Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Филиал «Минский радиотехнический колледж»

**УТВЕРЖДаю**

Директор МРК

\_\_\_\_\_\_\_\_\_\_\_\_\_ С.Н. Анкуда

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Регистрационный №\_\_\_\_\_\_\_\_

**Internet-программирование для мобильных**

**устройств**

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**

**по выполнению лабораторных работ**

для учащихся специальности

2-39-03-02 «Программируемые мобильные системы»

Минск 2019

**СОСТАВИТЕЛИ:** О.И. Терешко, преподаватель дисциплин общепрофессионального и специального циклов, О.Н. Виничук, преподаватель дисциплин общепрофессионального и специального циклов

**РЕКОМЕНДОВАНЫ К УТВЕРЖДЕНИЮ:**

Цикловой комиссией «Программное обеспечение информационных технологий» филиала БГУИР «Минский радиотехнический колледж»

Протокол № от

Заседанием педагогического совета филиала БГУИР «Минский радиотехнический колледж»

Протокол № \_\_\_\_\_от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Методическая экспертиза \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись ФИО

**Содержание**

# **Лабораторная работа № 1**

## **Тема работы: «Создание и форматирование простейшего HTML-документа. Организация системы ссылок сайта. Работа с изображениями. Создание системы навигации для перехода на внешние страницы и метки в пределах текущей. Размещение на странице графических объектов»**

**1. Цель работы**

Формирование умений создания простейших HTML-страниц, семантической разметки HTML-документа, создания ссылок, в том числе с графическими элементами привязки, внедрения графических объектов в HTML-документ.

**2. Задание**

Создать простейшую HTML-станицу сайта-визитки.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Для создания и форматирования Web-документов используется язык гипертекстовой разметки HTML (Hyper Text Markup Language). В HTML определен стандартный набор тегов (дескрипторов) – команд, определяющих форматирование документа. Теги заключаются в треугольные скобки < >. Большинство тегов, как правило, используются парами. Сначала указывается открывающий тег, который объясняет браузеру, что делать с последующим текстом. Затем следует закрывающий тег, ограничивающий область действия первого. Закрывающий тег отличается от открывающего наличием косой черты (слэша). В некоторых случаях закрывающий тег не требуется.

В HTML регистр символов, определяющих теги, не учитывается.

Web-документ ограничивается тегами <HTML> и </HTML>, которые определяют соответственно начало и конец документа.

В структуре HTML-документа выделяются заголовок (<HEAD>

</HEAD>) и тело документа (<BODY> </BODY>). Заголовок может содержать заключенное в теги <TITLE> </TITLE> заглавие (или название) страницы, а также META-информацию.

Для создания HTML-документа нужно выполнить следующую последовательность действий:

– запустить приложение «Блокнот»;

– набрать исходный текст документа, например:

<HTML>

<HEAD><TITLE> Заглавие документа </TITLE></HEAD>

<BODY>

Содержимое документа

</BODY>

</HTML>

HTML5 – язык для структурирования и представления содержимого всемирной паутины.

HTML-документ – это обычный текстовый документ, может быть создан как в обычном текстовом редакторе (Блокнот), так и в специализированном, с подсветкой кода (Notepad++, Visual Studio Code и т.п.). HTML-документ имеет расширение .html.

HTML-документ состоит из дерева HTML-элементов и текста. Каждый элемент обозначается в исходном документе начальным (открывающим) и конечным (закрывающим) тегом (за редким исключением).

Начальный тег показывает, где начинается элемент, конечный – где заканчивается. Закрывающий тег образуется путем добавления слэша / перед именем тега:

<имя тега> … </имя тега>.

Между начальным и закрывающим тегами находится содержимое тега – контент.

Одиночные теги не могут хранить в себе содержимого напрямую, оно прописывается как значение атрибута, например, тег <input type="button" value="Кнопка"> создаст кнопку с текстом Кнопка внутри.

Теги могут вкладываться друг в друга, например, <p><i>Текст</i></p>. При вложении следует соблюдать порядок их закрытия (принцип «матрёшки»), например, следующая запись будет неверной: <p><i>Текст</p></i>.

HTML-элементы могут иметь атрибуты (глобальные, применяемые для всех HTML-элементов, и собственные). Атрибуты прописываются в открывающем теге элемента и содержат имя и значение, указываемые в формате имя атрибута="значение". Атрибуты позволяют изменять свойства и поведение элемента, для которого они заданы.

Кроме того, для обеспечения корректного отображения документа современный стандарт требует использования одиночного тега <!DOCTYPE>. DOCTYPE отвечает за корректное отображение веб-страницы браузером. DOCTYPE определяет не только версию HTML (например, html), но и соответствующий DTD-файл в Интернете.

<!DOCTYPE html> <!— Объявление формата документа —>

<html>

<head> <!— Техническая информация о документе —>

<meta charset="UTF-8"> <!— Определяем кодировку символов документа —>

<title>...</title> <!— Задаем заголовок документа —>

<link rel="stylesheet" type="text/css" href="style.css"> <!— Подключаем внешнюю таблицу стилей —>

<script src="script.js"></script> <!— Подключаем сценарии —>

</head>

<body> <!— Основная часть документа —>

</body>

</html>

Стандарт HTML5 предоставил новые элементы для структурирования, группировки контента и разметки текстового содержимого. Новые семантические элементы позволили улучшить структуру веб-страницы, добавив смысловое значение заключенному в них содержимому (было <div id="header">, стало <header>). Для отображения внешнего вида элементов не задано никаких правил, поэтому элементы можно стилизовать по своему усмотрению. Для всех элементов доступны глобальные атрибуты.

Согласно спецификации HTML5, каждый элемент принадлежит к определенной (ноль или более) категории. Каждая из них группирует элементы со схожими характеристиками.

Элемент <header>. Группирует вводные и навигационные элементы, не является обязательным. Может содержать заголовки, оборачивать содержание раздела страницы, форму поиска или логотип. В HTML-документе может содержаться одновременно несколько элементов <header> и они могут располагаться в любой части страницы.

Элемент <nav>. Предназначен для создания блока навигации веб-страницы или всего веб-сайта, при этом не обязательно должен находиться внутри <header>. На странице может быть несколько элементов <nav>. Не заменяет теги <ul> или <оl>, он просто их обрамляет. Не все группы ссылок на странице должны быть обёрнуты <nav>, этот элемент предназначен в первую очередь для разделов, которые состоят из главных навигационных блоков.

Элемент <section>. Элемент представляет собой универсальный раздел документа. Группирует тематическое содержимое и обычно содержит заголовок. Не является блоком-оберткой, для этих целей уместнее использовать элемент <div>. В качестве содержимого может выступать оглавление, разделы научных публикаций, программа мероприятия. Домашняя страница сайта также может быть поделена на секции — блок вводной информации, новости и контакты. Элемент рекомендуется использовать только в том случае, если содержимое элемента будет явно указано в схеме документа.

Элемент <aside>. Группирует содержимое, связанное с окружающим его контентом напрямую, но которое можно счесть отдельным (т.е., удаление этого блока не повлияет на понимание основного содержимого). Чаще всего элемент позиционируется как боковая колонка (как в книгах) и включает в себя группу элементов: <nav>, цифровые данные, цитаты, рекламные блоки, архивные записи. Не подходит для блоков, просто позиционированных в стороне.

Элемент <footer>. Представляет собой нижний колонтитул содержащей его секции или корневого элемента. Обычно содержит информацию об авторе статьи, данные о копирайте и т.д. Если используется как колонтитул всей страницы, содержимое дополняется сведениями об авторских правах, ссылками на условия использования, контактную информацию, ссылками на связанное содержимое и т.п.

В одном веб-документе может быть несколько элементов <footer>. Как каждая страница, так и каждая статья может иметь свой элемент <footer>, более того, <footer> можно поместить в элемент <blockquote>, чтобы указать источник цитирования.

Элемент <main>. Элемент <main> представляет основное содержимое документа (содержимое элемента <body>). Контент, находящийся внутри элемента, должен быть уникальным и не повторяться во всех документах сайта, таких как навигационные ссылки, информация о копирайте, логотипы, формы поиска (в случае, если форма поиска является основной функцией документа).

Элемент <main> не может быть потомком таких элементов как <article>, <aside>, <footer>, <header> или <nav>.

Пример структурной разметки приведен на рисунке 1.1.

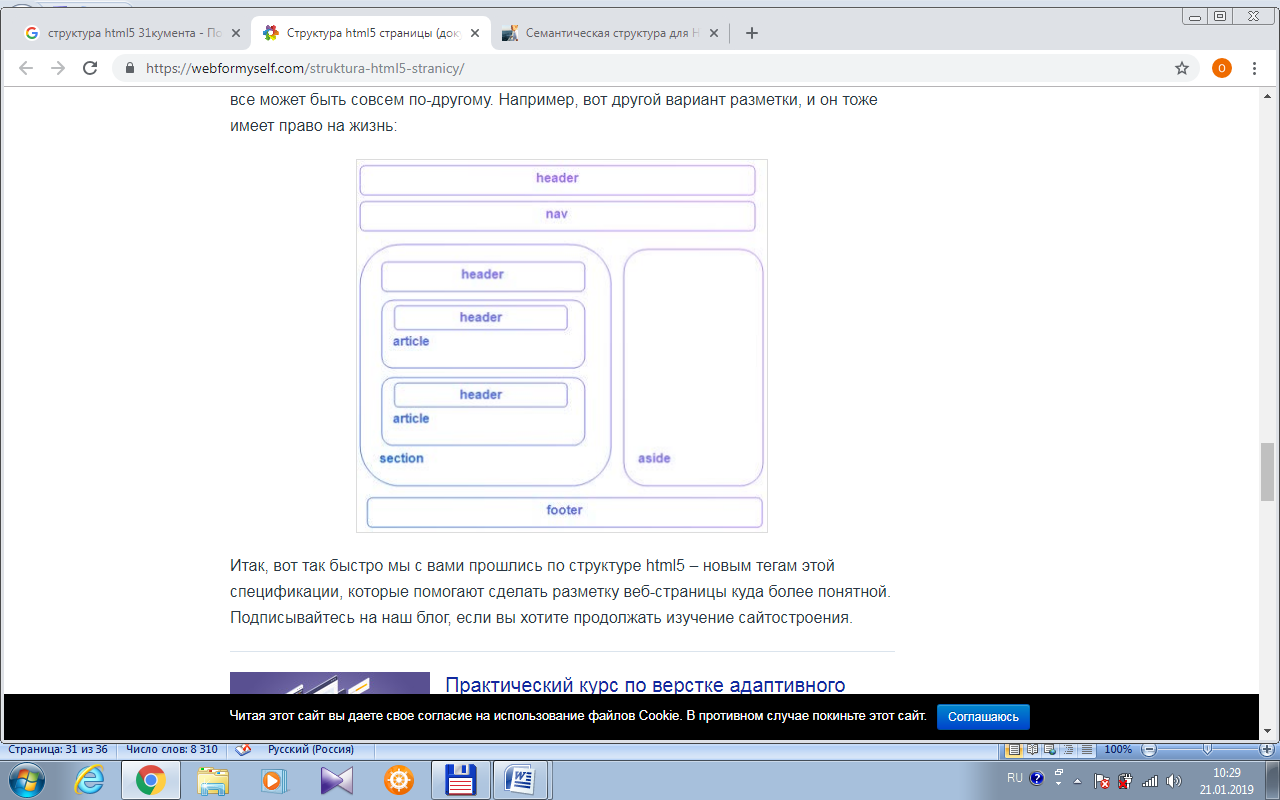


Рисунок 1.1 – Пример семантической разметки веб-документа

В спецификации не указаны жесткие правила по использованию семантических тегов, указаны лишь рекомендации по их использованию. Если нет четкого понимания, где и какой HTML5 тег использовать, лучше используйте div – чтобы не навредить и не нарушить структуру документа.

HTML-теги – основа языка HTML. Теги используются для разграничения начала и конца элементов в разметке.

Все HTML-элементы делятся на пять типов:

– пустые элементы – <area>, <base>, <col>, <embed>, <hr>, <img>, <input>, <keygen>, <link>, <menuitem>, <meta>, <param>, <source>, <track>, <wbr>;

– элементы с неформатированным текстом – <script>, <style>;

– элементы, выводящие неформатированный текст – <textarea>, <title>;

– элементы из другого пространства имён – MathML и SVG;

– обычные элементы – все остальные элементы.

Более подробный перечень семантических элементов HTML5 представлен в таблице 1.1.

