ВЫПОЛНЕНИЕ ЛАБОРАТОРНЫХ РАБОТ

ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

«ТЕХНОЛОГИИ ПОСТРОЕНИЯ

WEB-ИНТЕРФЕЙСОВ»

Учебно-методическое пособие

# Предисловие

Выполняя лабораторные работы, рассмотренные в данном учебно-методическом пособии, обучающиеся знакомятся с различными подходами   
к верстке web-страниц и используемыми при создании web-интерфейсов технологиями, такими, как языки разметки xhtml и html 5, язык программирования JavaScript и библиотека jQuery, библиотека для создания адаптивных страниц BootStrap.

Так же в процессе выполнения лабораторных работ по дисциплине «Технологии построения web-интерфейсов» обучающиеся знакомятся со средствами разработчика, встроенными в браузер, что позволяет им производить отладку и анализировать особенности реализации web-интерфейсов.

Описание каждой лабораторной работы содержит следующие части:

1. цель работы;
2. условия и порядок проведения лабораторной работы;
3. необходимые теоретические сведения;
4. описание последовательности действий обучающегося при выполнении работы;
5. вопросы для самоконтроля и задачи.

Выполнение лабораторной работы должно предваряться лекционным занятием по данной теме и практическими занятиями, на которых обучающимся демонстрируются базовые приемы использования лекционного материала.

Для выполнения лабораторных работ требуется следующее программное обеспечение:

редактор html-кода, например, Adobe Dreamweaver CC или новее;

программа для обработки растровых изображений, например, Adobe Photoshop CC или новее;

браузер с инструментами разработчика, например, Mozilla Firefox версии 66 или новее;

библиотека jQuery;

библиотека Bootstrap 3 или новее;

Содержание отчета обучающегося о выполненной работе должно включать краткое описание, листинги html, css и js файлов, снимки экрана, подтверждающие решение задач, поставленных в ходе лабораторной работы.

Продолжительность лабораторных работ может составлять   
2-4 академических часа в зависимости от уровня подготовки обучающихся (некоторые лабораторные работы требуют дополнительной работы обучающихся в компьютерном классе в часы самоподготовки.).

Авторы рекомендуют придерживаться порядка выполнения лабораторных работ, поскольку в некоторых лабораторных работах используются результаты, полученные в предшествующих работах.

# Методические рекомендации по выполнению лабораторной работы № 1. Разработка документа с помощью каскадной таблицы стилей

1. Цель работы

Целью выполнения работы является освоение обучающимися базовых тегов, применяемых в xhtml, а также селекторов и правил CSS.

1. Условия и порядок проведения лабораторной работы

Для проведения занятия необходимо знание лекционного материала по теме лекций №№ 2-4. В ходе лабораторной работы нужно создать документ с формой по стандарту xhtml 1.0 strict и файл css, содержащий набор связанных стилей, согласно указанным правилам. Занятие проводится в форме тренинга, в ходе которого сначала осуществляется перекрестный опрос по теме лекции №№ 2-4, затем обучающиеся выполняют задание индивидуально, последние 15 минут занятия они попарно проверяют код друг друга, выявляя ошибки и недочеты, формируя навыки по анализу корректности кода.

1. Необходимые теоретические сведения

При написании кода XHTML в основном придерживаются того же синтаксиса, который характерен для HTML, но этом разница между HTML и XHTML состоит в наборе некоторых обязательных правил:

1. Все XHTML-документы обязательно должны содержать элемент <!DOCTYPE> в первой строке кода.
2. Теги должны быть набраны в нижнем регистре. Это правило возникло из-за того, что XHTML чувствителен к регистру символов, поэтому для него теги <HR> и <hr> различаются. Чтобы не возникало путаницы, синтаксис принудительно заставляет указывать все теги, а также их атрибуты в нижнем регистре.
3. Значения любых атрибутов необходимо заключать в кавычки. Хотя в HTML также требуется заключать значения в кавычки, но их отсутствие никак не влияет на корректность кода. Так что можно сказать, что в HTML применение кавычек − это лишь рекомендация. В XHTML же использование кавычек возведено в правило и любые значения атрибутов требуется указывать только в них.
4. Требуется закрывать все теги. В HTML теги делятся на две категории − парные теги, называемые еще контейнерами и одинарные теги. Парные теги состоят из открывающего и закрывающего тега, причем в некоторых случаях закрывающий тег можно опустить. В XHTML закрывающий тег требуется всегда и везде. Некоторые разработчики игнорируют закрывающие теги </li>, </p>, </td>, но в XHTML их отсутствие считается ошибкой. Элемент <!DOCTYPE> не является частью XHTML-документа, поэтому для него закрывающего тега не требуется. Что касается одинарных тегов, то они должны завершаться слэшем перед закрывающей угловой скобкой: <br />.
5. Должна соблюдаться правильная вложенность тегов. XHTML критично относится к ошибкам следующего типа: некорректная вложенность одного тега в другой и расположение тега в несоответствующем контейнере. Каждый тег должен располагаться внутри другого тега, при этом недопустимо их «пересечение». Хотя валидатор кода и выдает ошибку в случае неправильного положения тегов, браузеры при этом правильно отображают web-страницу. Все теги имеют строгую иерархическую систему в том смысле, что каждый тег должен находиться внутри другого тега и никак иначе. На условной вершине находится корневой элемент <html>, а все остальные теги могут содержать внутри себя другие теги, которые называются дочерними. Соответственно дочерние теги располагаются в родительском элементе. Необходимо знать систему подчиненности тегов и следовать ей при написании кода XHTML.
6. Нельзя использовать сокращенные атрибуты тегов.
7. Вместо атрибута name следует указывать id. Атрибут name определяется в HTML для тегов <a>, <frame>, <iframe>, <img> и <map> и предназначен для обозначения элемента с целью последующего к нему обращения из скриптов. В XHTML атрибут name частично вышел из употребления, а вместо него следует использовать id.

### Разновидности стандарта

Любой XHTML-файл состоит из четырех разделов − <!DOCTYPE>, контейнера <html>, заголовка (<head>) и тела документа (<body>). Последние два элемента перекочевали из HTML и ничем не отличаются от своего родоначальника

Элемент <!DOCTYPE> сообщает браузеру о типе текущего документа и как его интерпретировать. Различают три типа документа по спецификации XHTML 1.0:

**XHTML 1.0 Strict:** используется в том случае, если в документе идет четкое разделение оформления и содержания. При этом код web-страницы содержит только теги разметки, а сам вид элементов задается через стили.

**XHTML 1.0 Transitional:** обычно применяется, когда правило разделения оформления и содержания выполняется не в полной мере. В этом случае допускается в коде документа использовать теги физического проектирования (например, тег <tt>) и лишь частично стили.

**XHTML 1.0 Frameset:** применяется, когда окно браузера делится на два или более фрейма.

**Кодировка документа**

В HTML кодировка документа задается с помощью тега <meta> расположенного внутри контейнера <head>. В XHTML же кодировка может задаваться двумя путями: аналогично HTML и с помощью команды <?xml version="1.0" encoding="UTF-8"?>, она должна располагаться в первой строке кода, перед <!DOCTYPE>. В данном случае UTF-8 обозначает кодировку текущего документа.

### Теги заголовка

Тег **<head>** предназначен для хранения других элементов, цель которых − помочь браузеру в работе с данными. Также внутри контейнера <head> находятся метатеги, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных.

Содержимое тега <head> не отображается напрямую на web-странице, за исключением тега <title>, устанавливающего заголовок окна web-страницы. Внутри контейнера <head> допускается размещать следующие элементы:

**<title>** −используется для отображения строки текста в левом верхнем углу окна браузера, а также на вкладке. Такая строка сообщает пользователю название сайта и другую информацию, которую добавляет разработчик.

**<meta>** −метатеги используются для хранения информации, предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Хотя тег <meta> всего один, он имеет несколько атрибутов, поэтому к нему и применяется множественное число.

**<link>** − устанавливает связь с внешним документом вроде файла со стилями или со шрифтами. В отличие от тега <a>, тег <link> размещается всегда внутри контейнера <head> и не создает ссылку.

**<style>** − тег <style> применяется для определения стилей элементов web-страницы. Тег <style> необходимо использовать внутри контейнера <head>. Можно задавать более чем один тег <style>.

### Теги тела документа

Внутри тела документа можно использовать следующие теги:

**<blockquote>** − предназначен для выделения длинных цитат внутри документа. Текст, обозначенный этим тегом, традиционно отображается как выровненный блок с отступами слева и справа (примерно по 40 пикселов), а также с пустым пространством сверху и снизу.

**<div>** − относится к универсальным блочным контейнерам и применяется в тех случаях, где нужны блочные элементы без дополнительных свойств. Также с помощью тега <div> можно выравнивать текст внутри этого контейнера с помощью атрибута align.

**<h1>,...,<h6>** − эта группа тегов определяет текстовые заголовки разного уровня, которые показывают относительную важность секции, расположенной после заголовка.

**<hr>** − рисует горизонтальную линию, которая по своему виду зависит от используемых атрибутов. Линия всегда начинается с новой строки, а после нее все элементы отображаются на следующей строке.

**<p>** − определяет параграф (абзац) текста.

**<a>** − является одним из важных элементов HTML и предназначен для создания ссылок. В зависимости от присутствия атрибутов name или href тег <a> устанавливает ссылку или якорь.

**<b>** − определяет жирное начертание шрифта.

**<big>** − увеличивает размер шрифта на единицу по сравнению с обычным текстом. В HTML размер шрифта измеряется в условных единицах от 1 до 7, средний размер текста, используемый по умолчанию, принят 3. Таким образом, добавление тега <big> увеличивает текст на одну условную единицу.

**<br>** − устанавливает перевод строки в том месте, где этот тег находится. В отличие от тега параграфа <p>, использование тега <br> не добавляет пустой отступ перед строкой.

**<em>** − предназначен для акцентирования текста. Браузеры отображают такой текст курсивным начертанием.

**<i>** − устанавливает курсивное начертание шрифта.

**<img>** − предназначен для отображения на web-странице изображений в графическом формате GIF, JPEG или PNG, адрес картинки находится внутри обязательного атрибута src.

**<ol>** − устанавливает нумерованный список, где каждый элемент списка начинается с числа или буквы и увеличивается по нарастающей, где тег **<li>** определяет отдельный элемент списка.

**<ul>** − устанавливает маркированный список, каждый элемент которого начинается с небольшого символа − маркера, где тег **<li>** определяет отдельный элемент списка. Внешний тег <ul> или <ol> устанавливает тип списка − маркированный или нумерованный.

**<dd>**, **<dt>**, **<dl>** − тройка элементов предназначена для создания списка определений. Каждый такой список начинается с контейнера <dl>, куда входит тег <dt>, создающий термин и тег <dd>, задающий определение этого термина.

**<table>** − служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов <tr> и <td>.

**<tr>** − служит контейнером для создания строки таблицы.

**<td>** − предназначен для создания одной ячейки таблицы. Тег <td> должен размещаться внутри контейнера <tr>, который в свою очередь располагается внутри тега <table>.

**<th>** − предназначен для создания одной ячейки таблицы, которая обозначается как заголовочная. Текст в такой ячейке отображается браузером обычно жирным шрифтом и выравнивается по центру.

**<form> −** устанавливает форму на web-странице. Форма предназначена для обмена данными между пользователем и сервером. Область применения форм не ограничена отправкой данных на сервер, с помощью клиентских скриптов можно получить доступ к любому элементу формы, изменять его и применять по своему усмотрению.

Тег **<input>** является одним из разносторонних элементов формы и позволяет создавать разные элементы интерфейса и обеспечить взаимодействие с пользователем. Главным образом <input> предназначен для создания текстовых полей, различных кнопок, переключателей и флажков. Хотя элемент <input> не требуется помещать внутрь контейнера <form>, определяющего форму, но если введенные пользователем данные должны быть отправлены на сервер, где их обрабатывает серверная программа, то указывать <form>обязательно. То же самое обстоит и в случае обработки данных с помощью клиентских приложений, например, скриптов на языке JavaScript.

Основной атрибут тега <input>, определяющий вид элемента **type**. Он позволяет задавать следующие элементы формы: текстовое поле (**text**), поле с паролем (**password**), переключатель (**radio**), флажок (**checkbox**), скрытое поле (**hidden**), кнопка (**button**), кнопка для отправки формы (**submit**), кнопка для очистки формы (reset), поле для отправки файла (file) и кнопка с изображением (**image**). Для каждого элемента существует свой список атрибутов, которые определяют его вид и характеристики. Кроме того, в HTML5 добавлено еще более десятка новых элементов.

Тег **<button>** создает на web-странице кнопки и по своему действию напоминает результат, получаемый с помощью тега <input> (с атрибутом type="button | reset | submit"). В отличие от этого тега, <button> предлагает расширенные возможности по созданию кнопок. Например, на подобной кнопке можно размещать любые элементы HTML, в том числе изображения. Используя стили можно определить вид кнопки путем изменения шрифта, цвета фона, размеров и других параметров.

Теоретически, тег <button> должен располагаться внутри формы, устанавливаемой элементом <form>. Тем не менее, браузеры не выводят сообщение об ошибке и корректно работают с тегом <button>, если он встречается самостоятельно. Однако, если необходимо результат нажатия на кнопку отправить на сервер, помещать <button> в контейнер <form> обязательно.

Тег **<select>** позволяет создать элемент интерфейса в виде раскрывающегося списка, а также список с одним или множественным выбором, как показано далее. Конечный вид зависит от использования атрибута size тега <select>, который устанавливает высоту списка. Ширина списка определяется самым широким текстом, указанным в теге **<option>**, а также может изменяться с помощью стилей. Каждый пункт создается с помощью тега <option>, который должен быть вложен в контейнер <select>. Если планируется отправлять данные списка на сервер, то требуется поместить элемент <select> внутрь формы. Это также необходимо, когда к данным списка идет обращение через скрипты.

Элемент **<fieldset>** предназначен для группирования элементов формы. Такая группировка облегчает работу с формами, содержащими большое число данных. Например, один блок может быть предназначен для ввода текстовой информации, а другой для флажков.

### Работа с CSS

Применение стилей позволяет задавать точные характеристики практически всех элементов web-страницы, а это значит, что можно точно контролировать внешний вид web-страницы в окне web-браузера.

Преимущества от использования стилей:

1. **Разграничение кода и оформления.** Идея о том, чтобы код HTML был свободен от элементов оформления вроде установки цвета, размера шрифта и других параметров, стара как мир. В идеале, web-страница должна содержать только теги логического форматирования, а вид элементов задается через стили. При подобном разделении работа над дизайном и версткой сайта может вестись параллельно.
2. **Разное оформление для разных устройств.** С помощью стилей можно определить вид web-страницы для разных устройств вывода: монитора, принтера, смартфона, планшета и тд.
3. **Расширенные по сравнению с HTML способы оформления элементов.** В отличии от HTML стили имеют гораздо больше возможностей по оформлению элементов web-страниц. Простыми средствами можно изменить цвет фона элемента, добавить рамку, установить шрифт, определить размеры, положение и многое другое.
4. **Ускорение загрузки сайта.** При хранении стилей в отдельном файле, он кэшируется и при повторном обращении к нему извлекается из кэша браузера. За счёт кэширования и того, что стили хранятся в отдельном файле, уменьшается код web-страниц и снижается время загрузки документов.
5. **Централизованное хранение.** Стили, как правило, хранятся в одном или нескольких специальных файлах, ссылка на которые указывается во всех документах сайта. Благодаря этому удобно править стиль в одном месте, при этом оформление элементов автоматически меняется на всех страницах, которые связаны с указанным файлом. Вместо того чтобы модифицировать десятки HTML файлов, достаточно отредактировать один файл со стилем и оформление нужных документов сразу поменяется.
6. **Единое стилевое оформление множества документов.** Применение единообразного оформление заголовков, основного текста и других элементов создает преемственность между страницами и облегчает пользователям работу с сайтом и его восприятие в целом. Разработчикам упрощается проектирование сайта.

Существует четыре метода задания таблиц стилей в документе HTML:

**1. Связывание** − это когда HTML-документ с помощью тега <link> связывается с внешним файлом, содержащим таблицу стилей. Важнейшее свойство стилевых таблиц заключается в том, что с помощью одного css-документа, находящегося в отдельном файле, можно задавать стили для целого набора web-страниц − это очень важно при разработке больших web-сайтов, поскольку, во-первых, способствует сохранению единства стиля, а во-вторых, предоставляет возможность быстро изменить что-либо сразу во многих файлах web-сайта. Для этого следует просто написать всю таблицу стилей (без тегов <style> и <*/*style>) в отдельном текстовом файле с расширением .css, а затем подключить ее во всех HTML-документах, которые должны ее использовать. Например, если файл таблицы стилей называется styles.css, то в раздел <head> каждого из использующих ее HTML-документов нужно вставить строку, в которой указывается эта таблица стилей. Например:

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

**2. Встраивание** (глобальные стили) − таблица стилей задается в разделе <style>. Таблица стилей располагается в заголовке HTML-документа, в разделе <head> . Она занимает место между тегами <style> и <*/*style>. Например:

<style type="text/css">

{селектор1 {свойство\_1: значение;   
свойство\_2: значение; ...}

{селектор2 {свойство\_1: значение;   
свойство\_2: значение; ...}

…

</style>.

**3. Вложение** (внутренние стили) − определение стиля для конкретного элемента с помощью параметра style, размещаемого внутри тега элемента. Например:

<div style="набор деклараций">элемент</div>

При этом тег <span> рекомендуется использовать для элементов разметки на уровне текста вместо тега <font>, а тег <div> рекомендуется использовать для элементов разметки на уровне блока вместо тега <p>.

**4. Импорт** − импорт внешнего CSS-файла с помощью директивы *@*import. Например:

<style>

@import url(style1.css);

@import url(style2.css);

</style>

### Виды селекторов

В CSS существует четыре разновидности селекторов:

* по элементу (по имени);
* по классу;
* по уникальному идентификатору;
* контекстный селектор.

### 1. Селектор по элементу

Это самый простой, самый привычный и самый понятный селектор. Например, чтобы все заголовки первого уровня были синего цвета на коричневом фоне и выводились шрифтом Courierс расстоянием между буквами в 12 пикселей, в CSS документа нужно написать следующее:

h1 {color: #054AFB; background: #652A03; font-family: Courier; letter-spacing: 12px}

Еще примеры:

a {color: orange; text-decoration: none} − все элементы <a> будут оранжевого цвета без подчеркивания.

hr {margin-top: 24px; width: 75%;} − горизонтальная линия будет составлять 75% от окна браузера, поля сверху − 24px.

Существует также обобщенный селектор по элементу − **универсальный селектор**, задающий свойства по умолчанию для всех элементов кода:

\* {text-align: justify; color: #263646; background: #DFDFCA; font-family: Verdana; font-size: 80%}

Аналогичным образом в качестве селектора можно использовать любой элемент HTML. Если же одинаковые элементы должны выводиться по-разному − например, часть заголовков должна быть с подчеркиванием, часть – без, в этом случае логичней воспользоваться селектором по классу.

### 2. Селектор по классу

Это селектор с использованием специального атрибута – class, который можно применять в любом элементе HTML. Например:

.menu {color: gray}

…

<h4 class="menu">О компании</h4>

<a href="services.htm" class="menu">Услуги</a>

Или:

h1.blue {color: blue}

Селектор по классу имеет следующие свойства:

Может использоваться неоднократно.

