Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

Факультет **Программной инженерии и компьютерной техники**

Образовательная программа **Разработка программно-информационных систем**

Направление подготовки (специальность) **09.03.04- Программная Инженерия**

О Т Ч Е Т

О ПРОХОЖДЕНИИ УЧЕБНОЙ ПРАКТИКИ (ПРАКТИКИ ПО ПОЛУЧЕНИЮ ПЕРВИЧНЫХ ПРОФЕССИОНАЛЬНЫХ УМЕНИЙ, В ТОМ ЧИСЛЕ ПЕРВИЧНЫХ УМЕНИЙ И НАВЫКОВ НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ ДЕЯТЕЛЬНОСТИ)

Тема задания: Разработка web-интерфейса метеостанции для испытательного полигона ОКБ “АВГИТ”

Обучающийся **Сенмас Атаулла** **Группа №** **P3319**

Руководитель практики от профильной организации: **Графкин Алексей Викторович, ООО «ОКБ Авгит», главный специалист по развитию персонала.**
Руководитель практики от университета: **Маркина Татьяна Анатольевна, старший преподаватель, факультет.**

Практика пройдена с оценкой _____

Подписи членов комиссии:

_____ _____
(подпись)

_____ _____
(подпись)

_____ _____
(подпись)

Дата _____

Санкт-Петербург **2018/2019**

СОДЕРЖАНИЕ

Введение

В настоящее время трудно представить себе организацию, не имеющую собственного сайта. Свою страничку в Интернете имеют торговые фирмы и промышленные предприятия, больницы и школы, органы власти и спортивные команды, институты и военкоматы, и т. д. и т. п. Более того – никого уже не удивляет наличие собственного сайта у частных лиц. Причем этим могут похвастаться не только известные и состоятельные люди (политики, писатели, спортсмены, и т.д.), но и простые граждане (студенты, домохозяйки, инженеры, врачи, даже школьники).

Многие люди все чаще задумываются о том, как самостоятельно создать сайт и разместить его в Интернете. Кто-то желает заняться этим просто из «спортивного интереса», кому-то необходим свой веб-ресурс для саморекламы или общения, а кому-то просто начальство велело: мол, надо сделать или подкорректировать сайт фирмы (отдела, подразделения, филиала и т.п.) – и точка. В любом случае, с проблемой создания и администрирования веб-ресурса зачастую сталкиваются люди, не имеющие ни малейшего представления о том, как это делается.

Бытует мнение, что разработкой сайтов и веб-программированием могут заниматься только люди, обладающие специальными знаниями и соответствующим уровнем подготовки. Но это верно лишь отчасти: как показывает практика, любой человек, хоть немного знакомый с компьютером, способен самостоятельно не только создать вполне приличный веб-ресурс, но и сделать его посещаемым. И для этого совсем необязательно ходить на дорогостоящие курсы или обкладываться со всех стороны толстыми учебниками с «навороченными» программными кодами и прочим малопонятным содержимым: всю необходимую информацию вы найдете в предлагаемой книге.

Вначале мы поговорим о том, с чего начинается процесс создания сайта: разработка концепции, выбор хостинга, и др. После этого мы расскажем о том, как создавать веб-страницы с помощью языка программирования HTML.

Информационное наполнение (контент) сайта

Ключевым элементом любого сайта является его информационное наполнение – контент. Ведь посетители приходят на веб-ресурс в первую очередь за информацией, поэтому сайт должен отличаться грамотным, тщательно выверенным и профессионально написанным контентом. При этом представленный на сайте текст должен не только доносить до посетителей информацию, но и заинтересовывать их, а также мотивировать к сотрудничеству, покупке, партнерству или хотя бы знакомству. Исключением в данном случае могут являться лишь специфические веб-ресурсы – вроде сервиса бесплатной почты.

При планировании информационного наполнения сайта важно определить тип и формат представления данных. Будет ли контент представлять собой только текст, либо в него будут добавлены графические объекты, звуковые файлы, Flash-элементы и т.п.? Какие страницы будут включены в состав веб-ресурса – только статические HTML-страницы или динамические, генерируемые на основании сформированных пользователем запросов? Отметим, что характерной особенностью динамичных страниц является то, что они предоставляют посетителю именно те сведения, которые его интересуют, причем именно в той форме, которая для данного посетителя является наиболее удобной для восприятия.

Понятие статической и динамической веб-страницы

Ранее мы уже кратко говорили о том, что представляют собой статические и динамические веб-страницы. В данном же разделе мы рассмотрим этот вопрос более детально.

Название статической страницы говорит само за себя: на такой странице представлена статичная, постоянная и не изменяющаяся информация. Вернее, изменить-то ее можно, но для этого необходимо внести соответствующие корректировки в программный код страницы.

Обычно файл статической страницы имеет HTML -формат. Веб-разработчик пишет HTML,-код, формируя при этом информационное наполнение сайта, файлу присваивается имя, после чего страница загружается на веб-сервер и становится доступной пользователям Интернета. Например, на корпоративном сайте статическая страница может содержать сведения о об истории компании, основных направлениях ее деятельности, и т. п.

Общие сведения о языке HTML

Расшифровывается аббревиатура HTML следующим образом: Hyper Text Markup Language, что в переводе на русский язык означает «гипертекстовый язык разметки». Иначе говоря HTML – это язык программирования, используемый для описания представленной на веб-странице информации. В состав этого языка входят специальные теги (команды), которые предназначены для создания различных эффектов, применения форматирования (например, курсивное или полужирное начертание), и т. п.

Возможно, у читателя возникнет вопрос: для чего, собственно, изучать язык HTML, если в настоящее время существует немало программных средств, специально предназначенных для создания страниц и сайтов (кстати, некоторые из них описываются далее в этой книге)?

Нужно ли тратить время на изучение языка программирования, когда сайт можно легко сконструировать и без этих знаний?

Действительно, современные программные продукты позволяют создавать веб-страницы даже без минимальных знаний азов программирования. Однако если вы всерьез планируете заняться веб-разработкой, а также если впоследствии вам придется заниматься обслуживанием и администрированием сайта – хотя бы элементарное понятие о структуре и использовании языка HTML иметь необходимо.

Чтобы было понятнее, приведем аналогию с автомобилем. Когда человек учится в автошколе, ему преподают не только Правила дорожного движения, но и читают курс устройства автомобиля. Ведь даже если человек не планирует заниматься самостоятельно ремонтом и обслуживанием машины, уж коль он сел за руль – он должен иметь хотя бы общее представление об устройстве того, чем он управляет. По крайней мере, он будет знать, что если во время движения машину «потянуло» в сторону – видимо, у нее спустило колесо, если застучал мотор – нужно срочно остановиться, а если машина заглохла – возможно, у нее кончился бензин, и т. д.

То же самое касается веб-разработки. Человек, который занимается созданием и администрированием сайтов, должен хотя бы в общих чертах понимать, что, например, полужирное начертание можно получить только после применения соответствующих тегов, а если веб-страница отображается какими-то непонятными символами – скорее всего, нужно разобраться с кодировкой символов, и т. д.