Таблица 1.1 – Семантические элементы HTML5

|  |  |
| --- | --- |
| **Тег** | **Описание** |
| **1** | **2** |
| <!DOCTYPE> | Объявляет тип документа и предоставляет основную информацию для браузера – его язык и версия. |
| <a> | Создаёт гипертекстовые ссылки. |
| <abbr> | Определяет текст как аббревиатуру или акроним. Поясняющий текст задаётся с помощью атрибута title. |
| <address> | Задает контактные данные автора/владельца документа или статьи. Отображается в браузере курсивом. |
| <area> | Представляет собой гиперссылку с текстом, соответствующей определенной области на карте-изображении или активную область внутри карты-изображения. Всегда вложен внутрь тега <map>. |
| <article> | Раздел контента, который образует независимую часть документа или сайта, например, статья в журнале, запись в блоге, комментарий. |
| <aside> | Представляет контент страницы, который имеет косвенное отношение к основному контенту страницы/сайта. |
| <audio> | Загружает звуковой контент на веб-страницу. |
| <b> | Задает полужирное начертание отрывка текста, не придавая акцент или важность выделенному. |
| <blockquote> | Выделяет текст как цитату, применяется для описания больших цитат. |
| <body> | Представляет тело документа (содержимое, не относящееся к метаданным документа). |
| <br> | Перенос текста на новую строку. |
| <button> | Создает интерактивную кнопку. Внутрь тега можно поместить содержимое – текст или изображение. |
| <caption> | Добавляет подпись к таблице. Вставляется сразу после тега <table>. |
| <code> | Представляет фрагмент программного кода, отображается шрифтом семейства monospace. |
| <del> | Помечает текст как удаленный, перечёркивая его. |
| <div> | Тег-контейнер для разделов HTML-документа. Используется для группировки блочных элементов с целью форматирования стилями. |
| <dl> | Тег-контейнер, внутри которого находятся термин и его описание. |
| <dt> | Используется для задания термина. |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| **1** | **2** |
| <em> | Выделяет важные фрагменты текста, отображая их курсивом. |
| <fieldset> | Группирует связанные элементы в форме, рисуя рамку вокруг них. |
| <figcaption> | Заголовок/подпись для элемента <figure>. |
| <figure> | Самодостаточный тег-контейнер для такого контента как иллюстрации, диаграммы, фотографии, примеры кода, обычно с подписью. |
| <footer> | Определяет завершающую область (нижний колонтитул) документа или раздела. |
| <form> | Форма для сбора и отправки на сервер информации от пользователей. Не работает без атрибута action. |
| <h1-h6> | Создают заголовки шести уровней для связанных с ними разделов. |
| <head> | Элемент-контейнер для метаданных HTML-документа, таких как<title>, <meta>, <script>, <link>, <style>. |
| <header> | Секция для вводной информации сайта или группы навигационных ссылок. Может содержать один или несколько заголовков, логотип, информацию об авторе. |
| <hr> | Горизонтальная линия для тематического разделения параграфов. |
| <html> | Корневой элемент HTML-документа. Сообщает браузеру, что это HTML-документ. Является контейнером для всех остальных html-элементов. |
| <i> | Выделяет отрывок текста курсивом, не придавая ему дополнительный акцент. |
| <iframe> | Создает встроенный фрейм, загружая в текущий HTML-документ другой документ. |
| <img> | Встраивает изображения в HTML-документ с помощью атрибута src, значением которого является адрес встраиваемого изображения. |
| <input> | Создает многофункциональные поля формы, в которые пользователь может вводить данные. |
| <label> | Добавляет текстовую метку для элемента <input>. |
| <legend> | Заголовок элементов формы, сгруппированных с помощью элемента <fieldset>. |
| <li> | Элемент маркированного или нумерованного списка. |
| <link> | Определяет отношения между документом и внешним ресурсом. Также используется для подключения внешних таблиц стилей. |
| <main> | Контейнер для основного уникального содержимого документа. На одной странице должно быть не более одного элемента <main>. |
| <map> | Создаёт активные области на карте-изображении. Является контейнером для элементов <area>. |
| <meta> | Используется для хранения дополнительной информации о странице. Эту информацию используют браузеры для обработки страницы, а поисковые системы — для ее индексации. В блоке <head> может быть несколько тегов <meta>, так как в зависимости от используемых атрибутов они несут разную информацию. |
| <nav> | Раздел документа, содержащий навигационные ссылки по сайту. |
| <noscript> | Определяет секцию, не поддерживающую сценарий (скрипт). |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| **1** | **2** |
| <object> | Контейнер для встраивания мультимедиа (например, аудио, видео, Java-апплеты, ActiveX, PDF и Flash). Также можно вставить другую веб-страницу в текущий HTML-документ. Для передачи параметров встраиваемого плагина используется тег <param>. |
| <ol> | Упорядоченный нумерованный список. Нумерация может быть числовая или алфавитная. |
| <optgroup> | Контейнер с заголовком для группы элементов <option>. |
| <option> | Определяет вариант/опцию для выбора в раскрывающемся списке <select>, <optgroup> или <datalist>. |
| <p> | Параграфы в тексте. |
| <param> | Определяет параметры для плагинов, встраиваемых с помощью элемента <object>. |
| <progress> | Индикатор выполнения задачи любого рода. |
| <q> | Определяет краткую цитату. |
| <s> | Отображает текст, не являющийся актуальным, перечеркнутым. |
| <script> | Используется для определения сценария на стороне клиента (обычно JavaScript). Содержит либо текст скрипта, либо указывает на внешний файл сценария с помощью атрибута src. |
| <section> | Определяет логическую область (раздел) страницы, обычно с заголовком. |
| <select> | Элемент управления, позволяющий выбирать значения из предложенного множества. Варианты значений помещаются в <option>. |
| <small> | Отображает текст шрифтом меньшего размера. |
| <source> | Указывает местоположение и тип альтернативных медиа ресурсов для элементов <video> и <audio>. |
| <span> | Контейнер для строчных элементов. Можно использовать для форматирования отрывков текста, например, выделения цветом отдельных слов. |
| <strong> | Расставляет акценты в тексте, выделяя полужирным. |
| <style> | Подключает встраиваемые таблицы стилей. |
| <sub> | Задает подстрочное написание символов, например, индекса элемента в химических формулах. |
| <sup> | Задает надстрочное написание символов. |
| <table> | Тег для создания таблицы. |
| <tbody> | Определяет тело таблицы. |
| <td> | Создает ячейку таблицы. |
| <textarea> | Создает большие поля для ввода текста. |
| <tfoot> | Определяет нижний колонтитул таблицы. |
| <th> | Создает заголовок ячейки таблицы. |
| <thead> | Определяет заголовок таблицы. |
| <time> | Определяет дату/время. |
| <title> | Заголовок HTML-документа, отображаемый в верхней части строки заголовка браузера. Также может отображаться в результатах поиска, поэтому это следует принимать во внимание предоставление названия. |
| <tr> | Создает строку таблицы. |
| <track> | Добавляет субтитры для элементов <audio> и <video>. |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| **1** | **2** |
| <u> | Выделяет отрывок текста подчёркиванием, без дополнительного акцента. |
| <ul> | Создает маркированный список. |
| <var> | Выделяет переменные из программ, отображая их курсивом. |
| <video> | Добавляет на страницу видео-файлы. Поддерживает 3 видео формата: MP4, WebM, Ogg. |

Для включения комментариев в HTML-код используются последовательности символов <!-- и -->:

<!-- Это комментарий -->

HTML-ссылки создаются с помощью элементов <a>, <area> и <link>. Ссылки представляют собой связь между двумя ресурсами, одним из которых является текущий документ.

Ссылки можно поделить на две категории:

– ссылки на внешние ресурсы – создаются с помощью тега <link> и используются для расширения возможностей текущего документа при обработке браузером;

– гиперссылки – ссылки на другие ресурсы, которые пользователь может посетить или загрузить.

Гиперссылки создаются с помощью парного тега <a></a>. Внутрь тега помещается текст, который будет отображаться на веб-странице. Текст ссылки отображается в браузере с подчёркиванием, цвет шрифта – синий, при наведении на ссылку курсор мыши меняет вид.

Обязательным параметром тега <a> является атрибут href, который задает URl-адрес веб-страницы:

<a href="http://google.com/">указатель ссылки</a>

Ссылка состоит из двух частей – указателя и адресной части. Указатель ссылки представляет собой фрагмент текста или изображение, видимые для пользователя. Адресная часть ссылки пользователю не видна, она представляет собой адрес ресурса, к которому необходимо перейти.

Адресная часть ссылки состоит из URl. URl (Uniform Resource Locator) – унифицированный адрес ресурса. При создании адресов для разделения слов между собой рекомендуется использовать дефис, а не символ подчёркивания. В общем виде URl имеющий следующий формат: метод доступа://имя сервера:порт/путь

Метод доступа, или протокол, осуществляет обмен данными между рабочими станциями в разных сетях. Наиболее распространенные протоколы передачи данных: file обеспечивает чтение файла с локального диска: file:/gallery/pictures/summer.html.

http предоставляет доступ к веб-странице по протоколу HTTP: http://google.com/.

https – специальная реализация протокола HTTP, использующая шифрование (как правило, SSL или TLS): https://google.com/.

mailto запускает сеанс почтовой связи с указанным адресатом и хостом: mailto: nika@gmail.com.

Якоря, или внутренние ссылки, создают переходы на различные разделы текущей веб-страницы, позволяя быстро перемещаться между разделами. Это оказывается очень удобным в случае, когда на странице слишком много текста. Внутренние ссылки также создаются при помощи тега <a> с разницей в том, что атрибут href содержит имя указателя – так называемый якорь, а не URl-адрес. Перед именем указателя всегда ставится знак #.

Следующая разметка создаст оглавление с быстрыми переходами на соответствующие разделы:

<h1>Времена года</h1>

<h2>Оглавление</h2>

<a href="#p1">Лето</a> <!--создаём якорь, указав #id элемента-->

<a href="#p2">Осень</a>

<a href="#p3">Зима</a>

<a href="#p4">Весна</a>

<p id="p1">...</p> <!--добавляем соответствующий id элементу-->

<p id="p2">...</p>

<p id="p3">...</p>

<p id="p4">...</p>

Если нужно сделать ссылку с одной страницы сайта на определенный раздел другой страницы, то необходимо задать id для этого раздела страницы, а затем добавить его к абсолютному адресу ссылки:

<th id="about-color">color</th>

<a href=" https://google.com//#about-color" class="site" target="\_blank">color</a>

Чтобы сделать кликабельное изображение, необходимо поместить элемент <img> внутрь тега <a>. Чтобы ссылка открывалась в другом окне, нужно добавить атрибут target="\_blank" для ссылки:

<a href="https://google.com/” target="\_blank"> <img src="image.jpg" alt="Это картинка"></a>

У ссылок появились новые возможности – по клику можно не только переходить на другие страницы и скачивать файлы, но и совершать звонки на телефоны, отправлять сообщения или звонить по скайпу:

Ссылка на телефонный номер:

<a href="tel:+375291234567">+375 39 123 45 67</a>

Ссылка на адрес электронной почты:

<a href="mailto:example@mail.ru">example@mail.ru</a>

Ссылка на скайп (позвонить):

<a href="skype:имя-пользователя?call">Skype</a>

Ссылка на скайп (открыть чат):

<a href="skype:имя-пользователя?chat">Skype</a>

Атрибуты тега <a> представлены в таблице 1.2.

Таблица 1.2 – Атрибуты тега <a>

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| **1** | **2** |
| download | Дополняет атрибут href и сообщает браузеру, что ресурс должен быть загружен в момент, когда пользователь щелкает по ссылке, вместо того, чтобы, например, предварительно открыть его. Разрешается использовать атрибут без указания его значения:  <a href="/images/logo.png" download><img src="/images/logo.png" alt="logo"></a>  <a href="/images/logo.png" download="logo"><img src="/images/logo.png" alt="logo"></a>  <a href="files/тест.pdf" download="Тест.pdf">Скачать тест</a> |
| href | URL-адрес документа, на который указывается ссылка. |
| hreflang | Определяет язык связанного веб-документа. Используется только вместе с атрибутом href. Принимаемые значения – аббревиатура, состоящая из пары букв, обозначающих [код языка](https://html5book.ru/goto/http:/www.w3schools.com/tags/ref_language_codes.asp), например:  <a href="http://www.anysite.ru" hreflang="en">Anysite</a> |
| media | Определяет, для каких типов устройств файл будет оптимизирован. Значением может быть любой медиа-запрос. |
| ping | Отслеживает URL-адреса ресурсов, по которым переходил пользователь. |
| rel | Дополняет атрибут href информацией об отношении между текущим и связанным документом. Принимаемые значения:  alternate – ссылка на альтернативную версию документа (например, печатную форму страницы, перевод или зеркало).  author – ссылка на автора документа.  bookmark – постоянный URL-адрес, используемый для закладок.  help – ссылка на справку. |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
|  | license – ссылка на информацию об авторских правах на данный веб-документ.  next/prev – указывает связь между отдельными URL. Благодаря этой разметке Google может определить, что содержание данных страниц связано в логической последовательности.  nofollow – запрещает поисковой системе переходить по ссылкам на данной странице или по конкретной ссылке.  noreferrer – указывает, что переходе по ссылке браузер не должен посылать заголовок HTTP-запроса (Referrer), в который записывается информация о том, с какой страницы пришел посетитель сайта.  prefetch – указывает, что целевой документ должен быть кэширован, т.е. браузер в фоновом режиме загружает содержимое страницы к себе в кэш.  search – указывает, что целевой документ содержит инструмент для поиска.  tag – указывает ключевое слово для текущего документа. |
| target | Указывает на то, в каком окне должен открываться документ, к которому ведет ссылка. Принимает следующие значения:  \_self – страница загружается в текущее окно;  \_blank – страница открывается в новом окне браузера;  \_parent – страница загружается во фрейм-родитель;  \_top – страница загружается в полное окно браузера. |

HTML-изображения добавляются на веб-страницы с помощью тега <img>. Использование графики делает веб-страницы визуально привлекательнее. Изображения помогают лучше передать суть и содержание веб-документа.

Элемент <img> представляет изображение и его резервный контент, кот орый добавляется с помощью атрибута alt. Так как элемент <img> является строчным, то рекомендуется располагать его внутри блочного элемента, например, <p> или <div>.

Тег <img> имеет обязательный атрибут src, значением которого является абсолютный или относительный путь к изображению:

<img src="image.png" alt="Пример кода">.

Атрибуты тега <img> представлены в таблице 1.3.

Таблица 1.3 – Атрибуты тега <img>

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| **1** | **2** |
| alt | Атрибут alt добавляет альтернативный текст для изображения. Выводится на |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| **1** | **2** |
|  | месте появления изображения до его загрузки или при отключенной графике, а также выводится всплывающей подсказкой при наведении курсора мыши на изображение.  Синтаксис: alt="описание изображения". |
| height | Атрибут height задает высоту изображения. Может указываться в px или %. Синтаксис: height: 300px. |
| longdesc | URL расширенного описания изображения, дополняющее атрибут alt. Синтаксис:longdesc="http://www.example.com/description.txt". |
| src | Атрибут src задает путь к изображению.Синтаксис: src="flower.jpg". |
| sizes | Задаёт размер изображения в зависимости от параметров отображения. Работает только при заданном атрибуте srcset. Значением атрибута является одна или несколько строк, указанных через запятую. |
| srcset | Создаёт список источников для изображения, которые будут выбраны, исходя из разрешения экрана. Может использоваться вместе или вместо атрибута src. Значением атрибута является одна или несколько строк, разделенных запятой.  <img src="flower.jpg"  srcset="  img/flower-mobile.jpg 320w,  img/flower-wide-mobile.jpg 480w,  img/flower-tablet.jpg 768w,  img/flower-desktop.jpg 1024w,  img/flower-hires.jpg 1280w"  sizes="  (max-width: 20em) 30vw,  (max-width: 30em) 60vw,  (max-width: 40em) 90vw"  alt="Роза в моём саду"> |
| width | Атрибут width задает ширину изображения. Может указываться в px или %. Синтаксис: width: 100%. |

Для использования изображения в качестве фона используется атрибут background с адресом фонового рисунка.

<body background="site/tsvet-fona-html.png">

Чтобы пометить или подписать фото на странице используется тег <figure> (от английского figure – рисунок). Этот тег указывает на то, что внутри него помещается контент типа иллюстраций, фотографий, диаграмм и т.п. Этот элемент поддерживает только [глобальные атрибуты](https://developer.mozilla.org/en-US/docs/HTML/Global_attributes). Тег <figcaption> (заголовок рисунка) позволяет добавить подпись к изображению.

<figure>

<img src="sea.jpg" alt="вид на море" style="width:60%;">

<figcaption>Вид на море

</figcaption>

</figure>

Элемент <picture> представляет собой контейнер для хранения нескольких элементов [<source>](https://webref.ru/html/source), которые поддерживают элемент [<img>](https://webref.ru/html/img). Это позволяет указывать разные изображения с учётом размера экрана, плотности пикселей, формата изображения и других параметров.

Внутри <picture> содержится ноль или несколько элементов [<source>](https://webref.ru/html/source), которые идут перед одним элементом [<img>](https://webref.ru/html/img):

<picture>

<source srcset="smaller.jpg" media="(max-width: 768px)">

<source srcset="default.jpg">

<img srcset="default.jpg" alt="My default image">

</picture>

Загружать можно альбомную (landscape) или книжную (portrait) версию изображения, в зависимости от ориентации устройства, к тому же можно одновременно проверять размеры в этих выражениях. Например:

<picture>

<source srcset="smaller\_landscape.jpg" media="(max-width: 40em) and (orientation: landscape)">

<source srcset="smaller\_portrait.jpg" media="(max-width: 40em) and (orientation: portrait)">

<source srcset="default\_landscape.jpg" media="(min-width: 40em) and (orientation: landscape)">

<source srcset="default\_portrait.jpg" media="(min-width: 40em) and (orientation: portrait)">

<img srcset="default\_landscape.jpg" alt="My default image">

</picture>

Этот код загружает уменьшенную альбомную (landscape) версию изображения для устройств с малым экраном и соответствующей ориентацией. И увеличенную версию того же изображения для устройств с большим экраном.

**5. Порядок выполнения работы**

1. Создайте простейшую HTML-страницу с использованием cемантических элементов HTML5.

2. Создать информационный сайт-визитку согласно варианту. На странице должны присутствовать:

– внутренние и внешние ссылки на другие страницы;

– ссылки на ваши социальные сети;

– изображения.

Варианты:

1. Библиотека

2. Книжный магазин

3. Аптека

4. СТО

5. Сервисный центр Apple

6. Ресторан

7. Красная книга

8. Музей

9. Школа

10. Курсы иностранных языков

11. Сайт знакомств

12. Часовая мастерская

13. Цветочный магазин

14. Клуб

15. Аэропорт

16. Муж на час

17. Доставка еды

18. Туристическое агентство

19. Магазин техники

20. Тренажерный зал

21. Рекламное агентсво

22. Афиша

23. Агентство недвижимости

24. Оптика

25. Поликлиника

26. Автосалон

27. Военкомат

28. Курсы программирования

29. Зоопарк

30. Магазин мебели

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1.Опишите структуру HTML-документа.

