента, могут находится либо на одном уровне, как например теги <body> </body>, либо закрывающий тег может находится в конце текста, как например закрывающие теги элементов title, h1 и p.

Равнозначные между собой элементы тоже можно размещать на одном уровне, в схеме HTML-документа что расположена выше, равнозначными по отношению друг к другу, являются head и body, h1 и p.

На самом деле, весь HTML-код можно записать в одну строку и браузер всё равно правильно покажет HTML-страницу. Правила синтаксиса языка HTML, где теги нужно записывать друг под другом и левее, существуют лишь для того, чтобы вебмастеру было удобнее создавать и изменять в дальнейшем код HTML-страницы.

Вот еще некоторые моменты, которые нужно учитывать при создании кода:

Сколько бы вы не поставили пробелов в текстовом редакторе, браузер покажет их как один пробел.

Переносы строк и табуляции в текстовом редакторе, не распознаются браузером.

Если вам нужно перенести строку, которая должна быть видна на HTML-странице, то используйте тег <br>.

Если вам нужна табуляция (например для создания "красной" строки), которая должна быть видна на HTML-странице, то используйте либо несколько спецсимволов неразрывного пробела   либо CSS-свойство text-indent.

Лекция 1.6 Знакомство с HTML. Синтаксис языка

HTML — язык разметки гипертекста. Гипертекст — информационная структура, позволяющая устанавливать смысловые связи между элементами текста на экране компьютера таким образом, чтобы можно было легко осуществлять переходы от одного элемента к другому. На практике в гипертексте некоторые слова выделяют путем подчеркивания или окрашивания в другой цвет (гиперссылки). Выделение слова говорит о наличии связи этого слова с некоторым документом, в котором тема, связанная с выделенным словом рассматривается более подробно.

Такие страницы как правило имеют расширение htm или html. Отдельный документ, выполненный в формате HTML, называется:

HTML-документом;

Web-документом;

Web-страницей.

Гиперссылка — фрагмент текста, который является указателем на другой файл или объект. Гиперссылки необходимы для того, чтобы обеспечить возможность перехода от одного документа к другому.

Группа Web-страниц, принадлежащих одному автору или одному издателю и взаимосвязанных общими гиперссылками, образует структуру, которая называется Web-узлом или Web-сайтом. Каждая HTML-страница имеет свой уникальный URL-адрес в Интернете.

Структура HTML-доукмента

Элемент — конструкция языка HTML. Это контейнер, содержащий данные и позволяющий отформатировать их определенным образом. Любая Web-страница представляет собой набор элементов. Одна из основных идей гипертекста возможность вложения элементов.

Тег — начальный или конечный маркеры элемента. Теги определяют границы действия элементов и отделяют элементы друг от друга. В тексте Web-страницы теги заключаются в угловые скобки, а конечный тег всегда снабжается косой чертой. Например: элемент, содержащий некоторый текст, ограничен начальным тегом (маркером) и конечным тегом (маркером) . Т.е. текст помещен между тегами как в контейнер. Здесь же можно увидеть, как осуществляется возможность вложения элементов. Тег вложен внутрь тега , поэтому конечный тег стоит перед . В данном примере тег указывает на то, что текст является отдельным абзацем, а тег задает, например, формат шрифта.

 Этот текст будет расположен в отдельном абзаце и выполнен зеленым цветом шрифта.

В результате такого форматирования на экране компьютера мы увидим текст зеленого цвета в отдельном абзаце.

Атрибут — параметр или свойство элемента. Это, по сути, переменная, которая имеет стандартное имя и которой может присваиваться определенный набор значений: стандартный или произвольный. Атрибуты располагаются внутри начального тега и отделяются друг от друга пробелами.

 Этот текст будет выровнен по центру экрана . В данном примере атрибут align (выравнивание) расположен внутри тега , следовательно он задает выравнивание этого абзаца. Значение атрибута равно "center", т.е. выравнивание абзаца будет по центру экрана.

Ниже приведена структура типичного Web-документа.

<HTML>	Этот тег указывает на начало HTML-документа
<HEAD>	Этот тег указывает на начало области заголовка Web-страницы. Служит для формирования общей структуры документа.
<TITLE>Название Web-страницы</title>	Элемент для размещения заголовка Web-страницы. Строка отображается в заголовке окна браузера.
<META http-equiv="Content-Type" content="text/html; charset=windows-1251">	Этот тег несет служебную информацию и не отображается на экране браузера. В данном случае идет речь о кодировке Web-страницы. Вам достаточно лишь каждый раз вставлять этот тег в таком виде на свою страничку. Тогда ваша страничка будет использовать кодировку windows-1251, наиболее распространенную на сегодняшний день.
<META name="Author" content="Ivanov Ivan">	Имя автора Web-страницы.
<META name="Keywords" content="WWW, HTML, docoment, страничка, структура">	Набор ключевых слов для поиска. Раньше этими словами пользовались поисковые машины, для отбора сайтов по тематике запроса. Сегодня эти слова поисковыми машинами практически не используются, однако полезно вставить этот тег на свою страничку и указать в нем ключевые слова, отражающие содержание вашего сайта.
</head>	Конец области заголовка Web-страницы.
<BODY bgcolor="blue">	Начало собственно содержимого Web-страницы. Тег <BODY> включает в себя атрибут bgcolor, который задает цвет вашей страницы. В данном случае голубой. Если не использовать

	этот атрибут, то по умолчанию цвет страницы будет белым.
<H2> Здесь расположен заголовок вашей странички </h2>	<H2> </h2> Элемент заголовка
<P> Здесь расположен текст первого абзаца вашей странички</p>	<P> </p> Элемент абзаца.
<P> Здесь расположен текст второго абзаца вашей странички</p>	
</body>	Конец содержимого Web-страницы.
</html>	Конец HTML-документа.

Теги можно записывать как строчными, так и заглавными латинскими символами.

Правила синтаксиса

1. Взаимное расположение элементов HTML, HEAD, TITLE, BODY должно быть стандартным на любой странице.

```
<HTML>
<HEAD>
<TITLE>.....</title>
</head>
<BODY>
.................
</body>
</html>
```

2. Необходимо всегда использовать конечные теги (не забывать </p>, </h1>, </table> и др.).

3. Не нарушать правила вложения тегов. Правильно: <H1>Заголовок крупный <H2> Заголовок поменьше </h2> </h1>. Не правильно: <H1>Заголовок крупный <H2> Заголовок поменьше </h1> </h2>

4. Любая полезная информация должна находится между начальным и конечным тегами, указывающими ее формат.

5. Все атрибуты располагаются в начальном теге.

Лекция 1.7 HTML: списки (графические форматы, графический объект как ссылка)

Форматирование текста

Элемент	Тег	Атрибуты	Пример
Абзац	<P> </p>	<P align="left"> </p> — выравнивание текста по левому краю экрана. <P align="center> </p> — выравнивание текста по центру экрана. <P align="right"> </p> — выравнивание текста по правому краю экрана. <P align="justify"> </p> — выравнивание текста по ширине страницы.	<P align="center"> Текст этого абзаца выровнен по центру экрана </p> Посмотреть
Принудительный переход на новую строку	 		Уронили мишку на пол. Оторвали мишке лапу. Все равно его не брошу, Потому что он хороший. Посмотреть
Выделение текста полужирным шрифтом	<B> 		Этот текст имеет обычное начертание, <B> а этот выделен полужирным шрифтом . Посмотреть
Выделение текста курсивом	<I> </i>		Этот текст имеет обычное начертание, <I> а этот выделен курсивом</i>. Посмотреть
Определение типа, размера и цвета шрифта.	<FONT> 	<FONT size=3> - абсолютный размер шрифта (возможные значения от 1 до 7).	<FONT size=1> Это шрифт 1 <FONT size=2> Это шрифт 2 <FONT size=3> Это шрифт 3

		<FONT color="blue"> — цвет шрифта <FONT face="arial"> — определение названия шрифта. <FONT size=3 color="blue" face="arial"> — все атрибуты могут быть использованы совместно внутри данного тега.	<FONT size=4> Это шрифт 4 <FONT size=5> Это шрифт 5 <FONT size=6> Это шрифт 6 <FONT size=7> Это шрифт 7 <FONT color="blue"> Это шрифт синего цвета <FONT face="arial" size=3 color="blue" > Это шрифт arial размером 3, цвет синий. Посмотреть
Цитата	<BLOCKQUOTE> </blockquote>		Это обычный текст абзаца. <BLOCKQUOTE> А это текст цитаты. </blockquote> А это снова обычный текст. Посмотреть
Маркированный список	<UL> <LI> <LI> <LI> 	Имеет атрибут type, задающий вид маркера в списке: “disc”, “square”, “circle” (по умолчанию “disc”). А каждый элемент списка определяется тегом <LI>	<UL> <LI> Первый пункт списка; <LI> Второй пункт списка; <LI> Третий пункт списка. Посмотреть
Нумерованный список	<OL> <LI> <LI> <LI> 	Имеет атрибут type, задающий тип нумерации: арабские цифры, римские цифры, буквы (по умолчанию арабские цифры). Каждый элемент списка определяется тегом <LI>	<OL> <LI> Первый пункт списка; <LI> Второй пункт списка; <LI> Третий пункт списка. Посмотреть

Управление цветом

Кодирование цвета используется для раскрашивания шрифтов, горизонтальных линий, фона и других элементов страницы. Цвета обозначаются английскими названиями или числовыми шестнадцатеричными кодами.

Стандартные цвета

Аквамарин		aqua	#00FFFF
Белый		white	#FFFFFF
Желтый		yellow	#FFFF00
Зеленый		green	#008000
Золотистый		gold	#FFD700
Индиго		indigo	#4B0080
Каштановый		maroon	#800000
Красный		red	#FF0000
Оливковый		oliv	#808000
Пурпурный		purple	#800080
Светло-зеленый		lime	#00FF00
Серебристый		silver	#C0C0C0
Серый		gray	#808080
Сизый		teal	#008080
Синий		blue	#0000FF
Ультрамарин		navy	#000080
Фиолетовый		violet	#EE80EE
Фуксиновый		fuchsia	#FF00FF
Черный		black	#000000

Градации красного

Код	Цвет	Код	Цвет
#010000		#800000	
#100000		#900000	
#200000		#A00000	
#300000		#B00000	
#400000		#C00000	
#500000		#D00000	
#600000		#E00000	
#700000		#FF0000	

Градации зеленого

Код	Цвет	Код	Цвет
#000100		#008000	
#001000		#009000	
#002000		#00A000	
#003000		#00B000	
#004000		#00C000	
#005000		#00D000	

#006000		#00E000	
#007000		#00FF00	

Градации синего

Код	Цвет	Код	Цвет
#000001		#000080	
#000010		#000090	
#000020		#0000A0	
#000030		#0000B0	
#000040		#0000C0	
#000050		#0000D0	
#000060		#0000E0	
#000070		#0000FF	

Градации оранжевого

Код	Цвет
#FFB000	1
#FFA800	2
#FFA000	3
#FF9800	4
#FF9000	5
#FF8800	6
#FF8000	7
#FF7800	8
#FF7000	9
#FF6800	10
#FF6000	11
#FF5800	12

Компьютерная радуга:

К
О
Ж
З
Г
С
Ф

Использование цвета при оформлении страницы

Цвет шрифта можно задать с помощью атрибута color в теге <FONT>, например:

<FONT color="FF5800"> Это цветной текст 1

<FONT color="blue"> Это цветной текст 2

Чтобы задать цвет фона страницы используется атрибут color внутри тега <BODY>, например:

<BODY color=" red">

Списки

Тэги списков

Существует три основных вида списков в HTML-документе:

пронуменрованный;

непронуменрованный;

список описаний;

Вы можете создавать вложенные списки, используя различные тэги списков или повторяя одни внутри других. Для этого просто необходимо разместить одну пару тэгов (стартовый и завершающий) внутри другой. Будут ли элементы вложенного списка иметь те же маркеры, обозначающие элемент списка — зависит от браузера. Более подробно смотри в разделе "Вложенные списки".

Пронумерованные списки

В пронумерованном списке браузер автоматически вставляет номера элементов по порядку. Это означает, что если Вы удалите один или несколько элементов пронумерованного списка, то остальные номера автоматически будут пересчитаны.

Пронумерованный список начинается стартовым тэгом <OL> и завершается тэгом </OL>. Каждый элемент списка начинается с тэга <LI> и заканчивается тегом </LI>. Например:

```
<OL>
   <LI>Программирование
   <LI>Алгоритмизация
   <LI>Проектирование
</OL>
```

Тэг <OL> может иметь параметры:

<OL TYPE=A|a|I|i|1 START=n> где: TYPE - вид счетчика:

A — большие латинские буквы (A,B,C...)

a — маленькие латинские буквы (a,b,c...)

I — большие римские цифры (I,II,III...)

i — маленькие римские цифры (i,ii,iii...)

1 — обычные цифры (1,2,3...)

START=n - число, с которого начинается отсчет. Например:

```
<OL TYPE=I START=15>
   <LI> Программирование
   <LI> Алгоритмизация
   <LI> Проектирование
</OL>
```

Непронумерованные списки.

Для непронумерованных списков браузер обычно использует маркеры для пометки элемента списка. Вид маркера, как правило, настраивает пользователь браузера.

Список начинается стартовым тэгом <UL> и завершается тэгом </UL>. Каждый элемент списка начинается с тэга <LI>. Например:

```
<UL>
   <LI>Программирование
   <LI>Алгоритмизация
   <LI>Проектирование
</UL>
```

Тэг <UL> может иметь параметр: TYPE=disc|circle|square>. Тип тэга <UL> определяет внешний вид маркера — по умолчанию (disc), круглый (circle) или квадратный (square). Например:

```
<UL TYPE=square>
   <LI>Программирование
   <LI>Алгоритмизация
   <LI>Проектирование
</UL>
```

Вложенные списки

Дадим пример вложенных списков:

```
<HTML>
   <HEAD>
      <TITLE> Список сотрудников </TITLE>
   </HEAD>
   <BODY>
         <H2> Список сотрудников нашей фирмы </H2>
         <H3> Составлено : 30 июля 2006 года </H3>
<p>Данный список содержит фамилии, имена и отчества всех сотрудников нашей компании. <P>
<p>Список может быть использован только в служебных целях. <P>
      <OL>
         <LI> Дирекция
            <UL>
               <LI> Иванов И.И.
               <LI> Петров К.В.
            </UL>
         </LI>
         <LI> Отдел маркетинга
            <UL>
               <LI> Варшавская Е.Л.
               <LI> Самсонов Д.М.
            </UL>
         </LI>
      </OL>
   </BODY>
</HTML>
```

Элемент списка <LI>

Тэг <UL> может иметь параметры: TYPE=disc|circle|squade> или <OL TYPE=A|a|I|i|1 VALUE=n>, в зависимости от того, в списке какого вида находится данный элемент.

Таким образом атрибут TYPE определяет вид маркера (см. <UL>) или счетчика (см.OL), а VALUE=n — значение для элемента пронумерованного списка (его номер). Все дальнейшие номера элементов списка будут отсчитываться от этого номера, а каждый элемент списка задаётся тегом <LI>. Например:

```
<OL TYPE=I START=15>
   <LI> Программирование
   <LI TYPE=i VALUE=25> Алгоритмизация
   <LI> Проектирование
</OL>
```

Программирование
Алгоритмизация
Проектирование

Список определений

Список определений начинается с тэга <DL> и завершается тэгом </DL>. Данный список служит для создание списков типа "термин"-"описание". Каждый термин начинается тэгом <DT> , а описание — тэгом <DD>. Например:

```
<DL>
   <DT> <B> Отдел маркетинга </B>
      <DD> Данный отдел занимается продвижением продуктов и услуг
   <DT> <B> Финансовый отдел </B>
      <DD> Данный отдел занимается всеми финансовыми операциями
   <DT> <B> Отдел кадров </B>
      <DD> Данный отдел занимается учетом и набором новых сотрудников фирмы, распределением отпусков, отслеживанием больничных листов и т.д.
</DL>
```

Лекция 1.8 HTML: таблицы, фреймы. Общие подходы к дизайну сайта. Разработка макета страницы

Как правило, веб-страница состоит из множества различных элементов, которые могут иметь сложную структуру. Поэтому при создании веб-страницы возникает необходимость нужным образом позиционировать эти элементы, стилизовать их так, чтобы они располагались на странице нужным образом. То есть возникает вопрос создания макета страницы, ее верстки.

Существуют различные способы, стратегии и виды верстки. Изначально распространенной была верстка на основе таблиц. Так как таблицы позволяет при необходимости очень легко и просто разделить все пространство веб-страницы на строки и столбцы. Строками и столбцами довольно легко управлять, в них легко позиционировать любое содержимое. Именно это и определило популярность табличной верстки.

Однако табличная верстка создает не самые гибкие по дизайну страницы, что является особенно актуальным аспектом в мире, где нет одного единственного разрешения экрана, за то есть большие экраны на телевизорах, малые экраны на планшетах и фаблетах, очень маленькие экраны на смартфонах и т.д. Все это многообразие экранов табличная верстка оказалась не в состоянии удовлетворить. Поэтому постепенно ей на смену пришла блочная верстка. Блочная верстка - это отосительно условное название способов и приемов верстки, когда в большинстве веб-страниц для разметки используется CSS-свойство **float**, а основным строительным элементов веб-страниц является элемент ** **, то есть по сути блок. Используя свойство float и элементы div или другие элементы, можно создать структуру страницы из нескольких столбцов, как при табличной верстке, которая будет значительно гибче.

Ранее в одной из прошлых тем рассматривалось действие свойства float. Теперь используем его для создания двухколоночной веб-страницы. Допустим, вверху и внизу у нас будут стандартно шапка и футер, а в центре - две колонки: колонка с меню или сайдбар и колонка с основным содержимым.

В начале определим все блоки. При работе с элементами, которые используют обтекание и свойство float, важен их порядок. Так, код плавающего элемента, у которого устанавливается свойство float, должен идти перед элементом, который обтекает плавающий элемент. То есть блок сайдбара будет идти до блока основного содержимого:

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="utf-8">
5       <title>Блочная верстка в HTML5</title>
6       <style>
7         div{
8           margin: 10px;
9           border: 1px solid black;
10          font-size: 20px;
11          height: 80px;
12        }
13        #header{
14          background-color: #ccc;
```

```
15                }
16                #sidebar{
17                   background-color: #ddd;
18                }
19                #main{
20                   background-color: #eee;
21                   height: 200px;
3                 }
                  #footer{
                     background-color: #ccc;
                  }
               </style>
            </head>
            <body>
               <div id="header">Шапка сайта</div>
               <div id="sidebar">Сайбар</div>
               <div id="main">Основное содержимое</div>
               <div id="footer">Футер</div>
            </body>
         </html>
```

То есть пока получается примерно следующая страница:

Высота, граница и отступы блоков в данном случае добавлены только для красоты, чтобы идентифицировать пространство блока и отделять его от других.

Далее, чтобы переместить блок сайдбара влево по отношению к блоку основного содержимого и получить эффект обтекания, нам надо указать у блока сайдбара свойство float: left и предпочтительную ширину. Ширина может быть фиксированной, например, 150 px или 8 em. Либо также можно использовать проценты, например, 30% - 30% от ширины контейнера body. С одной стороны, блоками с фиксированной шириной легче управлять, но с другой процентные значения ширины позволяют создавать более гибкие, резиновые блоки, которые изменяют размеры при изменении размеров окна браузера.

Последним шагом является установка отступа блока с основным содержимым от блока сайдбара. Поскольку при обтекании обтекающий блок может обтекать плавающий элемент и справа и снизу, если плавающий элемент имеет меньшую высоту, то нам надо установить отступ, как минимум равный ширине плавающего элемента. Например, если ширина сайдбара равна 150px, то для блока основного содержимого можно задать отступ в 170px, что позволит создать пустое пространство между двумя блоками.

При этом не стоит у блока основного содержимого указывать явным образом ширину, так как браузеры расширяют его автоматически, чтобы он занимал все доступное место.

Итак, принимая во внимание все выше сказанное, изменим стили блоков сайдбара и основного содержимого следующим образом:

```
1        #sidebar{
2           background-color: #ddd;
3           float: left;
```

```
4           width: 150px;
5       }
6       #main{
7           background-color: #eee;
8           height: 200px;
9           margin-left: 170px; /* 150px (ширина сайдбара) + 10px + 10px (2 отступа) */
1       }
```

В итоге у нас получится сайдбар по левую сторону от основного блока:

Высота блоков в данном случае указана условно для большей наглядности, в реальности, как правило, высоту будет автоматически устанавливать браузер.

Создание правого сайдбара будет аналогично, только теперь нам надо установить у сайдбара значение float: right, а у блока основного содержимого - отступ справа:

```
1       #sidebar{
2           background-color: #ddd;
3           float: right;
4           width: 150px;
5       }
6       #main{
7           background-color: #eee;
8           height: 200px;
9           margin-right: 170px;
1       }
```

При этом разметка html остается такой же, блок сайдбара по прежнему должен предшествовать блоку основного содержимого.

Лекция 1.9 HTML: формы

HTML формы — сложные элементы интерфейса. Они включают в себя разные функциональные элементы: поля ввода <input> и <textarea>, списки <select>, подсказки и т.д. Весь код формы заключается в элемент <form>.

Большая часть информации веб-форм передаётся с помощью элемента <input>. Для ввода одной строки текста применяется элемент <input type="text">, для нескольких строк — элемент <textarea>. Элемент <select> создает выпадающий список.

Элемент <label> создаёт надписи к полям формы. Существует два способа группировки надписи и поля. Если поле находится внутри элемента <label>, то атрибут for указывать не нужно.

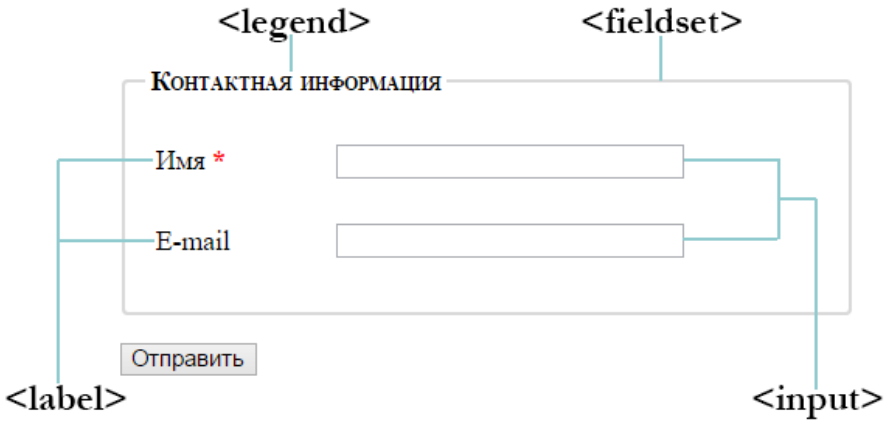
```
<label for="lastname">Last Name</label><input type="text" id="lastname">

<input type="text" id="lastname"><label for="lastname">Last Name</label>

<label>Last Name<input type="text" name="lastname"></label>
```

HTML

Поля формы можно разделять на логические блоки с помощью элемента <fieldset>. Каждому разделу можно присвоить название с помощью элемента <legend>.

```
<fieldset>
 <legend>Контактная информация</legend>
 <label>Имя<input type="text" required></label>
 <label>E-mail<input type="email" required></label>
</fieldset>
```

HTML



ГРУППИРОВКА ПОЛЕЙ ФОРМЫ

Чтобы сделать форму более понятной для пользователей, в поля формы добавляют текст, содержащий пример вводимых данных. Такой текст называется подстановочным и создаётся с помощью атрибута placeholder.

Обязательные для заполнения поля также необходимо выделять. До появления HTML5 использовался символ звездочки `*`, установленный возле названия поля. В новой спецификации появился специальный атрибут `required`, который позволяет отметить обязательное поле на уровне разметки. Этот атрибут дает указание браузеру (при условии, что тот поддерживает HTML5), указание не отправлять данные после нажатия пользователем кнопки отправить, пока указанные поля не заполнены.

```
<input type="text" required placeholder="Ваше имя">
```

HTML

Для изменения внешний вид текстового поля при получении фокуса, используется псевдокласс `focus`. Например, можно сделать фон текущего поля более темным или добавить цветную рамку, чтобы оно выделялось среди остальных:

```
input:focus {
background: #eaeaea;
}
```

CSS

Ещё один полезный html5-атрибут — атрибут `autofocus`. Он позволяет автоматически установить фокус на нужном начальном поле для элементов `<input>` и `<textarea>` (только в один элемент каждой формы).

Пример создания формы регистрации

```
<div class="form-wrap">
  <div class="profile"><img src="https://html5book.ru/wp-content/uploads/2016/10/profile-image.png">
    <h1>Регистрация</h1>
  </div>
  <form method="post" action="form.php">
    <div>
      <label for="name">Имя</label>
      <input type="text" name="name" required>
    </div>
    <div class="radio">
      <span>Пол</span>
      <label>
        <input type="radio" name="sex" value="мужской">мужской
        <div class="radio-control male"></div>
      </label>
      <label>
        <input type="radio" name="sex" value="женский">женский
        <div class="radio-control female"></div>
      </label>
    </div>
    <div>
      <label for="email">E-mail</label>
      <input type="email" name="email" required>
    </div>
    <div>
```

```
      <label for="country">Страна</label>
      <select name="country">
        <option>Выберите страну проживания</option>
        <option value="Россия">Россия</option>
        <option value="Украина">Украина</option>
        <option value="Беларусь">Беларусь</option>
      </select>
      <div class="select-arrow"></div>
    </div>
    <button type="submit">Отправить</button>
  </form>
</div>
```

HTML

Примечание

`action="form.php"` — ссылка на файл обработчика формы. Создайте файл в кодировке UTF-8, закачайте его на сервер и замените `action="form.php"` на путь к файлу на вашем сервере.

Лекция 1.10 Знакомство с CSS (каскадные таблицы стилей).

Использование стиля при оформлении сайта. Спецификации CSS1, CSS2

CSS является сложным языком, который отнимает совсем немного энергии. Он позволяет добавить макет и дизайн для наших страниц и обмениваться стилями от элемента к элементу и страницы к странице. Прежде чем мы сможем раскрыть все особенности CSS, есть несколько аспектов языка, которые вы должны в полной мере понимать.

Первое что важно знать, как именно отображаются стили. В частности, мы должны знать, как работают разные типы селекторов и как порядок этих селекторов может повлиять на отображение стилей. Мы также хотим понимать несколько основных значений свойств, которые постоянно появляются в CSS, в частности те, которые касаются цвета и размера.

Давайте заглянем под капот CSS, чтобы точно узнать что происходит.

Каскад

Мы начнём разбирать, как именно отображаются стили со взгляда на то, что известно как каскад и изучения несколько примеров каскада в действии. В CSS все стили идут каскадом сверху вниз, что позволяет добавлять другой стиль или переписывать его, тем самым таблицы стилей развиваются.

Скажем, к примеру, что мы выбрали все элементы абзаца в верхней части нашего стиля и установили для них цвет фона orange и шрифт размером 24 пикселя. Затем в нижней части нашего стиля мы снова выбираем все элементы абзаца и устанавливаем для них цвет фона green, как показано здесь.

```
p {
  background: orange;
  font-size: 24px;
}
p {
  background: green;
}
```

Поскольку селектор абзаца, который устанавливает зелёный цвет фона располагается после селектора абзаца, который задаёт оранжевый цвет фона, он будет иметь приоритет в каскаде. Все абзацы появятся на зелёном фоне. Размер шрифта останется 24 пикселя, потому что второй селектор абзаца не определил новый размер шрифта.

Каскадные свойства

Каскад работает со свойствами внутри отдельных селекторов. Опять же, скажем, к примеру, что мы выбрали все элементы абзаца и установили для них цвет фона orange. Затем прямо ниже свойства background и его значения мы добавляем ещё одно свойство и значение, которое задаёт цвет фона green, как показано здесь.

```
p {
  background: orange;
  background: green;
}
```

Поскольку объявление зелёного цвета фона написано после объявления оранжевого цвета фона, как и прежде, наши абзацы будут отображаться на зелёном фоне.

Все стили идут каскадом сверху нашей таблицы стилей до её низа. Есть, однако, случаи, когда каскад не так хорошо работает — порой, когда применяются различные типы селекторов и специфичность этих селекторов разбивает каскад. Давайте взглянем.

Вычисление специфичности

Каждый селектор в CSS имеет вес специфичности, он вместе с положением в каскаде определяет, как будут отображаться стили.

В уроке 1 «Создание первой веб-страницы» мы говорили о трёх разных видах селекторов: селектор типа, класс и идентификатор. Каждый из этих селекторов имеет различный вес специфичности.

У селектора типа низкий вес специфичности и значение балла 0-0-1. У селектора класса средний вес специфичности и значение балла 0-1-0. Наконец, у идентификаторов высокая специфичность и значение балла 1-0-0. Как мы видим, баллы специфичности вычисляются с помощью трёх колонок. В первой колонке количество идентификаторов, во второй классов, а третья колонка считает селекторы типа.

Важно отметить, что идентификатор имеет больший вес специфичности чем селектор класса, а класс больший вес, чем селектор типа.

Баллы специфичности

Баллы специфичности намеренно разделены дефисом, так как их значения не вычисляются по десятичной системе. У селекторов класса нет 10 баллов, у идентификаторов нет 100 баллов. Вместо этого эти баллы следует читать как 0-1-0 и 1-0-0 соответственно. Мы внимательно рассмотрим, почему эти значения пишутся через дефис в ближайшее время, когда мы станем комбинировать селекторы.

Чем выше вес специфичности селектора, тем больше первенства ему отдаётся при возникновении конфликта стилей. Например, если элемент абзаца выбирается с помощью селектора типа в одном месте и идентификатора в другом, то идентификатор будет иметь приоритет над селектором типа, независимо от того, где идентификатор появляется в каскаде.

HTML

```
<p id="food">...</p>
```

CSS

```
#food {
  background: green;
}
p {
  background: orange;
}
```

Здесь у нас есть элемент абзаца со значением атрибута id — food. В нашем CSS этот абзац выбирается двумя различными типами селекторов: один селектор типа, а второй идентификатор. Несмотря на то, что селектор типа указан после идентификатора в каскаде, идентификатор имеет приоритет над селектором типа, потому что он имеет больший вес специфичности, следовательно, абзац появится на зелёном фоне.

Специфичность веса разных типов селекторов невероятно важно помнить. Порой стили не могут появиться на элементах, как предполагалось, вероятно оттого, что специфичность веса наших селекторов нарушила каскад, поэтому наши стили не отображаются должным образом.

Понимание того, как каскад и специфичность работают — это огромное затруднение и мы будем продолжать освещать эту тему. А сейчас давайте посмотрим на то, как стать немного конкретнее и обоснованными с нашими селекторами, путём их

комбинации. Имейте в виду, что при комбинации селекторов, мы также меняем их специфичность.

Комбинация селекторов

Пока мы рассмотрели как использовать разные типы селекторов индивидуально, но мы также должны знать, как использовать эти селекторы вместе. Комбинируя селекторы мы можем быть более конкретными в том, какой элемент или группу элементов мы хотели бы выбрать.

Скажем, к примеру, мы хотим выбрать все элементы абзаца, которые находятся внутри элемента со значением атрибута класса hotdog и установить для них цвет фона как brown. Однако, если один из этих абзацев, случаем, содержит значение атрибута класса mustard, мы хотим установить его цвет фона как yellow. Наши HTML и CSS могут выглядеть следующим образом:

HTML

```
<div class="hotdog">
  <p>...</p>
  <p>...</p>
  <p class="mustard">...</p>
</div>
```

CSS

```
.hotdog p {
  background: brown;
}
.hotdog p.mustard {
  background: yellow;
}
```

Когда селекторы комбинируются они должны читаться справа налево. Самый крайний селектор справа, непосредственно перед открытой скобкой, известен как ключевой селектор. Он определяет, к каким именно элементам будут применяться стили. Любой селектор слева от ключевого будет служить уточнением.

Первый комбинированный селектор выше, .hotdog p, включает в себя два селектора: класс и селектор типа. Эти два селектора разделяются пробелом. Ключевым селектором выступает селектор типа, нацеленный на элементы абзаца. Поскольку этот селектор сочетается с классом hotdog, полный комбинированный селектор выбирает только элементы абзаца, которые находятся внутри элемента с классом hotdog.

Второй селектор выше, .hotdog p.mustard, включает в себя три селектора: два класса и один селектор типа. Единственное различие между вторым и первым селекторами является добавление класса mustard в конце селектора абзаца. Поскольку новый класс mustard находится в правой части комбинированного селектора, то он ключевой, а все отдельные селекторы идущие перед ним теперь уточняющие.

Пробелы в селекторах

В предыдущем комбинированном селекторе, .hotdog p.mustard, есть пробел между классом hotdog и селектором абзаца, но не между селектором абзаца и классом mustard. Использование пробелов и отказ от них — это большая разница в селекторах.

Поскольку нет пробела между селектором абзаца и классом mustard, это значит что будут выбраны только абзацы с классом mustard. Если бы селектор абзаца был удалён, а класс mustard содержал пробелы с двух сторон, то был бы выбран любой элемент с классом mustard, а не только абзацы.

Лучше всего не писать селектор типа перед селектором класса. Как правило, мы хотим выбрать любой элемент с данным классом, а не только один тип элемента. С учётом этого наш новый комбинированный селектор будет лучше писать как .hotdog .mustard.

Читая комбинированный селектор справа налево — он нацелен на абзацы со значением атрибута класса mustard, который располагается внутри элемента с значением атрибута класса hotdog.

Разные типы селекторов могут комбинироваться, чтобы обнаружить любой конкретный элемент на странице. Поскольку мы продолжим писать различные комбинированные селекторы, то увидим их мощь в жизни. Прежде, чем мы это сделаем, давайте взглянем на то, как меняется вес специфичности комбинированных селекторов.

Специфичность в комбинированных селекторах

Вес специфичности комбинированных селекторов может быть вычислен путём подсчёта каждого отдельного типа селектора в их комбинации.

Взглянем на наш комбинированный селектор выше. Первый селектор, .hotdog p, содержит селектор класса и селектор типа. Зная, что баллы класса это 0-1-0, а баллы селектора типа это 0-0-1, суммарные комбинированные баллы будут 0-1-1, это определяется путём суммирования каждого вида селектора.

Второй селектор, .hotdog p.mustard, содержит два селектора класса и один селектор типа. У комбинированного селектора будут баллы 0-2-1. 0 в первой колонке показывает нулевое число идентификаторов, 2 во второй колонке — два селектора класса, а 1 в последней колонке — один селектор типа.

Сравнивая два селектора, у второго селектора с двумя классами заметно более высокое значение веса специфичности и баллов. Как таковой, он будет иметь приоритет в каскаде. Если бы мы перевернули порядок этих селекторов в нашей таблице стилей, поместив более «тяжёлый» селектор выше «лёгкого» селектора, как показано здесь, вывод их стилей не будет затронут, в силу специфичности веса каждого селектора.

```
.hotdog p.mustard {
  background: yellow;
}
.hotdog p {
  background: brown;
}
```

В общем, мы хотим, чтобы вы всегда держали вес специфичности селекторов в поле зрения. Чем больше растёт вес специфичности, тем более вероятно, что наш каскад сломается.

Разделение стилей по нескольким классам

Одним из способов сохранить низкими веса специфичности наших селекторов является при возможности модульность, передача похожих стилей от элемента к элементу. Один из вариантов модульности — разделение на разные стили с помощью нескольких классов.

Элементы в HTML могут содержать более одного атрибута class, при этом их значения разделяются пробелами. За счёт этого мы можем применить некоторые стили ко всем элементам одного вида, а другие стили к конкретным элементам этого же вида.

Мы можем связать стили, которые хотим постоянно повторять с одним классом и разделить на дополнительные стили с другим классом.

Давайте взглянем на кнопки, к примеру. Скажем, мы хотим, чтобы у всех наших кнопок был размер шрифта 16 пикселей, но чтобы цвет фона кнопок мог варьироваться

в зависимости от того, где кнопки применяются. Мы можем создать несколько классов и распределить их по элементам, в зависимости от применения желаемых стилей.

HTML

```
<a class="btn btn-danger">...</a>
<a class="btn btn-success">...</a>
```

CSS

```
.btn {
  font-size: 16px;
}
.btn-danger {
  background: red;
}
.btn-success {
  background: green;
}
```

Здесь вы можете увидеть два элемента ссылок с несколькими значениями атрибутов класса. Первый класс, btn, используется для задания размера шрифта 16 пикселей каждому из элементов. Затем, первый элемент ссылки использует дополнительный класс btn-danger, чтобы применить красный цвет фона, а второй элемент ссылки использует дополнительный класс btn-success, чтобы применять зелёный цвет фона. Наши стили чистые и модульные.

Используя несколько классов мы можем делить стили на множество классов по своему желанию, сохраняя наш код компактным с низким весом специфичности. Понимание каскада и вычисление специфичности требует практики, которая занимает время до полного постижения, но в этом мы становимся с каждым уроком всё лучше.

Основные значения свойств CSS

Мы уже применяли небольшую часть значений основных свойств CSS, такие как значения цвета red и green. Вы, возможно, не слишком задумывались об этом, это нормально. Мы теперь потратим немного времени и перейдём к некоторым ранее используемым значениям свойств, а также исследуем часть основных свойств, которые вскоре будем использовать.

В частности, взглянем на свойства, которые относятся к цветам и размерам.

Цвета

Все цветовые значения в CSS определяются как цветовое пространство sRGB (или стандартный красный, зелёный и синий). Цвета в пределах этого пространства образуются путём смешивания вместе красного, зелёного и синего цветовых каналов, отражая способ, каким телевизоры и мониторы генерируют все разные цвета, которые они показывают. При смешивании различных уровней красного, зелёного и синего мы можем создать миллионы цветов и обнаружить почти любой желаемый цвет.

В настоящее время существует четыре основных способа представления цвета sRGB в CSS: ключевые слова, шестнадцатеричная запись, значения RGB и HSL.

Ключевые цвета

Значения ключевого слова — это названия (такие как red, green или blue), которые отображаются данным цветом. Названия ключевых слов и соответствующие им цвета определяется спецификацией CSS. Наиболее распространённые цвета, наряду с несколькими странностями, связаны с ключевым названием.

Полный список этих названий можно найти в спецификации CSS.

Цвет	Название	Шестн.	RGB	HSL
	black	#000000	rgb(0, 0, 0)	hsl(0, 0
	silver	#c0c0c0	rgb(192, 192, 192)	hsl(0, 0
	gray	#808080	rgb(128, 128, 128)	hsl(0, 0
	white	#ffffff	rgb(255, 255, 255)	hsl(0, 1
	maroon	#800000	rgb(128, 0, 0)	hsl(0, 1
	red	#ff0000	rgb(255, 0, 0)	hsl(0, 1
	purple	#800080	rgb(128, 0, 128)	hsl(300,
	fuchsia	#ff00ff	rgb(255, 0, 255)	hsl(300,
	green	#008000	rgb(0, 128, 0)	hsl(120,
	olive	#808000	rgb(128, 128, 0)	hsl(120,
	lime	#00ff00	rgb(0, 255, 0)	hsl(60,
	yellow	#ffff00	rgb(255, 255, 0)	hsl(60,
	navy	#000080	rgb(0, 0, 128)	hsl(240,
	blue	#0000ff	rgb(0, 0, 255)	hsl(240,
	teal	#008080	rgb(0, 128, 128)	hsl(180,
	aqua	#00ffff	rgb(0, 255, 255)	hsl(180,

Здесь мы применяем фон maroon к любому элементу с классом task и фон yellow к любому элементу с классом count.

```
.task {
  background: maroon;
}
.count {
  background: yellow;
}
```

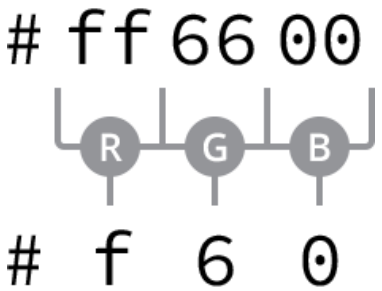
Ключевые цвета просты по своей природе, но их ограниченное количество, поэтому это не самый популярный способ выбора цвета.

Шестнадцатеричные цвета

Шестнадцатеричные значения цвета начинаются с решётки (#), затем идёт три или шесть символов. Используются цифры от 0 до 9 и буквы от A до F, в верхнем или нижнем регистре. Эти значения отображают красный, зелёный и синий цветовые каналы.

В шестисимвольной записи первые два символа представляют красный канал, третий и четвертый символы представляют зелёный канал, а последние две цифры представляют синий канал. В трёхсимвольной записи первый символ представляет красный канал, второй символ представляет зелёный канал, а последний символ представляет синий канал.

Если в шестисимвольной записи первые два символа одинаковы, третий и четвертый символы одинаковы и последние два символа одинаковы, то запись может быть сокращена до трёх символов. Для этого повторяющиеся символы из каждой пары должны использоваться только один раз. Например, оттенок оранжевого представлен шестнадцатеричным цветом #ff6600, который также может быть записан как #f60.



Шестисимвольное значение можно записать в виде трёх символов, когда каждый красный, зелёный и синий цветовые каналы содержат повторяющийся символ

Пары символов получаются путем преобразования чисел от 0 до 255 в шестнадцатеричный формат. Математика здесь немного сложна и достойна отдельной книги, но полезно знать, что 0 равен чёрному, а F равен белому.

Миллионы шестнадцатеричных цветов

Есть миллионы шестнадцатеричных цветов, свыше 16,7 млн, если быть точным. Во как…

Имеется 16 вариантов каждого символа в шестнадцатеричном цвете, от 0 до 9 и от A до F. Группируя символы парами мы получим 256 вариантов цвета (16×16 или 16^2). С тремя группами по 256 вариантов в каждой у нас есть в общей сложности более 16,7 млн цветов (256×256×256 или 256^3).

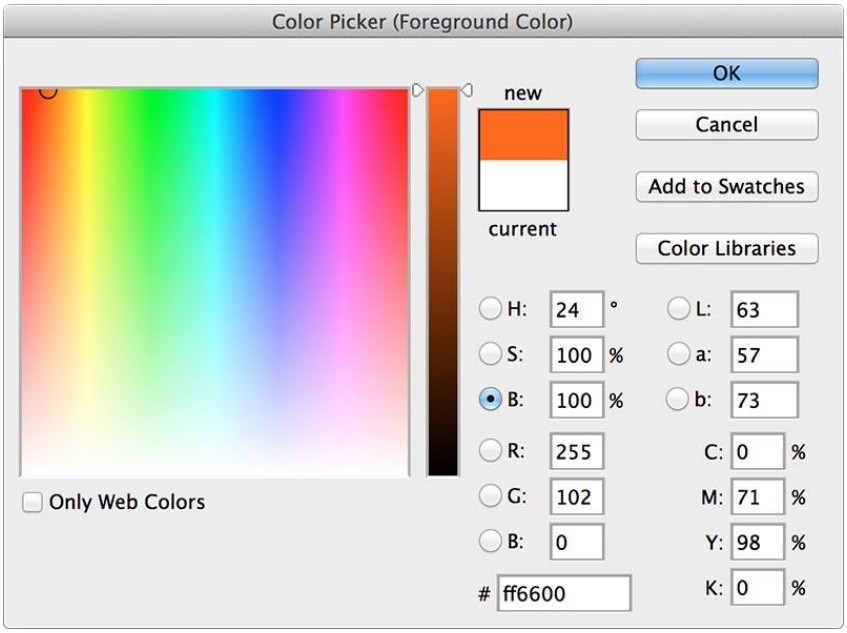
Для создания такого же цвета фона, как maroon и yellow выше, мы могли бы заменить значения ключевых слов на шестнадцатеричные значения цвета, как показано здесь.

```
.task {
  background: #800000;
}
.count {
  background: #ff0;
}
```

Шестнадцатеричные значения цвета были некоторое время повсеместно и стали довольно популярными, потому что предлагают большое число вариантов цвета. С ними, однако, несколько сложно работать, особенно если вы с ними не слишком хорошо знакомы. К счастью Adobe создал Adobe Color

, бесплатное приложение, которое предлагает цветовой круг для помощи в нахождении любого желаемого цвета и соответствующего шестнадцатеричного значения.

Кроме того, большинство приложений для редактирования изображений, такие как Adobe Photoshop, предоставляют возможность установления шестнадцатеричного значения цвета.



Палитра цветов в Adobe Photoshop показывает шестнадцатеричное значение цвета

Цвета RGB и RGBa

Значения цвета RGB задаются с помощью функции rgb(). Функция принимает три значения — красный, зелёный и синий, разделенные запятыми, каждое из которых является целым числом от 0 до 255. Значение 0 будет чисто чёрным, значение 255 будет чисто белым.

Как и следовало ожидать, первое значение в функции rgb() представляет красный канал, второе зелёный канал, а третье значение — синий канал.

Если бы мы хотели переделать оттенок оранжевого в качестве значения цвета RGB, то он будет представлен как rgb(255, 102, 0).

Кроме того, используя те же цвета maroon и yellow, что раньше, мы могли бы заменить ключевые слова или шестнадцатеричные значения на значения RGB.

```
.task {
_background: rgb(128, 0, 0);
}
.count {
_background: rgb(255, 255, 0);
}
```

Значения цвета RGB могут включать в себя альфа-канал или прозрачность с помощью функции rgba(). Эта функция требует четвёртое значение, которое должно быть числом от 0 до 1, включая десятичные знаки. Значение 0 создаёт полностью прозрачный цвет, в том смысле что он будет невидимым, а значение 1 создает полностью непрозрачный цвет. Любое десятичное значение от 0 до 1 создаст полупрозрачный цвет.

Если мы хотим, чтобы наш оранжевый оттенок стал на 50% прозрачным, то могли бы использовать RGBa значение цвета rgba(255, 102, 0, .5).

Мы также можем изменить прозрачность нашего цвета maroon и yellow. Следующий код устанавливает цвет фона maroon на 25% прозрачным и оставляет цвет фона yellow 100% непрозрачным.

```
.task {
_background: rgba(128, 0, 0, .25);
}
```

.count {
background: rgba(255, 255, 0, 1);
}

Значения цвета RGB становятся всё более популярными, особенно в связи с возможностью создания полупрозрачных цветов с помощью RGBa.

Цвета HSL и HSLa

Значения цветов HSL задаются с помощью функции hsl(), для которой устанавливается оттенок, насыщенность и яркость. Внутри скобок функция принимает три значения, разделённых запятыми, подобно rgb().

Первое значение, оттенок, является безразмерным числом от 0 до 360 и символизирует цветовой круг, а значение определяет угол цвета на этом круге.

Второе и третье значения, насыщенность и яркость, являются процентными значениями от 0 до 100%. Значение насыщенности определяет, насколько оттенок является насыщенным, где 0 будет серым, а 100% полностью насыщенным. Яркость определяет, насколько тёмным или светлым будет оттенок, где 0 полностью чёрный, а 100% полностью белый.

Возвращаясь к нашему оранжевому оттенку, значение цвета HSL мы бы записали как hsl(24, 100%, 50%).

Наши цвета фона maroon и yellow также могут быть заданы в HSL, как показано здесь.

.task {
background: hsl(0, 100%, 25%);
}
.count {
background: hsl(60, 100%, 50%);
}

Значение цвета HSL, подобно RGBa, может также включать альфа-канал или прозрачность с помощью функции hsla(). Поведение альфа-канала такое же что и у функции rgba(). Четвёртое значение, которое лежит в пределах между 0 и 1, включая десятичные числа, должно добавляться к функции для определения степени прозрачности.

Наш оттенок оранжевого цвета с прозрачностью 50% будет представлен как hsla(24, 100%, 50%, 0,5).

Тот же самый цвет фона maroon с прозрачностью 25% и на 100% непрозрачный yellow будут выглядеть следующими значениями HSLa.

.task {
background: hsla(0, 100%, 25%, .25);
}
.count {
background: hsla(60, 100%, 50%, 1);
}

Значение цвета HSL является новейшим значением доступным в CSS. Из-за его возраста и поддержки в браузерах он, однако, не так широко применяется.

На данный момент, шестнадцатеричные значения цвета остаются наиболее популярными, поскольку они широко поддерживается. Но когда необходима прозрачность, предпочтительными являются значения цвета RGBa. Эти предпочтения могут измениться в будущем, но сейчас мы будем использовать шестнадцатеричные значения и RGBa.

Размеры

Значения размеров в CSS похожи на цвета в том, что есть несколько различных типов значений для размера, все они служат разным целям. Есть два разных вида размеров — абсолютные и относительные, каждый из которых использует различные единицы измерения.

Мы будем придерживаться наиболее принятых и более простых значений на данный момент, более сложные значения потребуют гораздо больше усилий.

Абсолютные размеры

Абсолютные значения размера являются простейшими значениями, поскольку они привязаны к физическим единицам, таким как дюймы, сантиметры или миллиметры. Самая популярная абсолютная единица измерения известна как пиксель и представлена в виде записи px.

Пиксели

Пиксель равен 1/96 дюйма. Таким образом, в дюйме 96 пикселей. Точное измерение пикселя, тем не менее, может слегка отличаться на устройствах с высокой и низкой плотностью пикселей.

Пиксели существуют достаточно продолжительное время и, как правило, используются набором разных свойств. Этот код использует пиксели, чтобы установить размер шрифта всех абзацев как 14 пикселей.

```
p {
font-size: 14px;
}
```

С учётом смены устройств просмотра на альбомный режим и различных размеров экранов, пиксели потеряли часть своей популярности. В качестве абсолютных единиц измерения они не предлагают достаточной гибкости. Пиксели, однако, надёжны и отлично подходят для начала работы. Мы собираемся опираться на них совсем немного, пока досконально изучаем HTML и CSS.

Относительные размеры

В дополнение к абсолютным значениям размеров есть также относительные размеры. Они немного сложнее, так как не являются фиксированными единицами измерения, а основаны на размере другого измерения.

Проценты

Проценты, представленные в виде обозначения %, являются одним из самых популярных относительных значений. Размеры в процентах определяются по отношению к размеру другого объекта. Например, чтобы установить ширину элемента как 50%, мы должны знать ширину родительского элемента, т. е. элемента, в который он вложен, а затем определить 50% от ширины родительского элемента.

```
.col {
width: 50%;
}
```

Здесь мы установили ширину элемента с классом col как 50%. Эти 50% будут рассчитываться по отношению к ширине родительского элемента.

Проценты являются чрезвычайно полезными для установления высоты и ширины элементов и создания макета веб-страницы. Мы будем полагаться на них часто, чтобы они помогали нам в этих областях.

Em

Единица em также очень популярная относительная величина. em это размер, который вычисляется на основе размера шрифта элемента.

Одна единица em эквивалентна размеру шрифта элемента. Так, например, если для элемента задан размер шрифта 14 пикселей, а width как 5em, то ширина будет равна 70 пикселей (14 пикселей×5).

```
.banner {
_font-size: 14px;
_width: 5em;
}
```

Когда размер шрифта явно не указан для элемента, единица em будет связана с размером шрифта ближайшего родительского элемента с заданным размером шрифта.

em часто применяется для стилизации текста, в том числе размера шрифта, а также пространства вокруг текста, включая margin и padding. Мы изучим текст немного подробнее в Уроке 6 «Работа с типографикой».

Существует намного больше абсолютных и относительных единиц измерения, чем упомянутые. Тем не менее, пиксели, проценты, em — наиболее популярны и их мы собираемся использовать прежде всего.

Резюме

К сожалению, наш сайт Styles Conference проспал этот урок. Мы фокусировались на основах CSS, рассказывающих как именно всё работает, и некоторых основных значениях, которые мы уверенно использовали.

Подводя итоги, в этом уроке мы обсудили следующее:

Каскадирование стилей сверху вниз.

Что такое специфичность и как мы можем её вычислить.

Как комбинировать селекторы для определённых элементов или групп элементов.

Как применять несколько классов к одному элементу для разделения на разные стили для большей модульности.

Разные значения цвета, доступные для использования в CSS, в том числе ключевые слова, шестнадцатеричные значения, RGB и HSL.

Различные значения размера, доступные для использования в CSS, в том числе пиксели, проценты и единицы em.

Лекция 1.11 CSS: работа со списками

В этой статье учебника речь пойдет о работе со списками в CSS, вы научитесь изменять тип маркера, расположение маркера относительно элемента списка, создавать собственные маркеры и даже изменять цвет маркера.

Думаю, вы уже знаете, что в HTML 5 применяется два основных вида списков (если не брать в расчет списки описаний и элементы меню):

нумерованный (упорядоченный) список – HTML элемент **<ol>**

маркированный (неупорядоченные) список – HTML элемент **<ul>**

CSS предоставляет нам широкие возможности форматирования внешнего вида этих списков, давайте рассмотрим основные из них.

Изменение типа маркера

Первое свойство, которое хотелось бы рассмотреть это свойство **list-style-type**

, оно задает тип маркера элемента списка.

Меню навигации, к примеру, часто составляется из обычных маркированных списков (HTML элемент **<ul>**), по умолчанию маркер которых, отображается в форме закрашенного круга, чтобы его убрать необходимо, воспользоваться CSS свойством **list-style-type** со значением **none**:

```
ul {
list-style-type : none; /* убираем маркер у списка */
}
```

Для маркированных (неупорядоченных) списков (HTML элемент **<ul>**) предусмотрено три типа маркеров: в форме закрашенного кружка (**disc** - это значение по умолчанию), маркер в форме полого круга (**circle**) и в форме квадрата (**square**), а для нумерованных (упорядоченных) списков (элемент **<ol>**) все оставшиеся варианты. Полный перечень размещен в следующей таблице:

Значение	**Описание**
none Элемент списка Элемент списка Элемент списка	Маркер не отображается.
disc Элемент списка Элемент списка Элемент списка	Маркер в форме закрашенного кружка. **Это значение по умолчанию.**
armenian Элемент списка Элемент списка Элемент списка	Числовой маркер (традиционная Армянская нумерация).
circle Элемент списка Элемент списка Элемент списка	Маркер в форме круга.
cjk-ideographic Элемент списка	Простые идеографические числа.

Элемент списка Элемент списка	
decimal Элемент списка Элемент списка Элемент списка	Числовой маркер (десятеричные арабские числа, начинающихся с 1).
decimal-leading-zero Элемент списка Элемент списка Элемент списка	Числовой маркер (десятеричные арабские числа, начинающихся с 1 и дополненные начальным нулем - 01, 02, 03...).
georgian Элемент списка Элемент списка Элемент списка	Числовой маркер (традиционная Грузинская нумерация - an, ban, gan, ..., he, tan, in, in-an...).
hebrew Элемент списка Элемент списка Элемент списка	Числовой маркер (традиционная Еврейская нумерация.).
hiragana Элемент списка Элемент списка Элемент списка	Числовой маркер (японская слоговая азбука Хирагана - a, i, u, e, o, ka, ki...).
hiragana-iroha Элемент списка Элемент списка Элемент списка	Числовой маркер (японская слоговая азбука Хирагана ироха - i, ro, ha, ni, ho, he, to, ...).
katakana Элемент списка Элемент списка Элемент списка	Числовой маркер (японская слоговая азбука Катакана - A, I, U, E, O, KA, KI, ...).
katakana-iroha Элемент списка Элемент списка Элемент списка	Числовой маркер (японская слоговая азбука Катакана ироха - I, RO, HA, NI, HO, HE, TO, ...).
lower-alpha Элемент списка Элемент списка Элемент списка	Буквы ascii нижнего регистра (a, b, c, d...z).
lower-greek Элемент списка Элемент списка Элемент списка	Строчные греческие буквы (α, β, γ, δ, и т.д.).

lower-latin Элемент списка Элемент списка Элемент списка	Строчные латинские буквы (a, b, c, d,...z).
lower-roman Элемент списка Элемент списка Элемент списка	Римские числа в нижнем регистре (i, ii, iii, iv, v и т.д.).
square Элемент списка Элемент списка Элемент списка	Маркер в форме квадрата.
upper-alpha Элемент списка Элемент списка Элемент списка	Буквы ascii верхнего регистра (A, B, C, D,...Z).
upper-latin Элемент списка Элемент списка Элемент списка	Заглавные латинские буквы (A, B, C, D,...Z).
upper-roman Элемент списка Элемент списка Элемент списка	Римские цифры в верхнем регистре (I, II, III, IV, V и т.д.).

Обращаю Ваше внимание, что значения hebrew, cjk-ideographic, hiragana, hiragana-iroha, katakana, katakana-iroha не поддерживаются браузером *Internet Explorer*.

Давайте рассмотрим пример использования свойства **list-style-type** в тексте:

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример использования свойства list-style-type</title>
<style>
.test {
list-style-type : lower-roman; /* устанавливаем тип маркера - римские числа в нижнем регистре  */
color : orange; /* обратите внимание, что текст цвета соответствует цвету маркера */
}
```

```
.test2 {
list-style-type : circle; /* устанавливаем тип маркера в форме круга */
color : IndianRed; /* обратите внимание, что текст цвета соответствует цвету маркера */
}
</style>
</head>
<body>
<ul> /* список со значением типа маркера по умолчанию (disc). */
<li>Элемент списка</li>
<li>Элемент списка</li>
<li>Элемент списка</li>
</ul>
<ol class = "test"> /* нумерованный список с типом маркера lower-roman */
<li>Элемент списка</li>
<li>Элемент списка</li>
<li>Элемент списка</li>
</ol>
<ul class = "test2"> /* маркированный список с типом маркера circle */
<li>Элемент списка</li>
<li>Элемент списка</li>
<li>Элемент списка</li>
</ul>
</body>
</html>
```

В данном примере мы создали два стиля, первый устанавливает тип маркера - римские числа в нижнем регистре (значение **lower-roman**), мы его применили к нумерованном списку (HTML элемент **<ol>**), а к маркированному списку (HTML элемент **<ul>**) мы применили стиль, который устанавливает тип маркера в форме полого круга (значение **circle**).

Результат нашего примера:

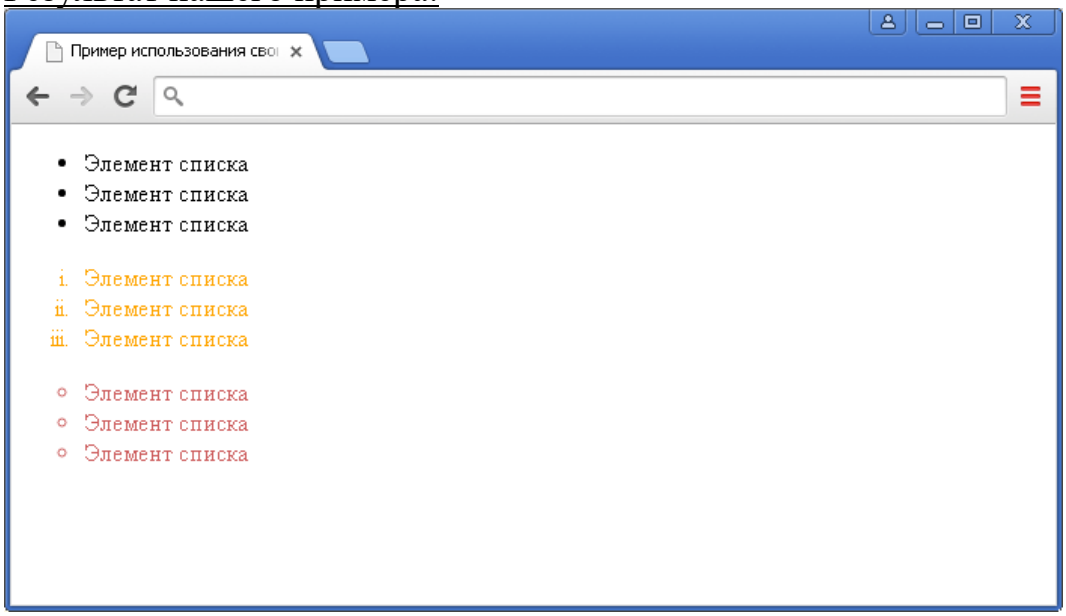


Рис. 1 Пример использования свойства list-style-type (установка типа маркера).

Обращаю Ваше внимание, что вы можете применить свойство **list-style-type** даже к отдельно взятому элементу списка (HTML элемент **<li>**), чтобы установить свой тип маркера, либо создать стили, которые будет применять определённый маркер к чётным, либо нечетным элементам списка, как рассмотрено в следующем примере:

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример чередования стилей свойства list-style-type</title>
<style>
li:nth-child(even) { /* четное чередование стиля */
list-style-type : none;
color : red;
}
li:nth-child(odd) { /* нечетное чередование стиля */
list-style-type : square;
color : green;
}
</style>
</head>
    <body>
        <ul>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
        </ul>
    </body>
</html>
```

В этом примере мы стилизовали все **нечетные** элементы списка - указали тип маркера *квадрат*, а цвет текста указали *зеленый*. **Четные** элементы списка получили следующий стиль: *красный* цвет текста и *отсутствие* маркера.

Результат нашего примера:

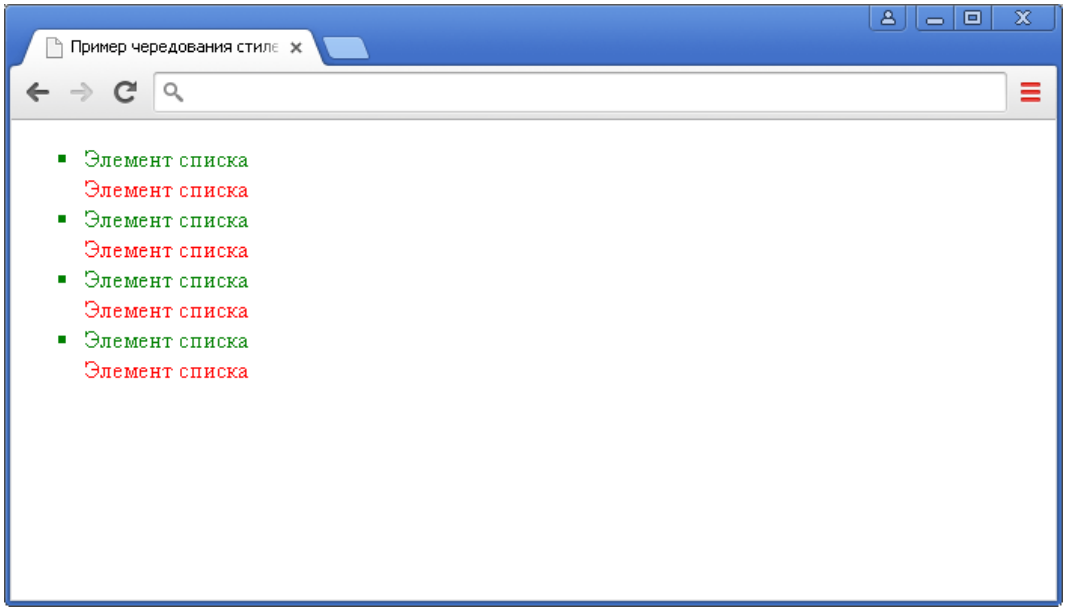


Рис. 2 Пример чередования стилей свойства list-style-type.

Расположение маркера относительно элемента списка

С помощью CSS свойства **list-style-position** вы можете определить расположение маркера, либо цифры относительно элемента списка. Для того, чтобы расположить маркер (цифру) внутри элемента списка вместе с содержимым, то необходимо использовать значение **inside**, а для того, чтобы расположить за границей элемента списка необходимо использовать значение **outside**.

Давайте для наглядности рассмотрим пример, в котором помимимо свойства **list-style-position** используем еще одно новое свойство, которое позволит нам установить границы элемента.

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример использования свойства list-style-position</title>
<style>
li {
border : 1px solid orange; /* устанавливаем сплошную границу размером 1px оранжевого цвета для всех элементов списка*/
}
.test {
list-style-position : outside; /* указываем, что маркер располагается слева от текста за границей элемента списка */
background-color : khaki; /* устанавливаем цвет заднего фона */
}
.test2 {
list-style-position : inside; /* указываем, что маркер располагается слева от текста внутри элемента вместе с содержимым */
background-color : khaki; /* устанавливаем цвет заднего фона */
}
</style>
</head>
    <body>
        <ul class = "test">
            <li>Элемент списка</li>
```

```
                <li>Элемент списка</li>
                <li>Элемент списка</li>
            </ul>
            <ol class = "test2">
                <li>Элемент списка</li>
                <li>Элемент списка</li>
                <li>Элемент списка</li>
            </ol>
        </body>
</html>
```

В данном примере для маркированного списка (HTML элемент <ul>) мы расположили маркер внутри элемента списка вместе с содержимым, использовав свойство **list-style-position** со значение **outside**, а для нумерованного списка (HTML элемент <ol>) разместили цифру слева от текста внутри элемента вместе с содержимым (свойство **list-style-position** со значение **inside**).

Универсальное свойство **border**, которое позволяет установить границу для всех элементов списка, мы использовали для лучшего понимания работы свойства **list-style-position**. Работа с границами элементов в скором времени будет подробно рассмотрена в учебнике в статье "**Границы элемента в CSS**".

Результат нашего примера:

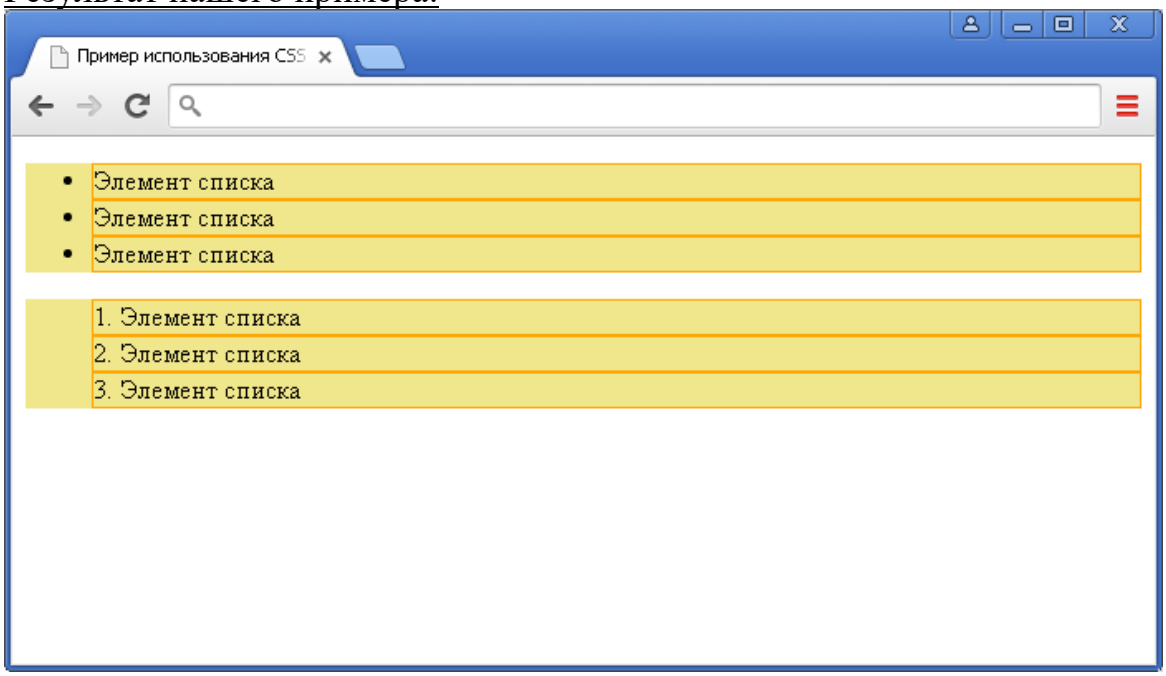


Рис. 3 Пример использования свойства list-style-position (расположение маркера / цифры в списках).