мы познакомимся с языков HTML именно на том минимальном уровне, который необходим каждому веб-разработчику, независимо от того, как он намеревается создавать и администрировать сайты – вручную или с помощью специально предназначенных программных средств.

Если бы не было языка HTML, все использование Интернета свелось бы к безликим текстовым посланиям, не имеющим какого-то форматирования, яркого красочного оформления, мультимедиа, эргономики и дизайна. Однако самой главной функцией языка программирования HTML является возможность связывать веб-страницы между собой с помощью гиперссылок. Эти гиперссылки могут находиться, например, на навигационных инструментах веб-ресурсов, либо следовать прямо в тексте. Например, если навигационная панель сайта содержит ссылку Услуги.

HTML «в действии», или Как просмотреть исходный код страницы

Не все пользователи Интернета знают, что при желании можно просмотреть исходный HTML-код любой веб-страницы. Кстати, многие именно с этого начинают свой первые опыты в сфере веб-разработки: посмотрев «реальное» отображение страницы, а после этого изучив ее HTML-код, можно получить общее представление о структуре, синтаксисе и приемах использования этого языка.

Функциональные возможности практически любого популярного Интернет-обозревателя (Internet Explorer, Mozilla Firefox, и др.) предусматривают просмотр HTML-кода текущей страницы, который обычно отображается в отдельном окне. Рассмотрим на конкретном примере, как это делается.

Откроем в Интернет-например, сайт «Эффекты дождя и воды» – https://se.ifmo.ru/~s228630/погода/demo/index.htm (рис. 1.1).

Основные теги языка HTML

Здесь мы познакомимся с наиболее востребованными тегами языка программирования HTML. Отметим, что многие теги имеют свои атрибуты, о которых также будет рассказано в данном разделе. Мы будем рассматривать все теги (даже те, с которыми уже познакомились) в алфавитном порядке.

ВНИМАНИЕ

Не забывайте, что многие теги HTML-языка являются парными – например,

<!DOCTYPEhtml>{<html><head>{<meta><title></title><link></head>}{<body>{ <header></header><h1></h1><h2></h2><h3></h3><h4></h4><h5></h5><nav></nav><canvas></canvas> } <form></form><input type="" name=""><img src=""><iframe src=""></iframe></body>}</html>, и т. д.

Использование HTML-интерфейса.

Нередко приходилось задаваться вопросом о том, какой интерфейс использовать в программе, для неискушенного пользователя, с тем, чтобы обеспечить простоту и удобство его использования. На моем первом рабочем месте начальник отдела так сформулировал эту задачу: - «Надо написать программу, которой сможет пользоваться человек все навыки, которого заключаются в умении пользоваться мышью».

Казалось бы, что задача невыполнима, однако такой интерфейс существует и реализован он в каждом браузере. Не случайно HTML-интерфейс широко используется в справочных руководствах к программам и все чаще внедряется в клиентское ПО, ведь использование гиперссылок настолько очевидно, что даже самый неподготовленный пользователь не требует дополнительных пояснений.

HTML-интерфейс, наряду со своей **простотой**, обладает другими свойствами, которые делаю его использование в приложениях для Windows все более привлекательным.

Гибкость. За счет использования CSS возможно создание индивидуального интерфейса для каждого пользователя, причем не только за счет выбора цветовых схем, шрифтов и наборов изображений. Возможна полная модернизация внешнего вида программы, например,

преобразование списка ссылок в TabControl, изменение относительного положение управляющих элементов, сокрытие информационных блоков без ущерба для внешнего вида программы.

Модифицируемость. Интерфейс программы может быть изменен без перекомпиляции исходных кодов, а грамотный пользователь может осуществить его настройку самостоятельно.

Интерактивность. Использование JavaScript позволяет организовать взаимодействие с пользователем в стиле столь популярной нынче технологии AJAX. Отображая необходимую информацию только по запросу, что особенно важно при ее больших объемах.

Простота. Если выше речь шла о простоте использования, то сейчас стоит упомянуть об относительной легкости разработки. По моему глубокому убеждению, которое подкреплено изрядным практическим опытом реализация элемента управления на JavaScript требует гораздо меньшего времени, чем при использовании MFC. Кроме того, в сети можно отыскать немало готового кода, который часто является свободно распространяемым и не защищены вообще никакими лицензиями.

Красота. Все попытки придумать синоним для этого слова, потерпели неудачу, а все потому, что именно оно в полной мере отражает тот факт, что web-интерфейс гораздо проще сделать красивым, нежели традиционный. Для того чтобы убедиться в этом достаточно побродить по просторам Интернета и учесть тот факт, что при использовании HTML-интерфейса в собственной программе нет необходимости заботиться о размере загружаемой страницы и изображений.

Пора переходить от слов к делу

Для включения в свою MFC-программу HTML-интерфейса, необходимо использовать класс **CHtmlView**, который предоставляет функциональность веб-браузера. Теперь разберем, тот минимум функций, который необходимо реализовать для взаимодействия основной программы с интерфейсом. В конце страницы размещена ссылка на архив с примером, а потому я остановлюсь лишь на некоторых интересных моментах.

В примере реализован класс **CJSHtmlView** унаследованный от вышеупомянутого - **CHtmlView**, который в дополнение к функциям своего предка реализует две дополнительные возможности: замещение текста HTML-элементов и запуск JavaScript кода.

Первая из них реализуется с помощью функции **CJSHtmlView:: SetHtmlElementValue(CString sId, CString sValue)**, которая ищет объект на HTML странице по его идентификатору и в случае успеха заменяет его содержимое строкой sValue, используя свойство innerHTML.

Думаю, что текст данной функции не вызовет никаких сложностей, а потому перейду ко второй возможности – взаимодействие с JavaScript

Часть кода html,На сайте погода

```
<!DOCTYPE html>
<html lang="en" class="no-js">
<head>
	<meta charset="UTF-8" />
	<meta http-equiv="X-UA-Compatible" content="IE=edge">
	<meta name="viewport" content="width=device-width, initial-scale=1">
	<title>погода</title>
	<meta name="description" content="Some WebGL experiments with raindrop effects" />
	<meta name="keywords" content="webgl, raindrops, effect, rain, web, video, background" />
```

```
        <meta name="author" content="Lucas Bebber for Codrops" />
        <link rel="shortcut icon" href="favicon.ico">
        <link rel="stylesheet" type="text/css" href="css/normalize.css" />
        <link
href='https://fonts.googleapis.com/css?family=Roboto:400,100,300,700,500,900'
rel='stylesheet' type='text/css'>
        <link rel="stylesheet" type="text/css" href="css/demo.css" />
        <link rel="stylesheet" type="text/css" href="css/style1.css" />
        <!--[if IE]>
                <script
src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
                <![endif]-->
                </head>

src="https://www.youtube.com/embed/SynzKC4fWp0" frameborder="0"
    /



       <div id="private">
          <center>
       <p1>Студенты:Университет ИТМО,Группа:P3319.</p1><br>
     <a href="https://vk.com/idsenmas28630" target="_blank">Сенмас Атаулла ,</a>
                <a href="https://vk.com/wahajzada" target="_blank">Вахадж Зада
М.Амин.</a><br>
                <p1>Руководитель практики:Маркина Т.А.,Графкин А.В.</p1><br>
                <p1>Тема задания:создать сайт для метеостанции(влажность,
температуру, вид ) С использованием языки html,css и
javascript(интерфейс).</p1>
                </center>
            </div>
<a class="current-demo" href="index.html">погода</a>
                                <a href="index2.html">вода</a>
                                <a href="index3.html">видео</a>
                                <a href="index4.html">кафе и Дождливый</a>
```

```
                    <a href="https://drive.google.com/drive/my-drive">Скачать код файла(погода)</a>
                </nav>
            </header>
```