2.Перечислите семантические элементы HTML5.

3. С помощью какого тега создается ссылка? Как создать ссылку на адрес электронной почты?

4.Дайте характеристику тегов, используемых в лабораторной работе.

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 2**

## **Тема работы: «Создание списков и таблиц на HTML-страниц. Использование таблиц для форматирования контента на странице (физические и логические элементы таблиц)»**

**1. Цель работы**

Формирование умений создания списков и таблиц различных видов и модификаций на HTML-странице.

**2. Задание**

Добавить вложенные и многоуровневые списки и таблицы различных модификаций в веб-документ.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

HTML-списки используются для группировки связанных между собой фрагментов информации. Существует три вида списков:

1) маркированный список – <ul> – каждый элемент списка <li> отмечается маркером;

2) нумерованный список – <ol> – каждый элемент списка <li> отмечается цифрой;

3) список определений – <dl> – состоит из пар термин <dt> – <dd> определение.

Маркированный список представляет собой неупорядоченный список (от англ. Unordered List). Создаётся с помощью парного тега <ul></ul>. В качестве маркера элемента списка выступает метка.

По умолчанию маркер отображается в виде черного кружочка. Это можно изменить с помощью параметра type:

– type="disc" – закрашенный кружочек.

– type="circle" – пустой кружочек.

– type="square" – закрашенный квадратик.

Браузеры по умолчанию добавляют следующее форматирование блоку списка:ul {padding-left: 40px; margin-top: 1em; margin-bottom: 1em;}.

Каждый элемент списка создаётся с помощью парного тега <li></li> (от англ. List Item):

<ul type="square">

<li>Microsoft</li>

<li>Google</li>

<li>Apple</li>

<li>IBM</li>

</ul>

Еще одну интересную возможность по настройке списков предоставляет стиль list-style-image. Он задает изображение, которое будет отображаться рядом с элементом списка:

<ul style="list-style-image:url(image.png);">

<li>iPhone 6S</li>

<li>Galaxy S7</li>

<li>Nexus 5X</li>

<li>Lumia 950</li>

</ul>

Нумерованный список создаётся с помощью парного тега <ol></ol>. Каждый пункт списка также создаётся с помощью элемента <li>. Браузер нумерует элементы по порядку автоматически и если удалить один или несколько элементов такого списка, то остальные номера будут автоматически пересчитаны.

Блок списка также имеет стили браузера по умолчанию: ol {padding-left: 40px; margin-top: 1em; margin-bottom: 1em;}.

Для тега <li> доступен атрибут value, который позволяет изменить номер по умолчанию для выбранного элемента списка. Например, если для первого пункта списка задать <li value="10">, то остальная нумерация будет пересчитана относительно нового значения.

Атрибуты нумерованного списка представлены в таблице 2.1.

Таблица 2.1 – Атрибуты тега <ol>

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| reversed | Атрибут reversed задает отображение списка в обратном порядке (например, 9, 8, 7…). |
| start | Атрибут start задает начальное значение, от которого пойдет отсчет нумерации, например, конструкция <ol start="10"> первому пункту присвоит порядковый номер «10». Также можно одновременно задавать тип нумерации, например, <ol type="I" start="10">. |
| type | Атрибут type задает вид маркера для использования в списке (в виде букв или цифр). Принимаемые значения: |
|  | 1 — значение по умолчанию, десятичная нумерация. A — нумерация списка в алфавитном порядке, заглавные буквы (A, B, C, D). a — нумерация списка в алфавитном порядке, строчные буквы (a, b, c, d). I — нумерация римскими заглавными цифрами (I, II, III, IV). i — нумерация римскими строчными цифрами (i, ii, iii, iv). |

<ol>

<li>Microsoft</li>

<li>Google</li>

<li>Apple</li>

<li>IBM</li>

</ol>

Списки определений создаются с помощью тега <dl></dl>. Для добавления термина применяется тег <dt></dt>, а для вставки определения – тег <dd></dd>.

Блок списка определений имеет следующие стили браузера по умолчанию: dl {margin-top: 1em; margin-bottom: 1em;}.

<dl>

<dt>Термин 1</dt>

<dd>Определение термина 1</dd>

<dt>Термин 2</dt>

<dd>Определение термина 2</dd>

</dl>

Зачастую возможностей простых списков не хватает, например, при создании оглавления никак не обойтись без вложенных пунктов. Разметка для вложенного списка будет следующей:

<ul>

<li>Пункт 1.</li>

<li>Пункт 2.

<ul>

<li>Подпункт 2.1.</li>

<li>Подпункт 2.2.

<ul>

<li>Подпункт 2.2.1.</li>

<li>Подпункт 2.2.2.</li>

</ul>

</li>

<li>Подпункт 2.3.</li>

</ul>

</li>

<li>Пункт 3.</li>

</ul>

Многоуровневый список используется для отображения элементов списка на разных уровнях с различными отступами. Разметка для многоуровневого нумерованного списка будет следующей:

<ol>

<li>пункт</li> <!-1.->

<li>пункт

<ol>

<li>пункт</li> <!-2.1.->

<li>пункт</li> <!-2.2.->

<li>пункт

<ol>

<li>пункт</li> <!-2.3.1.->

<li>пункт</li> <!-2.3.2.->

<li>пункт</li> <!-2.3.3.->

</ol>

</li> <!-2.3.->

<li>пункт</li> <!-2.4.->

</ol>

</li> <!-2.->

<li>пункт</li> <!-3.->

<li>пункт</li> <!-4.->

</ol>

Такая разметка по умолчанию создаст для каждого вложенного списка новую нумерацию, начинающуюся с единицы. Чтобы сделать вложенную нумерацию, нужно использовать следующие свойства:

counter-reset сбрасывает один или несколько счётчиков, задавая значение для сброса;

counter-increment задаёт значение приращения счётчика, т.е. с каким шагом будет нумероваться каждый последующий пункт;

content – генерируемое содержимое, в данном случае отвечает за вывод номера перед каждым пунктом списка.

<ol style=”

/\* убираем стандартную нумерацию \*/

list-style: none;”></ol>

HTML5 позволяет сделать контент вашего сайта полностью редактируемым, при этом нет необходимости вставлять скрытые поля для ввода текста. Все, что необходимо – это добавить атрибут contenteditable="true" (или без кавычек, как мы узнали из предыдущего пункта) к тому элементу, который вы хотите сделать доступным для редактирования. После этого, пользователь сможет редактировать его содержимое непосредственно со страницы.

‹h1›To-Do List‹/h1›

‹ul contenteditable=true›

‹li›Watch TV‹/li›

‹li›Listen to music‹/li›

‹li›Play videogames‹/li›

‹/ul›

HTML-таблицы упорядочивают и выводят на экран данные с помощью строк или столбцов. Таблицы состоят из ячеек, образующихся при пересечении строк и столбцов. Ячейки таблиц могут содержать любые HTML-элементы, такие как заголовки, списки, текст, изображения, элементы форм, а также другие таблицы. Каждой таблице можно добавить связанный с ней заголовок, расположив его перед таблицей или после неё.

Таблицы больше не используются для вёрстки веб-страниц и компоновки отдельных элементов, потому что такой приём не обеспечивает гибкость структуры и адаптивность сайта, существенно увеличивая HTML-разметку.

Таблица создаётся при помощи парного тега <table></table>. Данный тег является контейнером для элементов таблицы и все элементы должны находиться внутри него.

Строки или ряды таблицы создаются с помощью тега <tr>. Количество горизонтальных строк таблицы определяется количеством парных тегов <tr></tr>.

Элемент <th> создаёт заголовок столбца — специальную ячейку, текст в которой выделяется полужирным. Количество ячеек заголовка определяется количеством пар тегов <th></th>. Для элемента доступны атрибуты colspan, rowspan, headers.

Элемент <td> создаёт ячейки таблицы, внутрь которых помещаются данные таблицы. Парные теги <td></td>, расположенные в одном ряду, определяют количество ячеек в строке таблицы. Количество пар ячеек <td> должно быть равно количеству пар ячеек <th>. Для элемента доступны атрибуты colspan, rowspan, headers.

Ширина таблицы по умолчанию равна ширине её внутреннего содержимого. Чтобы установить ширину, нужно задать значение для свойства width.

Например, с помощью данной разметки можно создать таблицу, состоящую из двух столбцов и двух строк:

<table>

<tr><!--ряд с ячейками заголовков-->

<th>текст заголовка</th>

<th>текст заголовка</th>

</tr>

<tr><!--ряд с ячейками тела таблицы-->

<td>данные</td>

<td>данные</td>

</tr>

</table>

По умолчанию таблица и ячейки не имеют видимых границ. Границы задаются с помощью свойства border.

Промежутки между ячейками таблицы убираются с помощью свойства:

<table style=” border-collapse: collapse; border: 1px solid grey;”></table>

Атрибуты элементов таблицы приведены в таблице 2.2.

Таблица 2.2 – Атрибуты элементов таблицы

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| colspan | Количество ячеек в строке для объединения по горизонтали.  <td colspan="3">  Возможные значения: число от 1 до 999. |
| headers | Задает список ячеек заголовка, содержащих информацию о заголовке текущей ячейки данных. Предназначен для речевых браузеров.  <th id="идентификатор">...</th>  <th headers="идентификатор">...</th>  Принимаемые значения: список имен ячеек, разделенных пробелами; эти имена должны быть присвоены ячейкам через их атрибут **id.** |
| rowspan | Количество ячеек в столбце для объединения по вертикали.  <td rowspan="2">  Возможные значения: число от 1 до 999. |
| Span | Количество колонок, объединяемых для задания единого стиля, по умолчанию равно 1.  <col span="2">  Принимаемые значения: любое целое положительное число. |

Атрибут headers позволяет связать ячейки таблицы с заголовками. Этот атрибут предназначен для повышения доступности таблицы пользователям речевых браузеров, в обычных браузерах результат применения атрибута headers не заметен.

Для связывания ячеек между собой одной ячейке в теге <td> или <th> задается атрибут id, а второй ячейке — атрибут headers со значением, совпадающим со значением id.

<table style="width:100%">

<tr>

<th id="name">Name</th>

<th id="email">Email</th>

<th id="phone">Phone</th>

<th id="addr">Address</th>

</tr>

<tr>

<td headers="name">John Doe</td>

<td headers="email">someone@example.com</td>

<td headers="phone">+45342323</td>

<td headers="addr">Rosevn 56,4300 Sandnes,Norway</td>

</tr>

</table>

Элемент <caption> создает подпись таблицы. Добавляется непосредственно после тега <table>, вне строки или ячейки.

<table>

<caption>Перечень продуктов</caption>

<tr>

<th>№ п/п</th>

<th>Наименование товара</th>

<th>Ед. изм.</th>

<th>Количество</th>

<th>Цена за ед. изм., руб.</th>

<th>Стоимость, руб.</th>

</tr>

<tr>

<td>1.</td>

<td>Томаты свежие</td> <td>кг</td> <td>15,20</td> <td>69,00</td> <td>1048,80</td>

</tr>

<tr>

<td>2.</td>

<td>Огурцы свежие</td> <td>кг</td> <td>2,50</td> <td>48,00</td> <td>120,00</td>

</tr>

<tr>

<td colspan="5" style="text-align:right">ИТОГО:</td><td>1168,80</td>

</tr>

</table>

Элемент <thead> создает группу заголовков для строк таблицы с целью задания единого оформления. Используется в сочетании с элементами <tbody> и <tfoot> для указания каждой части таблицы.

Элемент должен быть использован в следующем порядке: как дочерний элемент <table>, после <caption> и <colgroup>, и перед <tbody>, <tfoot> и <tr> элементами. В пределах одной таблицы можно использовать один раз.

Элемент <tbody> группирует основное содержимое таблицы. Используется в сочетании с элементами <thead> и <tfoot>.

Элемент <tfoot> создает группу строк для представления информации о суммах или итогах, расположенную в нижней части таблицы. Используется в таблице один раз. Располагается после тега <thead>, перед тегами <tbody> и <tr>.

<table>

<thead>

<tr>

<th>№ п/п</th>

<th>Наименование товара</th>

<th>Ед. изм.</th>

<th>Количество</th>

<th>Цена за ед. изм., руб.</th>

<th>Стоимость, руб.</th>

</tr>

</thead>

<tfoot>

<tr>

<td colspan="5" style="text-align:right">ИТОГО:</td><td>1168,80</td>

</tr>

</tfoot>

<tbody>

<tr>

<td>1.</td>

<td>Томаты свежие</td> <td>кг</td> <td>15,20</td> <td>69,00</td> <td>1048,80</td>

</tr>

<tr>

<td>2.</td>

<td>Огурцы свежие</td> <td>кг</td> <td>2,50</td> <td>48,00</td> <td>120,00</td>

</tr>

</tbody>

</table>

Атрибуты colspan и rowspan объединяют ячейки таблицы. Атрибут colspan задает количество ячеек, объединенных по горизонтали, а rowspan – по вертикали.

<table>

<tr>

<th>№ п/п</th>

<th>Наименование товара</th>

<th>Ед. изм.</th>

<th>Количество</th>

<th>Цена за ед. изм., руб.</th>

<th>Стоимость, руб.</th>

</tr>

<tr>

<td>1.</td>

<td>Томаты свежие</td> <td>кг</td> <td>15,20</td> <td>69,00</td> <td>1048,80</td>

</tr>

<tr>

<td>2.</td>

<td>Огурцы свежие</td> <td>кг</td> <td>2,50</td> <td>48,00</td> <td>120,00</td>

</tr>

<tr>

<td colspan="5" style="text-align:right">ИТОГО:</td> <td>1168,80</td> <!-- Задаем количество ячеек по горизонтали для объединения-->

</tr>

</table>

Элемент <colgroup> создает структурную группу столбцов, выделяя логически однородные ячейки. Группирует один или более столбцов для единого форматирования, позволяя применить стили к столбцам вместо того, чтобы повторять стили для каждой ячейки и для каждой строки. Добавляется непосредственно после тегов <table> и <caption>.

Элемент <col> формирует группы столбцов, которые делят таблицу на разделы, не относящиеся к общей структуре, т.е. не содержащие информацию одного типа. Позволяет задавать свойства столбцов для каждого столбца в пределах элемента <colgroup>. С помощью атрибута style можно изменить основной цвет фона ячеек. Для элемента <col> доступен атрибут span, задающий количество столбцов для объединения.

<table>

<colgroup>

<col span="2" style="background:Khaki"><!-- С помощью этой конструкции задаем цвет фона для первых двух столбцов таблицы-->

<col style="background-color:LightCyan"><!-- Задаем цвет фона для следующего (одного) столбца таблицы-->

</colgroup>

<tr>

<th>№ п/п</th>

<th>Наименование</th>

<th>Цена, руб.</th>

</tr>

<tr>

<td>1</td>

<td>Карандаш цветной</td>

<td>20,00</td>

</tr>

<tr>

<td>2</td>

<td>Линейка 20 см</td>

<td>30,00</td>

</tr>

</table>

**5. Порядок выполнения работы**

1. Добавить на страницы, созданные в 1-ой лабораторной работе вложенные и многоуровневые списки в соответствии с темой работы.

2. Создайте список, в котором вместо маркера будет использовано изображение.

3. Создать таблицу и указать её параметры:

– границы;

– расстояние между ячейками;

– ширина.

4. Сгруппировать разделы таблицы (заголовок, основное представление, подведение итогов).

5. Используйте объединение ячеек по горизонтали и по вертикали.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Перечислите три вида списков. Какими тегами они создаются?