Создание собственных маркеров

Рано или поздно перед Вами встанет необходимость создать маркеры списка со своим дизайном, благодаря CSS свойству **list-style-image** мы это желание сможем реализовать в Вашем проекте.

Синтаксис свойства следующий:

```
ol {
list-style-image : url('images/primer.png'); /* указываем относительный путь к изображению */
}
ul {
```

```
list-style-image : url('http://basicweb.ru/images/mini5.png'); /* указываем абсолютный путь к изображению */
}
```

Значение в скобках соответствует пути к изображению, которое вы планируете использовать в роли маркера. Путь к изображению может быть как абсолютный, так и относительный. При указании относительного пути, важным моментом является то, что его **необходимо указывать относительно размещения таблицы стилей, а не страницы**.

Если вы планируете оформить собственные маркеры, то вам придется использовать программу для редактирования графики, либо воспользоваться уже готовыми наборами. Обратите внимание на такой момент, который может произойти, даже если Вы все сделали правильно, изображение может не загрузиться на страницу, в этом случае необходимо отредактировать изображение таким образом, чтобы его фон стал прозрачным.

Давайте рассмотрим пример использования собственных маркеров в документе:

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример использования CSS свойства list-style-image</title>
.test {
list-style-image : url('http://basicweb.ru/images/mini5.png'); /* указываем абсолютный путь к изображению, которое будет использовано в качестве маркера */
}
</style>
</head>
    <body>
        <ul class = "test">
            <li>Элемент списка</li>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
        </ul>
    </body>
</html>
```

В данном примере мы указываем **абсолютный путь** к изображению, которое будет использовано в качестве маркера.

Результат нашего примера:

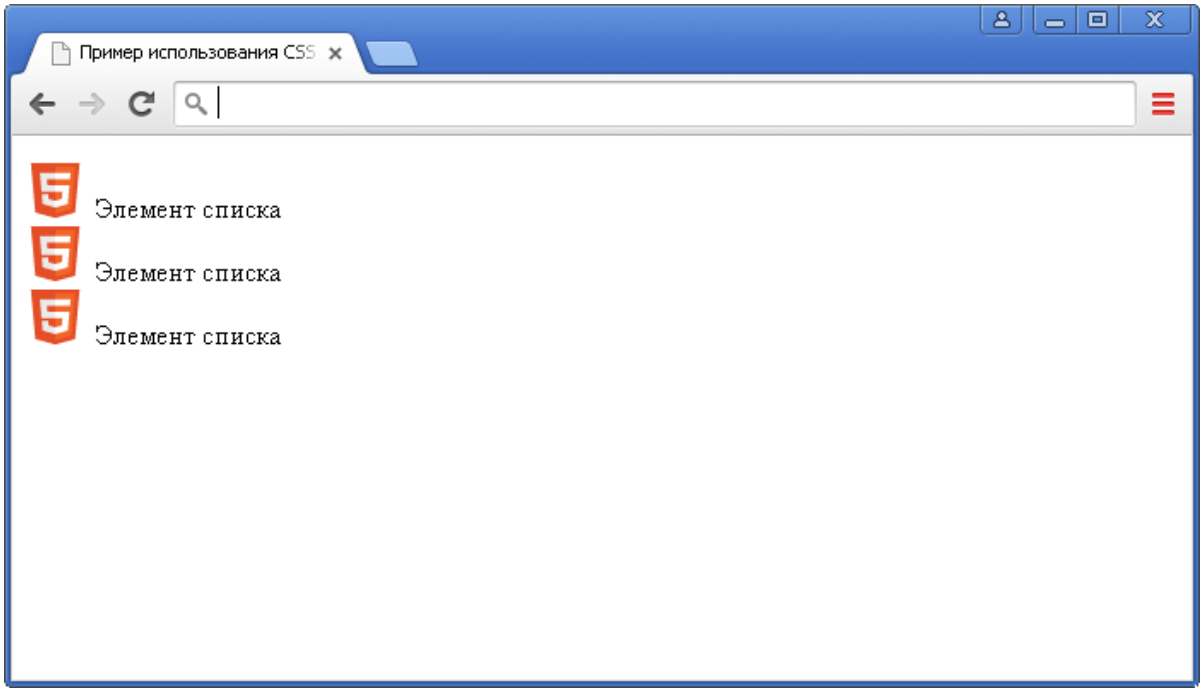


Рис. 4 Пример использования CSS свойства list-style-image (использование изображения в качестве маркера).

Изменение цвета маркера в CSS

В завершении этой статьи давайте рассмотрим продвинутый способ изменения цвета маркера без изменения цвета элемента, с использованием CSS свойства **content** и ранее рассмотренного псевдоэлемента **:before**:

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример изменения цвета маркера</title>
<style>
ul {
list-style : none; /* убираем маркеры у маркированного списка */
}
li:before  {/* Псевдоэлемент :before добавляет содержимое, указанное в свойстве content перед каждым элементом <li> */
content : "•"; /* вставляем содержимое, которое выглядит как маркер */
padding-right : 10px; /* устанавливаем правый внутренний отступ элемента. */
color : orange; /* устанавливаем цвет шрифта */
}
</style>
</head>
    <body>
        <ul>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
            <li>Элемент списка</li>
        </ul>
    </body>
</html>
```

Суть этого способа заключается в том, что мы перед каждым элементом списка (HTML элемент **<li>**) вставляем псевдоэлементом (**:before**) сгенерированный контент (CSS свойство **content**), который идентичен по внешнему виду маркеру в маркированном списке (HTML элемент **<ul>**), **только уже необходимого для нашей задачи цвета**.

Обращаю Ваше внимание, что в данном примере использовано свойство **padding-right**, которое нам позволило сделать внутренний отступ справа в каждом элементе списка (HTML элемент **<li>**). Если в данном случае не применять это свойство, то маркер будет находится в упор к тексту, что зрительно некрасиво. Работа с внутренними и внешними отступами элементов будет подробно рассмотрена в следующей статье учебника "**Блочная и строчная модель в CSS**".

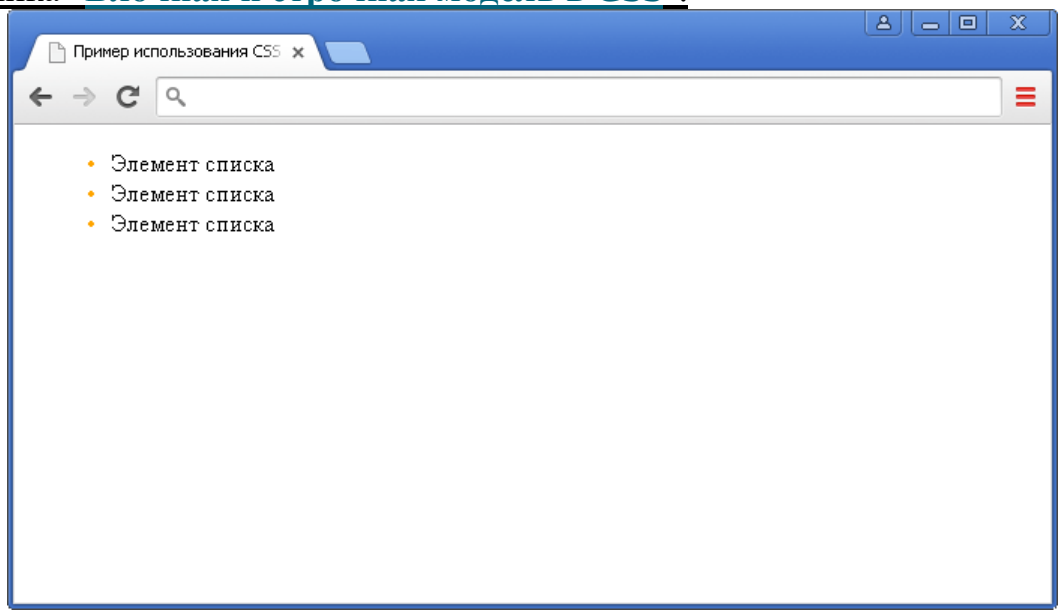


Рис. 5 Пример изменения цвета маркера с использованием свойства content и псевдоэлемента :before.

Лекция 1.12 CSS: работа с фоном

В этом учебнике мы уже рассмотрели с вами такие аспекты работы с изображениями как использование **свойств границ** (*цвет, стиль и толщина*), научились задавать **тень для изображений** (*Статья «**Тень элемента в CSS**»*), рассмотрели, как можно создать **эффект рамки**, добавив задний фон и пустой промежуток между границей и изображением (*Статья «**Блочная и строчная модель в CSS**»*). Научились делать **изображения плавающими** (*Статья «**Плавающие элементы в CSS**»*) и **позиционировать их** относительно краев родительского элемента, научились **управлять внешними отступами** между элементами, но всего этого пока не достаточно.

Для того чтобы Вы смогли создавать по настоящему красочные, яркие и уникальные сайты, вам необходимо познакомиться и изучить методы работы с задним фоном и с таким свойством как **background-image**

, которое позволяет задать одно или несколько фоновых изображений для элемента.

Фон элемента это общий размер элемента, включая значения внутренних отступов (**padding**) и границ (**border**), но, не включая значение внешних отступов – свойство **margin**.

В настоящее время браузеры работают с тремя графическими форматами:

GIF (англ. *Graphics Interchange Format* — формат для обмена изображениями).

JPEG (англ. *Joint Photographic Experts Group* — название организации-разработчика).

PNG (англ. *Portable network graphics* — растровый формат хранения графической информации).

Чтобы задать изображение в качестве заднего фона необходимо использовать свойство **background-image** и указать путь к файлу изображения, который может быть как относительный, так и абсолютный:

```
селектор {
background-image: url(images/main_bg.png);
}
```

Обращаю Ваше внимание на то, что при использовании относительного пути фоновое изображение необходимо указывать относительно адреса файла таблицы стилей, а не HTML страницы на которой предполагается использование фонового изображение.

Давайте рассмотрим пример, в котором установим задний фон, который представляет из себя три разноцветных квадрата размером **100** на **100** пикселей для элемента **<body>**:

```
<!DOCTYPE html>
<html>
<head>
	<title>Пример установки изображения в качестве фона</title>
<style>
body {
background-image: url("main_bg.png"); /* указываем путь к файлу изображения, которое будет использоваться как задний фон */
background-color: white; /* задаем задний фон для элемента */
}
</style>
</head>
	<body>
	</body>
</html>
```

По умолчанию, фоновое изображение размещается в верхнем левом углу элемента и повторяется как по вертикали, так и по горизонтали, в нашем примере это привело к тому, что элемент **<body>** полностью заполнился фоновым изображением.

Желательно всегда устанавливать цвет заднего фона в качестве альтернативы изображению, в этом случае, если изображение по каким-то причинам не будет загружено, то будет использован заданный Вами цвет. Запомните этот момент, мы не будем к нему возвращаться в статьях этого учебника.

Результат нашего примера:

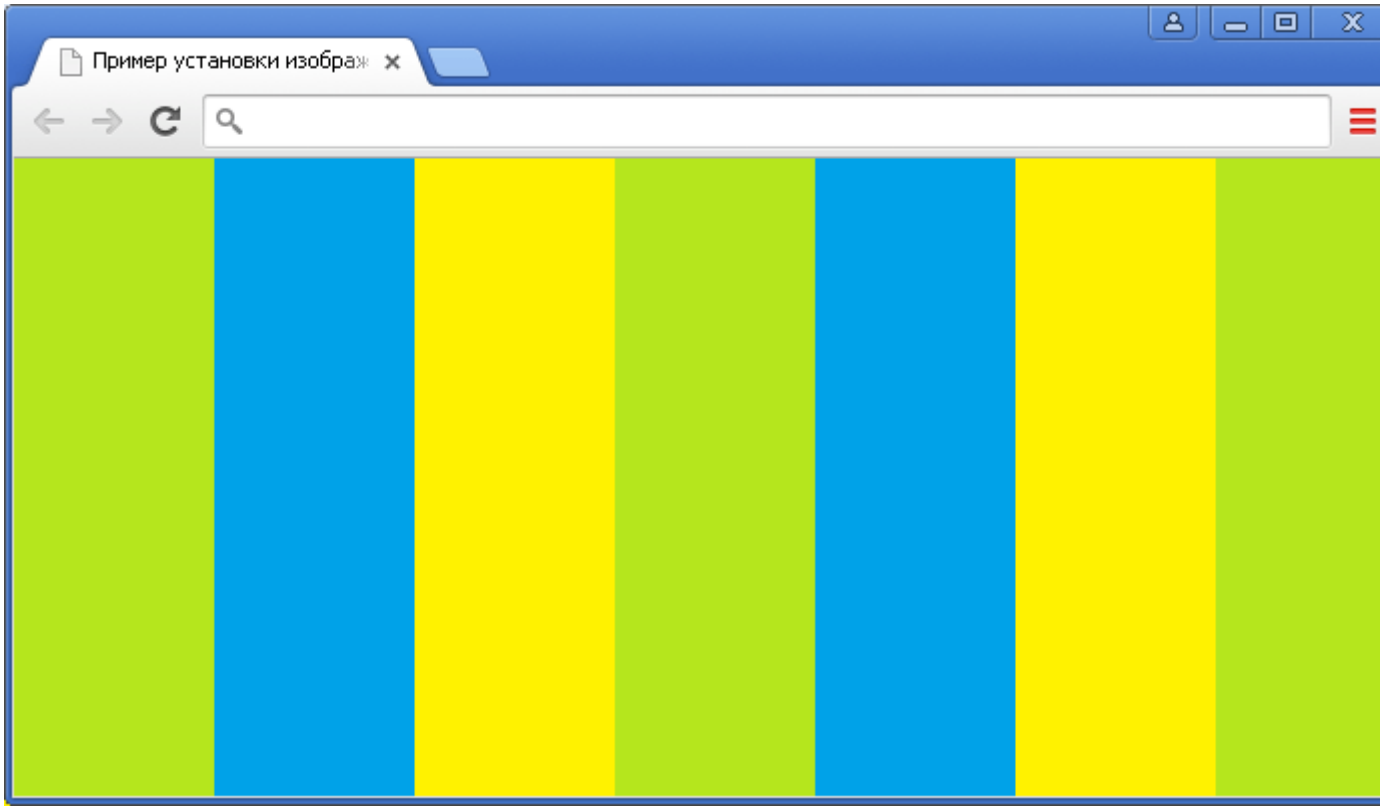


Рис. 115 Пример установки изображения в качестве фона.

Управление повтором фонового изображения

Как мы с вами установили из примера, фоновое изображение размещается по умолчанию в верхнем левом углу элемента и повторяется по вертикали и горизонтали. Давайте научимся изменять эти предопределенные значения и для начала рассмотрим, как управлять повтором изображения, а поможет нам в этом CSS свойство **background-repeat**.

Это свойство имеет следующие доступные значения:

Значение	Описание
repeat	Фоновое изображение будет повторяться как по вертикали, та **значение по умолчанию.**
repeat-x	Фоновое изображение будет повторяться по горизонтали (по оси x
repeat-y	Фоновое изображение будет повторяться по вертикали (по оси y).
no-repeat	Фоновое изображение не будет повторяться.

Для следующего примера используем задний фон, который представляет из себя два разноцветных квадрата размером 10 на 10 пикселей.

```
<!DOCTYPE html>
<html>
<head>
        <title>Пример управления повтором фонового изображения</title>
<style>
body {
background-image: url("main_bg.png"); /* указываем путь к файлу изображения, которое будет использоваться как задний фон */
}
div {
```

```
display: inline-block; /* устанавливаем, что элементы становятся блочно-строчными (чтобы выстроились в линейку) */
background-image: url("small_bg.png"); /* указываем путь к файлу изображения, которое будет использоваться как задний фон */
width: 200px; /* устанавливаем ширину элемента */
height: 200px; /* устанавливаем высоту элемента */
border: 1px solid; /* устанваливаем сплошную границу размером 1 пиксель */
margin-right: 10px; /* устанавливаем внешние отступы справа */
text-align: center; /* выравниваем текст по центру */
line-height: 200px; /* задаем высоту строки */
background-color: azure; /* указываем цвет заднего фона*/
}
.noRepeat {
background-repeat: no-repeat; /* фоновое изображение не будет повторяться. */
}
.repeatX {
background-repeat: repeat-x; /* фоновое изображение будет повторяться по горизонтали */
}
.repeatY {
background-repeat: repeat-y; /* фоновое изображение будет повторяться по вертикали */
}
</style>
</head>
    <body>
        <h1>Значение repeat для body (по умолчанию)</h1>
        <div class = "noRepeat">no-repeat</div>
        <div class = "repeatX">repeat-x</div>
        <div class = "repeatY">repeat-y</div>
    </body>
</html>
```

По аналогии с предыдущим примером для <body> мы установили задний фон, который **дублируется как по горизонтали, так и по вертикали.** Кроме того, мы создали для наших блоков **три класса**, которые определяют как будет повторяться фоновое изображение, используя различные значения для свойства **background-repeat**:

Для *первого* блока мы указали, что изображение не будет повторяться (значение **no-repeat**), это значение чаще всего используется в работе.

***Второй* блок** получил значение **repeat-x**, которое определяет, что фоновое изображение будет дублироваться по горизонтали.

Ну и в ***третьем* блоке** фон дублируется по вертикали (значение **repeat-y**).

Результат нашего примера:

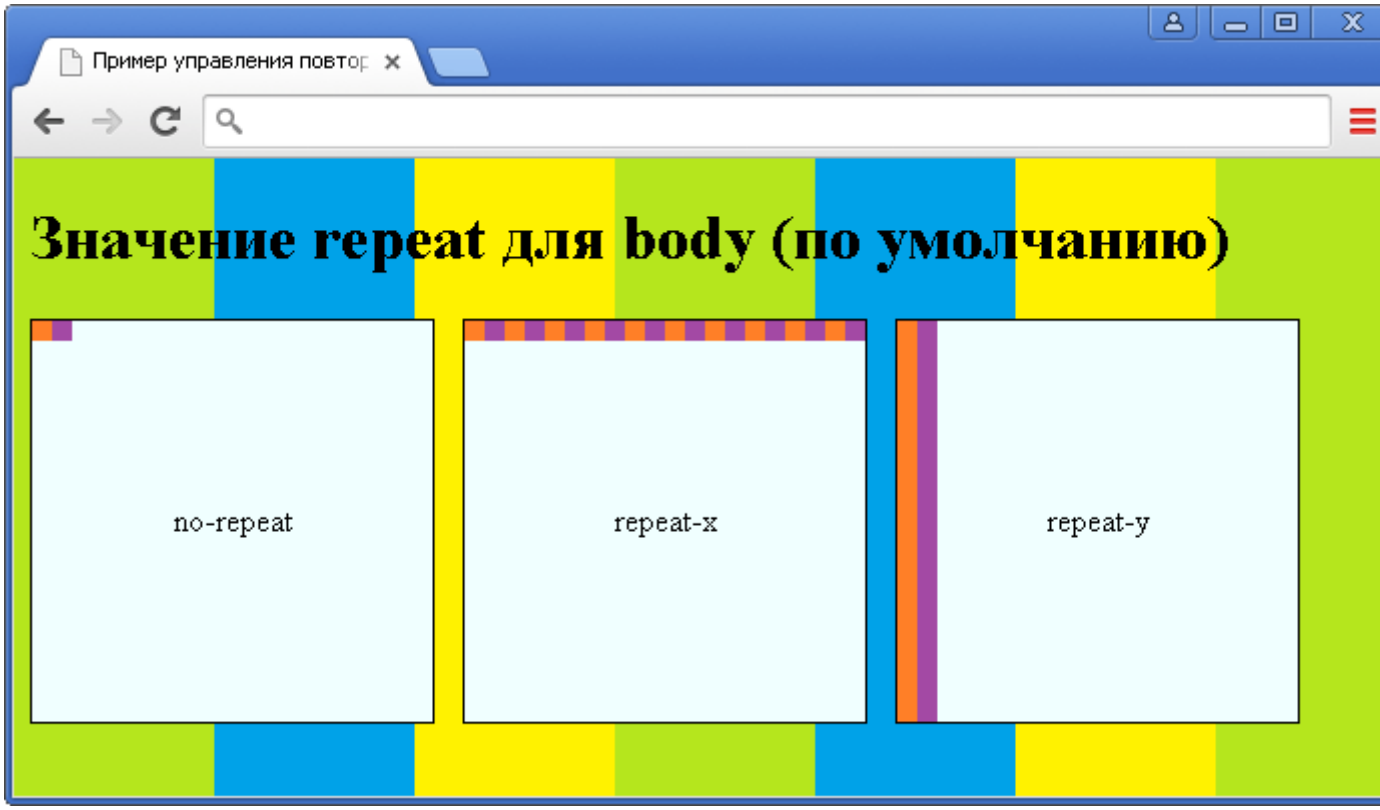
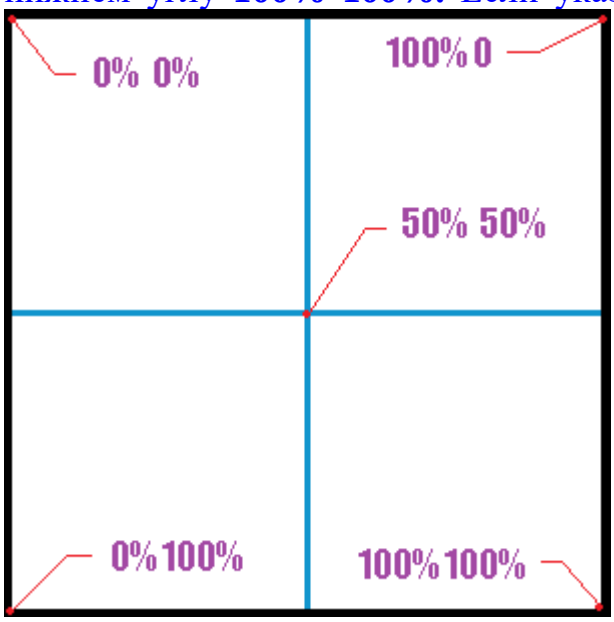


Рис. 116 Пример управления повтором фонового изображения.

Управление позицией фонового изображения

По умолчанию, фоновое изображение позиционируется в верхнем левом углу элемента, используя CSS свойство **background-position** мы можем изменить это положение с использованием **единиц измерения CSS**, либо используя **ключевые слова**:

Значение	Описание
left top left center left bottom right top right center right bottom center top center center center bottom	Задает положение изображения. Первое значение-горизонтальн значение вертикальное. Если вы указываете только одно ключевое слов "center"
x% y%	Задает положение изображения. *Первое* значение - горизонтальн значение вертикальное. Левый верхний угол имеет **0% 0% (это значение**

	нижнем углу **100% 100%.** Если указано только одно значение, то дру 
x y	Задает положение изображения. *Первое* значение - горизонтальн значение вертикальное. Левый верхний угол имеет **0 0.** Значения могут бы единицах измерения CSS. Если указано только одно значение, то другое можете совместно использовать **проценты** и **единицы измерения.**

Рассмотрим пример использования этого свойства:

```
<!DOCTYPE html>
<html>
<head>
        <title>Пример позиционирования фонового изображения</title>
<style>
div {
display: inline-block; /* устанавливаем, что элементы становятся блочно-строчными (чтобы выстроились в линейку) */
background-image: url("smile_bg.png"); /* указываем путь к файлу изображения, которое будет использоваться как задний фон */
background-repeat: no-repeat; /**/
width: 100px; /* устанавливаем ширину элемента */
height: 100px; /* устанавливаем высоту элемента */
border: 1px solid; /* устанваливаем сплошную границу размером 1 пиксель */
margin: 10px; /* устанавливаем внешние отступы со всех сторон */
text-align: center; /* выравниваем текст по центру */
line-height: 60px; /* указываем высоту строки */
background-color: azure; /* задаем цвет заднего фона */
}
.leftTop {background-position: left top;} /* задаем позицию ключевыми словами */
.leftCenter {background-position: left center;} /* задаем позицию ключевыми словами */
.leftBottom {background-position: left bottom;} /* задаем позицию ключевыми словами */
.rightTop {background-position: right top;} /* задаем позицию ключевыми словами */
```

```
.rightCenter {background-position: right center;} /* задаем позицию ключевыми словами */
.rightBottom {background-position: right bottom;} /* задаем позицию ключевыми словами */
.centerTop {background-position: center top;} /* задаем позицию ключевыми словами */
.centerCenter {background-position: center center;} /* задаем позицию ключевыми словами */
.centerBottom {background-position: center bottom;} /* задаем позицию ключевыми словами */
.userPosition {background-position: 20px 75%;} /* задаем позицию по горизонтали в пикселях, а по вертикали в процентах */
</style>
</head>
<body>
<div class = "leftTop">left top</div>
<div class = "leftCenter">left center</div>
<div class = "leftBottom">left bottom</div>
<div class = "rightTop">right top</div>
<div class = "rightCenter">right center</div>
<div class = "rightBottom">right bottom</div>
<div class = "centerTop">center top</div>
<div class = "centerCenter">center center</div>
<div class = "centerBottom">center bottom</div>
<div class = "userPosition">20px 75%</div>
</body>
</html>
```

В данном примере, мы создали **10 блоков** с различными классами, в которых заданы различные значения, связанные с позиционированием фоновых изображений. **Для первых девяти блоков** были использованы всевозможные *ключевые слова*, а для **последнего блока** было задано значение для *горизонтального позиционирования* ***в пикселях***, а для *вертикального* ***в процентах***.

Результат нашего примера:

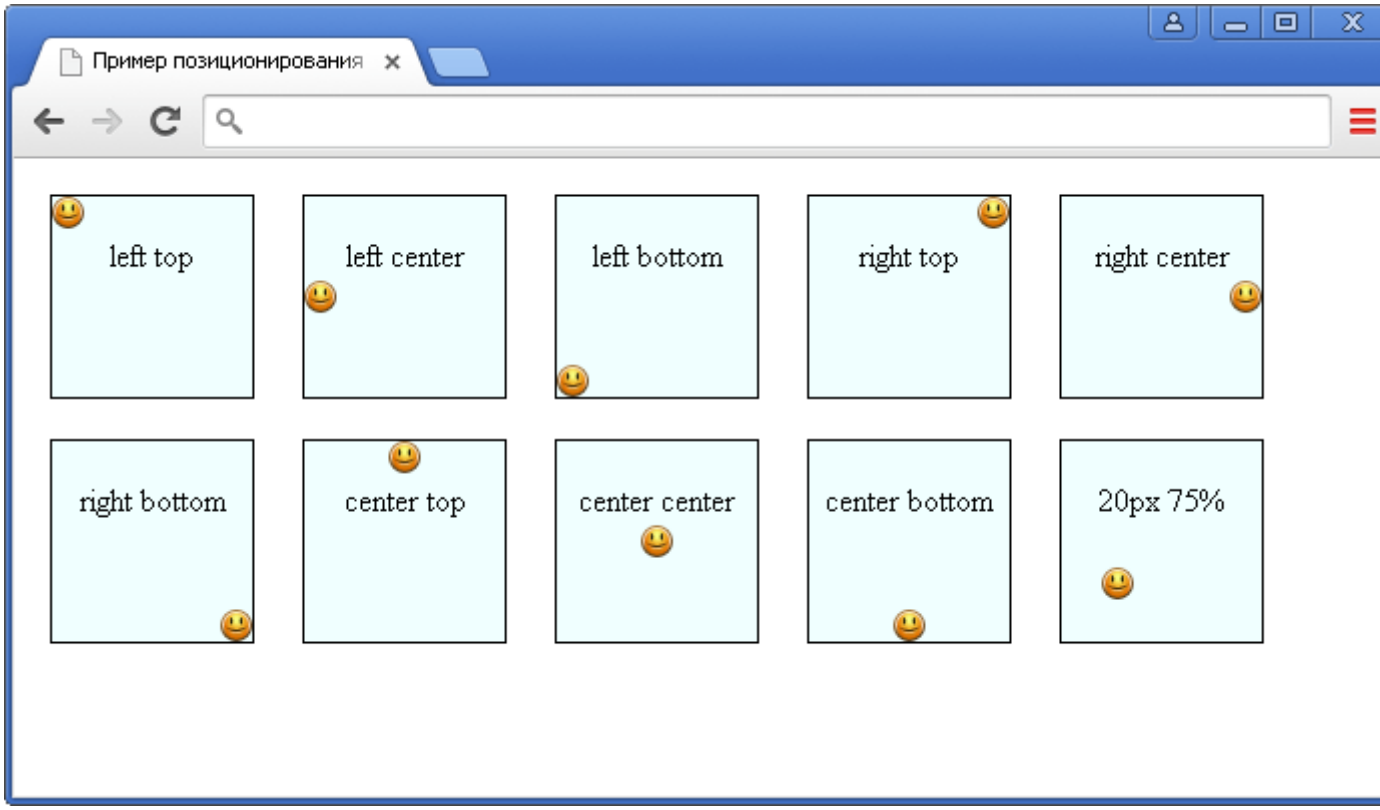


Рис. 117 Пример позиционирования фонового изображения.

Фиксированный задний фон

Когда вы прокручиваете содержимое страницы фоновое изображение, как правило, прокручивается вместе с содержимым. Это значение используется по умолчанию и подходит для большинства задач, но средствами CSS вы можете изменить такое поведение заднего фона, например, зафиксировав его.

Давайте с Вами рассмотрим, как с помощью свойства **background-attachment** сделать *"параллакс"* эффект.

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример фиксированного фонового изображения</title>
<style>
div {
height: 600px; /* устанавливаем высоту элемента */
}
.primerFixed {
background-image: url('nich.jpg'); /* указываем путь к файлу изображения, которое будет использоваться как задний фон */
background-attachment: fixed; /* указываем, что задний фон будет зафиксирован */
background-position: center; /* центрируем задний фон */
background-repeat: repeat-x; /* фоновое изображение будет повторяться по горизонтали */
}
</style>
</head>
<body>
    <div></div>
```

```
        <div class = "primerFixed"></div>
        <div></div>
    </body>
    </html>
```

В данном примере для всех элементов ** ** мы установили высоту равную **600px** и разместили **три таких блока** на странице. **Для среднего блока** мы создали класс **.primerFixed**, который задает для элемента фиксированный задний фон элемента, центрирует его и тиражирует по горизонтали (*ось x*). Для демонстрации эффекта *"параллакс"* откройте пример в **отдельном окне браузера и прокрутите страницу вниз.**

Результат нашего примера:

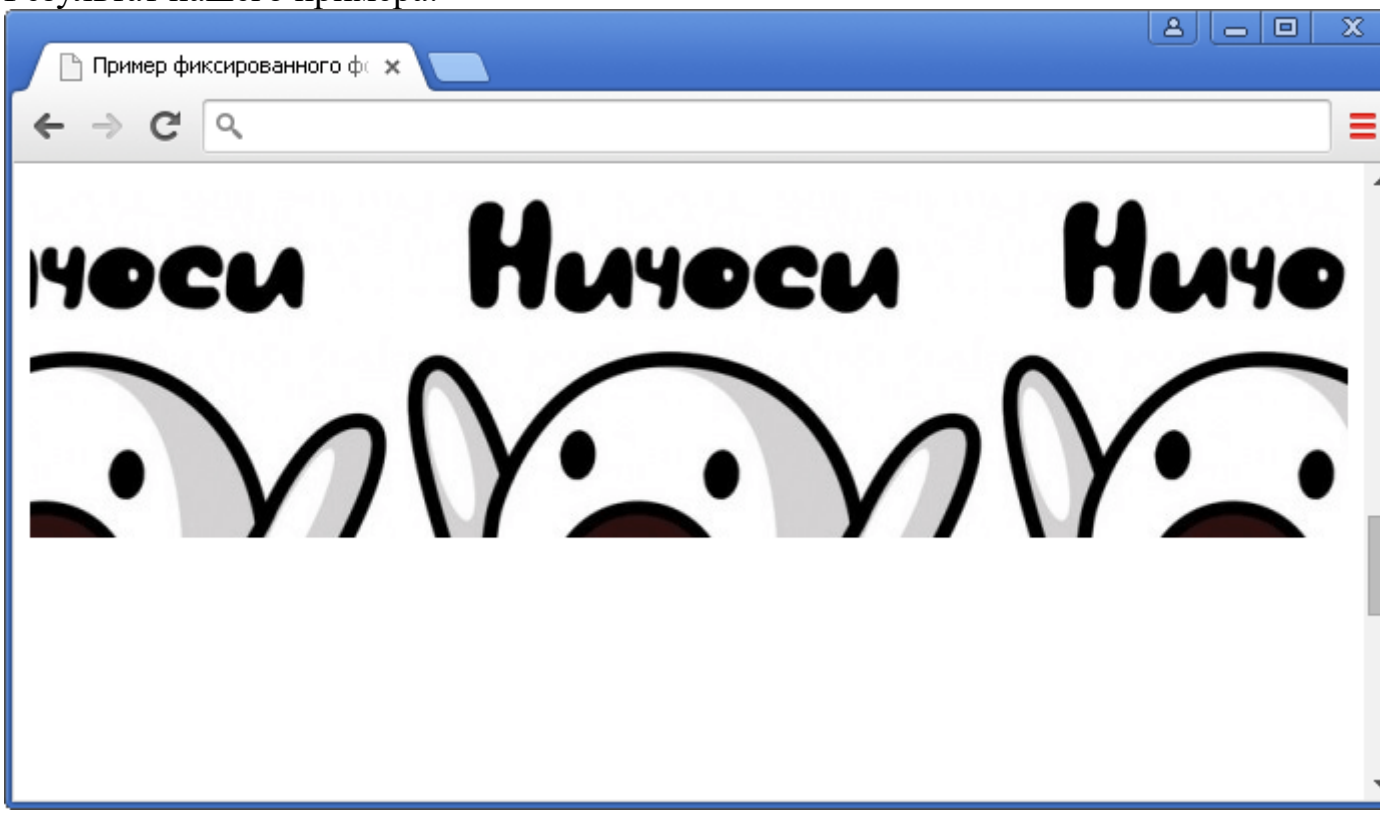


Рис. 118 Пример фиксированного фонового изображения.

Свойства CSS 3 для работы с фоновыми изображениями

Настало время углубить свои знания в работе с задним фоном элементов и познакомиться с новыми свойствами CSS, которые были введены в стандарте **CSS 3**.

И *первое* CSS свойство, которое мы рассмотрим - **background-origin**, оно определяет как позиционируется фоновое изображение, или изображения **по отношению к границе, внутреннему отступу и содержимому элемента.**

Возможные значения этого свойства:

Значение	Описание
padding-box	Фоновое изображение позиционируется от верхнего левого угла з заходит под границу элемента с этих сторон). **Это значение по умолчан**
border-box	Фоновое изображение позиционируется от верхнего левого угл заходит под границу элемента).
content-box	Фоновое изображение позиционируется от верхнего левого угла **со**

Рассмотрим применение этого свойства:

```
<!DOCTYPE html>
<html>
<head>
	<title>Пример использования свойства background-origin</title>
<style>
div {
width: 10em; /* устанавливаем ширину элемента */
height: 10em; /* устанавливаем высоту элемента */
border: 5px dashed orange; /* устанавливает пунктирную границу размером 5px оранжевого цвета */
background-image: url(manul.jpg); /* указываем путь к файлу изображения, которое будет использоваться как задний фон */
background-repeat: no-repeat; /* указываем, что фоновое изображение не будет повторяться */
display: inline-block; /* устанавливаем, что элементы становятся блочно-строчными (чтобы выстроились в линейку) */
margin-right: 15px; /* устанавливаем величину внешнего отступа от правого края элемента */
padding: 15px; /* устанавливаем величину внутреннего отступа для всех сторон элемента */
color: yellow; /* устанавливаем цвет шрифта желтый */
}
.test {background-origin: padding-box;}	/* устанавливаем, что фоновое изображение позиционируется от верхнего левого угла элемента (изображение не заходит под границу элемента с этих сторон) */
.test2 {background-origin: border-box;}	/* устанавливаем, что фоновое изображение позиционируется от верхнего левого угла элемента (изображение заходит под границу элемента) */
.test3 {background-origin: content-box;}	/* устанавливаем, что фоновое изображение позиционируется от верхнего левого угла содержимого элемента */
</style>
</head>
<body>
	<div class = "test">padding-box</div>
	<div class = "test2">border-box</div>
	<div class = "test3">content-box</div>
</body>
</html>
```

В данном примере мы разместили **три блока**, задали для них задний фон в виде изображения и указали для них различные значения свойства **background-origin**:

Первый **блок** (**padding-box**) - фоновое изображение позиционируется от верхнего левого угла элемента (изображение не заходит под границу элемента с этих сторон). **Это значение по умолчанию.**

Второй **блок** (**border-box**) - фоновое изображение позиционируется от верхнего левого угла элемента (изображение заходит под границу элемента со всех сторон).

Третий **блок** (**content-box**) - фоновое изображение позиционируется от верхнего левого угла содержимого элемента (изображение не заходит под границу элемента с этих сторон).

Результат нашего примера:

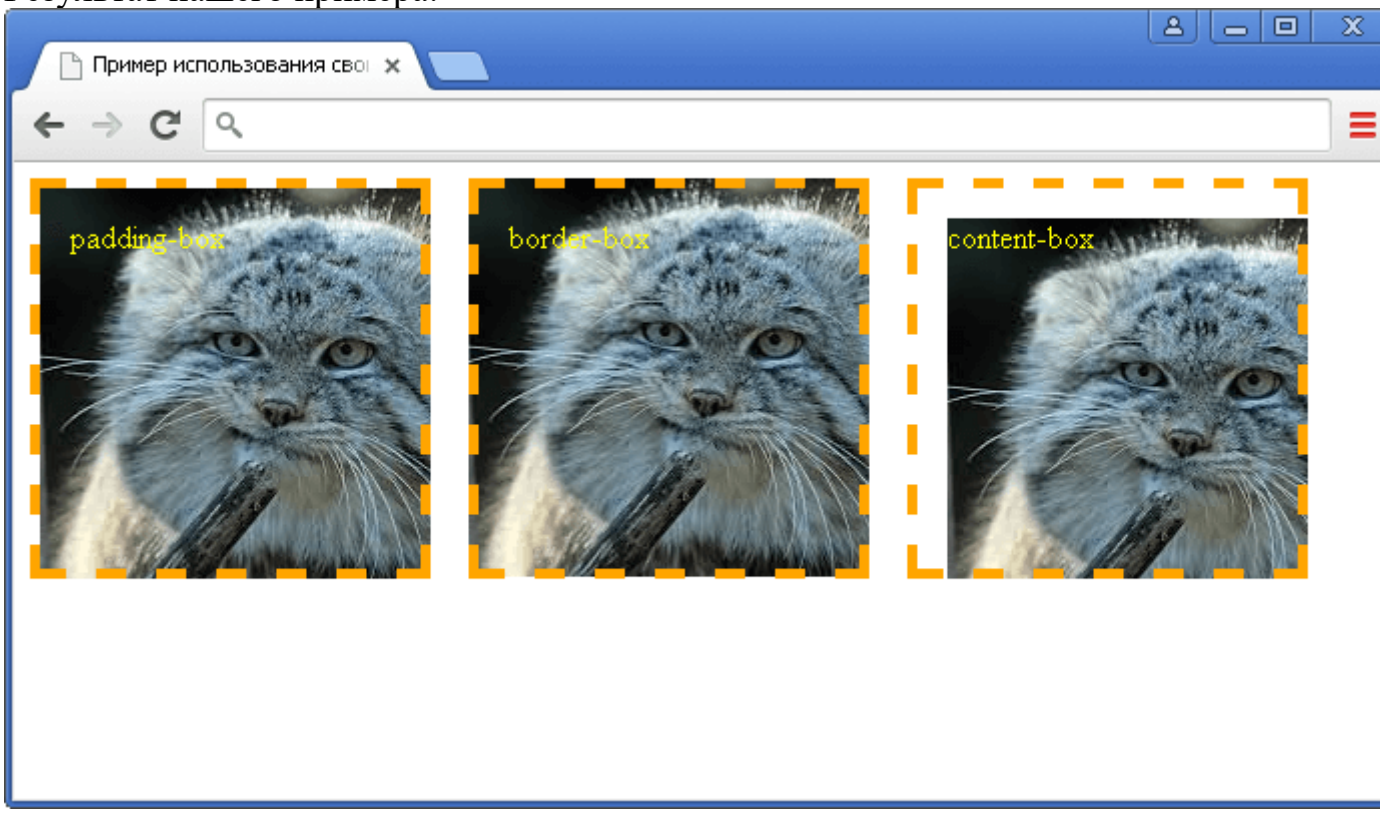


Рис. 119 Пример использования свойства background-origin.

И так на очереди следующее CSS свойство - **background-clip**, оно **определяет ту область элемента, для которой будет задан задний фон.**

У Вас может возникнуть вопрос: - *А в чем собственно заключается разница между свойством* ***background-origin*** *и* ***background-clip****?*

Разница заключается в том, что свойство background-clip в отличие от background-origin обрезает ту часть фона, которая выходит из указанных рамок. Свойство background-origin лишь определяет, как позиционируется фоновое изображение.

В таблице представлены возможные значения этого свойства:

Значение	Описание
border-box	Фон элемента занимает все пространство (включая границы элемента). **Это значение по умолчанию.**
padding-box	Фон элемента занимает все пространство (не включая границ элемента).
content-box	Фон элемента занимает все **содержимое** элемента (если у элемента установлены значения **padding** (внутренние отступы), то это пространство не будет окрашено).

Давайте рассмотрим следующий пример:

```
<!DOCTYPE html>
```

```
<html>
<head>
	<title>Пример использования свойства background-clip</title>
<style>
div {
width: 10em; /* устанавливаем ширину элемента */
height: 9em; /* устанавливаем высоту элемента */
background-image: url(medved.jpg); /* указываем путь к файлу изображения, которое будет использоваться как задний фон */
border: 5px dashed yellow; /* устанавливает пунктирную границу размером 5px черного цвета */
display: inline-block; /* устанавливаем, что элементы становятся блочно-строчными (чтобы выстроились в линейку) */
margin-right: 10px; /* устанавливаем величину внешнего отступа от правого края элемента */
padding: 15px; /* устанавливаем величину внутреннего отступа для всех сторон элемента */
}
.test {background-clip: border-box;}  /* устанавливаем, что фон элемента занимает все пространство (включая границы элемента) */
.test2 {background-clip: padding-box;}  /* устанавливаем, что фон элемента занимает все пространство (не включая границы элемента) */
.test3 {background-clip: content-box;}  /* устанавливаем, что фон элемента занимает все содержимое элемента */
</style>
</head>
<body>
	<div class = "test">border-box</div>
	<div class = "test2">padding-box</div>
	<div class = "test3">content-box</div>
</body>
</html>
```

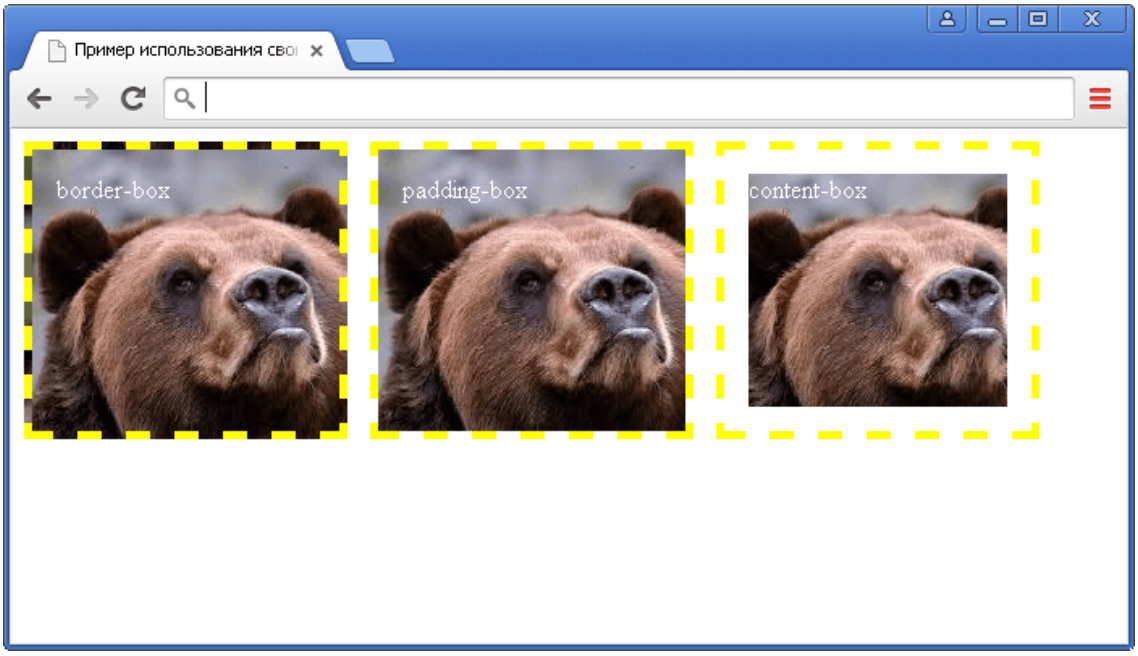
В этом примере мы разместили ***три* блока**, установили для них задний фон в виде изображения и указали различные значения свойства **background-clip**:

***Первый* блок** (**padding-box**) - фон элемента занимает все пространство. **Это значение по умолчанию.**

***Второй* блок** (**border-box**) - фон элемента занимает все пространство (не включая границы элемента).

***Третий* блок** (**content-box**) - фон элемента занимает все содержимое элемента.

Результат нашего примера:



Пример использования свойства background-clip.

Как вы могли заметить свойства **background-origin** и **background-clip**, **имеет смысл применять только тогда, когда у элемента есть внутренние отступы, либо границы.**

На очереди следующее свойство, которое позволит нам в полной мере управлять задним фоном элемента по своему усмотрению - это свойство **background-size**, оно имеет широкое применение в современной верстке сайтов, так как **позволяет управлять размером фонового изображения.**

Установить размер заднего фона допускается с использованием **единиц измерения CSS, процентов, либо ключевых слов**:

Значение	Описание
auto	Фоновое изображение содержит свою ширину и высоту. **Это значение по умолчанию.**
length	Устанавливает ширину и высоту фонового изображения. *Первое* значение устанавливает ширину, *второе* значение задает высоту. Если указано только одно значение, то для второго устанавливается значение **auto**.
%	Устанавливает ширину и высоту фонового изображения в процентах от родительского элемента. *Первое* значение устанавливает ширину, *второе* значение задает высоту. Если указано только одно значение, то для второго устанавливается значение **auto**.
cover	Масштабирует фоновое изображение под размеры элемента. Некоторые части фонового изображения могут быть скрыты из поля зрения.
contain	Масштабирует фоновое изображение, чтобы оно целиком поместилось внутри элемента.

Рассмотрим применение этого свойства:

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>Пример масштабирования фоновых изображений</title>
<style>
div {
width: 8em; /* устанавливаем ширину элемента */
height: 8em; /* устанавливаем высоту элемента */
border: 3px solid orange; /* устанваливаем сплошную границу размером 3 пикселя оранжевого цвета */
background-image: url(manul.jpg); /* указываем путь к файлу изображения, которое будет использоваться как задний фон */
background-repeat: no-repeat; /* указываем, что фоновое изображение не будет повторяться */
display: inline-block; /* устанавливаем, что элементы становятся блочно-строчными (чтобы выстроились в линейку) */
margin-right: 15px; /* устанавливаем внешний отступ с правой стороны */

margin-bottom: 15px; /* устанавливаем внешний отступ с нижней стороны */

color: yellow; /* устанавливаем цвет элемента */
}
.test {background-size: auto;} /* фоновое изображение содержит свою ширину и высоту */
.test2 {background-size: 100px 100px;} /* устанавливаем ширину и высоту фонового изображения в пикселях */
.test3 {background-size: 70% 70%;} /* устанавливаем ширину и высоту фонового изображения в процентах от родительского элемента */
.test4 {background-size: cover;} /* масштабируем фоновое изображение под размеры элемента */
.test5 {background-size: contain;} /* масштабируем фоновое изображение, чтобы оно целиком поместилось внутри элемента */
</style>
</head>
    <body>
        <div class = "test">auto</div>
        <div class = "test2">100px 100px</div>
        <div class = "test3">70% 70%</div><br>
        <div class = "test4">cover</div>
        <div class = "test5">contain</div>
    </body>
</html>
```

В данном примере мы разместили **пять блоков**, задали для них задний фон в виде изображения и указали для них различные значения свойства **background-size**:

Первый **блок** (**auto**) - фоновое изображение содержит свою ширину и высоту. **Это значение по умолчанию.**

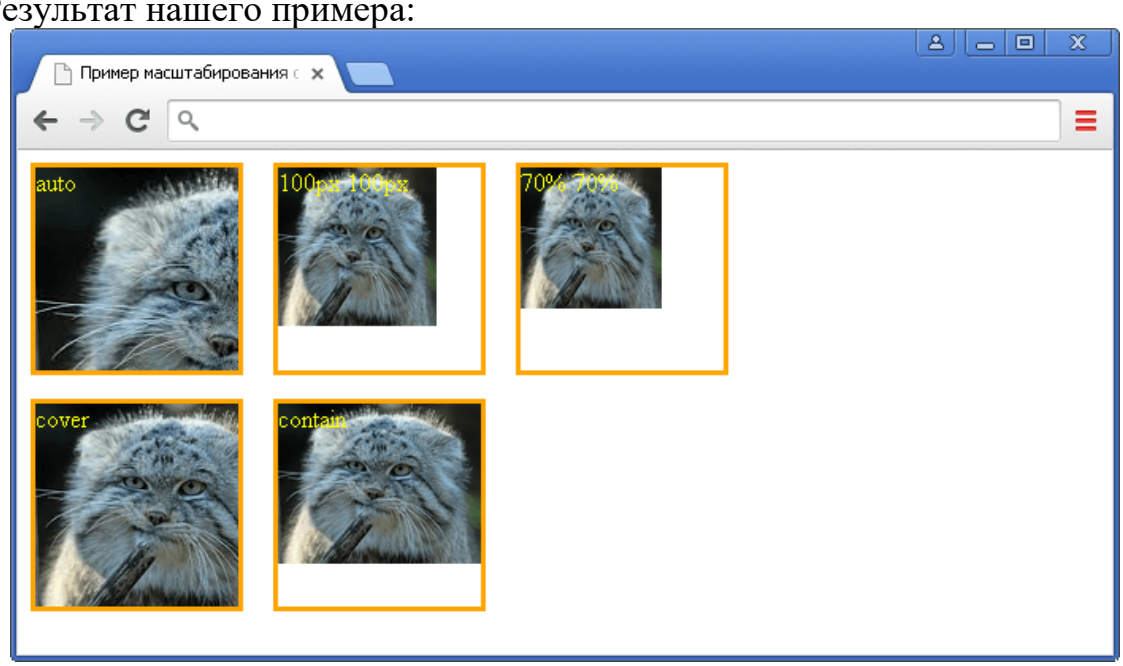
Второй **блок** (**100px 100px**) - первое значение устанавливает ширину, второе значение задает высоту в пикселях.

Третий **блок** (**70% 70%**) - первое значение устанавливает ширину, второе значение задает высоту в процентах.

***Четвертый* блок** (**cover**) – масштабирует фоновое изображение под размеры элемента (некоторые части фонового изображения скрываются из поля зрения).

***Пятый* блок** (**contain**) – масштабирует фоновое изображение, чтобы оно целиком поместилось внутри элемента.

Результат нашего примера:



Пример масштабирования фоновых изображений.

Универсальное свойство background

Мы с Вами рассмотрели **все свойства, которые предназначены для работы с фоновыми изображениями**. В большинстве случаев вводить длинные названия рассмотренных выше свойств непродуктивно, но это не значит, что мы зря потратили на это время - *без понимания как они работают по отдельности, вы не сможете грамотно их применять в одном объявлении.*

Существует более простой метод задать значения всех свойств для работы с задним фоном в одном объявлении, используя универсальное свойство background.

Свойство **background** имеет следующий синтаксис:

background: **"color image position/size repeat origin clip attachment**;

Где значения соответствуют вышерассмотренным нами свойствам:

background-color (**color | transparent**).
background-image (**url | none**).
background-position (**значение**).
background-size (**auto | length | cover | contain**).
background-repeat (**repeat | repeat-x |repeat-y | no-repeat**).
background-origin (**padding-box | border-box | content-box**).
background-clip (**border-box | padding-box | content-box**).
background-attachment (**scroll | fixed | local**).

Давайте рассмотрим пример использования универсального свойства **background**:

```
<!DOCTYPE html>
<html>
<head>
    <title>Пример использования универсального свойства background</title>
<style>
```

```
html, body {
height: 100%; /* устанавливаем высоту элементов */
width: 100%; /* устанавливаем ширину элементов */
margin: 0; /* убираем внешние отступы элемента */
padding: 0; /* убираем внутренние отступы элемента */
}
header {
width: 100%; /* устанавливаем ширину элемента */
min-height: 34%; /* устанавливаем минимальную высоту элемента */
background: white url('cat_g.jpg') bottom/cover no-repeat; /* указываем цвет заднего фона, фоновое изображение, позицию/масштабируем под размеры элемента, фон не повторяется */
}
.primer2 {
width: 100%; /* устанавливаем ширину элемента */
min-height: 66%; /* устанавливаем минимальную высоту элемента */
background: url('lis.png') top/contain no-repeat, url('cat_g.jpg') bottom/cover no-repeat;
/* Обратите внимание, что значения для различных изображений указываются через запятую */
/* Путь к изображению 1 позиция/масштаб повтор, Путь к изображению 2 позиция/масштаб повтор */
}
</style>
</head>
    <body>
        <header></header>
        <div class = "primer2"></div>
    </body>
</html>
```

И так, что мы сделали в этом примере:

Мы установили для элементов **<html>** и **<body>** высоту **100%**, убрали внутренние и внешние отступы.

Для элемента **<header>** задали минимальную высоту равную **34%** от родительского элемента, ширину установили **100%**. В качестве заднего фона установили изображение - **url('cat_g.jpg')**, позиционировали его по низу и масштабировали фоновое изображение под размеры элемента (**center** / **contain** - **background-position** / **background-size**). **Без косой черты, как и без позиции фонового изображения работать не будет.**

Для элемента ** ** с классом **.primer2** задали минимальную высоту равную **66%** от родительского элемента, ширину установили **100%**. В качестве заднего фона установили два различных изображения, позиционировали их по центру (**center**) и масштабировали их (первое изображение полностью помещается - значение **contain**, второе изображение масштабируется под размеры элемента **cover**).

Результат нашей работы:

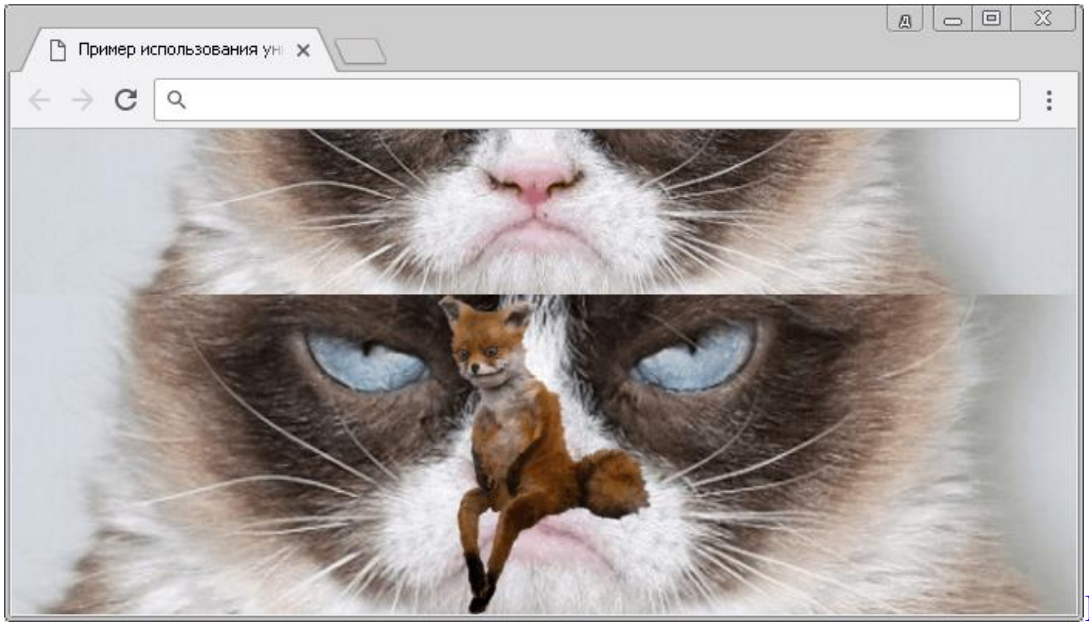


Рис. Пример использования универсального свойства background.

Обращаю Ваше внимание на то, что установка нескольких фоновых изображений в качестве заднего фона для одного элемента выполнена для демонстрации возможностей CSS. В большинстве случаев проще установить один задний фон для одного элемента, а уже этот элемент настроить и позиционировать в документе как вам необходимо. Подробное изучение позиционирования элементов будет освещено далее в учебнике в статье "**Позиционирование элементов в CSS**".

Лекция 1.13 CSS: позиционирование и видимость элементов

Эта статья учебника будет посвящена очень важной теме, которая связана с позиционированием элементов на странице, она потребует от Вас максимального внимания. Вы познакомитесь с такими типами позиционирования элементов как: *абсолютное*, *относительное*, *фиксированное* и *статическое*.

Позиционирование позволит Вам разместить тот, или иной элемент в том месте, где это Вам необходимо, цель этой статьи заключается в том, чтобы понять по каким правилам это происходит, какие при этом необходимо использовать CSS свойства и для чего.

Типы позиционирования элементов

Основное свойство CSS, которое позволяет управлять позиционированием элементов на странице это свойство **position**

, оно сообщает браузеру, какой тип позиционирования используется для элемента (*статический* - **static**, *относительный* - **relative**, *абсолютный* – **absolute**, или *фиксированный* - **fixed**).

Для полного понимания как происходит позиционирование элементов на любой странице, Вам необходимо детально изучить все виды позиционирования. Эта статья учебника предоставит Вам такую возможность, сейчас мы с Вами отдельно поговорим о каждом виде позиционирования и разберем, как и относительного чего происходит смещение элементов в документе.

Абсолютное позиционирование

Совместно со свойством **position** используются CSS свойства, которые управляют смещением позиционированного элемента:

top (смещение позиционированного элемента от верхнего края).
right (смещение позиционированного элемента от правого края).
bottom (смещение позиционированного элемента от нижнего края).
left (смещение позиционированного элемента от левого края).

В качестве значений, которые определяют смещение элемента, допускается использовать **физические единицы** (например, *пункты*), но чаще используют **визуальные единицы** – *пиксели*, **процентные значения** и значения *em*. Значения могут быть как *положительные*, так и *отрицательные*, как и люди, только значения.

При использовании абсолютного позиционирования (**position**: **absolute**) элемент сдвигается (позиционируется) **относительно заданного края его предка**, при этом предок **должен иметь** значение **position** отличное от, установленного по умолчанию - **static**, иначе отсчёт (смещение) будет вестись **относительно, указанного края окна браузера**.

Давайте начнем с простого примера, в котором мы будем позиционировать элементы (блоки) с абсолютным типом позиционирования.

```
<!DOCTYPE html>
```

```
<html>
<head>
	<meta charset = "UTF-8">
	<title>Абсолютное позиционирование элементов</title>
<style>
div {
position: absolute; /* абсолютное позиционирование элементов */
width: 100px; /* ширина элемента */
height: 100px; /* высота элемента */
}
.absolute {
top: 3em; /* смещение от вержнего края */
right: 50%; /* смещение от правого края */
background-color: green; /* цвет заднего фона */
}
.absolute2 {
bottom: 25%; /* смещение от нижнего края */
left: 40%; /* смещение от левого края */
background-color: yellow; /* цвет заднего фона */
</style>
</head>
	<body>
		<div class = "absolute">
			absolute
		</div>
		<div class = "absolute2">
			absolute2
		</div>
	</body>
</html>
```

И так, что мы сделали в этом примере:

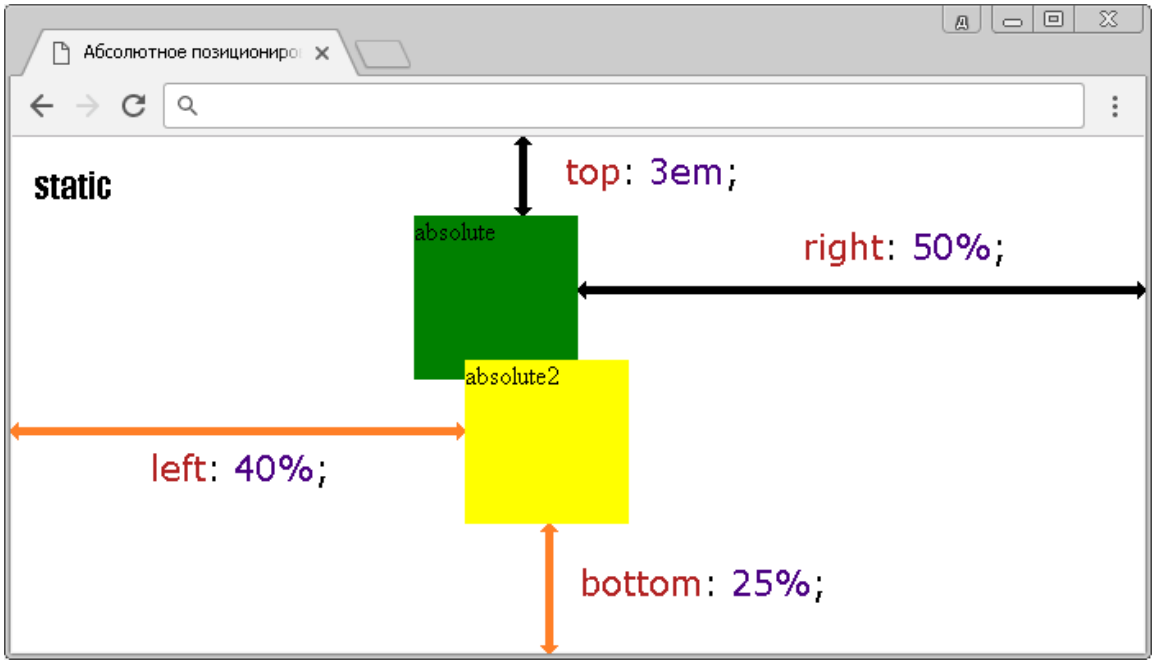
Разместили два блока ** ** шириной и высотой **100 пикселей** и указали для них, что они имеют *абсолютное позиционирование*.

Для *первого* блока мы указали, что он смещается от верхнего края окна на **3em**, а от правого на **50%**.

Для *второго* блока мы указали, что он смещается от нижнего края окна на **25%**, а от левого на **40%**.

Теперь важный момент, на который вы, скорее всего не обратили внимание. Почему наши элементы позиционируются относительно окна браузера? **Элементы будут позиционироваться относительно заданного края предка лишь в том случае, если их предок имеет значение свойства position отличное от, установленного по умолчанию - static, иначе смещение будет вестись относительно края окна браузера**. Подобные ситуации не раз будут возникать у Вас во время верстки страниц, запомните этот важный момент, мы еще к нему вернемся далее в этой статье учебника.

Результат нашего примера:



Пример абсолютного позиционирования элементов на странице.

Обратите внимание, что наш *второй* блок (желтый) наложился на *первый*, в конце предыдущей статьи учебника "**Работа с таблицами в CSS**", мы уже сталкивались со свойством **z-index**, благодаря ему, вы можете управлять видимостью элементов по *оси z*. Например, если задать для *первого* (зеленого) блока значение **z-index**: **1**, то уже он будет находиться выше по оси, чем *второй* (желтый блок) и будет полностью виден. Аналогичного эффекта можно добиться, если указать для желтого блока *отрицательное* значение свойства **z-index**.

Как вы заметили, элементы, которые имеют *абсолютное позиционирование*, **отделяются от основного потока страницы**, что может приводить к наслоению элементов друг на друга. **Еще один нюанс** работы с элементами, которые имеют *абсолютное позиционирование*, это то, что они **не могут быть плавающими**. Плавающими элементами могут быть только элементы, которые имеют *статическое позиционирование* (**static**), то есть то, которое установлено у элемента по умолчанию. Методы работы с плавающими элементами мы с Вами рассматривали в статье учебника "**Плавающие элементы в CSS**".

Относительное позиционирование

Следующий тип позиционирования, который мы рассмотрим это *относительное позиционирование*. Элементы, для которых задано *относительное позиционирование* (**position**: **relative**) смещаются (размещаются) **относительно положения в потоке документа**, или другими словами **относительно его текущей позиции**.

Давайте сразу перейдем к примеру, а затем поговорим обо всех нюансах, которые будут возникать при работе с *относительным позиционированием*.

```
<!DOCTYPE html>
<html>
<head>
        <meta charset = "UTF-8">
        <title>Относительное позиционирование элементов</title>
<style>
.static {
height: 50px; /* высота элемента */
background-color: red; /* цвет заднего фона */
```

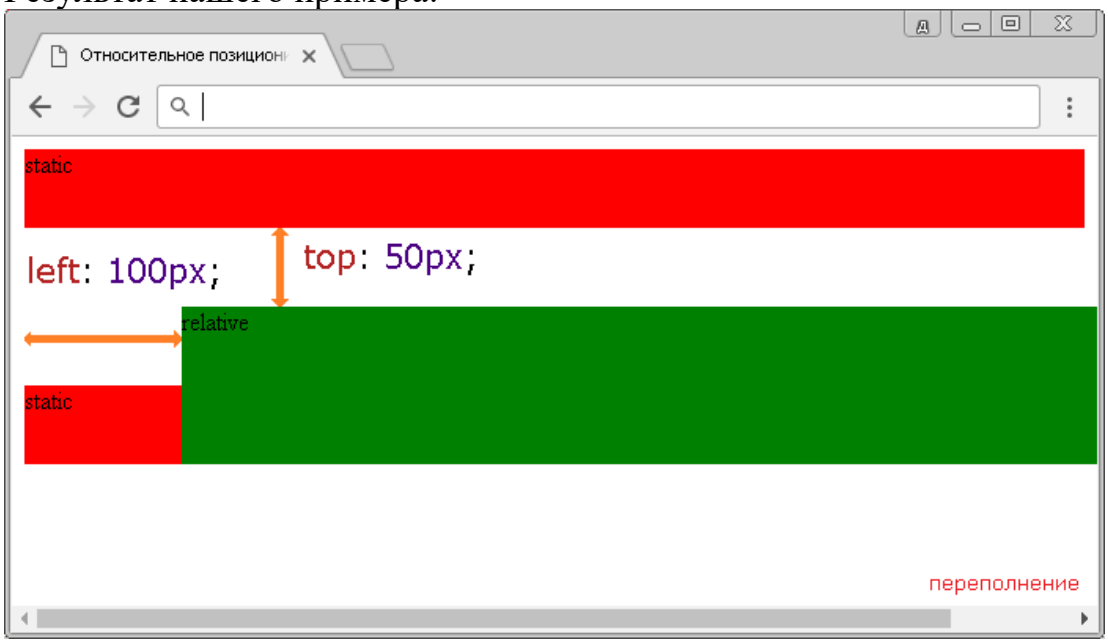
```
}
.relative {
position: relative; /* относительное позиционирование элемента */
height: 100px; /* высота элемента */
top: 50px; /* смещение от вержнего края */
left: 100px; /* смещение от левого края */
background-color: green; /* цвет заднего фона */
}
</style>
</head>
        <body>
                <div class = "static">
                        static
                </div>
                <div class = "relative">
                        relative
                </div>
                <div class = "static">
                        static
                </div>
        </body>
</html>
```

И так, что мы сделали в этом примере:

Для блоков (элементы), которые имеют *статическое позиционирование* (по умолчанию) установили высоту **50 пикселей** и цвет заднего фона – *красный*.