Общие сведения о языке CSS

CSS (Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

Возможности языка CSS

- Раздельное хранение представления и документа позволяет стилизовать этот документ для различных устройств, включая, монитор принтер, проектор и даже портативные устройства.
- Раздельное хранение представления и документа означает уменьшение размера документа, что, в свою очередь, ускоряет загрузку и отображение страницы, доставляя удовольствие посетителям.
- Язык CSS позволяет управлять как одним документом, так и миллионами документов. Для внесения изменения потребуется модифицировать необходимый стиль в одном CSS-файле, и это изменение автоматически отразится на всех связанных документах. В языке HTML это сделать невозможно.
- CSS-документы кэшируются. Это означает, что они загружаются в память браузера только один раз. При перемещении по сайту браузеру никогда не приходится заново интерпретировать стили. В результате мы получаем плавные переходы от страницы к странице и быструю загрузку страниц, что всегда является конечной целью.
- Отделив представление от структуры и содержимого, легко добиться доступности документа. Документы, в которых не используются сложные таблицы и большое количество элементов языка HTML, отвечающих за представление, являются более доступными, чем те документы, у которых перечисленные свойства присутствуют.

Правила языка CSS

Правила языка CSS состоят из селектора и блока объявлений. Селектор — это код элемента языка HTML, к которому будет применено правило стиля. Блок объявлений помещается в фигурные скобки, и, в свою очередь, состоит из одного или более объявлений, разделённых знаком точка с запятой. Каждое объявление представляет собой сочетание свойства CSS и значения, разделённых знаком двоеточие. Селекторы могут группироваться в одной строке через запятую. В таком случае свойство применяется к каждому из них.

```
селектор, селектор {
   свойство: значение;
   свойство: значение;
}
```

Рассмотрим на примере:

```
html{height:100%;}
```

```
body {
background:#0D343A;
background:-webkit-gradient(linear,0%0%,0%100%,from(rgba(13,52,58,1)  ),
to(#000000)  );
background: -moz-linear-gradient(top, rgba(13,52,58,1) 0%, rgba(0,0,0,1) 100%);
overflow:hidden;}
```

Типы таблиц стилей

- **Стиль браузера** — это стандартная таблица стилей, используемая браузером. Если никакие правила стиля не объявлены, применяются эти стандартные стили.
- **Стиль пользователя** — пользователь может написать свою таблицу стилей и переопределить любые стили, созданные вами, изменив настройки браузера. Такой подход используется редко, но может оказаться полезным для людей с ограниченными возможностями, например с плохим зрением. В данном случае пользователь создаст стили с высококонтрастными шрифтами больших размеров, которые переопределят ваши стили.
- **Стиль автора** — стиль, который добавляет к документу его разработчик

Способы добавления стилей

- **Встроенный стиль** — стиль, который определяется непосредственно в теге и применяется с использованием **атрибута style**. Такой подход полезен для стилей, единовременно применяемых к одному элементу, однако он не считается идеальным.

```
<p style="font-size: 21px; color: green;">Текст</p>
```

Внедренный стиль — этот стиль управляет представлением одного документа и размещается внутри тега-контейнера style в разделе документа head

```
<head>
.....
  <style>
    body {color: red;}
  </style>
</head>
```

Связанный стиль — это отдельный файл с расширением .css, в котором размещаются все CSS-правила (но без тегов языка HTML). Любой HTML-файл, к которому нужно применить стили, описанные в этой таблице, можно связать с ней, используя тег link в разделе документа head.

```
<head>
.....
  <link rel="stylesheet" href="style.css">
```

</head>

Импортированный стиль — этот тип похож на связанные стили, однако позволяет импортировать стили в связанную таблицу стилей или непосредственно в документ. Он полезен для реализации обходных путей и для управления множеством документов. Импортированный стиль (как и связанный) представляет собой код, записанный в отдельном файле с расширением .css. Импортировать содержимое CSS-файла можно в связанную таблицу стилей или непосредственно в html-документ с помощью правила, которое указывается в начале связанной таблицы стилей или блока <style> html-документа соответственно.

@import url("style.css");

Каскад

Каскад определяет способ применения правил, в случае с типами таблиц стилей:

- **Пользовательский** стиль переопределяет все остальные стили.
- **Встроенный стиль** превосходит внедренный, связанный и импортированный стили.
- **Внедренный стиль** имеет преимущественное значение перед связанным и импортированным стилями.
- **Связанные и импортированные стили** рассматриваются, как равные по значимости, и применяются везде, где не были применены другие типы таблиц стилей.
- **Стиль браузера** используется только в том случае, когда для данного элемента не было предоставлено ни одного стиля.

Например, в связанной таблице стилей (файле my_style.css) содержатся правила: первое определяет цвет абзаца — зеленый, второе задает цвет заголовка — желтый:

```
p {color: green;}
h1 {color: yellow;}
```

В html-документе используются три таблицы стилей: связанная, внедренная и встроенная. Во внедренном стиле определен красный цвет абзаца, а во встроенном (встроенная таблица применяется к первому абзацу) — синий.

```
<!DOCTYPE html>
<HTML>
 <HEAD>
  <META charset="utf-8">
  <LINK rel="stylesheet" href="my_style.css">
  <STYLE>
   p {color: red;}
  </STYLE>
 </HEAD>
 <BODY>
 <H1>Связанный стиль</H1>
  <P style="color: blue">Встроенный стиль.
  <P>Внедренный стиль.
```

```
 </BODY>
</HTML
```

Использование CSS-интерфейса.

CSS имеет новые возможности пользовательского интерфейса, такие как изменение размеров элементов, контуры, и проклейки коробки.