2. Для чего предназначен атрибут type при создании списков? В каких видах списков данный атрибут может использоваться?

3. С помощью какого тега создается заголовок таблицы?

4. С помощью каких атрибутов можно объединять ячейки по горизонтали и по вертикали?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 3**

## **Тема работы: «Создание форм средствами HTML. Сравнение элементов формы, паттерны ввода»**

**1. Цель работы**

Формирование умений создания форм на HTML-странице.

**2. Задание**

Создать форму, содержащую различные элементы.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

HTML-формы являются элементами управления, которые применяются для сбора информации от посетителей веб-сайта.

Пример формы-анкеты посетителя ресторана представлен на рисунке 3.1.

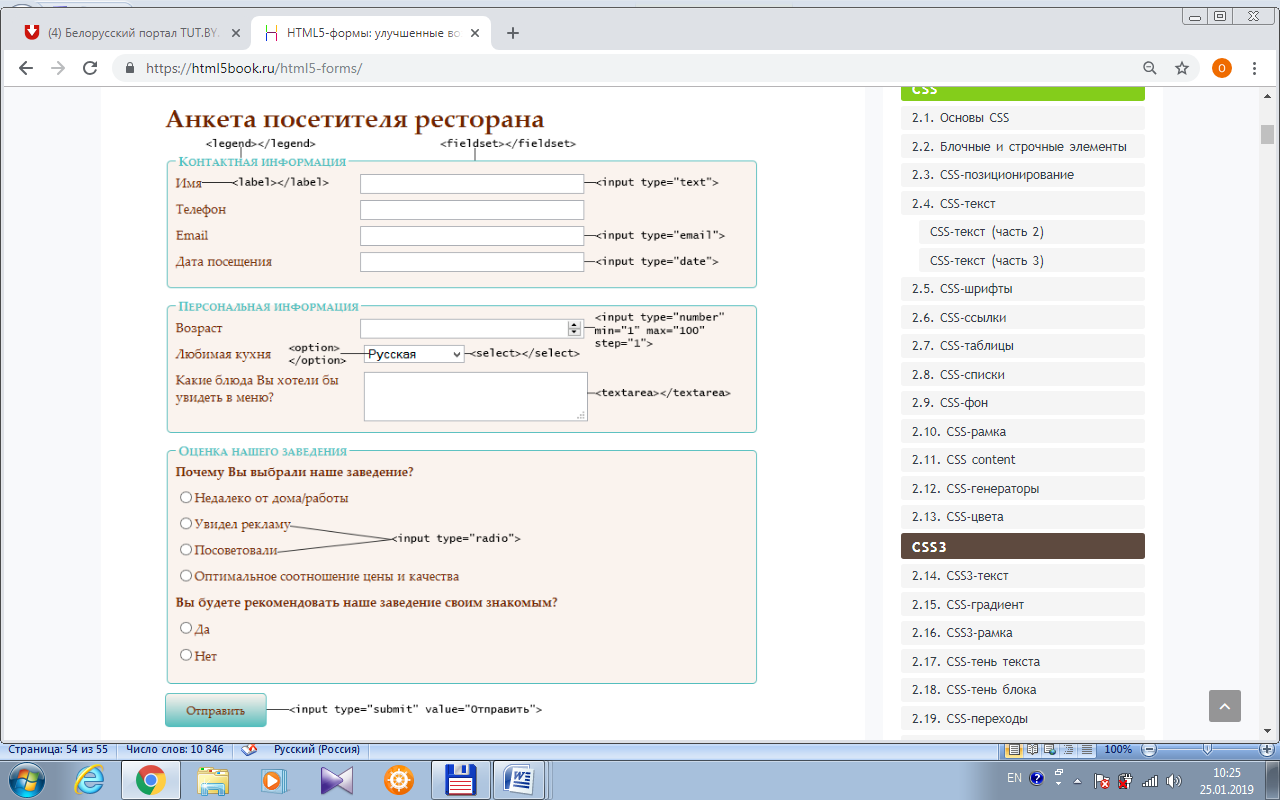


Рисунок 3.1 – Анкета посетителя ресторана

Основу любой формы составляет элемент <form>...</form>. Он не предусматривает ввод данных, так как является контейнером, удерживая вместе все элементы управления формы – поля. Атрибуты этого элемента содержат информацию, общую для всех полей формы, поэтому в одну форму нужно включать поля, объединенные логически.

Атрибуты тега <form> представлены в таблице 3.1.

Таблица 3.1 – Атрибуты тега <form>

|  |  |
| --- | --- |
| **Атрибут** | **Значение / описание** |
| action | Обязательный атрибут, который указывает url обработчика формы на сервере, которому передаются данные. Представляет из себя файл (например, action.php), в котором описано, что нужно делать с данными формы. |
| autocomplete | Отвечает за запоминание введенных в текстовое поле значений и автоподстановку их при последующем вводе:  on — означает, что поле не защищено, и его значение можно сохранять и извлекать,  off — отключает автозаполнение для полей форм. |
| enctype | Используется для указания MIME-типа данных, отправляемых вместе с формой, например, enctype="multipart/form-data". Указывается только в случае method="post".  application/x-www-form-urlencoded — тип содержимого по умолчанию, указывает на то, что передаваемые данные представляют список URL-кодированных переменных формы. Символы пробела (ASCII 32) будут закодированы как +, а специальный символ, например, такой как !будет закодирован шестнадцатиричной форме как %21.  multipart/form-data — используется для отправки форм, содержащих файлы, не-ASCII данные и бинарные данные, состоит из нескольких частей, каждая из которых представляет содержимое отдельного элемента формы.  text/plain — указывает на то, что передается обычный (не html) текст. |
| method | Задает способ передачи данных формы.  Метод get передает данные на сервер через адресную строку браузера.  Метод post применяется для пересылки данных больших объемов, а также конфиденциальной информации и паролей. Данные, отправляемые с помощью этого метода, не видны в заголовке URL, так как они содержатся в теле сообщения.  <form action="action.php" enctype="multipart/form-data" method="post"></form> |
| name | Задает имя формы, которое будет использоваться для доступа к элементам формы через сценарии, например, name="opros". |

Элемент <fieldset>...</fieldset> предназначен для группировки элементов, связанных друг с другом, разделяя, таким образом, форму на логические фрагменты.

Каждой группе элементов можно присвоить название с помощью элемента <legend>, который идет сразу за тегом <fieldset>. Название группы проявляется слева в верхней границе <fieldset>. Например, если в элементе <fieldset> хранится контактная информация:

<form>

<fieldset>

<legend>Контактная информация</legend>

<p><label for="name">Имя <em>\*</em></label><input type="text" id="name"></p>

<p><label for="email">E-mail</label><input type="email" id="email"></p>

</fieldset>

<p><input type="submit" value="Отправить"></p>

</form>

Группировка элементов формы с помощью <fieldset> представлена на рисунке 3.2.

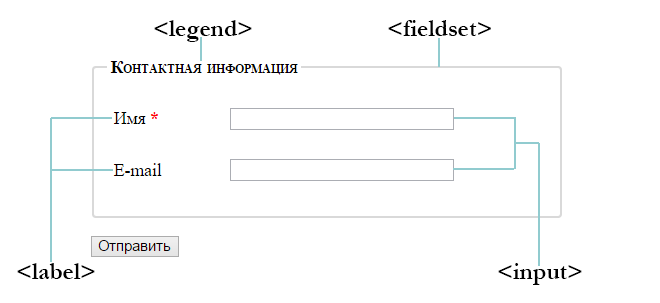


Рисунок 3.2 – Группировка элементов формы с помощью <fieldset>

Атрибуты тега <fieldset> представлены в таблице 3.2.

Таблица 3.2 – Атрибуты тега <fieldset>

|  |  |
| --- | --- |
| **Атрибут** | **Значение / описание** |
| disabled | Если атрибут присутствует, то группа связанных элементов формы, находящихся внутри контейнера <fieldset>, отключены для заполнения и редактирования. Используется для ограничения доступа к некоторым полям формы, содержащих ранее введенные данные. Атрибут используется без указания значения — <fieldset disabled>. |
| form | Значение атрибута должно быть равно атрибуту id элемента <form> в этом же документе. Указывает на одну или несколько форм, к которым принадлежит данная группа элементов. На данный момент атрибут не поддерживается ни одним браузером. |
| name | Определяет имя, которое будет использоваться для ссылки на элементы в JavaScript, или для ссылки на данные формы после заполнения и отправки формы. Является аналогом атрибута id. |

Элемент <input> создает большинство полей формы. Атрибуты элемента отличаются в зависимости от типа поля, для создания которого используется этот элемент.

Атрибуты тега <input> представлены в таблице 3.3.

Таблица 3.3 – Атрибуты тега <input>

|  |  |
| --- | --- |
| **Атрибут** | **Значение / описание** |
| **1** | **2** |
| accept | Определяет тип файла, разрешенных для отправки на сервер. Указывается только для <input type="file">. Возможные значения: file\_extension — разрешает загрузку файлов с указанным расширением, например, accept=".gif", accept=".pdf", accept=".doc" audio/\* — разрешает загрузку аудиофайлов video/\* — разрешает загрузку видеофайлов image/\* — разрешает загрузку изображений media\_type — указывает на медиа-тип загружаемых файлов. |
| alt | Определяет альтернативный текст для изображений, указывается только для <input type="image">. |
| autocomplete | Отвечает за запоминание введенных в текстовое поле значений и автоподстановку их при последующем вводе: on — означает, что поле не защищено, и его значение можно сохранять и извлекать, off — отключает автозаполнение для полей форм. |
| autofocus | Позволяет сделать так, чтобы в загружаемой форме то или иное поле ввода уже имело фокус (было выбрано), являясь готовым к вводу значения. |
| checked | Атрибут проверяет, установлен ли флажок по умолчанию при загрузке страницы для полей типа type="checkbox" и type="radio". |
| disabled | Отключает возможность редактирования и копирования содержимого поля. |
| form | Значение атрибута должно быть равно атрибуту id элемента <form> в этом же документе. Определяет одну или несколько форм, которым принадлежит данное поле формы. |
| formaction | Задает url файла, который будет обрабатывать введенные в поля данные при отправке формы. Задается только для полей типа type="submit" и type="image". Атрибут переопределяет значение атрибута action самой формы. |
| formenctype | Определяет, как будут кодироваться данные полей формы при отправке на сервер. Переопределяет значение атрибута enctypeформы. Задается только для полей типа type="submit" и type="image". Варианты: application/­x-www-form-urlencoded — значение по умолчанию. Все символы кодируются перед отправкой (пробелы заменяются на символ +, специальные символы преобразуются в значения ASCII HEX) multipart/form-data — символы не кодируются text/plain — пробелы заменяются на символ +, а специальные символы не кодируются. |
| formmethod | Атрибут определяет метод, который браузер будет использовать для отправки данных формы на сервер. Задается только для полей типа type="submit" и type="image". Переопределяет значение атрибута method формы. Варианты: |

Продолжение таблицы 3.3

|  |  |
| --- | --- |
| **1** | **2** |
|  | get — значение по умолчанию. Данные из формы (пара имя/значение) добавляются в url-адрес и отправляются на сервер: URL?имя=значение&имя=значение post — данные формы отправляются в виде http-запроса. |
| formnovalidate | Определяет, что данные полей формы не должны проверяться при отправке формы. Переопределяет значение атрибута novalidateформы. Можно использовать без указания значения атрибута. |
| formtarget | Определяет, где выводить ответ, полученный после отправки формы. Задается только для полей типа type="submit" и type="image". Переопределяет значение атрибута targetформы. \_blank — загружает ответ в новое окно/вкладку \_self — загружает ответ в то же окно (значение по умолчанию) \_parent – загружает ответ в родительский фрейм \_top – загружает ответ во весь экран framename – загружает ответ во фрейм с указанным именем. |
| height | Значение атрибута содержит количество пикселей без указания единицы измерения. Устанавливает высоту поля формы типа type="image", например, <input type="image" src="img\_submit.gif" height="50">. Рекомендуется одновременно устанавливать как высоту, так и ширину поля. |
| list | Является ссылкой на элемент <datalist>, содержит его id.Позволяет предоставить пользователю несколько вариантов на выбор, когда он начинает вводить значение в соответствующем поле. |
| max | Позволяет ограничить допустимый ввод числовых данных максимальным значением, значение атрибута может содержать целое или дробное число. Рекомендуется использовать этот атрибут вместе с атрибутом min. Работает со следующими типами полей: number, range, date, datetime, datetime-local, month, time и week. |
| maxlength | Атрибут задает максимальное количество символов, вводимых в поле. Значение по умолчанию 524288 символов. |
| min | Позволяет ограничить допустимый ввод числовых данных минимальным значением. |
| multiple | Позволяет пользователю ввести несколько значений атрибутов, разделяя их запятой. Применяется в отношении файлов и адресов электронной почты. Указывается без значения атрибута. |
| name | Определяет имя, которое будет использоваться для доступа к элементу <form>, к примеру, в таблицах стилей css. Является аналогом атрибута id. |
| pattern | Позволяет определять с помощью регулярного выражениясинтаксис данных, ввод которых должен быть разрешен в определенном поле. Например, pattern="[a-z]{3}-[0-9]{3}" — квадратные скобки устанавливают диапазон допустимых символов, в данном случае — любые строчные буквы, число в фигурных скобках указывает, что нужны три строчные буквы, после которых следует тире, далее — три цифры в диапазоне от 0 до 9. |
| placeholder | Содержит текст, который отображается в поле ввода до заполнения (чаще всего это подсказка). |
| readonly | Не позволяет пользователю изменять значения элементов формы, выделение и копирование текста при этом доступно. Указывается без значения атрибута. |

Продолжение таблицы 3.3

|  |  |
| --- | --- |
| **1** | **2** |
| required | Выводит сообщение о том, что данное поле является обязательным для заполнения. Если пользователь попытается отправить форму, не введя в это поле требуемое значение, то на экране отобразится предупреждающее сообщение. Указывается без значения атрибута. |
| size | Задает видимую ширину поля в символах. Значение по умолчанию — 20. Работает со следующими типами полей: text, search, tel, url, email и password. |
| src | Задает url изображения, используемого в качестве кнопки отправки данных формы. Указывается только для поля <input type="image">. |
| step | Используется для элементов, предполагающих ввод числовых значений, указывает величину увеличения или уменьшения значений в процессе регулировки диапазона (шаг). |
| type | button — создает кнопку. |
| checkbox — превращает поле ввода во флажок, который можно установить или очистить, например, У меня есть автомобиль |
| color — генерирует палитры цветов в поддерживающих браузерах, давая пользователям возможность выбирать значения цветов в шестнадцатеричном формате. |
| date — позволяет вводить дату в формате дд.мм.гггг. |
| datetime-local — позволяет вводить дату и время, разделенные прописной английской буквой Т по шаблону дд.мм.гггг чч:мм. |
| email — браузеры, поддерживающие данный атрибут, будут ожидать, что пользователь введет данные, соответствующие синтаксису адресов электронной почты. |
| file — позволяет загружать файлы с компьютера пользователя. Выберите файл: |
| hidden — скрывает элемент управления, который не отображается браузером и не дает пользователю изменять значения по умолчанию. |
| image — создает кнопку, позволяя вместо текста на кнопке вставить изображение.  Submit |
| month — позволяет пользователю вводить год и номер месяца по шаблону гггг-мм. |
| number — предназначено для ввода целочисленных значений. Его атрибуты min, max и step задают верхний, нижний пределы и шаг между значениями соответственно. Эти атрибуты предполагаются у всех элементов, имеющих численные показатели. Их значения по умолчанию зависят от типа элемента.  Укажите количество (от 1 до 5): |

Продолжение таблицы 3.3

|  |  |
| --- | --- |
| **1** | **2** |
|  | password — создает текстовые поля в форме, при этом вводимые пользователем символы заменяются на звездочки, маркеры, либо другие, установленные браузером значки.  Введите пароль: |
| radio — создает переключатель — элемент управления в виде небольшого кружка, который можно включить или выключить. Вегетарианец: |
| range — позволит создать такой элемент интерфейса, как ползунок, min / max — позволят установить диапазон выбора |
| reset — создает кнопку, которая очищает поля формы от введенных пользователем данных. |
| search — обозначает поле поиска, по умолчанию поле ввода имеет прямоугольную форму. Поиск: |
| submit — создает стандартную кнопку, активизируемую щелчком мыши. Кнопка собирает информацию с формы и отправляет ее для обработки. |
| text — создает текстовые поля в форме, выводя однострочное текстовое поле для ввода текста. |
| time — позволяет вводить время в 24-часовом формате по шаблону чч:мм. В поддерживающих браузерах оно отображается как элемент управления в виде числового поля ввода со значением, изменяемым с помощью мыши, и допускает ввод только значений времени. |
| url — поле предназначено для указания URL-адресов. |
| week — соответствующий инструмент-указатель позволяет пользователю выбрать одну неделю в году, после чего обеспечит ввод данных в формате нн-гггг. В зависимости от года число недель может быть 52 или 53. |
| value | Определяет текст, отображаемый на кнопке, в поле или связанный текст. Не указывается для полей типа file. |
| width | Значение атрибута содержит количество пикселей. Позволяет задать ширину полей формы. |

Элемент <textarea>...</textarea> используется вместо элемента <input type="text">, когда нужно создать большие текстовые поля. Текст, отображаемый как исходное значение, помещается внутрь тега. Размеры поля устанавливаются при помощи атрибутов cols – размеры по горизонтали, rows – размеры по вертикали. Высоту поля можно задать свойством height. Все размеры считаются исходя из размера одного символа моноширинного шрифта.