Разместили между блоков элемент с *относительным позиционированием* (**position**: **relative**), установили для него высоту **100 пикселей** и цвет заднего фона *зеленый*. Кроме того указали, что он **смещается относительно его текущей позиции** от верхнего края на **50 пикселей**, а с левого края на **100 пикселей**, вызывая при этом переполнение документа.

Результат нашего примера:



Пример относительного позиционирования элементов на странице.

Еще необходимо подчеркнуть из этого примера, то, что в отличие от *абсолютного позиционирования* другие элементы в документе реагируют на элементы с *относительным позиционированием*. Не смотря на то, что мы дали браузеру команду на перемещение элемента, браузер зарезервировал место под элемент, оставляя при этом пустое место, где элемент изначально должен находиться до перемещения.

На практике, Вам, скорее всего, не придется перемещать элементы, которые имеют *относительное позиционирование*. Основная идея *относительного позиционирования* заключается в том, чтобы не сдвинуть куда-то элемент, а **создать «контейнер» для элемента, который имеет *абсолютное позиционирование***. Другими словами, вложенные элементы будут смещаться не относительно края окна браузера, а относительно этого элемента, который будет иметь *относительное позиционирование* и находится **в основном потоке документа**. Более подробно этот момент мы рассмотрим далее в этой статье учебника.

Фиксированное позиционирование

Третий тип позиционирования, который мы рассмотрим это *фиксированное позиционирование*. При *фиксированном позиционировании* элемент сдвигается **относительно заданного края окна браузера**. Отличительная особенность этого позиционирования заключается в том, что при прокрутке страницы **элемент остается на одном месте**, то есть, грубо говоря, он прокручивается вместе со страницей (элемент зафиксирован).

Я думаю, что путешествуя по сети интернет, вы не раз встречали меню навигации, боковые панели, или даже кнопки "вверх страницы", которые были зафиксированы на одном месте. Всё это становится возможным благодаря *фиксированному позиционированию*.

Давайте рассмотрим пример, в котором мы оформим фиксированную боковую панель.

```
<!DOCTYPE html>
<html>
<head>
        <meta charset = "UTF-8">
        <title>Фиксированное позиционирование элементов</title>
<style>
html, body {
height: 100%;      /* высота элемента */
margin: 0; /* внешний отступ со всех сторон */
}
.fixed {
position: fixed; /* фиксированное позиционирование элемента */
height: 100%; /* высота элемента */
width: 15%; /* ширина элемента */
background-color: red; /* цвет заднего фона */
right: 0; /* смещение от правого края */
}
.container {
height: 2000px; /* высота элемента */
}
</style>
</head>
```

```
        <body>
            <div class = "fixed">
                fixed
            </div>
            <div class = "container">
            </div>
        </body>
    </html>
```

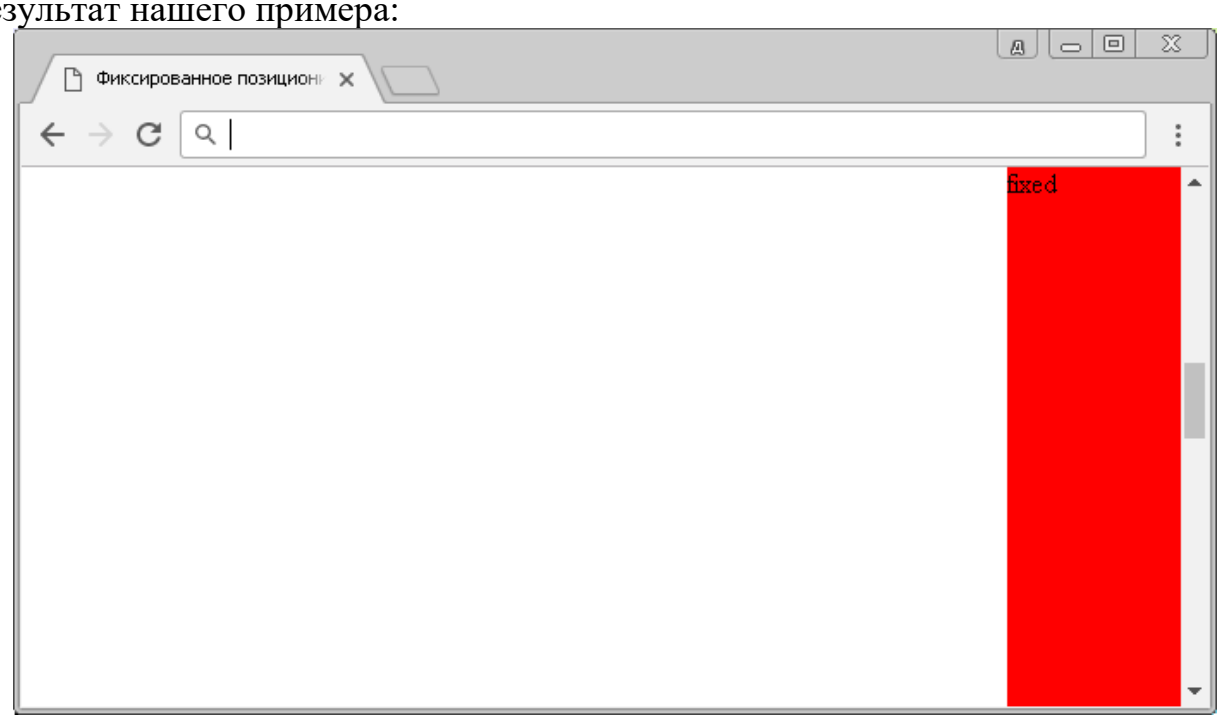
Давайте разберем, что мы сделали в этом примере:

Установили для элементов **<body>** и **<html>** высоту равную **100%**, это нам позволит задать высоту в процентах для нашей боковой панели. Кроме того, мы убрали внешние отступы (**margin**) для этих элементов, это необходимо, чтобы убрать встроенные стили браузера.

Для нашей боковой панели установили высоту равной родительскому элементу (**100%**), установили ширину **15%** от родительского элемента и установили цвет заднего фона *красный*. Кроме того указали, что наша боковая панель имеет *фиксированное позиционирование*, что позволяет её как будто прилепить к экрану. Чтобы наша панель отображалась справа, мы установили значение **right** равным **0** (смещение позиционированного элемента от правого края окна браузера).

Для демонстрации *фиксированного позиционирования* мы создали контейнер высотой **2000 пикселей**. Теперь если прокрутить страницу наша боковая панель останется на месте, а содержимое контейнера (основного содержимого) будет прокручиваться.

Результат нашего примера:



Пример фиксированного позиционирования элементов на странице.

Статическое позиционирование

Ну и заключительный тип позиционирования это *статическое позиционирование* (**static**), мы с Вами уже неоднократно говорили о нем. *Статическое позиционирование* это классическое **размещение элементов сверху вниз** (элементы отображаются в порядке, как они указаны **в потоке HTML документа**), оно считается значением по умолчанию для всех элементов.

Хочу обратить Ваше внимание на один факт, что вышерассмотренные свойства, отвечающие за смещение элементов, не допускается применять к элементам, которые имеют *статическое позиционирование*, то есть имеют позиционирование, которое установлено по умолчанию.

Продвинутое абсолютное позиционирование

Перед тем как перейти к рассмотрению продвинутого применения *абсолютного позиционирования*, хочу обратить Ваше внимание, на то, что если вы **не указываете** значение вертикальной позиции элемента с *абсолютным позиционированием* (**top** , или **bottom**), или наоборот горизонтальной позиции (**left**, или **right**), то **браузер оставит элемент в том же месте на странице, где он находится в общем потоке** (будет размещен поверх содержимого, если оно есть).

Мы уже с Вами узнали о том, что элемент с *абсолютным позиционированием* (**position**: **absolute**) позиционируется **относительно заданного края его предка**, при этом предок должен иметь значение **position отличное от, установленного по умолчанию - static, иначе** отсчёт (смещение) будет вестись **относительно, указанного края окна браузера**. Настало время рассмотреть подобный пример:

```
<!DOCTYPE html>
<html>
<head>
        <meta charset = "UTF-8">
        <title>Абсолютное позиционирование относительно предка</title>
<style>
.relative {
position: relative; /* относительное позиционирование элемента */
margin-top: 100px; /* внешний отступ от вержнего края */
width: 400px; /* ширина элемента */
height: 200px; /* высота элемента */
background-color: blue; /* цвет заднего фона */
}
.container {
height: 100px; /* высота элемента */
background-color: yellow; /* цвет заднего фона */
}
.absolute {
position: absolute; /* абсолютное позиционирование элемента */
top: 0; /* смещение от вержнего края */
right: 0; /* смещение от правого края */
width: 50px; /* ширина элемента */
height: 50px; /* высота элемента */
background-color: red; /* цвет заднего фона */
}
</style>
```

```
</head>
        <body>
        <div class = "relative">
                relative
                <div class = "container">
                        container
                        <div class = "absolute">
                                absolute
                        </div>
                </div>
        </body>
</html>
```

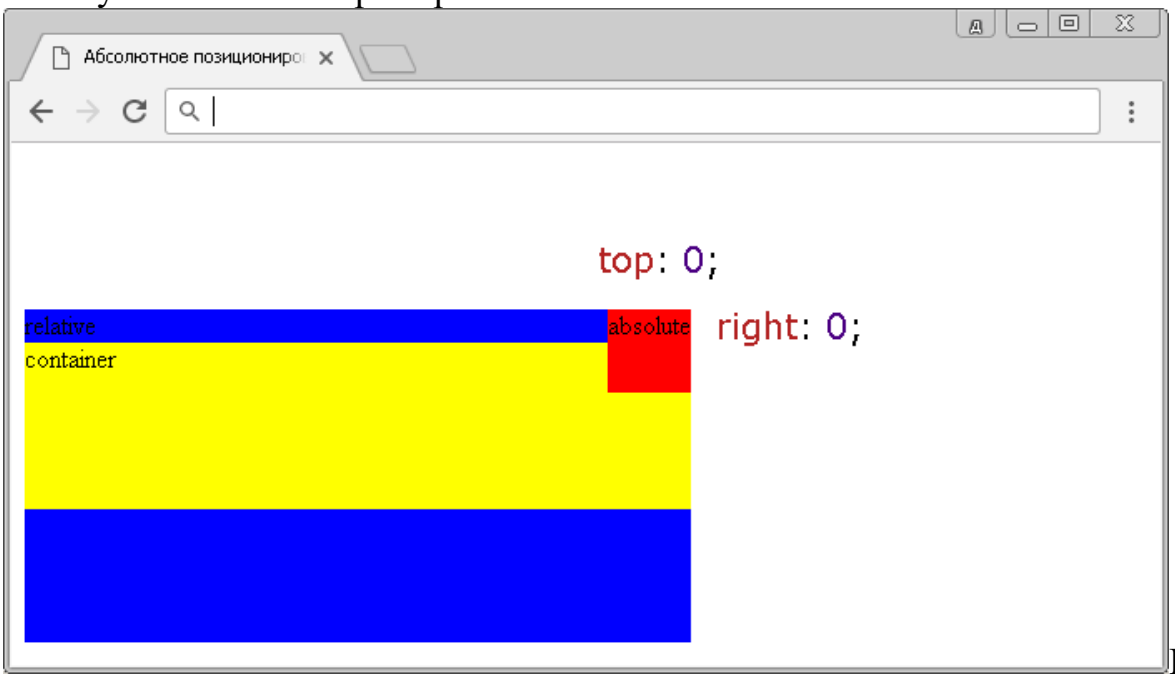
Давайте внимательно разберем, что мы сделали в этом примере:

Для начала мы разместили блок (элемент ** **), который имеет *относительное позиционирование*. Указали для него внутренний отступ от верха (**margin-top**) равный **100 пикселей**, задали ширину, высоту и цвет заднего фона.

Далее внутри него разместили блочный элемент (элемент ** **), который имеет высоту **100 пикселей** и цвет заднего фона *жёлтый*. Как вы понимаете, этот элемент имеет *статическое позиционирование* (значение по умолчанию), так как значение свойства **position** **не наследуется**, и он не унаследовал от родительского блока *относительное позиционирование*.

Затем мы поместили внутри нашего контейнера со *статическим позиционированием* элемент, который имеет *абсолютное позиционирование*. Указали для него ширину и высоту равными **50 пикселей** и цвет заднего фона *красный*. **Обратите внимание на самый важный момент, что этот элемент позиционируется не относительно окна браузера, не относительно родительского элемента, а относительно своего предка, который имеет позиционирование, отличное от статического! В итоге наш элемент мы разместили в верхнем правом углу его предка с *относительным позиционированием*.**

Результат нашего примера:



Пример абсолютного позиционирования элемента относительно его предка.

Давайте подытожим изученную в этой статье учебника информацию о позиционировании элементов:

Статическое позиционирование это классическое **размещение элементов сверху вниз** (элементы отображаются в порядке, как они указаны **в потоке HTML документа**), оно считается значением по умолчанию для всех элементов.

Элемент позиционируется **относительно окна браузера**, если у него *фиксированное позиционирование* (элемент зафиксирован при прокрутке документа).

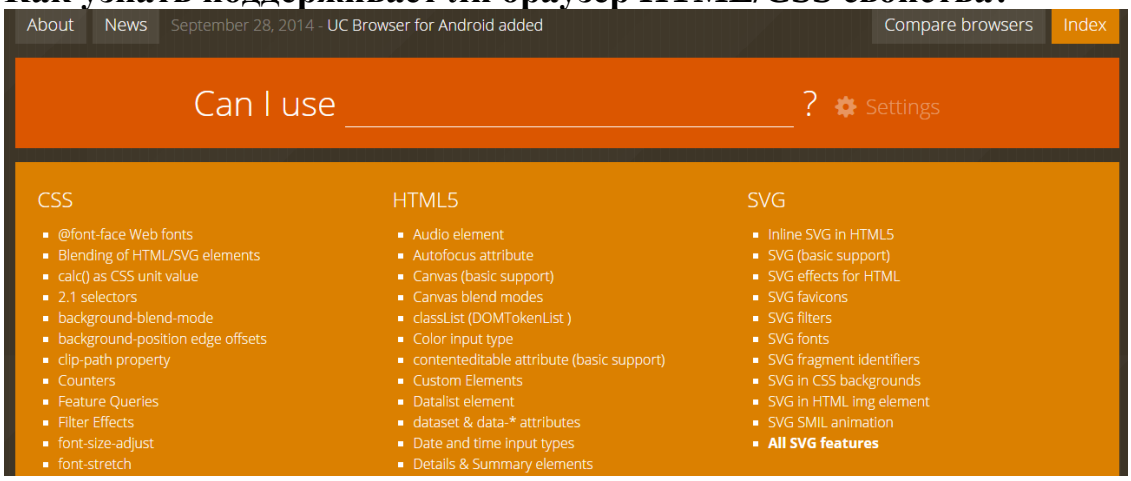
Элемент позиционируется **относительно окна браузера**, если у него *абсолютное позиционирование*, и он **не вложен в элемент, который имеет позиционирование, отличное от *статического***.

Элемент, для которого задано *относительное позиционирование* смещается **относительно положения в потоке документа** (**относительно его текущей позиции**).

Элемент позиционируется **относительно сторон другого элемента** в том случае, если он **имеет предка, или родителя с *абсолютным, относительным* или *фиксированным* позиционированием**.

Лекция 1.14 Поддержка CSS на уровне браузеров

Как узнать поддерживает ли браузер HTML/CSS свойства?



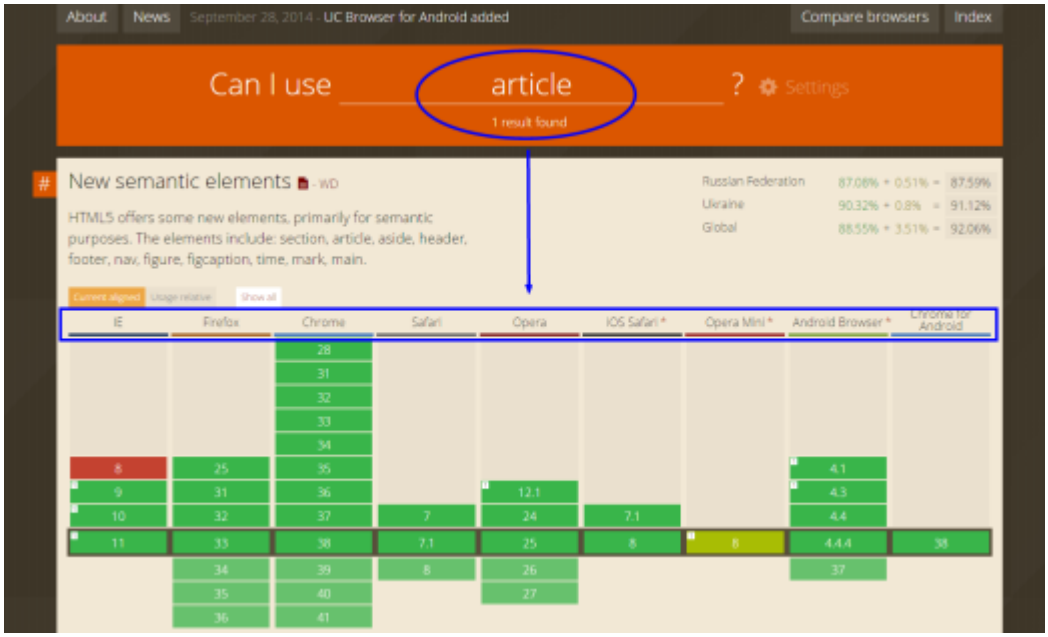
Частенько бывает, что необходимо уточнить, какой процент браузеров поддерживает то или иное CSS свойство или HTML элемент. Дать ответ в такой ситуации поможет отличный online сервис.

Знакомьтесь – мощный онлайн помощник, в вопросе **поддержки браузерами CSS/CSS3 свойств и HTML/HTML5 элементов**:

<u>Сервис по проверке совместимости CSS/HTML свойств с браузерами</u>

Как узнать поддерживает ли браузер HTML/CSS свойство?

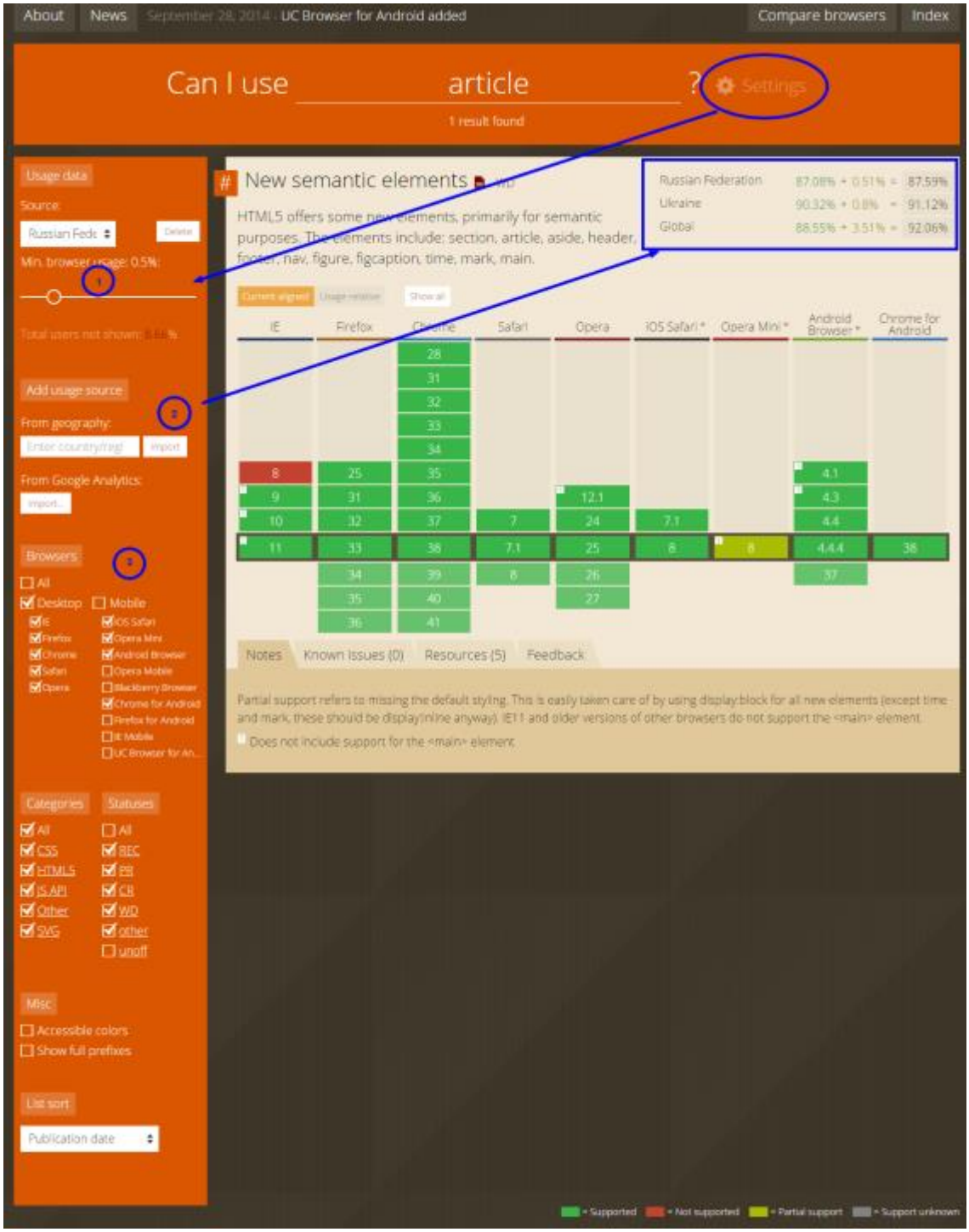
Узнать поддерживает браузер нужное вам свойство – очень просто, достаточно вбить его в поиск:



В скрине выше я вбил тег HTML5 **article** и мгновенно получил график поддержки данного тега, разными версиями браузеров. Все наглядно и очень просто

Обратите внимание на настройки и возможности

У этого сервиса есть очень полезные настройки:



Поддержка CSS на уровне браузеров

Лекция 1.15 Введение в JavaScript.

Преимущества и ограничения программ, работающих на стороне клиента

Давайте посмотрим, что такого особенного в JavaScript, чего можно достичь с его помощью и какие другие технологии хорошо с ним работают.

Что такое JavaScript?

Изначально *JavaScript* был создан, чтобы *«сделать веб-страницы живыми»*.

Программы на этом языке называются *скриптами*. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.

Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

Это отличает JavaScript от другого языка – Java

.

Почему JavaScript?

Когда JavaScript создавался, у него было другое имя – «LiveScript». Однако, язык Java был очень популярен в то время, и было решено, что позиционирование JavaScript как «младшего брата» Java будет полезно.

Со временем JavaScript стал полностью независимым языком со своей собственной спецификацией, называющейся ECMAScript, и сейчас не имеет никакого отношения к Java.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» JavaScript.

У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript».

Разные движки имеют разные «кодовые имена». Например:

V8 – в Chrome и Opera.

SpiderMonkey – в Firefox.

…Ещё есть «Trident» и «Chakra» для разных версий IE, «ChakraCore» для Microsoft Edge, «Nitro» и «SquirrelFish» для Safari и т.д.

Эти названия полезно знать, так как они часто используются в статьях для разработчиков. Мы тоже будем их использовать. Например, если «функциональность X поддерживается V8», тогда «X», скорее всего, работает в Chrome и Opera.

Как работают движки?

Движки сложны. Но основы понять легко.

Движок (встроенный, если это браузер) читает («парсит») текст скрипта.

Затем он преобразует («компилирует») скрипт в машинный язык.

После этого машинный код запускается и работает достаточно быстро.

Движок применяет оптимизации на каждом этапе. Он даже просматривает скомпилированный скрипт во время его работы, анализируя проходящие через него данные, и применяет оптимизации к машинному коду, полагаясь на полученные знания. В результате скрипты работают очень быстро.

Что может JavaScript в браузере?

Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого.

Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, Node.JS поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д.

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

Например, в браузере JavaScript может:

Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.

Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.

Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (технологии AJAX и COMET).

Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.

Запоминать данные на стороне клиента («local storage»).

Чего НЕ может JavaScript в браузере?

Возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесения ущерба данным пользователя.

Примеры таких ограничений включают в себя:

JavaScript на веб-странице не может читать/записывать произвольные файлы на жёстком диске, копировать их или запускать программы. Он не имеет прямого доступа к системным функциям ОС.

Современные браузеры позволяют ему работать с файлами, но с ограниченным доступом, и предоставляют его, только если пользователь выполняет определённые действия, такие как «перетаскивание» файла в окно браузера или его выбор с помощью тега <input>.

Существуют способы взаимодействия с камерой/микрофоном и другими устройствами, но они требуют явного разрешения пользователя. Таким образом, страница с поддержкой JavaScript не может незаметно включить веб-камеру, наблюдать за происходящим и отправлять информацию в ФСБ.

Различные окна/вкладки не знают друг о друге. Иногда одно окно, используя JavaScript, открывает другое окно. Но даже в этом случае JavaScript с одной страницы не имеет доступа к другой, если они пришли с разных сайтов (с другого домена, протокола или порта).

Это называется «Политика одинакового источника» (Same Origin Policy). Чтобы обойти это ограничение, обе страницы должны согласиться с этим и содержать JavaScript-код, который специальным образом обменивается данными.

Это ограничение необходимо, опять же, для безопасности пользователя. Страница http://anysite.com, которую открыл пользователь, не должна иметь доступ к другой вкладке браузера с URL http://gmail.com и воровать информацию оттуда.

JavaScript может легко взаимодействовать с сервером, с которого пришла текущая страница. Но его способность получать данные с других сайтов/доменов ограничена. Хотя это возможно в принципе, для чего требуется явное согласие (выраженное в заголовках HTTP) с удалённой стороной. Опять же, это ограничение безопасности.

Подобные ограничения не действуют, если JavaScript используется вне браузера, например — на сервере. Современные браузеры предоставляют плагины/расширения, с помощью которых можно запрашивать дополнительные разрешения.

Что делает JavaScript особенным?

Как минимум, *три* сильные стороны JavaScript:

Полная интеграция с HTML/CSS.

Простые вещи делаются просто.

Поддерживается всеми основными браузерами и включён по умолчанию.

JavaScript – это единственная браузерная технология, сочетающая в себе все эти три вещи.

Вот что делает JavaScript особенным. Вот почему это самый распространённый инструмент для создания интерфейсов в браузере.

Хотя, конечно, JavaScript позволяет делать приложения не только в браузерах, но и на сервере, на мобильных устройствах и т.п.

Языки «над» JavaScript

Синтаксис JavaScript подходит не под все нужды. Разные люди хотят иметь разные возможности.

Это естественно, потому что проекты разные и требования к ним тоже разные.

Так, в последнее время появилось много новых языков, которые *транспилируются* (конвертируются) в JavaScript, прежде чем запустятся в браузере.

Современные инструменты делают транспиляцию очень быстрой и прозрачной, фактически позволяя разработчикам писать код на другом языке, автоматически преобразуя его в JavaScript «под капотом».

Примеры таких языков:

CoffeeScript добавляет «синтаксический сахар» для JavaScript. Он вводит более короткий синтаксис, который позволяет писать чистый и лаконичный код. Обычно такое нравится Ruby-программистам.

TypeScript концентрируется на добавлении «строгой типизации» для упрощения разработки и поддержки больших и сложных систем. Разработан Microsoft.

Flow тоже добавляет типизацию, но иначе. Разработан Facebook.

Dart стоит особняком, потому что имеет собственный движок, работающий вне браузера (например, в мобильных приложениях). Первоначально был предложен Google, как замена JavaScript, но на данный момент необходима его транспиляция для запуска так же, как для вышеперечисленных языков.

Есть и другие. Но даже если мы используем один из этих языков, мы должны знать JavaScript, чтобы действительно понимать, что мы делаем.

Итого

JavaScript изначально создавался только для браузера, но сейчас используется на многих других платформах.

Сегодня JavaScript занимает уникальную позицию в качестве самого распространённого языка для браузера, обладающего полной интеграцией с HTML/CSS.

Многие языки могут быть «транспилированы» в JavaScript для предоставления дополнительных функций. Рекомендуется хотя бы кратко рассмотреть их после освоения JavaScript.

Лекция 1.16 Внедрение JavaScript в HTML-документ. основы синтаксиса

Прежде чем начать писать на языке JavaScript, нужно сначала его внедрить в HTML-документ, всего существует 3 способа внедрения языка JavaScript в HTML-документ, рассмотрим 2 из них.

Первый способ внедрения JavaScript сценария

Для того, чтобы **внедрить JavaScript в HTML-документ**, нужно использовать теги <script> </script>. Между этими тегами, помещают JavaScript-код:

```
<script>
 JavaScript-код...
</script>
```

Теги <script> </script> с расположенным между ними JavaScript-кодом, называют: программой, скриптом или сценарием. Можете называть так, как вам удобно, но обычно используют слово «скрипт».

Теги <script> </script> обычно располагают в голове HTML-документа, между тегами <head> </head>, однако их можно располагать и в теле HTML-документа, между тегами <body> </body>

Разместим сценарий в голове HTML-документа:

```
<html>
 <head>
  <title>Страница о снежном барсе</title>
  <script>
   alert("Привет это JavaScript!");
  </script>
 </head>
 <body>
  <h1>Снежный барс</h1>
  <p>
   Снежный барс (ирбис, ак барс) - крупное хищное млекопитающее из семейства
   кошачьих. Обитает в горных массивах Афганистана, Бирмы, Бутана, Индии,
   Казахстана, Киргизстана, Китая, Монголии, Непала, Пакистана, России,
   Таджикистана и Узбекистана. Ирбис отличается тонким, длинным, гибким телом,
   относительно короткими лапами, небольшой головой и очень длинным хвостом.
   Длина снежного барса вместе с хвостом составляет 200-230 см, вес до 55 кг.
   Окраска меха светлая, дымчато-серая с кольцеобразными или сплошными тёмными
   пятнами.</p>
  <p>
   Охотится снежный барс, в основном, на горных козлов и баранов, также в его
   рационе встречаются кабаны, фазаны и даже суслики. В силу труднодоступности
   местообитания вида, ирбисы до сих пор остаются малоизученными. Однако по
   приблизительным оценкам их количество варьируется в районе около 10 тысяч
   особей. По состоянию на 2013 год, охота на снежных барсов повсеместно
   запрещена.</p>
 </body>
</html>
```

Открыв данный HTML-документ в браузере, появится небольшое окошко, с сообщением: Привет это JavaScript!, помимо сообщения, в этом окошке будет кнопка ОК, нажав на которую, окошко закроется.

Второй способ внедрения JavaScript сценария

Первый способ внедрения JavaScript кода, удобно использовать при начале изучения языка JavaScript. Но в повседневной практике обычно используют второй способ внедрения.

Разместите между тегами <head> следующий код:

<script src="myscript.js"></script>

В той же папке где хранится ваша HTML-страница, создайте файл с именем myscript.js, в данный файл вы можете записывать любой JavaScript-код, но уже без использования тегов <script>

alert("Привет это JavaScript");

Рассмотрим функцию alert()

Давайте обратим внимание на строку alert("Привет это JavaScript");

alert(), это встроенная функция языка JavaScrpt, которая выводит небольшое окошко с текстом, текст нужно записывать внутри скобок и обрамлять двойными " " или одинарными кавычками ' '.

Функция alert() используется для вывода на экран, результатов работы (вычислений) программы (скрипта). Обычно в программировании на JavaScript, пользователь вводит данные в HTML-форму, а результаты обработки этих данных потом появляются на следующей странице или под формой.

Для того, чтобы использовать HTML-формы совместно с JavaScript, нужно знать DOM, поэтому мы пока будем использовать функцию alert(), как более простой и надёжный вариант, а данные для обработки будем вводить в самой программе.

Попробуйте, теперь записать другой текст в функцию alert() например своё имя:

alert("ВашеИмя");

Затем сохраните HTML-документ и обновите HTML-страницу. Должно появится окошко с надписью ВашеИмя. Во время изучения синтаксиса JavaScript, мы часто будем работать с функцией alert().

Помимо обычного текста, функция alert() также может выводить результаты вычислений.

Запишите в функцию alert(), следующие вычисления (вычисления записываются без кавычек):

2 + 2

10 * 7

33 / 5

В первом случае, строка alert(2 + 2); выведет окошко с сообщением 4, во втором случае 70, в третьем 6.6

Лекция 1.17 Внедрение JavaScript в HTML-документ. программное изменение содержания документа.

Программное изменение формата документа. Программное изменение положения элементов.

Элементы языка JavaScript

JavaScript позволяет "оживить" веб-страницу. Это реализуется путем добавления к статическому описанию фрагмент исполняемого кода. JavaScript -сценарий может взаимодействовать с любыми компонентами HTML -документа и реагировать на изменение их состояния.

JavaScript не является строго *типизированным языком*, в переменных могут храниться практически любые типы данных.

Как и программа на языке Java, сценарий JavaScript выполняется под управлением интерпретатора. Однако если Java -приложение или Java -*аплет* компилируется в байтовый код, то сценарий JavaScript интерпретируется на уровне исходного текста.

Следует отметить, что языковые конструкции JavaScript совпадают с соответствующими средствами C++ и Java.

Структура сценария

Сценарием JavaScript считается фрагмент кода, расположенный между дескрипторами <SCRIPT> и </SCRIPT> :

Текст HTML-документа

<SCRIPT>

Код сценария

</SCRIPT>

Текст HTML-документа

Переменные

В сценариях JavaScript переменные могут хранить данные любых типов: числа, строки текста, логические значения, ссылки на объекты, а также специальные величины, например "нулевое" значение null или значение *NaN*, которое сообщает о недопустимости операции.

Переменная в языке JavaScript объявляется с помощью ключевого слова var. Так, например, выражение

>

var selected = "first item";

создает переменную с именем select и присваивает ей в качестве значения строку символов "first item". Переменные могут объявляться также автоматически. Это происходит при присвоении значения переменной, не встречавшейся ранее в данном сценарии. Так, в следующем примере создается переменная с именем rating, которой присваивается числовое значение, равное 512.5:

rating = 512.5;

Объекты

В языке JavaScript не предусмотрены средства для работы с классами в том виде, в котором они реализованы в C++ или Java. Разработчик сценария не может создать подкласс на основе существующего класса, переопределить метод или выполнить какую-либо другую операцию с классом. Сценарию, написанному на языке JavaScript, в основном доступны лишь готовые объекты. Построение нового объекта приходится выполнять лишь в редких случаях.

Объекты содержат *свойства* (свойства объектов можно сравнить с переменными) и *методы*. Объекты, а также их свойства и методы идентифицируются именами. Объектами являются формы, изображения, гипертекстовые ссылки и другие компоненты веб-страницы, HTML -документ, отображаемый в окне браузера, окно браузера и даже сам браузер. В процессе работы JavaScript сценарий обращается к этим объектам, получает информацию и управляет ими.

Кроме того, разработчику сценария на языке JavaScript доступны объекты, не связанные непосредственно с HTML -документом. Их называют *предопределенными*, или *независимыми* объектами. С помощью этих объектов можно реализовать массив, описать дату и время, выполнить математические вычисления и решить некоторые другие задачи.

Первый объект, с которым необходимо познакомиться, чтобы написать простейший сценарий, - это объект document, который описывает HTML документ, отображаемый в окне браузера. Ниже приведен исходный текст веб-страницы, содержащей сценарий, действия которого сводятся к выводу строки текста в окне браузера.

```
<HTML>
<HEAD>
<TITLE>Первый сценарий JavaScript</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
document.write("Проверка сценария JavaScript");
</SCRIPT>
</BODY>
</HTML>
```

Имена *чувствительны к регистрам* символов, и если вы попытаетесь обратиться к текущему документу по имени Document, интерпретатор JavaScript отобразит сообщение об ошибке.

Основное назначение сценариев JavaScript - создавать динамически изменяющиеся объекты, корректировать содержимое HTML -документов в зависимости от особенностей окружения, осуществлять взаимодействие с пользователем и т.д.

Операции

Набор операторов в JavaScript, их назначение и правила использования в основном совпадают с принятыми в языке C++. Исключением является операция задаваемая символом "+".

В JavaScript символ "+" определяет как *суммирование* числовых значений, так и *конкатенацию* строк.

Так, например, в результате вычисления выражения

```
sum = 47 + 21;
```

переменной sum будет присвоено значение 68, а после выполнения операции

```
sum = "строка 1 " + "строка 2";
```

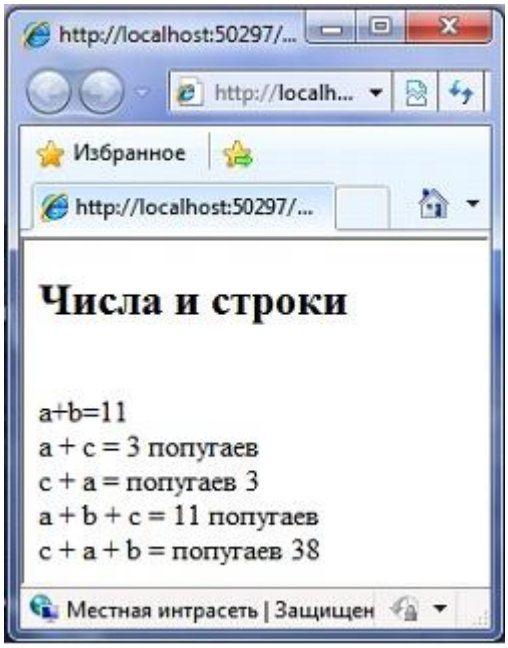
в переменную sum будет записана последовательность символов " *строка 1 строка 2* ".

Рассмотрим еще один пример:

```
<HTML>
<BODY>
<H2>Числа и строки</H2><BR>
```

```
<SCRIPT LANGUAGE="JavaScript">

var a = 3;
var b = 8;
var c = " попугаев ";

document.write("a+b="); document.write(a + b); document.write("<BR>");
document.write( "a + c = "); document.write(a+c);
document.write("<BR>");
document.write("c + a = "); document.write (c + a);
document.write ("<BR>");
document.write ("a + b + c = "); document.write(a + b + c);
document.write("<BR>");
document.write("c + a + b = "); document.write(c + a + b);
document.write("<BR>");

</SCRIPT>
</BODY>
</HTML>
```

В окне браузера приведенный выше HTML -код выглядит так, как показано на рис.7.2



Скриншот веб-страницы с JavaScript-сценарием

Первая строка отображает результат суммирования двух числовых значений, вторая и третья - результат конкатенации строки и *символьного представления* числа. Если операция суммирования чисел предшествует конкатенации, JavaScript вычисляет сумму чисел, представляет ее в символьном виде, затем производит конкатенацию двух строк. Если же первой в выражении указана *операция конкатенации*, то JavaScript сначала преобразует числовые значения в символьный вид, а затем выполняет конкатенацию строк.

Управляющие конструкции

Управляющие конструкции, используемые в языке C++, в основном применимы и в сценариях JavaScript.

В JavaScript дополнительно определены языковые конструкции, отсутствующие в C++, а именно: операторы for...in и with.

В примере 1 с помощью *оператора цикла* на веб-странице формируется таблица умножения чисел.

Пример 1.

```
<html>
<body>
<table>
<script language="JavaScript">
  document.write("<tr><td> </td>");
  for (i = 1; i < 10; i++) document.write("<td>"+i+" </td>");
  document.write("</tr>");
  for (i = 1; i < 10; i++)
  {
    document.write("<tr><td>" + i + " </td>");
    for (j = 1; j < 10; j++)
    {
      document.write("<td  bgcolor='#00ffa0'>" + (i*j) + " </td>");      }
    document.write("</tr>");
  }

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

Отдельного внимания заслуживает оператор new. Несмотря на то, что большинство объектов уже созданы браузером и доступны сценарию, в некоторых случаях приходится создавать объекты в процессе работы. Это относится к предопределенным объектам и объектам, определяемым разработчиком сценария. Для создания объекта используется оператор new, который вызывается следующим образом:

переменная = new   тип_ объекта (параметры)

Функции

Формат *объявления функции* выглядит следующим образом:

function имя функции ([параметры]) тело функции

Объявление функции начинается с ключевого слова function. Так же, как и в языке C для идентификации функции используется имя, при вызове функции могут передаваться параметры, а по окончании выполнения возвращаться значение. Однако, в отличие от C, тип возвращаемого значения и типы параметров не задаются. Ниже показаны два способа вызова функции

имя_функции ([параметры]) ;

переменная = имя функции ([параметры]) ;

Во втором случае значение, возвращаемое функцией, присваивается указанной переменной.

Лекция 1.18 Типы данных, переменные, выражения

Отличие JavaScript от других языков web-программирования.

Типы данных

Переменная в JavaScript может содержать любые данные. В один момент там может быть строка, а в другой – число:

```
// Не будет ошибкой
let message = "hello";
message = 123456;
```

Языки программирования, в которых такое возможно, называются «динамически типизированными». Это значит, что типы данных есть, но переменные не привязаны ни к одному из них.

Есть восемь основных типов данных в JavaScript. В этой главе мы рассмотрим их в общем, а в следующих главах поговорим подробнее о каждом.

Число

```
let n = 123;
n = 12.345;
```

Числовой тип данных (number) представляет как целочисленные значения, так и числа с плавающей точкой.

Существует множество операций для чисел, например, умножение *, деление /, сложение +, вычитание - и так далее.

Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: Infinity, -Infinity и NaN.

Infinity представляет собой математическую бесконечность ∞. Это особое значение, которое больше любого числа.

Мы можем получить его в результате деления на ноль:

```
alert( 1 / 0 ); // Infinity
```

Или задать его явно:

```
alert( Infinity ); // Infinity
```

NaN означает вычислительную ошибку. Это результат неправильной или неопределённой математической операции, например:

```
alert( "не число" / 2 ); // NaN, такое деление является ошибкой
```

Значение NaN «прилипчиво». Любая операция с NaN возвращает NaN:

```
alert( "не число" / 2 + 5 ); // NaN
```

Если где-то в математическом выражении есть NaN, то результатом вычислений с его участием будет NaN.

Математические операции – безопасны

Математические операции в JavaScript «безопасны». Мы можем делать что угодно: делить на ноль, обращаться со строками как с числами и т.д.

Скрипт никогда не остановится с фатальной ошибкой (не «умрёт»). В худшем случае мы получим NaN как результат выполнения.

Специальные числовые значения относятся к типу «число». Конечно, это не числа в привычном значении этого слова.

Подробнее о работе с числами мы поговорим в главе Числа.

BigInt

В JavaScript тип «number» не может содержать числа больше, чем 2^{53} (или меньше, чем -2^{53} для отрицательных). Это техническое ограничение вызвано их внутренним представлением. 2^{53} – это достаточно большое число, состоящее из 16 цифр, поэтому

чаще всего проблем не возникает. Но иногда нам нужны действительно гигантские числа, например в криптографии или при использовании метки времени («timestamp») с микросекундами.

Тип BigInt был добавлен в JavaScript, чтобы дать возможность работать с целыми числами произвольной длины.

Чтобы создать значение типа BigInt, необходимо добавить n в конец числового литерала:

```
// символ "n" в конце означает, что это BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```

Так как BigInt числа нужны достаточно редко, мы рассмотрим их в отдельной главе BigInt

.

Поддержка

В данный момент BigInt поддерживается только в браузерах Firefox и Chrome, но не поддерживается в Safari/IE/Edge.

Строка

Строка (string) в JavaScript должна быть заключена в кавычки.

```
let str = "Привет";
let str2 = 'Одинарные кавычки тоже подойдут';
let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

В JavaScript существует три типа кавычек.

Двойные кавычки: "Привет".

Одинарные кавычки: 'Привет'.

Обратные кавычки: `Привет`.

Двойные или одинарные кавычки являются «простыми», между ними нет разницы в JavaScript.

Обратные кавычки же имеют «расширенную функциональность». Они позволяют нам встраивать выражения в строку, заключая их в ${…}. Например:

```
let name = "Иван";
// Вставим переменную
alert( `Привет, ${name}!` ); // Привет, Иван!
// Вставим выражение
alert( `результат: ${1 + 2}` ); // результат: 3
```

Выражение внутри ${…} вычисляется, и его результат становится частью строки. Мы можем положить туда всё, что угодно: переменную name или выражение 1 + 2, или что-то более сложное.

Обратите внимание, что это можно делать только в обратных кавычках. Другие кавычки не имеют такой функциональности встраивания!

```
alert( "результат: ${1 + 2}" ); // результат: ${1 + 2} (двойные кавычки ничего не делают)
```

Мы рассмотрим строки более подробно в главе Строки.

Нет отдельного типа данных для одного символа.

В некоторых языках, например C и Java, для хранения одного символа, например "a" или "%", существует отдельный тип. В языках C и Java это char.

В JavaScript подобного типа нет, есть только тип string. Строка может содержать один символ или множество.

Булевый (логический) тип

Булевый тип (boolean) может принимать только два значения: true (истина) и false (ложь).

Такой тип, как правило, используется для хранения значений да/нет: true значит «да, правильно», а false значит «нет, не правильно».

Например:

```
let nameFieldChecked = true; // да, поле отмечено
let ageFieldChecked = false; // нет, поле не отмечено
```

Булевые значения также могут быть результатом сравнений:

```
let isGreater = 4 > 1;
alert( isGreater ); // true (результатом сравнения будет "да")
```

Мы рассмотрим булевые значения более подробно в главе Логические операторы.

Значение «null»

Специальное значение null не относится ни к одному из типов, описанных выше.

Оно формирует отдельный тип, который содержит только значение null:

```
let age = null;
```

В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках.

Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

В приведённом выше коде указано, что переменная age неизвестна или не имеет значения по какой-то причине.

Значение «undefined»

Специальное значение undefined также стоит особняком. Оно формирует тип из самого себя так же, как и null.

Оно означает, что «значение не было присвоено».

Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет undefined:

```
let x;
alert(x); // выведет "undefined"
```

Технически мы можем присвоить значение undefined любой переменной:

```
let x = 123;
x = undefined;
alert(x); // "undefined"
```

…Но так делать не рекомендуется. Обычно null используется для присвоения переменной «пустого» или «неизвестного» значения, а undefined – для проверок, была ли переменная назначена.

Объекты и символы

Тип object (объект) – особенный.

Все остальные типы называются «примитивными», потому что их значениями могут быть только простые значения (будь то строка или число, или что-то ещё). Объекты же используются для хранения коллекций данных или более сложных объектов. Мы разберёмся с ними позднее в главе Объекты после того, как узнаем больше о примитивах.

Тип symbol (символ) используется для создания уникальных идентификаторов объектов. Мы упоминаем здесь о нём для полноты картины, изучим этот тип после объектов.

Оператор typeof

Оператор typeof возвращает тип аргумента. Это полезно, когда мы хотим обрабатывать значения различных типов по-разному или просто хотим сделать проверку.

У него есть два синтаксиса:

Синтаксис оператора: typeof x.

Синтаксис функции: typeof(x).

Другими словами, он работает со скобками или без скобок. Результат одинаковый.

Вызов typeof x возвращает строку с именем типа:

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof 10n // "bigint"
typeof true // "boolean"
typeof "foo" // "string"
typeof Symbol("id") // "symbol"
typeof Math // "object"  (1)
typeof null // "object"  (2)
typeof alert // "function"  (3)
```

Последние три строки нуждаются в пояснении:

Math – это встроенный объект, который предоставляет математические операции и константы. Мы рассмотрим его подробнее в главе Числа. Здесь он служит лишь примером объекта.

Результатом вызова typeof null является "object". Это неверно. Это официально признанная ошибка в typeof, сохранённая для совместимости. Конечно, null не является объектом. Это специальное значение с отдельным типом. Повторимся, это ошибка в языке.

Вызов typeof alert возвращает "function", потому что alert является функцией. Мы изучим функции в следующих главах, где заодно увидим, что в JavaScript нет специального типа «функция». Функции относятся к объектному типу. Но typeof обрабатывает их особым образом, возвращая "function". Формально это неверно, но очень удобно на практике.

Итого

В JavaScript есть 8 основных типов.

number для любых чисел: целочисленных или чисел с плавающей точкой, целочисленные значения ограничены диапазоном $\pm 2^{53}$.

bigint для целых чисел произвольной длины.

string для строк. Строка может содержать один или больше символов, нет отдельного символьного типа.

boolean для true/false.

null для неизвестных значений – отдельный тип, имеющий одно значение null.

undefined для неприсвоенных значений – отдельный тип, имеющий одно значение undefined.

object для более сложных структур данных.

symbol для уникальных идентификаторов.

Оператор typeof позволяет нам увидеть, какой тип данных сохранён в переменной.

Имеет две формы: typeof x или typeof(x).

Возвращает строку с именем типа. Например, "string".

Для `null` возвращается `"object"` – это ошибка в языке, на самом деле это не объект.

В следующих главах мы сконцентрируемся на примитивных значениях, а когда познакомимся с ними, перейдём к объектам.

Задачи

Шаблонные строки

важность: 5

Что выведет этот скрипт?

```
let name = "Ilya";
alert( `hello ${1}` ); // ?
alert( `hello ${"name"}` ); // ?
alert( `hello ${name}` ); // ?
```

Лекция 1.19 Арифметические операторы в JavaScript.

Недостатки, достоинства JavaScript. Обычные применения языка

Математические операции являются одними из самых базовых и универсальных функций любого языка программирования. В JavaScript числа часто используются в общих задачах, таких как определение размеров окна браузера, вычисление окончательной цены денежной транзакции или расстояния между элементами в документе сайта.

Чтобы быть хорошим разработчиком, иметь высокие навыки в математике не обязательно, однако для этого важно знать, какие типы операций доступны в JavaScript и как использовать их для выполнения практических задач.

В отличие от других языков программирования, JavaScript имеет только один числовой тип данных; он не разделяет целые числа и числа с плавающей точкой.

Это руководство расскажет об арифметических операторах, операторах присваивания и порядке операций с числовыми данными JavaScript.

Арифметические операторы

Арифметические операторы – это символы, которые определяют математические операции и возвращают результат. К примеру, в 3 + 7 = 10 символ + определяет синтаксис операции сложения.

Многие операторы JavaScript знакомы вам из базовой математики, но есть также и несколько дополнительных операторов.

Все арифметические операторы JavaScript представлены в следующей таблице.

Оператор	Синтаксис	Пример	Определение
Сложение	+	x + y	Сумма x и y
Вычитание	—	x — y	Разница между x и y
Умножение	*	x * y	Производное x и y
Деление	/	x / y	Частное x и y
Модуль	%	x % y	Остаток x / y
Возведение в степень	**	x ** y	x в степени y
Инкремент	++	x++	x плюс один
Декремент	—	x—	x минус один

Сложение и вычитание

Операторы сложения и вычитания доступны в JavaScript и могут использоваться для нахождения суммы и разности числовых значений. JavaScript имеет встроенный калькулятор, а математические операции могут выполняться непосредственно в консоли.

Знак плюса позволяет складывать числа, например:

```
10 + 20;
30
```

Помимо операций с простыми числами JavaScript позволяет присваивать числа переменным и выполнять с ними вычисления. Для примера можно присвоить числовые значения переменным x и y, а результат поместить в z.

```
// Assign values to x and y
let x = 10;
let y = 20;
// Add x and y and assign the sum to z
let z = x + y;
console.log(z);
30
```

Символ минус позволяет вычитать числа или выполнять операции с переменными:

```
// Assign values to x and y
let x = 10;
let y = 20;
// Subtract x from y and assign the difference to z
let z = y - x;
console.log(z);
10
```

Также можно складывать и вычитать отрицательные числа и числа с плавающей запятой.

```
// Assign values to x and y
let x = -5.2;
let y = 2.5;
// Subtract y from x and assign the difference to z
let z = x - y;
console.log(z);
-7.7
```

В JavaScript есть одна интересная особенность, которую следует учитывать и знать, — это результат сложения числа и строки. Мы знаем, что 1 + 1 должно равняться 2, но это уравнение выдаст неожиданный результат.

Читайте также: Работа со строками в JavaScript

```
let x = 1 + "1";
console.log(x);
typeof x;
11
'string'
```

Вместо сложения чисел JavaScript преобразует все выражение в строки и объединяет их. Важно быть осторожным с динамической типизацией JavaScript, поскольку она может иметь нежелательные результаты.

Сложение и вычитание в JavaScript часто используются для прокрутки панели навигации.

```
function scrollToId() {
const navHeight = 60;
window.scrollTo(0, window.pageYOffset - navHeight);
```

```
}
window.addEventListener('hashchange', scrollToId);
```

В этом случае панель будет прокручиваться на 60 пикселей от id.

Умножение и деление

Операторы умножения и деления JavaScript используются для поиска производного и частного числовых значений.

Звездочка является оператором умножения.

```
// Assign values to x and y
let x = 20;
let y = 5;
// Multiply x by y to get the product
let z = x * y;
console.log(z);
```

100

Умножение можно использовать для расчета цены товара после введения налога с продаж.

```
const price = 26.5;    // Price of item before tax
const taxRate = 0.082; // 8.2% tax rate
// Calculate total after tax to two decimal places
let totalPrice = price + (price * taxRate);
totalPrice.toFixed(2);
console.log("Total:", totalPrice);
```

Total: 28.67

Слеш – оператор деления.

```
// Assign values to x and y
let x = 20;
let y = 5;
// Divide y into x to get the quotient
let z = x / y;
console.log(z);
```

4

Деление особенно полезно при расчете времени, например, при вычислении количества часов или процента правильных ответов в тесте.

Модуль числа

Модуль – еще один арифметический оператор, менее популярный, чем предыдущие. Представлен символом %. Он возвращает остаток при делении первого числа на второе.

К примеру, мы знаем, что 9 делится на 3 без остатка:

```
9 % 3;
```

0

Модуль числа позволяет определить четное или нечетное число, например:

```
// Initialize function to test if a number is even
const isEven = x => {
// If the remainder after dividing by two is 0, return true
if (x % 2 === 0) {
return true;
}
// If the number is odd, return false
```

```
return false;
}
// Test the number
isEven(12);
true
```

В этом примере 12 делится на 2, следовательно, это четное число.

В программировании модуль числа часто используется в сочетании с условными операторами.

Возведение в степень

Возведение в степень – один из самых новых операторов JavaScript. Синтаксис возведения в степень – две звездочки подряд (**).

К примеру, 10 в пятой степени (10^5) записывается так:

```
10 ** 5;
```

Операция 10 ** 5 имеет тот же результат, что 10 * 10, повторенная 5 раз.

```
10 * 10 * 10 * 10 * 10;
```

Также эту операцию можно записать с помощью метода Math.pow().

```
Math.pow(10, 5);
100000
```

Использование оператора возведения в степень – быстрый способ определить степень заданного числа, но, как обычно, при выборе между методом и оператором важно быть последовательными и писать код в одном стиле.

Инкремент и декремент

Операторы инкремента и декремента увеличивают или уменьшают числовое значение переменной на единицу. Они представлены двумя плюсами (++) или двумя минусами (—) и часто используются в циклах.

Обратите внимание: операторы инкремента и декремента могут использоваться только с переменными. Попытка использовать их с простыми числами приведет к ошибке.

```
7++
Uncaught ReferenceError: Invalid left-hand side expression in postfix operation
```

Операторы инкремента и декремента можно классифицировать как префиксные и постфиксные операции, в зависимости от того, где по отношению к переменной размещен оператор.

Префиксный инкремент записывается как ++x.

```
// Set a variable
let x = 7;
// Use the prefix increment operation
let prefix = ++x;
console.log(prefix);
8
```

Значение x увеличилось на 1. Постфиксный инкремент пишется как y++.

```
// Set a variable
let y = 7;
// Use the prefix increment operation
let postfix = y++;
console.log(postfix);
7
```

Постфиксная операция не увеличила значение. Это значение не будет увеличиваться до тех пор, пока выражение не будет оценено. Для этого нужно запустить операцию дважды:

```
let y = 7;
y++;
y++;
console.log(y);
```

8

Чаще всего эти операторы встречаются в циклах. В данном цикле for оператор запускается 10 раз, начиная с 0.

```
// Run a loop ten times
for (let i = 0; i < 10; i++) {
console.log(i);
}
```

В этом примере итерация цикла выполняется с помощью оператора инкремента.

Проще говоря, x++ можно воспринимать как сокращение от x = x + 1, а x—как сокращение от x = x – 1.

Операторы присваивания

Одним из наиболее часто используемых операторов является оператор присваивания, который уже встречался в этом мануале. Он представлен знаком равенства (=). Символ = используется для присвоения значения справа переменной слева.

```
// Assign 27 to age variable
let age = 27;
```

Помимо стандартного оператора присваивания JavaScript имеет составные операторы присваивания, которые комбинируют арифметический оператор с оператором =.

К примеру, оператор добавления начнет с исходного значения и добавит к нему новое значение.

```
// Assign 27 to age variable
let age = 27;
age += 3;
console.log(age);
```

30

По сути, age += 3 – то же самое, что и age = age + 3.

Все арифметические операторы можно объединять с оператором присваивания. Ниже приведена справочная таблица операторов присваивания в JavaScript.

Оператор	Синтаксис
Присваивание	=
Присваивание со сложением	+=
Присваивание с вычитанием	-=

Оператор	Синтаксис
Присваивание с умножением	*=
Присваивание с делением	/=
Присваивание по модулю	%=
Присваивание с возведением в степень	**=

Составные операторы присваивания часто используются в циклах, как инкременты и декременты.

Приоритет операторов

Операторы выполняются в порядке приоритетности, как и в обычной математике.

К примеру, умножение имеет более высокий приоритет, чем сложение.

```
// First multiply 3 by 5, then add 10
10 + 3 * 5;
25
```

Если сначала нужно выполнить операцию сложения, возьмите ее в круглые скобки – такие операции всегда имеют наивысший приоритет.

```
// First add 10 and 3, then multiply by 5
(10 + 3) * 5;
65
```

Ниже вы найдете таблицу приоритета арифметических операторов в JavaScript. Для инкремента и декремента постфикс имеет более высокий приоритет, чем префикс.

Инкремент/декремент, умножение/деление и сложение/вычитание имеют одинаковый уровень приоритета.

Оператор	Синтаксис
Круглые скобки	()
Инкремент	++
Декремент	—
Возведение в степень	**
Умножение	*
Деление	/
Сложение	+
Вычитание	—

Приоритет имеют не только арифметические операторы, но и операторы присваивания, логические операторы, условные операторы

Лекция 1.20 Понятие функции и методов

Соединение с внешним файлом JavaScript. JavaScript и производительность браузера. Размещение кода JavaScript. Безопасность JavaScript и ее отсутствие.

Функция - это блок программного кода на языке JavaScript, который определяется один раз и может выполняться, или вызываться, многократно. Возможно, вы уже знакомы с понятием «функция» под другим названием, таким как подпрограмма, или процедура. Функции могут иметь параметры: определение функции может включать список идентификаторов, которые называются параметрами и играют роль локальных переменных в теле функции.

При вызове функций им могут передаваться значения, или аргументы, соответствующие их параметрам. Функции часто используют свои аргументы для вычисления возвращаемого значения, которое является значением выражения вызова функции. В дополнение к аргументам при вызове любой функции ей передается еще одно значение, определяющее контекст вызова - значение в ключевом слове this.

Функции в языке JavaScript являются объектами и могут использоваться разными способами. Например, функции могут присваиваться переменным и передаваться другим функциям. Поскольку функции являются объектами, имеется возможность присваивать значения их свойствам и даже вызывать их методы.

В JavaScript допускается создавать определения функций, вложенные в другие функции, и такие функции будут иметь доступ ко всем переменным, присутствующим в области видимости определения.

Определение функций

Определение функции начинается с ключевого слова function, за которым указываются следующие компоненты:

Идентификатор, определяющий имя функции

Имя является обязательной частью инструкции объявления функции: оно будет использовано для создания новой переменной, которой будет присвоен объект новой функции. В выражениях определения функций имя может отсутствовать: при его наличии имя будет ссылаться на объект функции только в теле самой функции.

Пара круглых скобок вокруг списка из нуля или более идентификаторов, разделенных запятыми

Эти идентификаторы будут определять имена параметров функции и в теле функции могут использоваться как локальные переменные.

Пара фигурных скобок с нулем или более инструкций JavaScript внутри

Эти инструкции составляют тело функции: они выполняются при каждом вызове функции.

В следующем примере показано несколько определений функций в виде инструкций и выражений. Обратите внимание, что определения функций в виде выражений удобно использовать, только если они являются частью более крупных выражений, таких как присваивание или вызов функции, которые выполняют некоторые действия с помощью вновь объявленной функции:

```
// Выводит имена и значения всех свойств объекта obj
function printprops(obj) {
	for(var p in obj)
		console.log(p + ": " + obj[p] + "\n");
}
```

```
// Вычисляет расстояние между точками (x1,y1) и (x2,y2)
function distance(x1, y1, x2, y2) {
	var dx = x2 - x1;
	var dy = y2 - y1;
	return Math.sqrt(dx*dx + dy*dy);
}

// Рекурсивная функция (вызывающая сама себя), вычисляющая факториал
function factorial(x) {
	if (x <= 1) return 1;
	return x * factorial(x-1);
}

// Следующее выражение определяет функцию, вычисляющую
// квадрат аргумента. Обратите внимание,
// что она присваивается переменной
var square = function(x) { return x*x; }

// Выражения определения функций могут иметь имена, что позволяет
// производить рекурсивные вызовы
var f = function fact(x) {
	if (x <= 1)
		return 1;
	else
		return x*fact(x-1);
};

// Выражения определения функций иногда могут тут же вызываться:
var tensquared = (function(x) {return x*x;}(10));
```

Обратите внимание, что в выражениях определения функций имя функции может отсутствовать. Инструкция объявления функции фактически объявляет переменную и присваивает ей объект функции.

Выражение определения функции, напротив, не объявляет переменную. Однако в выражениях определения допускается указывать имя функции, как в функции вычисления факториала выше, которое может потребоваться в теле функции для вызова себя самой. Если выражение определения функции включает имя, данное имя будет ссылаться на объект функции в области видимости этой функции. Фактически имя функции становится локальной переменной, доступной только в теле функции. В большинстве случаев имя функции не требуется указывать в выражениях определения, что делает определения более компактными.

Обратите внимание, что большинство (но не все) функций в примере содержат инструкцию return. Инструкция return завершает выполнение функции и выполняет возврат значения своего выражения (если указано) вызывающей программе. Если выражение в инструкции return отсутствует, она возвращает значение undefined. Если инструкция return отсутствует в функции, интерпретатор просто выполнит все инструкции в теле функции и вернет вызывающей программе значение undefined.

Большинство функций в примере вычисляют некоторое значение, и в них инструкция return используется для возврата этого значения вызывающей программе.

Функция printprops() несколько отличается в этом смысле: ее работа заключается в том, чтобы вывести имена свойств объекта. Ей не нужно возвращать какое-либо значение, поэтому в функции отсутствует инструкция return. Функция printprops() всегда будет возвращать значение undefined. (Функции, не имеющие возвращаемого значения, иногда называются процедурами.)

Вызов функций

Программный код, образующий тело функции, выполняется не в момент определения функции, а в момент ее вызова. Вызов функций выполняется с помощью выражения вызова. Выражение вызова состоит из выражения обращения к функции, которое возвращает объект функции, и следующими за ним круглыми скобками со списком из нуля или более выражений-аргументов, разделенных запятыми, внутри.

Если выражение обращения к функции является выражением обращения к свойству - если функция является свойством объекта или элементом массива (т.е. методом) - тогда выражение вызова является выражением вызова метода. В следующем фрагменте демонстрируется несколько примеров выражений вызова обычных функций:

```
printprops({x:4, age: 24});
var d = distance(1,1,5,6);
var f = factorial(5) / factorial(12);
f = square(5);
```

При вызове функции вычисляются все выражения-аргументы (указанные между скобками), и полученные значения используются в качестве аргументов функции. Эти значения присваиваются параметрам, имена которых перечислены в определении функции. В теле функции выражения обращений к параметрам возвращают значения соответствующих аргументов.

При вызове обычной функции возвращаемое функцией значение становится значением выражения вызова. Если возврат из функции происходит по достижении ее конца интерпретатором, возвращается значение undefined. Если возврат из функции происходит в результате выполнения инструкции return, возвращается значение выражения, следующего за инструкцией return, или undefined, если инструкция return не имеет выражения.

Метод - это не что иное, как функция, которая хранится в виде свойства объекта. Если имеется функция func и объект obj, то можно определить метод объекта obj с именем method, как показано ниже:

```
// Определим простой объект и функцию
var obj = {};
function func(a, b) { return a+b;}

// Добавим в объект obj метод
obj.method = func;

// Теперь можно вызвать этот метод
var result = obj.method(4, 5);
```

Чаще всего при вызове методов используется форма обращения к свойствам с помощью оператора точки, однако точно так же можно использовать форму обращения к свойствам с помощью квадратных скобок. Например, оба следующих выражения являются выражениями вызова методов:

```
result = obj.method(4, 5);
result = obj['method'](4, 5);
```

Аргументы и возвращаемое значение при вызове метода обрабатываются точно так же, как при вызове обычной функции. Однако вызов метода имеет одно важное отличие: контекст вызова. Выражение обращения к свойству состоит из двух частей: объекта (в данном случае obj) и имени свойства (method). В подобных выражениях вызова методов объект obj становится контекстом вызова, и тело функции получает возможность ссылаться на этот объект с помощью ключевого слова this. Например:

```
var obj = {
	x: 0,
	y: 0,

	// Метод
	add: function(a, b) {
		this.x = a;
		this.y = b;
	},

	// Еще один метод
	sum: function() {
		return this.x + this.y
	}
};

// Вызов методов
obj.add(15, 4);
console.log(obj.sum());	// 19
```

Методы и ключевое слово this занимают центральное место в парадигме объектно-ориентированного программирования. Любая функция, используемая как метод, фактически получает неявный аргумент - объект, относительно которого она была вызвана. Как правило, методы выполняют некоторые действия с объектом, и синтаксис вызова метода наглядно отражает тот факт, что функция оперирует объектом.