Название чувствительно к регистру.

Имеет более низкий приоритет, чем селектор по идентификатору.

К любому тегу одновременно можно добавить несколько классов, перечисляя их в атрибуте class через пробел. Поскольку при добавлении нескольких классов они могут содержать одинаковые стилевые свойства, но с разными значениями, то берётся значение у класса, который описан в коде ниже.

### 3. Селектор по уникальному идентификатору

Эти селекторы автоматически применяют класс ко всем элементам, имеющим данный идентификатор id. В одном HTML-документе может быть только один элемент с определенным id. Например:

p#first {font-style: italic; font-weight: bold}

…

<p id="first">Первый абзац</p>

<p>Второй абзац</p>

Селектор по уникальному идентификатору имеет следующие свойства:

1. Может использоваться один раз.
2. Название чувствительно к регистру.
3. Имеет более высокий приоритет, чем класс.

### 4. Контекстные селекторы

Позволяют назначить свойства только для элементов, имеющих заданную структуру вложения. Например:

p b {background: blue}

Одной из разновидностью контекстных селекторов является **соседний селектор** − когда элементы web-страницы, когда они следуют непосредственно друг за другом в коде документа. Для управления стилем соседних элементов используется символ плюса (+), который устанавливается между двумя селекторами:

Селектор1 + Селектор2 {Описание правил стиля}

**Дочерним селектором** считается такой, который в дереве элементов находится прямо внутри родительского элемента. Его синтаксис задается следующим образом:

Селектор1 > Селектор2 {Описание правил стиля}

Удобнее всего применять дочерние селекторы для элементов, которые обладают иерархической структурой (например, таблицы и списки).

**Обобщенный селектор одного уровня** выбирает элемент, имеющий общий с ним родительский элемент и указанный после него в исходном коде. Элементы не обязательно должны следовать непосредственно друг за другом.

Приведенное ниже правило выбирает все элементы h2, разделяющие один родительский элемент (например, section или article) с элементом h1 и стоящие в документе после него:

H1 ~ H2 {font-weight: normal;}

1. Ход выполнения работы

1. Создать документ по стандарту xhtml 1.0 strict. Создать файл css, содержащий следующие связанные стили:

отмену всех отступов;

шрифт Tahoma размером 14 точек по умолчанию;

схлопывание границ у таблиц;

внутренний отступ тела документа 10px.

2. Последовательно создать форму заказа, изображенную на рисунке 1.1:

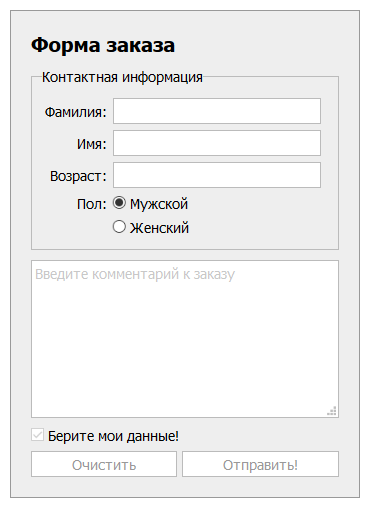


Рис. 1.1 – Форма заказа.

2.1. Создать контейнер div с классом order\_form, в котором будет размещаться форма со следующими параметрами:

ширина 308 точек;

внутренний отступ 20 точек;

фон цвета #eee;

рамка 1 точка, сплошная, цвета #999.

2.2. Внутри контейнера создать заголовок первого уровня:

размер шрифта: 20 точек,

жирный,

с нижним внутренним отступом в 10 точек.

При создании CSS селектора убедиться, что он влияет на заголовки, расположенные только в созданном ранее контейнере div.

2.3. За заголовком создать форму. Расположить в ней группу полей. Стиль группы:

внутренний отступ: 10 точек;

нижний внешний отступ: 10 точек;

рамка: 1 точка, сплошная, цвет #bbb.

2.4. Внутри группы создать таблицу 2x4 с прозрачной рамкой и внутренним отступом ячеек 3px. В ячейках первого столбца установить выравнивание по правому краю.

2.5. Создать текстовые поля внутри таблицы со стилем:

цвет текста: #999;

внутренний отступ 3 точки;

рамка: 1 точка, сплошная, цвет #bbb;

ширина: 200 точек.

2.6. Создать радиокнопки. Каждая кнопка с новой строки. По умолчанию выставить кнопку «Мужской».

2.7. Под группой полей создать текстовую область рамером 300x150 точек с замещающим текстом «Введите комментарий к заказу».

2.8. Создать отмеченный чекбокс, запрещенный к редактированию, с внешними отступами по вертикали в 10 точек;

2.9. Создать кнопки сброса и отправки формы со следующими стилями:

цвет фона белый;

внутренний отступ: по вертикали 3 точки, по горизонтали 40 точек;

цвет текста: #999;

внутренний отступ 3 точки;

рамка: 1 точка, сплошная, цвет #bbb.

Между кнопками выставить расстояние в 5 точек.

2.10. Убедиться, что все добавленные стили локализованы внутри контейнера формы.

1. Вопросы для самоконтроля и задачи
2. Таблицы стилей могут располагаться:

внутри html-документа, в разделе <head>;

внутри html-документа, в разделе <body>;

во внешнем файле;

все предыдущие ответы − правильные.

1. CSS предназначается для:

преобразования текстовых документов в документе HTML;

определения внешнего вида элементов, образующих html-документ;

cпособ разметки гипертекстовых документов.

1. Как подключить внешний файл CSS?

<style>stylesheet.css</style>

<style>@stylesheet.css</style>

<link rel="stylesheet" href=" stylesheet.css" />

<link>@import url(stylesheet.css)</link>

<link>stylesheet.css</link>

1. В записи A {color: orange; text-decoration: none} используется селектор:

по классу;

по имени;

по уникальному идентификатору.

1. Какой способ подключения лучше всего подходит, если стили нужно подключить к большому количеству файлов?

глобальные стили;

связанные стили;

внутренние стили;

экспортированные стили.

1. Соответствует ли данный фрагмент кода стандарту XHTML?

[code]

<form action="action.php" method="post">

<input type="text" value="Совет дня" />

<input type="checkbox" checked="checked" readonly />

<input type="submit" value="Узнать совет" />

</form>

[/code]

# Методические рекомендации по выполнению лабораторной работы № 2. Создание документа по макету с использованием табличной верстки

1. Цель работы

Целью выполнения работы является освоение обучающимися принципов построения интерфейса с помощью табличной верстки.

1. Условия и порядок проведения лабораторной работы

Для проведения занятия необходимо знание лекционного материала по теме лекции № 6. На занятии нужно выбрать макет страницы в формате psd и составить схему страницы. При составлении схемы страницы необходимо представить страницу как набор вложенных друг в друга таблиц. Сверстать документ с использованием таблиц, при формировании страницы использовать стили, характерные для табличной верстки. Занятие проводится в форме тренинга, в ходе которого сначала осуществляется перекрестный опрос по теме лекции № 6, затем обучающиеся выполняют задание индивидуально, последние 15 минут занятия они попарно проверяют код друг друга, выявляя ошибки и недочеты, формируя навыки по анализу корректности кода.

1. Необходимые теоретические сведения

Вёрстка html-документов − это процесс превращения работы дизайнера в web-страницу, отображаемую в браузере. Но один и тот же результат можно получить разными методами и уже от верстальщика зависит, какой из них предпочесть. От такого выбора зависит результат работы сайта, быстрота его отображения, доступность для разных устройств и браузеров. Вёрстка − это практическая дисциплина и только самостоятельно решая разные задачи и работая над проектами, можно освоить это искусство.

Как уже упоминалось ранее, HTML изначально задумывался как язык, которому не нужны средства оформления. Устаревшие версии браузеров не в полной мере поддерживали каскадные таблицы стилей, однако выпуск новых версий браузеров, поддерживающих, пусть и не в полной мере, актуальные версии CSS, серьёзно облегчил работу над сайтами. В итоге, начал происходить постепенный переход от табличной вёрстки к блочной или, как она ещё называется, вёрстке слоями, в которой расположение элементов на странице и их вид задаётся через стили.

В настоящее время многие верстальщики считают табличную вёрстку в каком-то смысле архаизмом. Тем не менее, существует ещё множество сайтов вроде yandex.ru, artlebedev.ru и других, свёрстанных именно с использованием таблиц. Такой консерватизм связан со следующими обстоятельствами:

1. Вёрстка слоями сложнее, чем таблицами, поскольку требует от разработчика глубоких знаний спецификации CSS.
2. Браузеры содержат разные ошибки при интерпретации стилевых свойств, поэтому следует знать особенности поведения основных браузеров, их ошибки и уметь обходить их.
3. Практически каждую задачу можно решить несколькими способами, от разработчика требуется выбрать оптимальный, т.е. наименее затратный по времени и усилиям. Для оценки оптимальности необходим практический опыт, чтобы иметь в запасе разные методы, применяемые в конкретных ситуациях.

Использование таблиц с невидимой границей − известный способ верстки, применяемый на множестве сайтов. Такая таблица фактически представляет собой модульную сетку, в которой удобно размещать отдельные элементы web-страницы. Простота и быстрота верстки, а также корректное отображение в различных браузерах − вот основные преимущества табличной верстки.

Благодаря универсальности таблиц и большому числу параметров, управляющих их видом, таблицы надолго стали определенным стандартом для верстки web-страниц. Таблица с невидимой границей представляет собой словно модульную сетку, в блоках которой удобно размещать элементы web-страницы. Тем не менее это не совсем правильный подход, ведь каждый объект HTML определен для своих собственных целей и, если он используется не по назначению, причем повсеместно, это значит, что альтернатив нет. Так оно и было долгое время, пока на смену таблицам при верстке сайтов не пришли слои. Это не значит, что слои теперь используются сплошь и рядом, но наметилась четкая тенденция − таблицы применяются для размещения табличных данных, а слои − для верстки и оформления.

### Преимущества табличной верстки

**Создание колонок.** Одноколонная модульная сетка применяется на сайтах достаточно редко. Действительно, основной текст и ссылки по сайту удобнее располагать в разных колонках, разделяя их тем самым. Таблицы же хорошо выступают в качестве многоколонной модульной сетки, каждая ячейка представляет собой отдельную колонку. Это позволяет легко создавать двух- и трехколонный макет документа. При изменении размера окна браузера, колонки сохраняют свой исходный вид, а не переносятся как слои друг под друга. К тому же высота разных колонок при использовании таблиц остается одинаковой, независимо от объема их содержимого.

**"Резиновый" макет.** Таблицы удачно подходят для "резинового" макета, ширина которого привязана к ширине окна браузера. Благодаря тому, что размер таблицы можно задавать в процентах, она занимает все отведенное ей свободное пространство. Также можно регулировать и высоту содержимого. Например, если текста немного, то "подвал" страницы может висеть в ее середине. Параметрами таблицы можно отрегулировать это так, что при небольшом тексте "подвал" плотно прилегает к нижнему краю окна браузера, независимо от размеров окна.

**"Склейка" изображений.** Рисунки часто разрезают на отдельные фрагменты, а затем собирают их вновь в одно целое, выкидывая одни фрагменты или заменяя их другими изображениями. Это требуется для различных дизайнерских изысков вроде создания эффекта перекатывания, анимации или уменьшения объема файлов. Таблицы позволяют легко обеспечить "склейку" нескольких рисунков в одно изображение. Каждая картинка помещается в определенную ячейку, параметры таблицы при этом устанавливаются такими, чтобы не возникло стыков между отдельными ячейками.

**Фоновые рисунки.** В ячейки таблицы разрешается добавлять фоновый рисунок, в зависимости от размеров ячейки он может повторяться по горизонтали, вертикали или сразу в двух направлениях. За счет этого приема на странице создаются декоративные линии, рамки самого разнообразного вида, добавляется тень под элементом.

**Выравнивание элементов.** Содержимое ячеек можно одновременно выравнивать по горизонтали и по вертикали, за счет чего расширяются возможности по размещению элементов относительно друг друга и на странице в целом.

**Особенности браузеров.** Браузеры достаточно вольно толкуют некоторые параметры CSS, поэтому создание универсального кода с применением слоев может стать настоящей головной болью для разработчиков. В этом смысле таблицы отображаются в разных браузерах практически одинаково, поэтому создание web-страниц упрощается.

### Недостатки табличной верстки

**Долгая загрузка.** Особенность таблиц такова, что пока последнее слово в самом низу таблицы не загрузится, на экране содержимое ячеек отображаться не будет. Браузеры используют такой подход, чтобы получить всю информацию о таблице для правильного форматирования ее содержимого. Но если таблица велика по высоте, может пройти достаточно много времени, прежде чем мы увидим нужную информацию. Существуют и способы обхода этого свойства, в частности, разбиение одной большой таблицы на несколько таблиц поменьше, а также использование стилевого свойства table-layout.

**Громоздкий код.** Таблицы содержат сложную иерархическую структуру вложенных тегов, которая увеличивает объем кода, и повышает сложность изменения отдельных параметров. В некоторых случаях для достижения желаемого результата приходится вкладывать одну таблицу внутрь другой, а это также влияет на размер "шелухи", т.е. кода, который не принимает непосредственного участия в отображении web-страницы.

**Плохая индексация поисковиками.** За счет того, что текст располагается в отдельных ячейках таблицы, в коде он может находиться достаточно далеко друг от друга. Такая раздробленность информации, а также значительная вложенность тегов затрудняет правильное индексирование страницы поисковыми системами. Как результат, документ не попадает в первую десятку выдачи запроса по ключевым словам, хотя вполне может и заслуживать это.

**Нет разделения содержимого и оформления.** В идеале HTML-код должен содержать только теги с указанием стилевого класса или идентификатора. А все оформление вроде цвета текста и положения элемента выносится в CSS и модифицируется отдельно. Такое разделение позволяет независимо править код страницы и менять вид отдельных ее элементов. Хотя к таблицам стиль легко добавляется, но обилие "лишних" тегов не позволяет действительно просто и удобно управлять видом отдельных компонентов страницы. К тому же не все параметры таблиц имеют свой стилевой синоним, поэтому в любом случае приходится обращаться к коду web-страницы и править его.

**Несоответствие стандартам.** В последнее время стандарты HTML и CSS прочно засели в умах web-разработчиков. Этому способствует развитие XHTML и XML, которые более "жестко" относятся к коду документа, появление новых версий браузеров, придерживающихся спецификации, и мода на верстку слоями. Что же говорит спецификация W3C относительно таблиц? А говорит она, что таблицы в первую и последнюю очередь нужны для размещения табличных данных. Все остальные способы использования таблиц осуждаются.

Работа с таблицами

**Вложенные таблицы.** Одну таблицу допускается помещать внутрь ячейки другой таблицы. Это требуется для представления сложных данных или в том случае, когда одна таблица выступает в роли модульной сетки, а вторая, внутри нее, в роли обычной таблицы.

**Размеры таблицы.** Размеры таблицы изначально не устанавливаются и вычисляются на основе содержимого ячеек. В итоге суммарная ширина таблицы складывается из следующих параметров:

* ширина содержимого ячеек;
* толщина всех границ между ячеек;
* поля вокруг содержимого, устанавливаемые через атрибут cellpadding;
* расстояние между ячейками, которое определяется значением cellspacing.

Если для таблицы установлена её ширина в процентах или пикселах, то содержимое таблицы подстраивается под указанные размеры. Так, браузер автоматически добавляет переносы строк в текст, чтобы он полностью поместился в ячейку, и при этом ширина таблицы осталась без изменений. Бывает, что ширину содержимого ячейки невозможно изменить, как это, например, происходит с рисунками. В этом случае ширина таблицы увеличивается, несмотря на указанные размеры. Чтобы избежать указанной ситуации применяют несколько средств:

* не добавляют в ячейку фиксированной ширины те изображения, размер которых превышает ширину ячейки. Способ, конечно, звучит банально, тем не менее, зная особенности ячеек, можно избежать неприятностей с их отображением;
* для тега <table> используют стилевое свойство table-layout со значением fixed. Применение этого свойства позволяет обрезать рисунок, если он не помещается целиком в ячейку.

Сделать единообразный вид таблицы во всех браузерах легко, для этого к ячейке с рисунком следует добавить свойство overflow со значением hidden. При этом всё, что не помещается в ячейку, будет "обрезано".

Ячейки в одной строке взаимосвязаны и их высота одинакова. Это позволяет делать макеты с колонками одной высоты. Порядок ячеек. Основой таблицы выступает строка и ячейка, формирование таблицы происходит слева направо и сверху вниз (рисунок 2.1).

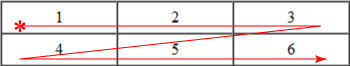


Рис. 2.1. Порядок создания ячеек

Неудобства этой схемы проявляются при активном использовании колонок и большом числе ячеек. Конечно, есть теги <col> и <colgroup>, но их возможности ограничены. Вставка новых ячеек или редактирование существующих может привести к ошибкам отображения таблицы.

**Загрузка таблицы.** Пока таблица не загрузится полностью, её содержимое не начнёт отображаться. Дело в том, что браузер, прежде чем показать содержимое таблицы, должен вычислить необходимые размеры ячеек, их ширину и высоту. А для этого нужно знать, что в этих ячейках находится. Поэтому браузер и ожидает, пока загрузится все, что находится в ячейках, и только потом отображает таблицу.

Исходя из этого факта, таблицы не используют для хранения большой информации. А чтобы ускорить загрузку табличного макета, его разбивают на отдельные таблицы или используют свойство table-layout, применение которого позволяет несколько повысить скорость отображения содержимого таблицы. В обычной таблице браузер анализирует все ячейки и затем уже изменяет ширину колонок на основе этой информации. Включение table-layout со значением fixed меняет алгоритм расчета − браузер анализирует только первую строку и ширину колонок строит согласно ей. За счёт уменьшения числа вычислений и происходит выигрыш скорости отображения таблицы в целом.

### Основные стили, необходимые для табличной верстки

**Цвет фона ячеек.** Цвет фона одновременно всех ячеек таблицы устанавливается через свойство background, которое применяется к селектору table. При этом следует помнить о правилах использования стилей, в частности, наследовании свойств элементов. Хотя свойство background не наследуется, для ячеек значением фона по умолчанию выступает transparent, т.е. прозрачность, поэтому эффект заливки фона получается и у ячеек. Если одновременно с table задать цвет у селектора td или th, то этот цвет будет установлен в качестве фона ячейки.

**Поля внутри ячеек.** Полем называется расстояние между краем содержимого ячейки и её границей. Обычно для этой цели применяется атрибут cellpadding тега **<**table**>**. Он определяет значение поля в пикселах со всех сторон ячейки. Допускается использовать стилевое свойство padding, добавляя его к селектору td.

Если применяется стилевое свойство padding для ячеек таблицы, то действие атрибута cellpadding тега **<**table**>** игнорируется.

**Расстояние между ячейками.** Для изменения расстояния между ячейками применяется атрибут cellspacing тега **<**table**>**. Влияние этого атрибута хорошо заметно при использовании границ вокруг ячеек или при заливке ячеек цветом, который выделяется на фоне страницы. В качестве замены cellspacing выступает стилевое свойство border-spacing, оно устанавливает расстояние между границами ячеек. Одно значение устанавливает одновременно расстояние по вертикали и горизонтали между границами ячеек. Если значений у этого свойства два, то первое определяет горизонтальное расстояние (т.е. слева и справа от ячейки), а второе − вертикальное (сверху и снизу).