Атрибуты тега <textarea> представлены в таблице 3.4.

Таблица 3.4 – Атрибуты тега <textarea>

|  |  |
| --- | --- |
| **Атрибут** | **Значение / описание** |
| autofocus | Устанавливает фокус на нужном начальном текстовом поле автоматически. |
| cols | Устанавливает ширину через количество символов. Если пользователь вводит больше текста, появляется полоса прокрутки. |
| disabled | Отключает возможность редактирования и копирования содержимого поля. |
| form | Значение атрибута должно быть равно значению атрибута idэлемента <form> в этом же документе. Определяет одну или несколько форм, которым принадлежит данное текстовое поле. |
| maxlength | Значение атрибута задает максимальное число символов для ввода в поле. |
| name | Задает имя текстового поля. |
| placeholder | Определяет короткую текстовую подсказку, которая описывает ожидаемое вводимое значение. |
| readonly | Отключает возможность редактирования содержимого поля. |
| required | Выводит сообщение о том, что данное поле является обязательным для заполнения. |
| rows | Указывает число, которое означает, сколько строк должно отображаться в текстовой области. |
| wrap | Определяет, должен ли текст сохранять переносы строк при отправке формы. Значение hard сохраняет перенос, а значение soft не сохраняет. Если используется значение hard, то должно указываться значение атрибута cols. |

Поля со списками дают возможность расположить большое количество пунктов компактно. Раскрывающиеся списки создаются при помощи элемента <select>...</select>. Они позволяют выбрать одно или несколько значений из предложенного множества. По умолчанию в поле списка отображается его первый элемент.

Для добавления в список пунктов используются элементы <option>...</option>, которые располагаются внутри <select>.

Для систематизации списков применяется элемент <optgroup>...</optgroup>, который создает заголовки в списках.

Для списков возможно изменить размер шрифта, тип шрифта, цвет и другие свойства текста, а также добавить границы, цвет фона и фоновое изображение.

Для больших и сложных списков есть возможность добавить подзаголовки — тег optgroup с атрибутом label (надпись):

<select name="book" size="1">

<optgroup label="Английские">

<option value="asp">ASP</option>

<option value="js">JavaScript</option>

<option value="php">PHP</option>

<option value="html" selected="selected">HTML</option>

</optgroup>

<optgroup label="Русские">

<option value="asp\_rus">ASP по-русски</option>

<option value="js\_rus">JavaScript по-русски</option>

</optgroup>

</select>

Для предоставления возможности выбора нескольких пунктов одновременно необходимо добавить атрибут multiple. Но в таком случае и атрибут size следует установить в значение, большее, чем 1.

<select name="book" size="4" multiple="multiple">

Атрибуты тега <select> представлены в таблице 3.5.

Таблица 3.5 – Атрибуты тега <select>

|  |  |
| --- | --- |
| **Атрибут** | **Значение / описание** |
| autofocus | Устанавливает автоматический фокус на элементе при загрузке страницы. |
| disabled | Отключает раскрывающийся список. |
| form | Определяет форму, которой принадлежит данный список. В качестве значения атрибута указывается идентификатор формы. |
| multiple | Дает возможность выбора одного или нескольких пунктов, для этого при выборе нужно нажать и удерживать нажатой клавишу Ctrl. |
| name | Определяет имя для выпадающего списка. Значение атрибута содержит название, отражающее тематику списка. |
| required | Выводит сообщение о том, что пользователь должен выбрать значение из раскрывающегося списка перед отправкой формы. |
| size | Задает количество одновременно видимых на экране элементов списка. Если количество элементов списка превышает установленное количество, появляется полоса прокрутки. Значение атрибута задается целым положительным числом. |

Атрибуты тега <option> представлены в таблице 3.6.

Таблица 3.6 – Атрибуты тега < option >

|  |  |
| --- | --- |
| **Атрибут** | **Значение / описание** |
| disabled | Делает недоступным для выбора элемент списка. |
| label | Задает укороченную версию для элемента, которая будет отражаться в выпадающем списке. Значение атрибута содержит текст, описывающий соответствующий пункт выпадающего списка. |
| selected | Отображает выбранный элемент списка по умолчанию при загрузке страницы браузером. |
| value | Указывает значение, которое будет отправлено на сервер при отправке формы. |

Атрибуты тега <optgroup> представлены в таблице 3.7.

Таблица 3.7 – Атрибуты тега < optgroup >

|  |  |
| --- | --- |
| **Атрибут** | **Значение / описание** |
| **1** | **2** |
| disabled | Отключает данную группу элементов списка для выбора. |

Продолжение таблицы 3.7

|  |  |
| --- | --- |
| **1** | **2** |
| label | Задает заголовок для группы элементов выпадающего списка. Значение атрибута содержит текст, недоступный для выбора, который будет располагаться над соответствующими пунктами списка. Текст выделяется в браузере жирным начертанием. |

HTML-элемент <datalist> содержит набор опций (<option>), доступных для выбора. Выбранное значение будет установлено для элемента <input>, с атрибутом list:

<label for="myBrowser">Choose a browser from this list:</label>

<input list="browsers" id="myBrowser" name="myBrowser" />

<datalist id="browsers">

<option value="Chrome">

<option value="Firefox">

<option value="Internet Explorer">

<option value="Opera">

<option value="Safari">

<option value="Microsoft Edge">

</datalist>

Надписи к элементам формы создаются с помощью элемента <label>...</label>. Существует два способа группировки надписи и поля. Если поле находится внутри элемента <label>, то атрибут for указывать не нужно.

<!-- с указанием атрибута for -->

<label for="comments">Когда вы последний раз летали на самолете?</label>

<textarea id="comments"></textarea>

<!-- без атрибута for -->

<p><label>Кошка<input id="cat" type="checkbox"></label></p>

Атрибут for определяет, к какому полю формы привязан данный элемент. Можно создавать поясняющие надписи к следующим элементам формы: <input>, <textarea>, <select>. Значение атрибута содержит идентификатор поля формы.

Элемент <button>...</button> создает кликабельные кнопки. В отличие от кнопок, созданных <input> (<input type="submit"></input>, <input type="image">, <input type="reset">, <input type="button">), внутрь элемента <button> можно поместить контент – текст или изображение.

Для корректного отображения элемента <button> разными браузерами нужно указывать атрибут type, например, <button type="submit"></button>.

Кнопки позволяют пользователям передавать данные в форму, очищать содержимое формы или предпринимать какие-либо другие действия. Можно создавать границы, изменять фон и выравнивать текст на кнопке.

Флажки в формах задаются с помощью конструкции <input type="checkbox">, а переключатель – при помощи <input type="radio">.

Флажков, в отличие от переключателей, в одной форме может быть установлено несколько. Если для флажков указан атрибут checked, то при загрузке станицы на соответствующих полях формы флажки уже будут установлены.

Элемент <label> применяется при реализации выбора с помощью переключателей и флажков. Можно выбрать нужный пункт, просто щелкая кнопкой мыши на тексте, связанном с ним. Для этого нужно поместить <input> внутрь элемента <label>.

**5. Порядок выполнения работы**

1. Создать форму содержащую элементы в соответствии с вариантом.

2. При описании элементов форм необходимо использовать различные атрибуты и значения.

3. Реализовать валидацию формы с помощью атрибута pattern.

Варианты:

1. Регистрация на сайте знакомств

2. Отправки заказа в магазине

3. Отзыв

4. Регистрация на сайте поиска вакансий

5. Поиск и фильтрация в магазине техники

6. Поиск и фильтрация на сайте вакансий

7. Добавление нового товара

8. Заказ билета на самолет

9. Покупка билета на концерт

10. Добавление блюда меню

12. Поиск и фильтрация людей на сайте знакомств

13. Запрос на выдачу кредита

14. Добавление информации о рейсе

15. Добавление информации о мероприятии

16. Поиска и фильтрации книг

17. Поиск и фильтрация фильмов

18. Поиск и фильтрация игр

19. Оформление денежного перевода

20. Вопрос на форуме программистов

21. Поиск и фильтрация вопросов на форуме программистов

22. Поиск и фильтрация платежей

23. Поиск и фильтрация блюд в ресторане

24. Поиска и фильтрации врачей в поликлинике

25. Добавление пациента

26. Заказ ремонта автомобиля

27. Поиск и фильтрация автомобилей

28. Поиск и фильтрация ресторанов

29. Регистрация фирмы

30. Заявка на замену паспорта

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Опишите назначение форм.

2. Перечислите атрибуты тега <FORM>.

3. Опишите назначение различных элементов формы.

4. Каким образом создаются выпадающие списки?

5. Приведите способы связывания надписи с элементами форм. Для каких элементов такое связывание будет работать?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 4**

## **Тема работы: «Использование аудио, видео в HTML-документе. Работа с геолокацией и картами»**

**1. Цель работы**

Формирование умений внедрения аудио и видео контента в HTML-документ, определения местоположения мобильного устройства.

**2. Задание**

Реализовать внедрение медиа контента, поддерживаемого всеми популярными браузерами, в веб-документ.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

HTML5-аудио предоставляет улучшенные возможности работы с аудио контентом. До недавнего времени единственным способом добавления звуковых файлов на веб-страницы было интегрирование фонового звука с помощью тега <bgsound>, который проигрывался во время просмотра пользователем страницы без возможности выключения.

Благодаря добавлению в спецификацию HTML5 нового элемента <audio>, появилась возможность добавлять аудио содержимое со встроенным программным интерфейсом без привлечения подключаемых модулей.

Тег <bgsound> определяет музыкальный файл, который будет проигрываться на веб-странице при ее открытии. Громкость, продолжительность звучания музыки и другие характеристики определяются с помощью атрибутов тега, а также могут управляться через скрипты. Этот тег должен размещаться только в контейнере <head>.

Тег <bgsound> не входит в спецификацию HTML и при его использовании код не пройдет валидацию. К тому же пользователи, как правило, отрицательно относятся к музыке, которая играет на сайте, поэтому используйте эту возможность с осторожностью и по необходимости.

Атрибуты тега <bgsound>:

– [balance](http://htmlbook.ru/html/bgsound/balance) – управляет балансом звука между правой и левой колонками;

– [loop](http://htmlbook.ru/html/bgsound/loop) – устанавливает, сколько раз проигрывать музыкальный файл;

– [src](http://htmlbook.ru/html/bgsound/src) – путь к музыкальному файлу;

– [volume](http://htmlbook.ru/html/bgsound/volume) – задает громкость звучания музыки.

<head>

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

<title>Тег BGSOUND</title>

<bgsound src="town.mid" loop="-1">

</head>

Для внедрения на страницу звукового файла используется тег ‹audio› с необходимыми атрибутами.

<audio src="name.ogg" controls></audio>

Атрибут controls добавляет отображение браузерами интерфейса управления аудио плеера, рисунок 4.1, – кнопки воспроизведения, паузы, громкости.

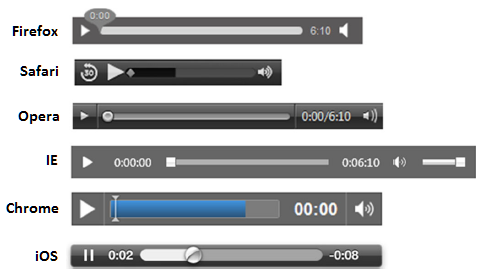


Рисунок 4.1 – Интерфейс аудиоплеера

Элемент <source> используется для добавления нескольких медиа-ресурсов для <audio> и <video>. Указывает на альтернативные видео/аудио файлы, которые браузер может выбрать из предложенных на основании поддерживаемого им типа носителя или кодека.

<audio controls>

<source src="name.ogg" type="audio/ogg">

<source src="name.mp3" type="audio/mpeg">

<a href="sounds/name.mp3">Скачать name.mp3</a>

</audio>

Например, браузер Mozilla Firefox работает с .ogg файлами, тогда как Webkit-браузеры работают с .mp3

Аудио кодек (декодер) представляет собой программу для преобразования цифровых данных в формат звукового файла или звукового потока. Популярными аудио форматами являются следующие:

– MP3 – самый популярный аудио формат, использующий сжатие с потерями и позволяющий уменьшить размер файла в несколько раз. В силу лицензионных отчислений не поддерживается Firefox и Opera;

– AAC (Advanced Audio Codec) – закрытый кодек, аналог MP3, но по сравнению с последним, поддерживает более высокое качество звука при сходном размере файла;

– Ogg Vorbis – бесплатный формат с открытым кодом, поддерживается в Firefox, Opera и Chrome. Обеспечивает хорошее качество звука, но недостаточно широко поддерживается аппаратными проигрывателями.

Атрибуты тега <audio> представлены в таблице 4.1.

Таблица 4.1 – Атрибуты тега <audio>

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| autoplay | Автоматическое воспроизведение аудио файла сразу же после загрузки страницы. |
| controls | Указывает браузеру, что нужно отобразить базовые элементы управления воспроизведением (начинать и останавливать воспроизведение, переходить в другое место записи, регулировать громкость). |
| loop | Циклическое воспроизведение аудио файла. |
| muted | Выключает звук при воспроизведении аудио файла. |
| preload | Атрибут, отвечающий за предварительную загрузку аудио контента. Не является обязательным, некоторые браузеры игнорируют его. Возможные значения:  auto – браузер загружает аудио файл полностью, чтобы он был доступен, когда пользователь начнет его воспроизведение. metadata – браузер загружает первую небольшую часть аудио файла, чтобы определить его основные характеристики. none – отсутствие автоматической загрузки аудио файла. |
| src | Содержит абсолютный или относительный URL-адрес аудио файла. |

‹audio autoplay controls›

    ‹source src="file.ogg" /›

    ‹source src="file.mp3" /›

    ‹a href="file.mp3"›Download this file.‹/a›

‹/audio›

Выводит на странице блок управления аудиофайлом, содержащим ссылку на скачивание этого файла. Звук будет воспроизводиться автоматически.