Обратите внимание: this - это именно ключевое слово, а не имя переменной или свойства. Синтаксис JavaScript не допускает возможность присваивания значений элементу this.

Аргументы и параметры функций

В языке JavaScript, в определениях функций не указываются типы параметров, а при вызове функций не выполняется никаких проверок типов передаваемых значений аргументов. Фактически при вызове функций в языке JavaScript не проверяется даже количество аргументов. В подразделах ниже описывается, что происходит, если число аргументов в вызове функции меньше или больше числа объявленных параметров. В них также демонстрируется, как можно явно проверить типы аргументов функции, если необходимо гарантировать, что функция не будет вызвана с некорректными аргументами.

Необязательные аргументы

Когда число аргументов в вызове функции меньше числа объявленных параметров, недостающие аргументы получают значение undefined. Часто бывает удобным писать функции так, чтобы некоторые аргументы были необязательными и могли опускаться при вызове функции. В этом случае желательно предусмотреть

возможность присваивания достаточно разумных значений по умолчанию параметрам, которые могут быть опущены. Например:

```
// Добавить в массив arr перечислимые имена
// свойств объекта obj и вернуть его. Если аргумент
// arr не не был передан, создать и вернуть новый массив
function getPropertyNames(obj, /* необязательный */ arr) {
	if (arr === undefined)
		arr = [];		// Если массив не определен, создать новый

	for(var property in obj)
		arr.push(property);

	return arr;
}

// Эта функция может вызываться с 1 или 2 аргументами:

var a = getPropertyNames({x:1, y:1});	// Получить свойства объекта в новом массиве
getPropertyNames({z:5},a);		// добавить свойства нового объекта в этот массив

console.log(a);		// ['x', 'y', 'z']
```

Обратите внимание, что при объявлении функций необязательные аргументы должны завершать список аргументов, чтобы их можно было опустить. Программист, который будет писать обращение к вашей функции, не сможет передать второй аргумент и при этом опустить первый: он будет вынужден явно передать в первом аргументе значение undefined. Обратите также внимание на комментарий /* необязательный */ в определении функции, который подчеркивает тот факт, что параметр является необязательным.

Списки аргументов переменной длины

Если число аргументов в вызове функции превышает число имен параметров, функция лишается возможности напрямую обращаться к неименованным значениям. Решение этой проблемы предоставляет объект Arguments. В теле функции идентификатор arguments ссылается на объект Arguments, присутствующий в вызове. Объект Arguments - это объект, подобный массиву, позволяющий извлекать переданные функции значения по их номерам, а не по именам.

Предположим, что была определена функция func, которая требует один аргумент x. Если вызвать эту функцию с двумя аргументами, то первый будет доступен внутри функции по имени параметра x или как arguments[0]. Второй аргумент будет доступен только как arguments[1]. Кроме того, подобно настоящим массивам, arguments имеет свойство length, определяющее количество содержащихся элементов. То есть в теле функции func, вызываемой с двумя аргументами, arguments.length имеет значение 2.

Объект Arguments может использоваться с самыми разными целями. Следующий пример показывает, как с его помощью проверить, была ли функция вызвана с правильным числом аргументов, - ведь JavaScript этого за вас не сделает:

```
function func(x, y, z) {
	// Сначала проверяется, правильное ли количество аргументов передано
```

```
        if (arguments.length != 3) {
            throw new Error("Функция func вызвана с " + arguments.length
                + " аргументами, а требуется 3.");
        }

        // А теперь сам код функции...
    }
```

Обратите внимание, что зачастую нет необходимости проверять количество аргументов, как в данном примере. Поведение по умолчанию интерпретатора JavaScript отлично подходит для большинства случаев: отсутствующие аргументы замещаются значением undefined, а лишние аргументы просто игнорируются.

Объект Arguments иллюстрирует важную возможность JavaScript-функций: они могут быть написаны таким образом, чтобы работать с любым количеством аргументов. Следующая функция принимает любое число аргументов и возвращает значение самого большого из них (аналогично ведет себя встроенная функция Math.max()):

```
    function maxNumber()
    {
        var m = Number.NEGATIVE_INFINITY;
        // Цикл по всем аргументам, поиск и
        // сохранение наибольшего из них
        for(var i = 0; i < arguments.length; i++)
            if (arguments[i] > m)
                m = arguments[i]; // Вернуть наибольшее значение

        return m;
    }

    var largest = maxNumber(1, 10, 100, 2, 3, 1000, 4, 5, 10000, 6);        // 10000
```

Функции, подобные этой и способные принимать произвольное число аргументов, называются функциями с переменным числом аргументов (variadic functions, variable arity functions или varargs functions). Этот термин возник вместе с появлением языка программирования C.

Обратите внимание, что функции с переменным числом аргументов не должны допускать возможность вызова с пустым списком аргументов. Будет вполне разумным использовать объект arguments[] при написании функции, ожидающей получить фиксированное число обязательных именованных аргументов, за которыми может следовать произвольное число необязательных неименованных аргументов.

Не следует забывать, что arguments фактически не является массивом - это объект Arguments. В каждом объекте Arguments имеются пронумерованные элементы массива и свойство length, но с технической точки зрения это не массив. Лучше рассматривать его как объект, имеющий некоторые пронумерованные свойства.

Помимо элементов своего массива объект Arguments определяет свойства callee и caller. При попытке изменить значения этих свойств в строгом режиме ECMAScript 5 гарантированно возбуждается исключение TypeError. Однако в нестрогом режиме стандарт ECMAScript утверждает, что свойство callee ссылается на выполняемую в данный момент функцию. Свойство caller не является стандартным, но оно присутствует во многих реализациях и ссылается на функцию, вызвавшую текущую.

Свойство caller можно использовать для доступа к стеку вызовов, а свойство callee особенно удобно использовать для рекурсивного вызова неименованных функций:

```
var factorial = function(x) {
	if (x <= 1) return 1;
	return x * arguments.callee(x-1);
};
```

Свойства и методы функций

Мы видели, что в JavaScript-программах функции могут использоваться как значения. Оператор typeof возвращает для функций строку «function», однако в действительности функции в языке JavaScript - это особого рода объекты. А раз функции являются объектами, то они имеют свойства и методы, как любые другие объекты. Существует даже конструктор Function(), который создает новые объекты функций. В следующих подразделах описываются свойства и методы функций.

Свойство length

В теле функции свойство arguments.length определяет количество аргументов, переданных функции. Однако свойство length самой функции имеет иной смысл. Это свойство, доступное только для чтения, возвращает количество аргументов, которое функция ожидает получить - число объявленных параметров.

В следующем фрагменте определяется функция с именем check(), получающая массив аргументов arguments от другой функции. Она сравнивает свойство arguments.length (число фактически переданных аргументов) со свойством arguments.callee.length (число ожидаемых аргументов), чтобы определить, передано ли функции столько аргументов, сколько она ожидает. Если значения не совпадают, генерируется исключение. За функцией check() следует тестовая функция func(), демонстрирующая порядок использования функции check():

```
// Эта функция использует arguments.callee, поэтому она
// не будет работать в строгом режиме
function check(args) {
	var actual = args.length;		// Фактическое число аргументов
	var expected = args.callee.length;	// Ожидаемое число аргументов

	if (actual !== expected)		// Если не совпадают, генерируется исключение
		throw new Error("ожидается: " + expected + "; получено " + actual);
}

function func(x, y, z) {
	// Проверить число ожидаемых и фактически переданных аргументов
	check(arguments);

	// Теперь выполнить оставшуюся часть функции
	return x + y + z;
}
```

Свойство prototype

Любая функция имеет свойство prototype, ссылающееся на объект, известный как объект прототипа. Каждая функция имеет свой объект прототипа. Когда функция используется в роли конструктора, вновь созданный объект наследует свойства этого объекта прототипа.

Прототипы и свойство prototype обсуждались в предыдущей статье.

Методы call() и apply()

Методы call() и apply() позволяют выполнять косвенный вызов функции, как если бы она была методом некоторого другого объекта. Первым аргументом обоим методам, call() и apply(), передается объект, относительно которого вызывается функция; этот аргумент определяет контекст вызова и становится значением ключевого слова this в теле функции. Чтобы вызвать функцию func() (без аргументов) как метод объекта obj, можно использовать любым из методов, call() или apply():

```
func.call(obj);
func.apply(obj);
```

Любой из этих способов вызова эквивалентен следующему фрагменту (где предполагается, что объект obj не имеет свойства с именем m):

```
obj.m = func;          // Временно сделать func методом obj
obj.m();               // Вызывать его без аргументов.
delete obj.m;          // Удалить временный метод.
```

В строгом режиме ECMAScript 5 первый аргумент методов call() и apply() становится значением this, даже если это простое значение, null или undefined. В ECMAScript 3 и в нестрогом режиме значения null и undefined замещаются глобальным объектом, а простое значение - соответствующим объектом-оберткой.

Все остальные аргументы метода call(), следующие за первым аргументом, определяющим контекст вызова, передаются вызываемой функции. Метод apply() действует подобно методу call(), за исключением того, что аргументы для функции передаются в виде массива. Если функция способна обрабатывать произвольное число аргументов, метод apply() может использоваться для вызова такой функции в контексте массива произвольной длины.

В следующем примере демонстрируется практическое применение метода call():

```
// Ниже определены две функции, отображающие свойства и
// значения свойств произвольного объекта. Способ
// отображения передаются в виде аргумента func
function print1(func, obj)
{
	for (n in obj)
		func(n +': ' + obj[n]);
}

function print2(func, objDevice, obj)
{
	for (n in obj)
		func.call(objDevice, n +': ' + obj[n]);
}


var obj = {x:5, y:10};

print2(document.write, document, obj);  // Работает корректно
print2(console.log, console, obj);

print1(document.write, obj);            // Возникнет исключение Illegal invocation, т.к.
```

```
print1(console.log, obj);                    // невозможно вызвать эти методы без объекта контекста
```

Метод bind()

Метод bind() впервые появился в ECMAScript 5, но его легко имитировать в ECMAScript 3. Как следует из его имени, основное назначение метода bind() состоит в том, чтобы связать (bind) функцию с объектом. Если вызвать метод bind() функции func и передать ему объект obj, он вернет новую функцию. Вызов новой функции (как обычной функции) выполнит вызов оригинальной функции func как метода объекта obj. Любые аргументы, переданные новой функции, будут переданы оригинальной функции. Например:

```
// Функция, которую требуется привязать
function func(y) {
    return this.x + y;
}

var obj = {x:1};            // Объект, к которому выполняется привязка
var g = func.bind(obj);     // Вызов g(x) вызовет obj.func(x)
```

Такой способ связывания легко реализовать в ECMAScript 3, как показано ниже:

```
// Возвращает функцию, которая вызывает func как метод объекта obj
// и передает ей все свои аргументы
function bind(func, obj) {
    if (func.bind)
        return func.bind(obj); // Использовать метод bind, если имеется
    else return function() {
        // Иначе связать, как показано ниже
        return func.apply(obj, arguments);
    };
}
```

Метод bind() в ECMAScript 5 не просто связывает функцию с объектом. Он также выполняет частичное применение: помимо значения this связаны будут все аргументы, переданные методу bind() после первого его аргумента. Частичное применение - распространенный прием в функциональном программировании и иногда называется каррингом (currying).

Лекция 1.21 Объекты и свойства

Причины внимания к объектам. Создание объектов.

Объект является фундаментальным типом данных в языке JavaScript. Объект - это составное значение: он объединяет в себе набор значений (простых значений или других объектов) и позволяет сохранять и извлекать эти значения по именам.

Объект является неупорядоченной коллекцией свойств, каждое из которых имеет имя и значение. Имена свойств являются строками, поэтому можно сказать, что объекты отображают строки в значения. Такое отображение строк в значения может называться по-разному: возможно, вы уже знакомы с такой фундаментальной структурой данных, как «хеш», «словарь» или «ассоциативный массив». Однако объект представляет собой нечто большее, чем простое отображение строк в значения.

Помимо собственных свойств объекты в языке JavaScript могут также наследовать свойства от других объектов, известных под названием «прототипы». Методы объекта - это типичные представители унаследованных свойств, а «наследование через прототипы» является ключевой особенностью языка JavaScript.

Объекты в языке JavaScript являются динамическими - обычно они позволяют добавлять и удалять свойства - но они могут использоваться также для имитации статических объектов и «структур», которые имеются в языках программирования со статической системой типов. Кроме того, они могут использоваться (если не учитывать, что объекты отображают строки в значения) для представления множеств строк.

Любое значение в языке JavaScript, не являющееся строкой, числом, true, false, null или undefined, является объектом. И даже строки, числа и логические значения, не являющиеся объектами, могут вести себя как неизменяемые объекты (имеют объекты-обертки String, Number и т.п.).

Объекты являются изменяемыми значениями и операции с ними выполняются по ссылке, а не по значению. Если переменная x ссылается на объект, и выполняется инструкция var y = x;, в переменную y будет записана ссылка на тот же самый объект, а не его копия. Любые изменения, выполняемые в объекте с помощью переменной y, будут также отражаться на переменной x.

Свойство имеет имя и значение. Именем свойства может быть любая строка, включая и пустую строку, но объект не может иметь два свойства с одинаковыми именами. Значением свойства может быть любое значение, допустимое в языке JavaScript, или (в ECMAScript 5) функция чтения или записи (или обе).

В дополнение к именам и значениям каждое свойство имеет ряд ассоциированных с ним значений, которые называют атрибутами свойства:

- Атрибут writable определяет доступность значения свойства для записи.
- Атрибут enumerable определяет доступность имени свойства для перечисления в цикле for/in.
- Атрибут configurable определяет возможность настройки, т.е. удаления свойства и изменения его атрибутов.

До появления стандарта ECMAScript 5 все свойства в объектах, создаваемые программой, доступны для записи, перечисления и настройки. В ECMAScript 5 предусматривается возможность настройки атрибутов ваших свойств.

В дополнение к свойствам каждый объект имеет три атрибута объекта:

- Атрибут prototype содержит ссылку на другой объект, от которого наследуются свойства.

- Атрибут class содержит строку с именем класса объекта и определяет тип объекта.
- Флаг extensible (в ECMAScript 5) указывает на возможность добавления новых свойств в объект.

Наконец, ниже приводится описание некоторых терминов, которые помогут нам различать три обширные категории объектов в языке JavaScript и два типа свойств:

Объект базового языка

Это объект или класс объектов, определяемый спецификацией ECMAScript. Массивы, функции, даты и регулярные выражения (например) являются объектами базового языка.

Объект среды выполнения

Это объект, определяемый средой выполнения (такой как веб-браузер), куда встроен интерпретатор JavaScript. Объекты HTMLElement, представляющие структуру веб-страницы в клиентском JavaScript, являются объектами среды выполнения. Объекты среды выполнения могут также быть объектами базового языка, например, когда среда выполнения определяет методы, которые являются обычными объектами Function базового языка JavaScript.

Пользовательский объект

Любой объект, созданный в результате выполнения программного кода JavaScript.

Собственное свойство

Это свойство, определяемое непосредственно в данном объекте.

Унаследованное свойство

Это свойство, определяемое прототипом объекта.

Создание объектов

Объекты можно создавать с помощью литералов объектов, ключевого слова new и (в ECMAScript 5) функции Object.create().

Литералы объектов

Самый простой способ создать объект заключается во включении в программу литерала объекта. Литерал объекта - это заключенный в фигурные скобки список свойств (пар имя/значение), разделенных запятыми. Именем свойства может быть идентификатор или строковый литерал (допускается использовать пустую строку). Значением свойства может быть любое выражение, допустимое в JavaScript - значение выражения (это может быть простое значение или объект) станет значением свойства.

Ниже приводится несколько примеров создания объектов:

var empty = {}; // Объект без свойств

var point = { x:0, y:0 }; // Два свойства

var point2 = { x:point.x, y:point.y+1 }; // Более сложные значения

var site = {

"url site": "www.professorweb.ru", // Имена свойств с пробелами

'desc-text': "Платформа .NET Framework", // и дефисами, поэтому исп. кавычки

author: { // Значением этого свойства является

firstname: "Alexandr", // объект. Обратите внимание, что

```
            surname: "Frolov"            // имена этих свойств без кавычек.
        }
};
```

В ECMAScript 5 (и в некоторых реализациях ECMAScript 3) допускается использовать зарезервированные слова в качестве имен свойств без кавычек. Однако в целом имена свойств, совпадающие с зарезервированными словами, в ECMA-Script 3 должны заключаться в кавычки. В ECMAScript 5 последняя запятая, следующая за последним свойством в литерале объекта, игнорируется. В большинстве реализаций ECMAScript 3 завершающие запятые также игнорируются, но IE интерпретирует их наличие как ошибку.

Литерал объекта - это выражение, которое создает и инициализирует новый объект всякий раз, когда производится вычисление этого выражения. Значение каждого свойства вычисляется заново, когда вычисляется значение литерала. Это означает, что с помощью единственного литерала объекта можно создать множество новых объектов, если этот литерал поместить в тело цикла или функции, которая будет вызываться многократно, и что значения свойств этих объектов могут отличаться друг от друга.

Создание объектов с помощью оператора new

Оператор new создает и инициализирует новый объект. За этим оператором должно следовать имя функции. Функция, используемая таким способом, называется конструктором и служит для инициализации вновь созданного объекта. Базовый JavaScript включает множество встроенных конструкторов для создания объектов базового языка. Например:

```
var o = new Object();          // Создать новый пустой объект: то же, что и {}
var a = new Array();           // Создать пустой массив: то же, что и []
var d = new Date();            // Создать объект Date, представляющий текущее время
var r = new RegExp("js");      // Создать объект RegExp для операций сопоставления с шаблоном
```

Помимо этих встроенных конструкторов имеется возможность определять свои собственные функции-конструкторы для инициализации вновь создаваемых объектов. О том, как это делается, рассказывается в следующей статье.

Object.create()

Стандарт ECMAScript 5 определяет метод Object.create(), который создает новый объект и использует свой первый аргумент в качестве прототипа этого объекта. Дополнительно Object.create() может принимать второй необязательный аргумент, описывающий свойства нового объекта.

Object.create() является статической функцией, а не методом, вызываемым относительно некоторого конкретного объекта. Чтобы вызвать эту функцию, достаточно передать ей желаемый объект-прототип:

```
// obj наследует свойства x и y
var obj = Object.create({x:1, y:2});
```

Чтобы создать объект, не имеющий прототипа, можно передать значение null, но в этом случае вновь созданный объект не унаследует ни каких-либо свойств, ни базовых методов, таких как toString() (а это означает, что этот объект нельзя будет использовать в выражениях с оператором +):

```
// obj2 не наследует ни свойств, ни методов
var obj2 = Object.create(null);
```

Если в программе потребуется создать обычный пустой объект (который, например, возвращается литералом {} или выражением new Object()), передайте в первом аргументе Object.prototype:

```
// obj3 подобен объекту, созданному
// с помощью {} или new Object()
var obj3 = Object.create(Object.prototype);
```

Возможность создавать новые объекты с произвольными прототипами (скажем иначе: возможность создавать «наследников» от любых объектов) является мощным инструментом, действие которого можно имитировать в ECMAScript 3 с помощью функции, представленной в примере ниже:

```
// inherit() возвращает вновь созданный объект, наследующий свойства
// объекта-прототипа p. Использует функцию Object.create() из ECMAScript 5,
// если она определена, иначе используется более старый прием.
function inherit(p) {
    if (p == null) throw TypeError(); // p не может быть значением null

    if (Object.create)                      // Если Object.create() определена...
        return Object.create(p);   // использовать ее.

    var t = typeof p;                       // Иначе выяснить тип и проверить его
    if (t !== "object" && t !== "function")
        throw TypeError();

    function f() {};                        // Определить пустой конструктор.
    f.prototype = p;                        // Записать в его свойство prototype
                                                    // ссылку на объект p.

    return new f();                             // Использовать f() для создания
                                                    // "наследника" объекта p.
}
```

Реализация функции inherit() приобретет больше смысла, как только мы познакомимся с конструкторами в следующей статье. А пока просто считайте, что она возвращает новый объект, наследующий свойства объекта в аргументе. Обратите внимание, что функция inherit() не является полноценной заменой для Object.create(): она не позволяет создавать объекты без прототипа и не принимает второй необязательный аргумент, как Object.create().

Получение и изменение свойств

Получить значение свойства можно с помощью операторов точки (.) и квадратных скобок ([]). Слева от оператора должно находиться выражение, возвращающее объект. При использовании оператора точки справа должен находиться простой идентификатор, соответствующий имени свойства. При использовании квадратных скобок в квадратных скобках должно указываться выражение, возвращающее строку, содержащую имя требуемого свойства:

```
// Простой объект
var user = { login:'kot86', name:'Alexandr', age:26 };

var login = user.login;          // Получить свойство "login" объекта user
var name = user.name;            // Получить свойство "name" объекта user
```

var age = user['age']; // Получить свойство "age" объекта user

Чтобы создать новое свойство или изменить значение существующего свойства, также используются операторы точки и квадратные скобки, как в операциях чтения значений свойств, но само выражение помещается уже слева от оператора присваивания:

user.age = 28; // Изменить значение свойства 'age'

user['login'] = 'kot84'; // Изменить значение свойства 'login'

user['surname'] = 'Frolov'; // Создать новое свойство 'surname'

В ECMAScript 3 идентификатор, следующий за точкой, не может быть зарезервированным словом: нельзя записать обращение к свойству o.for или o.class, потому что for является ключевым словом, а class - словом, зарезервированным для использования в будущем.

Если объект имеет свойства, имена которых совпадают с зарезервированными словами, для доступа к ним необходимо использовать форму записи с квадратными скобками: o["for"] и o["class"]. Стандарт ECMAScript 5 ослабляет это требование (как это уже сделано в некоторых реализациях ECMAScript 3) и допускает возможность использования зарезервированных слов после оператора точки.

Прототипы

Каждый объект в языке JavaScript имеет второй объект (или null, но значительно реже), ассоциированный с ним. Этот второй объект называется прототипом, и первый объект наследует от прототипа его свойства.

Все объекты, созданные с помощью литералов объектов, имеют один и тот же объект-прототип, на который в программе JavaScript можно сослаться так: Object.prototype.

Объекты, созданные с помощью ключевого слова new и вызова конструктора, в качестве прототипа получают значение свойства prototype функции-конструктора. Поэтому объект, созданный выражением new Object(), наследует свойства объекта Object.prototype, как если бы он был создан с помощью литерала в фигурных скобках {}. Аналогично прототипом объекта, созданного выражением new Array(), является Array.prototype, а прототипом объекта, созданного выражением new Date(), является Date.prototype.

Object.prototype - один из немногих объектов, которые не имеют прототипа: у него нет унаследованных свойств. Другие объекты-прототипы являются самыми обычными объектами, имеющими собственные прототипы.

Все встроенные конструкторы (и большинство пользовательских конструкторов) наследуют прототип Object.prototype. Например, Date.prototype наследует свойства от Object.prototype, поэтому объект Date, созданный выражением new Date(), наследует свойства от обоих прототипов, Date.prototype и Object.prototype. Такая связанная последовательность объектов-прототипов называется цепочкой прототипов.

Наследование

Объекты в языке JavaScript обладают множеством «собственных свойств» и могут также наследовать множество свойств от объекта-прототипа. Чтобы разобраться в этом, необходимо внимательно изучить механизм доступа к свойствам. В примерах этого раздела для создания объектов с определенными прототипами используется функция inherit(), показанная выше.

Предположим, что программа обращается к свойству x объекта obj. Если объект obj не имеет собственного свойства с таким именем, выполняется попытка отыскать свойство x в прототипе объекта obj. Если объект-прототип не имеет собственного

свойства с этим именем, но имеет свой прототип, выполняется попытка отыскать свойство в прототипе прототипа. Так продолжается до тех пор, пока не будет найдено свойство x или пока не будет достигнут объект, не имеющий прототипа. Как видите, атрибут prototype объекта создает цепочку, или связанный список объектов, от которых наследуются свойства.

```
var  obj  = {};      // obj наследует методы объекта Object.prototype
obj.x  = 1;          // и обладает собственным свойством x.

var  p  = inherit(obj);      // p наследует свойства объектов obj и Object.prototype
p.y  =  2;           // и обладает собственным свойством y.

var  q  = inherit(p);      // q наследует свойства объектов p, obj и Object.prototype
q.z  =  3;           // и обладает собственным свойством z.

var s = q.toString();       // toString наследуется от Object.prototype
var d = q.x + q.y       // Результат 3: x и y наследуются от obj и p
```

Теперь предположим, что программа присваивает некоторое значение свойству x объекта obj. Если объект obj уже имеет собственное свойство (не унаследованное) с именем x, то операция присваивания просто изменит значение существующего свойства. В противном случае в объекте obj будет создано новое свойство с именем x. Если прежде объект obj наследовал свойство x, унаследованное свойство теперь окажется скрыто вновь созданным собственным свойством с тем же именем.

Операция присваивания значения свойству проверит наличие этого свойства в цепочке прототипов, чтобы убедиться в допустимости присваивания. Например, если объект obj наследует свойство x, доступное только для чтения, то присваивание выполняться не будет. Однако если присваивание допустимо, всегда создается или изменяется свойство в оригинальном объекте и никогда в цепочке прототипов. Тот факт, что механизм наследования действует при чтении свойств, но не действует при записи новых значений, является ключевой особенностью языка JavaScript, потому что она позволяет выборочно переопределять унаследованные свойства:

```
var unitcircle = { r:1 };            // Объект, от которого наследуется свойство
var c = inherit(unitcircle);         // c наследует свойство r

c.x = 1; c.y = 1;                          // c определяет два собственных свойства
c.r = 2;                                   // c переопределяет унаследованное свойство

console.log(unitcircle.r);           // => 1: объект-прототип не изменился
```

Существует одно исключение из этого правила, когда операция присваивания значения свойству терпит неудачу или приводит к созданию/изменению свойства оригинального объекта. Если объект obj наследует свойство x и доступ к этому свойству осуществляется посредством методов доступа, то вместо создания нового свойства x в объекте obj производится вызов метода записи нового значения. Однако обратите внимание, что метод записи вызывается относительно объекта obj, а не относительно прототипа, в котором определено это свойство, поэтому, если метод записи определяет какие-либо свойства, они будут созданы в объекте obj, а цепочка прототипов опять останется неизменной.

Ошибки доступа к свойствам

Выражения обращения к свойствам не всегда возвращают или изменяют значение свойства. В этом разделе описываются ситуации, когда операции чтения или записи свойства терпят неудачу.

Попытка обращения к несуществующему свойству не считается ошибкой. Если свойство x не будет найдено среди собственных или унаследованных свойств объекта obj, выражение обращения к свойству obj.x вернет значение undefined.

Однако попытка обратиться к свойству несуществующего объекта считается ошибкой. Значения null и undefined не имеют свойств, и попытки обратиться к свойствам этих значений считаются ошибкой:

```
// Простой объект
var user = { login:'kot86', name:'Alexandr', age:26 };

var a = user.password;                    // undefined: свойство отсутствует

// Возбудит исключение TypeError.
// Значение undefined не имеет свойства length
var len = user.password.length;
```

Если нет уверенности, что user и user.password являются объектами (или ведут себя подобно объектам), нельзя использовать выражение user.password.length, так как оно может возбудить исключение. Ниже демонстрируются два способа защиты против исключений подобного рода:

```
// Более наглядный и прямолинейный способ
var len = undefined;
if (user) {
        if (user.password)
                len = user.password.length;
}

// Более краткая и характерная для JavaScript альтернатива
// получения длины значения свойства password
var len = user && user.password && user.password.length;
```

Попытки установить значение свойства для других значений не всегда оканчиваются успехом: некоторые свойства доступны только для чтения и не позволяют изменять их значения. Кроме того, некоторые объекты не позволяют добавлять в них новые свойства. Однако самое интересное, что подобные неудачи, как правило, не приводят к возбуждению исключения:

```
// Свойства prototype встроенных конструкторов доступны только для чтения
Object.prototype = 0;      // Присваивание не возбудит исключения;
                           // значение Object.prototype не изменится
```

Этот исторически сложившийся недостаток JavaScript исправлен в строгом режиме, определяемом стандартом ECMAScript 5. Все неудачные попытки изменить значение свойства в строгом режиме приводят к исключению TypeError.

Правила, позволяющие определить, когда попытка выполнить операцию присваивания завершится успехом, а когда неудачей, просты и понятны, но их сложно выразить в достаточно краткой форме. Попытка присвоить значение свойству p объекта obj потерпит неудачу в следующих случаях:

- Объект obj имеет собственное свойство p, доступное только для чтения: нельзя изменить значение свойства, доступного только для чтения. (Обратите, однако,

внимание на метод defineProperty(), который представляет собой исключение, позволяющее изменять значения настраиваемых свойств, доступных только для чтения.)

- Объект obj имеет унаследованное свойство p, доступное только для чтения: унаследованные свойства, доступные только для чтения, невозможно переопределить собственными свойствами с теми же именами.
- Объект obj не имеет собственного свойства p; объект obj не наследует свойство p с методами доступа и атрибут extensible объекта obj имеет значение false. Если свойство p отсутствует в объекте obj и для него не определен метод записи, то операция присваивания попытается добавить свойство p в объект obj. Но поскольку объект obj не допускает возможность расширения, то попытка добавить в него новое свойство потерпит неудачу.

Удаление свойств

Оператор delete удаляет свойство из объекта. Его единственный операнд должен быть выражением обращения к свойству. Может показаться удивительным, но оператор delete не оказывает влияния на значение свойства - он оперирует самим свойством:

```
// Простой объект
var user = { login:'kot86', name:'Alexandr', age:26 };

delete user.login;          // Теперь объект user не имеет свойства login
delete user['name'];        // Теперь объект user не имеет свойства name
```

Оператор delete удаляет только собственные свойства и не удаляет унаследованные. (Чтобы удалить унаследованное свойство, необходимо удалять его в объекте-прототипе, в котором оно определено. Такая операция затронет все объекты, наследующие этот прототип.)

Выражение delete возвращает значение true в случае успешного удаления свойства или когда операция удаления не привела к изменению объекта (например, при попытке удалить несуществующее свойство). Выражение delete также возвращает true, когда этому оператору передается выражение, не являющееся выражением обращения к свойству:

```
obj = {x:1};        // obj имеет собственное свойство x и наследует toString
delete obj.x;       // Удалит x и вернет true
delete obj.x;       // Ничего не сделает (x не существует) и вернет true
delete obj.toString;        // Ничего не сделает (toString не собственное свойство) и вернет true
delete 1;                   // Бессмысленно, но вернет true
```

Оператор delete не удаляет ненастраиваемые свойства, атрибут configurable которых имеет значение false. (Однако он может удалять настраиваемые свойства нерасширяемых объектов.) Ненастраиваемыми являются свойства встроенных объектов, а также свойства глобального объекта, созданные с помощью инструкций объявления переменных и функций. Попытка удалить ненастраиваемое свойство в строгом режиме вызывает исключение TypeError. В нестрогом режиме (и в реализациях ECMAScript 3) в таких случаях оператор delete просто возвращает false:

```
delete Object.prototype;  // Удаление невозможно - ненастраиваемое свойство

var x = 1;                  // Объявление глобальной переменной
delete this.x;              // Это свойство нельзя удалить
```

```
function f() {}          // Объявление глобальной функции
delete this.f;       // Это свойство также нельзя удалить
```

Проверка существования свойств

Объекты в языке JavaScript можно рассматривать как множества свойств, и нередко бывает полезно иметь возможность проверить принадлежность к множеству - проверить наличие в объекте свойства с данным именем. Выполнить такую проверку можно с помощью оператора in, с помощью методов hasOwnProperty() и propertyIsEnumerable() или просто обратившись к свойству.

Оператор in требует, чтобы в левом операнде ему было передано имя свойства (в виде строки) и объект в правом операнде. Он возвращает true, если объект имеет собственное или унаследованное свойство с этим именем:

```
var obj = { x:1 }
"x" in obj;                // true: obj имеет собственное свойство "x"
"y" in obj;                // false: obj не имеет свойства "y"
"toString" in obj;   // true: obj наследует свойство toString
```

Метод hasOwnProperty() объекта проверяет, имеет ли объект собственное свойство с указанным именем. Для наследуемых свойств он возвращает false:

```
var obj = { x:1 }
obj.hasOwnProperty('x');        // true: obj имеет собственное свойство "x"
obj.hasOwnProperty('y');        // false: obj не имеет свойства "y"
obj.hasOwnProperty('toString');        // false: toString - наследуемое свойство
```

Метод propertyIsEnumerable() накладывает дополнительные ограничения по сравнению с hasOwnProperty(). Он возвращает true, только если указанное свойство является собственным свойством, атрибут enumerable которого имеет значение true. Свойства встроенных объектов не являются перечислимыми. Свойства, созданные обычной программой на языке JavaScript, являются перечислимыми, если не был использован один из методов ECMAScript 5, представленных ниже, которые делают свойства неперечислимыми.

Часто вместо оператора in достаточно использовать простое выражение обращения к свойству и использовать оператор !== для проверки на неравенство значению undefined:

```
var obj = { x:1 }
obj.x !== undefined;                // true: obj имеет свойство "x"
obj.y !== undefined;                // false: obj не имеет свойства "y"
obj.toString !== undefined;         // true: obj наследует свойство toString
```

Однако оператор in отличает ситуации, которые неотличимы при использовании представленного выше приема на основе обращения к свойству. Оператор in отличает отсутствие свойства от свойства, имеющего значение undefined.

Перечисление свойств

Вместо проверки наличия отдельных свойств иногда бывает необходимо обойти все имеющиеся свойства или получить список всех свойств объекта. Обычно для этого используется цикл for/in, однако стандарт ECMAScript 5 предоставляет две удобные альтернативы.

Инструкция цикла for/in выполняет тело цикла для каждого перечислимого свойства (собственного или унаследованного) указанного объекта, присваивая имя свойства переменной цикла. Встроенные методы, наследуемые объектами, являются неперечислимыми, а свойства, добавляемые в объекты вашей программой, являются

перечислимыми (если только не использовались функции, описываемые ниже, позволяющие сделать свойства неперечислимыми). Например:

```
// Простой объект с тремя перечислимыми свойствами
var user = { login:'kot86', name:'Alexandr', age:26 };

user.propertyIsEnumerable('toString');          // false, toString - встроенный метод
for (n in user)
    console.log(n);
```

Некоторые библиотеки добавляют новые методы (или другие свойства) в объект Object.prototype, чтобы они могли быть унаследованы и быть доступны всем объектам. Однако до появления стандарта ECMAScript 5 отсутствовала возможность сделать эти дополнительные методы неперечислимыми, поэтому они оказывались доступными для перечисления в циклах for/in. Чтобы решить эту проблему, может потребоваться фильтровать свойства, возвращаемые циклом for/in. Ниже приводятся два примера реализации такой фильтрации:

```
for (n in user)
{
    if (!user.hasOwnProperty(n)) continue;
    console.log(n);
}

for (n in user)
{
    if (typeof user[n] === 'function') continue;
    console.log(n);
}
```

В дополнение к циклу for/in стандарт ECMAScript 5 определяет две функции, перечисляющие имена свойств. Первая из них, Object.keys(), возвращает массив имен собственных перечислимых свойств объекта.

Вторая функция ECMAScript 5, выполняющая перечисление свойств - Object.getOwnPropertyNames(). Она действует подобно функции Object.keys(), но возвращает имена всех собственных свойств указанного объекта, а не только перечислимые. В реализациях ECMAScript 3 отсутствует возможность реализовать подобные функции, потому что ECMAScript 3 не предусматривает возможность получения неперечислимых свойств объекта.

Методы чтения и записи свойств

Выше уже говорилось, что свойство объекта имеет имя, значение и набор атрибутов. В ECMAScript 5 значение может замещаться одним или двумя методами, известными как методы чтения (getter) и записи (setter). Свойства, для которых определяются методы чтения и записи, иногда называют свойствами с методами доступа, чтобы отличать их от свойств с данными, представляющих простое значение.

Когда программа пытается получить значение свойства с методами доступа, интерпретатор вызывает метод чтения (без аргументов). Возвращаемое этим методом значение становится значением выражения обращения к свойству. Когда программа пытается записать значение в свойство, интерпретатор вызывает метод записи, передавая ему значение, находящее справа от оператора присваивания. Этот метод

отвечает за «установку» значения свойства. Значение, возвращаемое методом записи, игнорируется.

В отличие от свойств с данными, свойства с методами доступа не имеют атрибута writable. Если свойство имеет оба метода, чтения и записи, оно доступно для чтения/записи. Если свойство имеет только метод чтения, оно доступно только для чтения. А если свойство имеет только метод записи, оно доступно только для записи (такое невозможно для свойств с данными) и попытки прочитать значение такого свойства всегда будут возвращать undefined.

Самый простой способ определить свойство с методами доступа заключается в использовании расширенного синтаксиса определения литералов объектов:

```
var obj = {
    // Обычное свойство с данными
    data_prop: value,

    // Свойство с методами доступа определяется как пара функций
    get accessor_prop() { /* тело функции */ },
    set accessor_prop(value) { /* тело функции */ }
};
```

Свойства с методами доступа определяются как одна или две функции, имена которых совпадают с именем свойства и с заменой ключевого слова function на get и/или set.

Обратите внимание, что не требуется использовать двоеточие для отделения имени свойства от функции, управляющей доступом к свойству, но по-прежнему необходимо использовать запятую после тела функции, чтобы отделить метод от других методов или свойств с данными.

Для примера рассмотрим следующий объект, представляющий декартовы координаты точки на плоскости. Для представления координат X и Y в нем имеются обычные свойства с данными, а также свойства с методами доступа, позволяющие получить эквивалентные полярные координаты точки:

```
var p = {
    // x и y обычные свойства с данными, доступные для чтения/записи
    x: 1.0, y: 1.0,

    // r - доступное для чтения/записи свойство с двумя методами доступа.
    // Не забывайте добавлять запятые после методов доступа
    get r() {
        return Math.sqrt(this.x*this.x + this.y*this.y);
    },
    set r(newvalue) {
        var oldvalue = Math.sqrt(this.x*this.x + this.y*this.y);
        var ratio = newvalue/oldvalue;
        this.x *= ratio;
        this.y *= ratio;
    },

    // theta - доступное только для чтения свойство с единственным методом
чтения
    get theta() { return Math.atan2(this.y, this.x); }
```

```
};
```

Обратите внимание на использование ключевого слова this в методах чтения и записи выше. Интерпретатор будет вызывать эти функции, как методы объекта, в котором они определены, т.е. в теле функции this будет ссылаться на объект точки. Благодаря этому метод чтения свойства r может ссылаться на свойства x и y, как this.x и this.y.

Свойства с методами доступа наследуются так же, как обычные свойства с данными, поэтому объект p, определенный выше, можно использовать как прототип для других объектов точек. В новых объектах можно определять собственные свойства x и y, и они будут наследовать свойства r и theta.

Атрибуты объекта

Все объекты имеют атрибуты prototype, class и extensible. Все эти атрибуты описываются в подразделах ниже.

Атрибут prototype

Атрибут prototype объекта определяет объект, от которого наследуются свойства. Важно понимать, что когда в программном коде встречается ссылка prototype, она обозначает обычное свойство объекта, а не атрибут prototype.

Атрибут prototype устанавливается в момент создания объекта. Для объектов, созданных с помощью литералов, прототипом является Object.prototype. Прототипом объекта, созданного с помощью оператора new, является значение свойства prototype конструктора. А прототипом объекта, созданного с помощью Object.create(), становится первый аргумент этой функции (который может иметь значение null).

Стандартом ECMAScript 5 предусматривается возможность определить прототип любого объекта, если передать его методу Object.getPrototypeOf(). В ECMAScript 3 отсутствует эквивалентная функция, но зачастую определить прототип объекта obj можно с помощью выражения obj.constructor.prototype.

Объекты, созданные с помощью оператора new, обычно наследуют свойство constructor, ссылающееся на функцию-конструктор, использованную для создания объекта. И как уже говорилось выше, функции-конструкторы имеют свойство prototype, которое определяет прототип объектов, созданных с помощью этого конструктора.

Обратите внимание, что объекты, созданные с помощью литералов объектов или Object.create(), получают свойство constructor, ссылающееся на конструктор Object(). Таким образом, constructor.prototype ссылается на истинный прототип для литералов объектов, но обычно это не так для объектов, созданных вызовом Object.create().

Чтобы определить, является ли один объект прототипом (или звеном в цепочке прототипов) другого объекта, следует использовать метод isPrototypeOf(). Чтобы узнать, является ли p прототипом obj, нужно записать выражение p.isPrototypeOf(obj). Например:

```
var p = {x:1};                  // Определить объект-прототип.
var obj = Object.create(p);                     // Создать объект с этим прототипом.

p.isPrototypeOf(obj);                           // => true: obj наследует p
Object.prototype.isPrototypeOf(p);    // => true: p наследует Object.prototype
```

Атрибут class

Атрибут class объекта - это строка, содержащая информацию о типе объекта. Ни в ECMAScript 3, ни в ECMAScript 5 не предусматривается возможность изменения этого атрибута и предоставляются лишь косвенные способы определения его значения.

По умолчанию метод toString() (наследуемый от Object.prototype) возвращает строку вида:

[object class]

Поэтому, чтобы определить класс объекта, можно попробовать вызвать метод toString() этого объекта и извлечь из результата подстроку с восьмого по предпоследний символ. Вся хитрость состоит в том, что многие методы наследуют другие, более полезные реализации метода toString(), и чтобы вызвать нужную версию toString(), необходимо выполнить косвенный вызов с помощью метода Function.call().

В примере ниже определяется функция, возвращающая класс любого объекта, переданного ей:

```
// Название класса объекта
function classof(obj) {
    if (obj === null) return "Null";
    if (obj === undefined) return "Undefined";

    return Object.prototype.toString.call(obj).slice(8,-1);
}
```

Этой функции classof() можно передать любое значение, допустимое в языке JavaScript. Числа, строки и логические значения действуют подобно объектам, когда относительно них вызывается метод toString(), а значения null и undefined обрабатываются особо.

Атрибут extensible

Атрибут extensible объекта определяет, допускается ли добавлять в объект новые свойства. В ECMAScript 3 все встроенные и определяемые пользователем объекты неявно допускали возможность расширения, а расширяемость объектов среды выполнения определялась каждой конкретной реализацией. В ECMAScript 5 все встроенные и определяемые пользователем объекты являются расширяемыми, если они не были преобразованы в нерасширяемые объекты, а расширяемость объектов среды выполнения по-прежнему определяется каждой конкретной реализацией.

Стандарт ECMAScript 5 определяет функции для получения и изменения признака расширяемости объекта. Чтобы определить, допускается ли расширять объект, его следует передать методу Object.isExtensible(). Чтобы сделать объект нерасширяемым, его нужно передать методу Object.preventExtensions(). Обратите внимание, что после того как объект будет сделан нерасширяемым, его нельзя снова сделать расширяемым. Отметьте также, что вызов preventExtensions() оказывает влияние только на расширяемость самого объекта. Если новые свойства добавить в прототип нерасширяемого объекта, нерасширяемый объект унаследует эти новые свойства.

Назначение атрибута extensible заключается в том, чтобы дать возможность «фиксировать» объекты в определенном состоянии, запретив внесение изменений. Атрибут объектов extensible часто используется совместно с атрибутами свойств configurable и writable, поэтому в ECMAScript 5 определяются функции, упрощающие одновременную установку этих атрибутов.

Метод Object.seal() действует подобно методу Object.preventExtensions(), но он не только делает объект нерасширяемым, но и делает все свойства этого объекта недоступными для настройки. То есть в объект нельзя будет добавить новые свойства, а существующие свойства нельзя будет удалить или настроить. Однако существующие свойства, доступные для записи, по-прежнему могут быть изменены.

После вызова Object.seal() объект нельзя будет вернуть в прежнее состояние. Чтобы определить, вызывался ли метод Object.seal() для объекта, можно вызвать метод Object.isSealed().

Метод Object.freeze() обеспечивает еще более жесткую фиксацию объектов. Помимо того, что он делает объект нерасширяемым, а его свойства недоступными для настройки, он также делает все собственные свойства с данными доступными только для чтения. (Это не относится к свойствам объекта с методами доступа, обладающими методами записи; эти методы по-прежнему будут вызываться инструкциями присваивания.) Чтобы определить, вызывался ли метод Object.freeze() объекта, можно вызвать метод Object.isFrozen().

Важно понимать, что Object.seal() и Object.freeze() воздействуют только на объект, который им передается: они не затрагивают прототип этого объекта. Если в программе потребуется полностью зафиксировать объект, вам, вероятно, потребуется зафиксировать также объекты в цепочке прототипов.

Сериализация объектов

Сериализация объектов - это процесс преобразования объектов в строковую форму представления, которая позднее может использоваться для их восстановления. Для сериализации и восстановления объектов JavaScript стандартом ECMAScript 5 предоставляются встроенные функции JSON.stringify() и JSON.parse(). Эти функции используют формат обмена данными JSON. Название JSON происходит от «JavaScript Object Notation» (форма записи объектов JavaScript), а синтаксис этой формы записи напоминает синтаксис литералов объектов и массивов в языке JavaScript:

```
var obj = {x:1, y:{z:[false,null,""]}};        // Определить испытательный объект
var s = JSON.stringify(obj);                   // s == '{"x":1,"y":{"z":[false,null,""]}}'
var p = JSON.parse(s);                         // p - копия объекта obj
```

Синтаксис формата JSON является лишь подмножеством синтаксиса языка JavaScript и не может использоваться для представления всех возможных значений, допустимых в JavaScript. Поддерживаются и могут быть сериализованы и восстановлены: объекты, массивы, строки, конечные числовые значения, true, false и null. Значения NaN, Infinity и -Infinity сериализуются в значение null. Объекты Date сериализуются в строки с датами в формате ISO, но JSON.parse() оставляет их в строковом представлении и не восстанавливает первоначальные объекты Date.

Объекты Function, RegExp и Error и значение undefined не могут быть сериализованы или восстановлены. Функция JSON.stringify() сериализует только перечислимые собственные свойства объекта. Если значение свойства не может быть сериализовано, это свойство просто исключается из строкового представления. Обе функции, JSON.stringify() и JSON.parse(), принимают необязательный второй аргумент, который можно использовать для настройки процесса сериализации и/или восстановления, например, посредством определения списка свойств, подлежащих сериализации, или функции преобразования значений во время сериализации

Лекция 1.22 Объектная модель документа в JavaScript

Что такое объектная модель браузера?

Веб-страницы бывают статическими и динамическими, последние отличаются тем, что в них используются сценарии (программы) на языке JavaScript.

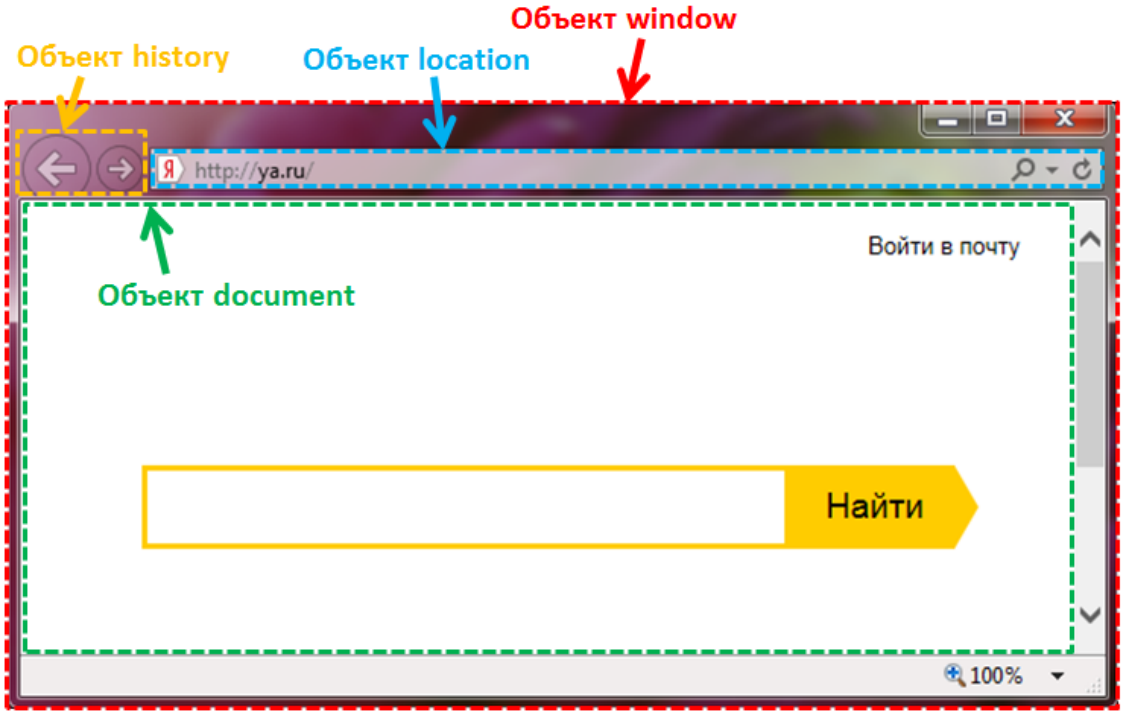
В сценариях JavaScript браузер веб-разработчику предоставляет множество "готовых" объектов, с помощью которых он может взаимодействовать с элементами веб-страницы и самим браузером. В совокупности все эти объекты составляют **объектную модель браузера (BOM – Browser Object Model)**.

На самом верху этой модели находится глобальный объект window. Он представляет собой одно из окон или вкладку браузера с его панелями инструментов, меню, строкой состояния, HTML страницей и другими объектами. Доступ к этим различным объектам окна браузера осуществляется с помощью следующих основных объектов: navigator, history, location, screen, document и т.д. Так как данные объекты являются дочерними по отношению к объекту window, то обращение к ним происходит как к свойствам объекта window.

Например, для того чтобы обратиться к объекту screen, необходимо использовать следующую конструкцию: window.screen. Но если мы работаем с текущим окном, то "window." можно опустить. Например, вместо window.screen можно использовать просто screen.

Из всех этих объектов, наибольший интерес и значимость для разработчика представляет **объект document, который является корнем объектной модели документа (DOM – Document Object Modal)**. Данная модель в отличие от объектной модели браузера стандартизована в спецификации и поддерживается всеми браузерами.

Объект document представляет собой HTML документ, загруженный в окно (вкладку) браузера. С помощью свойств и методов данного объекта Вы можете получить доступ к содержимому HTML-документа, а также изменить его содержимое, структуру и оформление.



Примечание: Объектная модель браузера не стандартизована в спецификации, и поэтому её реализация может отличаться в разных браузерах.

Основная задача при создании динамических веб-страниц в основном сводится к отбору нужных объектов (элементов) и выполнению над ними различных действий. Результаты этих действий сразу отображаются на экране пользователя, а точнее в тех местах, за которые эти объекты отвечают.

Основные объекты BOM

Основные объекты **Browser Object Model**: window, navigator, history, location, screen, document.

Объект window

window – самый главный объект в браузере, который отвечает за одно из окон (вкладок) браузера. Он является корнем иерархии всех объектов доступных веб-разработчику в сценариях JavaScript. Объект window кроме глобальных объектов (document, screen, location, navigator и др.) имеет собственные свойства и методы, которые предназначены для:

открытия нового окна (вкладки);

закрытия окна (вкладки) с помощью метода close();

распечатывания содержимого окна (вкладки);

передачи фокуса окну или для его перемещения на задний план (за всеми окнами);

управления положением и размерами окна, а также для осуществления прокручивания его содержимого;

изменения содержимого статусной строки браузера;

взаимодействия с пользователем посредством следующих окон: alert (для вывода сообщений), confirm (для вывода окна, в котором пользователю необходимо подтвердить или отменить действия), prompt (для получения данных от пользователя);

выполнения определённых действий через определённые промежутки времени и др.

Если в браузере открыть несколько вкладок (окон), то браузером будет создано столько объектов window, сколько открыто этих вкладок (окон). Т.е. каждый раз открывая вкладку (окно), браузер создаёт новый объект window связанный с этой вкладкой (окном).

Рассмотрим следующие примеры:

Вызвать метод write объекта document, который расположен в текущей вкладке (окне) браузера:

window.document.write("Строчка текста");

<!-- Так как данный метод мы вызываем для текущего окна, то "window." можно опустить: -->

document.write("Строчка текста");

Вызвать метод alert для текущий вкладки (окна) браузера:

window.alert("Сообщение");

<!-- Так как данный метод мы вызываем для текущего окна, то "window." можно опустить: -->

alert("Строчка текста");

Объект navigator

navigator – информационный объект с помощью которого Вы можете получить различные данные, содержащиеся в браузере:

информацию о самом браузере в виде строки (User Agent);

внутреннее "кодовое" и официальное имя браузера;

версию и язык браузера;

информацию о сетевом соединении и местоположении устройства пользователя;

информацию об операционной системе и многое другое.

Объект history

history – объект, который позволяет получить историю переходов пользователя по ссылкам в пределах одного окна (вкладки) браузера. Данный объект отвечает за кнопки forward (вперёд) и back (назад). С помощью методов объекта history можно имитировать нажатие на эти кнопки, а также переходить на определённое количество ссылок в истории вперёд или назад. Кроме этого, с появлением HTML5 History API веб-разработчику стали доступны методы для добавления и изменения записей в истории, а также событие, с помощью которого Вы можете обрабатывать нажатие кнопок forward (вперёд) и back (назад).

Объект location

location – объект, который отвечает за адресную строку браузера. Данный объект содержит свойства и методы, которые позволяют: получить текущий адрес страницы браузера, перейти по указанному URL, перезагрузить страницу и т.п.

Объект screen

screen – объект, который предоставляет информацию об экране пользователя: разрешение экрана, максимальную ширину и высоту, которую может иметь окно браузера, глубина цвета и т.д.

Объект document

document – HTML документ, загруженный в окно (вкладку) браузера. Он является корневым узлом HTML документа и "владельцем" всех других узлов: элементов, текстовых узлов, атрибутов и комментариев. Объект document содержит свойства и методы для доступа ко всем узловым объектам. document как и другие объекты, является частью объекта window и, следовательно, он может быть доступен как window.document.

Лекция 1.23 Объектная модель документа в JavaScript. Объекты как ассоциативные массивы

. Объектный литерал..

Как мы знаем из главы Типы данных, в JavaScript существует 8 типов данных. Семь из них называются «примитивными», так как содержат только одно значение (будь то строка, число или что-то другое).

Объекты же используются для хранения коллекций различных значений и более сложных сущностей. В JavaScript объекты используются очень часто, это одна из основ языка. Поэтому мы должны понять их, прежде чем углубляться куда-либо ещё.

Объект может быть создан с помощью фигурных скобок {…} с необязательным списком свойств. Свойство – это пара «ключ: значение», где ключ – это строка (также называемая «именем свойства»), а значение может быть чем угодно.

Мы можем представить объект в виде ящика с подписанными папками. Каждый элемент данных хранится в своей папке, на которой написан ключ. По ключу папку легко найти, удалить или добавить в неё что-либо.

Пустой объект («пустой ящик») можно создать, используя один из двух вариантов синтаксиса:

```
let user = new Object(); // синтаксис "конструктор объекта"
let user = {};  // синтаксис "литерал объекта"
```

Обычно используют вариант с фигурными скобками {...}. Такое объявление называют литералом объекта или литеральной нотацией.

Литералы и свойства

При использовании литерального синтаксиса {...} мы сразу можем поместить в объект несколько свойств в виде пар «ключ: значение»:

```
let user = {     // объект
  name: "John",  // под ключом "name" хранится значение "John"
  age: 30        // под ключом "age" хранится значение 30
};
```

Свойства объекта также иногда называют полями объекта.

У каждого свойства есть ключ (также называемый «имя» или «идентификатор»). После имени свойства следует двоеточие ":", и затем указывается значение свойства. Если в объекте несколько свойств, то они перечисляются через запятую.

В объекте user сейчас находятся два свойства:

1. Первое свойство с именем "name" и значением "John".
2. Второе свойство с именем "age" и значением 30.

Можно сказать, что наш объект user – это ящик с двумя папками, подписанными «name» и «age».

Мы можем в любой момент добавить в него новые папки, удалить папки или прочитать содержимое любой папки.

Для обращения к свойствам используется запись «через точку»:

```
// получаем свойства объекта:
alert( user.name ); // John
alert( user.age ); // 30
```

Значение может быть любого типа. Давайте добавим свойство с логическим значением:

```
user.isAdmin = true;
```

Для удаления свойства мы можем использовать оператор delete:

```
delete user.age;
```

Имя свойства может состоять из нескольких слов, но тогда оно должно быть заключено в кавычки:

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // имя свойства из нескольких слов должно быть в кавычках
};
```

Последнее свойство объекта может заканчиваться запятой:

```
let user = {
  name: "John",
  age: 30,
}
```

Это называется «висячая запятая». Такой подход упрощает добавление, удаление и перемещение свойств, так как все строки объекта становятся одинаковыми.

Квадратные скобки

Для свойств, имена которых состоят из нескольких слов, доступ к значению «через точку» не работает:

```
// это вызовет синтаксическую ошибку
user.likes birds = true
```

JavaScript видит, что мы обращаемся к свойству user.likes, а затем идёт непонятное слово birds. В итоге синтаксическая ошибка.

Точка требует, чтобы ключ был именован по правилам именования переменных. То есть не имел пробелов, не начинался с цифры и не содержал специальные символы, кроме $ и _.

Для таких случаев существует альтернативный способ доступа к свойствам через квадратные скобки. Такой способ сработает с любым именем свойства:

```
let user = {};

// присваивание значения свойству
user["likes birds"] = true;

// получение значения свойства
alert(user["likes birds"]); // true

// удаление свойства
delete user["likes birds"];
```

Сейчас всё в порядке.

Обратите внимание, что строка в квадратных скобках закавычена (подойдёт любой тип кавычек).

Квадратные скобки также позволяют обратиться к свойству, имя которого может быть результатом выражения. Например, имя свойства может храниться в переменной:

```
let key = "likes birds";

// то же самое, что и user["likes birds"] = true;
user[key] = true;
```

Здесь переменная key может быть вычислена во время выполнения кода или зависеть от пользовательского ввода. После этого мы используем её для доступа к свойству. Это даёт нам большую гибкость.

Пример:

```
let user = {
  name: "John",
  age: 30
};

let key = prompt("Что вы хотите узнать о пользователе?", "name");

// доступ к свойству через переменную
alert( user[key] ); // John (если ввели "name")
```

Запись «через точку» такого не позволяет:

```
let user = {
  name: "John",
  age: 30
};

let key = "name";
alert( user.key ); // undefined
```

Вычисляемые свойства

Мы можем использовать квадратные скобки в литеральной нотации для создания вычисляемого свойства.

Пример:

```
let fruit = prompt("Какой фрукт купить?", "apple");

let bag = {
  [fruit]: 5, // имя свойства будет взято из переменной fruit
};

alert( bag.apple ); // 5, если fruit="apple"
```

Смысл вычисляемого свойства прост: запись [fruit] означает, что имя свойства необходимо взять из переменной fruit.

И если посетитель введёт слово "apple", то в объекте bag теперь будет лежать свойство {apple: 5}.

По сути, пример выше работает так же, как и следующий пример:

```
let fruit = prompt("Какой фрукт купить?", "apple");
let bag = {};

// имя свойства будет взято из переменной fruit
bag[fruit] = 5;
```

…Но первый пример выглядит лаконичнее.

Мы можем использовать и более сложные выражения в квадратных скобках:

```
let fruit = 'apple';
let bag = {
  [fruit + 'Computers']: 5 // bag.appleComputers = 5
```

```
};
```

Квадратные скобки дают намного больше возможностей, чем запись через точку. Они позволяют использовать любые имена свойств и переменные, хотя и требуют более громоздких конструкций кода.

Подведём итог: в большинстве случаев, когда имена свойств известны и просты, используется запись через точку. Если же нам нужно что-то более сложное, то мы используем квадратные скобки.

Свойство из переменной

В реальном коде часто нам необходимо использовать существующие переменные как значения для свойств с тем же именем.

Например:

```
function makeUser(name, age) {
  return {
    name: name,
    age: age
    // ...другие свойства
  };
}

let user = makeUser("John", 30);
alert(user.name); // John
```

В примере выше название свойств name и age совпадают с названиями переменных, которые мы подставляем в качестве значений этих свойств. Такой подход настолько распространён, что существуют специальные короткие свойства для упрощения этой записи.

Вместо name:name мы можем написать просто name:

```
function makeUser(name, age) {
  return {
    name, // то же самое, что и name: name
    age   // то же самое, что и age: age
    // ...
  };
}
```

Мы можем использовать как обычные свойства, так и короткие в одном и том же объекте:

```
let user = {
  name,  // тоже самое, что и name:name
  age: 30
};
```

Ограничения на имена свойств

Мы можем использовать только строки и символы в качестве ключей свойств. Все другие типы данных будут автоматически преобразованы к строке.

Например, если использовать как ключ число 0, то оно превратится в строку "0":

```
let obj = {
  0: "Тест" // то же самое что и "0": "Тест"
};
```

// обе функции alert выведут одно и то же свойство (число 0 преобразуется в строку "0")

alert(obj["0"]); // Тест

alert(obj[0]); // Тест (то же свойство)

Зарезервированные слова разрешено использовать как имена свойств.

Как мы уже знаем, имя переменной не может совпадать с зарезервированными словами, такими как «for», «let», «return» и т.д.

Но для свойств объекта такого ограничения нет:

```
let obj = {
  for: 1,
  let: 2,
  return: 3
};

alert( obj.for + obj.let + obj.return );  // 6
```

В принципе, разрешены любые имена свойств, но есть специальное свойство __proto__, которое по историческим причинам имеет особое поведение.

Например, его значение всегда должно быть объектом:

```
let obj = {};
obj.__proto__ = 5;
alert(obj.__proto__); // [object Object], работает не так, как мы ожидали
```

Как мы видим, присвоение примитивного значения 5 игнорируется.

Мы более подробно исследуем происходящее свойство __proto__ позже, в главе Прототипное наследование

.

Сейчас важно знать, что такое поведение __proto__ может стать источником ошибок и даже уязвимостей, если мы намереваемся хранить в объекте произвольные данные и позволяем посетителям указывать ключи.

Посетитель может указать "__proto__" в качестве ключа, и логика присваивания будет нарушена (как показано выше).

Позже мы изучим, как обойти эту проблему:

1. Можно заставить объект обрабатывать __proto__ как обычное свойство, мы это увидим в главе Методы прототипов, объекты без свойства __proto__.
2. Есть другая структура данных Map, которую мы изучим в главе Map и Set, она поддерживает произвольные ключи.

Проверка существования свойства, оператор «in»

Особенность объектов в том, что можно получить доступ к любому свойству. Даже если свойства не существует – ошибки не будет! При обращении к свойству, которого нет, возвращается undefined. Это позволяет просто проверить существование свойства – сравнением его с undefined:

```
let user = {};
alert( user.noSuchProperty === undefined ); // true означает "свойства нет"
```

Также существует специальный оператор "in" для проверки существования свойства в объекте.

Синтаксис оператора:

```
"key" in object
```

Пример:

```
let user = { name: "John", age: 30 };
```

```
alert( "age" in user ); // true, user.age существует
alert( "blabla" in user ); // false, user.blabla не существует
```

Обратите внимание, что слева от оператора in должно быть имя свойства. Обычно это строка в кавычках.

Если мы опускаем кавычки, это значит, что мы указываем переменную, в которой находится имя свойства. Например:

```
let user = { age: 30 };
let key = "age";
alert( key in user ); // true, имя свойства было взято из переменной key
```

Оператор «in» для свойств со значением „undefined“

Обычно строгого сравнения "=== undefined" достаточно для проверки наличия свойства. Но есть особый случай, когда оно не подходит, и нужно использовать "in".

Это когда свойство существует, но содержит значение undefined:

```
let obj = {
  test: undefined
};
alert( obj.test ); // выведет undefined, значит свойство не существует?
alert( "test" in obj ); // true, свойство существует!
```

В примере выше свойство obj.test технически существует в объекте. Оператор in сработал правильно.

Подобные ситуации случаются очень редко, так как undefined обычно явно не присваивается. Для «неизвестных» или «пустых» свойств мы используем значение null. Таким образом, оператор in является экзотическим гостем в коде.

Цикл «for…in»

Для перебора всех свойств объекта используется цикл for..in. Этот цикл отличается от изученного ранее цикла for(;;).

Синтаксис:

```
for (key in object) {
  // тело цикла выполняется для каждого свойства объекта
}
```

К примеру, давайте выведем все свойства объекта user:

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};
for (let key in user) {
  // ключи
  alert( key );  // name, age, isAdmin
  // значения ключей
  alert( user[key] ); // John, 30, true
}
```

Обратите внимание, что все конструкции «for» позволяют нам объявлять переменную внутри цикла, как, например, let key здесь.

Кроме того, мы могли бы использовать другое имя переменной. Например, часто используется вариант "for (let prop in obj)".

Упорядочение свойств объекта

Упорядочены ли свойства объекта? Другими словами, если мы будем в цикле перебирать все свойства объекта, получим ли мы их в том же порядке, в котором мы их добавляли? Можем ли мы на это рассчитывать?

Короткий ответ: свойства упорядочены особым образом: свойства с целочисленными ключами сортируются по возрастанию, остальные располагаются в порядке создания. Разберёмся подробнее.

В качестве примера рассмотрим объект с телефонными кодами:

```
let codes = {
  "49": "Германия",
  "41": "Швейцария",
  "44": "Великобритания",
  // ..,
  "1": "США"
};
for (let code in codes) {
  alert(code); // 1, 41, 44, 49
}
```

Если мы делаем сайт для немецкой аудитории, то, вероятно, мы хотим, чтобы код 49 был первым.

Но если мы запустим код, мы увидим совершенно другую картину:

- США (1) идёт первым
- затем Швейцария (41) и так далее.

Телефонные коды идут в порядке возрастания, потому что они являются целыми числами: 1, 41, 44, 49.

Целочисленные свойства? Это что?

Термин «целочисленное свойство» означает строку, которая может быть преобразована в целое число и обратно без изменений.

То есть, "49" – это целочисленное имя свойства, потому что если его преобразовать в целое число, а затем обратно в строку, то оно не изменится. А вот свойства "+49" или "1.2" таковыми не являются:

```
// Math.trunc - встроенная функция, которая удаляет десятичную часть
alert( String(Math.trunc(Number("49"))) ); // "49", то же самое ⇒ свойство целочисленное
alert( String(Math.trunc(Number("+49"))) ); // "49", не то же самое, что "+49" ⇒ свойство не целочисленное
alert( String(Math.trunc(Number("1.2"))) ); // "1", не то же самое, что "1.2" ⇒ свойство не целочисленное
```

…С другой стороны, если ключи не целочисленные, то они перебираются в порядке создания, например:

```
let user = {
  name: "John",
  surname: "Smith"
};
user.age = 25; // добавим ещё одно свойство
// не целочисленные свойства перечислены в порядке создания
for (let prop in user) {
  alert( prop ); // name, surname, age
}
```

Таким образом, чтобы решить нашу проблему с телефонными кодами, мы можем схитрить, сделав коды не целочисленными свойствами. Добавления знака "+" перед каждым кодом будет достаточно.