Свойство border-spacing действует только в том случае, если для селектора table не задано свойство border-collapse со значением collapse.

Браузер Internet Explorer до седьмой версии включительно не поддерживает свойство border-spacing, поэтому в этом браузере для таблиц будет применяться значение cellspacing заданное по умолчанию (обычно оно равно 2px).

При добавлении к селектору table свойства border-collapse со значением collapse, атрибут cellspacing игнорируется, а значение border-spacing обнуляется.

Границы и рамки. По умолчанию границы в таблице изначально нет, а её добавление происходит с помощью атрибута border тега <table>. Браузеры по-разному отображают такую границу, поэтому лучше не указывать этот атрибут вообще, а рисование границ возложить на стили. Рассмотрим два метода, непосредственно связанных со стилями.

Использование атрибута cellspacing. Известно, что атрибут cellspacing тега <table> задаёт расстояние между ячейками таблицы. Если используется разный цвет фона таблицы и ячеек, то между ячейками возникнет сетка линий, цвет которых совпадает с цветом таблицы, а толщина равна значению атрибута cellspacing в пикселах. Это не совсем удобный способ создания границ, поскольку он имеет ограниченную область применения. Так можно получить только одноцветную сетку, а не вертикальные или горизонтальные линии в нужных местах.

Применение свойства border. Стилевое свойство border одновременно устанавливает цвет границы, её стиль и толщину вокруг элемента. Когда требуется создать отдельные линии на разных сторонах, лучше использовать производные − border-left, border-right, border-top и border-bottom, эти свойства соответственно определяют границу слева, справа, сверху и снизу.

Применяя свойство border к селектору table, мы добавляем рамку вокруг таблицы в целом, а к селектору td или th − рамку вокруг ячеек.

Выравнивание содержимого ячеек. По умолчанию текст в ячейке таблицы выравнивается по левому краю. Исключением из этого правила служит тег <th>, он определяет заголовок, в котором выравнивание происходит по центру. Чтобы изменить способ выравнивания применяется стилевое свойство text-align.

Выравнивание по вертикали в ячейке всегда происходит по её центру, если это не оговорено особо. Это не всегда удобно, особенно для таблиц, у которых содержимое ячеек различается по высоте. В таком случае выравнивание устанавливают по верхнему краю ячейки с помощью свойства vertical-align.

Пустые ячейки. Браузеры иначе отображают ячейку, внутри которой ничего нет. "Ничего" в данном случае означает, что внутрь ячейки не добавили ни рисунок, ни текст, причём пробел в расчёт не принимается. Естественно, вид ячеек различается только в том случае, если вокруг них установлена граница. При использовании невидимой рамки, вид ячеек, независимо от того, есть в них что-нибудь или нет, совпадает.

Старые браузеры не отображали цвет фона пустых ячеек вида <td bgcolor="#ffcc00"></td>, поэтому в том случае, когда требовалось оставить ячейку без содержимого, но отобразить цвет фона, внутрь ячейки добавляли неразделяемый пробел (&nbsp;). Пробел не всегда подходит, особенно когда нужно установить высоту ячейки 1–2 пиксела, из-за чего широкое распространение получил однопиксельный прозрачный рисунок. Действительно, такой рисунок можно масштабировать на свое усмотрение, но он на web-странице никак не отображается.

К счастью эпоха однопиксельных рисунков и всяческих распорок на их основе прошла. Браузеры достаточно корректно работают с таблицами и без присутствия содержимого ячеек.

Для управления видом пустых ячеек используется свойство empty-cells, при значении hide граница и фон в пустых ячейках не отображается. Если все ячейки в строке пустые, то строка прячется целиком. Ячейка считается пустой в следующих случаях:

* нет вообще никаких символов;
* в ячейке содержится только перевод строки, символ табуляции или пробел;
* значение visibility установлено как hidden.

Добавление неразрывного пробела &nbsp; воспринимается как видимое содержание, т.е. ячейка уже будет не пустой.

### Верстка с помощью таблиц

**Макет страницы.** Верстальщик получает макет страницы в виде графического файла − например, в формате GIF, PNG или JPEG. Если макет сложный и насыщенный изображениями, он может быть представлен в формате PSD, чтобы из него было удобно извлекать отдельные графические компоненты. Текстовое содержание и изображения для web-страницы также часто прилагаются к макету в отдельных файлах.

**Спецификация.** Кроме макета, дающего только общее представление о будущей странице, необходима спецификация − точное описание параметров страницы. Спецификация так же необходима для создания web-страницы, как чертежи − для постройки здания.

Спецификация задаёт характеристики страницы, которые нельзя точно определить непосредственно из макета, например:

* адреса гиперссылок;
* отступы между компонентами страницы;
* кегль шрифтов;
* толщину линий;
* специальные эффекты (например, изменённый интерлиньяж или межбуквенный интервал);

Спецификация также определяет поведение интерактивных элементов на вебстранице, в частности:

* как посещённые гиперссылки должны отличаться от непосещённых;
* как должны вести себя гиперссылки при наведении указателя мыши и активизации;
* как различные элементы управления (текстовые поля, кнопки и т. д.) должны реагировать на различные действия пользователя (наведение указателя мыши, ввод текста, щелчок и т. д.);
* должны ли другие компоненты web-страницы реагировать на действия пользователя (например, при наведении указателя мыши на некоторые компоненты может всплывать подсказка; пользователь может изменять размер некоторых компонентов или перетаскивать их).

Спецификация обычно не задаёт характеристики, которые непосредственно видны из макета, − например, точные коды цветов (их всегда можно узнать, просмотрев макет в графическом редакторе) или гарнитуры шрифтов.

Перед тем как начать разрабатывать код в соответствии со спецификацией, надо убедиться, что она составлена грамотно. Бывает, что в спецификации не хватает данных, − например, в макете видны отступы, но ни из макета, ни из спецификации нельзя узнать их величину. То же самое иногда происходит с кеглем шрифта и толщиной рамок. В этих случаях недостаточно квалифицированные разработчики устанавливают размеры "на глаз", тем самым усугубляя ошибку спецификации.

**Фиксированные размеры.** Размеры web-страницы и её отдельных компонентов должны подстраиваться под размеры окна браузера и другие настройки на компьютерах пользователей. Поэтому все размеры (кроме толщины рамок, размеров растровых изображений и некоторых экзотических объектов) должны задаваться в масштабируемых единицах. Например, размеры шрифта уместно задавать в относительных единицах еm. Отступы также можно задавать в еm (тогда они будут масштабироваться при изменении размера шрифта) или в процентах (тогда они будут масштабироваться при изменении ширины контейнера). В некоторых спецификациях можно встретить грубые нарушения этого принципа: например, требуется, чтобы ширина страницы составляла ровно 1024 пиксела, или чтобы кегль шрифта был ровно 16 пикселов. Грамотный кодировщик не станет верстать страницу по таким спецификациям.

**Нарушение принципов практичности и удобства использования.** В спецификации могут быть требования, делающие страницу неудобной для пользователей. Например, плохая спецификация может потребовать, чтобы гиперссылки никак внешне не отличались от основного текста, или чтобы на странице присутствовала анимация, которая запускается без запроса пользователя.

При разработке кода методом табличной вёрстки рекомендуется придерживаться стандартной последовательности действий:

* составить схему страницы по графическому макету;
* определить структуру таблиц;
* составить код HTML в соответствии со структурой таблиц;
* задать оформление CSS для страницы и элементов таблиц;
* заполнить содержанием и оформить ячейки таблиц по отдельности.

**Разрезка изображений при верстке.** Разрезание изображения на фрагменты с последующим их объединением в одну целую картинку − давний прием, вошедший в арсенал вёрстки web-страниц. Предварительно подготовленный рисунок разрезают на части в графическом редакторе, сохраняют части как отдельные графические изображения, а затем соединяют их вместе с помощью таблицы. Возможности верстки с разрезкой изображений:

1. **Создание ссылок.** Отдельные рисунки при необходимости можно превращать в ссылки, причём для них можно назначать своё описание (атрибут title) и альтернативный текст (атрибут alt), который виден при отключении показа картинок в браузере или при наведении курсора мыши на изображение.
2. **Эффект перекатывания.** Набор отдельных фрагментов позволяет создавать эффект перекатывания − динамическое изменение одного рисунка на другой при наведении на него курсора мыши, и обратно на прежний, когда курсор уводится прочь.
3. **Уменьшение объема файлов.** Отдельными частями изображения удобней манипулировать, подбирая для них графический формат и его параметры таким образом, чтобы объём файла был минимален при сохранении приемлемого качества изображения. В итоге набор графических файлов будет занимать меньше места, и загружаться быстрее, чем один файл, содержащий целый рисунок.
4. **Анимированный GIF.** Использование анимированного GIF-a для изображений большого размера чревато существенным увеличением объёма файла. Но можно пойти на хитрость и заменить лишь часть изображения анимацией, а остальные фрагменты оставить статичными. При этом общий объём нескольких файлов будет гораздо меньше, чем анимирование одного изображения.
5. **Конструктор из деталей изображения.** Изображения на web-странице по своей природе прямоугольны, но, разрезав один рисунок на составляющие элементы, получим конструктор, из которого можно сложить другую фигуру. Это напоминает детские кубики, на одну из сторон которых наклеена картинка. Складывать подобные фигуры на web-странице требуется в силу разных причин, например, вместо фрагмента изображения требуется добавить текст. Кроме того, некоторые рисунки можно заменить их фоновым аналогом и тогда конечное изображения, сохраняя свою целостность, будет занимать всю доступную область документа.
6. **Психологический аспект загрузки.** Когда один рисунок состоит из множества фрагментов, то браузер скачивает их в несколько потоков и показывает те, которые загрузились в первую очередь. Поэтому изображение появляется как элементы мозаики. А это сразу привлекает внимание и кажется, что загрузка происходит быстрее. Так что с технической стороны один рисунок грузится быстрее, а с позиции человеческого восприятия кажется, что набор маленьких рисунков быстрее появляется.

### Особенности табличной вёрстки

Составление схемы по макету − наиболее сложный этап вёрстки, т. к. требует опыта и, зачастую, творческого подхода. Принципы разделения страницы на прямоугольные участки помогают в работе, но не обеспечивают точный алгоритм преобразования графического макета в табличный код. Если бы такой алгоритм существовал, составление схем давно было бы поручено программным средствам. Опытный разработчик, взглянув на макет, мысленно представляет себе схему, причём иногда не одну. Это умение развивается на практике по мере приобретения собственного опыта и изучения чужих хороших сайтов. Не забывайте, что у многих элементов таблиц есть оформление по умолчанию. Применяя табличную вёрстку, учитывайте следующие особенности:

* Ширина таблицы определяется её содержанием. Чтобы увеличить ширину таблицы до ширины её контейнера (например, элемента body, как в последнем примере), можно задать для элемента table объявление width: 100%;.
* Таблица отображается без рамок. Чтобы создать рамки, пользуйтесь собирательным свойством border или его частными свойствами, которые можно применять к элементам table, th, tr, caption.
* Между ячейками таблицы есть зазоры. Чтобы устранить их, задайте объявление border-collapse: collapse; для элемента table или вообще для всех элементов (с помощью универсального селектора, как сделано в примере).
* У элемента body есть внешние отступы, поэтому даже при ширине 100% таблица не достигнет краёв окна браузера. Чтобы отменить внешние отступы, задайте объявление margin: 0; (лучше всего − не только для body, но и для всех элементов).
* В ячейках таблиц (элементах th и td) есть внутренние отступы размером   
  1 пиксел. Чтобы отменить их, задайте объявление padding: 0; (лучше всего − не только для ячеек, но и для всех элементов).
* Содержание ячеек th горизонтально выравнивается по центру и отображается полужирным шрифтом. Чтобы изменить это оформление, используйте свойства text-align и font-weight.
* Содержание ячеек th и td вертикально выравнивается по центру. Чтобы изменить это оформление, используйте свойство vertical-align − например, со значением top.
* Для изображения (элемента img) или другого замещаемого элемента, вложенного в ячейку таблицы, свойство vertical-align имеет значение baseline, поэтому под элементом может оказаться зазор. Чтобы изображение плотно примыкало к краям ячейки, можно сделать его блочным (display: block;) или задать другое значение vertical-align (например, bottom).
* Если в ячейке таблицы есть содержание, её высота равна как миним ум высоте строки, которая, в свою очередь, составляет 120% от размера шрифта. Это тоже иногда приводит к лишним отступам, так что приходится уменьшать значение свойства font-size, в том числе до 0.
* Иногда целесообразно вкладывать целые таблицы в ячейки других таблиц. Это может понадобиться при особенно сложной композиции страницы или если обычные таблицы с данными вкладываются в таблицы для вёрстки.

1. Ход выполнения работы

1. Выбрать макет страницы в формате psd. В выбранном макете должны быть представлены следующие элементы:

шапка (с дополнительными элементами, например, поле поиска, логотип, форма, иконки, контакты и т.п.);

главное меню;

несколько колонок различной ширины с контентом различной структуры;

подвал (с дополнительными элементами, например, блоки ссылок, контакты, форма, иконки и т.п.).

2. Составить схему страницы. При составлении схемы страницы необходимо представить страницу как набор вложенных друг в друга таблиц. Пример схемы приведен на рисунке 2.1.

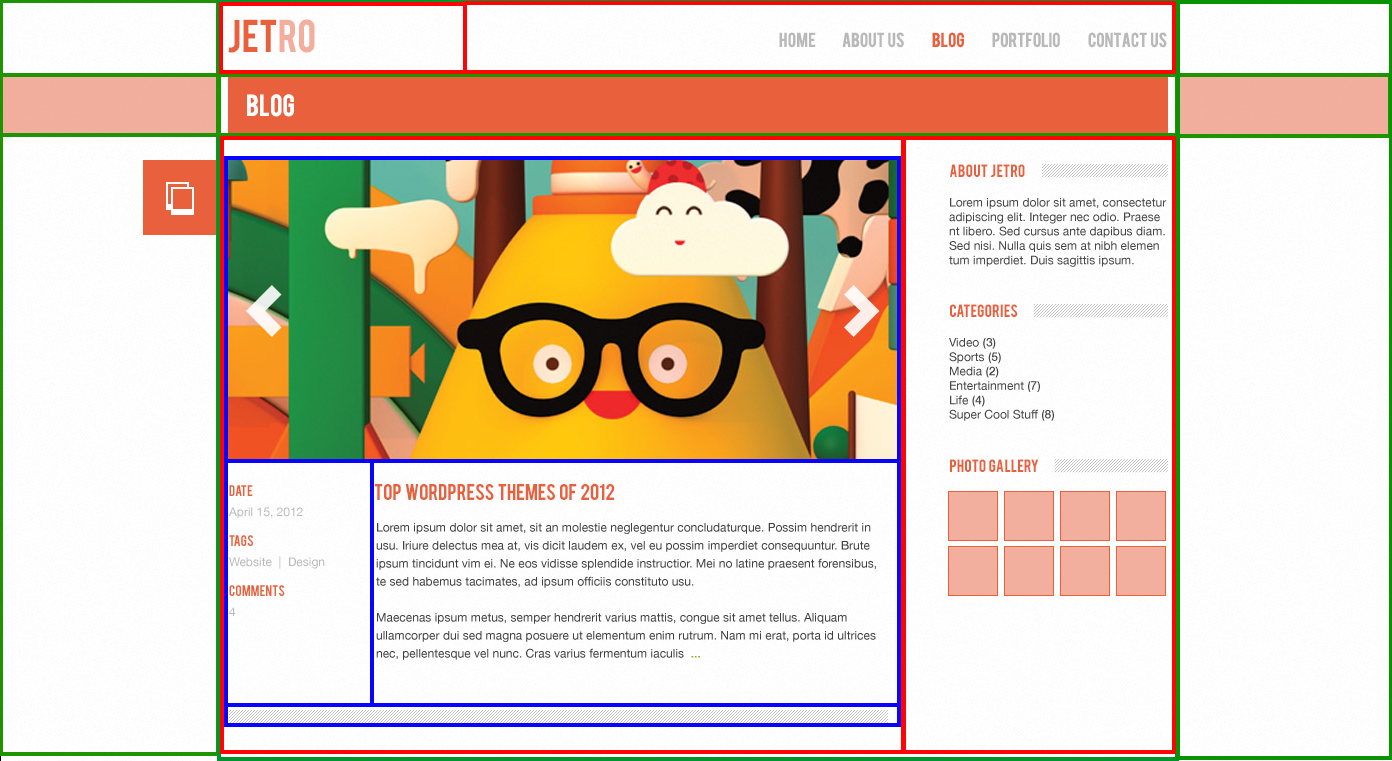


Рис. 2.1 – Схема страницы.

На данной схеме весь документ сформирован зеленой таблицей. В нее вложена красная таблица, а в красную – синяя.

3. Создать структуру проекта, представленную на рисунке 2.2.:

font (с дополнительными элементами, например, поле поиска, логотип, форма, иконки, контакты и т.п.);

images (для изображений, относящихся к дизайну страницы);

sources (исходный макет в формате png, класть psd-файл не нужно);

index.html (файл с html-кодом страницы);

style.css (файл со связанными стилями).

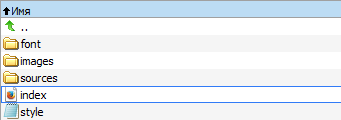


Рис. 2.2 – Структура проекта.

4. Создать в css-файле стили по умолчанию, характерные для табличной верстки. При таком подходе к верстке целесообразно установить ширину таблиц по умолчанию в 100% и выравнивание в ячейках по верхнему краю. Так же имеет смысл убрать отступы между ячейками.

5. Сверстать документ с использованием таблиц. При формировании верстки учесть следующие требования:

все стилевые преобразования осуществить с помощью css-правил, вынесенных в связанный стилевой файл;

не использовать стилевые атрибуты тегов, например: <table width="100"> (вместо этого использовать css: table {width:100px;});

разделить макет на смысловые блоки, для содержимого блока использовать смысловые class и id, стараться уменьшить количество используемых определений в html коде;

каждый смысловой блок закомментировать, например <!-- главное меню --> в html и /\*главное меню\*/ в css; перед комментарием – пробельная строка;

использовать смещение на 1 табуляцию для вложенных блоков;

важно добиться, чтобы верстка не искажалась при добавлении большого количества текста в любой блок верстки;

верстка должна соответствовать стандарту xhtml 1.0, можно использовать онлайн валидатор для проверки кода (например, http://validator.w3.org).

1. Вопросы для самоконтроля и задачи
2. Что представляет из себя верстка с помощью таблиц?
3. Каковы недостатки табличной вестки?
4. Назовите основные преимущества табличной верстки.
5. Зачем и как производится «склейка» изображений?
6. Какие особенности характерны для табличной верстки?

легко выравнивать по высоте;

сложно выравнивать по высоте;

компактный код;

громоздкий код;

возможно отображение по мере загрузки документа.

1. Какие особенности характерны для табличной верстки?

строгий порядок следования элементов;

элементы на странице могут располагаться не в порядке их следования в документе;

ширина контейнера зависит от содержимого;

контейнер занимает всю доступную ширину;

высота колонок разная, в зависимости от содержания;

высота колонок связана, их высота одинакова.