<video src="videofile.ogg" autoplay poster="posterimage.jpg">

Sorry, your browser doesn't support embedded videos,

but don't worry, you can

<a href="videofile.ogg">download it</a>

and watch it with your favorite video player!

</video>

Атрибуты тега <source> представлены в таблице 4.2.

Таблица 4.2 – Атрибуты тега <source>

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| media | Определяет тип медиа-устройства (т.е. для каких устройств оптимизирован файл). |
| src | Содержит абсолютный или относительный URL-адрес медиафайла. |
| type | Определяет MIME-тип медиафайла. |

Для встраивания видео контента в документ используется элемент HTML5 <video>.

До недавнего времени, единственным способом вставки видеоконтента на HTML-страницу было интегрирование Flash-контента.

HTML5-видео – новый стандарт для размещения мультимедийных файлов в сети с оригинальным программным интерфейсом без привлечения подключаемых модулей. Особую популярность это приобрело тогда, когда на формат HTML5 перешел крупнейший видеохостинг YouTube.com.

Видео элемент может содержать один или несколько источников видео. Чтобы указать источник видео, необходимо использовать атрибут src или элемент [<source>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/source); браузер сам определит наиболее подходящий источник. Внешний вид видеоплеера в различных браузерах представлен на рисунке 4.2.

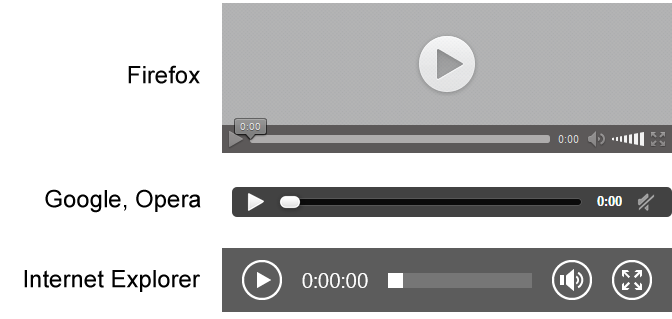


Рисунок 4.2 – Внешний вид видеоплеера в различных браузерах

Не все браузеры могут отображать HTML5 видео, поэтому под тегом ‹source› можно указать ссылку для скачивания видео, либо интегрировать Flash плеер.

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

<video controls>

<source src="somevideo.webm" type="video/webm">

<source src="somevideo.mp4" type="video/mp4">

I'm sorry; your browser doesn't support HTML5 video in WebM with VP8/VP9 or MP4 with H.264.

</video>

Атрибут controls отвечает за появление элементов управления видеоплеером.

С помощью атрибута poster можно добавить изображение, которое браузер будет использовать, пока загружается видео или пока пользователь не нажмет на кнопку воспроизведения, а также задать высоту и ширину видео.

Как и в случае с аудиофайлами, рекомендуется перечислять в <source> все форматы, начиная с более предпочтительного. Также нужно указывать MIME-тип для каждого видеофайла.

<video controls width="400" height="300">

<source src="video.mp4" type="video/mp4"><!-- MP4 для Safari, IE9, iPhone, iPad, Android, и Windows Phone 7 -->

<source src="video.webm" type="video/webm"><!-- WebM/VP8 для Firefox4, Opera, и Chrome -->

<source src="video.ogv" type="video/ogg"><!-- Ogg/Vorbis для старых версий браузеров Firefox и Opera -->

<object data="video.swf" type="application/x-shockwave-flash"><!-- добавляем видеоконтент для устаревших браузеров, в которых нет поддержки элемента video -->

<param name="movie" value="video.swf">

</object>

</video>

<video width="480" controls poster="https://archive.org/download /webmvp8.gif" >

<source src="https://archive.org/download/webmvp8\_512kb.mp4" type="video/mp4">

<source src="https://archive.org/download/webmvp8.ogv" type="video/ogg">

<source src="https://archive.org/webmvp8.webm" type="video/webm">

Your browser doesn't support HTML5 video tag.

</video>

<source src="//msiter.ru/movie.ogg" type="video/ogg" media="screen and (min-width:320px)">

Атрибуты тега <video> представлены в таблицу 4.3.

Таблица 4.3 – Атрибуты тега <video>

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| **1** | **2** |
| autoplay | Автоматическое воспроизведение видеоофайла сразу же после загрузки страницы. |
| controls | Указывает браузеру, что нужно отобразить базовые элементы управления воспроизведением (воспроизведение, пауза, громкость). |

Продолжение таблицы 4.3

|  |  |
| --- | --- |
| **1** | **2** |
| height | Задает высоту окна для отображения видеоданных, возможные значения: px или % |
| loop | Логический атрибут; если указан, то по окончанию проигрывания, видео автоматически начнет воспроизведение с начала. |
| muted | Логический атрибут, который определяет значение по умолчания для аудио дорожки, содержащуюся в видео. Если атрибут указан, то аудио дорожка воспроизводиться не будет. Значение атрибута по умолчанию - "ложь", и это означает, что звук будет воспроизводиться, когда видео воспроизводится. |
| poster | URL файла изображения, которое будет отображаться во время загрузки видеофайла или до тех пор, пока пользователь не нажмет на кнопку PLAY. Если этот атрибут не указан, ничего не отображается до тех пор, пока не будет доступен первый кадр; то первый кадр отображается как рамка постера. |
| preload | Атрибут, отвечающий за предварительную загрузку видеоконтента. Не является обязательным, некоторые браузеры игнорируют его. Возможные значения:  auto  браузер загружает видеофайл полностью, чтобы он был доступен, когда пользователь начнет его воспроизведение (по умолчанию).  metadata – браузер загружает первую небольшую часть видеофайла, чтобы определить его основные характеристики.  none – отсутствие автоматической загрузки видеофайла.  Спецификация рекомендует использовать metadata. |
| src | Содержит абсолютный или относительный URL-адрес видеофайла. |
| width | Задает ширину окна для отображения видеоданных, возможные значения: px или % |

Элемент <embed> определяет контейнер для внешнего приложения или интерактивного содержимого (другими словами, плагина).

Большинство браузеров поддерживало данный элемент на протяжении долгого времени. Тем не менее, данный тег не был включен в спецификацию HTML4, его добавили в спецификацию HTML5.

С помощью <embed> на веб-страницы можно добавлять не только видеофайлы, но и векторные изображения в формате :

<embed src="movie.mov" type="video/quicktime" width="640" height="480">

<embed src="helloworld.swf">

Атрибуты тега <embed> представлены в тыблице 4.4

Таблица 4.4 – Атрибуты тега <embed>

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| height | Определяет высоту встраиваемого контента в px или %. |
| src | Содержит абсолютный или относительный URL-адрес медиафайла. |
| type | Определяет MIME-тип встраиваемого файла. |
| width | Определяет ширину встраиваемого контента в px или %. |

При просмотре видео проигрыватель должен его декодировать. Одни проигрыватели используют программное декодирование видеопотока, другие используют аппаратное декодирование.

Видео в формате .avi на сайте средствами HTML5 не воспроизводится. Поэтому его необходимо перекодировать в эти три формата с соответствующими видео и аудиокодеками для вывода на сайте. Для этого можно использовать видеоконвертеры.

На данный момент браузеры поддерживают три основных видео формата, таблица 4.5.

Таблица 4.5 – Основные видеоформаты

|  |  |  |
| --- | --- | --- |
| **Формат** | **Видеокодек** | **Аудиокодек** |
| .mp4 | H.264 | AAC |
| .ogg/.ogv | Theora | Vorbis |
| .webm | VP8 | Vorbis |

Элемент <track> используется в качестве дочернего элемента <audio> и <video>. Добавляет текстовую дорожку для субтитров, заголовков медиафайлов или другой текстовой информации, которая должна быть видна во время воспроизведения аудио или видео файла.

Атрибуты тега <track> представлены в таблице 4.6.

Таблица 4.6 – Атрибуты тега <track>

|  |  |
| --- | --- |
| **Атрибут** | **Описание, принимаемое значение** |
| **1** | **2** |
| default | Указывает, что данная дорожка воспроизводится по умолчанию. Только один элемент <track> может содержать данный атрибут. |
| kind | Указывает тип текстовой дорожки, по умолчанию выводятся субтитры (subtitles). Принимаемые значения:  captions — перевод диалогов и звуковых эффектов, отображаемый в виде текста поверх видео (для глухих пользователей).  chapters — добавляет названия глав в виде списка для навигации по медиафайлу.  descriptions — добавляет звуковое описание происходящего в видео (для слепых пользователей).  metadata — метаданные, используемые скриптами, не отображаются для |

Продолжение таблицы 4.6

|  |  |
| --- | --- |
| **1** | **2** |
|  | пользователей.  subtitles — текстовое дублирование звуковой дорожки видео, отображается в виде субтитров к видео. |
| label | Добавляет название дорожки. Если этот атрибут не задан, браузер применит значение по умолчанию. |
| src | Содержит абсолютный или относительный URL-адрес аудио- или видео файла. |
| srclang | Язык воспроизводимой дорожки. |

<!-- Simple audio playback -->

<audio

src="http://developer.mozilla.org/@api/deki/files/2926/=AudioTest\_(1).ogg"

autoplay>

Ваш браузер не поддерживает <code>audio</code> элемент.

</audio>

<!-- Audio playback with captions -->

<audio src="foo.ogg">

<track kind="captions" src="foo.en.vtt" srclang="en" label="English">

<track kind="captions" src="foo.sv.vtt" srclang="sv" label="Svenska">

</audio>

<!-- Video with subtitles -->

<video src="foo.ogg">

<track kind="subtitles" src="foo.en.vtt" srclang="en" label="English">

<track kind="subtitles" src="foo.sv.vtt" srclang="sv" label="Svenska">

</video>

Пример размещения видео на сайте:

<video controls width="710" height="538" poster="/examples/media/martynko.png" preload="none">

<source src="/examples/media/martynko.mp4" type="video/mp4">

<source src="/examples/media/martynko.webm" type="video/webm">

<source src="/examples/media/martynko.ogv" type="video/ogg">

</video>

С помощью css-стилей можно придать аудио плееру необычный вид.

Также выровнять видеоплеер на странице можно обернув элемент <video> в контейнер <div> с присвоенным классом, для которого задаются ширина и высота, соответствующие размерам вашего видео. Далее, с помощью css-свойств можно задать отступы, выравнивание на странице и т.д.

HTML5 Geolocation API дает возможность пользователям обмениваться информацией об их местоположении (координаты долготы и широты в браузере) в веб-приложениях.

Для этого пользователь должен разрешить возможность определения своего расположения.

Geolocation API поддерживается большеством современных браузеров: Firefox 3.5+, Opera 10.6+, Chrome 5+, Safari 5+ и Internet Explorer 9+.

Необходимо проверить поддерживает или нет HTML5 Geolocation API браузер пользователя.

Это можно сделать с помощью кода:

<script type="text/javascript">

if(navigator.geolocation) {

alert("Geolocation API поддерживается");

} else {

alert("Geolocation API не поддерживается в вашем браузере");

}

</script>

Определить местоположение пользователя можно, используя следующий код:

navigator.geolocation.getCurrentPosition(function(position) {

var latitude = position.coords.latitude;

var longitude = position.coords.longitude;

}

Соединим два примера кода в один, получим простейший пример определения положения пользователя:

<!DOCTYPE HTML>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>HTML5 Geolocation API</title>

<script type="text/javascript">

if(navigator.geolocation) {

navigator.geolocation.getCurrentPosition(function(position) {

var latitude = position.coords.latitude;

var longitude = position.coords.longitude;

alert(latitude+' '+longitude);

});

} else {

alert("Geolocation API не поддерживается в вашем браузере");

}

</script>

</head>

<body>

<p>Пример определения местоположения пользователя – HTML5 Geolocation API</p>

</body>

</html>

При просмотре примера в верху окна браузера может появиться информационная панель вида, представленного на рисунке 4.3.

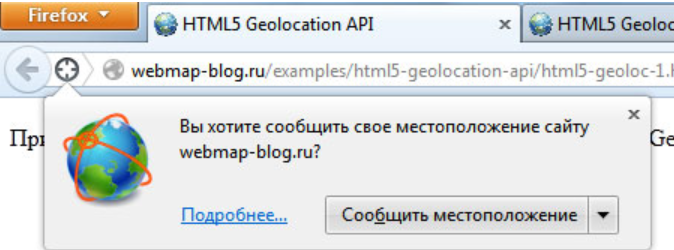


Рисунок 4.3 – Информационная панель с сообщением

Информационная панель с сообщением, нужно разрешить сообщить местоположение, после чего увидим окно с координатами, рисунок 4.4.

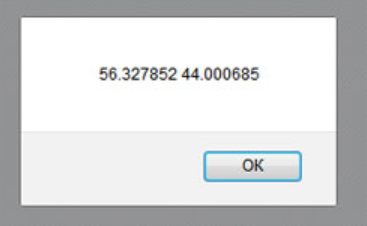


Рисунок 4.4 – Окно с координатами

**5. Порядок выполнения работы**

1.Добавить аудио- файл на html- страницу таким образом, чтобы мелодию можно было прослушать в любом браузере. Установить циклическое проигрывание.

2. Добавить видео- контент на html-страницу, отображаемый во всех современных браузерах. В качестве изображения до загрузки файла использовать рекламный постер.

3. Добавить текстовую дорожку к медиа-файлам с названием файла.

4. Используя объект navigator определить местоположение пользователя.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Как добавить аудио и видео на HTML-страницу?

2. Для чего предназначен тег <source>?

3. С помощью какого тега можно добавить субтитры и заголовок?

4. Каким образом можно получить информацию о местоположении пользователя?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
3. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
5. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
6. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 5**

## **Тема работы: «Подключение и использование каскадных таблиц стилей»**

**1. Цель работы**

Формирование умений внедрения CSS в HTML-документ различными способами и использования его свойств для управления внешним видом HTML-документа.

**2. Задание**

Реализовать внедрении CSS различными способами, форматирование внешнего вида элемнтов веб-документа.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

CSS (Cascading Style Sheets), или каскадные таблицы стилей, используются для описания внешнего вида документа, написанного языком разметки.

Объявление стиля состоит из двух частей: элемента веб-страницы – селектора, и команды форматирования – блока объявления. Селектор сообщает браузеру, какой именно элемент форматировать, а в блоке объявления (код в фигурных скобках) перечисляются форматирующие команды – свойства и их значения. Правило описания CSS-стиля представлено на рисунке 5.1.

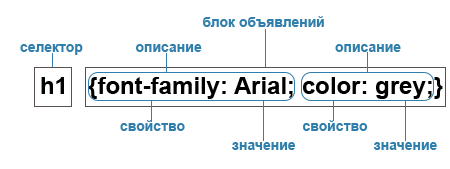


Рисунок 5.1 – Правило описания CSS-стиля

Существуют четыре способа применения таблиц стилей к документу:

1) связывание;

2) встраивание;

3) оперативное определение;

4) импорт.

В случае связывания создается внешняя таблица стилей. Внешняя таблица стилей представляет собой текстовый файл с расширением .css, в котором находится набор CSS-стилей элементов. Файл создаётся в редакторе кода, так же как и HTML-страница. Внутри файла могут содержатся только стили, без HTML-разметки. Внешняя таблица стилей подключается к веб-странице с помощью тега <link>, расположенного внутри раздела <head></head>. Такие стили работают для всех страниц сайта.

К каждой веб-странице можно присоединить несколько таблиц стилей, добавляя последовательно несколько тегов <link>, указав в атрибуте тега media назначение данной таблицы стилей. rel="stylesheet" указывает тип ссылки (ссылка на таблицу стилей):

<head>

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

<link rel="stylesheet" href="css/assets.css" media="all">

</head>

Атрибут type="text/css" не является обязательным по стандарту HTML5, поэтому его можно не указывать. Если атрибут отсутствует, по умолчанию используется значение type="text/css".