Пример:

```
let codes = {
  "+49": "Германия",
  "+41": "Швейцария",
  "+44": "Великобритания",
  // ..,
  "+1": "США"
};
for (let code in codes) {
  alert( +code ); // 49, 41, 44, 1
}
```

Теперь код работает так, как мы задумывали.

Копирование по ссылке

Одним из фундаментальных отличий объектов от примитивных типов данных является то, что они хранятся и копируются «по ссылке».

Примитивные типы: строки, числа, логические значения – присваиваются и копируются «по значению».

Например:

```
let message = "Hello!";
let phrase = message;
```

В результате мы имеем две независимые переменные, каждая из которых хранит строку "Hello!".

Объекты ведут себя иначе.

Переменная хранит не сам объект, а его «адрес в памяти», другими словами «ссылку» на него.

Проиллюстрируем это:

```
let user = {
  name: "John"
};
```

Сам объект хранится где-то в памяти. А в переменной user лежит «ссылка» на эту область памяти.

Когда переменная объекта копируется – копируется ссылка, сам же объект не дублируется.

Если мы представляем объект как ящик, то переменная – это ключ к нему. Копирование переменной дублирует ключ, но не сам ящик.

Например:

```
let user = { name: "John" };
let admin = user; // копируется ссылка
```

Теперь у нас есть две переменные, каждая из которых содержит ссылку на один и тот же объект:

Мы можем использовать любую из переменных для доступа к ящику и изменения его содержимого:

```
let user = { name: 'John' };
let admin = user;
admin.name = 'Pete'; // изменено по ссылке из переменной "admin"
```

alert(user.name); // 'Pete', изменения видны по ссылке из переменной "user"

Приведённый выше пример демонстрирует, что объект только один. Как если бы у нас был один ящик с двумя ключами и мы использовали один из них (admin), чтобы войти в него и что-то изменить, а затем, открыв ящик другим ключом (user), мы бы увидели эти изменения.

Сравнение объектов

Операторы равенства == и строгого равенства === для объектов работают одинаково.

Два объекта равны только в том случае, если это один и тот же объект.

Например, две переменные ссылаются на один и тот же объект, они равны:

```
let a = {};
let b = a; // копирование по ссылке
alert( a == b ); // true, обе переменные ссылаются на один и тот же объект
alert( a === b ); // true
```

В примере ниже два разных объекта не равны, хотя и оба пусты:

```
let a = {};
let b = {}; // два независимых объекта
alert( a == b ); // false
```

Для сравнений типа obj1 > obj2 или для сравнения с примитивом obj == 5 объекты преобразуются в примитивы.

Мы скоро изучим, как работают такие преобразования объектов, но, по правде говоря, сравнения такого рода необходимы очень редко и обычно являются результатом ошибки программиста.

Объекты-константы

Объект, объявленный через const, может быть изменён.

Пример:

```
const user = {
  name: "John"
};
user.age = 25; // (*)
alert(user.age); // 25
```

Может показаться, что строка (*) должна вызвать ошибку, но нет, здесь всё в порядке. Дело в том, что объявление const защищает от изменений только само значение user. А в нашем случае значение user – это ссылка на объект, и это значение мы не меняем. В строке (*) мы действуем внутри объекта, мы не переназначаем user.

Если же мы попытаемся присвоить user другое значение, то const выдаст ошибку:

```
const user = {
  name: "John"
};
// Ошибка (нельзя переопределять константу user)
user = {
  name: "Pete"
};
```

…Но что делать, если мы хотим сделать константами свойства объекта? Как сделать так, чтобы попытка изменить user.age = 25 выдавала ошибку? Это тоже возможно. Мы рассмотрим эту тему в главе Флаги и дескрипторы свойств.

Клонирование и объединение объектов, Object.assign

Как мы узнали ранее, при копировании переменной объекта создаётся ещё одна ссылка на тот же самый объект.

Но что, если нам всё же нужно дублировать объект? Создать независимую копию, клон?

Это выполнимо, но немного сложно, так как в JavaScript нет встроенного метода для этого. На самом деле, такая нужда возникает редко. В большинстве случаев нам достаточно копирования по ссылке.

Но если мы действительно этого хотим, то нам нужно создавать новый объект и повторять структуру дублируемого объекта, перебирая его свойства и копируя их.

Например так:

```
let user = {
  name: "John",
  age: 30
};
let clone = {}; // новый пустой объект
// скопируем все свойства user в него
for (let key in user) {
  clone[key] = user[key];
}
// теперь в переменной clone находится абсолютно независимый клон объекта.
clone.name = "Pete"; // изменим в нём данные
alert( user.name ); // в оригинальном объекте значение свойства `name` осталось прежним – John.
```

Кроме того, для этих целей мы можем использовать метод Object.assign.

Синтаксис:

```
Object.assign(dest, [src1, src2, src3...])
```

- Аргументы dest, и src1, ..., srcN (может быть столько, сколько нужно) являются объектами.
- Метод копирует свойства всех объектов src1, ..., srcN в объект dest. То есть, свойства всех перечисленных объектов, начиная со второго, копируются в первый объект. После копирования метод возвращает объект dest.

Например, объединим несколько объектов в один:

```
let user = { name: "John" };
let permissions1 = { canView: true };
let permissions2 = { canEdit: true };
// копируем все свойства из permissions1 и permissions2 в user
Object.assign(user, permissions1, permissions2);
// now user = { name: "John", canView: true, canEdit: true }
```

Если принимающий объект (user) уже имеет свойство с таким именем, оно будет перезаписано:

```
let user = { name: "John" };
// свойство name перезапишется, свойство isAdmin добавится
Object.assign(user, { name: "Pete", isAdmin: true });
// now user = { name: "Pete", isAdmin: true }
```

Мы также можем использовать Object.assign для простого клонирования:

```
let user = {
  name: "John",
  age: 30
```

```
};
let clone = Object.assign({}, user);
```

Все свойства объекта user будут скопированы в пустой объект, и ссылка на этот объект будет в переменной clone. На самом деле, такое клонирование работает так же, как и через цикл, но короче.

До сих пор мы предполагали, что все свойства пользователя примитивны. Но свойства могут быть ссылками на другие объекты. Что с ними делать?

Например, есть объект:

```
let user = {
  name: "John",
  sizes: {
    height: 182,
    width: 50
  }
};
alert( user.sizes.height ); // 182
```

Теперь при клонировании недостаточно просто скопировать clone.sizes = user.sizes, поскольку user.sizes – это объект, он будет скопирован по ссылке. А значит объекты clone и user в своих свойствах sizes будут ссылаться на один и тот же объект:

```
let user = {
  name: "John",
  sizes: {
    height: 182,
    width: 50
  }
};
let clone = Object.assign({}, user);
alert( user.sizes === clone.sizes ); // true, один и тот же объект
// user и clone обращаются к одному sizes
user.sizes.width++;       // меняем свойство в одном объекте
alert(clone.sizes.width); // 51, видим результат в другом объекте
```

Чтобы исправить это, мы должны в цикле клонирования делать проверку, не является ли значение user[key] объектом, и если это так – копируем и его структуру тоже. Это называется «глубокое клонирование».

Существует стандартный алгоритм глубокого клонирования, Structured cloning algorithm. Он решает описанную выше задачу, а также более сложные задачи. Чтобы не изобретать велосипед, мы можем использовать реализацию этого алгоритма из JavaScript-библиотеки lodash, метод _.cloneDeep(obj).

Итого

Объекты – это ассоциативные массивы с рядом дополнительных возможностей.

Они хранят свойства (пары ключ-значение), где:

- Ключи свойств должны быть строками или символами (обычно строками).
- Значения могут быть любого типа.

Чтобы получить доступ к свойству, мы можем использовать:

- Запись через точку: obj.property.
- Квадратные скобки obj["property"]. Квадратные скобки позволяют взять ключ из переменной, например, obj[varWithKey].

Дополнительные операторы:

- Удаление свойства: delete obj.prop.
- Проверка существования свойства: "key" in obj.
- Перебор свойств объекта: цикл for for (let key in obj).

Объекты присваиваются и копируются по ссылке. Другими словами, переменная хранит не «значение объекта», а «ссылку» (адрес в памяти) на это значение. Поэтому копирование такой переменной или передача её в качестве аргумента функции приводит к копированию этой ссылки, а не самого объекта. Все операции с использованием скопированных ссылок (например, добавление или удаление свойств) выполняются с одним и тем же объектом.

Чтобы сделать «настоящую копию» (клон), мы можем использовать Object.assign или _.cloneDeep(obj).

То, что мы изучали в этой главе, называется «простым объектом» («plain object») или просто Object.

В JavaScript есть много других типов объектов:

- Array для хранения упорядоченных коллекций данных,
- Date для хранения информации о дате и времени,
- Error для хранения информации об ошибке.
- … и так далее.

У них есть свои особенности, которые мы изучим позже. Иногда люди говорят что-то вроде «тип данных Array» или «тип данных Date», но формально они не являются отдельными типами, а относятся к типу данных Object. Они лишь расширяют его различными способами.

Объекты в JavaScript очень мощные. Здесь мы только немного углубились в действительно огромную тему. Мы будем плотно работать с объектами и узнаем о них больше в следующих частях учебника.

Задачи

Привет, object

важность: 5

Напишите код, выполнив задание из каждого пункта отдельной строкой:

1. Создайте пустой объект user.
2. Добавьте свойство name со значением John.
3. Добавьте свойство surname со значением Smith.
4. Измените значение свойства name на Pete.
5. Удалите свойство name из объекта.

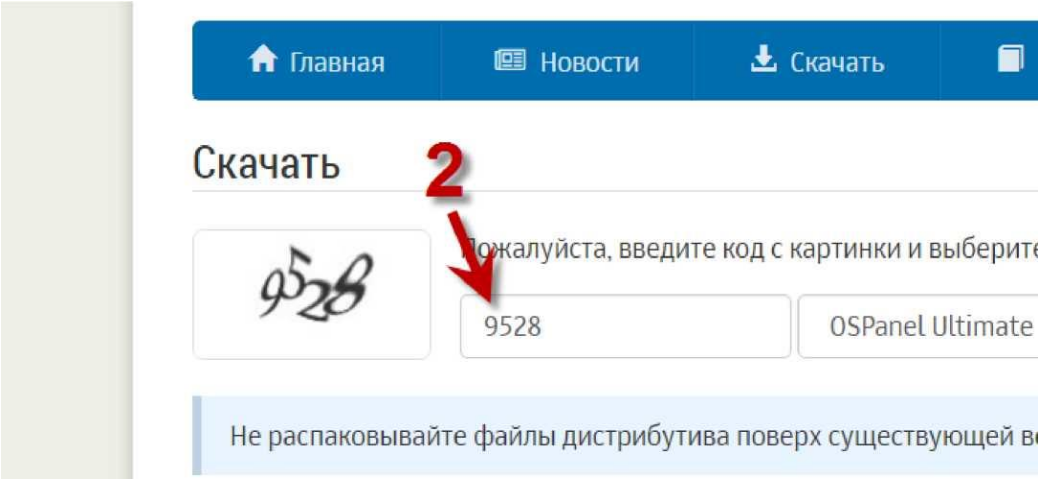
2 ЛАБОРАТОРНО-ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Практическая работа. Установка сервера

Activity 1. Внимательно изучите материал
Подробная инструкция

Установка Open Server

Прежде всего, нам необходимо скачать сам дистрибутив. Для этого перейдите на официальный сайт в раздел «Скачать», выберите нужную версию дистрибутива (1). Стоит отметить, что продукт поставляется в трех версиях: Ultimate, Premium, Basic, чем они отличаются межу собой наглядно видно в таблице сравнения на сайте. Я всегда использую «ULTIMATE» (с максимальными характеристиками). Далее введите код с картинки (2) и нажмите кнопку «Скачать» (3).

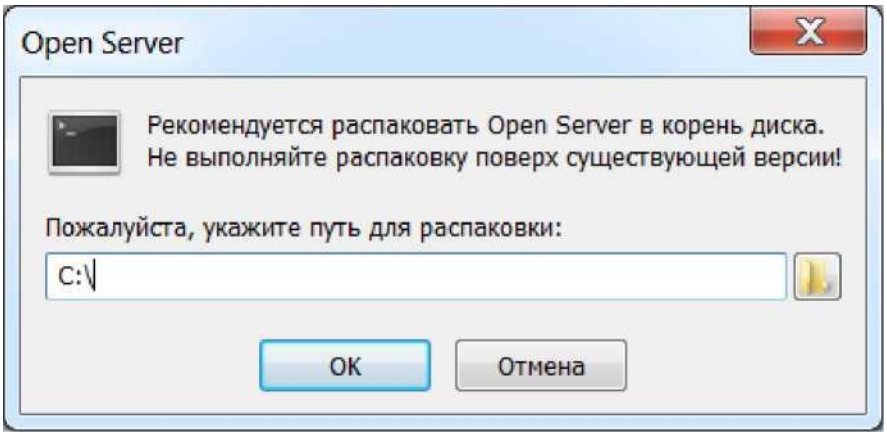


Дистрибутивы

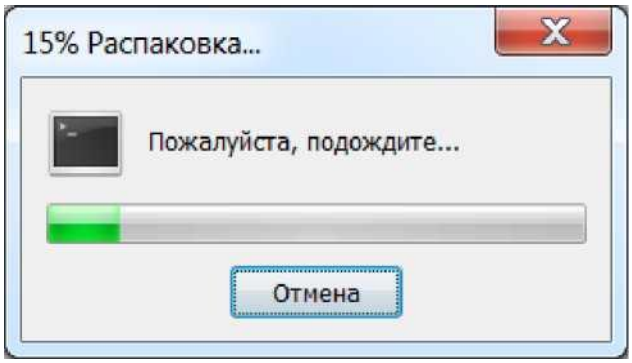
Характеристики дистрибутива
Базовые модули
Git for Windows
ImageMagick
Mongo D B + Rock Mon go
PostgreSQL+ PhpPgAdmin

Как только дистрибутив скачан, начинаем установку.

Дистрибутив представлен в виде самораспаковывающегося архива. Запускаем его и нам сразу же предлагают выбрать диск, куда будет распакован Open Server. Я оставляю по умолчанию диск «С», если вы хотите установить на другой диск (либо же на флешку или переносной диск), то укажите соответствующий путь для распаковки. После того, как диск выбран, начинаем распаковку, нажимаем кнопку «Ok».



Ждем пока распакуется дистрибутив.

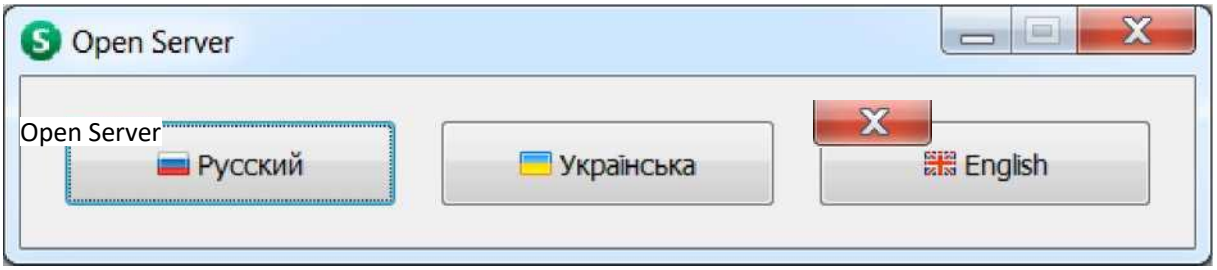


Теперь мы можем запустить программу. Поскольку программа портативная, то никаких ярлыков на рабочем столе или в меню «Пуск» не создается. Поэтому перейдите на диск, куда был распакован Open Server (у меня это C:\OpenServer\) и вы увидите два экзешных файла для запуска Open Server: для 32 (Open Serverx86.exe) и 64 (Open Server x64.exe) разрядной системы Windows. В соответствии с вашей операционной системой Windows вы можете вывести один из ярлыков на рабочий стол, чтобы всегда иметь возможность быстрого доступа для запуска Open Server.

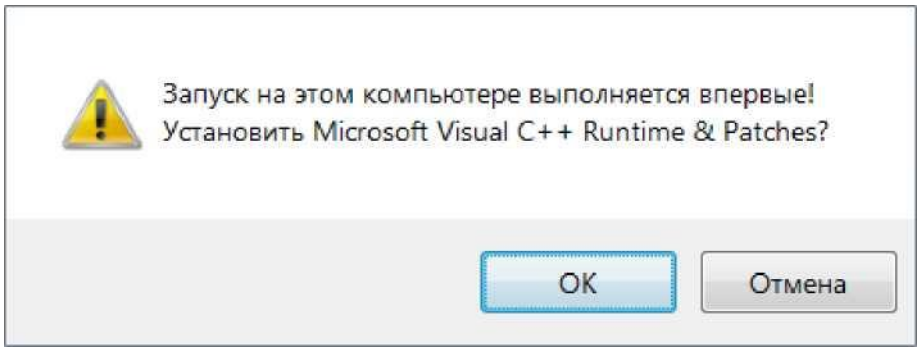
c:\OpenServerV[1].*			
ТИмя	**Тип**	**Размер**	**Дата**
Ml		**<папка>**	**14.07.2017 20:35**

[domains]		**<папка>**	**14.07.2017 20:18**
bfmodulesl		**<папка>**	**14.07.2017 20:18**
b [progs]		**<папка>**	**14.07.2017 20:18**
b [userdata]		**<папка>**	**14.07.2017 20:29**
Open Server *64	**e*e**	**8 753 152**	**08.10.2016 13:59**
Open Server *86	**e*e**	**5 947 392**	**08.10.2016 13:59**

При первом запуске вам предложат выбрать язык.

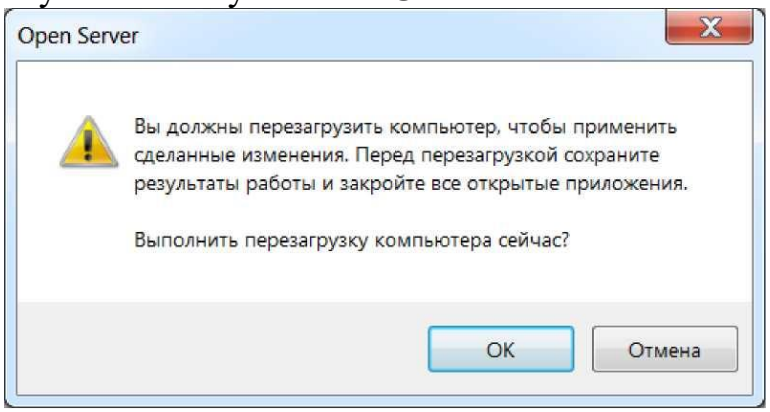


Так же при первом запуске Опен Сервера вам предложат установить патчи для Microsoft Visual C++. Нажимаем «Ok».



Ждем пока закончится установка необходимых библиотек.

И последнее, что нам остается сделать - перезагрузить компьютер, чтобы все изменения вступили в силу. Жмем «Ok».



Кактолько компьютерпере загрузится, можно приступать к аботе с локальным сервером

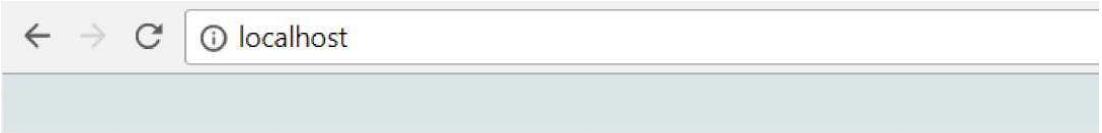
Запуск и проверка работы Open Server

Запустите Опен Сервер и вы увидите, что в трее появился значок в виде красного флажка. Это означает, что программа активирована, но сам сервер пока еще не запущен.

Чтобы запустить сервер, кликните по значку любой кнопкой мыши и в открывшемся контекстном меню выберите пункт - «Запустить». Как только локальный сервер запустится, флажок перекрасится в зеленый цвет.

Теперь Опен Сервер запущен и готов к работе.

Чтобы проверить его работоспособность, перейдите в браузер и в адресной строке введите: http://localhost/. Если вы увидите сообщение: «Добро пожаловать в Open Server!», значит вы все сделали правильно и локальный сервер готов к работе.



Activity 2. Установите сервер на своем компьютере. Результат работы прикрепите в виде скриншотов

ВАШЕ УТОЧНЕНИЕ:

Activity 3. Ответьте на вопросы

Локальный сервер - это ...

Назовите популярные Веб-серверы

Упрощенная схема сайта содержит ...

контент

панель администрирования

шапка сайта

подвал

Практическая работа. HTML: списки

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ.

1. Запустите текстовый и наберите следующий текст:

<HTML>
<HEAD>
<T1^E>Вторая страница </TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>

2. Сохраните файл с именем «ВашаФамилия 18.html»
3. Сверните Блокнот и откройте этот файл при помощи Браузера.
4. После первого слова <BODY> добавьте следующий код:

<UL>
<LI> Первый пункт списка </LI>
<LI> Второй пункт списка </LI>
<LI> Третий пункт списка </LI>
</UL>

5. Сохраните файл и откройте его при помощи браузера. На экране должен появиться маркированный список.
6. Измените данный фрагмент следующим образом и убедитесь, что все по-прежнему работает.

<UL TYPE="disk" >
<LI> Первый пункт списка <LI> Второй пункт списка <LI> Третий пункт списка
</UL>

Изменяя "disk" на circle и square посмотрите, как изменяется вид маркеров.

7. После этого фрагмента добавьте следующий текст:

<OL>
<LI> Первый пункт списка <LI> Второй пункт списка
<LI> Третий пункт списка </OL>

Сохраните файл, откройте его в браузере и убедитесь, что появился нумерованный список.

8. Измените его следующим образом:

<OL TYPE=i START=3>
<LI> Первый пункт списка <LI> Второй пункт списка <LI> Третий пункт списка </OL>

Изменяя i на 1, i, I, a, A убедитесь, меняется вид нумерации, а изменяя цифру 3 можно установить любой начальный номер.

9. Если внутрь нумерованного списка добавить маркированный, следующим образом:

<OL TYPE=”1” START="2">
<LI> Пункт два </LI>

<UL>

<LI> Первый пункт списка <LI> Второй пункт списка <LI> Третий пункт списка

</UL>

<LI> Пункт три </LI>

<LI> Пункт четыре </LI>

</OL>

То на экране должен получится комбинированный список:

2. Пункт два
   - Первый пункт списка
   - Второй пункт списка
   - Третий пункт списка
3. Пункт три
4. Пункт четыре

2. ПРАКТИЧЕСКАЯ ЧАСТЬ.

1. Создать страницу со списком группы используя нумерованный список
2. Создать веб страницу по образцу, используя списки.

Продукция сельского хозяйства

I.пшеница
II.ячмень
III.сахарный тростник
IV.фрукты
V.крупный рогатый скот овиы

Экспорт и импорт Австралии

- Партнеры по экспорту
- США
- Южная Корея
- Новая Зеландия
- Китай
- Партнеры по импорту

Главные статьи импорта

Нефть и нефтепродукты автомобили- компьютеры и офисное оборудование золото, телекоммуникационное оборудование и детали к ним медицинская продукция

Главные статьи экспорта

Минеральное (железные руды, бокситы, уголь) и сельскохозяйственное сырье

Практическая работа. Принципы гипертекстовой разметки. Структура документа

ЦЕЛЬ занятия: познакомить учащихся со структурой веб-документа и принципами гипертекстовой разметки.

Структура документов.

```
<HTML>
<HEAD>
  <TITLE>
    заголовок документа
  </TITLE>
</HEAD>
   <BODY>
      текст
     текст
      .........
     текст
  <ADDRESS>
    текст
  </ADDRESS>
   </BODY>
  </HTML>
```

Документ в целом должен быть отмечен как документ в формате HTML. Для этого он должен начинаться тегом <HTML> и заканчиваться тегом </HTML>. Документ состоит из **2 частей**: ***заголовка (Head), тела документа (Body).***

Часть документа <HEAD>

Формируя структуру документа необходимо использовать тэги **<HEAD>** и **</HEAD>**. Раздел документа HEAD определяет его заголовок и не является обязательным тэгом. Тэги, находящиеся внутри раздела HEAD (кроме названия документа, описываемого с помощью тэга <TITLE>), не отображаются на экране. Контейнер **<TITLE>** является единственным обязательным тэгом заголовка и служит для того, чтобы дать документу название.

Замечание: Обязательность названия документа, вообще говоря, носит характер настоятельной рекомендации. Документ без тэга <TITLE> также будет отображаться браузерами. При этом различные браузеры в качестве заголовка окна будут выдавать различную информацию. Так ранние версии браузера Netscape выдавали строчку "No title". Другие браузеры либо не показывают ничего, либо отображают адрес загруженного файла, повторяя информацию панели Location браузера

Часть документа <BODY>

Атрибуты тега тела документа ***<BODY>:***

BACKGROUND

Атрибут задает графическое изображение, которое как черепица заполнит фон документа. Файл с изображением должен быть сохранен в формате GIF или JPEG.

Синтаксис: <BODY BACKGROUND="(URL)(путь) имя файла">
В данном случае файл с изображением фона лучше размещать в том же каталоге, что и сам файл, тогда (URL) и (путь) указывать не нужно.

BGCOLOR

Этот атрибут задает цвет фона документа при помощи шестнадцатеричных значений интенсивности цветов RGB, или при помощи строчного литерала, соответствующего названию цвета.
Синтаксис: <BODY BGCOLOR="#ff0000"> или <BODY BGCOLOR="RED">

TEXT

Этот атрибут задает используемый по умолчанию цвет текста, который не является гиперссылкой. По умолчанию такой текст будет черным.
Синтаксис: <BODY TEXT=" цвет ">

LINK

Этот атрибут задает цвет гиперссылки (еще не просмотренной гиперссылки), в большинстве браузеров он задан по умолчанию темно синим.
Синтаксис: <BODY LINK="цвет" >

ALINK

Этот атрибут задает цвет активной гиперссылки, он меняет цвет гиперссылки в момент щелчка по ней мышью, не желательно задавать ему цвет фона по понятным причинам.
Синтаксис: <BODY ALINK="цвет" >

VLINK

Этот атрибут задает цвет посещенной (просмотренной) гиперссылки, не желательно задавать ему цвет фона и цвет атрибута LINK по понятным причинам.
Синтаксис: <BODY VLINK="цвет" >

BGPROPERTIES

Этот атрибут задает свойства фонового изображения. В данный момент браузерами поддерживается единственное его значение fixed, запрещающее скроллинг изображения.
Синтаксис: <BODY BGPROPERTIES="fixed" >

TOPMARGIN

Этот атрибут задает верхнюю границу страницы в пикселях.
Синтаксис: <BODY TOPMARGIN=число >

BOTTOMMARGIN

Этот атрибут задает нижнюю границу страницы в пикселах.
Синтаксис: <BODY BOTTOMMARGIN=число >

LEFTMARGIN

Этот атрибут задает границу страницы в пикселях слева.
Синтаксис: <BODY LEFTMARGIN=число >

RIGHTMARGIN

Этот атрибут задает границу страницы в пикселях справа.
Синтаксис: <BODY RIGHTMARGIN=число >

SCROLL

Устанавливает наличие или отсутствие полос прокрутки окна.

Синтаксис: <BODY SCROLL=YES|NO >

ЗАДАНИЕ: Создайте простую веб-страницу с *указанием структуры документа: заголовка и тела документа.*

Использование контейнера

Создание тела документа с

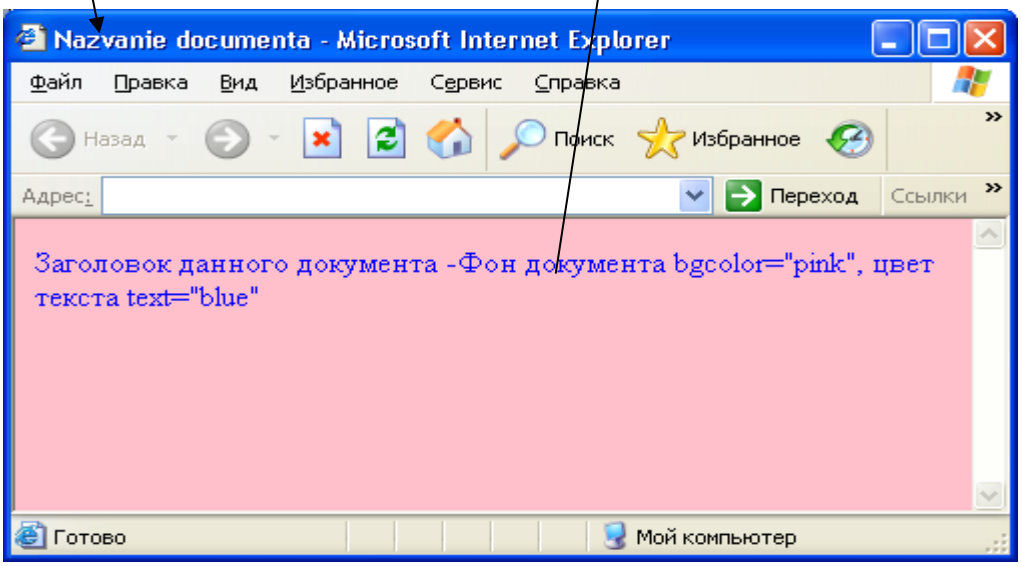


Рисунок 1.1 Задание

Практическая работа. Структурное форматирование документа

Цель: познакомится с тегами для форматирования документа

Элемент <COMMENT >

Элемент позволяет скрыть от пользователя *комментарии* к исходному коду, а так же для сокрытия сценариев Java Script от браузеров, которые не поддерживают их.

Синтаксис: <COMMENT> Текст комментария </COMMENT> Полностью аналогичен старому варианту задания комментариев *Синтаксис: <!- - Текст комментария //- ->*

ЗАДАНИЕ: Создайте веб-документ с *использованием комментариев, как отображено на рисунке 2.1.*

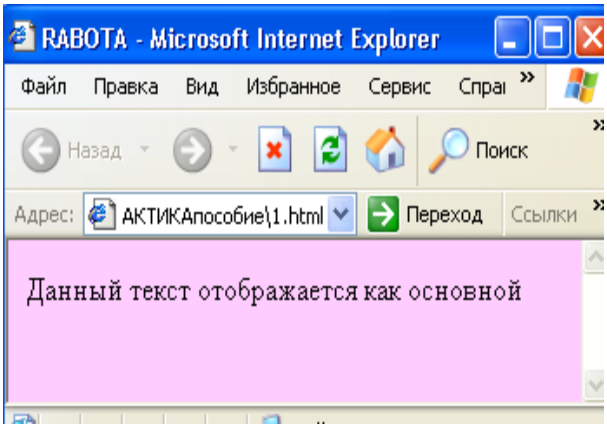


Рисунок 2.1. Использование

Элементы <H*n*>

Уровни заголовков документа могут иметь шесть различных уровней (размеров) и представляют собой фрагменты текста. Соответствующие каждому уровню гарнитура и размер шрифта зависят от браузера, стилю <H1> назначается самый большой и самый жирный шрифт, а стилю <H6> назначается самый маленький и самый невзрачный шрифт.

Элемент может иметь атрибут align, который указывает отступ left, center, right.

*Синтаксис: <H**n** align=**отступ**> Текст заголовка </H**n**>*

ЗАДАНИЕ: *Применяя уровни заголовков, создайте веб-страницу как на рисунке 2.2.*

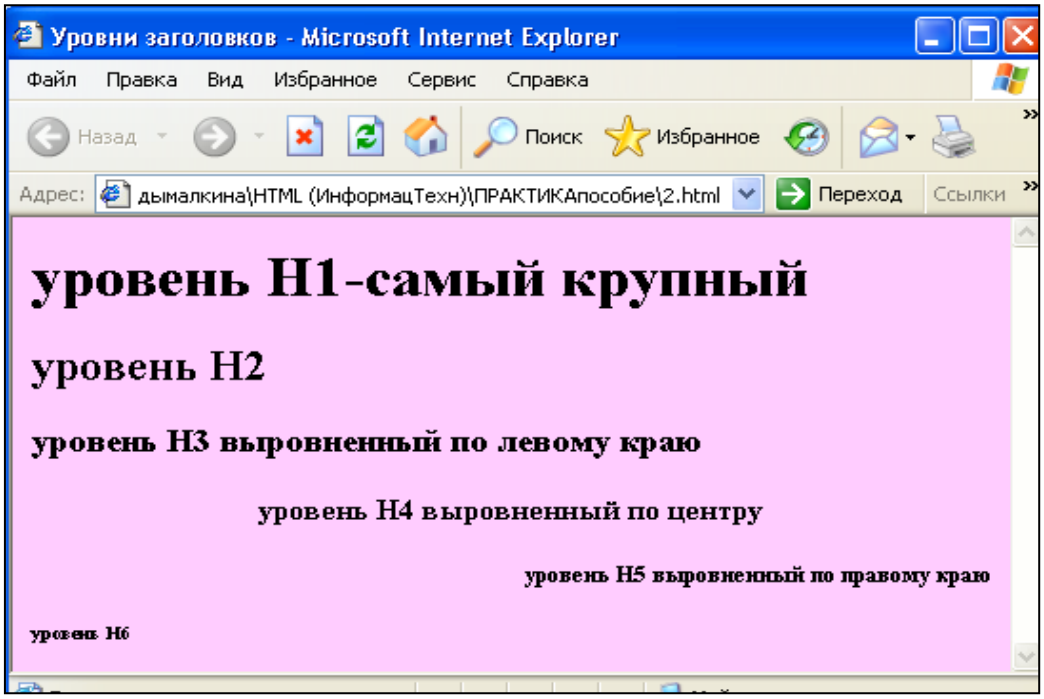


Рисунок 2.2. Рассмотрение уровней

Элемент <HR>

Используется для проведения разделительной линии в документе. Он может иметь атрибуты :

color, задающий цвет линии;

size высота в пикселах;

width ширина в пикселях или процентах от ширины экрана;

align режим выравнивания (значения - left, center, right);

noshade вывод «плоской» линии (так как по умолчанию браузер рисует линии «объемными»).

*Синтаксис: <HR align="**center**" size=**n** width=**n** color="**цвет**" noshade>*

ЗАДАНИЕ: Используя элемент *линия с различным применением атрибутов, создайте веб-страницу как на рисунке 2.3.*

рисование линии без указания параметров

пример 2

линия шириной 50% браузера,высотой 20 пикселей,выравненная по правому краю

пример 3

линия шириной 50% браузера,высотой 20 пикселей,выравненная по правому краю, с применением параметра NOSHADE

пример 4

линия шириной 300 пикселей,размером(высотой) 50 пикселей браузера, цветом "hotpink",выравненная по левому краю

Рисунок 2.3. Рисование линий разных вариаций

Элемент <P>

Этот элемент задает один из способов разбиения текста на абзацы. Он может иметь вложенный атрибут align, который указывает отступ left, center, justify, right. Каждый следующий абзац игнорирует значение атрибута align, заданное для предыдущего абзаца.

Атрибут STYLE - стиль текста (<P style="font-size:20; font-style:italic; color:red")

*Синтаксис: <P align=**отступ**> Текст абзаца </P>*

Элемент <DIV>

Позволяет выделить в структуре документа несколько разделов. Он является блочным элементом, функционирующим во многом подобно элементу <P>. Если закрывающий тэг </P> опущен, то <DIV> эффективно заменяет его и начинает новый абзац. Он может иметь атрибут align, который указывает отступ left, center, justify, right. Каждый следующий раздел игнорирует значение атрибута align, заданное для предыдущего раздела.

*Синтаксис: <DIV align=**отступ**> Текст раздела </DIV>*

Элемент

Этот элемент задает разрыв текста с переходом на новую строку. Атрибут CLEAR= в таге
 используется для того, чтобы остановить в указанной вами точке обтекание текстом объекта и затем продолжить текст в пустой области за объектом, вставленного в текст нестандартным способом. Продолжающийся за объектом текст выравнивается в соответствии со значениями LEFT, RIGHT или ALL атрибута CLEAR=:

<BR CLEAR="LEFT"> - текст будет продолжен, начиная с ближайшего пустого левого угла

RIGHT – правого угла

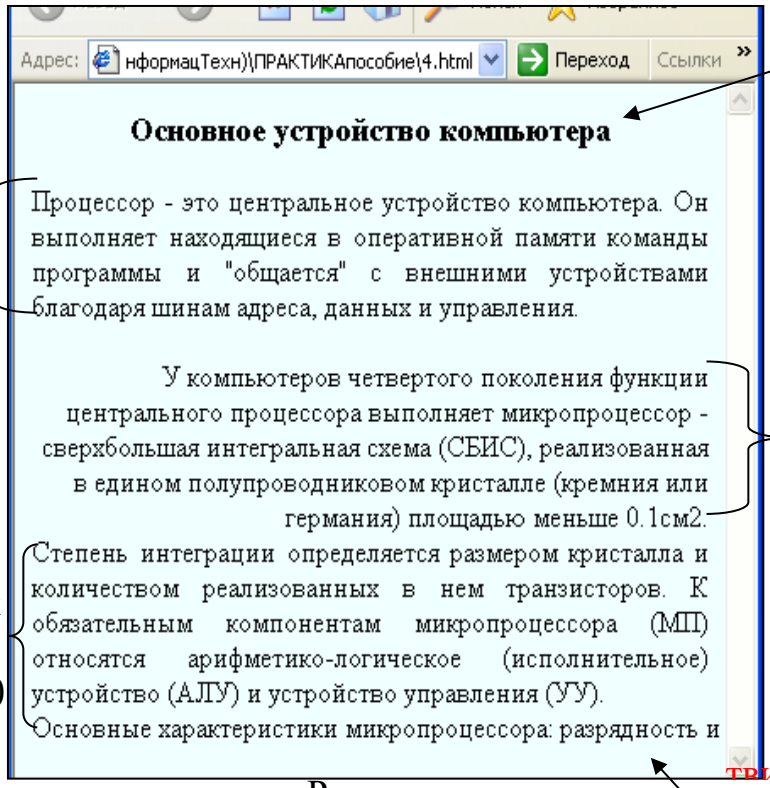
ALL - текст будет продолжен, как только и левое и правое поля окажутся пустыми

Каждый следующий абзац игнорирует, заданное для предыдущего абзаца значение clear.

*Синтаксис: <BR clear=**обтекание**>*

Перевод текста может быть отменен тэгами <NOBR> и </NOBR>

ЗАДАНИЕ: *Примените теги <P>, <Div>, <BR>, <NOBR> с их атрибутами как показано на рисунке 2.4.*



К данному тексту применен тег разделов <DIV>, с атрибутом выравнивания по центру

Новый абзац заданный тегом <P>, с атрибутом выравнивания по ширине

Данный текст переведен на новую строку. Он задан как новый абзац (тег <P>), текст которого

Этот фрагмент представлен как новый блок (с пом. тега<DIV>) в абзаце, который уже выравнен по ширине

быстродейс

твие

Рисунок 2.4.Применение тегов структурного

К данной строке применен тег <NOBR>, позволяющий отобразить строку без разрыва (т.е. слово «быстродействие» отображается дальше в этой же строке)

Элемент <PRE>

Тег позволяет осуществить отображение текста без форматирования. Весь текст, заключенный в тэги <PRE> и </PRE> будет визуализирован браузером точно так, как он визуализирован в исходном коде документа, кроме того текст выводится моноширинным шрифтом, что значительно упрощает задачу форматирования текста в колонки. Элемент поддерживается не всеми браузерами, он может иметь атрибут width, который задает ширину отводимого пространства под текст в символах. Элемент сменил собой устаревшие элементы

<XMP>,<LISTING>и<PLAINTEXT>.

*Синтаксис: <PRE width=**число символов** >**...текст.. .**</PRE>*

ЗАДАНИЕ: *Примените тег <Pre>, для создания расписания. (см.рис.2.5).*

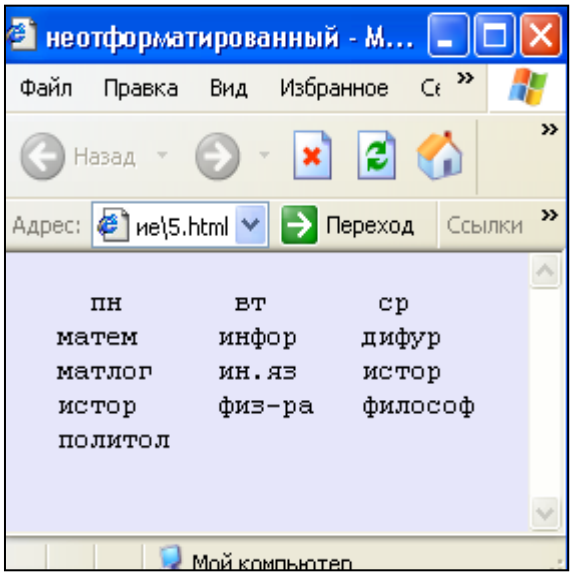


Рисунок 2.5. Написание текста без форматирования

Элемент <CENTER>

Выравнивание по центру.

Синтаксис: <CENTER> текст </CENTER> полностью идентичен конструкции <DIV align=center> Текст раздела </DIV>

Элемент <ADDRESS>

Тэг <ADDRESS> применяется для идентификации автора документа и для указания адреса автора. Сюда же обычно помещаются сведения об авторских правах. Этот элемент располагается либо в начале, либо в самом конце документа. Часто в этом элементе указывают дату создания и последнего обновления документа. Это позволяет читателям определить, знакомились ли они ранее с версией, которую просматривают. Текст, заключенный между этими тэгами, обычно показывается браузерами курсивом.

*Синтаксис: < ADDRESS>**контактная информация** </ADDRESS>*

Элемент <BLOCKQUOTE>

Бывают случаи, когда в текст HTML-документа необходимо включить какую-либо длинную цитату. Для выделения цитат из основного текста существует тэг <BLOCKQUOTE>. Он является контейнером и может содержать любые тэги форматирования.

<BLOCKQUOTE> - добавляет поля слева и справа от текста при указании определенных атрибутов.

Атрибуты (n -число пикселей):

LEFTMARGIN=n –задает левое поле для всей страницы

TOPMARGIN=n - определяет верхнее поле

Синтаксис: <BLOCKQUOTE> ***Текст*** *</BLOCKQUOTE>*

Элемент <SPAN>

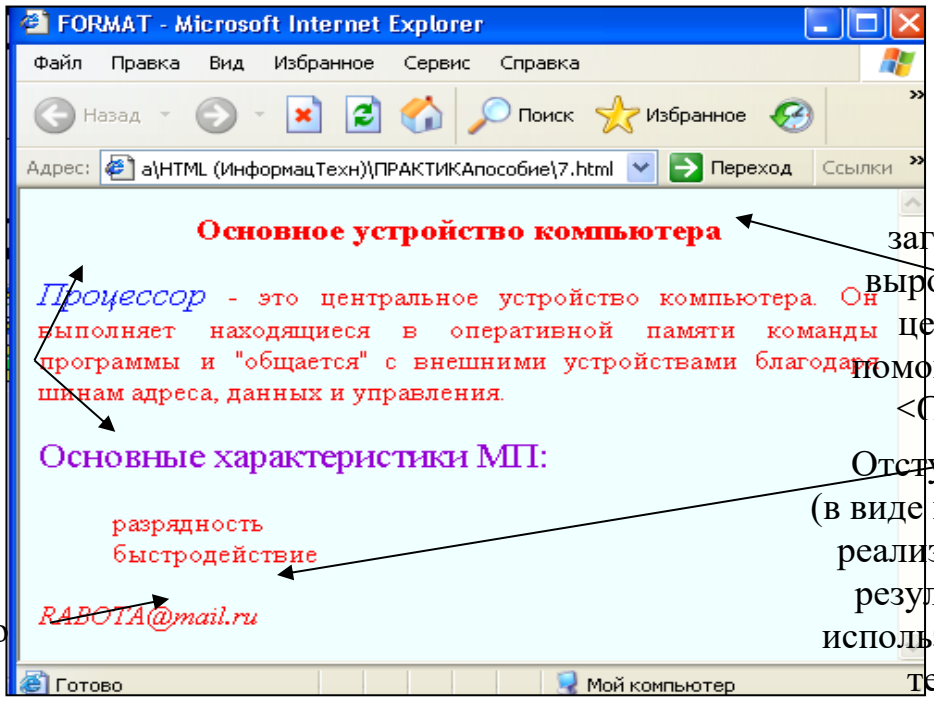
Тег <SPAN> определяет контейнер для внутреннего текста, позволяет выделить некоторое количество текста для последующего его форматирования, но в отличие от <DIV> не начинает новый абзац. Атрибуты:

TITLE - всплывающая подсказка

UNSELECTABLE - запрещает или разрешает выделять текст в данном элементе (on - запрещает, off - разрешает)

STYLE – стиль текста (font-size: 20pt; font-style: italic; color: red)

Синтаксис: <SPAN TITLE="подсказка" UNSELECTABLE=off > ***Текст*** *</SPAN>*



Данные фрагменты текста отформатированы с помощью тега <Span>

Использование тега <ADDRESS>

Данный заголовок выровнен по центру с помощью тега <Center>

Отступ текста (в виде цитаты), реализован в результате использования тега

Рисунок 2.6.Указание отступов. Дополнительные параметры выравнивания

ЗАДАНИЕ: *Используя теги <CENTER>, <BLOCKQUOTE>, <ADDRESS> создайте документ как на рисунке 2.6.*

ПРОЕКТ №1 «Месяца времен года». Используя теги и соответствующие атрибуты, рассмотренные ранее, создайте документ как на рисунке 2.7.

Цель: научится применять ранее изученные теги для создания веб-документа

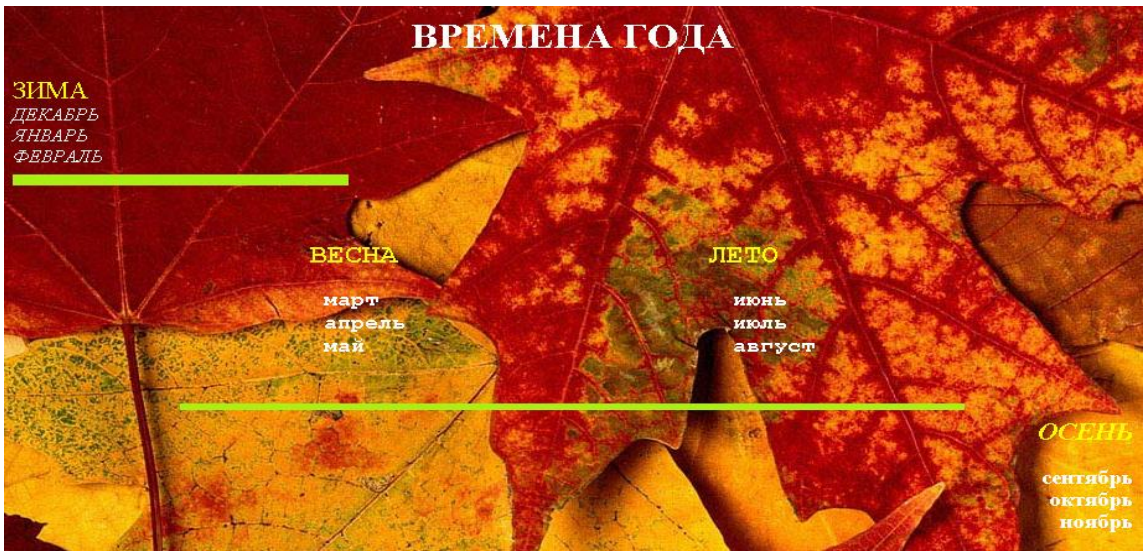


Рисунок 2.7. Итоговый документ проекта

Практическая работа. Форматирование символов

Цель: познакомится с тегами для форматирования символов.

Элементы физического стиля

Под физическом стилем принято понимать прямое указание браузеру на модификацию текущего шрифта.

Для выделения отдельных частей текста можно воспользоваться следующими тегами стилей:

<B>..</B> – жирное усиление (Bold);------- пример

<I> ..</I>- курсив (Italic);--------- *пример*

<U>..</U>- с подчеркиванием (Underline);------- пример

<S>..</S> - с перечеркиванием (STRIKE);-------- ~~пример~~

<TT>..</TT> - отображает текст моноширинным шрифтом.

<BIG>..</BIG> - увеличенный фонт

<SMALL>..</SMALL> - уменьшенный фонт

Верхние и нижние индексы:

_{..} - подстрочный индекс, например, H_2SO_4.

^{..} - надстрочный индекс, например, $(a^2 - b^2) = (a - b)(a + b)$.

Все вышеприведенные команды определяют физическое форматирование символов. Вы задаете конкретное изменение характеристик символов.

Синтаксис: <B> ***Текст*** *</B> (аналогично остальные теги стилей)..*

ЗАДАНИЕ: *Примените теги для физического стиля к тексту (см.рис.2.8).*

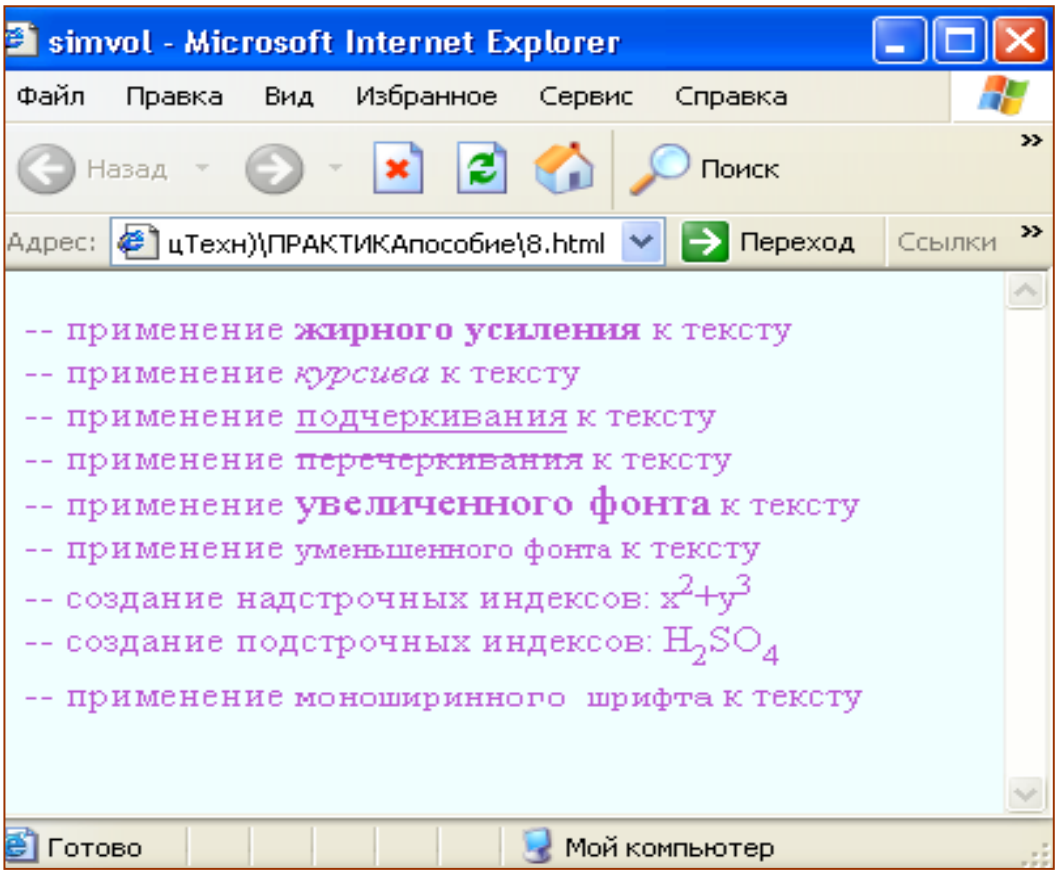


Рисунок 3.1. Использование тегов физического стиля

Элементы логического стиля

При использовании логических стилей автор документа не может знать заранее, что увидит на экране читатель. Разные браузеры толкуют одни и те же метки логических стилей по-разному. Некоторые браузеры игнорируют некоторые метки вообще и показывают нормальный текст вместо выделенного логическим стилем. Вот самые распространенные логические стили.

<EM>...</EM>

От английского emphasis - акцент. Используется для выделения, важных фрагментов текста курсивом (наклонный текст). Элементы EM и I аналогичны по своему действию.

<STRONG>...</STRONG>

От английского strong emphasis - сильный акцент. Используется для отображения текста жирным шрифтом. Элементы B и STRONG аналогичны по своему действию.

<CODE>...</CODE>

Аналог тегов, выполняет точно такие же функции. Задумывался для отображения формул, или программного кода.

Атрибуты: TITLE -всплывающая подсказка.

<SAMP>...</SAMP>

От английского sample - образец. Рекомендуется использовать для демонстрации образцов сообщений, выводимых на экран программами.

Используется для выделения нескольких символов шрифтом фиксированной ширины.

<VAR>...</VAR>

От английского variable - переменная. Используется для выделения переменных в листинге программы. Рекомендуется использовать для написания имен переменных. Обычно такой текст отображается курсивом.

<CITE>...</CITE>

Оформляет находящийся текст как цитату или ссылку на источник. Обычно используется для коротких цитат (в отличие от элемента BLOCKQUOTE). Цитируемый текст отображается курсивом.

Атрибуты: TITLE -всплывающая подсказка.

<DFN>..</DFN>

Элемент <DFN> отмечает текст как определение (DeFeNition). Элемент <DFN> используется с целью обозначения терминов и определений по типу словарей, глоссариев.

<DL>..</DL> - позволяет создать отдельные абзацы с отступом без нумерации или маркеров

<BLINK>..</BLINK>

В некоторых Web–документах можно встретить мигающие строки текста, призванные привлечь внимание пользователя. Для этих целей используется дескриптор <BLINK>.

<INS>..</INS>

Элемент <INS> используется с целью выделения особым шрифтом слова или текста, когда требуется показать явно, что текст был вставлен после опубликования документа.

<DEL>..</DEL>

Элемент <DEL> используется с целью выделения особым шрифтом слова или текста, когда требуется показать явно, что текст был удален после опубликования документа.

ЗАДАНИЕ: *Примените теги для логического стиля к тексту (см.рис.3.2).*

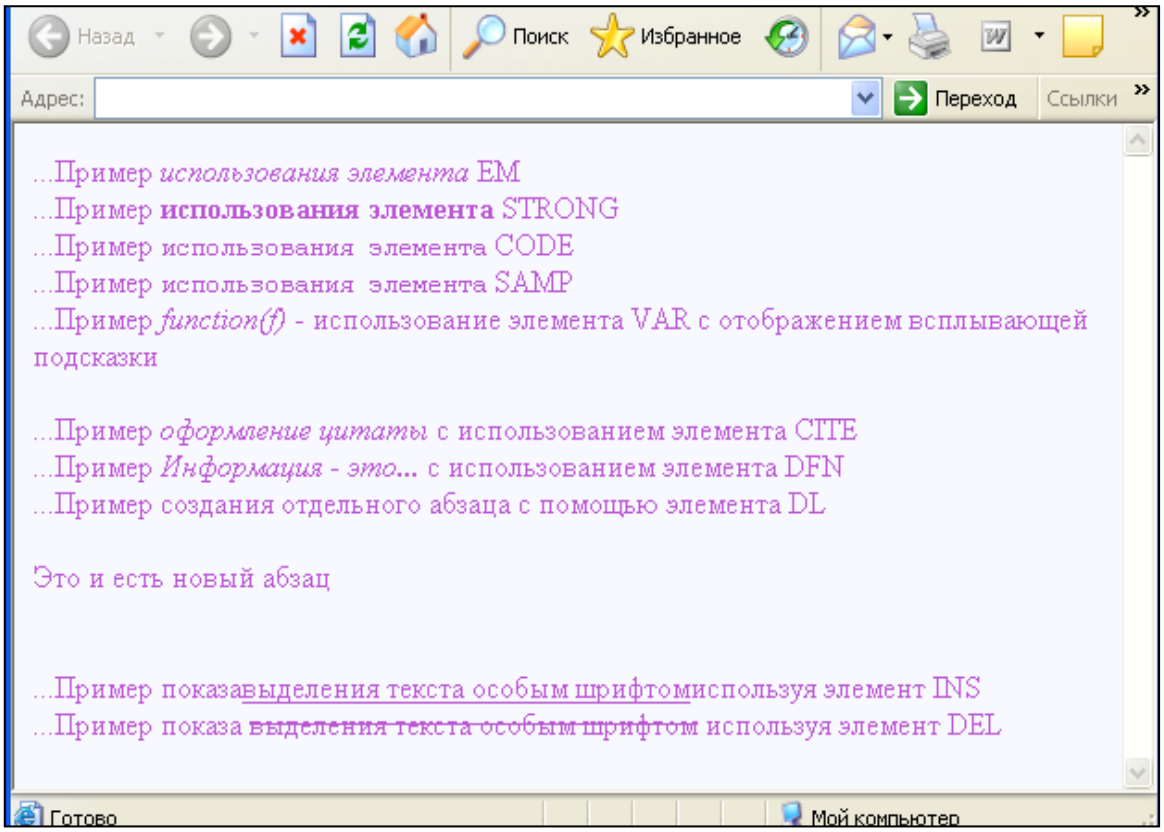


Рисунок 3.2. Использование тегов

Элемент <BASEFONT>..</BASEFONT>

Тег <basefont> не имеет завершающего (парного) тега. Определяет основной шрифт, которым должен отображаться текст документа. Впоследствии Вы можете легко изменить шрифт в любой части документа используя тег <font>. После закрывающего тэга <font> действие тэга <BASEFONT> восстанавливается. Значения параметров шрифтов, используемых по

умолчанию, могут неоднократно переопределяться в документе, т.е. тэг <BASEFONT> может появляться в документе любое количество раз. Действие элемента BASEFONT не распространяется на текст, заключенный в ячейки таблиц и другие сепаративные элементы html.

Атрибуты:

size - позволяет задать базовый размер шрифта во всем документе. Возможные значения: целые числа от 1 до 7 включительно, по умолчанию значение его задается равным 3.

face - определяет гарнитуру шрифта (Параметр face пока рекомендуется не использовать, т.к. его поддерживают не все типы браузеров).

Синтаксис: <BASEFONT size=n>

Элемент <FONT>…</FONT>

Элемент <FONT> используется с целью выделения особым шрифтом слова или текста.

Атрибуты:

size - определяет размер текста:

абсолютные размеры: в пределах от 1 до 7,где 1 - самый мелкий шрифт. По умолчанию равен 3;

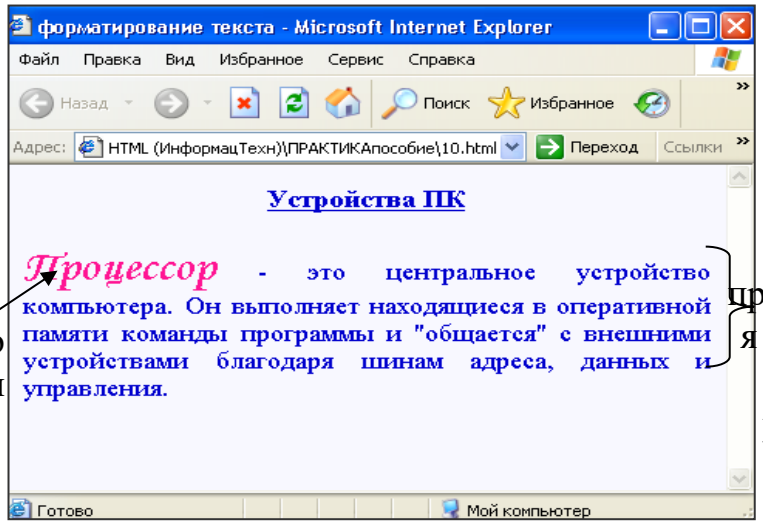
относительные размеры: с указанием знака от -2 до +4, где -2 самый мелкий. Вычисляется путем сложения с базовым размером, определенным с помощью параметра SIZE элемента BASEFONT.

color – определяет цвет текста, задается либо RGB-значением в шестнадцатеричной системе, либо одним из 16 базовых цветов.

face - определяет гарнитуру шрифта (Этот параметр служит для указания типа шрифта, которым программа просмотра пользователя будет выводить текст (если такой шрифт имеется на компьютере). Можно указать как один, так и несколько названий шрифтов, разделяя их запятыми. Это весьма полезное свойство, так как в разных системах могут быть почти идентичные шрифты, называющиеся по-разному. Другим важным качеством является задание предпочтения использования шрифтов. Список шрифтов просматривается слева направо. Если на компьютере пользователя нет шрифта, указанного в списке первым, то делается попытка найти следующий шрифт и т.д. Если такого шрифта не будет найдено, то данное указание будет проигнорировано и будет использован шрифт, установленный по умолчанию. Совет: число различных шрифтов, используемых в одном документе, не должно превышать трех. ПРИМЕР: <FONT FACE="Verdana", "Arial", "Helvetica">).

Синтаксис: <FONT size=n color="цвет" face=”шрифт”> Текст </FONT>

ЗАДАНИЕ: *Примените теги <BASEFONT> и <FONT> к тексту (см.рис.2.10).*



К слову «процессор» применен элемент <font> с

К тексту применяются значения цвета, размера

Рисунок 3.3. Применение тегов указания параметров текста

ПРОЕКТ №2 «Рисование грамоты». Используя теги и соответствующие атрибуты, рассмотренные ранее, создайте документ (см.рис.3.4). В качестве фона данного документа используется графическое изображение. Применение различных цветов к тексту осуществляется за счет своевременного открытия и закрытия тегов, позволяющих форматировать текст.

Цель: продолжать развивать профессиональные навыки при работе с текстов в веб-документах.

Рисунок 3.4. Итоговый документ проекта

ПРАКТИЧЕСКАЯ РАБОТА. СПИСКИ

Цель: познакомится с тегами для создания различных видов списков.

Нумерованный список

<OL>..</OL> - *упорядоченный (нумерованный)* список *(OL - Ordered Lists).*

В нумерованном списке все записи пронумерованы. Каждый элемент списка начинается с команды: : (List Item <LI>)

В команде <LI> может быть параметр TYPE, который определяет тип нумерации и может иметь следующие значения:

Атрибут	Назначение
COMPACT	Представляет список в более компактном виде.
TYPE=A	Устанавливает маркер в виде прописных букв (A, B, C ...).
TYPE=a	Устанавливает маркер в виде строчных букв (a, b, c ...).
TYPE=I	Устанавливает маркер в виде больших римских цифр (I, II, III ...)
TYPE=i	Устанавливает маркер в виде маленьких римских цифр (i, ii, iii ...) .
TYPE=1	Устанавливает маркер в виде арабских цифр (1, 2, 3 ...) .
START=n	Устанавливает начальный маркер в текущем списке (при чем значение n указывается цифрой не зависимо от типа нумерации:пример. type=A start=3, в итоге нумерация будет начинаться с буквы C).

Синтаксис:

```
<OL TYPE="тип" START=число>
 <LI>Первый элемент</LI>
 <LI>Второй элемент</LI>
   ...
</OL>
```

ЗАДАНИЕ: *Создайте нумерованный список с использованием элемента <OL> (см.рис.4.1). Во втором списке с помощью атрибута START тега <OL> нумерация начинается с 3-его номера.*

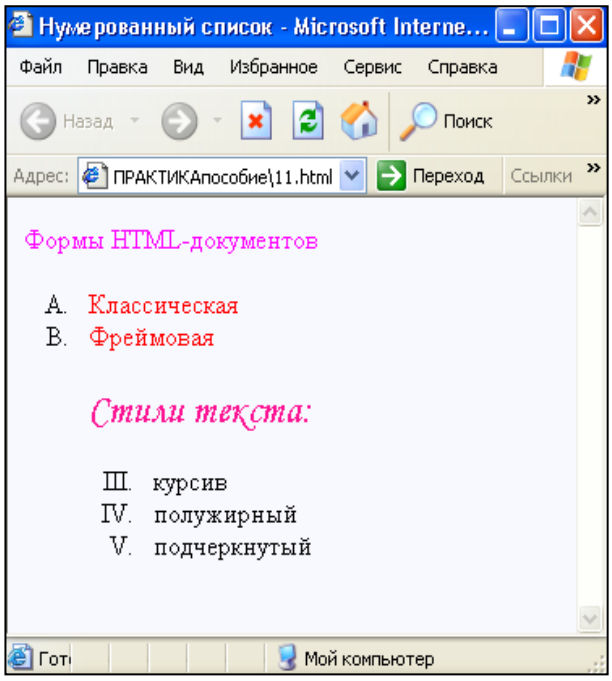


Рисунок 4.1.Создание

Маркированный (ненумерованный) список

<UL>..</UL> - *маркированный (ненумерованный)* список использует для выделения записей специальными символами (UL-Unordered Lists). Каждый элемент списка начинается с команды: (List Item <LI>)

Маркированный список поддерживает атрибут type, который определяет тип маркировочного символа и может иметь следующие значения:

Атрибут	Назначение
TYPE=DISK	круглая жирная точка
TYPE=CIRCLE	окружность
TYPE=SQUARE	маленький черный квадрат

Можно включать такие атрибуты в сами теги <LI>, чтобы сменить тип маркера в середине списка. После появления нового атрибута все последующие маркеры в списке будут иметь такой же вид.

<u>Атрибуты маркеров в ненумерованном списке</u>

PLAN= создает ненумерованные списки без маркеров.

SRC= используется для того, чтобы задать GIF- файл вместо обычного маркера

DINGBAT= позволяет создавать специальные типографские символы dingbats, поддерживаемые браузером. Эти символы имеют вид картинок, которые используются в качестве маркеров в списках.

Синтаксис:

```
<UL TYPE="тип">
<LI>Первый элемент</LI>
<LI>Второй элемент</LI>
</UL>
```

ЗАДАНИЕ: *Создайте нумерованный список с использованием элемента <UL> (см.рис.4.2). Во втором списке тип нумерации прерывается во втором элементе списка написанием атрибута TYPE= square.*

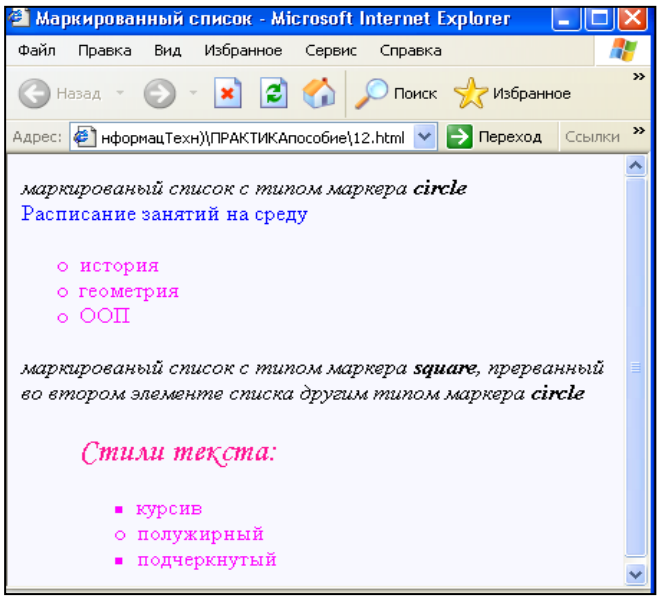


Рисунок 4.2. Создание нумерованных списков

Список-определение

<DL>..</DL> - В *списке-определении* все записи сдвинуты влево при помощи табуляции. Используется с целью задания словарей, глоссариев и прочих перечней. Данный список служит для создание списков типа *"термин"-"описание"*. Каждый термин начинается тэгом <DT> , а описание - тэгом <DD>.