В этой главе вы узнаете о следующих свойствах пользовательского интерфейса:

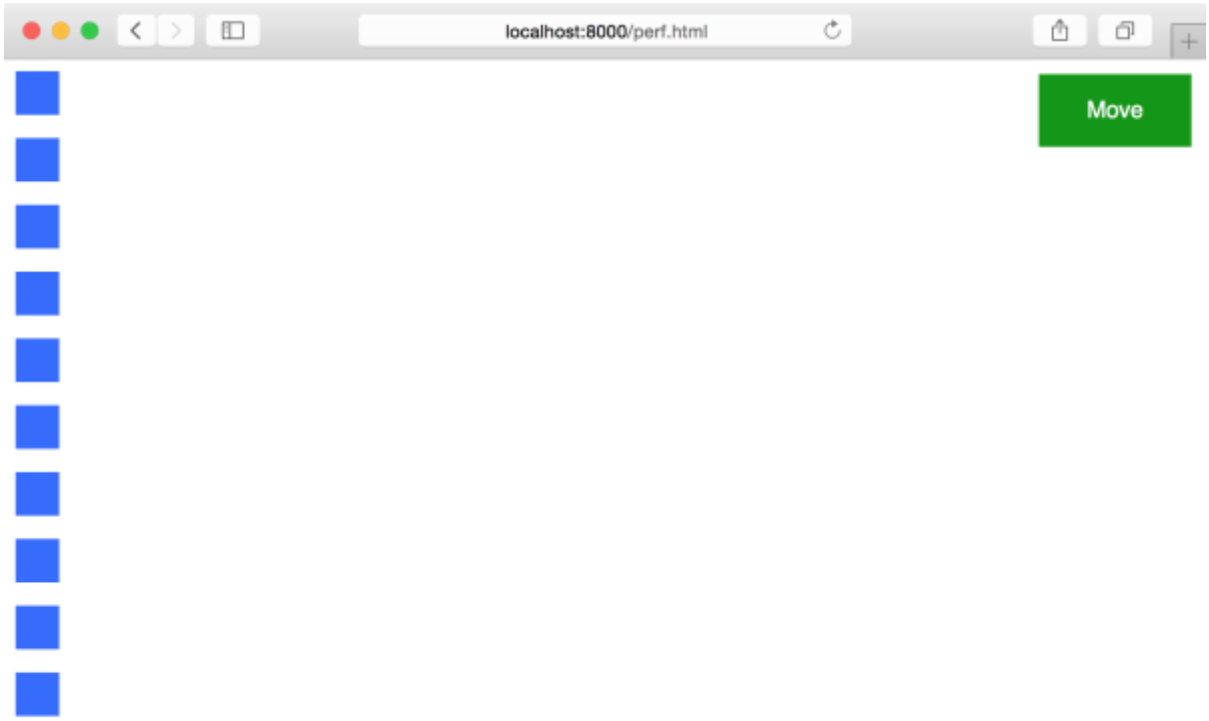
свойства CSS и значения, запускающие перестройку, довольно сильно влияют на производительность. Они могут замедлить адаптивность интерфейсов (рендеринг страницы, плавность анимации и производительность во время прокрутки страницы). Спад особенно заметен на слабых устройствах типа смартфонов и смарт тв.

Для разбора на практике мы сравним два стандартных документа: пример A и B. В обоих случаях будем двигать несколько div'ов из X-положения (0) в X-положение (1000). В обоих примерах будут использоваться CSS-переходы. В примере A мы будем анимировать свойство left. В примере B мы будем анимировать свойство transform с помощью трансформаций.

Разметка в обоих примерах одинаковая (результат можно посмотреть на Рисунке 1.2):

```
  <html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>Performance example</title>
    <style type="text/css"
     > /* CSS will go here. */
    </style>
 </head>
 <body>
  <button type="button" id="move">Move</button>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>
<div></div>

<script type="text/javascript" src="toggle-move-class.js"> ↵</script>
</body>
</html
```



Часть кода CSS,в сайт погода

```
@font-face {
	font-family: 'icomoon';
	src: url('../fonts/meteocons/icomoon.eot?26rom9');
	src: url('../fonts/meteocons/icomoon.eot?26rom9#iefix') format('embedded-opentype'), url('../fonts/meteocons/icomoon.ttf?26rom9') format('truetype'), url('../fonts/meteocons/icomoon.woff?26rom9') format('woff'), url('../fonts/meteocons/icomoon.svg?26rom9#icomoon') format('svg');
	font-weight: normal;
	font-style: normal;
}.icon {
	font-family: 'icomoon';
	speak: none;
	font-style: normal;
	font-weight: normal;
	font-variant: normal;
	text-transform: none;
	line-height: 1;
	/* Better Font Rendering =========== */
```

```
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
}.icon--radioactive:before {
        content: "\e905";
}.icon--sun:before {
        content: "\e900";
}.icon--drizzle:before {
        content: "\e902";
}.icon--rainy:before {
        content: "\e903";
}.icon--storm:before {
        content: "\e906";
}.slideshow {
        position: relative;
        height: 100vh;
        width: 100vw;
        background: url(../img/city.jpg) no-repeat center top;
        background-size: cover;
        overflow: hidden;
}.slideshow::before {
        content: '';
        position: absolute;
        left: 0;
        top: 0;
        width: 100%;
        height: 100%;
        background: rgba(255, 255, 255, 0.2);
}.slide {
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        text-align: center;
        padding: 35vh 0 0 0;
```

```
        pointer-events: none;
        line-height: 1;
        font-family: "Roboto", sans-serif;
}.slide--current {
        pointer-events: auto;
}.slideshow__nav {
        text-align: center;
        width: 100%;
        display: -webkit-flex;
        display: flex;
        -webkit-flex-wrap: wrap;
        flex-wrap: wrap;
        -webkit-justify-content: center;
        justify-content: center;
        -webkit-align-items: center;
        align-items: center;
        position: absolute;
        bottom: 28vh;
}.nav-item {
        display: block;
        padding: 1em;
        color: #252445;
        -webkit-transition: color 0.3s;
        transition: color 0.3s;
}.nav-item:hover,
.nav-item:focus,
.nav-item--current {
        color: #fff;
}.nav-item .icon {
        font-size: 2em;
}.nav-item span {
        display: block;
        margin: 0.25em 0 0 0;
        font-weight: 700;
}
```

```
/* Content */
.slide__element {
	opacity: 0;
	color: #252445;
	-webkit-transform: translate3d(0, 50px, 0);
	transform: translate3d(0, 50px, 0);
	-webkit-transition: -webkit-transform 0.3s, opacity 0.3s;
	transition: transform 0.3s, opacity 0.3s;
}.slide--current .slide__element {
	opacity: 1;
	-webkit-transform: translate3d(0, 0, 0);
	transform: translate3d(0, 0, 0);
}.slide__element--temp {
	font-size: 8em;
	font-weight: 700;
	margin: 0 0 0.1em;
}.slide__element--temp small {
	font-size: 0.25em;
}.slide__element--info {
	font-size: 2em;
}.slide__element--date {
	font-size: 1em;
	font-weight: 700;
	margin: 0 0 1em;
}@media screen and (max-height: 39em) {
	.slideshow {
		font-size: 75%;
	}.slide {
		padding-top: 35vh;
	}.slideshow__nav {
		bottom: 6em;
	}
}
```

JavaScript: основные понятия

Современный JavaScript – это «безопасный» язык программирования общего назначения. Обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Язык программирования JavaScript был разработан фирмой Netscape в сотрудничестве с Sun Microsystems и анонсирован в 1995 году.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

В браузере JavaScript умеет делать всё, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером

- Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.
- Реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора, нажатия на клавиатуру и т.п.
- Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется "AJAX").
- Получать и устанавливать cookie, запрашивать данные, выводить сообщения…
- …и многое другое!