Внутренние стили встраиваются в раздел <head></head> HTML-документа и определяются внутри тега <style></style>.

<head>

<style>

h1, h2 {

color: red;

font-family: "Times New Roman", Georgia, Serif;

line-height: 1.3em;

}

</style>

</head>

<body>

...

</body>

Для реализации встроенных стилей необходимо CSS-код прописать в HTML-файл, непосредственно внутри тега элемента с помощью атрибута style:

<p style="font-weight: bold; color: red;">Обратите внимание на этот текст.</p>

Такие стили действуют только на тот элемент, для которого они заданы

Внутренние стили имеют приоритет над внешними, но уступают встроенным стилям (заданным через атрибут style).

Правило @import позволяет загружать внешние таблицы стилей. Чтобы директива @import работала, она должна располагаться в таблице стилей (внешней или внутренней) перед всеми остальными правилами:

<style>

@import url(mobile.css);

p {

font-size: 0.9em;

color: grey;

}

</style>

Правило @import также используется для подключения веб-шрифтов:

@import url(https://fonts.googleapis.com/css?family=Open+Sans&subset=latin,cyrillic);

Селекторы представляют структуру веб-страницы. С их помощью создаются правила для форматирования элементов веб-страницы. Селекторами могут быть элементы, их классы и идентификаторы, а также псевдоклассы и псевдоэлементы.

Универсальный селектор. Соответствует любому HTML-элементу. Например,

\* {margin: 0;}

обнулит внешние отступы для всех элементов сайта. Также селектор может использоваться в комбинации с псевдоклассом или псевдоэлементом:

\*:after {CSS-стили},

\*:checked {CSS-стили}.

Селектор элемента. Селекторы элементов позволяют форматировать все элементы данного типа на всех страницах сайта. Например,

h1 {font-family: Lobster, cursive;}

задаст общий стиль форматирования всех заголовков h1.

Селектор класса. Селекторы класса позволяют задавать стили для одного и более элементов с одинаковым именем класса, размещенных в разных местах страницы или на разных страницах сайта. Например, для создания заголовка с классом headline необходимо добавить атрибут class со значением headline в открывающий тег <h1> и задать стиль для указанного класса. Стили, созданные с помощью класса, можно применять к другим элементам, не обязательно данного типа.

<h1 class="headline">Инструкция пользования персональным компьютером</h1>

.headline {

text-transform: uppercase;

color: lightblue;}

Селектор идентификатора. Селектор идентификатора позволяет форматировать один конкретный элемент. Идентификатор id должен быть уникальным и на одной странице может встречаться только один раз.

<div id="sidebar"></div>

#sidebar {

width: 300px;

float: left;

}

CSS-стили задаются и работают следующим образом:

1) стили могут наследоваться от родительского элемента (наследуемые свойства или с помощью значения inherit);

2) стили, расположенные в таблице стилей ниже, отменяют стили, расположенные в таблице выше;

3) к одному элементу могут применяться стили из разных источников. Проверить, какие стили применяются, можно в режиме разработчика браузера. Для этого над элементом нужно щёлкнуть правой кнопкой мыши и выбрать пункт «Посмотреть код» (или что-то аналогичное). В правом столбце будут перечислены все свойства, которые заданы для этого элемента или наследуются от родительского элемента, а также файлы стилей, в которых они указаны, и порядковый номер строки кода;

4) при определении стиля можно использовать любую комбинацию селекторов – селектор элемента, псевдокласса элемента, класса или идентификатора элемента.

<div id="wrap" class="box clear"></div>

div {border: 1px solid #eee;}

#wrap {width: 500px;}

.box {float: left;}

.clear {clear: both;}

В таблице 5.1 перечислены некоторые часто используемые CSS свойства элементов и их назначение.

Таблица 5.1 – Наиболее часто используемые свойства CSS

|  |  |  |
| --- | --- | --- |
| **Имя** | **Значения** | **Описание** |
| **1** | **2** | **3** |
| background | [background-color || background-image || background-repeat || background-attachment || background-position] | inherit | Управление фоном элемента |
| background-color | <color> | transparent | inherit | Цвет фона |
| background-image | <uri> | none | inherit | Фоновое изображение |
| background-position | [ [<percentage> | <length> ]{1,2} | [ [top | center | bottom] || [left | center | right] ] ] | inherit | Положение фоновой картинки |
| background-repeat | repeat | repeat-x | repeat-y | no-repeat | inherit | Повторение фоновой картинки |
| border | [ border-width || border-style || <color> ] | inherit | Границы элемента |
| border-collapse | collapse | separate | inherit | Объединение/разделение смежных границ |
| border-color | <color>{1,4} | transparent | inherit | Цвет границы |
| border-style | <border-style>{1,4} | inherit | Стиль линии границы |
| border-top border-right border-bottom border-left | [ border-top-width || border-style || <color> ] | inherit | Управление стилем заданной границы |
| border-width | <border-width>{1,4} | inherit | Толщина линии границы |
| bottom | <length> | <percentage> | auto | inherit | Низ элемента |
| clear | none | left | right | both | inherit | Запрет заполнения свободного пространства рядом с элементом |
| clip | <shape> | auto | inherit | Обрезка содержимого элемента |
| color | <color> | inherit | Цвет содержимого |
| cursor | [ [<uri> ,]\* [ auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help ] ] | inherit | Форма курсора |
| display | inline | block | list-item | run-in | compact | marker | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit | Способ отображения элемента |
| empty-cells | show | hide | inherit | Отображение пустых ячеек таблицы |
| float | left | right | none | inherit | Свободное размещение элемента |

Продолжение таблицы 5.1

|  |  |  |
| --- | --- | --- |
| font | [ [ font-style || font-variant || font-weight ]? font-size [ / line-height ]? font-family ] | caption | icon | menu | message-box | small-caption | status-bar | inherit | Управление шрифтом |
| font-family | [[ <family-name> | <generic-family> ],]\* [<family-name> | <generic-family>] | inherit | Гарнитура |
| font-size | <absolute-size> | <relative-size> | <length> | <percentage> | inherit | Кегль |
| font-style | normal | italic | oblique | inherit | Стиль шрифта |
| font-variant | normal | small-caps | inherit | Варианты отображения шрифта |
| font-weight | normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit | Толщина шрифта |
| height | <length> | <percentage> | auto | inherit | Ширина элемента |
| left | <length> | <percentage> | auto | inherit | Положение левой границы элемента |
| line-height | normal | <number> | <length> | <percentage> | inherit | Высота строки |
| list-style | [ list-style-type || list-style-position || list-style-image ] | inherit | Стиль списка |
| margin | <margin-width>{1,4} | inherit | Внешний отступ |
| margin-top margin-right margin-bottom margin-left | <margin-width> | inherit | Внешний отступ по заданной стороне |
| padding | <padding-width>{1,4} | inherit | Внутренний отступ |
| padding-top padding-right padding-bottom padding-left | <padding-width> | inherit | Внутренний отступ по заданной стороне |
| position | static | relative | absolute | fixed | inherit | Позиционирование элемента |
| right | <length> | <percentage> | auto | inherit | Положение правой границы |
| text-align | left | right | center | justify | <string> | inherit | Выравнивание текстового блока |
| text-decoration | none | [ underline || overline || line-through || blink ] | inherit | Текстовые эффекты |
| text-indent | <length> | <percentage> | inherit | Абзацный отступ |
| text-transform | capitalize | uppercase | lowercase | none | inherit | Начертание текста |
| top | <length> | <percentage> | auto | inherit | Положение верхней границы элемента |
| vertical-align | baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit | Вертикальное выравнивание в пределах блока |
| visibility | visible | hidden | collapse | inherit | Управление видимостью элемента |
| white-space | normal | pre | nowrap | inherit | Управление пробелами между словами |

Продолжение таблицы 5.1

|  |  |  |
| --- | --- | --- |
| width | <length> | <percentage> | auto | inherit | Ширина элемента |
| z-index | auto | <integer> | inherit | Порядок перехода по клавише Tab |

**5. Порядок выполнения работы**

1. В файле style.css определите цвет фона и текста документа, свойства полос прокрутки.

2. Подключите файл style.css к документу Lab5.html (методом связывания).

3. Оформите внешний вид таблицы в соответствии с рисунком 5.2.

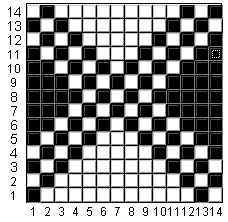


Рисунок 5.2 – Пример оформления таблицы

4. Добавьте на HTML-страницу ссылки, выполните их стилистическое оформлении посредством CSS.

5. Выполните задание по варианту, оформив страницу с помощью таблиц стилей.

Вариант 1. Задание приведено на рисунке 5.3.

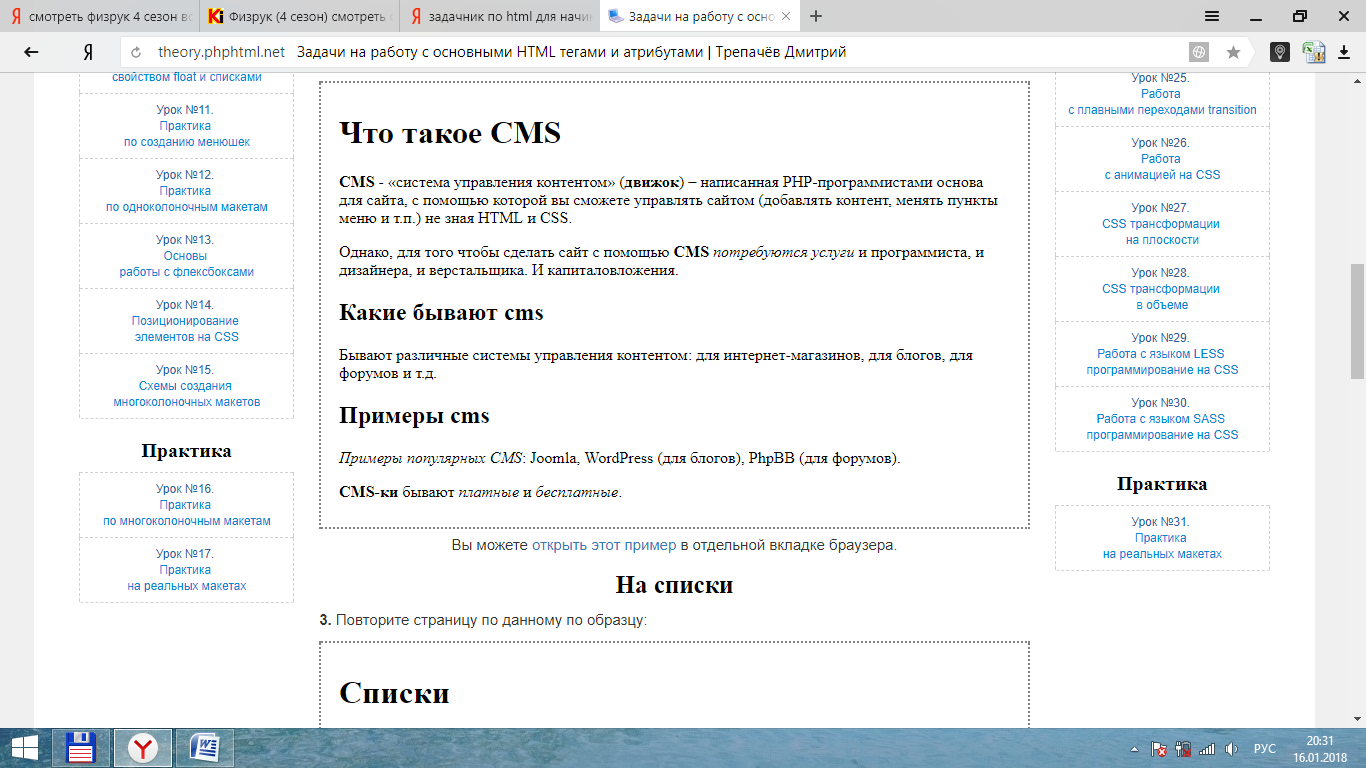


Рисунок 5.3 – Задание для варианта 1

Вариант 2. Задание приведено на рисунке 5.4.

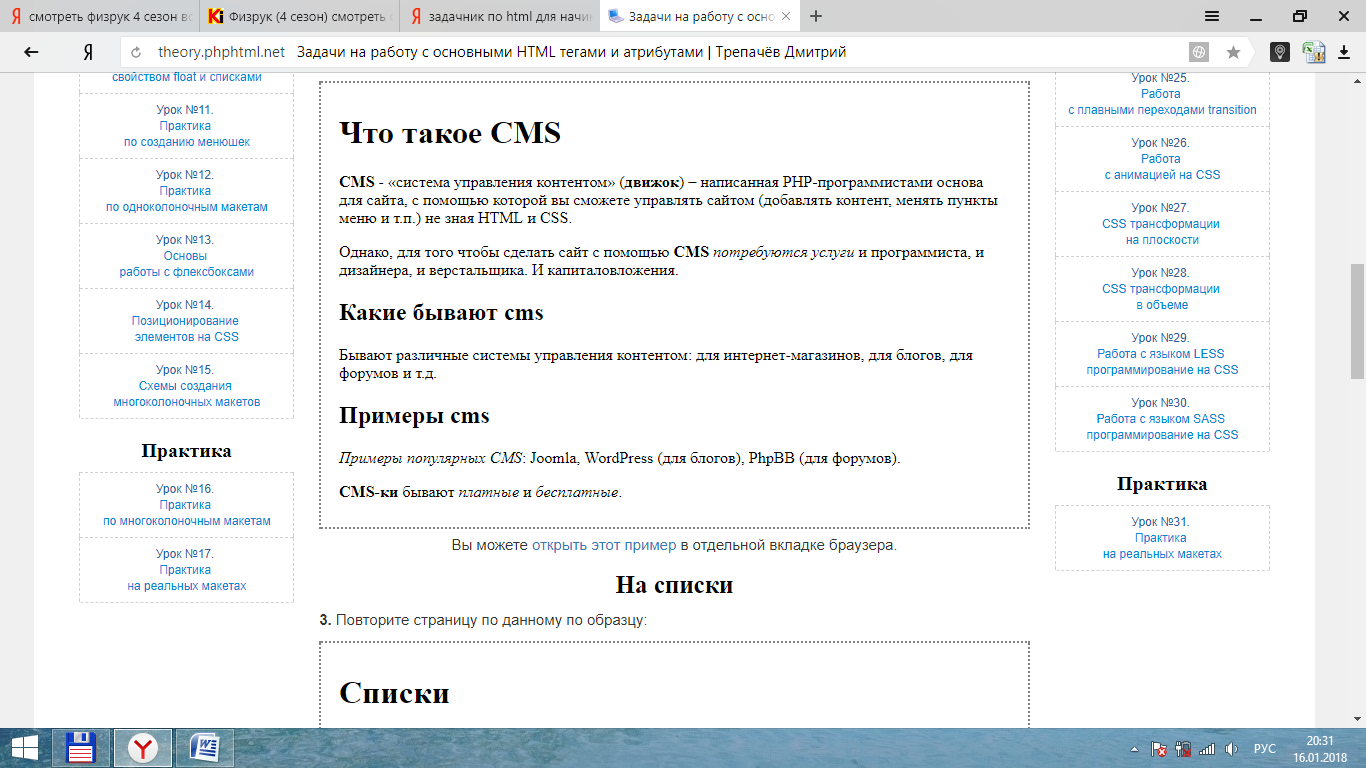


Рисунок 5.4 – Задание для варианта 2

Вариант 3. Задание приведено на рисунке 5.5.