Синтаксис:

<DL >
<DT> термин 1 <DD> определение 1
<DT> термин 2 <DD> определение 2
</DL>

ЗАДАНИЕ: *Создайте список типа: «СПИСОК-ОПРЕДЕЛЕНИЕ» с использованием элемента <DL> (см.рис.4.3).*

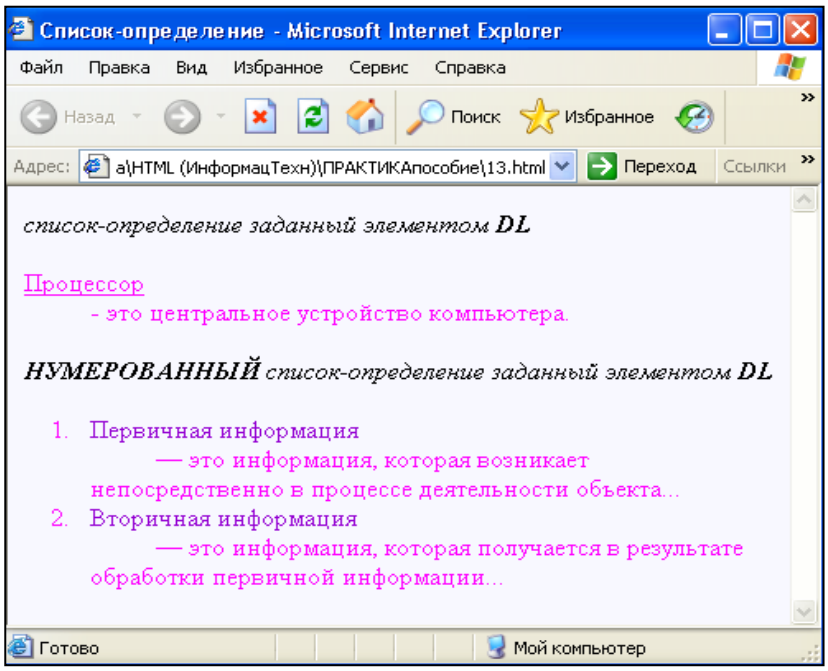


Рисунок 4.3. Создание списков типа:

Вложенный список

Во ВЛОЖЕННЫХ СПИСКАХ списки можно встраивать один в другой, то есть образуется несколько уровней вложенности.

<MENU>..</MENU>

Элемент <MENU> используется с целью создания списков по логическому определению, такие списки могут связываться с другими элементами документа логически.

Синтаксис: <MENU><LI> элемент списка <LI> элемент списка </MENU>

<DIR>..</DIR>

Элемент <DIR> используется с целью создания списков по логическому определению, такие списки могут связываться с другими элементами документа логически.

Синтаксис: <DIR><LI> элемент списка <LI> элемент списка </DIR>

ЗАДАНИЕ: *Создайте вложенный список с использованием элементов создания списков (см.рис.4.4).*

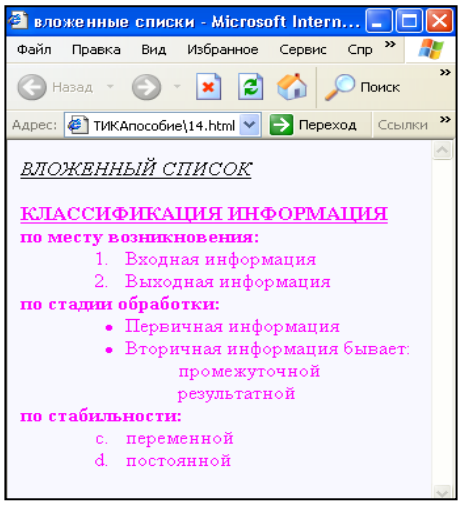


Рисунок 4.4. Создание вложенных списков

ПРОЕКТ №3 «Услуги оздоровительного комплекса». Используя теги и соответствующие атрибуты, рассмотренные ранее, создайте документ (см.рис.4.5).

Цель: закрепить ЗУН по работе с разными видами списков.

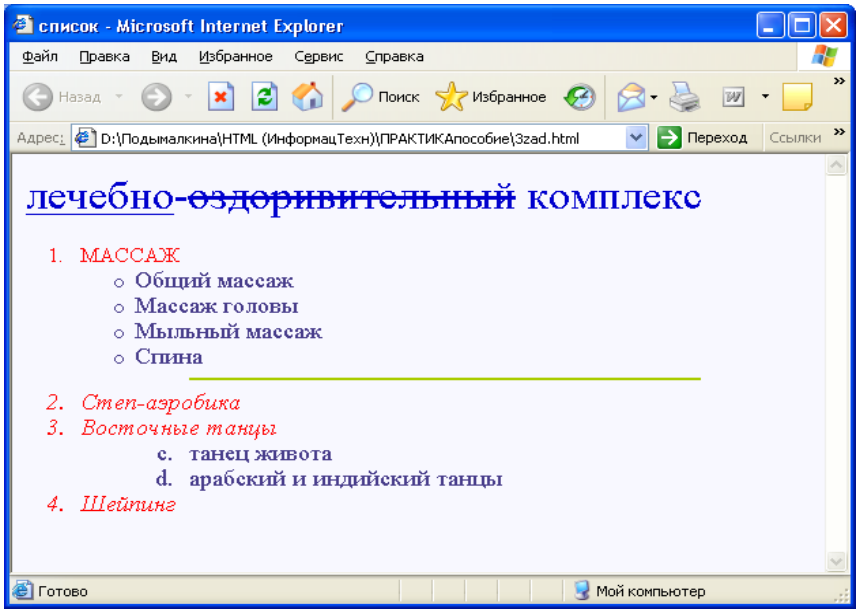


Рисунок 4.5. Итоговый документ проекта «Услуги оздоровительного комплекса»

Практическая работа .Вставка объектов в документ

Цель: познакомится с тегами для возможности вставки объектов в документ

Так как язык **HTML** создавался как язык разметки текста и только для этих целей, то естественно, что в нем изначально не была предусмотрена поддержка мультимедийных объектов. В последнее же время многие компании-разработчики стали предоставлять пользователям их браузеров доступ к мультимедиа-содержимому, встраивая в них различные **plug-ins**. Они позволяют представить мультимедиа-содержимое как **внутреннее** содержимое **Web**-страницы и могут располагаться как во всё окно, так и в заданных размерах. На практике получается следующее - если браузер подгружает файл с таким содержимым впервые, то пользователю предлагается скачать **плагин**, который запишется на жесткий диск и в следующий раз браузер передаст мультимедиа-содержимое на обработку уже сразу ему.

Элемент <IMG>

Используется для вставки в тело документа графического изображения. Под графическим изображением подразумевают: *маленькие пиктограммы, рисунки, графические объекты и карты изображений*, занимающие большую часть окна браузера. Кроме того, элемент **<IMG>** поддерживает различные атрибуты, определяющие расположение изображения относительно окружающего текста и содержания **Web** страницы в целом. Теперь рассмотрим, как вставить изображение в страницу Web тегом HTML, который заставляет браузер выводить изображение, является <IMG> со следующим общим форматом:

Синтаксис: **<IMG SRC="picture.gif">**

Замыкающего тега не требуется. Здесь SRC означает источник (SouRCe), а имя файла представляет собой имя выводимого графического файла. Изображения на странице Web могут быть использованы и в качестве гипертекстовых ссылок, как и обычный текст.

Атрибуты элемента <IMG>:

ALIGN –атрибут выравнивания изображений

При включении графического изображения в документ можно указывать его расположение относительно текста или других элементов страницы. Способ выравнивания изображения задается следующими значениями параметра **ALIGN** тэга <IMG>:

ЗНАЧЕНИЕ АТРИБУТА *ALIGN*	НАЗНАЧЕНИЕ
TOP	Верхняя граница изображения выравнивается по самому высокому элементу текущей строки
TEXTTOP	Верхняя граница изображения выравнивается по самому высокому текстовому элементу текущей строки
MIDDLE	Выравнивание середины изображения по базовой линии текущей строки
ABSMIDDLE	Выравнивание середины изображения посередине текущей строки

BASELINE или BOTTOM	Выравнивание нижней границы изображения по базовой линии текущей строки
ABSBOTTOM	Выравнивание нижней границы изображения по нижней границе текущей строки
LEFT	Изображение прижимается к левому полю окна. Текст обтекает изображение с правой стороны
RIGHT	Изображение прижимается к правому полю окна. Текст обтекает изображение с левой стороны

Поясним действие параметров выравнивания, приведенных выше в таблице. Все значения параметров выравнивания изображений можно условно разделить на две группы по их принципу действия.

К одной группе относятся два значения параметра — LEFT и RIGHT. При использовании любого из этих параметров мы получаем так называемое "плавающее" изображение. В этом случае изображение прижимается к соответствующему краю окна просмотра браузера, а последующий текст (или другие элементы) "обтекают" изображение с противоположной стороны. Здесь текст, размещаемый рядом с изображением, может занимать несколько строчек.

К другой группе значений параметров относятся все остальные. При их использовании изображение как бы встраивается в строчку текста, а параметры выравнивания задают расположение изображения относительно строки текста. Таким образом, в отличие от плавающих изображений, здесь изображение является обычным элементом строки. Это легко понять, если представить, что изображение является просто одной буквой строки текста, правда, достаточно большой (типа буквицы).

ЗАДАНИЕ: *Добавьте в веб-страницу изображения и примените разные параметры выравнивания (см.рис.5.1).*

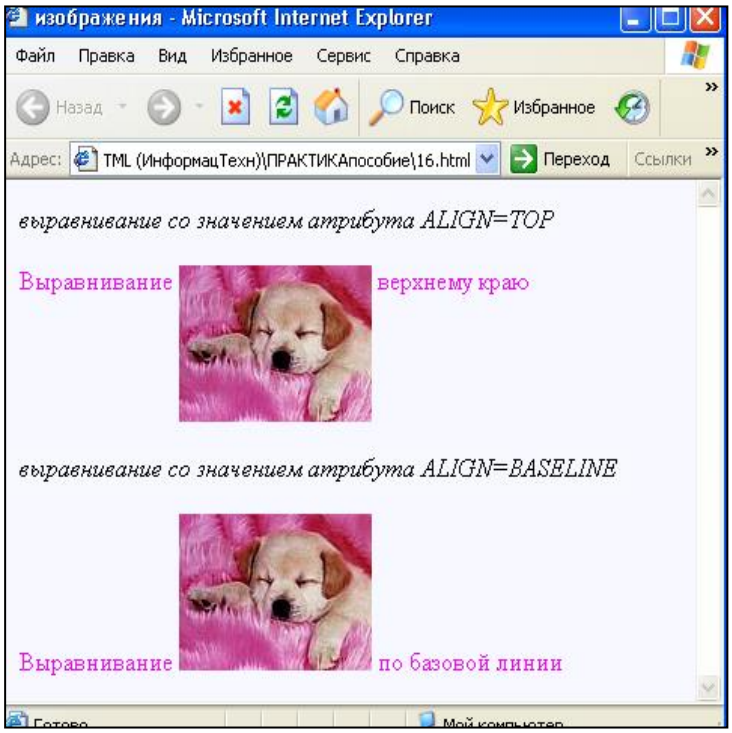


Рисунок 5.1. Применения параметров выравнивания к

ЗАДАНИЕ: *Создайте плавающее изображение (см.рис.5.2).* В данном задании изображение прижато к правому краю окна просмотра браузера, а последующий текст располагается с левой стороны от изображения. Количество строк, располагаемое рядом с изображением, может изменяться в зависимости от размеров шрифта текста, а также размеров окна просмотра.

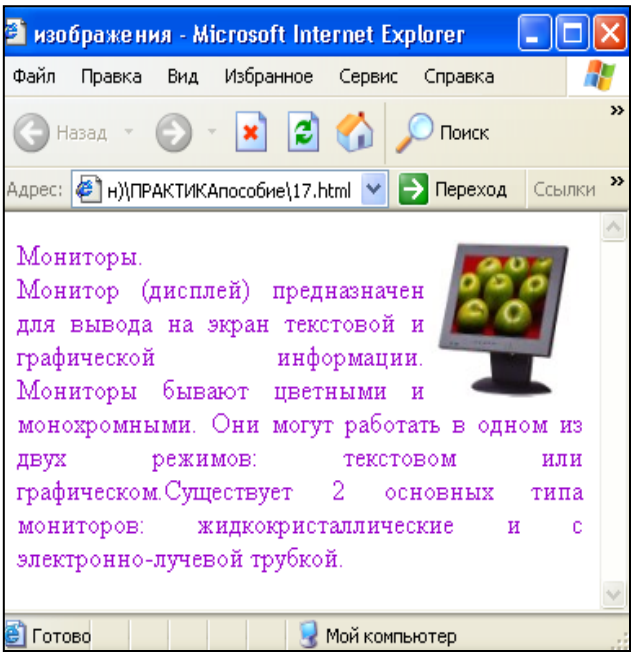


Рисунок 5.2. Создание

Текст, не поместившийся рядом с изображением, автоматически продолжается ниже.

Задание размеров выводимого изображения

WIDTH =n – ширина вставляемого изображения

HEIGHT=n - высота вставляемого изображения

Тэг встраивания изображений имеет два необязательных параметра, указывающих размеры изображения при отображении — WIDTH и HEIGHT. Значения параметров могут указываться как в пикселях, так и в процентах от размеров окна просмотра.

Синтаксис: **WIDTH =300** аналогично **WIDTH =50%**

HEIGHT=250 аналогично **HEIGHT=60%**

Значения параметров ширины и высоты изображения могут не совпадать с истинными размерами изображения. В этом случае браузеры автоматически при загрузке изображений выполняют его перемасштабирование. Заметим, что неаккуратное задание параметров может привести к изменению пропорций рисунка и, как следствие, к его искажению. Любой из этих параметров может быть опущен. Если задан только один из параметров, то при загрузке рисунка второй параметр будет вычисляться автоматически из условий сохранения пропорций.

ЗАДАНИЕ: *Добавьте изображения и задайте размеры выводимых изображений (см.рис.5.3). Действительные размеры отображаемого рисунка— 200x150. Этот рисунок отображен трижды в разном масштабе. Первое изображение (слева на право) вставлено без указания параметров размеров; у второго – ширина* ***width=150****, а его высота автоматически масштабируется; у третьего – ширина масштабируется автоматически, а высота* ***height=25%*** *(25% от размеров отображаемого браузера). У первого изображения добавлен*

атрибут выравнивания по нижней границе, для того, чтобы изображения отображались в одну строку (естественно, если позволяет ширина браузера).

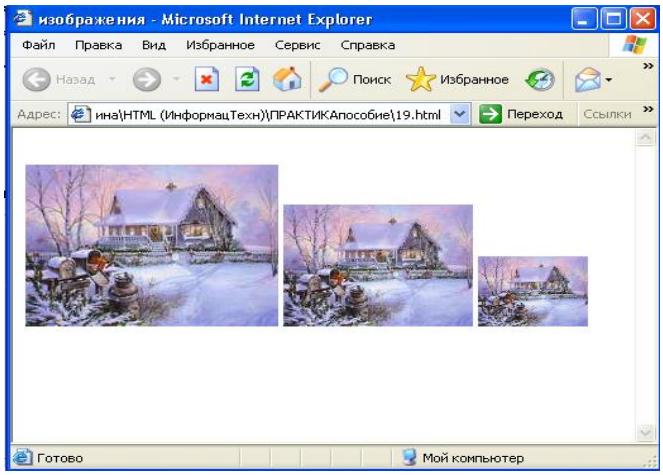


Рисунок 5.3. Указание

BORDER – задание рамки отображаемого изображения

Изображение, встраиваемое на страницу, можно поместить в рамку различной ширины. В качестве значения параметра используется число, означающее толщину рамки в пикселях. По умолчанию рамка вокруг изображений не рисуется. Исключением из этого правила является случай, когда изображение *является ссылкой*.

*Синтаксис: <IMG SRC=”1.gif” BORDER=**n**>*

ЗАДАНИЕ: *Задайте рамки отображаемых изображений (см.рис.5.4). Первое изображение (слева на право) вставлено без указания дополнительных параметров (а именно без рамок), у второго – атрибут **border=10**, третье изображение имеет рамку толщиной 50 пикселей.*

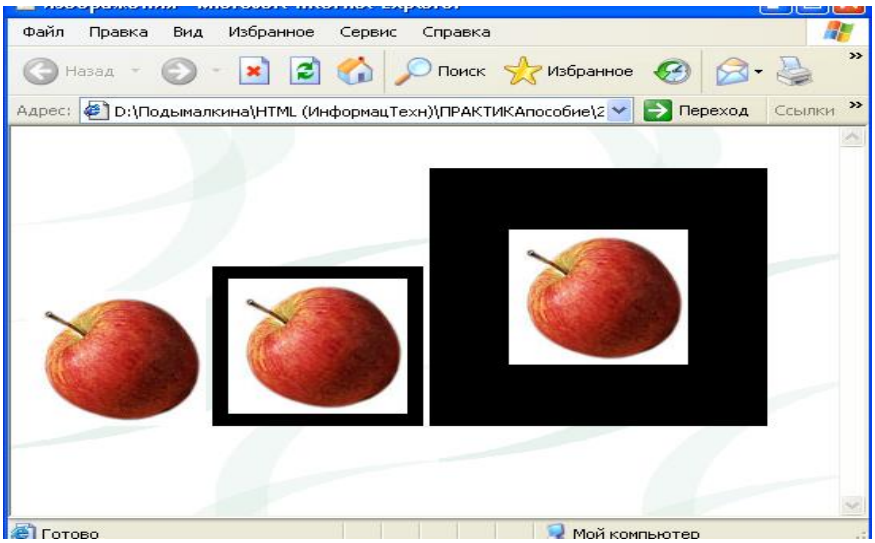


Рисунок 5.4. Задание рамок различных размеров

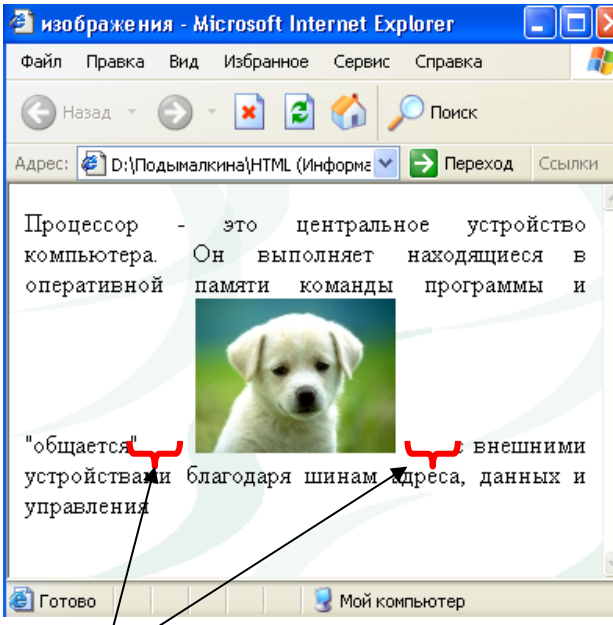
HSPACE – задание пустого пространства в пикселях справа и слева от рисунка

VSPACE – задание пустого пространства в пикселях сверху и снизу от рисунка

Для тэга <IMG> можно задавать параметры HSPACE и VSPACE, значения которых определяют отступы от изображения, оставляемые пустыми, соответственно, по горизонтали и вертикали. Это гарантирует, что между текстом и изображением останется пространство, необходимое для нормального восприятия.

*Синтаксис: <IMG SRC=”1.gif” HSPACE=**n** VSPACE =**n**>*

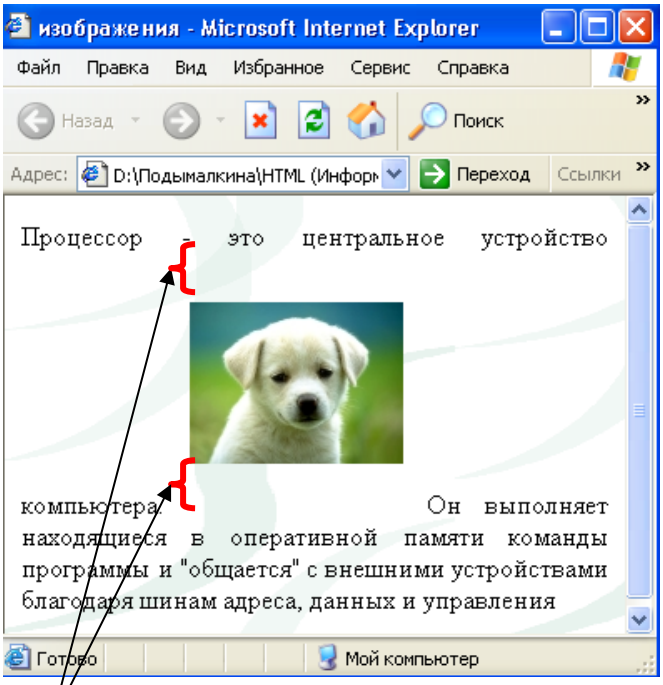
ЗАДАНИЕ: *Добавьте изображения с указанием отступов слева и справа от добавляемого изображения (см.рис.5.5). В данном документе в качестве фона используется графическое изображение, которое отображается в результате написания атрибута **Background** тела документа <BODY>.*



Отступы слева и справа от изображения заданы атрибутом **HSPACE**

Рисунок 5.5. Указание отступов от изображения слева(справа)

ЗАДАНИЕ: *Добавьте изображения с указанием отступов сверху и снизу от добавляемого изображения (см.рис.5.6). В данном документе в качестве фона используется графическое изображение, которое отображается в результате написания атрибута **Background** тела документа <BODY>.*



Отступы снизу и сверху от изображения

Рисунок 5.6. Указание отступов от изображения

ISMAP

Этот атрибут разрешает использовать изображение, отдельные части которого *связаны со ссылками* и позволяют выполнять переходы. Такие рисунки называют **MAP** *карта*, они используются совместно с элементом **<A>**. В этом случае реакцию на щелчок по карте отрабатывает сервер.

*Синтаксис: <A href=" **URL карты** "><IMG src=" **URL изображения** " ismap>*

USEMAP

Этот атрибут разрешает использовать изображение, отдельные части которого *связаны с картами*, они используются совместно с элементом **<MAP>**. В этом случае реакцию на клик по карте отрабатывает браузер.

*Синтаксис: <IMG src=" **URL изображения** " usemap= **"URL" карты(#имя карты)** >*

LOWSRC

Этот атрибут задает **URL** файла с низкокачественной копией основного изображения, которое может быть визуализировано до изображения, заданного атрибутом **src**.

*Синтаксис: <IMG lowsrc= **URL** >*

DYNSRC

Этот атрибут задает **URL** файла с видео клипом или сценой **VRML**.

*Синтаксис: <IMG dynsrc= **URL** >*

START

Этот атрибут используется совместно с атрибутом **DYNSRC**, он задает возможность управления процессом воспроизведения клипа, принимает значения:

FILEOPEN-сразу после загрузки;

MOUSEOVER-при наведении курсора.

*Синтаксис: <IMG dynsrc= **URL** start= **mouseover** >*

LOOP

Этот атрибут задает количество воспроизведений видео клипа. Он может принимать значения от **1** до **INFINITE** - *бесконечно*.

*Синтаксис: <IMG loop= **значение** >*

CONTROLS - добавление кнопок управления

Синтаксис написания кода документа, позволяющего отобразить видеофрагмент:

<IMG DYNSRC="URL видео файла" START= MOUSEOVER|FILEOPEN CONTROLS LOOP=n|INFINITE>

ПРОЕКТ №4 «Новости АВТОмира». Используя теги и соответствующие атрибуты, рассмотренные ранее, создайте документ (см.рис.5.7).

Цель: закрепить ЗУН при работе с изображениями, развивать профессиональные навыки используя язык html.

Рисунок 5.7. Итоговый документ проекта «Новости АВТОмира»

АКТИВНЫЕ ИЗОБРАЖЕНИЯ

Активные изображения (**image maps**), или изображения, чувствующие щелчки мыши, позволят вам создать на своем узле графические меню произвольной формы.

Как сделать активное изображение

Процесс создания активного изображения состоит из двух этапов. Сначала вы должны определить области на картинке, которые вы хотите сделать активными, а потом соотнести их со ссылками на другие URL. Активные области задаются перечислением их координат (в пикселях).

Элементы создания активных областей:

Элемент <MAP>

Элемент **<MAP>** применяется для представления графического изображения в виде ***карты с активными областями***. Он поддерживает атрибут **name**, который задает его имя, и включает внутри себя элемент **<AREA>**, который и задает собственно активные области карты, связанные ссылками с прочими ресурсами.

*Синтаксис: <MAP name=" **имя** "> <AREA **атрибуты** > </MAP>*

Элемент <AREA>

Элемент **<AREA>** задает активные области карты, щелчком по которым можно осуществить ссылку. Элемент не имеет конечного тэга. С изображениями карт удобно работать в стандартном приложении для **Windows** - редакторе **Paint** , для него изображение должно быть трансформировано в формат **BMP**. Используя сетку в режимах увеличения нажатием **Ctrl+G** и выбрав инструмент **Выделение**, когда указатель мыши принимает вид тонкого крестика, координаты курсора можно определить с точностью до одного пикселя. Такая точность может очень пригодиться при задании координат активных областей карты. Элемент **<AREA>** поддерживает различные **АТРИБУТЫ:**

href

Этот атрибут указывает **URL** ссылки.

*Синтаксис: <AREA href=" **URL** ">*

alt

Этот атрибут задает альтернативный текст для браузеров, которые не поддерживают данный элемент.

*Синтаксис: <AREA alt=" **текст подсказки** ">*

title

Этот атрибут задает альтернативный текст для браузеров, который всплывает при наведении курсора на данный элемент.

*Синтаксис: <AREA title=" **текст подсказки** ">*

shape

Этот атрибут задает форму активной области на карте и её координаты, он может принимать значения:

"circle" coords=X,Y,R, где **X,Y,R** - координаты центра круга и его радиус;

"poly" coords=X1,Y1,X2,Y2,X3,Y3..., где**X1,Y1,X2,Y2,X3,Y3...** - координаты вершин многоугольника;

"rect" coords=X1,Y1,X3,Y3 – если многоугольник-прямоугольник, то достаточно указать его **верхнюю левую** и **правую нижнюю** вершины.

*Синтаксис: <AREA " **circle** " coords= **X,Y,R** >*

ЗАДАНИЯ: *Создайте графическое изображение в виде* ***карты с активными областями*** *Изображение, отображенное в документе на рис.2.21 является обычная картинка с отображением холмов. Фигуры – треугольник, прямоугольник и овал, добавлены в графическом редакторе* **Paint** *элементом панели инструментов. Фигуры добавлены с той целью, что бы конкретно было видно, какие области являются активными (активные области появляются в результате задания координат нужных областей с помощью атрибута элемента <AREA>). Работа активных областей заключается в том, что по щелчку по одной из этих, указанных координатами областей, открывается определенный документ (адрес документа, который нужно открыть по нажатию на активную область, указывается с помощью атрибута HREF).*

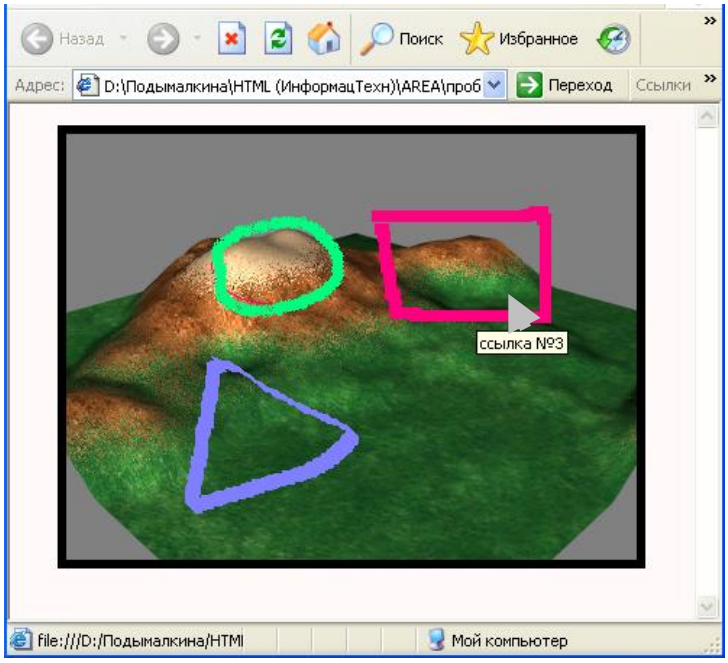


Рисунок 5.8.Созание карты с

Элемент <BANNER>

Позволяет зафиксировать какую-либо информацию на экране вне зависимости от того, какая часть окна просматривается в данный момент. Данный элемент также редко поддерживается браузерами, добиться же подобного решения можно при помощи фреймов или слоев.

*Синтаксис: <BANNER>**Текст или рисунок** </BANNER>*

Элемент <BGSOUND>

Используется для задания *фонового звучания* сразу же после загрузки страницы. Он может не иметь конечного тэга и может иметь атрибуты:

Src - указывает **URL** звукового файла в формате **WAV**, **AU** или **MIDI**.

loop - задает количество воспроизведений звукового файла. Он может принимать значения **1** или **infinite** (*бесконечно*).

Синтаксис: <BGSOUND src= ***URL*** *loop=* ***значение*** *>*

Элемент <MARQUEE> - используется с целью создания в документе бегущей строки.

Синтаксис:<MARQUEE **атрибуты**> **Текст строки ИЛИ картинка** </MARQUEE>

Он может иметь атрибуты:

bgcolor

Этот атрибут задает цвет фона бегущей строки. *Синтаксис: <MARQUEE bgcolor="**цвет**">*

height

Этот атрибут высоту бегущей строки в пикселях или процентах от всего окна. *Синтаксис: <MARQUEE height=**число**>*

align

Этот атрибут задает выравнивание бегущей строки по верхнему краю **top**, по середине **middle** или по нижнему краю **bottom**. *Синтаксис: <MARQUEE align= **...**>*

direction

Этот атрибут задает направление движения бегущей строки: **left** влево, **right** вправо, **up** вверх, **down** вниз. *Синтаксис: <MARQUEE direction="**...**">*

behavior

Этот атрибут задает поведение бегущей строки, либо **scroll** прокрутка, либо **slide** прокрутка с остановкой, либо **alternate** движение от края к краю. *Синтаксис: <MARQUEE behavior="**...**">*

hspace

Этот атрибут задает смещение в пикселах в право бегущей строки. *Синтаксис: <MARQUEE hspace=**число**>*

vspace

Этот атрибут задает пустое пространство выше и ниже бегущей строки. *Синтаксис: <MARQUEE vspace=**число**>*

loop

Этот атрибут задает количество проходов бегущей строки. *Синтаксис: <MARQUEE loop=**число**>*

scrollamount

Этот атрибут задает скорость движения бегущей строки, если его значение равно **1**, то она будет еле ползти, если его значение больше **10**, то она будет двигаться очень быстро. *Синтаксис: <MARQUEE scrollamount=**число**>*

scrolldelay

Этот атрибут задает временной интервал между шагами бегущей строки, с помощью него можно заставить строку двигаться рывками. *Синтаксис: <MARQUEE scrolldelay=**число**>*

ЗАДАНИЯ: *Создайте бегущую строку в документе с указанием значений атрибутов.*

Рисунок 5.9. Создания эффекта

Элемент <EMBED>

Применяется при внедрении в страницы мультимедиа содержимого и других файлов. В зависимости от возможностей браузера элемент обрабатывается либо браузером, либо в специально запущенном приложении. *Синтаксис:* <EMBED src="**file.swf**" menu=**true** quality=**high** bgcolor=**#000066** WIDTH=**760** HEIGHT=**410** TYPE="**application/x-shockwave-flash**" PLUGINSPAGE="**http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash**">

</EMBED>

Элемент **<EMBED>** может поддерживать следующие атрибуты:

Атрибут	Назначение
SRC=name.avi	Имя звукового, видео клипа.
WIDTH=X	Указывает ширину(X) вставляемого объекта в пикселах.
HEIGHT=Y	Указывает высоту (Y) вставляемого объекта в пикселах.
AUTOSTART=true	Если не указывать свойство AUTOSTART=true, то видеоролик не будет проигрываться автоматически при загрузке документа, а его можно будет воспроизвести с помощью кнопок управления. (TRUE | FALSE).
REPEAT=TRUE	Разрешает или запрещает повторять звуковой/видео клип (TRUE | FALSE).
LOOP	Задает количество повторений, принимает значения true или false
PLAY_LOOP=Z	Если Вы указали REPEAT=TRUE, укажите здесь вместо Z количество повторений.
HIDDEN	позволяет скрыть панель управления, принимает значения true или false
BGCOLOR	Задает фон вставляемого объекта
ALT	Задает альтернативное содержание
TYPE	Указывает на тип мультимедиа файла
QUALITY	Указывает на качество мультимедиа файла
PLUGINSPAGE	указывает на URL плагина для проигрывания мультимедиа файла

ПРОЕКТ №5 «В мире животных». Используя теги и соответствующие атрибуты, рассмотренные ранее, создайте документ (см.рис.5.10). В данном документе необходимо создать карту с активными областями. В созданном документе по нажатию на определенную область должен открываться новый документ. К примеру, по нажатию на синий треугольник (в котором отображается медведь), открывается документ, изображенный на рисунке 2.22.3 Аналогично и остальные активные области. Заголовок *«В мире животных»* должен отображаться в виде бегущей строки (при чем бегущая строка от края до края). Графическое изображение, вставляемое в документ изначально выглядит так, как изображено на рисунке 2.22.1. Затем все преобразования осуществляются в графическом редакторе **Paint.** Координаты, которые следует указывать при указании активных областей, следует смотреть в редакторе **Paint** при форматировании изображения (дорисовывании нужных фигур на картинке).

Цель: научится создавать картинки с активными областями.

Рисунок 5.10. Исходное

Бег
vnaя

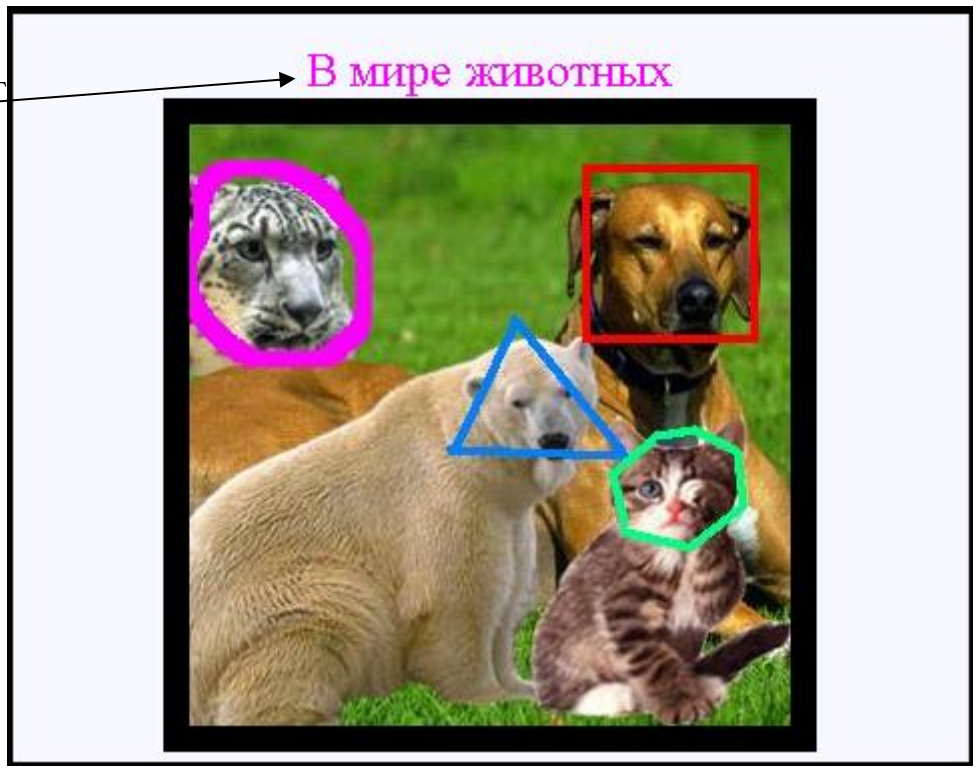


Рисунок 5.11. Итоговый документ с созданными

Рисунок 5.11. Пример
документа, на который указывает

Практическая работа. Гипертекстовые ссылки

Цель: познакомится с тегами для создания гипертекстовых ссылок на языке html.

Одним из важнейших понятий HTML-документов являются ***ссылки***. Само название — HTML, язык разметки гипертекста, указывает на принцип организации таких документов. Структура гипертекстовой сети задается гипертекстовыми ссылками. *Гипертекстовый* документ — это документ, содержащий ссылки на другие документы, позволяющие при помощи нажатия кнопки мыши быстро перемещаться от одного документа к другому. За основу гипертекста взят принцип организации энциклопедических словарей, которых во многих статьях есть ссылки на другие. Существует много типов мультимедийных объектов, которые могут быть размещены на Web-странице. В современных гипертекстовых документах в дополнение к самому тексту часто используют разнообразную графику, видео- и аудиообъекты, а в качестве ссылок часто применяют изображения.

<A>..</A>

Организация ссылок

Для записи гипертекстовой ссылки используется контейнер **<A...>.....</A>**, который называют **"якорь" (anchor).** Якорь имеет несколько атрибутов, главным из которых является ***HREF*** (HyperText Reference). Простую ссылку можно записать в виде:

<A HREF="http://polyn.net.kiae.su/index.html ">Индекс базы данных "Полынь" </A>

атрибут ***HREF***- обозначает гипертекстовую ссылку. Форма записи этого адреса называется ***универсальным локатором ресурсов URL (Universe Resource Locator).*** *Данная ссылка указывает на документ с именем "index.html" на машине(сервере) "polyn.net.kiae.su", доступ к которой осуществляется по протоколу HTTP.*

HTML использует URL для представления гипертекстовых ссылок и ссылок на сетевые сервисы внутри HTML-документа. Первая часть URL (до двоеточия) описывает метод доступа или сетевой сервис. Другая часть URL (после двоеточия) интерпретируется в зависимости от метода доступа.

Гипертекстовые ссылки делятся на 2 класса: - **контекстные** гиперт. ссылки

общие гиперт. ссылки

КОНТЕКСТНЫЕ ссылки вмонтированы в тело документа, в то время как **ОБЩИЕ** ссылки связаны со всем документов в целом и могут быть использованы при просмотре любого фрагмента.

<BASE> - связан с формой представления в гипертекстовой ссылке в форме URL.

HTML разрешает использовать как полную форму адреса, так и не полную.

Указателем ссылки может быть слово, группа слов или изображение. Внешний вид ссылки зависит от его типа, способов создания и установок

программы просмотра читателя. Указатели бывают двух типов — **текстовые и графические**. ***Текстовые*** указатели обычно представляют собой слово или несколько слов, выделенных на экране подчеркиванием. Цвет текстового указателя может регулироваться автором и установками программы просмотра. Приведем пример записи для текстового указателя ссылки:

Синтаксис:<A HREF="example.html">Этот текст является указателем ссылки</A>

В качестве ссылки можно использовать ***графическое изображение***. По принципу действия графические ссылки ничем не отличаются от текстовых. Они не подчеркиваются и не выделяются цветом, а для их выделения браузеры обычно вокруг такого изображения рисуют рамку. Пример графического указателя ссылки:

Синтаксис:<A HREF="example.html"><IMG SRC="picture.gif "></A>

Второй частью ссылки является URL-адрес. Это не что иное, как адрес Web-страницы, которая будет загружена при щелчке мышью на указателе. Указание адреса может быть относительным или абсолютным.

Замечание: Относительный указатель работает по-другому, если в HTML-документе используется тэг <BASE>. *Относительные* указатели удобны в использовании. Намного проще вставить только имя файла, а не весь длинный URL-адрес. Они также позволяют вам перемещать файлы в пределах вашего сервера без больших изменений в межстраничной адресации. URL-адрес, полностью определяющий компьютер, каталог и файл, называется *абсолютным*. В отличие от относительных, абсолютные указатели могут ссылаться на файлы, расположенные на других компьютерах.

Внутренние ссылки

Кроме ссылок на другие документы, часто бывает полезно включить ссылки на разные части текущего документа. Например, большой документ читается лучше, если он имеет оглавление со ссылками на соответствующие разделы. Для построения внутренней ссылки сначала нужно создать указатель, определяющий место назначения. Например, если вы хотите сделать ссылку на текст определенной главы документа, нужно разместить там указатель и дать ему имя при помощи параметра NAME тэга <A>. При этом параметр HREF не используется, и браузер не выделяет содержимое тэга <A>.

Синтаксис: <A NAME= chapter _5> </A>

Обратите внимание, что в приведенном примере отсутствует содержимое тэга <A>. Обычно именно так и делают, поскольку здесь нет необходимости как-то выделять текст, а требуется лишь указать местоположение. После того как место назначения определено, можно приступить к созданию ссылки на него. Для этого, вместо указания в параметре HREF адреса документа, как это делалось ранее, поместим туда имя ссылки с префиксом #, говорящим о том, что это внутренняя ссылка.

<A HREF="#chapter_5">Глава 5</A>

Теперь, если пользователь щелкнет кнопкой мыши на словах "глава 5", браузер выведет соответствующую часть документа в окне просмотра.

Ссылки на документы различных типов

Когда пользователь щелкает мышью на ссылке, указывающей на другую Web-страницу, она выводится непосредственно в окне браузера. Если же ссылка указывает на документ иного типа, программа просмотра принимает документ и затем решает, что с ним делать дальше. Следующими действиями браузера могут быть:

Браузер знает этот тип документа и умеет с ним обращаться. Например, если вы создали ссылку на графический файл формата GIF, и пользователь щелкнул мышью на этой ссылке, его программа просмотра очистит окно и загрузит указанное изображение. В некоторых случаях браузер может дополнительно использовать подключаемый программный модуль (plug-in), без которого задача не была бы решена.

Браузер не распознает тип принятого документа и не знает, что с ним делать дальше. В этом случае он обратится к вспомогательным программам, имеющимся на машине пользователя. Если подходящая программа найдется, браузер запустит ее и передаст ей полученный документ для обработки. Например, если пользователь щелкнет на ссылке на видеофайл формата AVI, браузер загрузит файл, найдет программу для демонстрации AVI-файлов и запустит ее. Видеофайл будет показан в дополнительном небольшом окне.

ЗАДАНИЕ: *Создайте гипертекстовые ссылки по тексту, изображению; создание якоря или внутреннее ссылки.*

Создание ссылки по тексту:

<A HREF=”Document1.html”>*Открыть другой документ***</A>**

Создание ссылки по изображению:

<A HREF=”Page.html”> *<IMG src=”kart.jpg”>* **</A>**

Создание якоря:

<A NAME=”#CHAR”>Работа со ссылками</A> - создание якоря с именем **CHAR**

<A HREF=”1.HTML#CHAR”>возврат в начало</A> - обращение к якорю **CHAR** в документе, приведенном в примере

ПРОЕКТ №6 «Создание ссылок в галереи звуков и клипов.». Используя элементы вставки видео-, звуковых файлов создайте документ, который бы отображал и проигрывал видеоклипы автоматически. В этом же документе необходимо создать гипертекстовую ссылку, осуществляющую загрузку звукового файла. Основные элементы вставки видео-, звуковых файлов: Embed, Img, BGsound с соответствующими атрибутами.

Первая ссылка *на звуковой файл* должна быть **по тексту**.

Вторая ссылка *на другой документ* (любой ранее созданный) должна быть **по изображению (картинке).** В открывшемся документе необходимо создать ссылку, позволяющую вернуться назад на предыдущий документ, из которого было вызвано открытие активного в данный момент документа.

Цель: продолжать развивать профессиональные навыки при работе с разными видами ссылок.

2.7. Занятие 7. Таблицы

Цель: познакомится с тегами и атрибутами для создания таблиц.

Элемент <TABLE>

Используется с целью внедрения таблиц в Web страницу. Они удобны тем, что браузер сам прорисовывает рамку таблицы. Кроме того таблицы позволяют решать чисто дизайнерские задачи: выравнивать части таблицы друг относительно друга, размещать рядом рисунки и текст, управлять цветовым оформлением, разбивать текст на колонки и т.д.

<CAPTION>…</CAPTION>

Таблица может иметь заголовок, хотя заголовок не является обязательным. Метка <CAPTION> может включать атрибут ***ALIGN***. Допустимые значения: (<CAPTION ALIGN=BOTTOM>Название таблицы</CAPTION>)

ALIGN=TOP - заголовок помещается над таблицей

ALIGN=BOTTOM - заголовок помещается под таблицей.

<TR>…</TR> - строки таблицы (*table rows*)

<TD>…</TD> - ячейки таблицы (*table data*)

<TH>…</TH> - заголовок для столбцов и строк таблицы *(table header)*.

ЗАДАНИЕ: *Создание таблицы размерностью 2 строки*2 столбца (см.рис.7.1).*

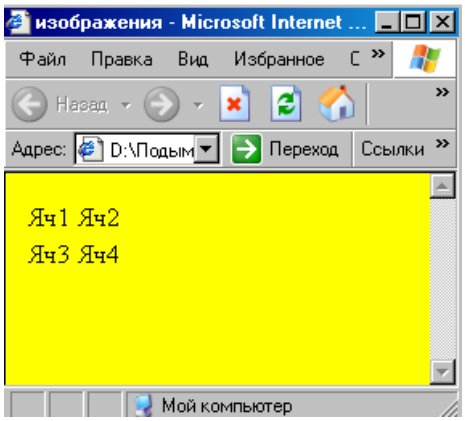
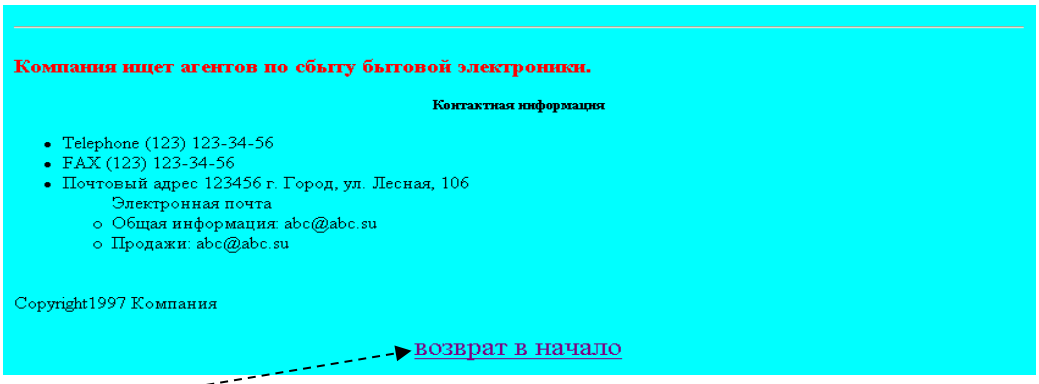


Рисунок 7.1.
Создание простейшей

Ссылка на «Документ№1»

Ссылка на «Документ№2»



Использование ЯКОРЯ

Рисунок 6.1. Создание

Элемент <TABLE> может иметь АТРИБУТЫ:

border

Этот атрибут задает ширину внешнего обрамления (рамки) таблицы в пикселях.

<TABLE border=число >

ЗАДАНИЕ: *Создание таблицы размерностью 2 строки*2 столбца с указанием рамок (границ) таблицы (см.рис.7.2).*

Код документа изображенного на рис. 7.2:

```
<html>
<head>
<title>изображения</title>
</head>
<body bgcolor="yellow"><p align=justify>
<TABLE BORDER=3>
<tr><td>Ячейка1</td><td>Ячейка2</td></tr
<tr><td>Ячейка3</td><td>Ячейка4</td></tr
</table>
</body></html>
```

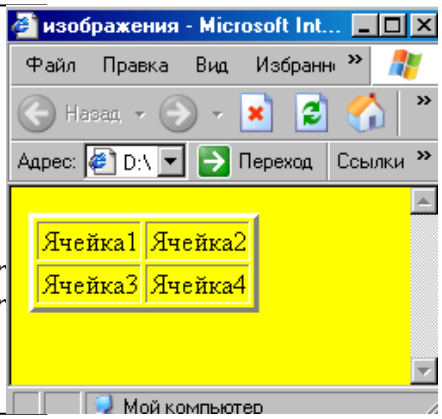


Рисунок 7.2.Создание таблицы с

bordercolor

Этот атрибут задает цвет рамки, используется только с атрибутом border. *<TABLE border=число bordercolor=цвет >*

bordercolorlight

Этот атрибут задает цвет рамки (левой и верхней рамки), используется только с атрибутом border. *<TABLE border=число bordercolorlight=цвет>*

bordercolordark

Этот атрибут задает цвет рамки (правой и нижней рамки), используется только с атрибутом border. *<TABLE border=число bordercolordark=цвет>*

ЗАДАНИЕ: *Создание рамок таблиц различными способами и задание их цвета (см.рис.7.3)..*

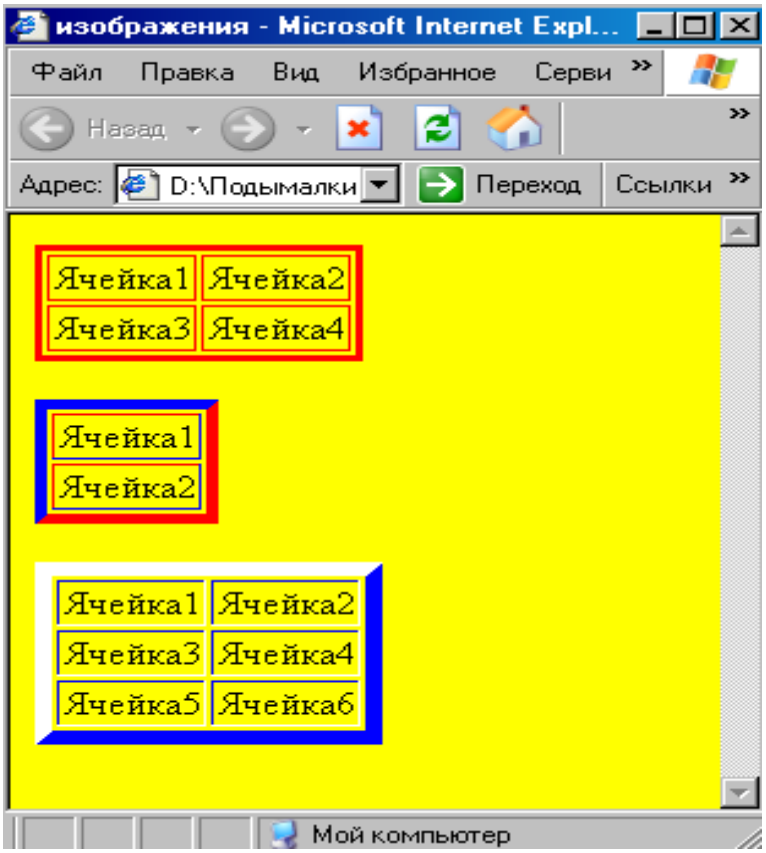


Рисунок 7.3.Создание рамок таблицы дополнительными способами

bgcolor

Этот атрибут задает фоновый цвет ячейкам. *<TABLE bgcolor=цвет>*

background

Этот атрибут задает фоновый рисунок ячейкам. *<TABLE background=URL файла изображения>*

Замечание: Цвет фона или фон можно задавать как у всей таблицы, так и отдельно у строк или ячеек таблицы.

ЗАДАНИЕ: Создайте таблицу, *закрашивая фон ячеек таблицы различными способами и задание их цвета (см.рис.7.4).*

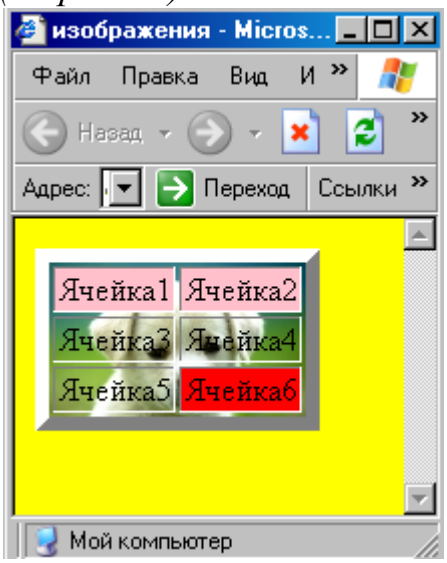


Рисунок 7.4.Способы

width

Этот атрибут задает ширину таблицы в пикселях или процентах от всего окна. *<TABLE width=число или %>*

height

Этот атрибут задает высоту таблицы в пикселях или процентах от всего окна.

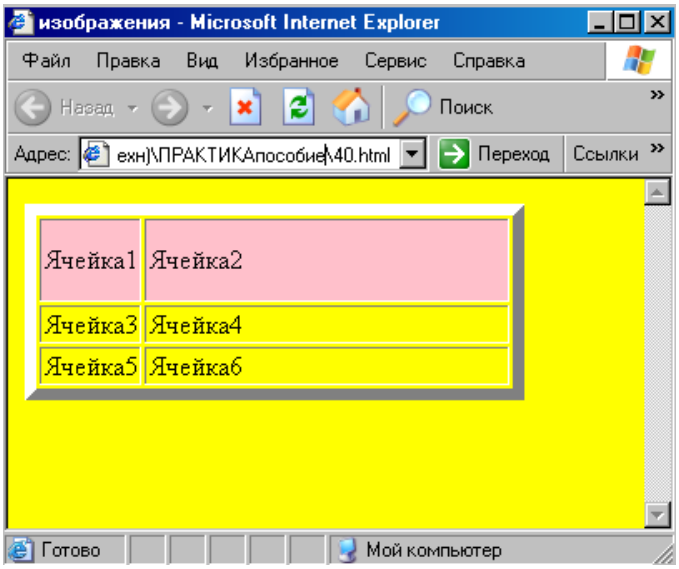


Рисунок 7.5.Создание таблицы с указанием ширины и высоты ячеек

ЗАДАНИЕ: *Создайте таблицу с указанием ширины и высоты таблицы, ячеек таблицы в отдельности (см.рис.7.5).*

align

Этот атрибут задает режим горизонтального выравнивания таблицы на странице, он может принимать значения left, center и right. *<TABLE align= способ >*

valign

Устанавливает вертикальное выравнивание текста в ячейке. Допустимые значения:

VALIGN=TOP - выравнивание по верхнему краю;

VALIGN=MIDDLE - выравнивание по центру;

VALIGN=BOTTOM - выравнивание по нижнему краю.

ЗАДАНИЕ: Создайте таблицу применяя выравнивание *таблицы и содержимого ячеек таблицы (см.рис.7.6).*

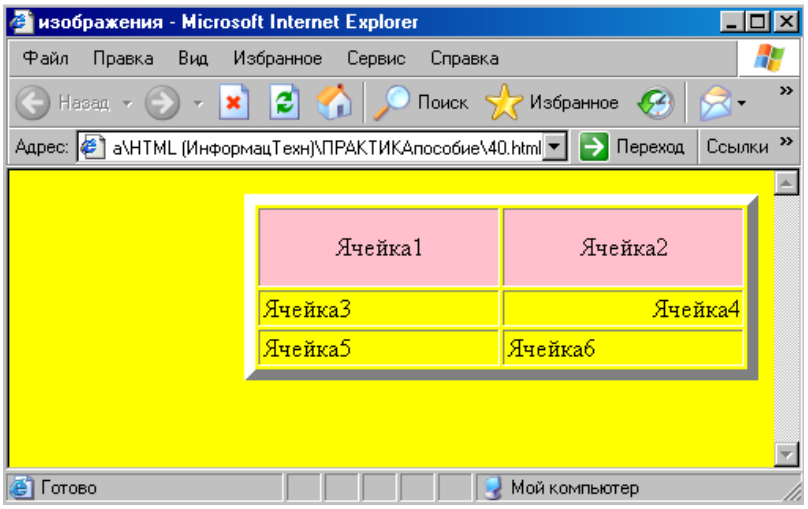


Рисунок 7.6. Использование атрибутов выравнивания содержимого таблицы

cellspacing

Этот атрибут устанавливает расстояние между рамками ячеек таблицы в пикселах.

<TABLE cellspacing=число > (например, CELLSPACING=2).

cellpadding

Этот атрибут задает отступ между содержимым ячейки и обрамлением таблицы в пикселях.

<TABLE cellpadding=число >

ЗАДАНИЕ: *Использование атрибута, задающего отступы в таблице (см.рис.7.7).*

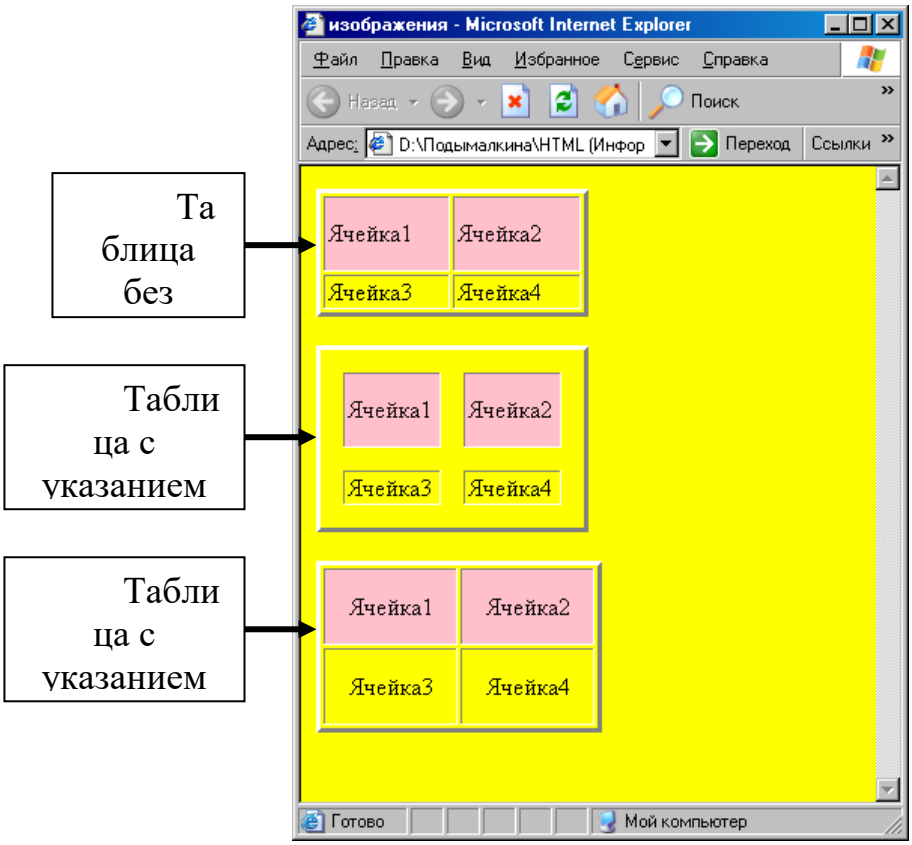


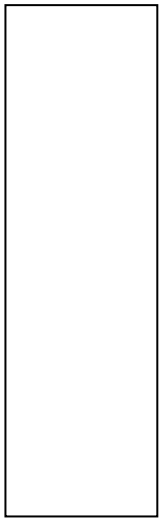
Рисунок 7.7. Задание

rowspan (*Row Span, соединение строк)*

Указывает какое количество ячеек будет объединено по вертикали для указанной ячейки. Если вы указали в атрибуте ROWSPAN= число, большее единицы, то соответствующее количество строк должно находиться под растягиваемой ячейкой.

<TD rowspan=количество строк >

ЗАДАНИЕ: *Создайте таблицу, применяя знания по объединению ячеек по вертикали (см.рис.7.8).*



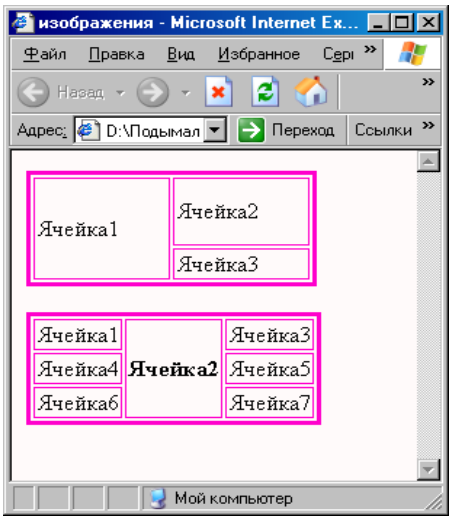


Рисунок 7.8.
Объединение ячеек таблицы

colspan *(Column Span, соединение столбцов)*

Указывает какое количество ячеек будет объединено по горизонтали для указанной ячейки. Таги <TD> и <TH> модифицируются с помощью атрибута COLSPAN= (Column Span, соединение столбцов).

<TD colspan=количество столбцов >

ЗАДАНИЕ: *Создайте таблицу, применяя знания по объединению ячеек столбцов(по горизонтали) (см.рис.7.9).*

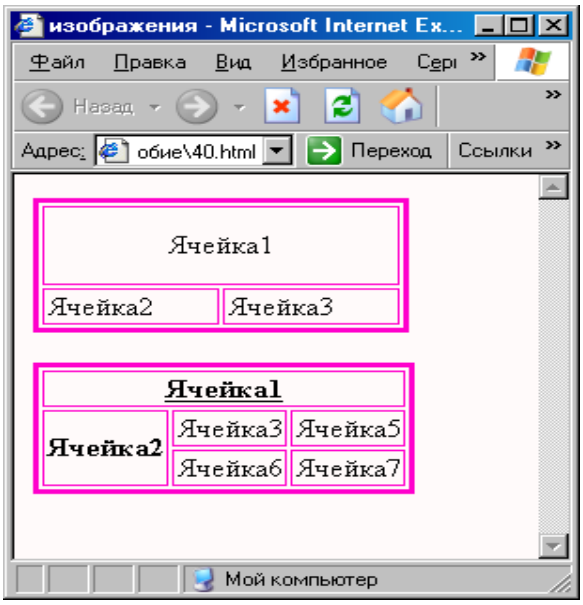


Рисунок 7.9. Объединение

ЗАДАНИЕ: *Создайте таблицу, применяя знания по объединению строк и столбцов (см. рис.7.10)*

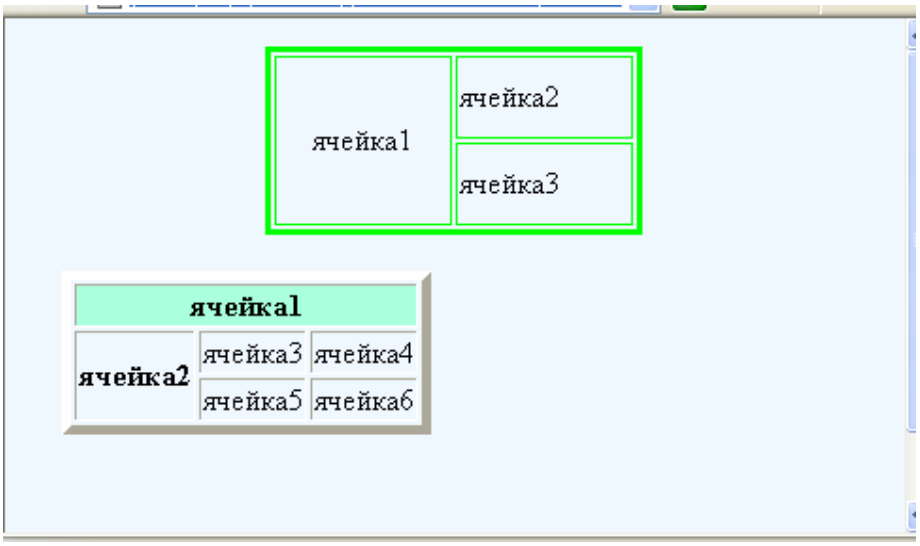


Рисунок 7.10.Пример объединения ячеек

RULES (дополнительные возможности оформления таблиц) Этот атрибут управляет отображением **внутренних границ** таблицы. *<TABLE rules=all >*

none - нет линий, значение используется по умолчанию.

groups - линии отображаются только между группами строк THEAD, TFOOT, и TBODY или группами столбцов COLGROUP и COL.

rows - линии отображаются только между строками.

cols - линии отображаются только между столбцами.

all - линии отображаются между строками и столбцами.

ЗАДАНИЕ: *Создание внутренних границ таблицы (см.рис.7.11).*

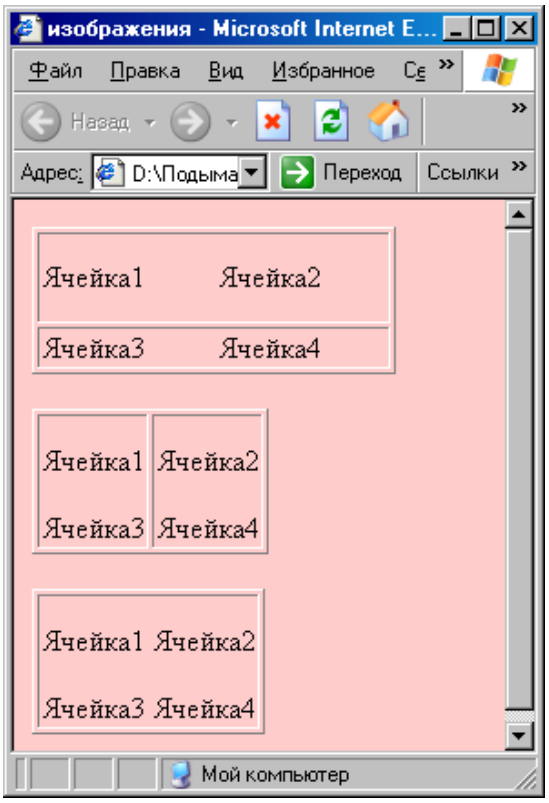


Рисунок 7.11.
Создание внутренних

FRAME (дополнительные возможности оформления таблиц)

Этот атрибут указывает отображение только **внешней рамки**. *<TABLE frame=void>*

Возможные значения:

Void - сторон нет. Это значение используется по умолчанию.

Above - только верхняя часть.

Below - только нижняя часть.

Hsides - только верхняя и нижняя часть.

Vsides - только левая и правая части.

Lhs - только левая часть.

Rhs - только права часть.

Box - все четыре части.