1. Какие особенности характерны для табличной верстки?

легко выравнивать по высоте;

сложно выравнивать по высоте;

возможно отображение по мере загрузки документа;

отображение страницы требует загрузки всего документа;

высота колонок разная, в зависимости от содержания;

высота колонок связана, их высота одинакова.

# Методические рекомендации по выполнению лабораторной работы № 3. Создание документа по макету с использованием блочной верстки

1. Цель работы

Целью выполнения работы является освоение обучающимися принципов построения интерфейса с помощью блочной верстки.

1. Условия и порядок проведения лабораторной работы

Для проведения занятия необходимо знание лекционного материала по теме лекции № 7. В ходе лабораторной работы нужно составить схему страницы по выбранному ранее макету формата psd. При составлении схемы страницы необходимо представить страницу как набор последовательно размещенных блоков. Сверстать документ с использованием слоев, использовать при этом стили, характерные для верстки слоями. Тег <table> использовать только для табличных данных.Занятие проводится в форме тренинга, в ходе которого сначала осуществляется перекрестный опрос по теме лекции № 7, затем обучающиеся выполняют задание индивидуально, последние 15 минут занятия они попарно проверяют код друг друга, выявляя ошибки и недочеты, формируя навыки по анализу корректности кода.

1. Необходимые теоретические сведения

### Строчные и блочные области

Области элементов делятся на две основные категории: строчные и блочные области. Строчная область (**inline box**) размещается на одной строке с соседними строчными областями. Элементы **а** и **img** (a также **em**, **strong** и многие другие) no умолчанию порождают строчные области. Строчная область имеет форму прямоугольника, который целиком размещается внутри одной строки. Блочные элементы всегда начинаются с новой строки, а строчные таким способом не акцентируются.

Блочная область (**block box**) всегда имеет форму одного прямоугольника, перед которым и после которого переводится строка. Таким образом, блочная область, в отличие от строчной, по умолчанию не отображается рядом с другими областями. Расположить рядом с блочной областью другую блочную или строчную область можно с помошью специальных средств CSS − например, плавающих элементов или абсолютного позиционирования.

Разница между блочными и строчными элементами следующая: строчные элементы могут содержать только данные или другие строчные элементы, а в блочные допустимо вкладывать другие блочные элементы, строчные элементы, а также данные. Иными словами, строчные элементы никак не могут хранить блочные элементы.

Блочные элементы занимают всю доступную ширину, например, окна браузера, а ширина строчных элементов равна их содержимому плюс значения отступов, полей и границ.

### Характеристика блочной верстки

Несмотря на то, что термин "слой" достаточно устойчиво используется при разработке сайта, расшифровке этого понятия в литературе по построению практически не уделяется внимание. В дальнейшем мы будем активно использовать термин "слой", поэтому вначале необходимо определить, что же под ним подразумевается.

Первоначально слои ввела компания Netscape, включив в свой браузер поддержку тега <layer>. Этот тег позволял прятать/показывать текущее содержимое, устанавливать положение относительно окна браузера, накладывать один слой поверх других и загружать данные в содержимое слоя из файла. Что характерно, все эти параметры легко менялись с помощью JavaScript и это расширяло возможности по созданию действительно динамического контента на странице. Несмотря на столь впечатляющий набор возможностей, тег <layer> не был включен в спецификацию HTML и так и остался лишь расширением браузера Netscape.

Однако необходимость в указанных возможностях, которые бы применялись на сайтах, уже назрела, и в конце 1996 года синтаксис для работы со слоями был разработан и одобрен в рабочем проекте W3C консорциума "CSS Positioning (CSS-P)". Основная нагрузка ложилась на стили, с их помощью можно управлять видом любого элемента, в том числе менять значения динамически через JavaScript. К сожалению, объектные модели браузеров для доступа к элементам разнились, поэтому приходилось писать достаточно сложный код, который бы учитывал эти различия.

В настоящее время разработчики популярных браузеров стали придерживаться спецификаций HTML и CSS, что сильно облегчило жизнь создателям сайтов, поскольку снизило время на отладку сайта в разных браузерах. Тем не менее, различия в подходах у браузеров существуют и при их возникновении разработчики придерживаются следующих форм работы. Верстка с помощью слоев заключается в конструктивном использовании тегов <div> и стилей. При этом придерживаются следующих принципов:

1. **Разделение содержимого и оформления.** Код HTML должен содержать только теги разметки и теги логического форматирования, а любое оформление выносится за пределы кода в стили. Такой подход позволяет независимо управлять видом элементов страницы и ее содержимым. Благодаря этому над сайтом может работать несколько человек, при этом каждый выполняет свою функцию самостоятельно от других. Дизайнер, верстальщик и программист работают над своими задачами автономно, снижая время на разработку сайта.
2. **Активное применение тега <div>.** При использовании слоев существенное значение уделяется универсальному тегу <div>, который выполняет множество функций. Фактически это основа, на которую "навешиваются" стили, превращая ее то в игрушку, то в зверушку. Совершенно не значит, что применяется только один этот тег, нужно ведь и рисунки вставлять и оформлять текст. Но при верстке с помощью слоев тег <div> является кирпичиком верстки, ее базовым фундаментом. Благодаря этому тегу HTML-код распадается на ряд четких наглядных блоков, за счет чего верстка слоями называется также блочной версткой. Код при этом получается более компактным, чем при табличной верстке, к тому же поисковые системы его лучше индексируют.
3. **Таблицы применяются только для представления табличных данных.** При верстке слоями, конечно же, используются таблицы, но только в тех случаях, когда они нужны, например, для наглядного отображения чисел и других табличных данных. Вариант, когда от таблиц предлагается отказаться вообще, является нецелесообразным и, более того, вредным. Использование стилей не является обязательной характеристикой верстки слоями, и для табличной верстки стили могут применяться достаточно активно. Другое дело, что это становится стандартом де-факто и без стилей теперь никак не обойтись.

### Особенности верстки слоями

**Пример блока.** Любой блочный элемент состоит из набора свойств, подобно капустным листам накладываемых друг на друга. Основой блока выступает его контент (это может быть текст, изображение и др.), ширина которого задается свойством width, а высота через height; вокруг контента идут поля (padding), они создают пустое пространство от контента до внутреннего края границ; затем идут собственно сами границы (border) и завершают блок отступы (margin), невидимое пустое пространство от внешнего края границ. Порядок влияния этих свойств на блок четко определён и не может быть нарушен. На рисунке 3.1 показан блок в виде набора этих свойств.

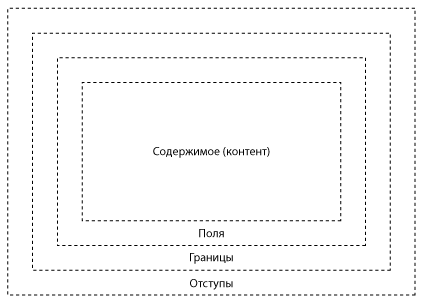


Рис. 3.1. Свойства, формирующие блочный элемент

На деле ни одно из этих свойств не является обязательным, в том числе и контент, поэтому вы можете формировать любые блоки, содержащие комбинации свойств padding, border и margin или вообще обойтись без них.

Расстояние от внутреннего края границы или края блока до воображаемого прямоугольника, ограничивающего содержимое блока, называют полем. Из-за того, что значения полей могут различаться на каждой стороне, применяют выражения "верхнее поле" или "поле сверху", и им подобные для других сторон. Обозначение "поля" следует понимать как одинаковое значение полей для всех сторон. Основное предназначение полей − создать пустое пространство вокруг содержимого блочного элемента, например, текста, чтобы он не прилегал плотно к краю элемента. Использование полей повышает читабельность текста и улучшает внешний вид страницы. Значения полей не могут быть отрицательными!

Линии вокруг полей элемента на одной, двух, трёх или всех четырёх его сторонах называют границами. У каждой линии есть толщина, стиль и цвет. Для создания рамки применяется универсальное свойствоborder**,** одновременно задающее все эти параметры, а для создания линий на отдельных сторонах элемента можно воспользоваться свойствами border-left, border-top, border-right и border-bottom, соответственно устанавливающих границу слева, сверху, справа и снизу. Расстояние от линии до текста регулируется значением padding.

Отступом называется пустое пространство от внешнего края границы, полей или содержимого блока. Как уже упоминалось, границы с полями не обязательны и могут отсутствовать, так что способ формирования отступов зависит от ситуации. Как и в случае с полями, применяют выражения "верхний отступ" или "отступ сверху", и им подобные для других сторон. Обозначение "отступы" следует понимать как одинаковое значение отступов для всех сторон. Для отступов характерны следующие особенности:

* отступы прозрачны, на них не распространяется цвет фона или фоновая картинка, заданная для блока. Однако если фон установлен у родительского элемента, он будет заметен и на отступах;
* отступы в отличие от полей могут принимать отрицательное значение, это приводит к сдвигу всего блока в указанную сторону. Так, если задано margin-left: -10px, это сдвинет блок на десять пикселов влево;
* для отступов характерно явление под названием "схлопывание", когда отступы у близлежащих элементов не суммируются, а объединяются меж собой;
* отступы, заданные в процентах, вычисляются от ширины контента блока. Это касается как вертикальных, так и горизонтальных отступов.

**Вычисление ширины и высоты блока.** Ширина блока − это комплексная величина и складывается из нескольких значений свойств:

* width − ширина контента, т.е. содержимого блока;
* padding-left и padding-right − поле слева и справа от контента;
* border-left и border-right − толщина границы слева и справа;
* margin-left и margin-right − отступ слева и справа.

Если значение width не задано, то оно по умолчанию устанавливается как auto. В этом случае ширина блока будет занимать всю доступную ширину при сохранении значений полей, границ и отступов. Под доступной шириной в данном случае подразумевается ширина контента у родительского блока, а если родителя нет, то ширина контента браузера:

Допустим, для слоя написан следующий стиль:

width: 300px; /\* Ширина контента \*/

margin: 7px; /\* Значение отступов \*/

border: 4px solid black; /\* Параметры границы \*/

padding: 10px; /\* Поля вокруг текста \*/

Ширина блока согласно этой записи будет равна 342 пиксела, эта величина получается складыванием значения ширины контента плюс отступ слева, граница слева и поле слева, плюс поле справа, граница справа и отступ справа. Суммируем все числа:

Ширина = 300 + 7 + 7 + 4 + 4 + 10 + 10 = 342

Надо отметить, что блочная модель с формированием ширины несет в себе кучу неудобств. Постоянно приходится заниматься вычислениями, когда требуется задать определенную ширину блока. Также начинаются проблемы при сочетании разных единиц измерения, в частности, процентов и пикселов. Предположим, что ширина контента задана как 90%, если сюда приплюсовать поля и границы, заданные в пикселах, то нельзя вычислить суммарную ширину блока, поскольку проценты напрямую в пикселы не переводятся (в CSS3 так суммировать можно, только поддерживается эта возможность далеко не всеми браузерами). В итоге может получиться так, что общая ширина блока превысит ширину web-страницы, что приведёт к появлению горизонтальной полосы прокрутки. Выходов из подобной ситуации два − поменять алгоритм блочной модели и воспользоваться вложенными слоями.

1. Ход выполнения работы

1. Выбрать макет страницы в формате psd. В выбранном макете должны быть представлены следующие элементы:

шапка (с дополнительными элементами, например, поле поиска, логотип, форма, иконки, контакты и т.п.);

главное меню;

несколько колонок различной ширины с контентом различной структуры;

подвал (с дополнительными элементами, например, блоки ссылок, контакты, форма, иконки и т.п.).

2. Составить схему страницы. При составлении схемы страницы необходимо представить страницу как набор последовательно размещенных блоков. Пример схемы приведен на рисунке 3.2.

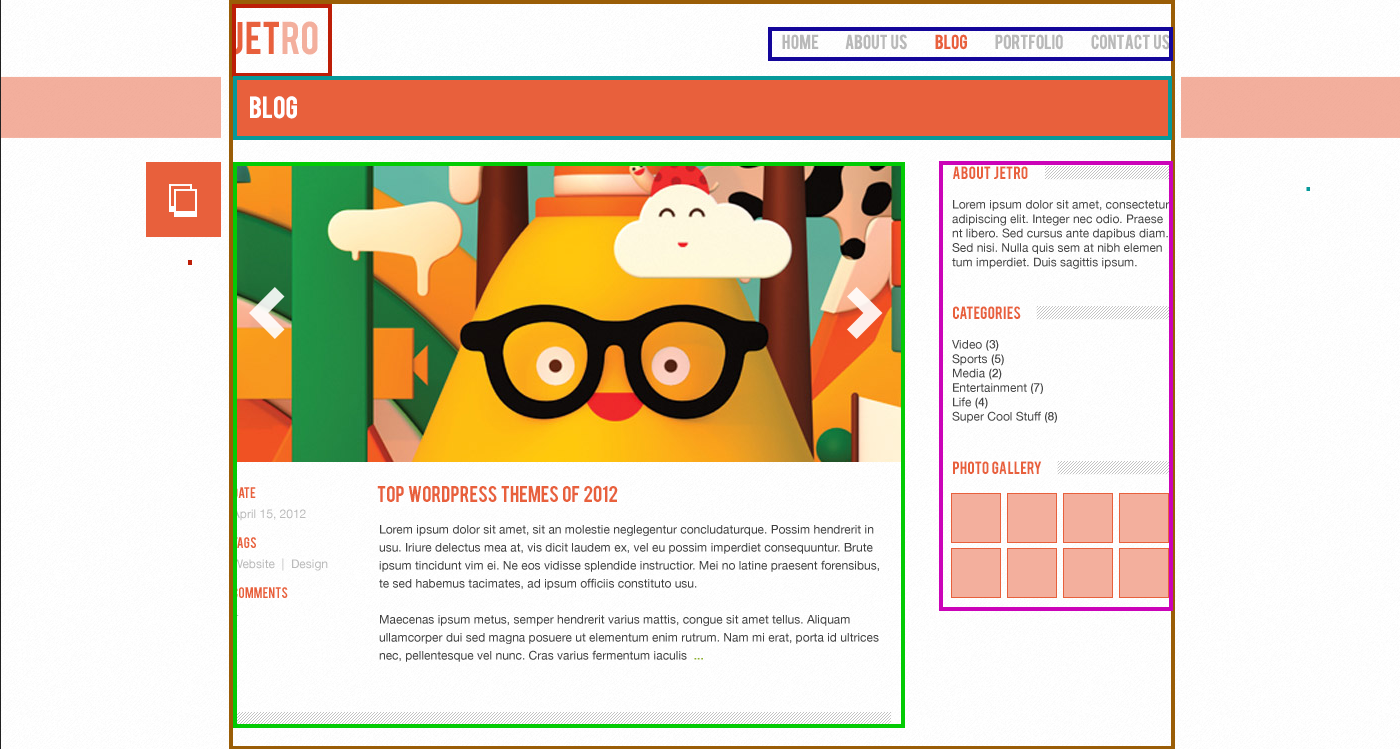


Рис. 3.2 – Схема страницы.

На данной схеме основной каркас основной части документа заключен в темно- коричневый блок. Внутри него последовательно расположены красный, синий, бирюзовый, зеленый и розовый блоки.

3. Создать структуру проекта, представленную на рисунке 3.3:

font (с дополнительными элементами, например, поле поиска, логотип, форма, иконки, контакты и т.п.);

images (для изображений, относящихся к дизайну страницы);

sources (исходный макет в формате png, класть psd-файл не нужно);

index.html (файл с html-кодом страницы);

style.css (файл со связанными стилями).

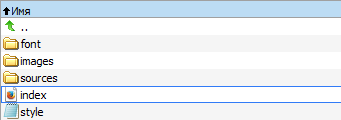


Рис. 3.3 – Структура проекта.

4. Создать в css-файле стили по умолчанию, характерные для верстки слоями. При таком подходе к верстке целесообразно убрать все отступы у элементов.

5. Сверстать документ с использованием слоев. Тег <table> использовать только для табличных данных. При формировании верстки учесть следующие требования:

все стилевые преобразования осуществить с помощью css-правил, вынесенных в связанный стилевой файл;

не использовать стилевые атрибуты тегов, например: <table width="100"> (вместо этого использовать css: table {width:100px;});

разделить макет на смысловые блоки, для содержимого блока использовать смысловые class и id, стараться уменьшить количество используемых определений в html коде;

каждый смысловой блок закомментировать, например <!-- главное меню --> в html и /\*главное меню\*/ в css; перед комментарием – пробельная строка;

использовать смещение на 1 табуляцию для вложенных блоков;

важно добиться, чтобы верстка не искажалась при добавлении большого количества текста в любой блок верстки;

верстка должна соответствовать стандарту xhtml 1.0, можно использовать онлайн валидатор для проверки кода (например, http://validator.w3.org).

1. Вопросы для самоконтроля и задачи
2. Что представляет из себя блочная верстка?
3. Каковы достоинства и недостатки блочной верстки?
4. Какие особенности характерны для блочной верстки?

легко выравнивать по высоте;

сложно выравнивать по высоте;

компактный код;

громоздкий код;

возможно отображение по мере загрузки документа.

1. Какие особенности характерны для блочной верстки?

строгий порядок следования элементов;

элементы на странице могут располагаться не в порядке их следования в документе;

ширина контейнера зависит от содержимого;

контейнер занимает всю доступную ширину;

высота колонок разная, в зависимости от содержания.

1. Два блока (A и B) соприкасаются внешними отступами (margin), при этом внешний отступ A равен 30, а B равен -40, в этом случае:

отступ между ними равен -10;

отступ между ними равен 70;

отступ между ними равен 0;

отступ между ними равен -70.

1. Вычислите ширину блока со следующим стилем:

width: 200px;

margin: 10px;

border: 5px solid black;

padding: 10px;

# Методические рекомендации по выполнению лабораторной работы № 4. Построение интерфейса с использованием сценариев JavaScript

1. Цель работы

Целью выполнения работы является выработка у обучающихся умений по работе с DOM с использованием JavaScript.

1. Условия и порядок проведения лабораторной работы

Для проведения занятия необходимо знание лекционного материала по теме лекции № 8. На занятии необходимо реализовать проверку правильности заполнения формы заказа на языке JavaScript. За основу в данной работе взята верстка, реализованная в рамках лабораторной работы 1. Занятие проводится в форме тренинга, в ходе которого сначала осуществляется перекрестный опрос по теме лекции № 8, затем обучающиеся выполняют задание индивидуально, последние 15 минут занятия они попарно проверяют код друг друга, выявляя ошибки и недочеты, формируя навыки по анализу корректности кода.

1. Необходимые теоретические сведения

### Введение в JavaScript

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страничка – тут же выполняются. Программы на JavaScript – обычный текст. Они не требуют какой-то специальной подготовки.

Основатель Netscape Communications и бывший участник команды Mosaic Марк Андриссен считал, что WWW должен стать более динамичным. Анимации, взаимодействие с пользователями и другие виды интерактивности должны стать неотъемлемой частью интернета будущего. WWW нуждался в лёгком скриптовом языке, способном работать с DOM, который в те дни не был стандартизирован. Существовало одно «но», являвшееся на тот момент серьёзным вызовом: этот язык не должен был предназначаться для крупных разработчиков и прочих людей, имевших отношение к инженерной стороне вопроса. Java в те дни уже активно развивалась и твёрдо заняла эту нишу. Таким образом, новый скриптовый язык должен был предназначаться для совершенно иной аудитории − дизайнеров. Очевидно, что WWW был статичным, а HTML был достаточно молод и прост в освоении даже для тех, кто не имел ничего общего с программированием. Следовательно, всё, что должно было стать частью браузера и сделать WWW более динамичным, должно быть максимально понятным для далёких от программирования людей.