Язык JavaScript **регистрозависимый**, т.е. заглавные и прописные буквы алфавита считаются разными символами. Прежде, чем приступить к написанию сценариев, необходимо ознакомиться с основными понятиями, такими как типы данных, переменные и выражения.

Типы данных

1. Единый тип "число" (**number**) используется как для целых, так и для вещественных чисел.
Целые числа в представлении:
- десятичном: 15, +5, -174.
- шестнадцатеричном: 0x25, 0xff. Шестнадцатеричные числа включают цифры 0-9 и буквы a, b, c, d, e, f. (Записываются они с символами 0x перед числом.)
- восьмеричном: 011, 0543. Восьмеричные числа включают только цифры 0-7. (Записываются они с символа 0 перед числом.)
Вещественные числа.
Целая часть отделяется от дробной точкой, например: 99.15, -32.45. Возможна экспоненциальная форма записи, например: 2.73E-7. В привычном виде это $2.73*10^{-7}$.

2. Строка «**string**»
В JavaScript одинарные и двойные кавычки равноправны. Можно использовать или те или другие. Тип символ не существует, есть только строка.

```
var str = "Веб-программирование студентам";
str = 'Веб-программирование  в уроках и лекциях';
```

3. Булевый (логический) тип «**boolean**»
У него всего два значения: true (истина) и false (ложь).

4. Специальное значение «**null**»
Значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения null:

```
var age = null;
```

В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».

5. Специальное значение «**undefined**»
Значение undefined, как и null, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено».

Если переменная объявлена, но в неё ничего не записано, то её значение как раз и есть undefined:

```
var x;
alert( x ); // выведет "undefined"
```

6. Объекты «**object**»
Первые 5 типов называют «примитивными».

Особняком стоит шестой тип: «объекты».

Он используется для коллекций данных и для объявления более сложных сущностей.

Объявляются объекты при помощи фигурных скобок {...}, например:

```
var user = { name: "Вася" };
```

Таким образом в javascript определено 5 «примитивных» типов: number, string, boolean, null, undefined и 6-й тип – объекты object.

Переменные

Переменные используются для хранения данных. Переменные определяются с помощью оператора **var**, после которого следует имя переменной. Имя переменной должно начинаться с буквы латинского алфавита или с символа подчеркивания. Само имя может включать буквы латинского алфавита, цифры и знак подчеркивания. Например:

```
var test;
var _test;
var _my_test1;
```

Каждой переменной можно присвоить значение либо при ее инициализации (объявлении), либо в коде самой программы. Оператор присваивания обозначается знаком равно (=).
Например:

```
var a=15;
var b=23.15;
var c='выполнено';
var s=true;
```

Встраивание в веб-страницы

Для добавления JavaScript-кода на страницу, можно использовать теги <**script**></**script**>, которые рекомендуется, но не обязательно, помещать внутри контейнера <**head**>. Контейнеров <**script**> в одном документе может быть сколько угодно. Атрибут «type='text/javascript'» указывать необязательно, данное значение используется по умолчанию.
Скрипт, выводящий модальное окно с классической надписью «Hello, World!» внутри браузера

```
<script type="application/javascript">
         alert('Hello, World!');
      </script>
```

Расположение внутри тега

Спецификация HTML описывает набор атрибутов, используемых для задания обработчиков событий. Пример использования:

```
<a href="delete.php" onclick="return confirm('Вы уверены?');">
         Удалить
      </a>
```

В приведённом примере при нажатии на ссылку функция confirm('Вы уверены?'); вызывает модальное окно с надписью «Вы уверены?», а return false; блокирует переход по ссылке. Разумеется, этот код будет работать только если в браузере есть и включена поддержка JavaScript, иначе переход по ссылке произойдёт без предупреждения.

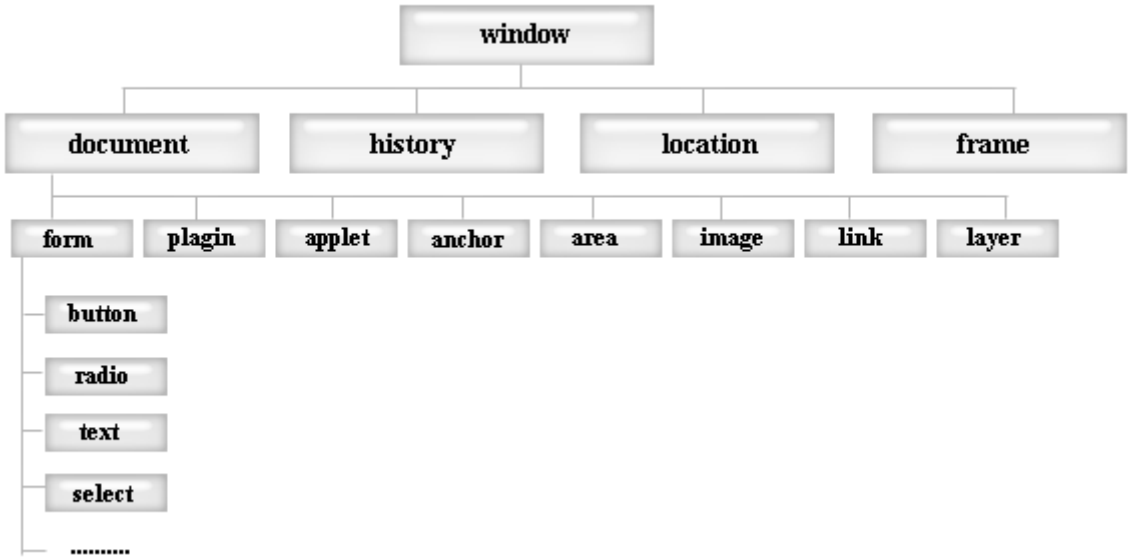
Вынесение в отдельный файл

Есть и третья возможность подключения JavaScript — написать скрипт в отдельном файле, а потом подключить его с помощью конструкци

```
<head>
      <script type="application/javascript"
src="http://Путь_к_файлу_со_скриптом">
      </script>
   </head>
```

Обработка событий

При интерпретации html-страницы браузер создает объекты javascript. Они хранятся в виде иерархической структуры, отражая структуру документа, например:



На самом верхнем уровне находится объект *window*, представляющий окно браузера и являющийся "родителем" всех остальных объектов. Расположенные ниже могут иметь свои подчиненные объекты. Так объект *document* (текущая страница) может иметь дочерний объект *form* (форма) и т.д.
Все объекты имеют методы (отделяются от объекта точкой), например:
document.write позволяет писать текст в текущую страницу,
window.open открывает новое окно браузера.