Border - все четыре части.

ПРОЕКТ №7 «Морской пейзаж». Используя ранее рассмотренные элементы, создайте документ, как изображено на рисунке 7.12. Основная таблица в документе состоит из двух частей. Первая часть таблицы – это таблица, при создании которой используются атрибуты объединения ячеек. Вторая таблица – создается с использованием атрибутов из категории «**дополнительные возможности оформления таблиц**». Надписи создаются с использованием элемента таблицы *<Caption>*. В данном документе в качестве фона таблицы и самого документа используются графические изображения.

Цель: закрепить знания по работе с таблицами.

Таблица1

Краткие сведения об океанах

ОКЕАН	Площадь-млн.кв.км	Глубина-м		Места. где океан достигает максимальной глубины
		средняя	максим.	
Атлантический	92	3600	8742	Пуэрто-Рико,желоб
Индийский	76	3700	7709	Зондийский желоб
Северный Ледовитый	15	1220	5527	Гренландское море
Тихий	179	4000	11022	Марианский желоб

Таблица №1

Рисунок 7.12. Итоговый документ проекта «Морской пейзаж»

ПРАКТИЧЕСКАЯ РАБОТА. СПЕЦИАЛЬНЫЕ СИМВОЛЫ

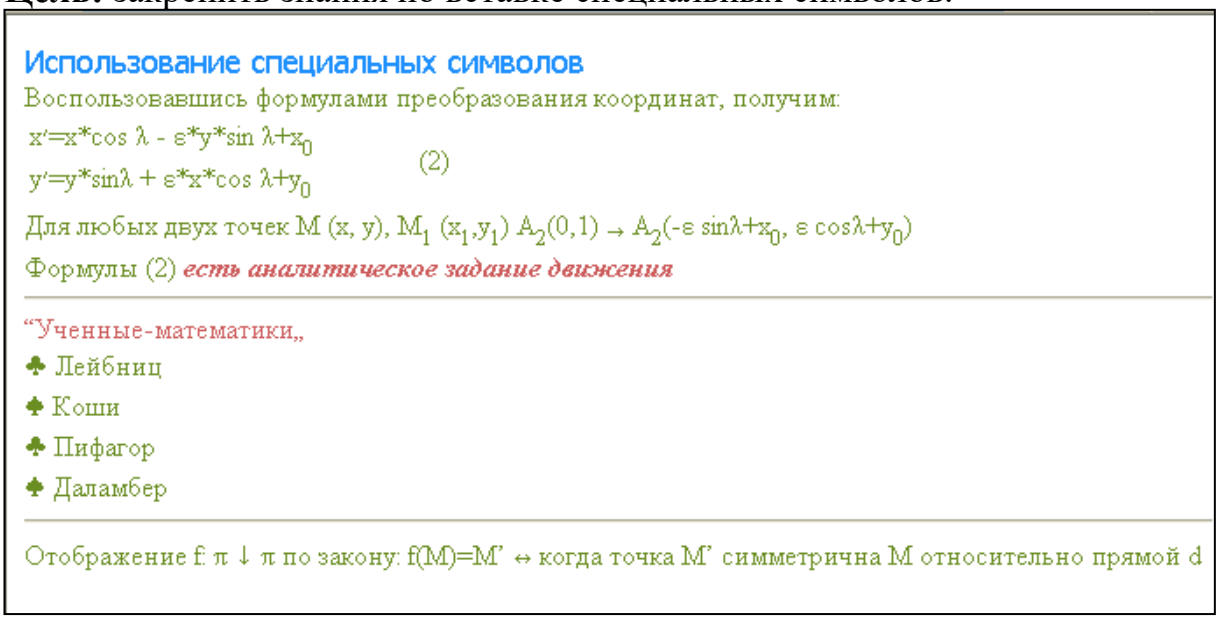
Цель: познакомиться с тегами для возможности вставки специальных символов.

Различные символы можно вводить с помощью специальных кодов. Коды начинаются с символа "амперсанд"(&). За ним следует название символа либо его числовой код в десятичной или шестнадцатеричной системе. Завершает код символ "точка с запятой"(;).

***Синтаксис*:** *Написание строки:* x^2+sinλ >tgψ *осуществляется следующим образом:* **x&#178 + sin &#955 &#62 tg &#968**

ПРОЕКТ №8 «Математические формулы». Используя таблицу специальных символов создайте документ (см.рис.8.1).

Цель: закрепить знания по вставке специальных символов.



Использование специальных символов

Воспользовавшись формулами преобразования координат, получим:

$x'=x*\cos\lambda - \varepsilon*y*\sin\lambda+x_0$

$y'=y*\sin\lambda + \varepsilon*x*\cos\lambda+y_0$ (2)

Для любых двух точек M (x, y), M_1 (x_1,y_1) $A_2(0,1) \rightarrow A_2(-\varepsilon \sin\lambda+x_0, \varepsilon \cos\lambda+y_0)$

Формулы (2) *есть аналитическое задание движения*

"Ученные-математики„

- Лейбниц
- Коши
- Пифагор
- Даламбер

Отображение f: π ↓ π по закону: f(M)=M' ↔ когда точка M' симметрична M относительно прямой d

Рисунок 8.1. Итоговый документ проекта

Практическая работа. Фреймы.

Цель: познакомится с тегами для работы с фреймами.

В каком-то смысле фрейм - это именно то, что означает данное слово: рамка вокруг картинки, окошко или страница. На экране фреймы представляют собой прямоугольники.

При использовании фреймов, окно браузера разбивается на несколько подокон, в каждом из которых можно отображать любые Web–документы, осуществлять их прокрутку независимо от других окон.

Использование <FRAME> позволяет помещать в окна одной страницы несколько отдельных страниц, произвольно менять их размеры и организовывать изменение содержимого одного окна после выполнения пользователем действий в другом окне. Это позволяет использовать их в качестве инструмента навигации. Тэги <FRAMESET> и </FRAMESET> в данном случае заменяют тэги <BODY> и </BODY> соответственно. Внутри пары <FRAMESET> и </FRAMESET> могут быть использованы только тэги <FRAME>, <FRAMESET> и <NOFRAMES>. По сути, создается несколько отдельных страниц фреймов, которые выводятся на экран одновременно в виде нескольких окошек одного документа. Другими словами, прежде чем организовывать отдельные документы в виде фреймов одного документа, необходимо прежде создать сами документы. Для пользователей, чей браузер не поддерживает фреймы, необходимо создать альтернативный документ или сообщение о том, каким браузером нужно воспользоваться для просмотра данного документа и заключить его в тэги <NOFRAME> и </NOFRAME>

Элемент <FRAMESET>…</FRAMESET>

Использование элемента <FRAMESET> позволяет задать размеры фреймов, на которые делится экран и их состав (экран можно разделить на несколько горизонтальных или вертикальных фреймов).

Атрибуты элемента <FRAMESET>:

BORDER - задает толщину обрамления в пикселях для всех производных фреймов.

FRAMEBORDER - задает наличие или отсутствие обрамления у фреймов. Значение 1 соответствует наличию, а 0 - отсутствию обрамления.

COLS = “value” - определяет количество и размер *вертикальных подокон* в создаваемом наборе кадров. Значения ширины различных кадров перечисляются в кавычках и через запятую.

ROWS = “value” - определяет количество и размер *горизонтальных подокон* в создаваемом наборе кадров. Значения высоты различных кадров перечисляются в кавычках и через запятую.
Синтаксис: <FRAMESET COLS="25%,75%" ROWS="20%,80%">

Синтаксис используемых видов описания величин подокон:

value -числовое значение определяет высоту(ширину) в пикселях *(это неудобно, т.к. различные браузеры имеют различный размер рабочего поля)*

value% -значение величины подокна в процентах от 1 до 100%

value -данный символ говорит о том, что все оставшееся место будет принадлежать данному фрейму*

Например, команда: <FRAMESET ROWS= «100,»> создает 2 кадра: верхний составляет 100 пикселей в высоту, нижний - оставшееся пространство.*

Элемент <FRAME>

Элемент <FRAME> определяет содержимое заданных фреймов.

Атрибуты элемента <FRAME>:

SRC=”url” - задает имя документа, который будет отображен внутри фрейма

FRAMEBORDER - задает наличие или отсутствие обрамления у фреймов. Может принимать значения 1или 0. Значение 1 соответствует наличию, а 0 - отсутствию обрамления.

MARGINWIDTH - аналогично атрибуту *CELLPADDING*. Он задает величину разделительных полос между фреймами сбоку. Наименьшее значение этого атрибута =1. Нельзя указывать 0(ноль), можно просто ничего не присвоить.

MARGINHEIGHT - действует так же, как и *MARGINWIDTH*. Задает поля в верхней и нижней части фрейма.

NORESIZE - лишает пользователя возможности изменить размеры текущего и смежного фреймов с помощью мыши. *Синтаксис: <FRAME noresize >*

SCROLLING - задает наличие у фреймов полос прокрутки, принимает значения *YES, NO* и *AUTO*.

NAME - задает имя для определенного фрейма, по которому можно будет обращаться к нему с помощью атрибута *TARGET* в ссылках <A href>.

TARGET - Когда пользователь щелкает мышью на одной из ссылок в одном (к примеру в левом) фрейме. Соответствующая страница должна появиться в другом (к примеру в правом) фрейме, а оглавление остается неизменным. Чтобы добиться этого, нужно определить целевой фрейм *TARGET*, в котором будет отображаться страница для каждого пункта оглавления. Задание целевых фреймов осуществляется в ссылках первого (левого) фрейма. Поэтому всем кадрам во фреймовой структуре были присвоены имена. Другой (правый) фрейм называли *name=“main”*, так что нужно в каждой ссылке добавить атрибут *TARGET=”main”*, в итоге чего соответствующая страница появится во фрейме *main*. Т.е. он позволяет определить, к какому фрейму относится та или иная операция.

«Волшебные» целевые фреймы.

Предназначены для особо исключительных случаев. Их имена всегда начинаются с символа подчеркивания (_). Волшебные фреймы не обязательно указывать во фреймовой структуре.

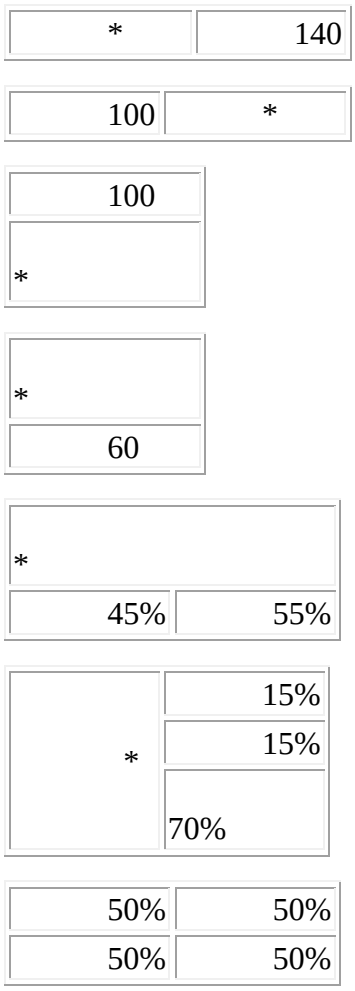
Target=”_BLANK” – если атрибут TARGET ссылается на blank, то документ всегда будет появляться в новом пустом окне.

“_SELF” –указывает на то, что выбранная страница загружается в тот же фрейм, где была активизирована ссылка

“_PARENT” – документы, вызываемые по этой ссылке, появляются в родительской фреймовой структуре.

“_TOP” – при указании данной ссылки, документы появятся в отдельном окне вне его. Независимо от вашего желания браузер откроет новое окно просмотра.

ЗАДАНИЕ: Создайте фреймы по образцу



1

2

3

4

5

6

7

ЗАДАНИЕ: *Создайте вертикальные фреймы.*

Необходимо создать фрейм с двумя вертикальными окнами (см.рис.10.1), в каждом из которых следует отобразить ранее созданные документы. В данном случае в первом левом окне следует разместить документ №1(рис.5.8), во втором правом окне – документ №2 (см.рис.5.4). В процессе создания документа правой части фрейма нужно дать имя, чтобы в последующем осуществлять ссылку на него. Необходимо, чтобы по нажатию на фигуры документа, размещенного в левой части фрейма, открывающиеся документы открывались именно в правой части фрейма (см.рис.10.2). По нажатию на текст «ВОССТАНОВЛЕНИЕ ТИТУЛЬНОГО ЛИСТА» необходимо чтобы в правой части фрейма отобразился изначальный документ (см.рис.5.4).

Для осуществления такого отображения документов следует указать соответствующее значение атрибута TARGET. При создании документа, размещенного в левой части фрейма, следует указать атрибут TARGET.

Остальные документы следует создать самостоятельно.

Рисунок 10.1. Создание вертикальных фреймов

Создание горизонтальных фреймов осуществляется аналогичным образом.

По нажатию на фигуру **«круг»** в правой части фрейма открывается документ.

Рисунок 10.2.Открытие документа - как результат работы гиперссылки

Элемент <IFRAME>.

Разместить фрейм в обычном HTML-документе (в пределах элемента <BODY>) можно с помощью элемента <IFRAME>. Фреймы, создаваемые этими элементами, называются *ПЛАВАЮЩИМИ*.

Имя элемента плавающего фрейма <IFRAME> происходит от сокращения английского термина «*inline frame*» - строчный фрейм. Контейнер <IFRAME> может размещаться в любом месте элемента тела документа <BODY>.

SRC = “URL” - задает имя документа, который будет отображен внутри плавающего фрейма.

FRAMEBORDER - задает наличие или отсутствие обрамления у фреймов. Может принимать значения 1 или 0. Значение 1 соответствует *наличию*, а 0 - *отсутствию* обрамления.

MARGINHEIGHT - задает толщину верхнего и нижнего обрамления в пикселях.

MARGINWIDTH - задает толщину правого и левого обрамления в пикселях.

NAME - задает имя фрейма, по которому можно будет обращаться к нему в ссылках и направлять в него содержимое.

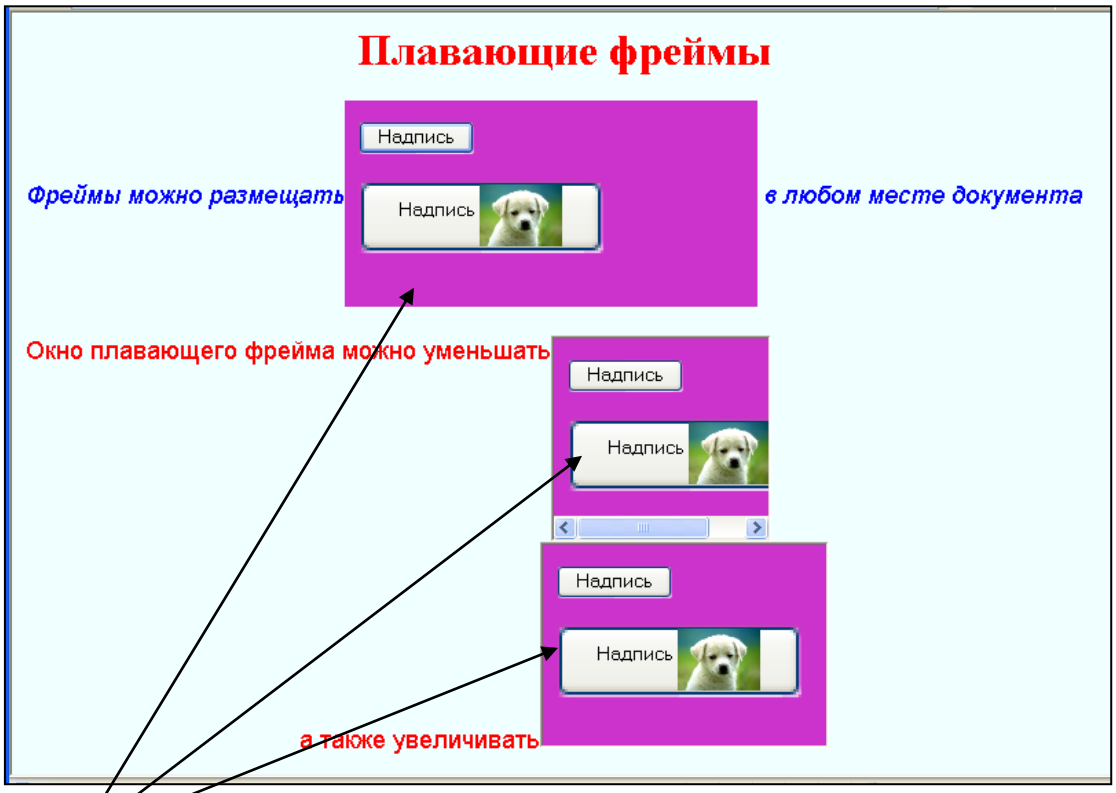
ALIGN - позволяет позиционировать кадр по отношению к тексту, принимает значения: *left, right, center, middle, top, bottom*.

SCROLLING - задает наличие у кадра полос прокрутки, принимает значения YES, NO и AUTO.

WIDTH - определяет ширину кадра в пикселях.

HEIGHT - определяет высоту кадра в пикселях.

ЗАДАНИЕ: *Создайте документ с плавающими фреймами (см.рис.10.3).*



Плавающие

Рисунок 10.3.Создание плавающих фреймов в документе

Примеры простейших макетов страниц с фреймами.

Горизонтальное деление экрана производится при помощи атрибута *ROWS*, а вертикальное при помощи атрибута *COLS*. Значения атрибутов могут быть выражены в пикселях, процентах или * для обозначения оставшейся части экрана.

Например:

cols=50%,50%

Деление области просмотра на равные, правую и левую, части.

cols=20%,80%

Деление области просмотра на неравные, правую и левую, части.

*rows=100,20%,**

Деление области просмотра на три части: первой отведено 100 пикселей, второй - 20% доступного пространства, а третьей - все остальное.

cols=,3**

Деление области просмотра на неравные, правую и левую, части. Правая часть в три раза шире левой.

cols=25%,75%

Деление области просмотра аналогично предыдущему примеру.

ЗАДАНИЕ: *Создайте фреймы более сложной структуры (см.рис.10.4). Документы, отображенные в окнах фреймовой структуры, необходимо создать самостоятельно. При создании фреймов следует запретить возможность изменения размеров фреймовых окон.*

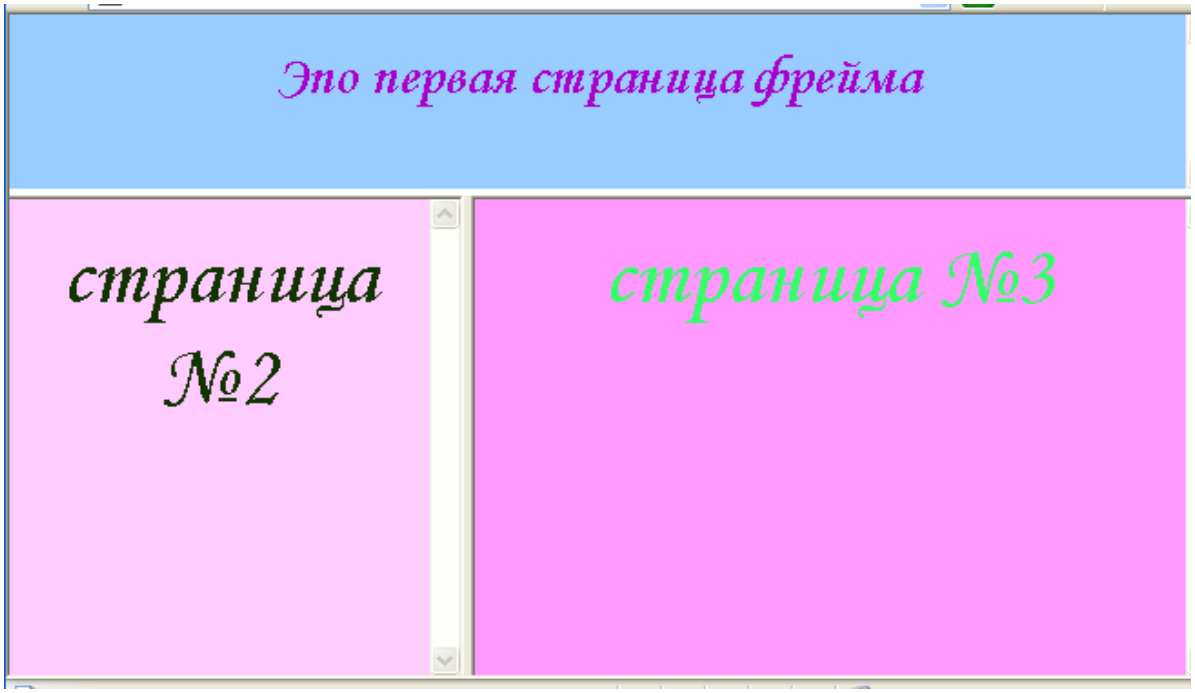


Рисунок 10.4. Создание комплексной

.

ПРОЕКТ №10 Пробный учебник «Кратко о HTML».

Цель проекта: закрепить ЗУН по работе с фреймами

Используя ранее рассмотренные элементы и их атрибуты, создайте небольшой электронный учебник по HTML. Выполнение такового задания заключается в следующем:

изначально следует создать документ, содержащий фреймовую структуру (см.рис.10.5);

далее необходимо создать три отдельных документа, которые будут отображаться в подокнах фреймов (а именно: документ №1-документ отображающийся в верхней части фрейма; документ №2 – в левой нижней части фреймов; документ №3 – в правой нижней части фрейма);

в документе №2 следует создать гипертекстовые ссылки на соответствующие документы, которые необходимо будет создать;

далее, необходимо создать еще 4 документа, содержание которых должно соответствовать оглавлению документа №2 (а именно: документ №4 – должен содержать данные о форматировании текста; документ №5 – о возможностях работы с картинками; документ №6 – о создании таблиц; документ №7 – о гипертекстовых ссылках). Каждый документ должен содержат не только теорию, но и практические примеры;

при создании ссылок на соответствующие документы, следует учесть следующее:

по нажатию на элементы оглавления документа №2 *«форматирование текста»* и *«создание таблиц»* - необходимо чтобы открывающиеся документы появлялись в правом окне нижней части фрейма;

документы элементов *«работа с изображениями»* и *«гипертекстовые ссылки»* - в новом пустом окне;

по нажатию на *картинку* (в данном случае компьютер) необходимо чтобы в окне правой нижней части фрейма отобразилось изображение, отображающееся изначально, т.е. документ №3.

Оформление недостающих документов осуществлять самостоятельно в произвольном виде.

Рисунок 10.5. Итоговый документ проекта

Практическая работа. Формы.

Форма — это инструмент, с помощью которого HTML-документ может отправить информацию в заранее определенную точку внешнего мира. Формы применяются для опроса посетителей, покупки чего-либо, отправки электронной почты.

Принцип работы форм следующий: посетитель, зашедший к вам на страничку заполняет форму, а после нажатия определенной кнопки форма берет данные из заполненных полей и отправляет их в назначенное место.

Формы размещаются между тэгами <FORM></FORM>. HTML-документ может содержать в себе несколько форм, но они не должны находиться одна внутри другой. Тэг <FORM> может содержать следующие атрибуты:

ACTION	Обязательный атрибут. Определяет, где находится обработчик формы.
METHOD	Определяет, каким образом данные из формы будут переданы обработчику. Допустимые значения: METHOD=POST и METHOD=GET. Если значение атрибута не установлено, по умолчанию предполагается METHOD=GET.
ENCTYPE	Определяет, каким образом данные из формы будут закодированы для передачи обработчику.

Для внесения информации пользователем в форму используется элемент <INPUT> Это и есть поля, в которые пользователь вводит информацию. Каждый элемент <INPUT> включает атрибут NAME=имя, определяющий имя данного поля (идентификатор поля). В таблице представлены основные типы, применяемых элементов <INPUT>:

TYPE=text	Определяет окно для ввода строки текста. Может содержать дополнительные атрибуты SIZE=число (ширина окна ввода в символах) и MAXLENGTH=число (максимально допустимая длина вводимой строки в символах): <INPUT TYPE=text SIZE=20 NAME=User VALUE="LENIN INC"> Определяет окно шириной 20 символов для ввода текста. По умолчанию в окне находится текст LENIN INC, который пользователь может изменить.
TYPE=password	Определяет окно для ввода пароля. Абсолютно аналогичен типу text, только вместо

	символов вводимого текста показывает на экране звездочки (*): <INPUT TYPE=password NAME=PW SIZE=20 MAXLENGTH=10> Определяет окно шириной 20 символов для ввода пароля. Максимально допустимая длина пароля — 10 символов.
TYPE=radio	Определяет радиокнопку. Может содержать дополнительный атрибут CHECKED (показывает, что кнопка отмечена). В группе радиокнопок с одинаковыми именами может быть только одна помеченная радиокнопка: <INPUT TYPE=radio NAME=Question VALUE="Yes" CHECKED> Да <INPUT TYPE=radio NAME=Question VALUE="No"> Нет <INPUT TYPE=radio NAME=Question VALUE="Possible"> Возможно Да Нет Возможно Определяет группу из трех радиокнопок, подписанных Yes, No и Possible. Первоначально помечена первая из кнопок. Если пользователь не отметит другую кнопку, обработчику будет передана переменная Question со значением Yes. Если пользователь отметит другую кнопку, обработчику будет передана переменная Question со значением No или Possible.
TYPE=checkbox	Определяет квадрат, в котором можно сделать пометку. Может содержать дополнительный атрибут CHECKED (показывает, что квадрат помечен). В отличие от радиокнопок, в группе квадратов с одинаковыми именами может быть несколько помеченных квадратов: <INPUT TYPE=checkbox NAME=Comp VALUE="CPU"> Процессоры <INPUT TYPE=checkbox NAME=Comp VALUE="Video" CHECKED> Видеоадаптеры <INPUT TYPE=checkbox NAME=Comp VALUE="Scan"> Сканеры <INPUT TYPE=checkbox NAME=Comp VALUE="Modem" CHECKED> Модемы

	Процессоры Видеоадаптеры Сканеры Модемы Определяет группу из четырех квадратов. Первоначально помечены второй и четвертый квадраты. Если пользователь не произведет изменений, обработчику будут переданы две переменные: Comp=Video и Comp=Modem.
TYPE=hidden	Определяет скрытый элемент данных, который не виден пользователю при заполнении формы и передается обработчику без изменений. Такой элемент иногда полезно иметь в форме, которая время от времени подвергается переработке, чтобы обработчик мог знать, с какой версией формы он имеет дело. <INPUT TYPE=hidden NAME=version VALUE="1.1"> Определяет скрытую переменную version, которая передается обработчику со значением 1.1.
TYPE=submit	Определяет кнопку, при нажатии на которую запускается процесс передачи данных из формы обработчику: <INPUT TYPE=submit VALUE="Отправить">
TYPE=reset	Определяет кнопку, при нажатии на которую очищаются поля формы. Поскольку при использовании этой кнопки данные обработчику не передаются, кнопка типа reset может и не иметь атрибута name: <INPUT TYPE=reset VALUE=" Сброс ">

Формы могут содержать поля для ввода большого текста <TEXTAREA>:

<TEXTAREA NAME=address ROWS=5 COLS=50> Наберите здесь сообщение</TEXTAREA>

Атрибут NAME определяет имя, под которым содержимое окна будет передано обработчику. Атрибут ROWS устанавливает высоту окна в строках. Атрибут COLS устанавливает ширину окна в символах. Текст, размещенный между тэгами <TEXTAREA></TEXTAREA>, представляет собой содержимое окна по умолчанию. Пользователь может его отредактировать или просто стереть.

Кроме всего этого формы могут содержать меню выбора, которое начинается открывающимся тэгом <SELECT> (содержит обязательный атрибут NAME, определяющий имя меню) и завершается закрывающимся </SELECT>. Между ними находятся тэги <OPTION>, определяющие элемент меню. Обязательный атрибут VALUE устанавливает значение, которое будет передано обработчику, если выбран этот элемент меню. тэг <OPTION> может включать атрибут selected, показывающий, что данный элемент выбран/отмечен по умолчанию.

```
<SELECT NAME="имя">
<OPTION VALUE="option_1" selected>текст 1
<OPTION VALUE="option_2">текст 2
<OPTION VALUE="option_n">текст n
</SELECT>
```

Тэг <SELECT> может также содержать атрибут MULTIPLE, присутствие которого показывает, что из меню можно выбрать несколько элементов. Большинство Обозревателей показывают меню <SELECT MULTIPLE> в виде окна, в котором находятся элементы меню. Высоту окна в строках можно задать атрибутом SIZE=число.

```
<SELECT MULTIPLE SIZE=3 NAME="имя">
<OPTION VALUE="option_1" selected>текст 1
<OPTION VALUE="option_2">текст 2
<OPTION VALUE="option_n">текст n
</SELECT>
```

ЗАДАНИЕ: Создайте E-mail форму

Цель: закрепить ЗУН при работе с формами

ПРОЕКТ №11 Создайте web-страничку, содержащую формы:

Текста;

Пароля;

Радиокнопку;

Checkbox;

Кнопку, при нажатии на которую запускается процесс передачи данных из формы обработчику;

Кнопку, при нажатии на которую очищаются поля формы;

Поля для ввода большого текста;

Меню выбора двух видов;

E-mail форму.

Все формы должны быть логически связанны между собой информацией. Страница должна быть красочно оформлена и иметь минимум два перехода (ссылки).

E-mail форма.

Для того чтобы отправить сообщение по электронной почте не обязательно набирать его непосредственно в почтовой программе, а можно это сделать, воспользовавшись формами на странице. Кроме этого можно еще организовать и проверку полей формы, на тот случай, если пользователь решил поиграться.

Имя:	
Тема:	
Сообщение:	
Отправить	Сброс

Данная форма реализуется следующим кодом:

```
<FORM NAME="mailer" METHOD="post" ACTION="" ENCTYPE="text/plain" onSubmit="(document.mailer.action
+= mailtoandSubject)">
<table border=2 align=center cellspacing=1 cellpadding=2 BgColor=#000FFF>
<tr><td><FONT size="2" color="#FFFFFF">Имя:</font></td>
<td><INPUT TYPE="text" NAME="Name" size="24" onChange="msg(this.form)"></td></tr>
<tr><td><FONT size="2" color="#FFFFFF">Тема:</font></td>
<td><INPUT TYPE="text" NAME="Subject" size="24" onChange="msg(this.form)"></td></tr>
<tr><td><FONT size="2" color="#FFFFFF">Сообщение:</font></td>
<td><TEXTAREA NAME="Message" COLS=40 ROWS=6 onChange="msg(this.form)"></TEXTAREA></td></tr>
<tr><td colspan=2 align=center><INPUT TYPE = "submit" VALUE = "Отправить" ONCLICK="return checkIt()">
<INPUT TYPE=reset VALUE=" Сброс "></td></tr></table>
</FORM>
```

САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ

Вариант 1

Практическое задание.

Задача по HTML и CSS: создание HTML таблиц и работа с ними

Выполните все пункты задания и сравните с результатом. Для поиска неизвестных вам тегов и свойств используйте поиск справочника.

1. Создайте папку в удобном для вас месте на вашем компьютере
2. В этой папке создайте новый HTML документ - *index.html*
3. В *index.html* создайте HTML скелет документа
4. Создайте новый CSS файл - *style.css*
5. Подключите CSS файл к HTML файлу
6. Создайте таблицу состоящую из 5 строк и 5 столбцов, укажите таблице класс *table*
7. Первую строку оберните в тег *thead* и столбцы переделайте в заглавные столбцы (*th*)
8. Последнюю строку оберните в тег *tfoot*
9. Оставшиеся строки оберните в тег *tbody*
10. Обозначьте столбцы первой строки заголовками, прописав текст **(первый - №, второй - Имя, третий - E-mail, четвертый - Пол, пятый - Дата)** в тегах *th*
11. Заполните строки, которые находятся в *tbody* любыми произвольными данными
12. В последней строке, которая находится в *tfoot*, объедините все столбцы в один, используя соответствующий атрибут для *td*
13. В получившимся столбце напишите текст **"Всего: 3"**
14. В CSS файле создайте селектор *.table td, .table th* и создайте сплошную границу толщиной в 1 пиксель и цветом #ccc; и задайте внутренний отступ сверху и снизу по 5px, слева и справа по 10px
15. Создайте селектор *.table* и объедините границы ближайших столбцов при помощи специального свойства (ищите в поиске "поведение границ таблицы"), так же задайте таблице **ширину 500px**;
16. Создайте селектор *.table thead* и измените цвет фона на *#f0f1f4*;
17. Создайте селектор *.table tfoot* и измените цвет фона на *#121212*;, цвет текста на *#fff;* и текст прижать к правому краю

№	Имя	E-mail	Пол	Дата
1	Дмитрий	dmitry@mail.com	М	21.11.2020
2	Александр	alex@mail.com	М	23.11.2020
3	Виктория	vika@mail.com	Ж	24.11.2020
				Всего: 3

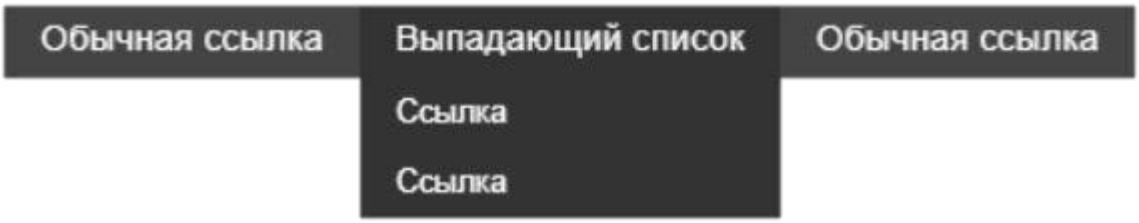
Вариант 2

Практическое задание.

Задача по HTML и CSS: создайте многоуровневую навигацию (выпадающее меню)

Можно использовать только HTML и CSS.

- Используйте ненумерованный список для создания структуры навигации
- Выпадающее меню по умолчанию должно быть скрыто
- При наведении на элемент навигации, если внутри есть выпадающее меню - показываем его
- При перемещении курсора на выпадающее меню, оно не должно закрываться
- Цвета можете выбирать по вашему вкусу, в примере использовались *#444* и *#333* для фона и цвет текста *#fff*



Вариант 3

Практическое задание.

Скачайте файлы задания и создайте композицию, которую вы видите на изображении, используя относительное и абсолютное позиционирование, а так же z-index. После выполнения сравните с результатом.

1. Размер композиции 600x400
2. Фон композиции вставьте через CSS
3. Каждый элемент композиции - это картинка (*img*), которую нужно расположить в нужную позицию относительно композиции. Используйте классы и стилизуйте их
4. Попробуйте подобрать примерно похожие значения позиций элементов

Вариант 4

Практическое задание. Повторите страницу по данному по образцу:

position-2.zip

Скачайте файлы задания и создайте модальное окно, как на изображение, которое вы видите под заданием. После выполнения сравните с результатом.

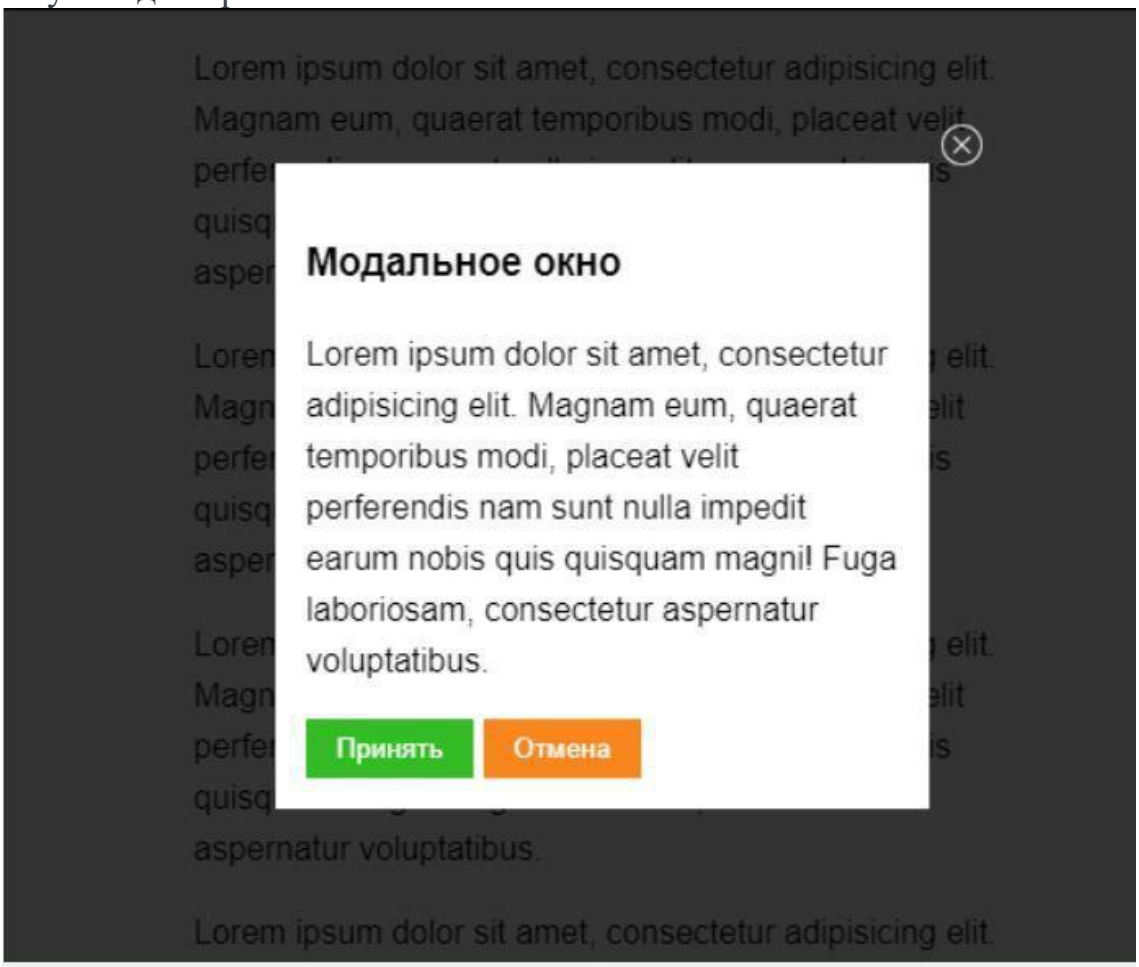
1. После блока с классом *text* создайте блок с классом *modal*
2. В блоке с классом *modal* добавьте заголовок третьего уровня и напишите текст **"Модальное окно"**
3. После заголовка в модальном окне добавьте абзац с произвольным текстом, можете использовать сайт-генератор случайного текста lipsum.com
4. После абзаца добавьте две кнопки используя тег *button* с типом *button*
5. В первой кнопке напишите текст **"Принять"**, а во второй кнопке напишите текст **"Отмена"**
6. Первая кнопка будет с классами *btn* и *success*, вторая с классами *btn* и *cancel*
7. Создайте еще одну обычную кнопку, в которой вместо текста будет изображение, которое находится в папке *img* и задайте ей класс *modal-close*
8. Переходим в CSS и стилизуем модальное окно. Ширина модального окна 320px, цвет фона белый, внутренний отступ 15px со всех сторон, позиционирование фиксированное. Позиция сверху 75px и по горизонтали по центру (используйте *left: 50%;* и отрицательный *margin-left* равный 1/2 ширины модального окна).

9. Размер кнопки *modal-close* 20x20, кнопка должна выходить за пределы модального окна относительно его. Используйте отрицательные значения *top, right* равные размеру кнопки

10. Кнопки *success* и *cancel* - должны идти в строку, иметь внутренний отступ сверху и снизу по 7px, слева и справа по 15px, шрифт и размер как у *body*. Фон кнопки *success* - #35bb27, а кнопки *cancel* - #f9861f, цвет текста белый.

11. Перед модальным окном создайте блок с классом *mask*, фон которой должен быть черным с непрозрачностью 80%. Маска должна быть фиксированной, на всю ширину и высоту, перекрывать контент, но не модальное окно.

12. Дополнительно: Попробовать переделать структуру модального окна используя *flexbox*. Блок модального окна на всю ширину и высоту с фоном как у маски, внутри контент модального окна со стилями как у *modal*. При помощи свойств *флексбоксов*выравниваем контент модального окна по центру по горизонтали. Смотрите результат реализации во втором примере. Данный способ реализации немного проще, гибче и предпочтительней. Но первый способ нужен для практики.



ПРОЕКТИРОВАНИЕ

Вариант	Тема	Описание	Примечание
1.	**Интернет-магазин**	Для товара храните в базе данных его код, название, цену, описание и фотографию (можно хранить название файла с фотографией, а сами файлы помещать в какую-нибудь папку). Пользователь может заказать товар - для этого он нажимает на кнопку "**Заказать**" и вводит свой адрес. Для каждого заказа храните в базе данных его дату, адрес покупателя и код товара. Должна быть также возможность добавлять и удалять товары, а также просматривать список товаров, страницу с конкретным товаром и список заказов. При удалении товара заодно удаляйте с диска файл с изображением.	Формы: добавление товара, заказ товара.
2.	**Фотогалерея с категориями**	Для фотографии храните в базе данных его код, название соответствующего файла, название фотографии, описание фотографии, место съемки, размер файла, код категории. Можно хранить в базе данных только название файла с фотографией, а сами файлы помещать в какую-нибудь папку. Для каждой категории храните в базе данных ее код и название.	Формы: добавление фотографии, добавление категории.

		Обеспечьте возможность добавления и удаления фотографий и категорий, просмотра списка всех категорий, списка всех фотографий в конкретной категории, просмотра одной фотографии. При удалении записи о фотографии из базы данных заодно удаляйте с диска файл с фотографией.	
3.	**Архив mp3 с категориями**	Для аудиофайла храните в базе данных его код, название файла, название песни, имя исполнителя, длительность звучания песни, размер файла, код категории. Сам файл храните в какой-нибудь папке на диске. Обеспечьте возможность просмотра списка категорий, списка песен в конкретной категории, просмотра определенной песни (то есть должна быть страница, на которой выводятся все данные об аудиофайле и можно прослушать песню). Должна быть также возможность добавлять песни и удалять их, а также добавлять и удалять категории. Для каждой категории храните в базе данных ее код и название. При удалении записи об аудиофайле из базы данных заодно удаляйте с диска сам файл.	Формы: добавление песни, добавление категории.

4.	**Видеогалерея с категориями**	Для видеофайла храните в базе данных его код, название файла, название видеоролика, описание видеоролика, длительность видеоролика, размер файла, код категории. Сам файл храните в какой-нибудь папке на диске. Обеспечьте возможность просмотра списка категорий, списка всех видеофайлов в конкретной категории, просмотра определенного файла (то есть должна быть страница, на которой выводятся все данные о видеофайле и можно посмотреть это видео). Должна быть также возможность добавлять файлы, просматривать их список и удалять их, а также добавлять и удалять категории. Для каждой категории храните в базе данных ее код и название. При удалении записи о видеофайле из базы данных заодно удаляйте с диска сам файл.	Формы: добавление видеоролика, добавление категории.
5.	**Гостевая книга**	Для каждой записи (сообщения) вашей гостевой книги храните в базе данных ее код, имя автора записи, e-mail автора записи, ICQ автора записи, дату добавления записи, текст записи. Для каждого ответа на запись храните его код, текст и код записи, к которой	Формы: добавление сообщения, добавление ответа на сообщение. Обратите внимание, что на странице для ответа на сообщение выводится поле для ответа как **textarea**, в которое можно ввести текст, а остальные

		относится этот ответ. Должна быть страница, на которой выводятся все сообщения. Обеспечьте возможность просмотра одного сообщения со всеми ответами на него, добавления сообщения, удаления сообщения, ответа на сообщение.	поля записи выводятся как текст (не для редактирования).
6.	**Форум**	Для каждого сообщения в базе данных храните его код, автора, текст, дату добавления и код темы. Для каждой темы - ее код и название. Обеспечьте возможность создания темы, ответа на тему, просмотра списка всех тем, просмотр темы (то есть всех ее сообщений), удаления темы, удаления сообщения. Ответить на тему - значит добавить в нее сообщение.	Формы: добавление темы, добавление сообщения.
7.	**Новостная система с категориями**	Для каждой новости храните в базе данных ее код, автора, текст, дату добавления и код категории. Для каждой категории - ее код и название. Обеспечьте возможность создания новости, создания категории, удаления новости, удаления категории, просмотра списка всех категорий, просмотра списка всех новостей в конкретной категории, просмотра конкретной новости.	Формы: добавление новости, добавление категории.
8.	**Ротатор баннеров**	Для каждого баннера храните в базе данных его код, название файла с	Формы: добавление баннера, добавление категории.

		изображением, ссылка, на которую будет переходить пользователь по нажатию на баннер, количество кликов - нажатий на данный баннер, количество показов баннера, код категории. Сами файлы с изображениями хранятся в какой-либо папке. Для каждой категории храните ее код и название. Обеспечьте возможность добавления и удаления баннеров, добавления и удаления категорий, просмотра списка категорий, списка баннеров в категории. При удалении баннера из базы данных заодно удаляйте с диска файл с изображением. На главной странице должен отображаться всегда только один баннер, выбранный случайным образом. Для этого сначала выберите случайным образом категорию, а затем, также случайным образом, - баннер, относящийся к этой категории.	
9.	**Доска объявлений с категориями**	Для каждого объявления храните в базе данных его код, автора, текст, дату добавления и код категории. Для каждой категории - ее код и название. Обеспечьте возможность добавления объявления, добавления	Формы: добавление объявления, добавление категории.

		категории, удаления объявления, удаления категории, просмотра списка всех категорий, просмотра списка всех объявлений в конкретной категории, просмотра конкретного объявления.	
10.	**Система управления контентом**	Для каждой страницы храните в базе данных ее код, автора, текст, дату добавления и код раздела. Для каждого раздела - его код, название и текстовое описание. Обеспечьте возможность создания страницы, создания раздела, удаления страницы, удаления раздела. Физически страницу на диске можно не создавать, достаточно, чтобы ее текст хранился в базе данных. Ссылки на все страницы выводятся в левой части главной страницы как пункты меню сайта. При нажатии на какой-либо пункт меню справа отображается текст соответствующей страницы, а также имя автора, дата добавления и название раздела. При этом название раздела отображается как гиперссылка, при нажатии на которую загружается страница с описанием этого раздела. Обеспечьте также возможность просмотра всех категорий и просмотра списка	Формы: добавление страницы, добавление раздела.

		страниц в определенной категории.	
11.	**Блог с облаком тегов**	Для каждой дневниковой записи храните в базе данных ее код, название записи, текст записи, автора записи, дату добавления записи. Для каждого тега храните в базе данных его код и название. Создайте третью таблицу в базе данных, связывающую теги с записями блога (поля: код, код записи, код тега). Обеспечьте возможность добавления и удаления записей и тегов, просмотра блога как списка всех записей, просмотра одной записи, списка тегов. В форме добавления записи в блог должен выводиться список со множественным выбором, в который выводятся все названия тегов из соответствующей таблицы базы данных.	Формы: добавление записи в блог, добавление тега.
12.	**Статьи с облаком тегов**	Для каждой статьи храните в базе данных ее код, автора, текст, дату добавления. Для каждого тега храните в базе данных его код и название. Физически страницу на диске можно не создавать, достаточно, чтобы ее текст хранился в базе данных. Создайте третью таблицу в базе данных, связывающую теги со статьями (поля:	Формы: добавление статьи, добавление тега.

		код, код статьи, код тега). Обеспечьте возможность добавления и удаления как статей, так и тегов, просмотра списка статей, просмотра конкретной статьи и просмотра списка тегов. В форме добавления статьи должен выводиться список со множественным выбором, в который выводятся все названия тегов из соответствующей таблицы базы данных.	
13.	**Портфолио с категориями**	Для каждой работы храните в базе данных ее код, название файла с изображением, название работы, описание работы, сроки, в которые выполнялась работа, код категории. Можно хранить в базе данных только название файла с фотографией, а сами файлы помещать в какую-нибудь папку. Для каждой категории храните в базе данных ее код и название. Обеспечьте возможность добавления и удаления работ и категорий, просмотра списка всех категорий, просмотра списка всех работ в конкретной категории, просмотра одной работы. При удалении записи о работе из базы данных заодно удаляйте с диска файл с изображением.	Формы: добавление работы, добавление категории.

14.	**Система "Вопрос-ответ"**	Для каждого вопроса храните в базе данных его код, имя автора, текст вопроса, дату добавления. Для каждого ответа - его код, код вопроса и текст ответа. Обеспечьте возможность добавления вопроса в конкретную категорию, просмотра вопроса (с ответом, если есть ответ), удаления вопроса, ответа на вопрос (на один вопрос может быть несколько ответов), просмотра списка вопросов в конкретной категории, просмотра списка категорий, добавления и удаления категорий.	Формы: добавление вопроса, добавление категории, ответ на вопрос.
15.	**Блог с категориями**	Для каждой дневниковой записи храните в базе данных ее код, название записи, текст записи, автора записи, дату добавления записи, код категории. Для каждой категории храните в базе данных ее код и название. Обеспечьте возможность добавления и удаления записей и категорий, просмотра блога как списка всех записей, просмотра списка записей в конкретной категории, просмотра одной записи.	Формы: добавление записи в блог, добавление категории.
16.	**Статьи с категориями**	Для каждой статьи храните в базе данных ее код, автора, текст, дату добавления и код раздела. Для каждого раздела - его код, название и текстовое	Формы: добавление статьи, добавление раздела.

		описание. Физически страницу на диске можно не создавать, достаточно, чтобы ее текст хранился в базе данных. Обеспечьте возможность добавления и удаления как страниц, так и разделов, просмотра статей по разделам, просмотра конкретной статьи и списка разделов.	
17.	**Календарь событий**	Для каждого события храните в базе данных его код, описание, дату, код места проведения. Для каждого места проведения - его код, название и фотографию (можно хранить в базе данных только названия файлов с фотографиями, а сами файлы хранить в какой-нибудь папке на диске). Обеспечьте возможность создания события, создания описания места проведения, удаления события, удаления описания места проведения, просмотра списка мест проведения событий. При удалении описания места удаляйте с диска и соответствующий файл с изображением. События должны просматриваться по одному либо в виде календаря. В случае, если на определенную дату есть события, дата в ячейке календаря должна выводиться в виде	Формы: добавление события, добавление места проведения.

		гиперссылки, при нажатии на которую загружается страница со списком всех событий, которые относятся к этой дате.	
18.	**Почтовая рассылка**	Для каждого сообщения рассылки храните в базе данных его код, название, текст, дату добавления. Для каждого подписчика - его код, e-mail и имя. Обеспечьте возможность добавления сообщения, регистрации подписчика, удаления подписчика, просмотра списка всех сообщений и списка всех подписчиков. При добавлении сообщения рассылки оно автоматически должно отсылаться на e-mail всем подписчикам.	Формы: добавление сообщения рассылки, регистрация подписчика.
19.	**Регистрация пользователей**	Для каждого пользователя храните в базе данных его код, имя, e-mail, пароль, дату регистрации и фотографию. Храните в базе данных только название файла с фотографией, а сами файлы помещайте в какую-нибудь папку на диске. При удалении учетной записи пользователя из базы данных заодно удаляйте с диска файл с его фотографией. Обеспечьте пользователю возможность заносить других пользователей в список друзей. Для этого	Формы: регистрация, вход на сайт под логином и паролем.

		создайте вторую таблицу в базе данных, со следующими полями: код записи **id**, код первого пользователя **id1**, код второго пользователя **id2**, подтвержденность связи **isChecked**. При отображении профиля пользователя должна выводиться кнопка "**Добавить в друзья**". При нажатии на эту кнопку во второй таблице создается запись, в которую заносятся коды обоих пользователей и 0 для признака подтвержденности связи. Обеспечьте пользователю возможность просмотра списка предложений дружбы и принятия или отклонения каждого из них. Выбрать предложения дружбы для пользователя с кодом 35 можно путем следующего SQL-запроса: SELECT id1 FROM table2 WHERE id2=35 AND isChecked=0; Не забудьте сделать страницы для регистрации пользователей и для просмотра профиля пользователя и списка его друзей.	
20.	**Статьи с возможностью голосования**	Для каждой статьи храните в базе данных ее код, автора, текст, дату добавления. Для каждого голосования - его код, код	Формы: добавление статьи, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").

		статьи и рейтинг - численную оценку статьи от 1 до 5. Физически страницу на диске для статьи можно не создавать, достаточно, чтобы ее текст хранился в базе данных. Обеспечьте возможность добавления и удаления статей, просмотра списка статей (отсортированных по рейтингу/по дате добавления), просмотра одной из статей и голосования за каждую из них.	
21.	**Фотогалерея с возможностью голосования**	Для фотографии храните в базе данных его код, название соответствующего файла, название фотографии, описание фотографии, место съемки, размер файла. Можно хранить в базе данных только название файла с фотографией, а сами файлы помещать в какую-нибудь папку. Для каждого голосования храните в базе данных его код, код фотографии и рейтинг - численную оценку фотографии от 1 до 5. Обеспечьте возможность добавления и удаления фотографий, просмотра списка всех фотографий (отсортированных по рейтингу/по дате добавления), просмотра одной фотографии, голосования за каждую из	Формы: добавление фотографии, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").

		них. При удалении записи о фотографии из базы данных заодно удаляйте с диска файл с фотографией.	
22.	**Архив документов MS Office**	Для каждого документа MS Office (*.doc, *.docx, *.ppt, *.pptx и т.д.) храните в базе данных его код, название файла, размер файла, код категории. Сам файл храните в какой-нибудь папке на диске. Обеспечьте возможность просмотра определенного файла со ссылкой для его скачивания, просмотра списка категорий и списка файлов в определенной категории. Должна быть также возможность добавлять файлы и удалять их, а также добавлять и удалять категории. Для каждой категории храните в базе данных ее код и название. При удалении записи о файле из базы данных заодно удаляйте с диска сам файл.	Формы: добавление файла, добавление категории.
23.	**Архив mp3 с возможностью голосования**	Для аудиофайла храните в базе данных его код, название файла, название песни, имя исполнителя, длительность звучания песни, размер файла. Сам файл храните в какой-нибудь папке на диске. Обеспечьте возможность просмотра списка песен (отсортированных по рейтингу/по дате	Формы: добавление песни, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").

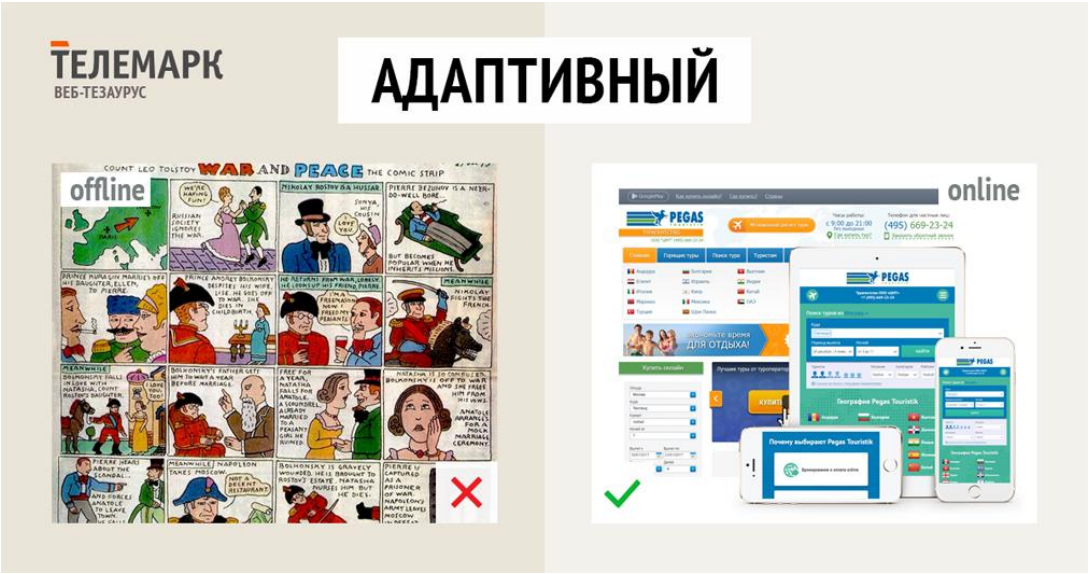
		добавления), просмотра определенной песни (то есть должна быть страница, на которой выводятся все данные об аудиофайле и можно прослушать песню). Должна быть также возможность добавлять песни и удалять их, а также голосовать за них. Для каждого голосования храните в базе данных его код, код файла и рейтинг - численную оценку файла от 1 до 5. При удалении записи об аудиофайле из базы данных заодно удаляйте с диска сам файл.	
24.	**Видеогалерея с возможностью голосования**	Для видеофайла храните в базе данных его код, название файла, название видеоролика, описание видеоролика, длительность видеоролика, размер файла. Сам файл храните в какой-нибудь папке на диске. Обеспечьте возможность просмотра списка видеофайлов (отсортированных по рейтингу/по дате добавления), просмотра определенного файла (то есть должна быть страница, на которой выводятся все данные о видеофайле и можно посмотреть это видео) с возможностью проголосовать за него. Должна быть также возможность добавлять	Формы: добавление видеоролика, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").

		файлы и удалять их. Для каждого голосования храните в базе данных его код, код файла и рейтинг - численную оценку файла от 1 до 5. При удалении записи о видеофайле из базы данных заодно удаляйте с диска сам файл.	
25.	**Новостная система с возможностью голосования**	Для каждой новости храните в базе данных ее код, автора, текст, дату добавления. Для каждого голосования - его код, код новости и рейтинг - численную оценку новости от 1 до 5. Обеспечьте возможность создания новости, удаления новости, просмотра списка всех новостей, отсортированного по рейтингу/по дате добавления, просмотра одной новости с возможностью оценить ее числом от 1 до 5.	Формы: добавление новости, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").
26.	**Портфолио с возможностью голосования**	Для каждой работы храните в базе данных ее код, название файла с изображением, название работы, описание работы, сроки, в которые выполнялась работа. Можно хранить в базе данных только название файла с фотографией, а сами файлы помещать в какую-нибудь папку. Для каждого голосования храните в базе данных его код, код работы и рейтинг - численную оценку	Формы: добавление работы, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").

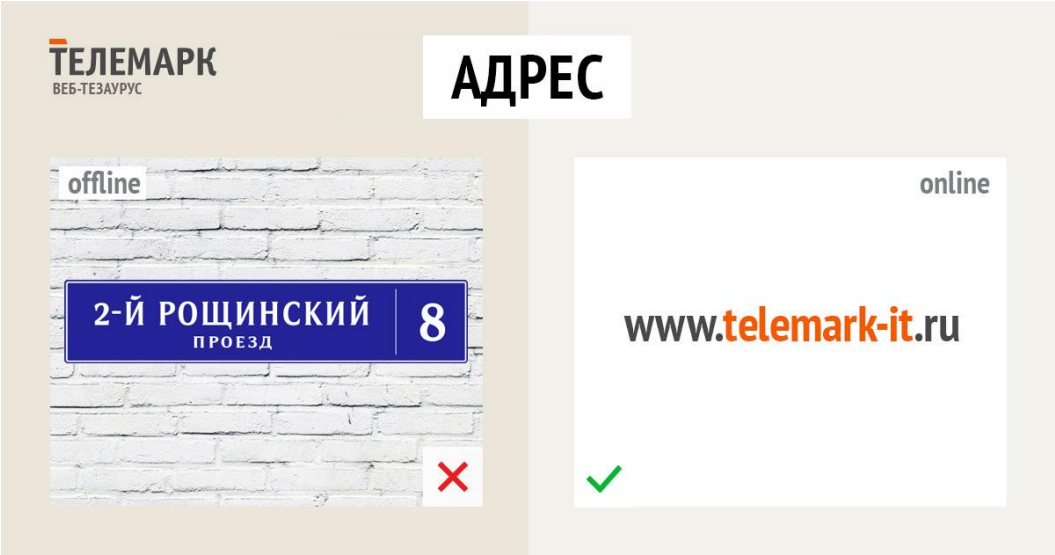
		работы от 1 до 5. Обеспечьте возможность добавления и удаления работ, просмотра списка всех работ, просмотра одной работы с возможностью оценить ее числом от 1 до 5. При удалении записи о работе из базы данных заодно удаляйте с диска файл с изображением.	
27.	**Блог с возможностью голосования**	Для каждой дневниковой записи храните в базе данных ее код, название записи, текст записи, автора записи, дату добавления записи. Для каждого голосования - его код, код дневниковой записи и рейтинг - численную оценку записи от 1 до 5. Обеспечьте возможность добавления и удаления записей, просмотра блога как списка всех записей (отсортированных по рейтингу/по дате добавления), просмотра одной записи с возможностью оценить ее числом от 1 до 5.	Формы: добавление записи в блог, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").
28.	**Доска объявлений с возможностью голосования**	Для каждого объявления храните в базе данных его код, автора, текст, дату добавления. Для каждого голосования - его код, код объявления и рейтинг - численную оценку объявления от 1 до 5. Обеспечьте возможность добавления объявления, удаления объявления, просмотра списка всех	Формы: добавление объявления, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").

		объявлений, отсортированных по рейтингу/по дате добавления.	
29.	**Система "Вопрос-ответ" с возможностью голосования**	Для каждого вопроса храните в базе данных его код, автора, текст вопроса, текст ответа, дату добавления. Для каждого голосования - его код, код вопроса и численную оценку ответа от 1 до 5. Обеспечьте возможность добавления вопроса, удаления вопроса, ответа на вопрос (на один вопрос можно ответить только один раз), просмотра списка вопросов, отсортированного по рейтингу/по дате добавления, просмотра одного вопроса с возможностью оценить его числом от 1 до 5. Очевидно, что при добавлении вопроса поле для ответа на него не выводится и записывается в базу данных пустым. Наоборот, на странице для ответа на вопрос выводится поле для ответа как **textarea**, в которое можно ввести текст, а остальные поля записи выводятся как текст (не для редактирования).	Формы: добавление вопроса, ответ на вопрос, голосование (выпадающий список с числами от 1 до 5 и кнопка "**Оценить**").

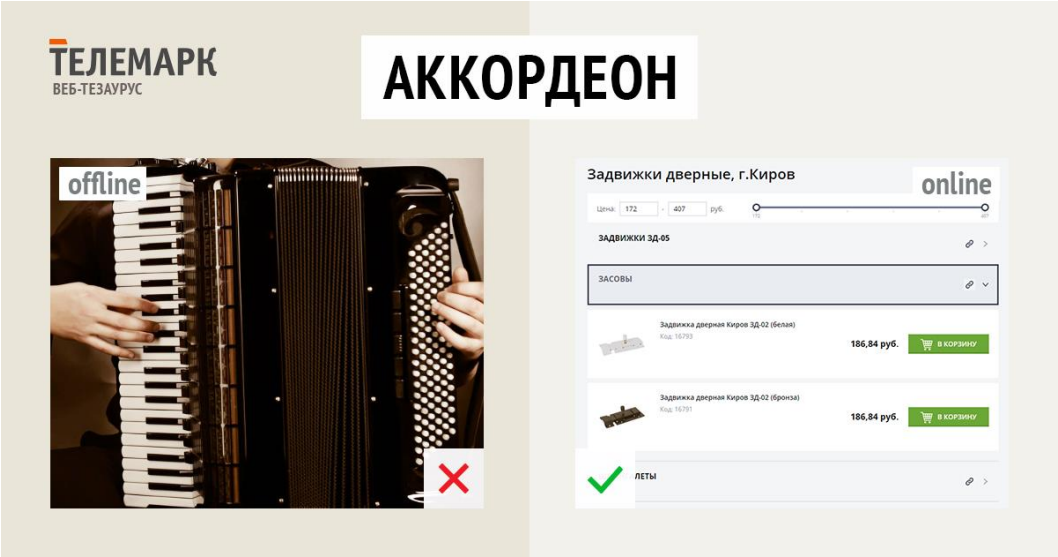
ТЕЗАУРУС



Вы по—прежнему считаете, что адаптивный и отзывчивый сайты синонимы? В адаптивном сайте спецкод определяет тип устройства (настольный ПК, телефон или планшет), а затем подгружает соответствующую HTML—версию сайта. А теперь сравним: отзывчивый дизайн — один макет для всех устройств, адаптивный дизайн — один макет для каждого вида устройства.

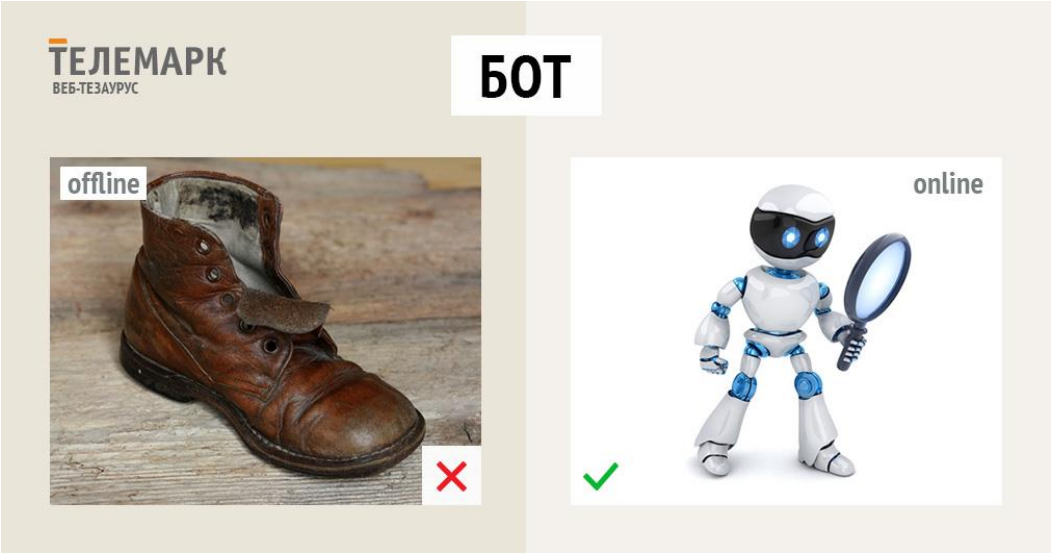


Адрес сайта — это место, где физически расположен ваш сайт и где его может найти браузер, чтобы показатель посетителю по запросу. В структуре адреса сайта, как и в адресе проживания человека, нет лишних элементов. В адресе обязательно присутствуют: http или https — это протокол передачи гипертекста, www.telemark-it.ru — доменное имя сайта, в котором обозначен также и региональный компонент (.ru — национальный домен верхнего уровня для России).