### Возможности JavaScript

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.

JavaScript – быстрый и мощный язык, но браузер накладывает на его исполнение некоторые ограничения. Это сделано для безопасности пользователей, чтобы злоумышленник не мог с помощью JavaScript получить личные данные или как-то навредить компьютеру пользователя. Этих ограничений нет там, где JavaScript используется вне браузера, например, на сервере. Кроме того, современные браузеры предоставляют свои механизмы по установке плагинов и расширений, которые обладают расширенными возможностями, но требуют специальных действий по установке от пользователя

Большинство возможностей JavaScript в браузере ограничено текущим окном и страницей:

* JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы. Он не имеет прямого доступа к операционной системе.
* Современные браузеры могут работать с файлами, но эта возможность ограничена специально выделенной директорией. Возможности по доступу к устройствам также прорабатываются в современных в современных стандартах и частично доступны в некоторых браузерах.
* JavaScript, работающий в одной вкладке, не может общаться с другими вкладками и окнами, за исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника (одинаковый домен, порт, протокол).
* Есть способы это обойти, но они требуют специального кода на оба документа, которые находятся в разных вкладках или окнах. Без него, из соображений безопасности, перейти из одной вкладки в другую при помощи JavaScript нельзя.
* Из JavaScript можно легко посылать запросы на сервер, с которого пришла страница. Запрос на другой домен тоже возможен, но менее удобен, т.к. и здесь есть ограничения безопасности.

Синтаксис JavaScript устраивает не всех: одним он кажется слишком свободным, другим – наоборот, слишком ограниченным, третьи хотят добавить в язык дополнительные возможности, которых нет в стандарте. Это нормально, ведь требования и проекты у всех разные.

В последние годы появилось много языков, которые добавляют различные возможности «поверх» JavaScript, а для запуска в браузере – при помощи специальных инструментов «трансляторов» превращаются в обычный JavaScript-код. Это преобразование происходит автоматически и совершенно прозрачно, при этом неудобств в разработке и отладке практически нет.

При этом разные языки выглядят по-разному и добавляют совершенно разные вещи:

* например, язык CoffeeScript сосредоточен на большей ясности и краткости кода. Как правило, его особенно любят программисты на Ruby;
* язык Dart интересен тем, что он не только транслируется в JavaScript, как и другие языки, но и имеет свою независимую среду выполнения, которая даёт ему ряд возможностей и доступна для встраивания в приложения (вне браузера). Он разрабатывается компанией Google.

### Основы языка JavaScript

**В чем писать код?**

Notepad++ − свободный текстовый редактор с открытым исходным кодом для Windows с подсветкой синтаксиса большого количества языков программирования и разметки.

IDE (Integrated Development Environment) – «интегрированная среда разработки», означает редактор, который расширен большим количеством «наворотов», умеет работать со вспомогательными системами, такими как багтрекер, контроль версий, и много чего ещё. Как правило, IDE загружает весь проект целиком, поэтому может предоставлять автодополнение по функциям всего проекта, удобную навигацию по его файлам и т.п.

Браузеры − клавиша F12 под Windows или комбинация клавиш Cmd+Opt+J под Mac откроет инструменты разработчика и вкладку Console, в которой можно править код. Можно набирать и запускать команды, область для команд на рисунке находится справа, запуск команд осуществляется нажатием Ctrl+Enter (для Mac – Cmd+Enter).

Пример сценария (фраза «Hello, World!», отображаемая с помощью JavaScript):

<html>

<head><title>Hello World</title></head>

<body>

<scripttype=”text/javascript”> document.write(“Hello, World!”) </script>

<noscript> Ваш браузер не поддерживает JavaScript, или его поддержка отключена </noscript>

</body>

</html

В дополнение к внесению кода JavaScript непосредственно в HTML-документы вы можете включать в них файлы с кодом JavaScript или со своего сайта, или из любого места в Интернете. Для этого используется следующий синтаксис:

<script type=»text/javascript» src=»script.js»></script>

А для извлечения файла из Интернета применяется этот синтаксис:

<script type=”text/javascript” src=”http://someserver.com/script.js”> </script>

В самих файлах сценариев не должно быть никаких тегов <script> или </script>, поскольку они там не нужны: браузеру и так известно, что будет загружаться файл JavaScript. Применение этих тегов в файлах JavaScript приводит к возникновению ошибки. Включение файлов сценариев − предпочтительный способ использования на сайте файлов JavaScript, принадлежащих сторонним производителям

В отличие от PHP точка с запятой коду JavaScript, имеющему в строке одну инструкцию, не требуется. Поэтому следующая строка вполне имеет право на существование:

x += 10

Но при необходимости иметь в строке более одной инструкции их нужно разделить точками с запятыми:

x += 10; y -= 5; z = 0

Последнюю точку с запятой можно опустить, поскольку последняя инструкция будет завершена символом новой строки. В правилах использования точки с запятой есть исключения. Если пишутся URL-закладки, JavaScript или инструкция завершается ссылкой на переменную или функцию и первый символ расположенной ниже строки является левой круглой или фигурной скобкой, нужно обязательно поставить точку с запятой, иначе сценарий JavaScript даст сбой. Поэтому при любых сомнениях нужно ставить точку с запятой.

В силу общих наследственных черт, приобретенных у языка программирования C, языки PHP и JavaScript имеют много общего и между собой, в частности в приемах комментирования кода. В первую очередь это касается однострочных комментариев:

// Это комментарий

В этой технологии используется пара прямых слешей (//), информирующая JavaScript о том, что все остальные символы должны быть проигнорированы. А затем наступает черед многострочных комментариев:

/\* Это раздел многострочного комментария, не подвергаемого интерпретации \*/

Многострочный комментарий начинается с последовательности символов /\* и заканчивается символами \*/. Нужно лишь запомнить, что использовать вложенные многострочные комментарии не допускается, поэтому важно убедиться в отсутствии большого закомментированного участка кода, в котором уже имеются многострочные комментарии.

В JavaScript нет никаких идентификационных символов переменных, таких, как знак доллара ($) в PHP. Вместо этого в отношении имен переменных действуют следующие правила:

* имена переменных могут включать только буквы a–z, A–Z, цифры 0–9, символ $ и символ подчеркивания (\_);
* никакие другие символы, включая пробелы или знаки пунктуации, использовать в именах переменных не допускается;
* первым в имени переменной может быть символ из диапазонов a–z, A–Z, символ $ или символ подчеркивания \_ (и никаких цифр);
* имена чувствительны к регистру (имена Count, count и COUNT принадлежат трем разным переменным);
* ограничений на длину имени переменной не существует.

JavaScript допускает использование символа $, и он может быть первым символом в имени переменной или функции. Такая возможность означает, что благодаря этому можно переносить на JavaScript большие объемы кода PHP значительно быстрее, xотя не рекомендуется его применять в данном качестве.

### Типы данных

В JavaScript существует несколько основных типов данных:

**Number** − единый тип число используется как для целых, так и для дробных чисел. Существуют специальные числовые значения Infinity (бесконечность) и NaN (ошибка вычислений). Например, бесконечность Infinity получается при делении на ноль.

**String** − в некоторых языках программирования есть специальный тип данных для одного символа. Например, в языке С это char. В JavaScript есть только тип «строка» string (в JavaScript одинарные и двойные кавычки равноправны).

**Boolean** − у него всего два значения: true (истина) и false (ложь). Как правило, такой тип используется для хранения значения типа да/нет.

**Null** − значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения 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».

**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. Можно присвоить undefined и в явном виде, хотя это делается редко.

**Object** − используется для коллекций данных и для объявления более сложных сущностей.

Объявляются объекты при помощи фигурных скобок {…}.

### Функции, операторы, логические переменные

В JavaScript нет явного преобразования типов, осуществляемого с помощью операторов (int) или (float). Когда нужно, чтобы значение имело определенный тип данных, используется одна из встроенных функций JavaScript, показанных в таблице 4.1.

Таблица 4.1.

Типы данных и функции JavaScript

|  |  |
| --- | --- |
| Преобразование в тип данных | Используемая функция |
| Int, Integer | parseInt() |
| Bool, Boolean | Boolean() |
| Float, Double, Real | parseFloat() |
| String | String() |
| Array | split() |

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков.

Получение длины символами происходит следующим образом:

var str = “My\n”; // 3 символа.

Alert( str.length ); // 3

Получение символа на позиции, при первой позиции равной 0:

var str = “jQuery”;

alert( str.charAt(0) ); // “j”

Или же…

var str = «Я – современный браузер!»;

alert( str[0] ); // «Я»

str[0]

Содержимое строки в JavaScript нельзя изменять. Нельзя взять символ посередине и заменить его. Как только строка создана – она такая навсегда. Можно лишь создать целиком новую строку и присвоить в переменную вместо старой, например:

var str = “строка”;

str = str[3] + str[4] + str[5];

alert( str ); // ока

str.indexOf

Для поиска подстроки есть метод **str.indexOf** (Подстрока(Начальная позиия)). Он возвращает позицию, на которой находится подстрока или -1, если ничего не найдено. Например:

var str = «Widget with id»;

alert( str.indexOf(«Widget») ); // 0, т.к. «Widget» найден прямо в начале str

alert( str.indexOf(«id») ); // 1, т.к. «id» найден, начиная с позиции 1

alert( str.indexOf(«widget») ); // -1, не найдено, так как поиск учитывает регистр

Метод substring(start, end) возвращает подстроку с позиции start до, но не включая end:

var str = «stringify»;

alert(str.substring(0,1)); // «s», символы с позиции 0 по 1 не включая 1.

Операторы в JavaScript могут использоваться в математических операциях, для внесения изменений в строки, а также в операциях сравнения и логических операциях (И, ИЛИ и т. Д.). Математические операторы JavaScript во многом похожи на обычные арифметические операторы (таблица 4.2).

Таблица 4.2.

Арифметические операторы

|  |  |
| --- | --- |
| Оператор | Описание |
| + | Сложение |
| - | Вычитание |
| \* | Умножение |
| / | Деление |
| % | Деление по модулю (остаток от деления) |
| ++ | Инкремент |
| –– | Декремент |

Операторы присваивания используются для присваивания значений переменным. Их линейка начинается простым знаком равенства (=) и продолжается сочетаниями +=, -= и т. Д. (таблица 4.3).

Таблица 4.3.

Операторы присваивания

| Оператор | Пример |
| --- | --- |
| = | a=9 |
| += | a=c+2 |
| += | a=c+’’string’’ |
| -= | a=c-5 |
| \*= | a=c\*3 |
| /= | a=c/3 |
| %= | a=c%7 |

Операторы сравнения обычно используются с такими конструкциями, как инструкция if, в которой требуется сравнивать два элемента. Например, может потребоваться узнать, достигло ли значение переменной, подвергаемой автоприращению, определенной величины или меньше ли значение другой переменной установленной величины и т. Д. (таблица 4.4).

Таблица 4.4.

Операторы сравнения

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Пример |
| == | Равно | a== |
| != | Не равно | a!= |
| > | Больше | a> |
| < | Меньше | a< |
| >= | Больше или равно | a>== |
| <= | Меньше или равно | a🡸 |
| === | Равно(и того же типа) | a=== |
| !== | Не равно(и того же типа) | a!== |

В Javascrypt также активно применяются логические переменные (таблица 4.5):

Таблица 4.5.

Логические переменные

| Оператор | Описание | Пример |
| --- | --- | --- |
| && | И | j == 1 && k == 2 |
| || | Или | j < 100 || j > 0 |
| ! | Не | ! (j == k) |

Функция **alert** выводит на экран окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмёт «ОК»

Окно сообщения, которое выводится, является модальным окном. Слово «модальное» означает, что посетитель не может взаимодействовать со страницей, нажимать другие кнопки и т.п., пока не разберётся с окном. В данном случае – пока не нажмёт на «OK».

Alert( «Привет» );

Функция **prompt** выводит модальное окно с заголовком title, полем для ввода текста, заполненным строкой по умолчанию default и кнопками OK/CANCEL. Пользователь должен либо что-то ввести и нажать OK, либо отменить ввод кликом на CANCEL или нажатием Esc на клавиатуре.

Result = prompt(title, default);

Функция **confirm** выводит окно с вопросом question с двумя кнопками: OK и CANCEL. Результатом будет true при нажатии OK и false – при CANCEL(Esc).

Var isAdmin = confirm(“Вы – администратор?”)

alert( isAdmin );

Код внутри инструкции **if** выполняется только в том случае, если заданное выражение вычисляется как **true**. Многострочные инструкции if заключаются в фигурные скобки, но, для однострочных инструкций скобки можно опустить. Поэтому допустимы все следующие инструкции:

if (a > 100) { b=2 document.write(«a больше 100») }

if (b == 10) document.write(“b равно 10”)

Если условие не было соблюдено, то с помощью инструкции else может быть выполнен альтернативный блок кода:

if (a > 100) { document.write(“a больше 100”) }

else { document.write(«a меньше или равно 100»)

Инструкция **switch** применяется в том случае, когда одиночная переменная или результат вычисления выражения может иметь несколько значений, для каждого из которых нужно применить свою функцию.

1. Ход выполнения работы
2. В форме, полученной в рамках лабораторной работы № 1, внести следующие изменения (см. рисунок 4.1):

добавить названия полей с помощью атрибута name: surname (фамилия), name (имя), age (возраст), gender (пол), comment (комментарий), agree (чекбокс «Берите мои данные»);

чекбокс agree сделать активным и не установленным по умолчанию;

кнопку отправки формы сделать типа «button».

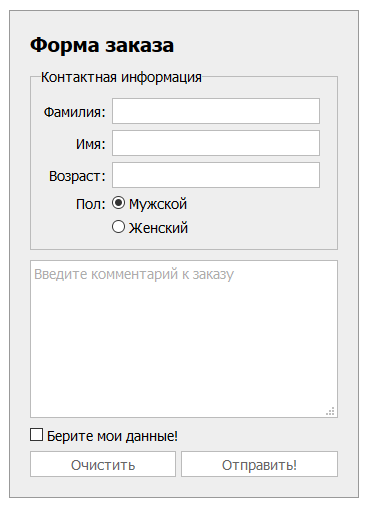


Рис. 4.1 – Форма заказа.

2. Подсветка полей будет производится зеленым цветом, если поле заполнено верно, и красным – если есть ошибки. В css-файле введем классы good и bad для посветки тегов input и textarea следующими стилями:

good: цвет фона #D0FDD4, цвет рамки #209F3A;

bad: цвет фона #FDD0D0, цвет рамки #E74343.

3. Ошибки, выявленные при заполнении формы, будем помещать в абзацы текста с классом error, расположенные непосредственно перед кнопкой «Очистить». Создадим стиль для таких абзацев: цвет текста: #FF2D31, нижний внутренний отступ 5 точек.

4. В случае, когда форма прошла проверку, сделаем кнопку «Отправить» активной. Для этого создадим для нее класс active со следующими стилями: цвет фона #209F3A, цвет текста #fff, курсор типа pointer.

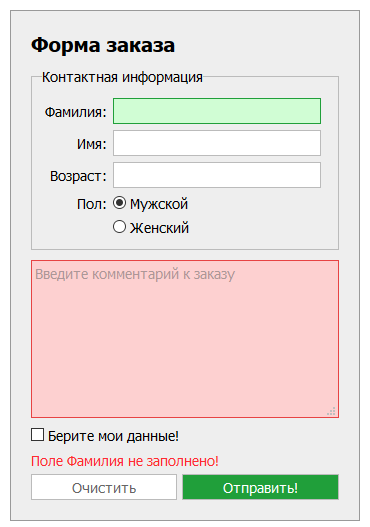


Рис. 4.2 – Проверка подсветки полей.

5. Проверим созданные стили, применив к полю name класс good, к полю comment класс bad, к кнопке «Отправить» класс active. Так же создадим одну ошибку с текстом «Поле Фамилия не заполнено». Результаты проверки представлены на рисунке 2.

6. Отменим подсветку полей, выполненную на шаге 5. Подключим js-файл непосредственно под формой:

<script src=”validation.js”></script>

В подключенном файле создадим каркас функций для проверки формы.

1. Создадим заготовку для функции checkFields(), которая будет проверять все поля на наличие ошибок, формировать массив ошибок errors:

var checkFields = function(){

var errors = [];

…

//обновляем сообщения об ошибках

console.log(errors);

updateErrors(errors);

return errors;

}

1. Создадим функцию updateErrors(\_\_errors), которая будет формировать блок с сообщениями об ошибках перед кнопкой «Очистить»:

var updateErrors = function(\_\_errors) {

//чистим сообщения с ошибками

var errorElements = document.body.querySelectorAll(“.error”);

while (errorElements.length > 0) {

//удаляем узлы с ошибками

…

}

//добавляем актуальные ошибки

\_\_errors.forEach(function(error){

//создаем узлы с ошибками

…

});

}

1. Создадим функцию валидации формы validateForm(), которая будет делать кнопку отправки активной, если форма заполнена правильно:

var validateForm = function(){

if(checkFields().length === 0) {

//добавить на кнопку класс active

…

}

else {

//убрать с кнопки класс active

…

}

}

1. Создадим функцию отправки формы sendForm():

var sendForm = function(){

if(checkFields().length === 0)

alert(«Форма отправлена!»);

}

1. Повесим на кнопку «Отправить» обработчик события onclick:

<input type=”button” value=”Отправить!” onclick=”sendForm();”/>

7. Протестируем созданные функции.

7.1. Начнем с функции updateErrors(\_\_errors). Создадим несколько сообщений об ошибках. В js-файле в конце вызовем функцию:

updateErrors([«Новая ошибка»]);

После вызова функции предварительно созданные сообщения об ошибках должны исчезнуть и появится ошибка «Новая ошибка». Текст ошибки также будет продублирован в консоль (см. рисунок 4.3).

7.2. Функция checkFields() пока что является заготовкой и всегда возвращает пустой массив ошибок. То есть форма должна быть успешно провалидирована с помощью validateForm(). Проверим это командой в js-файле:

validateForm();

После выполнения этой команды кнопка «Отправить» стала зеленой и при клике на нее произошла отправка формы.

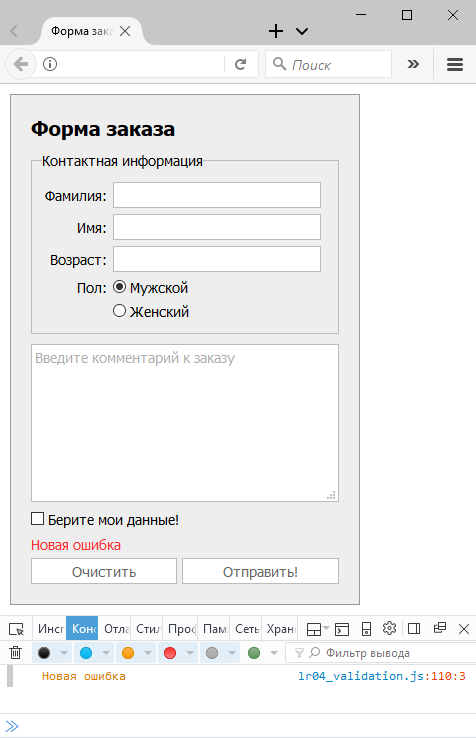


Рис.4.3 – Проверка функции updateErrors().