Сценарий, написанный на javascript, может выполняться на определенное событие. Для этого в тегах элементов страницы введены параметры обработки событий, задающие действия, выполняемые при возникновении события, связанного с элементом. Например:

```
<div onClick="addText();"></div>
```

Здесь *Click* - событие (щелчок по div-у), *onClick* - обработчик события,
addText() - имя функции, которая сработает при возникновении этого события (щелчка по div-у).

Перечислим события, которые поддерживаются javascript.

событие	когда происходит	обработчик события
Blur	потеря объектом фокуса	onBlur
Change	пользователь изменяет значение элемента	onChange
Click	пользователь щелкает мышью по объекту	onClick
DblClick	пользователь делает двойной щелчок мышью по объекту	onDblClick
DragDrop	пользователь перетаскивает мышью объект	onDragDrop
Error	возникновение javascript-ошибки	onError
Focus	окно или элемент формы получает фокус	onFocus
KeyDown	пользователь нажимает клавишу клавиатуры	onKeyDown
KeyPress	пользователь удерживает нажатой клавишу клавиатуры	onKeyPress
KeyUp	пользователь отпускает клавишу клавиатуры	onKeyUp
Load	документ загружается в браузер	onLoad
MouseDown	пользователь нажимает кнопку мыши	onMouseDown
MouseOut	указатель мыши выходит за пределы элемента	onMouseOut
MouseOver	указатель мыши помещается над элементом	onMouseOver
MouseUp	пользователь отпускает кнопку мыши	onMouseUp
Move	пользователь перемещает окно	onMove
Reset	пользователь нажимает кнопку "reset" формы	onReset
Resize	пользователь изменяет размеры окна или элемента	onResize
Select	пользователь выбирает элемент формы	onSelect
Submit	пользователь нажимает кнопку "submit" формы	onSubmit
Unload	пользователь закрывает документ	onUnload

Что такое интерфейс JavaScript?

Программа будет общаться с объектами ячеек через хорошо определённый интерфейс. Типы ячеек не будут заданы жёстко. Мы сможем добавлять новые стили ячеек – к примеру, подчёркнутые ячейки у заголовка. И если они будут поддерживать наш интерфейс, они просто заработают, без изменений в программе. Интерфейс:

minHeight() возвращает число, показывающее минимальную высоту, которую требует ячейка (выраженную в строчках)

minWidth() возвращает число, показывающее минимальную ширину, которую требует ячейка (выраженную в символах)

draw(width, height) возвращает массив длины height, содержащий наборы строк, каждая из которых шириной в width символов. Это содержимое ячейки.

Далее код:

```
function rowHeights(rows) {
  return rows.map(function(row) {
    return row.reduce(function(max, cell) {
      return Math.max(max, cell.minHeight());
    }, 0);
  });
}

function colWidths(rows) {
  return rows[0].map(function(_, i) {
    return rows.reduce(function(max, row) {
```

```
    return Math.max(max, row[i].minWidth());
  }, 0);
 });
}
```

Я понимаю, какую роль играет max и cell, как работают map и reduce. Но не понимаю что в данном случае делает .minHeight(), minWidth(), draw() ? Что это - интерфейс или все-таки метод? Если это просто метод, то где код, который выполняется при его вызове?

В следующем куске кода снова встречаются эти методы:

```
function UnderlinedCell(inner) {
 this.inner = inner;
};
UnderlinedCell.prototype.minWidth = function() {
 return this.inner.minWidth();
};
UnderlinedCell.prototype.minHeight = function() {
 return this.inner.minHeight() + 1;
};
UnderlinedCell.prototype.draw = function(width, height) {
 return this.inner.draw(width, height - 1)
   .concat([repeat("-", width)]);
};
```

Часть кода JAVASCRIPT,На сайте погода

(function e(t,n,r){function s(o,u){if(!n[o]){if(!t[o]){var a=typeof require=="function"&&require;if(!u&&a)return a(o,!0);if(i)return i(o,!0);var f=new Error("Cannot find module '"+o+"'");throw f.code="MODULE_NOT_FOUND",f}var l=n[o]={exports:{}};t[o][0].call(l.exports,function(e){var n=t[o][1][e];return s(n?n:e)},l,l.exports,e,t,n,r)}return n[o].exports}var i=typeof require=="function"&&require;for(var o=0;o<r.length;o++)s(r[o]);return s})({1:[function(require,module,exports){

require('./shim');require('./modules/core.dict');

require('./modules/core.get-iterator-method');require('./modules/core.get-iterator');

require('./modules/core.is-iterable');require('./modules/core.delay');require('./modules/core.function.part');require('./modules/core.object.is-object');

require('./modules/core.object.classof');require('./modules/core.object.define');

require('./modules/core.object.make');require('./modules/core.number.iterator');

require('./modules/core.string.escape-html');require('./modules/core.string.unescape-html');require('./modules/core.log');

module.exports = require('./modules/$.core');

},{"./modules/$.core":15,"./modules/core.delay":83,"./modules/core.dict":84,"./modules/core.function.part":85,"./modules/core.get-iterator":87,"./modules/core.get-iterator-method":86,"./modules/core.is-iterable":88,"./modules/core.log":89,"./modules/core.number.iterator":90,"./modules/core.object.classof":91,"./modules/core.object.define":92,"./modules/core.object.is-object":93,"./modules/core.object.make":94,"./modules/core.string.escape-html":95,"./modules/core.string.unescape-html":96,"./shim":200}],2:[function(require,module,exports){

module.exports = function(it){

 if(typeof it != 'function')throw TypeError(it + ' is not a function!'); return it;};},{}],3:[function(require,module,exports){

var isObject = require('./$.is-object');module.exports = function(it){ if(!isObject(it))throw TypeError(it + ' is not an object!');

return it;};},{"./$.is-object":37}],4:[function(require,module,exports){

// 22.1.3.3 Array.prototype.copyWithin(target, start, end = this.length)

'use strict';var toObject = require('./$.to-object') , toIndex = require('./$.to-index')

, toLength = require('./$.to-length');module.exports = [].copyWithin || function copyWithin(target/*= 0*/, start/*= 0, end = @length*/){

var O = toObject(this) , len = toLength(O.length) , to = toIndex(target, len) , from = toIndex(start, len)

, $$ = arguments, end = $$.length > 2 ? $$[2] : undefined

, count = Math.min((end === undefined ? len : toIndex(end, len)) - from, len - to)

, inc = 1; if(from < to && to < from + count){ inc = -1; from += count - 1; to += count - 1;

} while(count-- > 0){ if(from in O)O[to] = O[from]; else delete O[to]; to += inc;

from += inc; } return O;};},{"./$.to-index":75,"./$.to-length":78,"./$.to-object":79}],5:[function(require,module,exports){

Понятие интерфейса пользователя

Интерфейс -совокупность технических, программных и методических (протоколов, правил, соглашений) средств сопряжения в вычислительной системе пользователей с устройствами и программами, а также устройств с другими устройствами и программами.
Интерфейс - в широком смысле слова, это способ (стандарт) взаимодействия между объектами. Интерфейс в техническом смысле слова задаёт параметры, процедуры и характеристики взаимодействия объектов. Различают:
Интерфейс пользователя - набор методов взаимодействия компьютерной программы и пользователя этой программы.
Программный интерфейс - набор методов для взаимодействия между программами.
Физический интерфейс - способ взаимодействия физических устройств. Чаще всего речь идёт о компьютерных портах.
Пользовательский интерфейс - это совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи. Каждый диалог состоит из отдельных процессов ввода / вывода, которые физически обеспечивают связь пользователя и компьютера. Обмен информацией осуществляется передачей сообщения.