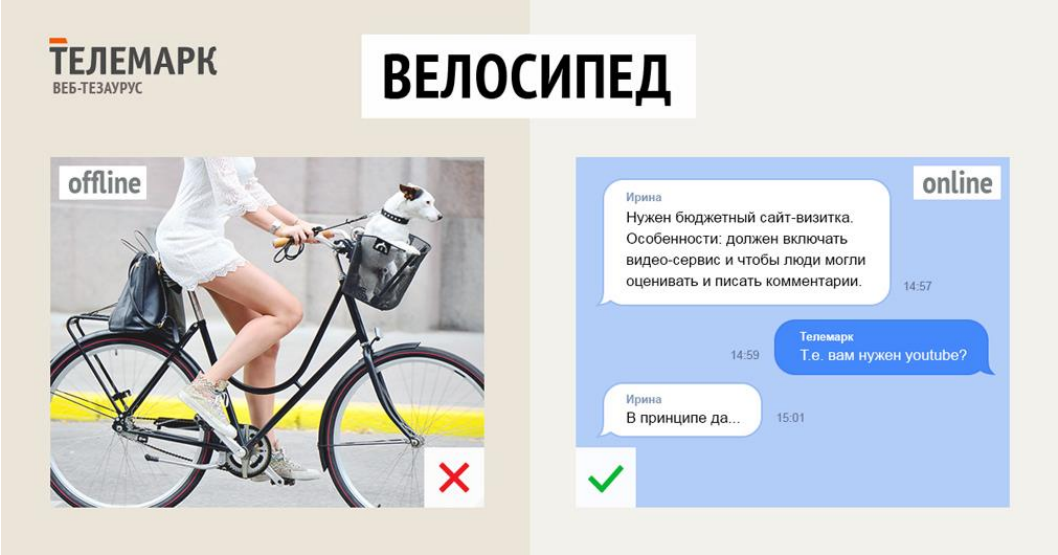


Меню в стиле «аккордеон» — один из самых востребованных типов меню при разработке сайта. Подобно одноименному музыкальному инструменту это меню раскрывается по клику, существенно экономя место на сайте. Это особенно актуально, когда на данной странице достаточно много контента, разбитого по разделам. Меню предоставляет посетителю возможность раскрыть только ту часть контента, которая его заинтересовала.

В IT эта характеристика обычно применяется к понятию «файл». В результате различных сбоев, например, при скачивании мы получаем неработоспособный битый файл, в котором утеряна вся или часть информации. С помощью восстановления иногда получается «вылечить» беднягу.

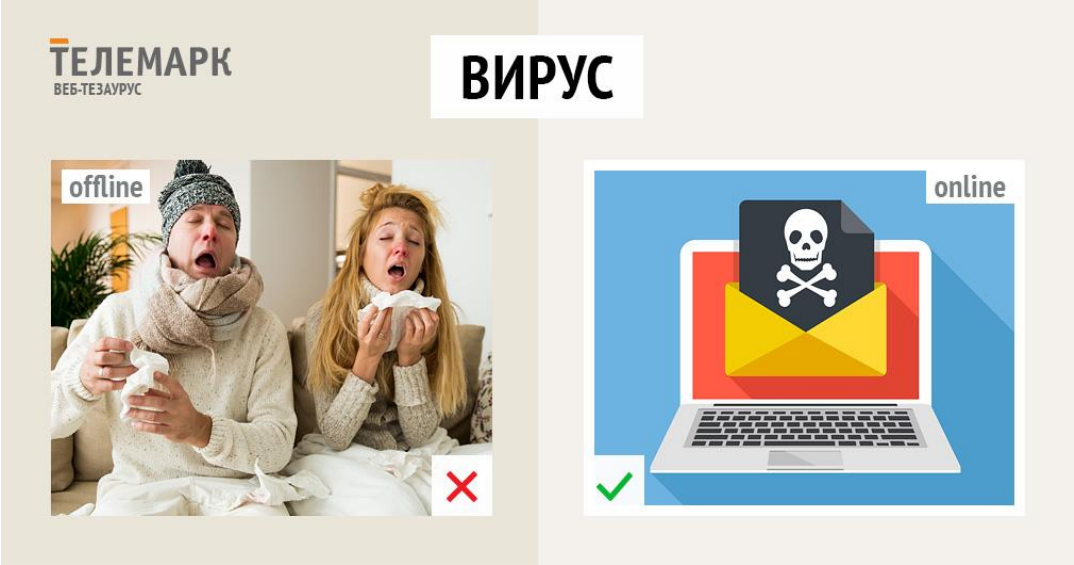


Сокращенная форма от слова «робот» предназначена специальной программе, которая выполняет определенный набор действий через пользовательский интерфейс. Образно говоря, шагающий человечек в ботах, который проверяет, удобно ли пользователю работать с вашим интерфейсом.



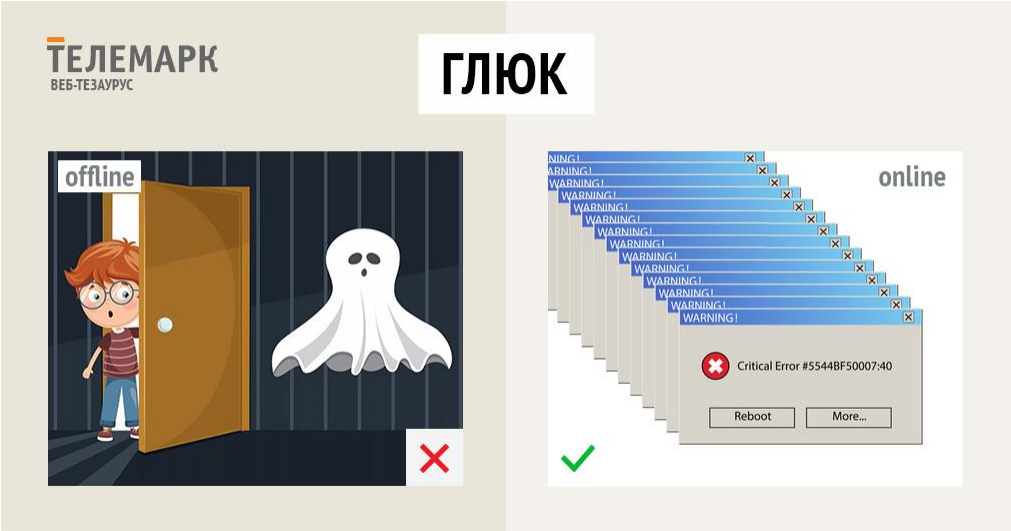
Это сленговое словечко используется в любой сфере жизни человека и везде несет одно и тоже значение, а именно «нечто неоригинальное, давно известное всему свету». Веб—сфера не является исключением.

Взлом программного обеспечения означает, что его защита устранена и теперь взломщик имеет полный доступ к закрытой ранее информации и функциональным возможностям объекта. Данное действие является противозаконным и, следовательно, уголовно наказуемым.

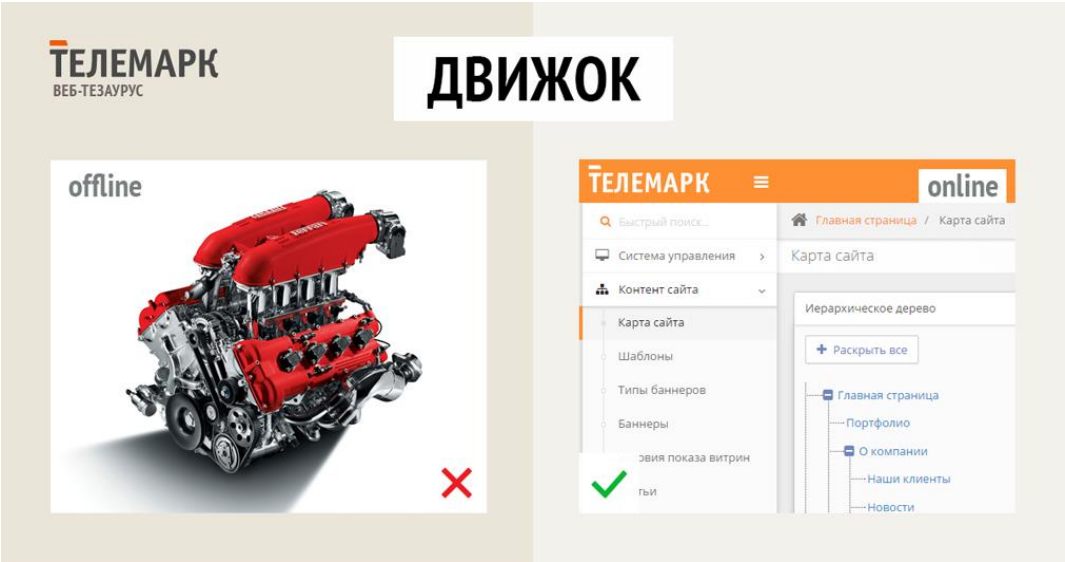


Вирусы отличаются крайней вредоносностью и невообразимым разнообразием, а также очень легко попадают внутрь организма. Компьютерный вирус обладает тем же «джентльменским набором». Единственное лечение — антивирусный препарат/программа, да и, пожалуй, крепкий иммунитет / надежное антивирусное программное обеспечение.

Г



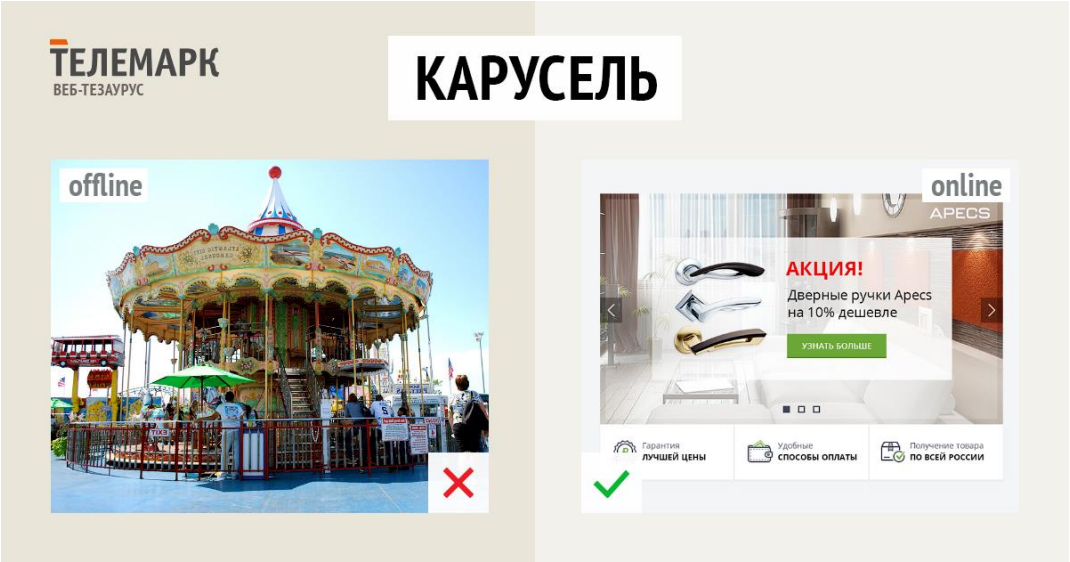
Что для знатоков немецкого языка «счастье, везение», то для русскоговорящих «галлюцинация и сбой в работе программы». Глюки — это неудавшаяся попытка программы выйти из зоны комфорта и заняться недокументированной деятельностью.



Движок, или система управления сайтом (CMS), — это сердце сайта. Задача движка — свести вместе дизайн, контент и функционал сайта. С его помощью разработчики комфортно взаимодействуют с базами данных, а менеджеры без IT—навыков могут по мере необходимости менять, добавлять или удалять контент.

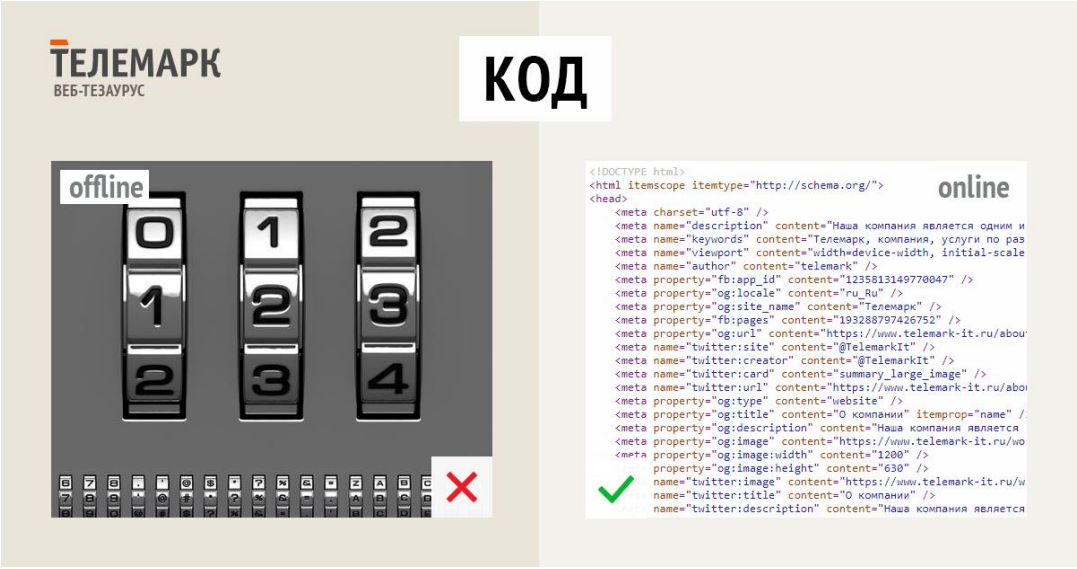
Дерево, как жизненная форма с главной осью, т.е. стволом, является 100% ассоциацией структуры данных в ИТ. В ней также есть корневой узел, корень, листовой (терминальный) узел и внутренний узел.

Заливать значит перенести (закачать) информацию с флешки, жесткого диска или любого другого носителя на сервер. Таким образом, залить в Интернет можно всё, что угодно: текст, картинки, фотографии, программы и др.



Карусель (или слайдер) на сайте — распространенный функционал, который часто используется для демонстрации специальных предложений. Автоматическое переключение картинок с текстом плюс подходящий интервал позволят удержать внимание посетителя и кратко ознакомить его с основной информацией.

Кнопки бывают разными, но их объединяет одно: при нажатии на нее происходит некое событие. В обычной жизни и в мире IT результат может быть непредсказуемым: «событие» может иметь одинаково негативный результат, если это кнопка на стуле преподавателя или «левая» кнопка на сайте, которые запускают цепочку весьма нерадостных последствий.



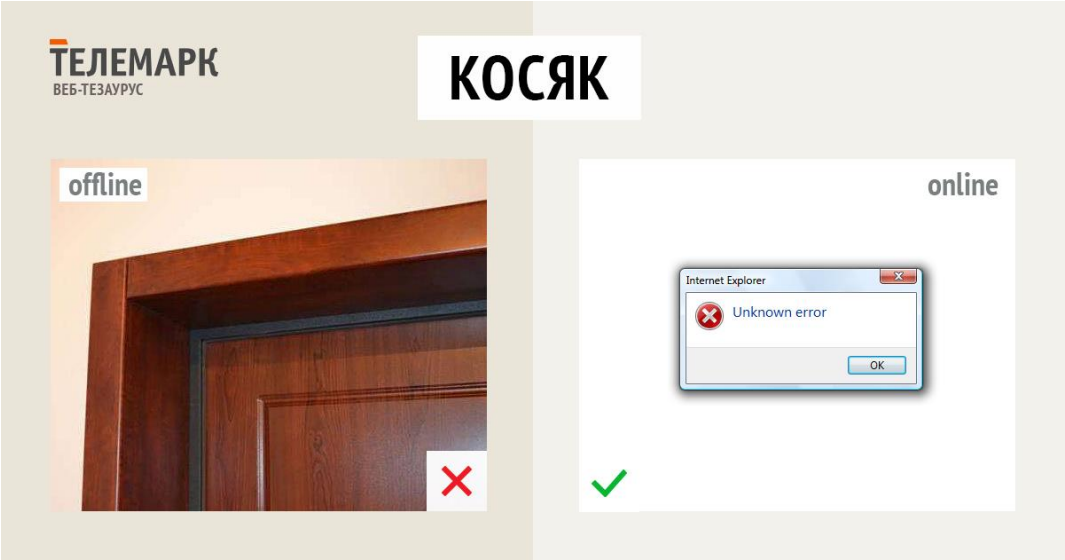
В обычной жизни код защищает материальные ценности от посягательств сторонних лиц. Эту же функцию он выполняет и в сфере IT: с помощью кода мы шифруем свои личные данные и ограничиваем доступ к ним. Кроме того, различные виды кода (php—код, js—код, html—код) хранят, передают и позволяют обрабатывать информацию, которую мы видим на сайте.

Соприкосновение, взаимодействие, связь, сотрудничество... Без контактов ни в повседневной жизни, ни в любой другой сфере нельзя достичь взаимодействия и наладить связь между кем или чем бы то ни было.

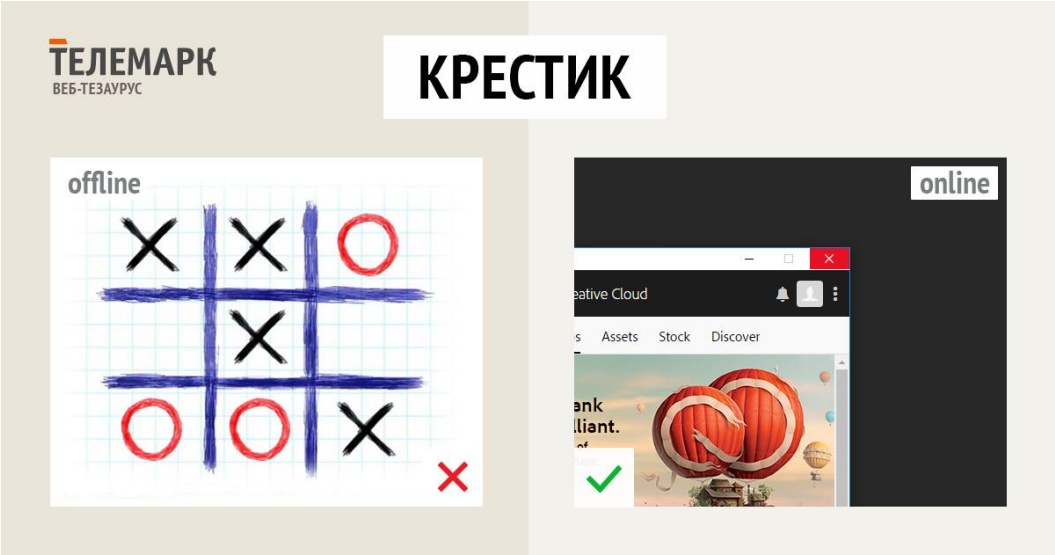
По сути контейнер в ИТ полностью соответствует своему значению в обычной жизни, а именно «ёмкость, вместилище». Как элемент интерфейса контейнер содержит в себе другие элементы интерфейса программы, объединяет их и позволяет ими управлять.

Ассоциация с магазинной тележкой задействует одну из самых важных потребностей человеческого организма — питание. Именно поэтому посетители стремятся положить что-то в виртуальную корзину. По этой же причине значок корзины не требует сопроводительного текста.

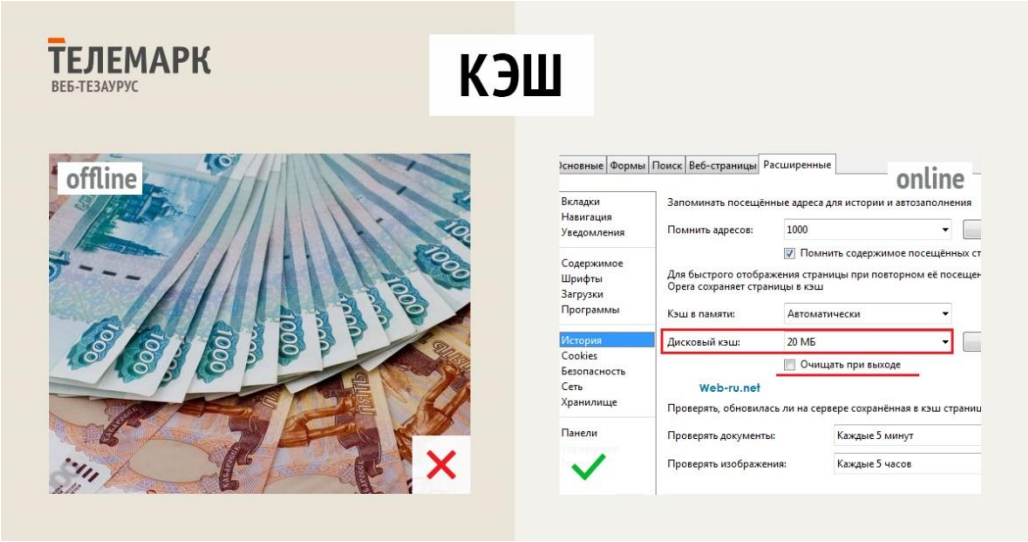
Костыль — обходной приём в программировании, который позволяет быстро и экономично решить проблему, устранив ее последствия, но не причины её появления. Однако это временное решение, которое не предполагает дальнейшего развития системы.



В ИТ, а также и в обычной жизни благодаря молодежному сленгу, брус дверной или оконной рамы превратился в банальную ошибку. Традиционное выражение «совершить ошибку» теперь красочно звучит как «накосячить».



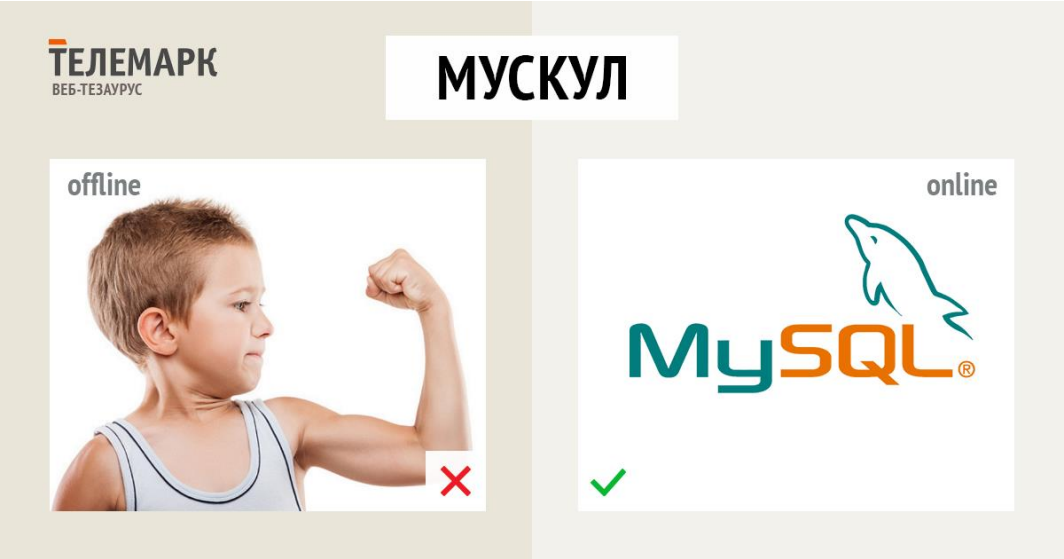
Спецсимвол «крестик» указывает на возможность закрыть страницу, форму, приложение. Не стоит его путать со знаком умножения. По одной из версий этот значок происходит от японского символа batsu («x») — символ некорректности.



Кэш представляет собой промежуточный буфер с быстрым доступом. Так же как и его клон в обычной жизни кэш позволяет быстро получить желаемое, однако по объему он гораздо меньше, чем банковский счет... то есть хранилище исходных данных.

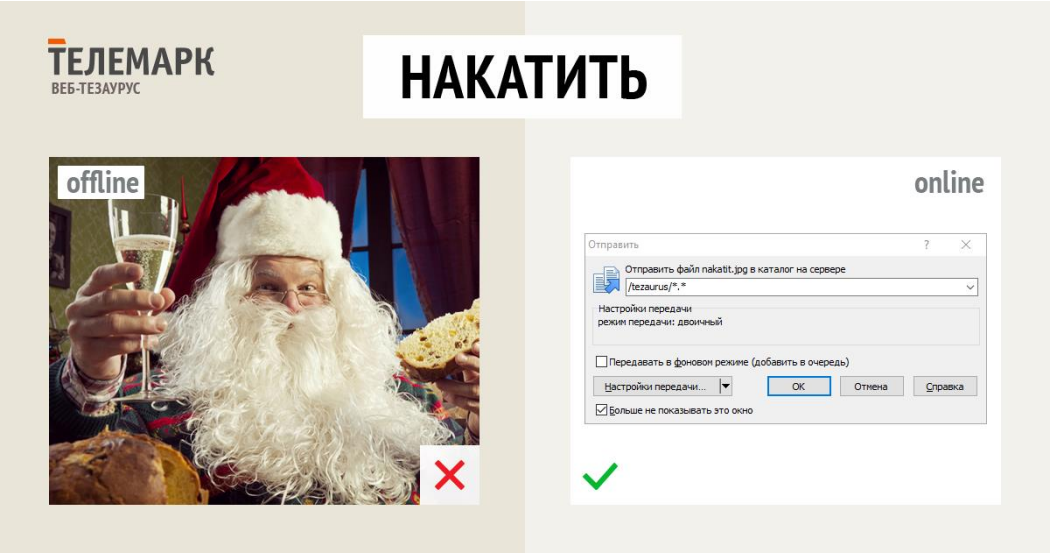
Л

Лепить из глины, создавая красивые фигурки, — это правильно и необходимо (душа требует красоты). Чем плотнее прилегают друг к другу слои, тем гармоничнее смотрится изделие. Однако в дизайне «лепить» строчки, не заботясь о приемлемом для глаз интерлиньяже, — это полный провал проекта в целом.

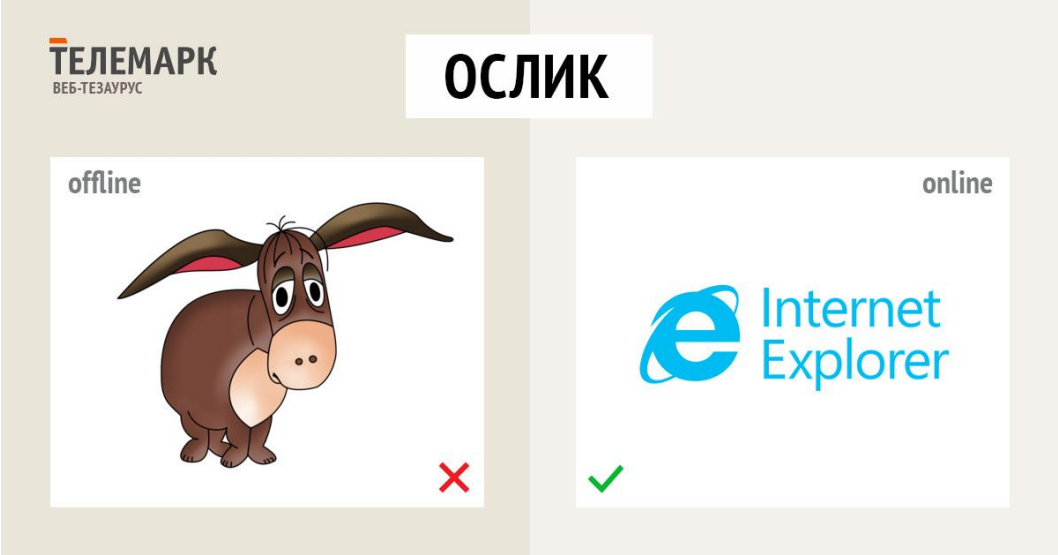


И опять на выручку приходит транслитерация, самый популярный прием освоения ИТ-терминов. Итак, поиграем мускулами, или май-эс-кью-Элами. MySQL — система управления реляционными базами данных. Широко используется при разработке информационных систем.

Н

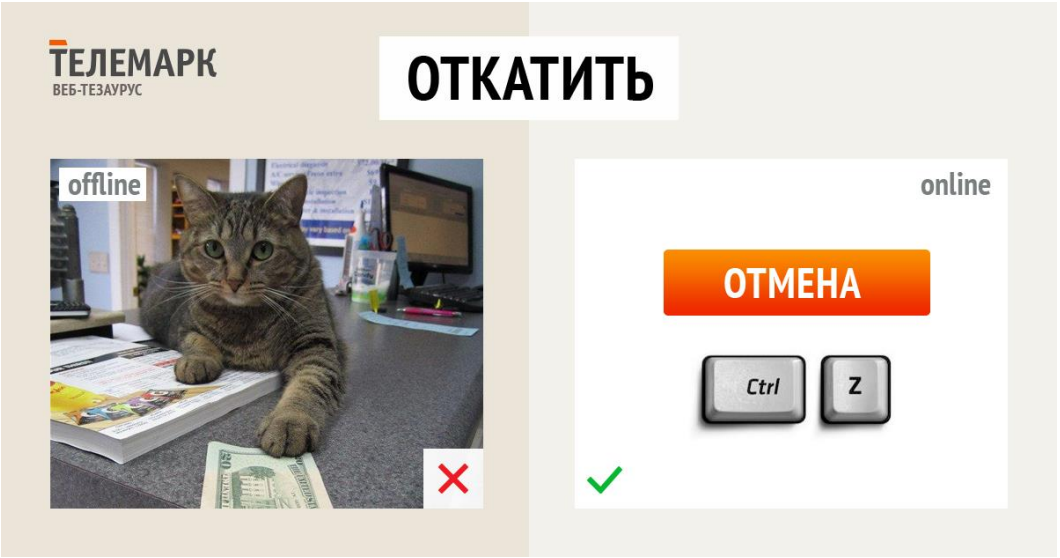


«Накатить» значит закачать новую информацию или код на сервер. Таким образом, и в реальной жизни, и в ИТ—мире действие приведет к изменению состояния. Если состояние Вам не понравится, вы всегда можете откатить (если, конечно, сохранили предыдущие настройки).

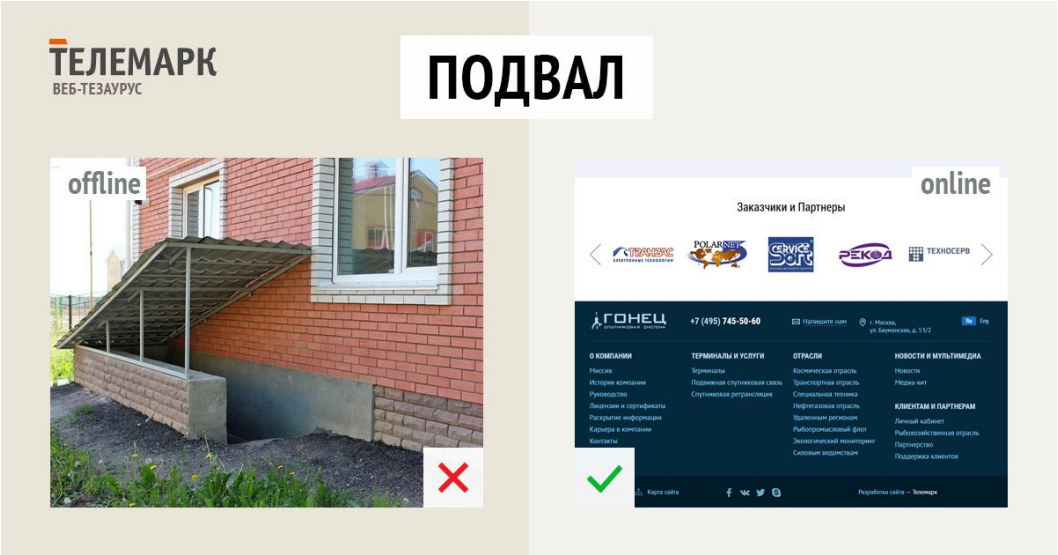


Так изящно называют браузер Internet Explorer, IE. Видимо, чудеса русской транслитерации вернули кого-то в детские воспоминания об ослике Иа. В любом случае разработка этой программы—браузера закончилась в 2015—м году, а на смену ей пришел Microsoft Edge и, начиная с Windows 10, стал неотъемлемой частью операционной системы семейства Windows.

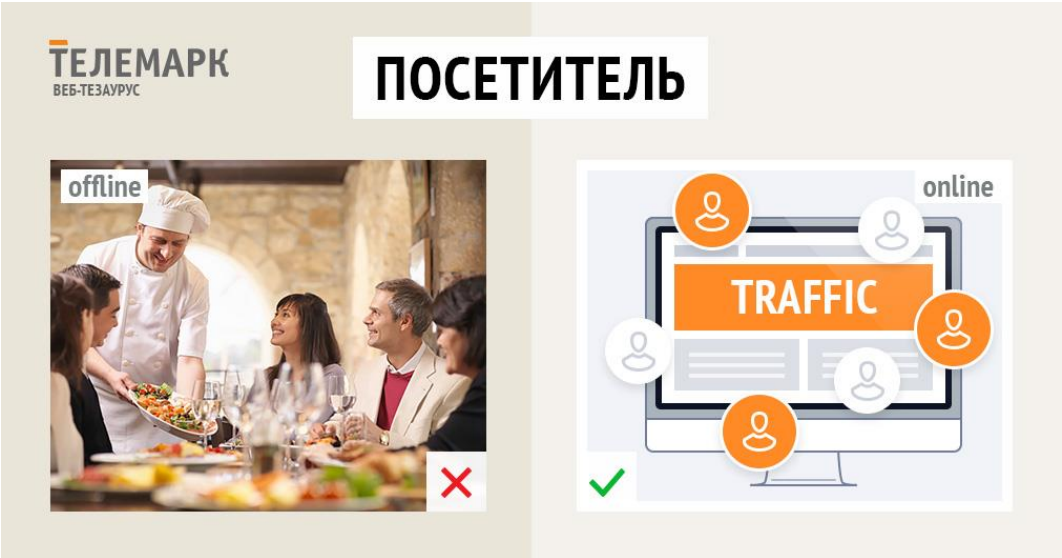
Отзывчивый сайт, в отличие от адаптивного, подстраивается под любое устройство, не меняя своей сущности (кода) и обеспечивая плавный переход от одного размера к другому. Таким образом, отзывчивость сайта помогает избежать дублей страниц, что очень нравится поисковым роботам.



На языке IT—специалистов откат системы означает возврат ее к прежнему состоянию. Эта необходимость обычно возникает, когда в новой версии обнаружены ошибки и требуется вернуть прежние настройки на время доработки новой версии.



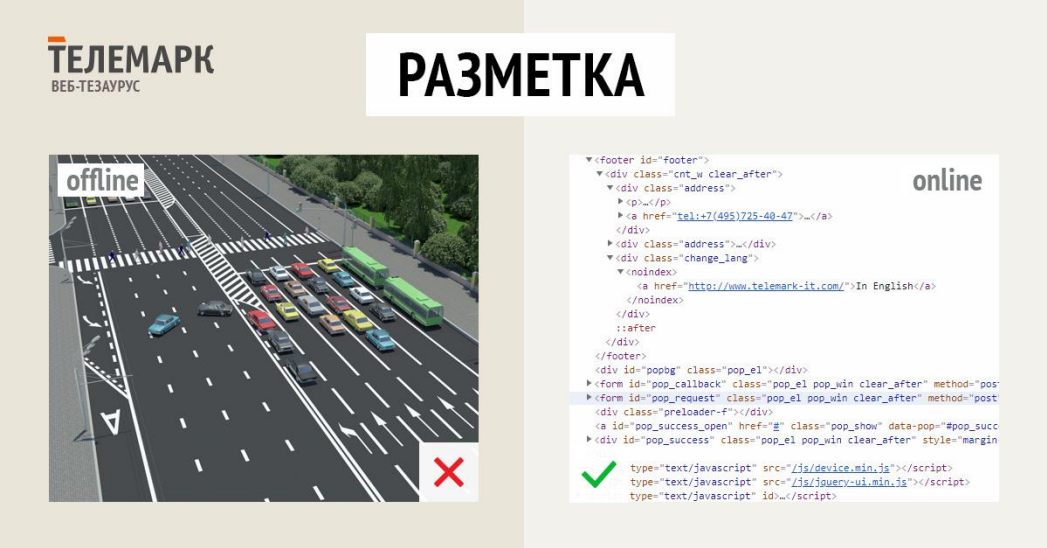
В подвалах домов обычно скрыта вся коммуникация, там тепло и сухо. В ИТ-сфере эта параллель сохраняется: в подвале сайта (его еще называют «футером») размещается дополнительная информация, копирайты, счетчик, кнопки соцсетей и/или дублируется основное меню и контактная информация компании.



Как и любое кафе, ресторан, кинотеатр, магазин, интернет-сайт прекратит свое существование, если не будет посетителей. Эта категория людей является потенциальными клиентами, потребителями предоставляемых услуг.

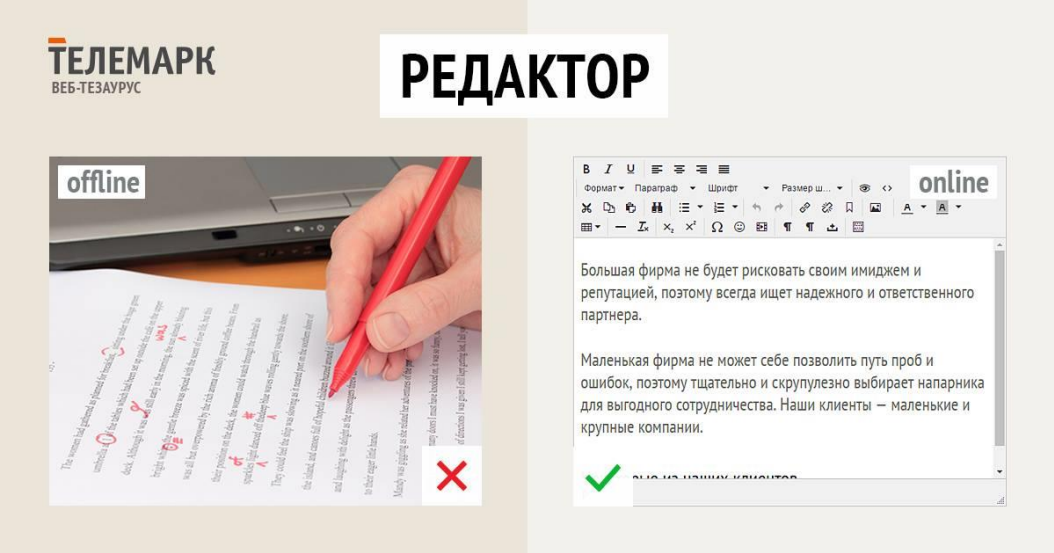
Р

По аналогии с кнопками в автомобильных радиоприемниках, которые были популярны лет эдак N назад и использовались для переключения радиостанций, радиокнопки в ИТ — это элемент интерфейса, с помощью которого пользователь выбирает одну опцию из определенного набора или группы.

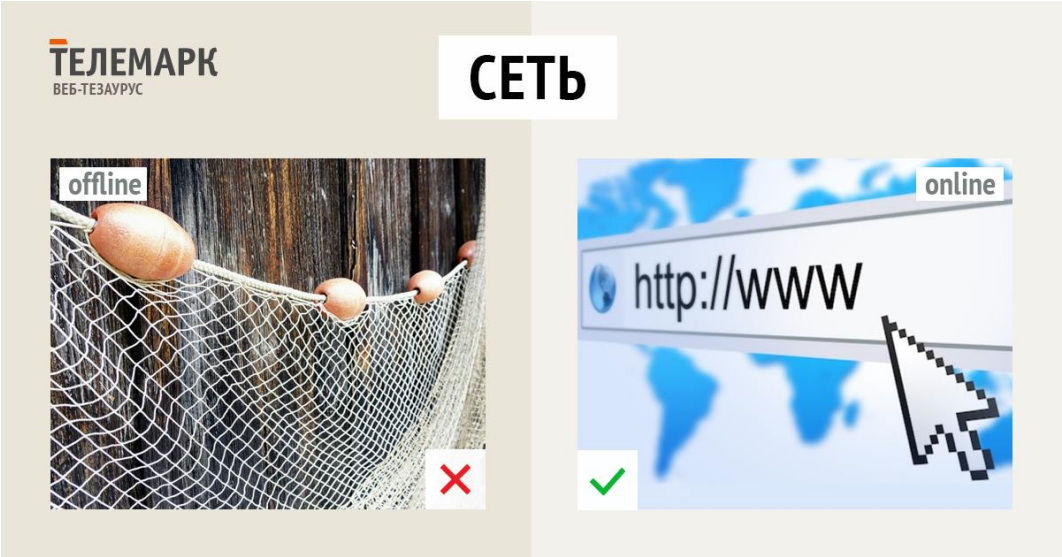


Для создания и успешного функционирования сайта разметка также важна, как и на дороге. С помощью разметки создается структура ресурса, а на ее основе происходит дальнейшее оформление сайта. Без структуры сайт просто не будет корректно работать.

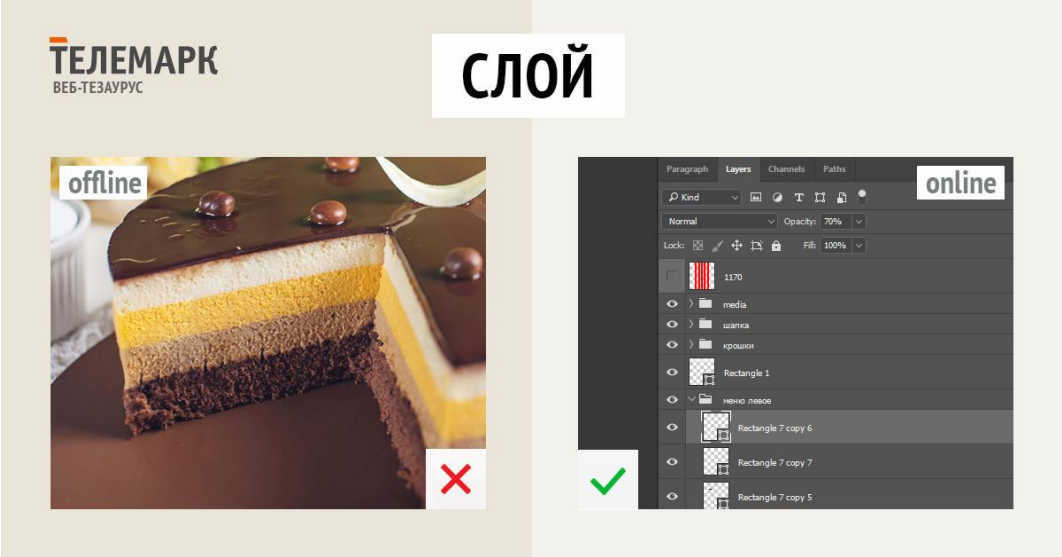
Разрешение монитора или изображения — это не их размеры, а количество точек на единицу площади. Чем выше разрешение, тем точнее представление оригинала, то есть тем выше качество изображения.



Визуальный (WYSIWYG*) редактор кода — это текстовый редактор, который позволяет создавать и редактировать контент страниц сайта. Обычно редактор кода встраивается в интегрированную среду разработки. Редактор позволяет добавлять изображения, таблицы, настраивать стили оформления текста, встраивать видео. WYSIWYG является аббревиатурой от англ. What You See Is What You Get, «что видишь, то и получишь».



Подобно крупной и мелкой рыбешке, в сеть Internet попадает и сохраняется любая информация, видео— и аудио—контент. Однако помимо ценных артефактов в нее очень часто попадают и рваные башмаки, то есть непроверенные сведения и откровенный фейк.

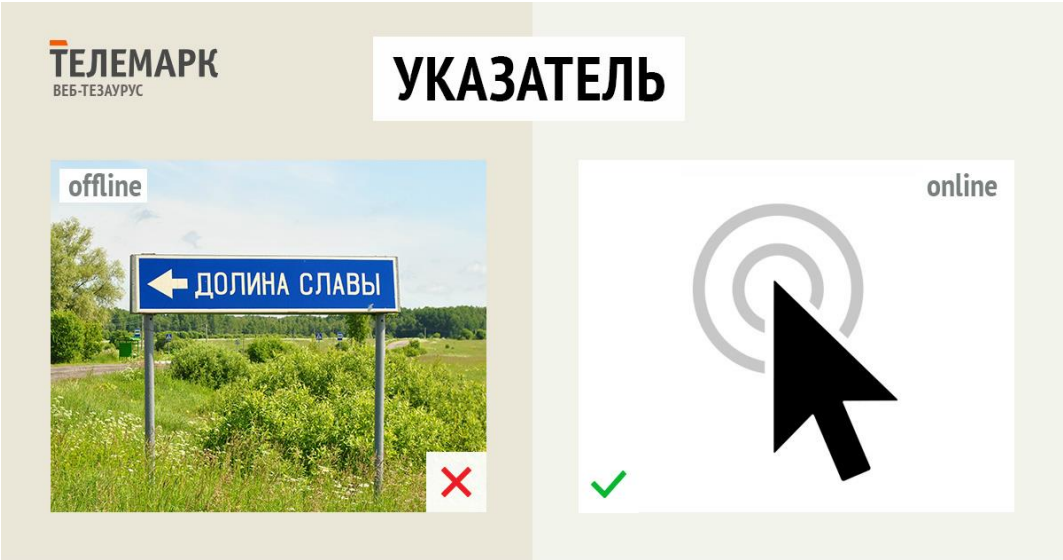


Слои в Photoshop, пожалуй, самая главная палитра программы. Каждый слой в картинке, как и слой в невероятно вкусном торте, наполнен собственным содержанием, сквозь который пробивается содержимое нижних слоев. Слои можно перемещать, чтобы изменить положение картинки, создавая неповторимое изображение.

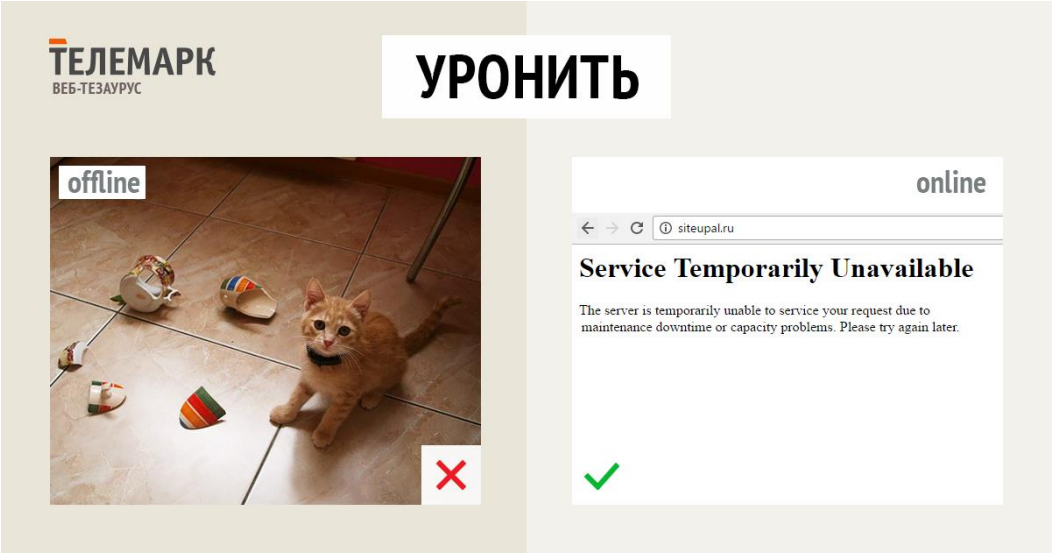
Значок @ используется для обозначения электронной почты. Например, адрес info@telemark-it.ru следует читать как «почтовый ящик с именем info на почтовом сервере telemark-it.ru». Как только не называют этот значок: слоновий хобот, булочка с корицей, улитка, мышонок и прочее. В России большинство предпочитает называть его «собака» или «собачка» (своеобразный эвфемизм).

В реальной жизни счетчик — это устройство, которое «считает» или измеряет какое-либо действие/явление, передавая данные в требуемых единицах. В ИТ счетчики, например, ставятся на сайт, чтобы отслеживать деятельность посетителей (количество переходов, время пребывания на сайте и многое другое).

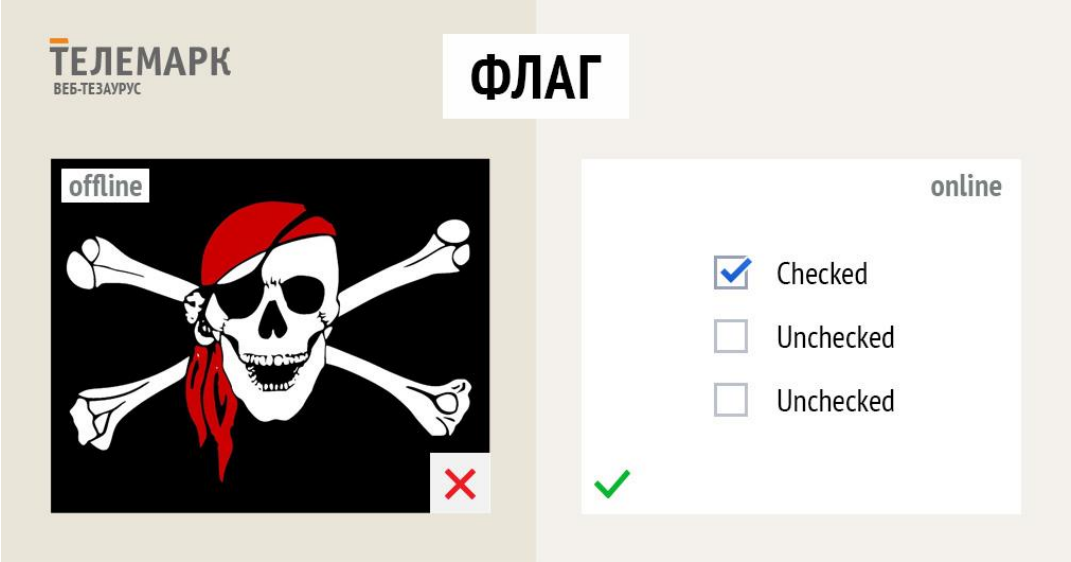
Сэндвич, или гамбургер, (спец.) — это кнопка-иконка в графическом интефейсе пользователя в виде трех горизонтальных линий (как слои сэндвича), которая разворачивает меню страниц или параметров.



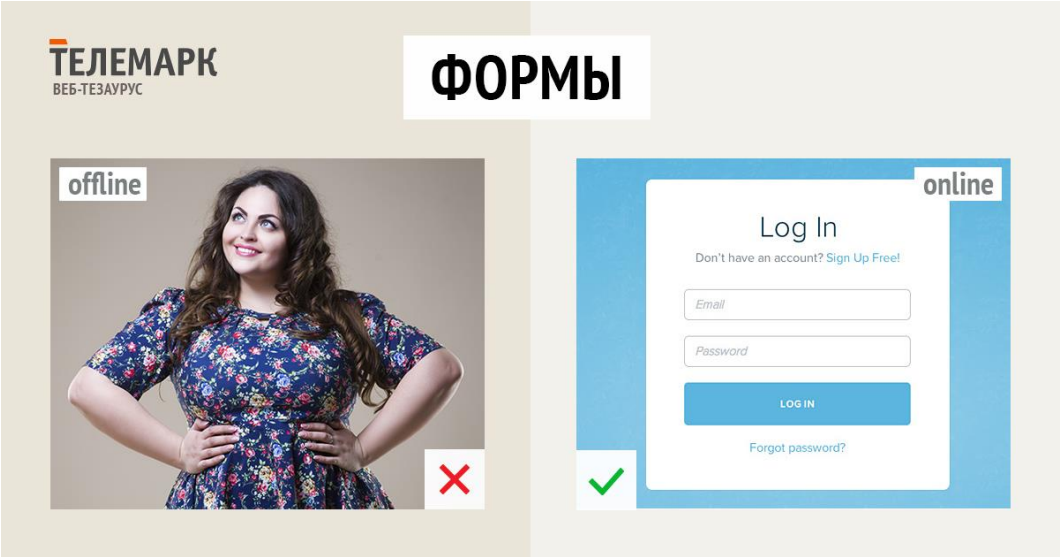
Указатель мыши — «путеводная звезда» наших перемещений на экране. Он повторяет движения мыши, указывая на объекты, с которыми мы хотим взаимодействовать. Стандартный вид указателя — белая стрелка. Но так как это элемент графического интерфейса, вид указателя можно отрисовать (стилизовать) под конкретный проект.



Уронить сервер означает вывести его из строя (а не уронить физически). Обычно это происходит вследствие большой нагрузки на сервер, когда превышено количество запросов или скрипты сайта работают некорректно. При этом в браузере может отдаваться 503 ошибка (Service Temporarily Unavailable).

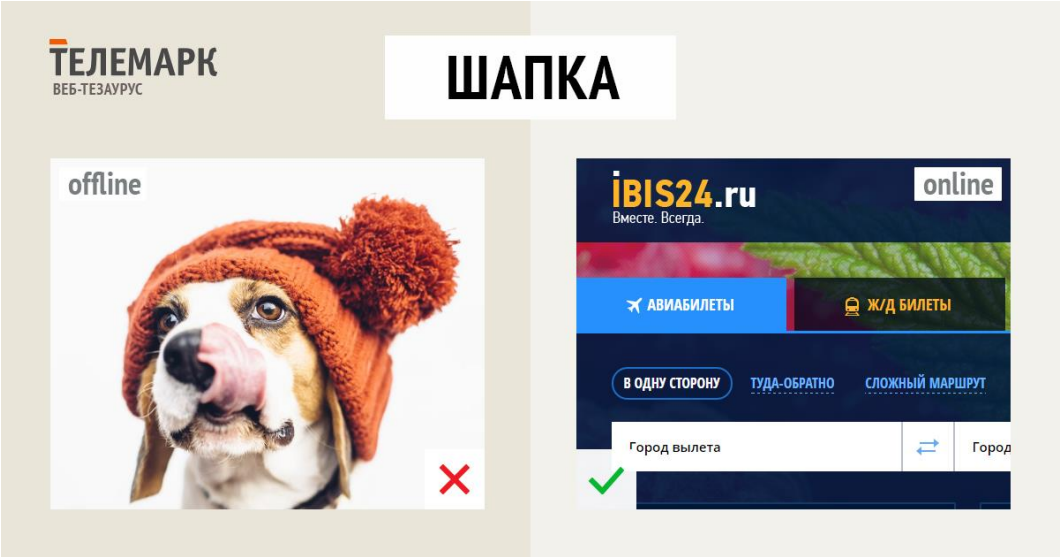


В ИТ флаг относится к кодированному представлению возможных состояний или статусов объектов. Собственно говоря, в реальной жизни флаг призван выделить тот или иной объект (страну, организацию, отрасль, город и т.д.) среди своего типа.



Привлекательные формы не только облегчат жизнь предпринимателям, но и будут способствовать продуктивной коммуникации с посетителями сайта. Главное — не увлекаться: формы должны быть компактными, требовать минимум информации от посетителя и соответствовать всем требованиям политики конфиденциальности.

«Навигация — это наше всё!» К сожалению, Гензель и Греттель ошиблись в выборе материала, и птицы склевали «указатели». Однако их принцип надежно закрепился в IT, и сейчас «хлебные крошки» показывают пользователю путь от корня сайта до текущей страницы.



В отличие от своего брата-омонима ИТ-шапка обязана всегда присутствовать на веб-ресурсе и находиться в верхней части сайта, ее нельзя «снять», даже если «мама разрешила». В шапке сайта присутствует самая важная информация для посетителя сайта, так как это первое, что он видит, когда заходит на сайт. Обычно здесь располагается логотип компании, номера телефонов, адрес и E-mail.

В отличие от русского разговорного эквивалента, означающего «разбираться в чем-то», слово «шарить» в ИТ происходит от английского «share — делиться». Оно стало особо популярным с развитием соцсетей, где можно расшарить картинку или группу, чтобы их увидело большое количество людей.

Настолько вкусное яблоко, что невозможно удержаться и не откусить от него хотя бы кусочек. То же самое можно сказать и об американской корпорации Apple, которая производит одни из самых востребованных ПК, планшетных компьютеров, телефонов, а также программное обеспечение.

Яша и Гоша — так ласково разработчики называют две самые крупные поисковые системы — Яндекс и Гугл. У каждого пользователя свои предпочтения, однако, утверждать можно лишь одно: обе поисковые системы ответят практически на любой вопрос. А иногда и повторят 100500 раз.

ТЕСТОВЫЕ ЗАДАНИЯ

Тест по разделу «Основы технологий Интернет-программирования»

Определите ошибку при вычислении факториала

```
function factorial(N){
  return N<=1?1:N*factorial(N-1);
}
```

неправильно описано условие выхода из рекурсии

✓ ошибок нет

неправильно передается параметр функции на следующий уровень вызова

Определите ошибку

```
<SCRIPT type="text/javascript">
var user_name = prompt ("Введите имя"," ");
document.write("Привет, " + User_name );
</SCRIPT>
```

нельзя использовать пустые строки в команде prompt

✓ нельзя использовать разный регистр — user_name и User_name

ошибки нет

Определите результат работы участка кода

```
<SCRIPT type="text/javascript">
 var result=0;
 var x= 12;
 var y= 5;
 result= x + -y;
 alert(result);
</SCRIPT>
```

выводится -x +-y

✓ выводится 7

выводится 19

Как разместить код JavaScript между тегами Script?

✓ <script type=»text/javascript»> </script>

<script type=text/javascript> </script>

<script «type=text/javascript»> </script>

Определите результат работы участка кода

```
<SCRIPT type="text/javascript">
 var x= 5;
 var y= 2;
 var result=0;
 result= x / y;
```

```
  alert(result);
</SCRIPT>
```

выводится «2/5»

выводится 2

✓ выводится «2.5»

При каких условиях выполнятся операторы в конструкции
if (условие1) { if (условие2) { операторы }}?

если условие2 верно, а условие1 неверно

если условие2 и условие1 неверны

✓ если условие1 и условие2 верны

если условие1 верно, а условие2 неверно

При каких a,b и c выполнится оператор2 в конструкции
if (a == b) { if (c == 10) { оператор1 } else { оператор2 }}?

a=5, b=5, c=10

a=15, b=10, c=10

✓ a=10, b=10, c=5

Что такое конструкция if/else?

выполнение некоторого действия , многократность процедуры зависит от выполнения некоторого условия

выбор дальнейшего хода программы в зависимости от выполнения некоторого условия

✓ выбор дальнейшего хода программы в зависимости от выполнения или невыполнения некоторого условия

При каких a, b и c выполнятся операторы в конструкции :

```
if (a == c)
{
 if  (c == 10)
  {
   if (b == 5)
     { операторы
     }
  }
}?
```

a = 10, c = 10, b = 8

✓ a = 10, c =10, b = 5

a = 10, c = 5, b = 5

a =5, c =10, b =5

Укажите недопустимые записи конструкции if

✓ if (условие) { if (условие) { операторы }

if (условие) { операторы }
✓ if (условие) { if (условие) { if (условие) } { операторы } }
✓ if (условие) операторы

Укажите недопустимые записи конструкции if/else
✓ if (условие) { операторы } else (условие) { операторы }
✓ else (условие) { операторы }
if (условие) { операторы } else { операторы }

Какой тип имеет свойство elements объекта Form?
такого свойства нет
скаляр
✓ массив

Каково будет значение переменной i после выполнения приведенного JavaScript кода?

```
var i = 1;
while (i <= 2)
{
  i+=5;
}
```

0
1
✓ 6

Чем задается поле ввода на форме?
✓ посредством одинарного тега <INPUT TYPE=»text»>
посредством одинарного тега <INPUT TYPE=»button»>
посредством одинарного тега <INPUT TYPE=»reset»>

Что называется циклом?
множество данных, размещенных в одной переменной
✓ средство организации повторяющихся операций
ошибка, при которой программа повторяется бесконечное количество раз

Что задает условие i<4 в синтаксисе оператора for?
задает начальное значение переменной, управляющей циклом
✓ условие, определяющее, сколько в цикле будет повторений
задает шаг приращения переменной, управляющей циклом

Найдите ошибку

```
<script>
function newcolor(color)  {  document.bgColor=color }
```

```
</script>
<FORM>
 <INPUT TYPE="button" VALUE="Голубой" onClick="newcolor('lightblue')">
 <INPUT TYPE="button" VALUE="Розовый" onClick="newcolor('pink')">
</FORM>
```

нельзя описывать функцию одной строкой
неверно передается параметр функции
✓ ошибки нет

В какой момент происходит событие onLoad?

✓ после завершение загрузки элемента
перед началом загрузки элемента
после начала загрузки элемента, до завершения

Определите, что произойдет в результате выполнения приведенного ниже кода?

```
<script>
function doit()
{
 var greeting="Привет, ";
 alert(greeting +  " " + document.myform.lname.value);
}
</script>
<FORM NAME="myform" action=""> Ваше имя:
 <INPUT TYPE="text" NAME="lname"><p>
 <INPUT TYPE="button" VALUE="Отправить" onClick="doit()">
</FORM>
```

✓ выводится сообщение «Привет» и имя, введенное пользователем
ошибка. Неправильно переданы параметры функции
выводится сообщение «Привет Ваше имя:»

Существует ли полиморфизм переменных в JavaScript?

✓ существует
не существует
полиморфизм переменных корректно обрабатывается только IE 6.0

Каков результат работы данного скрипта

```
<script language="javascript">
	mas=new Array(2);
	mas[3]=5;
	alert(mas[3]);
</script>
```

окно предупреждения с надписью «undefined»

✓ окно предупреждения с цифрой «5»
скрипт выдаст сообщение о синтаксической ошибке

Укажите длину массива после исполнения следующего кода:
a=new Array(2);
a[1]=null;
a.length == 0
a.length == 1
✓ a.length == 2

Сколько значений может иметь массив?
массив не имеет значений
одно
✓ множество

Какая функция переводит строку в верхний регистр?
toLowerCase
✓ toUpperCase
Up

Для чего предназначена функция substr(a,b)?
✓ выдает подстроку, длиной b, начиная с индекса a
переводит строку в нижний регистр
сообщает длину строки

Как можно изменить цвет фона в 5-й строке таблицы?
table.row(4).bgColor = «red»
✓ table.rows[4].bgColor = «red»
table.row.4.bgColor = «red»

Можно ли обратиться к элементу страницы, не как к части коллекции, а напрямую?
только в Internet Explorer
нельзя
✓ можно, используя прямое обращение по идентификатору элемента

Найдите правильное выражение:
document.img(«image1»)
✓ document.images[«image1»]

Как можно изменить цвет фона во второй ячейке 5-й строки таблицы?
table.row(5).cells(2).bgColor = «green»
✓ table.rows[4].cells[1].bgColor = «green»
table.row.5.cells.2.bgColor = «green»

Свойством какого объекта является status?

✓ window
frame
document

На что указывает аргумент метода clearTimeout()?

на стек
✓ на созданный поток
на JavaScript-код

Что содержится в свойстве vlinkColor объекта document?

✓ цвет посещенной гиперссылки
цвет непосещенной гиперссылки
цвет гиперссылки, над которой находится указатель мыши

Будет ли выдана ошибка JavaScript для document.links[0][«search»]=»test»;?

будет
✓ нет

Что выводится в результате работы участка кода?

```
<SCRIPT LANGUAGE="javascript">
  var a = document.referer;
  document.write(a)
</SCRIPT>
```

✓ адрес страницы, с которой пришел пользователь
символ «a»
URL страницы

Обладает ли фрейм свойством статуса?

✓ да
нет

В каком случае переменная или функция одного объекта может изменяться любой функцией или методом?

✓ если описать переменную или функцию как Static
если описать переменную или функцию как Privileged
если описать переменную или функцию как Public

Как называется код родительской функции?

✓ конструктор
базовый код
деструктор

Какая функция служит для задания параметров соединения?

✓ open

XMLHttpRequest

config

Каким образом Internet Explorer 6 взаимодействует с AJAX?

✓ посредством объекта ActiveX XMLHttp

посредством метода XMLHttpRequest

по умолчанию

Какой метод AJAX чаще используется при отправке данных на сервер?

LOAD

GET

SEND

✓ POST

Определите, в какой строке допущена ошибка?

(1) <SCRIPT LANGUAGE="javascript">

(2) document.write("текст для страницы"

(3) 😉

(4) </SCRIPT>

✓ в строке (3)

нет ошибки

в строке (2)

Что такое ошибка времени выполнения?

команда, на которую указывает сообщение, не укладывается в отведенный период времени

✓ команда, на которую указывает сообщение, не укладывается в логическую последовательность

команда, на которую указывает сообщение, неверно записана

Когда появляется ошибка определения переменных?

когда вызывается неправильная команда

✓ когда используются переменные, которые еще не определены

когда команда, на которую указывает сообщение, не укладывается в отведенный период времени

Какие ошибки допущены в приведенном ниже коде? (считать, что документ содержит только одну форму)

```
<SCRIPT TYPE="text/javascript">
function doit()
{
 alert("Длина вашего имени "+document.myform.myname.value.length);
```

```
}
</SCRIPT>
<FORM>
  <INPUT TYPE="text" name="myname">
  <INPUT TYPE="button" value="clickMe" onClick="doit();">
</FORM>
```

✓ не указано имя формы, следовательно, невозможно обратится к полю myname по иерархии document.myform.myname

неправильно объявлена функция doit

синтаксическая ошибка в методе alert

Определите ошибку

```
if confirm("Уверены, что хотите посетить INTUIT?")
{
  parent.location='http://www.intuit.ru/';
}
else
{
  alert("Тогда оставайтесь");
}
```

неправильно записан метод confirm

ошибки нет

✓ неправильно записано условие

Что называется стеком?

✓ область памяти, в которой сохраняются копии всех переменных на каждом уровне вызова рекурсивной функции

ошибка, возникающая при исчерпании памяти, выделяемой браузером для исполнения JavaScript-сценария

вызов функции из той же самой функции

Определите ошибку при вычислении факториала

```
function fact(n)
{
  var f=1
  f=n*fact(n-1)
  return f
}
```

ошибок нет

неправильно передается параметр функции на следующий уровень вызова

✓ неправильно описано условие выхода из рекурсии

В каком случае возникает ошибка переполнения стека?

при бесконечном цикле
при описании массива слишком большого размера
✓ при исчерпании памяти, выделяемой браузером для исполнения рекурсивной функции JavaScript-сценария

Как называется ситуация, когда функция вызывает саму себя?
ошибка времени исполнения
✓ рекурсия
цикл

Какая команда выводит окно с сообщением и полем ввода, и позволяет ввести данные в переменную?
readln
✓ prompt
insert

Что происходит при вызове метода setTimeout(«test()»,1000)?
реализуется выполнение test() в цикле
✓ создается новый поток для функции test()
функция test() вызывается рекурсивно через секунду

ЛИТЕРАТУРА

1. Круг С. Не заставляйте меня думать. – М.: Эксмо, 2017
2. Купер А., Рейман Р., Кронин Д., Кристофер Н. Интерфейс. Основы проектирования взаимодействия. 4-е изд. – СПб.: Питер, 2018
3. Кришна Г. Хороший интерфейс – невидимый интерфейс. – СПб.: Питер, 2016
4. Сидерхолм Д. CSS3 для веб-дизайнеров. .– М.: Манн, Иванов и Фербер, 2015
5. Кит Д. HTML5 для веб-дизайнеров. .– М.: Манн, Иванов и Фербер, 2015
6. КузнецовМ., Симдянов И. Самоучитель PHP 7. – СПб.: БХВ-Петербург, 2018
7. Грачев А. Создаем свой сайт на WordPress быстро, легко и бесплатно. 2-е издание. – СПб.: Питер, 2015