8. В функцию checkFields необходимо добавить функциональность для проверки полей формы.

8.1. Проверка текстовых полей.

Нетрудно заметить, что поля Фамилия, Имя и Комментарий являются текстовыми. Осуществим проверку того, что они заполнены и содержат хотя бы три символа. Поскольку проверка будет одинаковой для трех полей, целесообразно для данного вида проверки создать отдельную функцию validateStringField:

var validateStringField = function(ieldname, humanName) {

var errors = [];

var field = document.body.querySelector(“[name=’” + ieldname + “’]”);

if(field.value == ‘’ || field.value.length < 3){

//установим класс для подсветки

field.classList.add(“bad”);

field.classList.remove(“good”);

//добавим текст ошибки в массив

errors.push(field.value == ''

? «Поле « + humanName + « не заполнено»

: «Поле « + humanName + « содержит мало символов»);

}

else {

//установим класс для подсветки

field.classList.add(“good”);

field.classList.remove(“bad”);

}

return errors;

}

В функции checkFields разместим следующие строки:

//Проверим поля Фамилия, Имя и Комментарий

errors = errors.concat(validateStringField(“surname”, “Фамилия”));

errors = errors.concat(validateStringField(“name”, “Имя”));

errors = errors.concat(validateStringField(“comment”, “Комментарий”));

После этого, в форме успешно валидируются строковые поля. Для проверки заполните поля и нажмите кнопку «Отправить».

8.2. Реализуйте блок проверки возраста в функции checkFields() по аналогии с содержимым функции validateStringField(). Возраст должен быть задан в диапазоне между 18 и 45. Для получения числового значения поля используйте функцию parseInt:

var age = parseInt(ageField.value);

if (age > 18 && age < 45) …

8.3. Реализуйте блок проверки чекбокса «Берите мои данные». Для проверки, установлен чекбокс или нет, можно воспользоваться свойством «checked»:

if (agreeField.checked) …

1. Теперь в форме проверяются все поля. Однако, нам бы хотелось бы, чтобы форма валидировалась уже после нажатия на чекбокс. Для этого добавим в конец js-файла обработчик события:

document.body.querySelector(“[name=’agree’]”).addEventListener(“click”, validateForm);

1. Нажимать каждый раз на кнопку отправки для проверки полей формы неудобно, поэтому сделаем автоматическую ревалидацию формы после каждого нажатия на клавишу клавиатуры, после того, как форма хотя бы один раз была провалидирована (нажатием на чекбокс или кнопку «Отправить»). Для этого создадим функцию ревалидации:

var revalidateForm = function(){

var surnameClasses = document.body.

querySelector(“input[name=’surname’]”).classList;

if(surnameClasses.contains(“good”)

|| surnameClasses.contains(“bad”)

|| document.body

.querySelector(“input[name=’agree’]”).checked){

validateForm();

}

}

Добавим обработчики событий для отжатия клавиши клавиатуры:

document.body.querySelector(“[name=’surname’]”)

.addEventListener(“keyup”, revalidateForm);

document.body.querySelector(“[name=’name’]”)

.addEventListener(“keyup”, revalidateForm);

document.body.querySelector(“[name=’age’]”)

.addEventListener(“keyup”, revalidateForm);

document.body.querySelector(“[name=’comment’]”)

.addEventListener(“keyup”, revalidateForm);

1. Проверьте работоспособность формы. После первой валидации, проверка полей должна происходить после каждого нажатия на клавишу (см. рисунок 4.4).

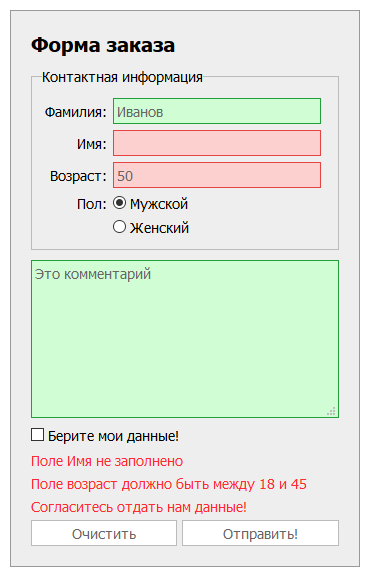


Рис. 4.4 – Валидация формы работает.

1. Вопросы для самоконтроля и задачи
2. Какие возможности JavaScript вы знаете?
3. Что такое AJAX?
4. Как выглядит комментарий в JavaScript?
5. В чем можно писать код JavaScript?
6. Какие типы данных JavaScript вы знаете?
7. Что делает Функция alert?

# Методические рекомендации по выполнению лабораторной работы № 5. Использование библиотеки jQuery для построения web-интерфейсов

1. Цель работы

Целью выполнения работы является освоение обучающимися основных возможностей библиотеки jQuery.

1. Условия и порядок проведения лабораторной работы

Для проведения занятия необходимо знание лекционного материала по теме лекции № 9. На занятии необходимо оптимизировать код проверки правильности заполнения формы заказа с помощью библиотеки jQuery. За основу в данной работе взята верстка, реализованная в рамках лабораторной работы 4 (шаги 1-6). Занятие проводится в форме тренинга, в ходе которого сначала осуществляется перекрестный опрос по теме лекции № 9, затем обучающиеся выполняют задание индивидуально, последние 15 минут занятия они попарно проверяют код друг друга, выявляя ошибки и недочеты, формируя навыки по анализу корректности кода.

1. Необходимые теоретические сведения

### Библиотека jQuery

При всей гибкости и эффективности JavaScript, а также при всем изобилии имеющихся в этом языке встроенных функций все же сохраняется потребность в дополнительных уровнях кода, позволяющих упростить, к примеру, получение эффектов анимации, обработку событий и применение технологии Ajax, то есть сделать то, чего нельзя достичь применением обычных средств JavaScript или CSS.

Более того, с годами, вследствие различных браузерных войн, то появлялись, то исчезали досаждающие и раздражающие несовместимости браузеров, ярко проявлявшиеся временами на различных платформах и в программах. В результате этого гарантировать одинаковый внешний вид web-страниц во всех устройствах порой можно было, только применяя требовавший утомительной разработки код JavaScript, учитывающий все расхождения всей линейки браузеров и их версий, выпущенных за последние годы. Одним словом, сложилась сложная ситуация.

Еще в 2014 году верстальщики не обходились без шаманских заклинаний при написании кода, поскольку Internet Explorer компании Microsoft во многих (но все же не во всех) областях догнал стандарты, в браузере Opera в качестве основы решено было применять WebKit (ту же самую технологию, которая использовалась компанией Google), а компания Apple ушла с рынка браузеров для PC-совместимых компьютеров.

До сих пор унаследованная несовместимость еще осталась, тем более, что некоторые до сих пор пользуются старыми браузерами. Плюс к тому, если есть желание создать что-либо более впечатляющее, чем обычные спецэффекты, вам все еще придется писать весьма существенный объем кода на JavaScript.

Для заполнения пробелов было разработано множество библиотек функций, предназначенных для сведения к минимуму различий между браузерами, и многие из этих библиотек позволяют, кроме этого, легко привязываться к объектной модели документа − DOM (Document Object Model), облегчают работу с Ajax, событиями и анимацией. В их число входят такие наиболее предпочтительные библиотеки, как AngularJS, jQuery, MooTools, Prototype, script.aculo.us и YUI (существует и множество других библиотек).

Почему именно библиотека jQuery? По сведениям http://w3techs.com, библиотека jQuery установлена более чем на 60 % сайтов и востребована шире, чем все ее конкуренты, вместе взятые (насколько можно судить по диаграммам источника). Кстати, если есть желание посмотреть на столбиковую диаграмму, отображающую процентное соотношение востребованности различных библиотек на любой текущий момент времени, введите в поисковой строке сайта similartech.com слово javascript.

Используя jQuery, вы получаете не только кросс-браузерную совместимость весьма высокого уровня (включая совместимость с Internet Explorer), но и быстрый и легкий доступ к операциям с HTML и DOM, возможность использования специальных функций для непосредственной работы с CSS, управления событиями, мощные средства для создания профессиональных эффектов и анимации, а также функции для управления обменом данными с сервером по технологии Ajax. Кроме того, jQuery является основой для широкого круга дополнительных модулей и других вспомогательных программ.

Есть два способа включения jQuery в ваши web-страницы. Можно перейти на сайт jQuery, выбрать нужную версию, загрузить ее на свой сайт и пользоваться всеми ее возможностями. Или же можно воспользоваться находящейся в свободном доступе сетью доставки контента − Content Delivery Network (CDN) и просто указать ссылку на нужную вам версию.

### Выбор подходящей версии jQuery

Перед тем как решить, стоит ли загружать jQuery и использовать ее функции непосредственно или же воспользоваться CDN, нужно выбрать версию jQuery. В большинстве случаев выбор очевиден, поскольку вы просто отдадите предпочтение наиболее свежей версии. Но если есть намерение использовать конкретные браузеры или же вы поддерживаете устаревший сайт, работа которого зависит от определенной версии jQuery, то последний выпуск этой библиотеки может вам не подойти.

В отличие от большинства других программных средств, для использования которых вы просто загружаете и устанавливаете самую новую из доступных версий, jQuery со временем совершенствовалась с учетом изменения движущих сил на рынке различных версий браузеров с их свойствами и недочетами.

В то же время в jQuery вносили различные усовершенствования, которые могли изменить работу ее новых версий на тех сайтах, которые были специально адаптированы под конкретную версию на момент ее выпуска (и под все сопутствующие ей особенности).

Разумеется, каждая более новая версия является улучшением предыдущей, и вероятность того, что вносимые усовершенствования коснутся всех основ, постоянно возрастает. Но пока вы полностью не протестируете новую версию и не убедитесь в том, что операции, играющие важную роль для вашего сайта, выполняются точно так же, лучше все же продолжать использовать прежнюю версию.

Также нужно решить, какую версию jQuery вам хотелось бы использовать: минимальную по размеру (сжатую), чтобы свести к минимуму требуемую полосу пропускания сети и сократить время загрузки, или несжатую (возможно, по причине того, что вам хочется вносить в нее самостоятельные правки, на что вы имеете полное право). Как правило, наиболее удачным выбором считается минимальная по размеру версия, но большинство web-серверов поддерживают архиватор gzip, позволяющий выполнять сжатие и распаковку на лету, поэтому данный вопрос теряет свою актуальность (хотя нужно учесть, что из минимизированной версии, кроме всего прочего, удалены все комментарии).

На сайте jquery.com/download каждая выпущенная версия jQuery фигурирует в списке как в сжатой, так и в несжатой форме. Вам остается всего лишь выбрать нужную версию, щелкнуть правой кнопкой мыши на соответствующей ссылке и сохранить версию на своем жестком диске. Оттуда ее можно будет выгрузить на ваш web-сервер, а затем включить в web-страницу с помощью <script>-тегов примерно таким образом (для минимизированной версии выпуска 1.11.1):

<script src='http://myserver.com/jquery-1.11.1.min.js'></script>

Библиотека jQuery поддерживается несколькими сетями доставки контента (CDN). Если вы пользуетесь одной из них, то можете избавить себя от хлопот, связанных с загрузкой новых версий, и выкладывать их на сервер, просто указав прямые ссылки на URL-адреса, поддерживаемые этими сетями.

Ко всему прочему, эти сети предоставляют свои услуги совершенно бесплатно и обычно используют каналы с высокой пропускной способностью, которые, возможно, являются самыми скоростными на свете. Кроме того, CDN-сети обычно хранят свой контент в нескольких различных географических пунктах и предоставляют файл с ближайшего к вам сервера, гарантируя тем самым наиболее быструю доставку.

В общем, если вам не нужно вносить изменения в исходный код jQuery (для чего требуется его размещение на ваших собственных web-серверах) и у ваших пользователей гарантированно имеется живое интернет-соединение, то, скорее всего, наилучшим вариантом будет использование CDN-сетей. Тем более что пользоваться ими довольно просто. остаточно знать имя нужного файла и используемого для его загрузки корневого каталога CDN. Например, все текущие и предыдущие версии можно получить через CDN-сеть, которая используется библиотекой jQuery, с помощью следующего кода:

<script src='http://code.jquery.com/jquery-1.11.1.min.js'></script>

Основной каталог доступен по адресу http://code.jquery.com/, и за ним нужно просто дописать имя нужного для включения файла (в данном случае это jquery-1.11.1.min.js).

Библиотеку jQuery предоставляют в своих сетях как Microsoft, так и Google, поэтому для ее включения можно воспользоваться любым из следующих двух вариантов:

<script src='http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.1.min.js'></script>

<script src='http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js'> </script>

В случае использования Microsoft CDN (aspnetcdn.com) в URL-адресе сначала нужно указать основной каталог ajax.aspnetcdn.com/ajax/jQuery/, а за ним — имя требуемого файла.

Но для Google нужно разбить имя файла (например, jquery-1.11.1.min.js) на имя каталога и имя файла таким вот образом: 1.11.1/jquery.min.js. А перед этим поА перед этим по перед этим поперед этим по этим поэтим по попоставить строку ajax.googleapis.com/ajax/libs/jquery/.

Еще одним преимуществом CDN-сетей является то, что вы всегда можете выбрать самую последнюю версию jQuery, следовательно, сберечь web-страницу и напрочь забыть о необходимости ее обновления до самого последнего выпуска. Чтобы включить последнюю версию (выпуска из серии 1.x) из CDN-сетей jQuery или Google, нужно воспользоваться одной из двух форм <script>-тега:

<script src='http://code.jquery.com/jquery-latest.min.js'></script>

<script src='http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js'></script>

Но при этом нужно проявить особую осторожность, поскольку вполне возможно, что какие-нибудь компоненты вашей web-страницы не смогут работать с будущим обновлением, и вам следует быть готовыми к тому, что какая-либо из web-страниц начнет вести себя не так, как нужно.

### Синтаксис jQuery

Больше всего людей, ранее незнакомых с jQuery, удивляет символ $, который действует как фабричный метод jQuery. Он был выбран из расчета допустимости в JavaScript, краткости и отличия от имен обычной переменной, объекта или функции (метода).

Им обозначается вызов функции jQuery (что также при желании можно сделать). Замысел его использования заключается в сохранении краткости и приятного внешнего вида кода, а также избавлении от излишнего набора текста при каждом обращении к jQuery. Кроме того, при виде этого символа другие, ранее незнакомые с вашим кодом разработчики сразу же понимают, что в коде используется jQuery (или подобная ей библиотека).

В наипростейшем виде обращение к jQuery осуществляется набором символа $, за которым следуют заключенный в скобки селектор, точка и метод, применяемый к выбранному элементу (или элементам).

Например, для изменения семейства шрифтов всех абзацев на моноширинное можно воспользоваться следующей инструкцией:

$('p').css('font-family', 'monospace')

А для добавления границы к элементу <code> можно применить такую инструкцию:

$('code').css('border', '1px solid #aaa')

Взглянем на часть полноценного примера 5.1, где фрагменты, относящиеся к использованию jQuery, выделены полужирным шрифтом.

Пример 5.1.

Простой пример применения jQuery

<!DOCTYPE html>

<head>

<title>Первый пример jQuery</title>

<script src='jquery-1.11.1.min.js'></script>

</head>

<body>

В jQuery в качестве имен функции используются либо <code>$()</code>,

либо <code>jQuery()</code>.

<script>

$('code').css('border', '1px solid #aaa')

</script>

</body>

</html>

Если наряду с jQuery используются и другие библиотеки, может оказаться, что в них определена собственная $-функция, для решения данной проблемы можно в отношении этого символа вызвать метод noConflict, который освобождает этот символ от управляющей функции, позволяя другим библиотекам воспользоваться им:

$.noConflict()

После этого для доступа к jQuery следует вызывать функцию jQuery. Или же использование символа $ можно подменить именем объекта по вашему выбору:

jq = $.noConflict()

Теперь в тех местах, где прежде применялся символ $, можно воспользоваться ключевым словом jq.

После того как вы увидели, насколько просто можно включить jQuery в web-страницу и обратиться к функциям этой библиотеки, перейдем к рассмотрению используемых в ней селекторов, которые (я уверен, что вы будете рады это узнать) работают точно так же, как CSS. По сути, их применение является основой работы большинства функций jQuery.

Вам остается лишь подумать о том, как бы вы оформили стиль одного или нескольких элементов с применением CSS, а затем можете использовать тот же самый селектор (или селекторы) для применения операций jQuery к этим выбранным элементам. Это означает, что вы можете воспользоваться селекторами элементов, селекторами идентификаторов, селекторами классов и любыми их сочетаниями.

### **Работа с селекторами в jQuery**

Чтобы объяснить применение селекторов в jQuery, сначала посмотрим на один из более фундаментальных методов jQuery, css, с помощью которого можно динамически менять любое свойство CSS. Этому методу передаются два аргумента: имя свойства, к которому осуществляется обращение, и значение, которое к этому свойству применяется:

css('font-family', 'Arial')

Как будет показано в следующих разделах, сам по себе этот метод применять невозможно, поскольку его нужно использовать в селекторе jQuery, который выберет один или несколько элементов, чьи свойства должны быть изменены этим методом. В следующем примере содержимому всех <p>-элементов предписывается отображение с полным выравниванием по ширине:

$('p').css('text-align', 'justify')

Метод css можно также использовать для возвращения (а не для установки) вычисленного значения, для чего ему предоставляется только имя свойства (а второй аргумент опускается). В этом случае возвращается значение первого же соответствующего селектору элемента. Например, выполнение следующего кода приведет к возвращению цвета текста того элемента, чей идентификатор (ID) имеет значение elem, и это значение будет в том же формате, в котором цвет задается при применении метода rgb:

color = $('#elem').css('color')

Следует помнить, что возвращаемое значение является вычисленным. Иными словами, jQuery будет вычислять и возвращать значение, используемое браузером на момент вызова метода, а не то исходное значение, которое могло быть присвоено свойству посредством таблицы стилей или любым другим способом.

Следовательно, если текст, к примеру, показан синим цветом, значением, присвоенным переменной color в предыдущей инструкции, будет rgb (0, 0, 255), даже если цвет изначально был установлен с использованием имени цвета blue или с использованием строк шестнадцатеричных чисел #00f или #0000ff. Но это вычисленное значение всегда будет в форме, которая может быть снова назначена элементу (или любому другому элементу) при использовании в качестве второго аргумента метода css.

**Селектор элемента.** Для выбора элемента, обрабатываемого с помощью jQuery, нужно просто указать его имя внутри круглых скобок, следующих за символом $ (или за именем функции jQuery). Например, если нужно изменить цвет фона всех элементов <blockquote>, можно воспользоваться следующей инструкцией:

$('blockquote').css('background', 'lime')

**Селектор идентификатора.** Ссылаться на элементы можно также по их идентификаторам (ID), если перед именем идентификатора поместить символ #. Следовательно, чтобы, к примеру, добавить границу к элементу с идентификатором advert, можно воспользоваться такой инструкцией:

$('#advert').css('border', '3px dashed red')

**Селектор класса.** Можно также воздействовать на группу элементов в соответствии с используемым ею классом. Например, для подчеркивания всех элементов, применяющих класс new, можно воспользоваться следующей инструкцией:

$('.new').css('text-decoration', 'underline')

**Сочетание селекторов.** Как и при использовании CSS, селекторы можно сочетать друг с другом, составляя единый jQuery-выбор, для чего, как в следующем примере, применяются запятые:

$('blockquote, #advert, .new').css('font-weight', 'bold')

В примере 5.2 все типы селекторов собраны вместе, а инструкции jQuery выделены полужирным шрифтом.