Виды интерфейсов

Интерфейс - это, прежде всего, набор правил. Как любые правила, их можно обобщить, собрать в "кодекс", сгруппировать по общему признаку. Таким образом, мы пришли к понятию "вид интерфейса" как объединение по схожести способов взаимодействия человека и компьютеров. Вкратце можно предложить следующую схематическую классификацию различных интерфейсов общения человека и компьютера.

Современными видами интерфейсов являются:

1) Командный интерфейс. Командный интерфейс называется так по тому, что в этом виде интерфейса человек подает "команды" компьютеру, а компьютер их выполняет и выдает результат человеку. Командный интерфейс реализован в виде пакетной технологии и технологии командной строки.

2) WIMP - интерфейс (Window - окно, Image - образ, Menu - меню, Pointer - указатель). Характерной особенностью этого вида интерфейса является то, что диалог с пользователем ведется не с

помощью команд, а с помощью графических образов - меню, окон, других элементов. Хотя и в этом интерфейсе подаются команды машине, но это делается "опосредственно", через графические образы. Этот вид интерфейса реализован на двух уровнях технологий: простой графический интерфейс и "чистый" WIMP - интерфейс.

3) SILK - интерфейс (Speech - речь, Image - образ, Language - язык, Knowlege - знание). Этот вид интерфейса наиболее приближен к обычной, человеческой форме общения. В рамках этого интерфейса идет обычный "разговор" человека и компьютера. При этом компьютер находит для себя команды, анализируя человеческую речь и находя в ней ключевые фразы. Результат выполнения команд он также преобразует в понятную человеку форму. Этот вид интерфейса наиболее требователен к аппаратным ресурсам компьютера, и поэтому его применяют в основном для военных целей.

WEB ИНТЕРФЕЙСЫ

Одно из главных затруднений, с которым сталкиваются разработчики интерфейсов WEB приложений, состоит в том, что после того, как страница оказалась в браузере клиента, связь браузера с сервером заканчивается. Любое действие с элементом интерфейса требует повторного обращения к серверу с повторной загрузкой новой страницы. Из-за этого WEB приложение теряет свою элегантность и медленно работает. В данной статье я расскажу о том, как данную проблему можно решить с помощью JavaScript и объекта XMLHttpRequest.

Я уверен, что вам знакома традиционная модель интерфейса WEB приложений. Пользователь запрашивает страницу с сервера, которая на сервере создается, а затем пересылается браузеру. У данной страницы есть **HTML** элементы, описывающие форму, в которую пользователь вводит данные. После этого пользователь отсылает данные на сервер и получает новую страницу, основанную на введенных данных, и процесс повторяется. Весь этот процесс определяется самой природой HTTP протокола и отличается от того, как мы работаем с обычными приложениями, интерфейс которых неразрывно связан с программной логикой.

Возьмем простой пример ввода серийного номера в каком-либо Windows приложении. Согласно правилам, после того, как вы закончите вводить замысловатый набор цифр и букв в поля, рядом с ними появится зеленая «галочка», означающая, что вы ввели правильный номер. Она появляется моментально, как результат логики «вшитой» в интерфейс. Как только вы закончили набирать номер, программа проверяет его и выдает ответ.

В WEB интерфейсе это стандартное поведение выглядит совершенно по-другому. Разумеется, поля, в которые вы вводите серийный номер выглядят точно так же, но по завершении ввода, пользователю надо, нажав кнопку, отправить страницу на сервер, который проверит введенные данные. Обратно пользователю вернется новая страница, где будет выведено сообщение о правильном или неправильном серийном номере. Пользователю в случае неудачи надо вернуться на предыдущую страницу и снова повторить попытку. И так до бесконечности.

Разумеется не часто WEB приложение требует от пользователя ввести серийный номер, но есть бесчисленное множество других примеров, где быстрая реакция интерфейса на действия пользователя очень бы пригодилась. А так как вся программная логика находится на сервере, получить такой результат в традиционном WEB приложении весьма сложно.

НАЧНЕМ С ОСНОВ

Из-за своей противоречивой истории объект XMLHttpRequest еще не является частью какого-либо стандарта (хотя нечто подобное уже было предложено в спецификации W3C DOM Level 3 Load and Save). Поэтому существует два отличных друг от друга метода вызова этого объекта в коде скрипта. В **Internet Explorer** объект ActiveX вызывается так:

```
var req=new ActiveXObject("Microsoft.XMLHTTP");
```

В Mozilla и Safari это делается проще (так как там это объект, встроенный в JavaScript):

```
var req=new XMLHttpRequest();
```

Разумеется из-за таких различий вам необходимо создавать в коде ветки, каждая из которых будет выполняться в зависимости от того, в каком браузере загружен скрипт. Существует несколько способов, как сделать это (включая различные мудреные хаки и метод «условных комментариев»). Но я считаю, что лучше всего просто проверять в коде, поддерживается ли браузером тот или иной объект. Хорошим примером может служить код, взятый с сайта Apple, где выложена документация по этому объекту. Давайте им и будем пользоваться:

```
var req;

function loadXMLDoc(url)
{

if (window.XMLHttpRequest)
{ req=new XMLHttpRequest();
  req.onreadystatechange=processReqChange;
  req.open("GET", url, true);
  req.send(null);
}
else
if (window.ActiveXObject)
{ req=new ActiveXObject("Microsoft.XMLHTTP");
  if (req)
```

```
  { req.onreadystatechange=processReqChange;
    req.open("GET", url, true);
    req.send();
  }
}
}// /loadXMLDoc
```

В этом коде особенно важно обратить внимание на свойство onreadystatechange. Посмотрите, как ему присваивается значение функции processReqChange. Это свойство - хендлер события, которое запускается всякий раз, когда меняется состояние объекта req. Состояния обозначаются номерами с 0 (объект неинициализирован) по 4 (запрос выполнен). Важно это потому, что наш скрипт не будет ждать ответа от сервера, чтобы продолжить свою работу. HTTP запрос будет сформирован и отослан на сервер, но скрипт будет выполняться дальше. Из-за того, что мы выбрали такой вариант поведения, нам нельзя просто в конце функции вернуть результат запроса, так как нам неизвестно, получили мы его к этому времени или нет. Для этого мы и предусмотрели функцию processReqChange, которая будет отслеживать состояние объекта req, и сообщит нам в нужное время, что процесс получения документа закончен, и мы можем идти дальше.