Пример 5.2.

Использование jQuery с различными селекторами

<!DOCTYPE html> <html>

<head>

<title>Второй пример jQuery</title>

<script src='jquery-1.11.1.min.js'></script>

</head>

<body>

<blockquote>

При всей гибкости и эффективности JavaScript, а также при всем изобилии имеющихся в этом языке встроенных функций все же сохраняется потребность в дополнительных уровнях кода, позволяющих упростить, к примеру, получение эффектов анимации, обработку событий и применение технологии Ajax, то есть сделать то, чего нельзя достичь применением обычных средств JavaScript или CSS:

</blockquote>

<div id='advert'>Это реклама</div>

<p>Это мой <span class='new'>новый</span> сайт</p>

<script>

$('blockquote').css('background', 'lime')

$('#advert').css('border', '3px dashed red')

$('.new').css('text-decoration', 'underline')

$('blockquote, #advert, .new').css('font-weight', 'bold')

</script>

</body>

</html>

**Обработка событий.** Если бы библиотека jQuery умела только подменять CSS-стили, толку от нее было бы маловато, и она, конечно же, способна на гораздо большее. Продолжим исследование и посмотрим, как она обрабатывает события.

Как вы, наверное, помните, большинство событий инициируется действиями пользователя: при прохождении указателя мыши над элементом, щелчке кнопкой мыши или нажатии клавиши. Но существуют и другие события, которые могут инициироваться, к примеру, по завершении загрузки документа.

Прикрепить ваш собственный код к этим событиям с помощью jQuery не составит труда, причем сделано это будет безопасным способом, не блокирующим для другого кода получение такого же доступа к этим событиям. Вот, к примеру, как можно заставить код jQuery откликнуться на щелчок на элементе:

$('#clickme').click(function(){

$('#result').html('You clicked the button!')

})

Когда будет сделан щелчок на элементе с идентификатором clickme, свойство innerHTML элемента со значением ID, равным result, будет обновлено с использованием jQuery-функции html.

Конкретизация замысла показана в примере 5.3.

Пример 5.3.

Обработка события

<!DOCTYPE html> <html>

<head>

<title>События jQuery</title>

<script src='jquery-1.11.1.min.js'></script>

</head>

<body>

<button id='clickme'>Нажми меня</button>

<p id='result'>Я - абзац</p>

<script>

$('#clickme').click(function()

{

$('#result').html('Вы щелкнули на кнопке!')

})

</script>

</body>

</html>

**Ожидание готовности документа.** Поскольку тому, что достигается средствами jQuery, мы во многом обязаны весьма тесной связи этой библиотеки с объектной моделью документа — DOM, вам, прежде чем воздействовать на какие-либо части страницы, скорее всего, придется дождаться ее загрузки. Без jQuery это может быть выполнено с помощью события onload, но есть более эффективный кросс-браузерный jQuery-метод под названием ready, который можно вызвать для включения в работу в самый ранний из возможных моментов времени, даже раньше, чем наступит событие onload. Это означает, что jQuery может начать работать на странице намного быстрее и с минимальными задержками для пользователя.

Чтобы воспользоваться этой возможностью, поместите свой код jQuery внутрь следующей структуры:

$('document').ready(function(){

// Сюда нужно поместить ваш код

})

Теперь код будет ждать готовности документа и только после этого будет вызван методом ready. На самом деле можно набрать еще меньший объем кода и воспользоваться более краткой версией, показанной в примере 5.4.

Пример 5.4.

Наименьший по объему код охватывающей функции,   
запускаемой по готовности документа.

$(function(){

// Сюда нужно поместить ваш код

})

Если выработать привычку помещения своих jQuery-инструкций в одну из этих двух структур, то не придется сталкиваться с тем типом ошибок, которые могут выдаваться при попытке слишком раннего обращения к DOM.

Единственная ситуация, при которой размещение сценариев в самом конце страницы может вызвать возражение, связана с тем, что документ выглядит как готовый к работе, но фактически он к ней еще не готов, или с тем, что все внешние таблицы стилей еще не загружены (реально определить это можно только тестированием), что вводит пользователей в заблуждение относительно возможности работы с документом до того, как к этому будет готов ваш сценарий. В таких случаях применяйте функцию ready, и все будет в порядке. Если же у вас есть сомнения, поместите свой сценарий в конец страницы и воспользуйтесь функцией ready, и тогда возьмете все самое лучшее от обоих вариантов.

1. Ход выполнения работы

1. Подключим библиотеку jQuery. Для этого в head документа поместим код:

<script src="jquery-3.2.1.min.js"></script>

2. В подключенном файле validation.js создадим каркас функций для проверки формы.

2.1 Создадим заготовку для функции checkFields(), которая будет проверять все поля на наличие ошибок, формировать массив ошибок errors:

var checkFields = function(){

var errors = [];

...

//обновляем сообщения об ошибках

console.log(errors);

updateErrors(errors);

return errors;

}

2.2 Создадим функцию updateErrors(\_\_errors), которая будет формировать блок с сообщениями об ошибках перед кнопкой "Очистить":

var updateErrors = function(\_\_errors) {

//чистим сообщения с ошибками:

$(".error").remove();

//добавляем актуальные ошибки

\_\_errors.forEach(function(error){

//создаем узлы с ошибками

$("input[type='reset']").before(…);

});

}

2.3. Создадим функцию валидации формы validateForm(), которая будет делать кнопку отправки активной, если форма заполнена правильно:

var validateForm = function(){

if(checkFields().length === 0) {

$("input[type='button']").addClass("active");

}

else {

$("input[type='button']").removeClass("active");

}

}

2.4. Создадим функцию отправки формы sendForm():

var sendForm = function(){

if(checkFields().length === 0)

alert("Форма отправлена!");

}

2.5. Повесим на кнопку "Отправить" обработчик события onclick:

<input type="button" value="Отправить!" onclick="sendForm();"/>

3. Протестируем созданные функции.

3.1. Начнем с функции updateErrors(\_\_errors). Создадим несколько сообщений об ошибках. В js-файле в конце вызовем функцию:

updateErrors(["Новая ошибка"]);

После вызова функции предварительно созданные сообщения об ошибках должны исчезнуть и появится ошибка "Новая ошибка". Текст ошибки также будет продублирован в консоль (см. рисунок 5.1).

3.2. Функция checkFields() пока что является заготовкой и всегда возвращает пустой массив ошибок. То есть форма должна быть успешно провалидирована с помощью validateForm(). Проверим это командой в js-файле:

validateForm();

После выполнения этой команды кнопка "Отправить" стала зеленой и при клике на нее произошла отправка формы.

4. В функцию checkFields необходимо добавить функциональность для проверки полей формы.

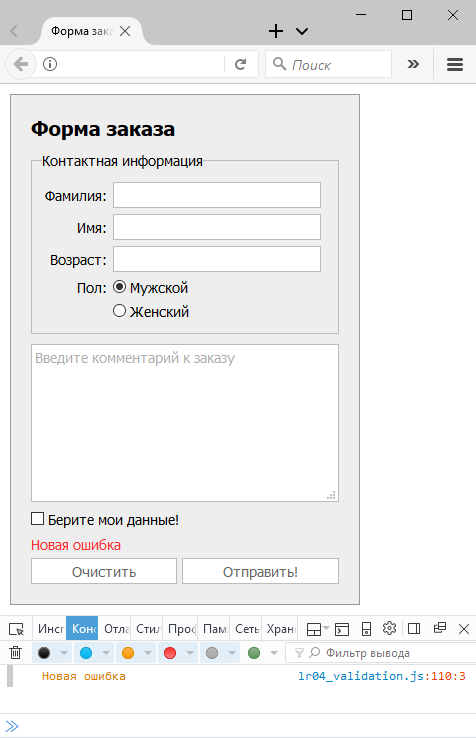


Рис. 5.1 – Проверка функции updateErrors().

4.1. Проверка текстовых полей. Нетрудно заметить, что поля Фамилия, Имя и Комментарий являются текстовыми. Осуществим проверку того, что они заполнены и содержат хотя бы три символа. Поскольку проверка будет одинаковой для трех полей, целесообразно для данного вида проверки создать отдельную функцию validateStringField:

var validateStringField = function(fieldName, translation) {

var errors = [];

var field = $("[name='" + fieldName + "']");

if(field.val() == '' || field.val().length < 3){

field.removeClass("good").addClass("bad");

errors.push(field.val() == ''

? "Поле " + translation + " не заполнено"

: "Поле " + translation + " содержит мало символов");

}

else {

field.removeClass("bad").addClass("good");

}

return errors;

}

Обратите внимание на возможность использования в jQuery "цепочек":

field.removeClass("bad").addClass("good");

Таким образом с объектом можно совершать сразу последовательность действий.

В функции checkFields добавим вызовы проверки текстовых полей. Библиотека jQuery позволяет использовать метод $.each() для перебора массивов. Воспользуемся им, для задания списка полей, которые следует проверить:

$.each({"surname": "Фамилия", "name": "Имя", "comment": "Комментарий"}, function(fieldName, translation){

errors = errors.concat(validateStringField(fieldName, translation));

});

Таким образом мы легко сможем проверить другие поля, просто добавив их в список проверки.

После этого в форме успешно валидируются строковые поля. Для проверки заполните поля и нажмите кнопку "Отправить".

4.2. Реализуйте блок проверки возраста в функции checkFields() по аналогии с содержимым функции validateStringField(). Возраст должен быть задан в диапазоне между 18 и 45. Для получения числового значения поля используйте функцию parseInt:

var ageField = $("input[name='age']");

var age = parseInt(ageField.val());

Обратите внимание, что в jQuery для взятия значения поля формы используется метод val() (а не свойство value).

4.3. Реализуйте блок проверки чекбокса "Берите мои данные". Для проверки, установлен чекбокс или нет, можно воспользоваться методом prop():

if(agreeField.prop("checked")) ...

5. Теперь в форме проверяются все поля. Однако, нам бы хотелось бы, чтобы форма валидировалась уже после нажатия на чекбокс. Для этого добавим в конец js-файла обработчик события:

$("[name='agree']").click(validateForm);

6. Нажимать каждый раз на кнопку отправки для проверки полей формы неудобно, поэтому сделаем автоматическую ревалидацию формы после каждого нажатия на клавишу клавиатуры, после того, как форма хотя бы один раз была провалидирована (нажатием на чекбокс или кнопку "Отправить"). Для этого создадим функцию ревалидации:

var revalidateForm = function(){

var surname =$("input[name='surname']");

if(surname.hasClass("good") || surname.hasClass("bad")

|| $("input[name='agree']").prop("checked")){

validateForm();

}

}

Добавим обработчики событий для отжатия клавиши клавиатуры:

$.each(["surname", "name", "age", "comment"], function(key, fieldName){

$("[name='" + fieldName + "']").keyup(revalidateForm);

});

7. Проверьте работоспособность формы. После первой валидации, проверка полей должна происходить после каждого нажатия на клавишу (см. рисунок 5.2).

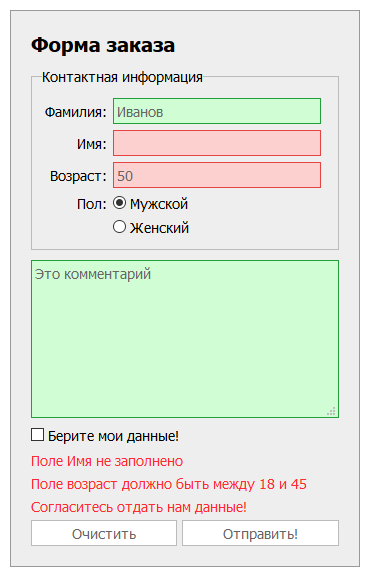


Рис. 5.2 – Валидация формы работает.

8. Оцените экономию кода, полученную в результате использования библиотеки jQuery, в процентах по сравнению с кодом, полученным в лабораторной работе 4. Для каких операций потребовалось написать значительно меньше кода?

1. Вопросы для самоконтроля и задачи
2. Для чего нужна библиотека jQuery?
3. Как можно подключить библиотеку jQuery?
4. Как обозначается вызов функции jQuery?
5. Какой инструкцией в jQuery можно изменить цвет фона всех элементов <p>?
6. Как воздействовать на группу элементов в соответствии с используемым ею классом?

# Методические рекомендации по выполнению лабораторной работы № 6. Адаптивная верстка web-страницы

1. Цель работы

Целью выполнения работы является формирование у обучающихся знаний о возможностях библиотеки Bootstrap и выработка умений по созданию адаптивной верстки с ее помощью.

На занятии необходимо сформировать адаптивную web-страницу с использованием библиотеки Bootstrap. Создаваемая страница должна занимать по ширине доступное пространство окна браузера и перестраиваться для удобного просмотра на мобильных устройствах (см. рисунок 6.1).

|  |  |
| --- | --- |
| C:\Users\dCm\AppData\Local\Microsoft\Windows\INetCache\Content.Word\full_lg.png | C:\Users\dCm\AppData\Local\Microsoft\Windows\INetCache\Content.Word\full_md.png |

Рис. 6.1 – Поведение макета при различной ширине просмотра.

1. Условия и порядок проведения лабораторной работы

Для проведения занятия необходимо знание лекционного материала по теме лекции № 11. Занятие проводится в форме тренинга, в ходе которого сначала осуществляется перекрестный опрос по теме лекции № 11, затем обучающиеся выполняют задание индивидуально, последние 15 минут занятия они попарно проверяют код друг друга, выявляя ошибки и недочеты, формируя навыки по анализу корректности кода.

1. Необходимые теоретические сведения

Всего лишь несколько лет назад сайты могли создаваться с фиксированной шириной в расчете на то, что все конечные пользователи получат удобные условия работы. Эта фиксированная ширина (обычно 960 пикселов или около того) была не слишком велика для экранов ноутбуков, а пользователи, имеющие мониторы с высоким разрешением, просто видели с обеих сторон большие поля.

Но в 2007 году на телефонах iPhone компании Apple впервые появились понастоящему удобные условия просмотра информации и условия доступа людей к web-данным и работы с ними изменились навсегда. В середине 2015 года в статистической системе gs.statcounter.com утверждалось, что данный показатель увеличился до 33,47 %. Для сравнения, в компании North America Mobile этот показатель достиг 25,86 %.

Использование мобильных устройств растет во всем мире, с другой стороны, активно применяются 27- и 30-дюймовые мониторы. Растет и разрыв между наименьшими и наибольшими экранами, на которых просматривается web-содержимое. Благо есть решение, подходящее для постоянно расширяющейся арены браузеров и устройств. Нормальная работа сайта на множестве устройств и экранов возможна благодаря так называемому адаптивному дизайну, построенному на основе применения HTML5 и CSS3. Эта технология позволяет разметке и возможностям сайта подстраиваться под имеющееся окружение (размер экрана, тип ввода и возможности устройства или браузера).

Понятие «адаптивный web-дизайн» было введено Итаном Маркоттом в 2010 году. В статье на сайте www.alistapart.com/articles/responsive-web-design/ он свел воедино три уже существо­вавшие на тот момент технологии (гибкий макет на основе сетки, подстраиваемые по размеру изображения и элементы мультимедиа, а также медиазапросы) в еди­ный унифицированный подход, который он назвал адаптивным web-дизайном, то есть, адаптивный web-дизайн явля­ется представлением web-содержимого в наиболее удобном формате для окна про­смотра и устройства, обращающегося за этим содержимым.

Более того, адаптивный web-дизайн, создаваемый с помощью HTML5 и CSS3, может быть реализован без необходимости использования конечных решений на серверной стороне.

Зачастую люди путают два понятия − резиновый и адаптивный web-дизайн, однако, они не являются чистокровными синонимами друг друга. Адаптивный web-дизайн подразумевает работу с различными макетами сайта в целях создания максимального удобства их использования на всевозможных типах устройств, учитывая всевозможные разрешения экранов.

Он «разумно» подгоняет общую картинку под экран в зависимости от размера дисплея на аппарате. Ну а резиновый дизайн − это, по сути, один из элементов адаптивного web-дизайна. Он, в свою очередь, лишь вносит изменения в пропорции web-страницы.

Таким образом, можно прийти к выводу о том, что адаптивный web-дизайн представляет собой комбинацию из резинового, а также прочих прогрессивных улучшений. Хотя из-за близких по духу определений порой и может возникать путаница. Особенно неприятно, когда разногласия в этом плане происходят с заказчиком, поэтому будьте внимательны и всегда уточняйте, что клиент имел ввиду, говоря о резиновом дизайне сайта.

На ранней стадии развития наиболее характерным для адаптивного дизайна было построение, начинающееся с «рабочего стола», то есть дизайна с фиксированной шириной. Затем, чтобы этот дизайн работал на экранах меньшего размера, содержимое автоматически переформатировалось или удалялось. Но процесс не стоял на месте, и стало понятно, что все, от дизайна до содержимого и разработки, получается намного лучше, если действовать в обратном направлении, начиная с небольших экранов и работая по нарастающей.

Благодаря популярности и возможности повсеместного использования адаптивного web-дизайна теперь стало намного легче коммуницировать владельцу сайта с пользователями. Некоторые представления об адаптивном web-дизайне сложились уже у большинства людей. Понятие единого кода, способного работать практически на всех устройствах, становится весьма привлекательным.

При запуске проекта с адаптивным дизайном почти всегда возникает вопрос поддержки со стороны браузеров. При наличии столь широкого спектра браузеров и устройств вряд ли имеет смысл реализовывать полную поддержку изменений каждого отдельно взятого браузера. Как правило, чем старее браузер, тем больший объем работы и кода требуется для достижения желаемого результата или выравнивания эстетического восприятия с тем, которое пользователь получает при работе с современными браузерами. Поэтому рациональнее иметь менее вариативный и благодаря этому более быстродействующий код за счет распределения создаваемых впечатлений по уровням, обеспечивая получение самых совершенных визуальных эффектов и возможностей только на наиболее восприимчивых к ним браузерах.

За исключением экстремальных ситуаций, создаваемые сайты должны работать как минимум на каждом из самых распространенных браузеров. Кроме основных функциональных возможностей, для каждого web-проекта имеет смысл заранее определить, на каких платформах нужно в полной мере создать наилучшие впечатления от его работы, а на каких вполне возможно будет согласиться на визуальные или функциональные отступления.

На практике легче начинать с представления, создаваемого самым простым, базовым уровнем, и заниматься его усложнением (этот подход называется постепенным усложнением), чем подходить к решению проблемы с противоположной стороны — заниматься сначала созданием представления самого высокого уровня, а затем предпринимать попытки отступления для работы на платформах с более скромными возможностями (этот подход называется постепенным упрощением).

При рассмотрении вопроса о том, какие функции какими платформами и версиями браузеров поддерживаются, можно ознакомиться с сайтом http:// caniuse.com − там предоставляется весьма простой интерфейс для выявления, поддержкой какого браузера пользуются интересующие нас функции.

### **Библиотека Bootstrap**

Bootstrap − это свободный набор инструментов для создания [сайтов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82) и [web-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Включает в себя [HTML](https://ru.wikipedia.org/wiki/HTML)- и [CSS](https://ru.wikipedia.org/wiki/CSS)-шаблоны оформления для [типографики](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0), web-форм, кнопок, меток, блоков навигации и прочих компонентов web-интерфейса, включая [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-расширения.

Bootstrap использует современные наработки в области CSS и HTML, поэтому необходимо быть внимательным при поддержке старых браузеров.

Чтобы подключить данную библиотеку, можно зайти на [getbootstrap.com](http://getbootstrap.com/), там можно скачать .zip архив целиком либо сформировать для себя список элементов и компонентов, которые вы хотите применить в работе. Так, можно избавиться от лишнего, оставив только нужное. Есть даже такие разработчики, которые берут от Bootstrap только сетку и вполне довольны этим. Вы же можете смело менять переменные стилей, отвечающие за цвета и формы, шрифты.

После того как выбрали нажмите «Compile and Download», начнет скачиваться архив с файлом bootstrap.min.css и уже минифицированным bootstrap.min.css, скомпилированным специально для вас! Да, вот такая оптимизация уже на уровне выбора элементов, да еще и с минификацией, позволяет сделать файл стилей на 25% легче чем не минифицированные и еще легче за счет отказа от ненужных элементов.

Затем нужно подключить этот файл. Делается это просто, на всякий случай напомню, как именно:

<link rel='stylesheet' href='/css/bootstrap.min.css' type='text/css' media='all'>

Если вам нужен будет джейквери на сайте, то не забудьте подключить и его:

<script src="http://code.jquery.com/jquery-latest.min.js"></script>

Либо подключите последнюю версию с Гугл-АПИ или скачайте библиотеку себе на сайт и подгружайте со своего хостинга. Затем можно добавить и .js файл самого Bootstrap:

<script src="js/bootstrap.min.js"></script>

Данный скрипт нужен для того, чтобы в старых версиях IE было все нормально:

<script src="js/respond.min.js"></script>

Теперь Bootstrap подключен и настроен.

Основные преимущества Bootstrap:

1. Экономия времени − потому что используются уже готовые классы и дизайн.
2. Адаптивность, высокая скорость и оптимизация, стандартизация интерфейсов − динамичные макеты Bootstrap качественно отображаются на самых разных устройствах без необходимости внесения изменений в разметку.
3. Дизайн − единые шаблоны и стилевое оформление элементов макета и всех страниц на сайте в целом. И при этом Bootstrap кросс-браузерный и хорошо отображается во всех браузерах Safari, Firefox, IE, EDGE и тех, что на основе Chromium (движок Blink на основе Webkit: Яндекс.Браузер, Opera, Google Chrome). Регулярное обновление и дополнение фреймворка самыми современными возможностями HTML и CSS вносит некоторые ограничения в использовании с IE7 и IE8 − не забудьте проверить.
4. Простота и открытость − использовать Bootstrap настолько просто, что с ним справляются даже школьники и начинающие web-разработчики, а открытый исходный код позволяет самому участвовать в разработке, модифицировать под свои нужды или просто пользоваться хорошим бесплатным решением.

При этом код HTML, JavaScript и CSS в Bootstrap продуман и рассмотрен под микроскопом сотнями разработчиков со всего мира − все для того, чтобы рядовые web-мастера и верстальщики могли легко и просто настроить сетку сайта или встроить необходимые элементы в интерфейс.

Также в Bootstrap используется динамический язык стилей LESS, которые расширяет возможности CSS − разработчики могут управлять цветами, создавать вложенные колонки и переменные.

Развитие интернета и web-сайтов показало, что в 2016 году половина пользователей интернета посещает сайты с мобильных устройств и небольших экранов. Это значит, что современный web-разработчик должен думать о функционировании сайта не только на стационарных устройствах, но и на смартфонах и планшетах с тачскрином.

Создание отдельной мобильной версии для сайта конечно выход, но тогда потребуется делать в два раза больше работы на разработку и поддержание кода, а это не всегда экономически эффективно. Это может позволить себе крупная компания, которая для мобильных пользователей делает еще и приложение для смартфонов.

Концепция отзывчивого web-дизайна, воплощенного в фреймворке Бутстрап, решает именно эту проблему: сайт одинаково «отзывается» и отображает информацию наиболее полным образом вне зависимости от типа экрана и размера устройства. Содержание и цветовая гамма не меняются, меняется только форма и способ сгруппировать информационные и навигационные блоки сайта наиболее удобным для пользователя способом.

### **Компоненты и инструменты в Bootstrap**

Бутстрап имеет широкий список инструментов, облегчающих жизнь разработчика:

1. **Сетка** − адаптивная 12-колоночная сетка с фиксированными размерами колонок.
2. **Шаблон** − может быть резиновым или фиксированным.
3. **Типографика** − можно добротно оформить код, цитаты, абзацы, заголовки, заголовки со вторичным текстом, подзаголовки, выравнивание текста, аббревиатуры и т.п. Оформление уже прописано в css-классах, вам достаточно подключить стили и сделать верную разметку документа классами.
4. **Медиа** − позволяет красиво оформлять картинки и видео.
5. **Таблицы** − можно оформить таблицу, в том числе добавив возможность сортировки.
6. **Навигация** − для создания навигационного меню на сайте. Bootstrap упрощает ее, предлагая табы, постраничную навигацию, хлебные крошки, вкладки, боковое и главное меню, навигационный тулбар и пр.
7. **Формы** − позволяет создавать различные формы: в одну или несколько строк, с подсказками и валидацией полей, с радио- или чекбоксами, лейблами и выпадающими кнопками. При этом вы сможете подсвечивать неправильно заполненную часть формы, давая предупреждение пользователю при возникновении ошибки в заполнении.
8. **Алерты** − для оформления диалоговых окон. В основном это всплывающие окна или подсказки − ошибка (danger), предупреждение (warning), успех (success), подсказка (info). И использовать просто, просто дописываете необходимый класс и все готово.
9. **Кнопки** − в т.ч. «выпадающие» кнопки.
10. **Прогресс**-бары.
11. **Шрифт** из иконок − прекрасная возможность добавить красивых элементов в [оформлении сайта](http://blogwork.ru/oformlenie-sajta/), при этом можно даже отказаться от формирования и использования иконочных спрайтов. Правда с ограничениями: у исходной копии иконки может быть только один цвет.

В общих чертах это примерно ½ от всех возможностей (многие из пунктов имеют целые ответвления по возможности оформления). Бутстрап позволяет не создавать свои CSS-конструкции, а пользоваться готовым решением, творить на основе понятного и очень продуманного инструмента. Вместо того чтобы собирать все свои решения и наработки, web-разработчику достаточно пользоваться инструментарием Bootstrap.

Использование Bootstrap 3 подразумевает четкую стандартизацию классов и стилей, благодаря чему требуется меньше времени чтобы разобраться в коде при переходе из проекта в проект.

1. Ход выполнения работы

1. Создадим web-страницу по стандарту html 5 и подключим библиотеки jQuery и Bootstrap:

<link href="css/bootstrap.min.css" rel="stylesheet">

<script src="js/jquery-3.2.1.min.js"></script>

<script src="js/bootstrap.min.js"></script>

2. Создадим и подключим css-файл template.css. В него будем помещать дополнительные css правила.

<link href="css/template.css" rel="stylesheet">

3. Установим область просмотра для мобильных устройств:

<meta name="viewport" content="width=device-width, initial-scale=1">

4. С использованием компонента navbar создадим меню, изображенное на рисунке 6.2. В выпадающее меню "Страницы" поместим три страницы: Иванов, Петров, Сидоров.

C:\Users\dCm\AppData\Local\Microsoft\Windows\INetCache\Content.Word\screenshot--2017-11-29-17-10-11.png

Рис. 6.2 – Меню с использованием компонента navbar.

Проверим, что при уменьшении ширины страницы меню принимает следующий вид:

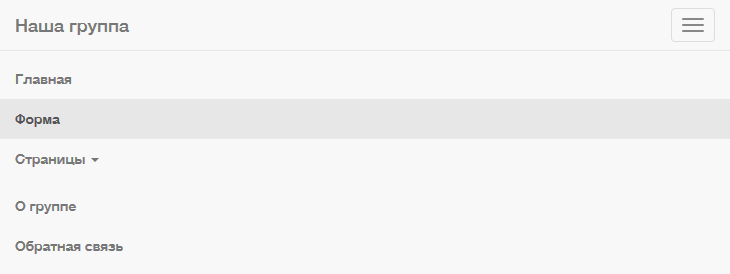


Рис. 6.3 – Меню при уменьшении ширины экрана.

5. Зафиксируем меню в верхней части экрана. Для этого в контейнере меню выставим класс navbar-fixed-top. Для того, чтобы остальная часть страницы не заезжала под меню в подключенном файле template.css пропишем следующий стиль:

body {padding-top: 70px;}

6. Создадим основной контейнер верстки:

<div class="container">

<div class="row">

</div>

</div>

Внутри контейнера создадим две колонки в пропорции 3 (слева) к 9 (справа) таким образом, что при сжатии страницы до размера md левая колонка опускается под содержимое правой колонки. Для перестановки колонок использовать классы col-md-push-\*.

7. В основной колонке создадим хлебные крошки (см. рисунок 6.4)

C:\Users\dCm\AppData\Local\Temp\screenshot--2017-11-29-17-23-27.png

Рис. 6.4 – Хлебные крошки.

8. Вставим в основное окно компонент jumbotron:

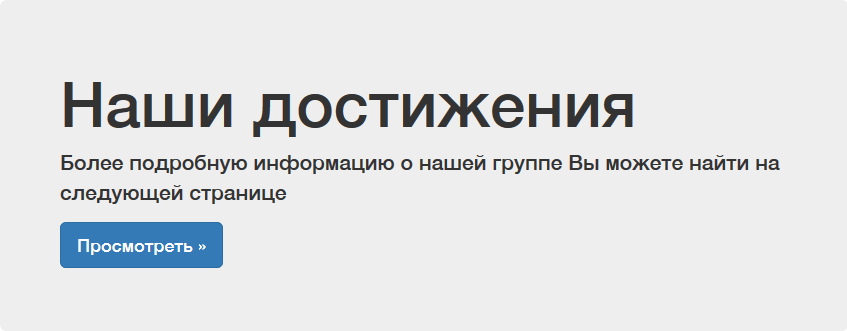


Рис. 6.5 – Jumbotron.

9. Используя компонент Thumbnail создадим в основном окне галерею персонажей. Для каждого блока зададим размер в ¼ сетки для среднего экрана (рисунок 6.6 (а)), и ½ для маленького (рисунок 6.6 (б)):

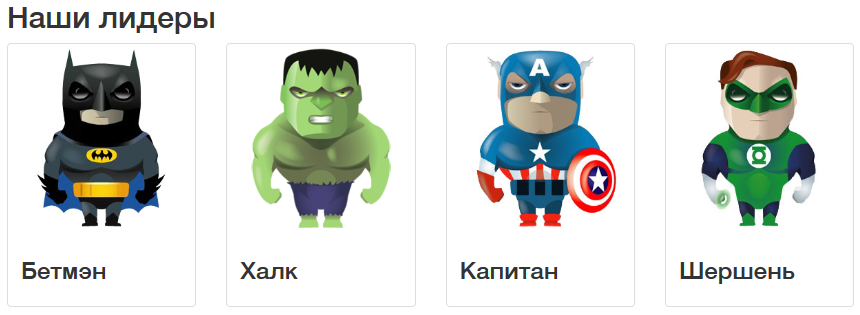


Рис. 6.6 (а) – Thumbnail.



Рис. 6.6 (б) – Thumbnail.

10. Добавим в созданную галерею героев шкалу их успехов с помощью компонента Progressbar. Установим для героев размеры элемента progressbar в 50%, 20%, 75% и 100% соответственно:

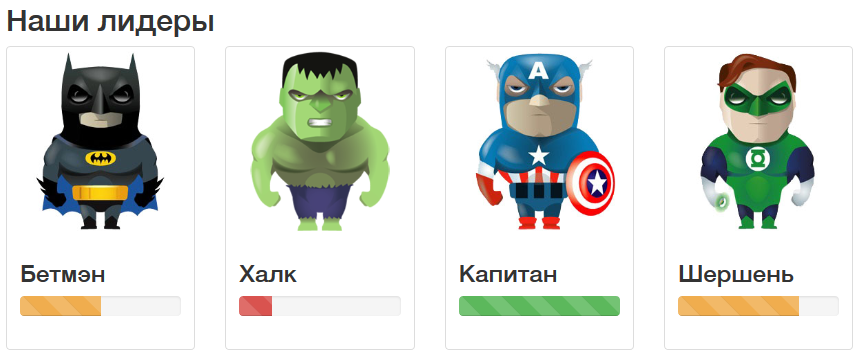


Рис. 6.7 – Использование Progressbar.

11. Добавим в основное окно таблицу с успеваемостью (см. рисунок 6.8). Для оформления таблицы используем классы Bootstrap: table, table-striped, table-bordered, success, danger. Предупреждение перед таблицей выведем с помощью компонента alert:



Рис. 6.8 – Успеваемость.

12. Завершим создание дизайна основного окна с помощью компонента Tabs:

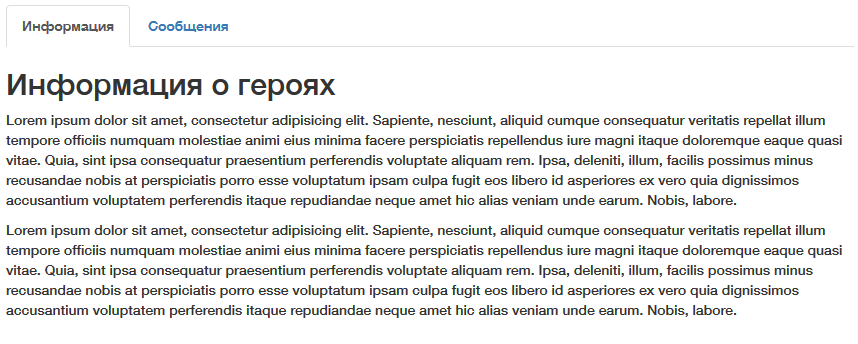


Рис. 6.9 – Tabs.

13. В левую колонку добавим меню с помощью компонента List group. Метку "new" поставим с помощью span с классом badge:

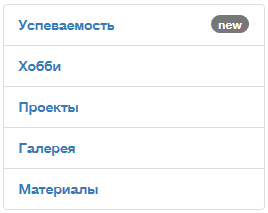


Рис. 6.10 – List group.

14. Создадим форму входа в личный кабинет. Для оформления текста используем заговок h3 и тег p с классом text-muted. Клавишу выделим с помощью тега kbr. Для создания полей формы используем контейнеры input-group и иконки из шрифта glyphicon.

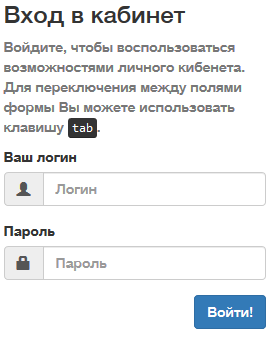


Рис. 6.11 – Форма входа.

15. Создадим блок цитаты с помощью тега blockquote:

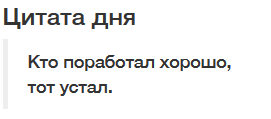


Рис. 6.12 – Цитата.

16. За основным контейнером создадим подвал:

<footer class="footer">

<div class="container">

<p class="text-muted pull-right">© Сделано инициативной группой на <a href="http://getbootstrap.com/docs/3.3/"> Bootstrap</a>.</p>

</div>

</footer>

Стилизуем подвал с помощью следующего правила в template.css:

.footer {margin-top: 30px; height: 50px; background-color: #f5f5f5;}

17. Проверим адаптивность верстки при изменении горизонтального размера окна.

1. Вопросы для самоконтроля и задачи
2. Чем различаются резиновый и адаптивный дизайны?
3. Для чего нужна библиотека Bootstrap и каковы ее основные преимущества?
4. Как подключить библиотеку Bootstrap?
5. Что из себя представляют основные инструменты Bootstrap?

# СПИСОК ЛИТЕРАТУРЫ

1. Гоше Х. Д. HTML5 для профессионалов [Текст] / Х. Д. Гоше ; пер. с англ. Е. Шикаревой, В. Крупника. - 2-е изд. - СПб. : Питер, 2015. - 559 с.
2. Дронов В. А.    HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов [Текст] / В. А. Дронов. - СПб. : БХВ-Петербург, 2014. - 414 с.
3. Ллойд Й.    Создай свой веб-сайт с помощью HTML и CSS [Текст] / Й. Ллойд ; пер. с англ. О. Потаповой. - СПб. : Питер, 2013. - 412 с.
4. Никсон Р.    Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS HTML5 [Текст] / Р. Никсон ; пер. с англ. Н. Вильчинского. - 3-е изд. - СПб. : Питер, 2015.
5. Прохоренок Н. А.    HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера [Текст] / Н. А. Прохоренок, В. А. Дронов. - 4-е изд., перераб. и доп. - СПб. : БХВ-Петербург, 2015. - 747 с.
6. Роббинс Д.    HTML5, CSS3 и JavaScript. Исчерпывающее руководство [Текст] / Д. Роббинс ; пер. с англ. М. А. Райтмана. - М. : ЭКСМО, 2014. - 528 с.
7. Робсон Э. Изучаем HTML, XHTML и CSS [Текст] / Э. Робсон, Э. Фримен ; пер. с англ. В. Черник. - 2-е изд. - СПб. : Питер, 2015. - 720 с.
8. Ташков П. А.    Веб-мастеринг : HTML, CSS, JavaScript, PHP, CMS, AJAX, раскрутка [Текст] / П. А. Ташков. - СПб. : ПИТЕР, 2010. - 512 с.
9. Хольцшлаг М.    Языки HTML и CSS: для создания Web-сайтов [Текст] : [учеб. пособие] / М. Хольцшлаг ; пер. с англ. А. Климович. - М. : ТРИУМФ, 2006. - 303 с.
10. Фрэйн Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств. 2-е изд. — СПб.: Питер, 2017. − 272 с.
11. Электронный ресурс. URL: http://htmlbook.ru (дата обращения : 20.04.2019).

# СОДЕРЖАНИЕ

[Предисловие 3](#_Toc12866874)

[Методические рекомендации по выполнению лабораторной работы № 1. Разработка документа с помощью каскадной таблицы стилей 4](#_Toc12866875)

[Методические рекомендации по выполнению лабораторной работы № 2. Создание документа по макету с использованием табличной верстки 16](#_Toc12866876)

[Методические рекомендации по выполнению лабораторной работы № 3. Создание документа по макету с использованием блочной верстки 30](#_Toc12866877)

[Методические рекомендации по выполнению лабораторной работы № 4. Построение интерфейса с использованием сценариев JavaScript 38](#_Toc12866878)

[Методические рекомендации по выполнению лабораторной работы № 5. Использование библиотеки jQuery для построения web-интерфейсов 55](#_Toc12866879)

[Методические рекомендации по выполнению лабораторной работы № 6. Адаптивная верстка web-страницы 70](#_Toc12866880)

[СПИСОК ЛИТЕРАТУРЫ 81](#_Toc12866881)

[СОДЕРЖАНИЕ 82](#_Toc12866882)