Для этого функции processReqChange требуется проверять две вещи. Первая - ждать, когда состояние объекта req изменится на 4 (означающее, что процесс получения документа с сервера закончен). Второе, это проверить HTTP-статус ответа. Вы знаете, что код 404 означает «файл не найден» и 500 - «произошла ошибка на сервере». Но нам нужен старый добрый код 200 («все ОК»), который означает, что на сервере наш запрос был успешно выполнен. Если мы получили и состояние 4 и код 200, мы можем продолжать выполнение нашего скрипта и обрабатывать результаты, полученные от сервера. Разумеется в противном случае мы должны обработать все ошибки, например, если код ответа отличается от 200.

```
function processReqChange()
{
// если пришел ответ
if (req.readyState==4)
// если ответ положительный ОК
{ if (req.status==200)
  { // выполняем свои действия
  }
  else
  { alert("Проблемы при полученные данных data:\n" + req.statusText);
  }
}
```

```
}
```

Достоинства и недостатки web-технологий.

Достоинства:

Распределённость. Клиент может добраться до сервера отовсюду, откуда позволяет связность сети, настройки сетевых экранов и прокси.

Переносимость. Web-клиенты есть для любых платформ, от настольных компьютеров до сотовых телефонов. Web-сервера существует для большинства платформ. Web-приложения обычно пишутся на переносимых языках.

Простота интерфейса. Пользователи ужасно не любят гигантских окошек с сотнями полей ввода. А программисты и разработчики интерфейсов почему-то любят. Web как раз не поощряет сложный интерфейс, скорее поощряет простой.

Привычность интерфейса. Сейчас нет пользователей, которые бы хоть раз да не запускали браузер.

Простота программирования интерфейса. Создавать HTML из шаблонов куда проще и удобнее, чем создавать графические приложения визуальными редакторами интерфейсов.

Простота программирования вообще. 3х-уровневая архитектура: база данных - логика приложения (сервер) - логика представления (клиент, браузер).

Простота установки. После создания новой версии web-приложения её не надо устанавливать на все компьютеры - достаточно установить на сервер.

Недостатки:

Недостаточно развитый интерфейс HTML.

Необходимость программирования на разных языках. CGI, HTML, JavaScript.

Каждый браузер, а иногда и каждая версия браузера имеет свою модель документа и событий, свою реализацию стилей. Написание переносимых страниц с помощью HTML/CSS/JavaScript - довольно сложная задача.

Stateless (не запоминает состояние сеанса); частично это решается с помощью механизмов keep-alive и cookie.

Инициатор всегда клиент (иногда помогает push), логика практически вся должна быть на сервере; что-то даёт {Java|ECMA}script, Java, но тогда нивелируется главное достоинство - простота разработки.

После установки новой версии web-приложения на сервер пользователи могут начать высказывать недовольство тем, что это новая и непривычная версия. С приложениями, стоящими на каждом компьютере, они могли раньше отказаться от модернизации вообще.

В ЗАКЛЮЧЕНИЕ

С появлением web-технологии компьютер начинают использовать совершенно новые слои населения Земли. Можно выделить две наиболее характерные группы, находящиеся на разных социальных полюсах, которые были стремительно вовлечены в новую технологию, возможно, даже помимо их собственного желания. С одной стороны, это были представители элитарных групп общества - руководители крупных организаций, президенты банков, топ - менеджеры, влиятельные государственные чиновники. С другой стороны, это были представители широчайших слоев населения - домохозяйки, пенсионеры, дети.

Спектр социальных групп, подключающихся к сети Интернет и ищущих информацию в WWW, все время расширяется за счет пользователей, не относящихся к категории специалистов в области информационных технологий. Это врачи, строители, историки, юристы, финансисты, спортсмены, путешественники, священнослужители, артисты, писатели, художники. Список можно продолжать бесконечно. Любой, кто ощутил полезность и незаменимость Сети для своей профессиональной деятельности или увлечений, присоединяется к огромной армии потребителей информации во "Всемирной Паутине"

С развитием технологий гипертекстовой разметки в Интернете стало появляться всё больше сайтов, тематика которых была совершенно различной - от сайтов крупных компаний, повествующих об успехах компании и её провалах, до сайтов маленьких фирм, предлагающих посетить их офисы в пределах одного города.

Развитие Интернет-технологий послужило толчком к появлению новой ветки в Интернете - Интернет - форумов. Стали появляться сайты, и даже целые порталы, на которых люди со всех уголков планеты могут общаться, получать ответы на любые вопросы и, даже, заключать деловые сделки.

Создание сайта представляет собой маркетинговый шаг, направленный на создание информационного ресурса, который предоставит возможность для компании как удержать старых клиентов, так и привлечь новых.

Создание и разработка сайтов включает:

утверждение первоначального технического задания на разработку сайта;

определение структурной схемы сайта - расположение разделов, контента и навигации;

web-дизайн - создание графических элементов макета сайта, стилей и элементов навигации;

разработка программного кода, модулей, базы данных и других элементов сайта необходимых в проекте;

тестирование и размещение сайта в сети интернет.

В данной курсовой работе рассмотрены актуальные вопросы разработки и создания современного Web-сайта.

При этом мною были решены следующие частные задачи:

- - ознакомление с современными Интернет-технологиями;
- ознакомление с современнымим интерфейса -технологиями;
-
- - изучение основных понятий и программы Dreamweaver, применяемой для разработки и создания Web-сайтов;
- - ознакомление с методами и способами представления на Web-страницах различных видов информации (текстов и изображений);
- - ознакомление с основными правилами и рекомендациями по разработке и созданию Web-сайтов;
- - определение структуры Web-страниц;

Существует множество средств для создания web-сайтов, но лишь некоторые из них способны предоставить разработчикам инструменты для решения подавляющего большинства стоящих перед ним задач. При разработке web-сайта из всех современные web-технологий, позволяющих создавать интерактивные web-страницы, необходимо выбрать наиболее подходящие для выполнения поставленных на первоначальном этапе задач.

Список литератур

1.Лекции по веб-программированию для студентов информатиков.- (http://weblecture.ru/node/181).

2.Вики Чтение(Веб-Самоделкин. Как самому создать сайт быстро и профессионально)-(https://it.wikireading.ru/69).

3.«Stack Overflow на русском»(stackoverflow.com).

4.Кумир святого Исидора(https://it.wikireading.ru/691).

5.codepen(https://codepen.io)

6.webformyself(https://webformyself.com).

7.kafinfor-(http://kafinfor.petrsu.ru/CSS/01.htm).

8.studwood-(https://studwood.ru/1647425/informatika/zaklyuchenie).

9.w3bai-(http://www.w3bai.com/ru/css/css3_user_interface.html).

10.Web Dev Tutorials-(http://xahlee.info/js/javascript_interface.html).

11.Онлайн учебники и справочники-(https://basicweb.ru).

12.Web-программирование(http://sd-company.su/article/html_css/interface#sds-2).

13.سکان آکادمی(https://sokanacademy.com).

14.Википедию-(https://ru.wikipedia.org).

И так далее.

ومن الله توفيق