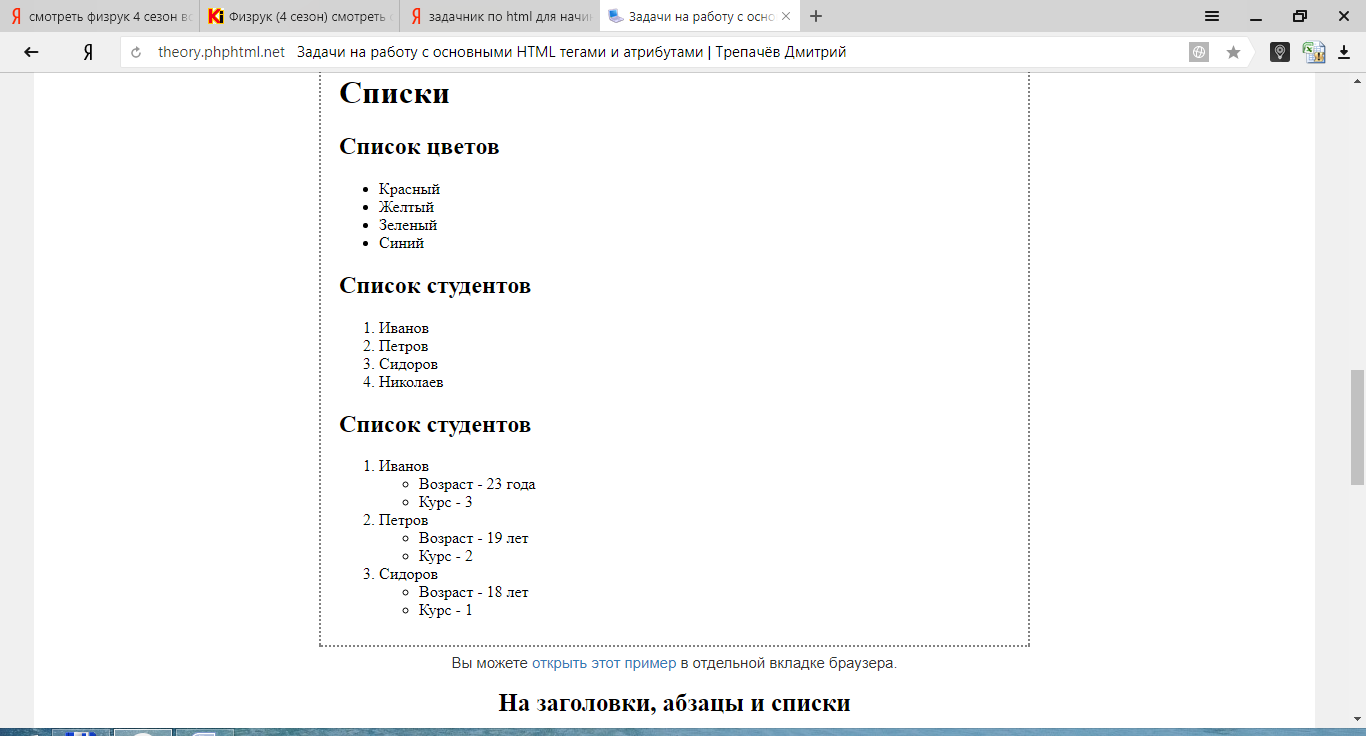


Рисунок 5.5 – Задание для варианта 3

Вариант 4. Задание приведено на рисунке 5.6.

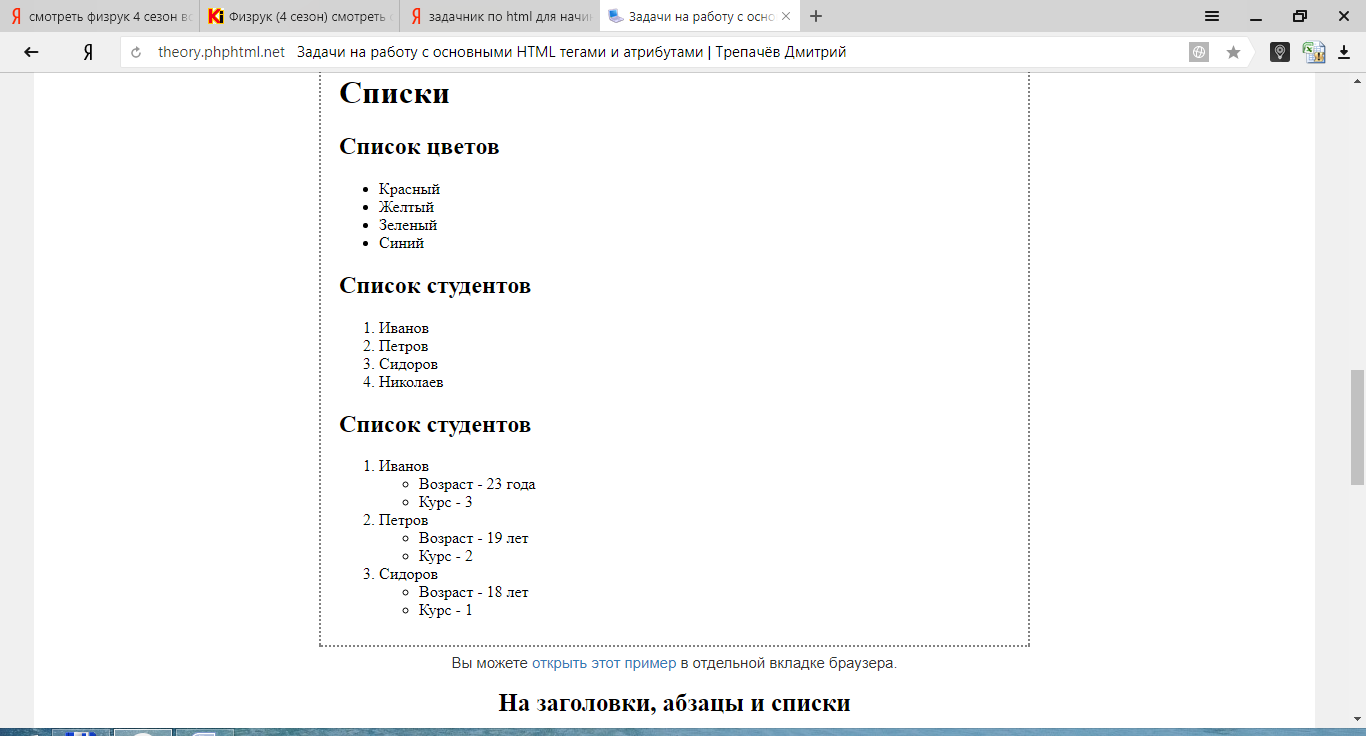


Рисунок 5.6 – Задание для варианта 4

Вариант 5. Задание приведено на рисунке 5.7.

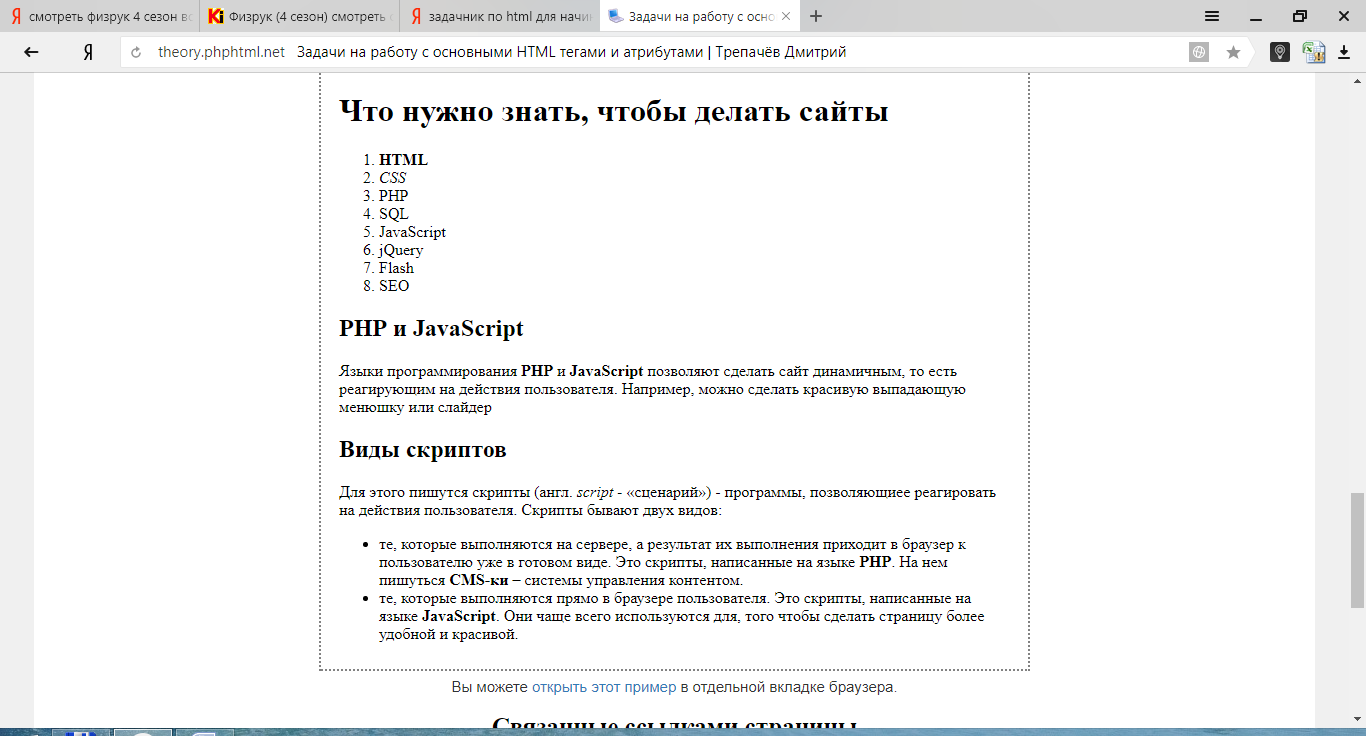


Рисунок 5.7 – Задание для варианта 5

Вариант 6. Задание приведено на рисунке 5.8.

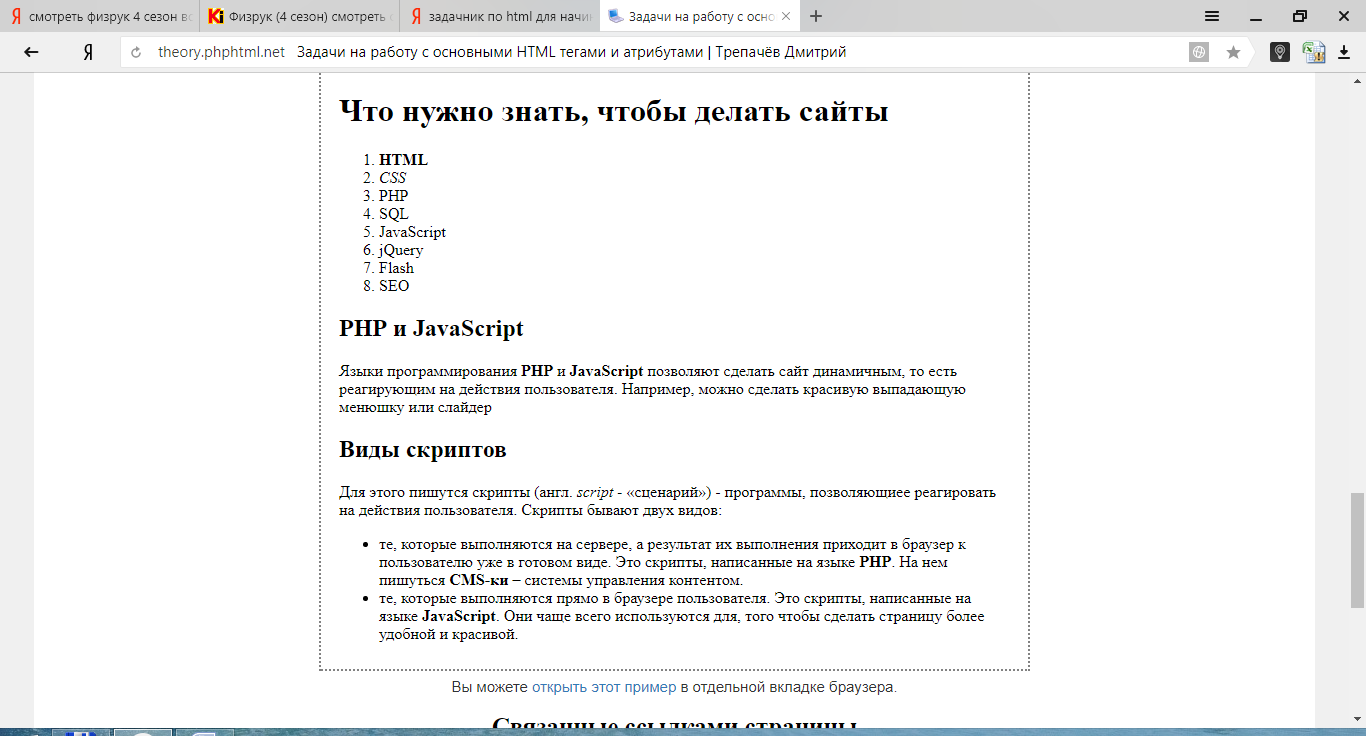


Рисунок 5.8 – Задание для варианта 6

Ваниант 7. Задание приведено на рисунке 5.9.

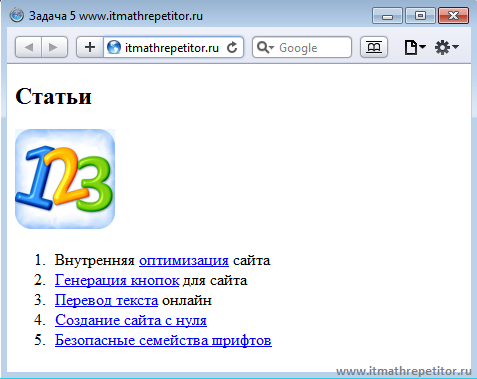


Рисунок 5.9 – Задание для варианта 7

Вариант 8. Задание приведено на рисунке 5.10.

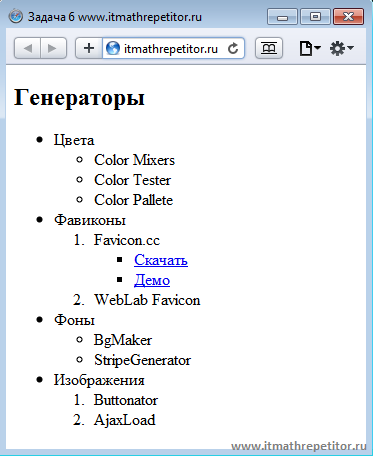


Рисунок 5.10 – Задание для варианта 8

Вариант 9. Задание приведено на рисунке 5.11.

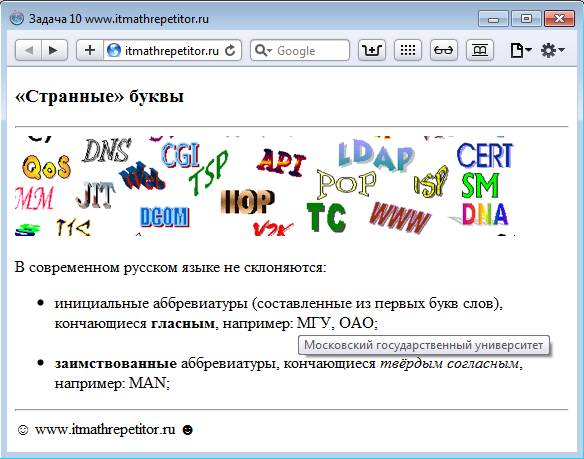


Рисунок 5.11 – Задание для варианта 9

Вариант 10. Задание приведено на рисунке 5.12.

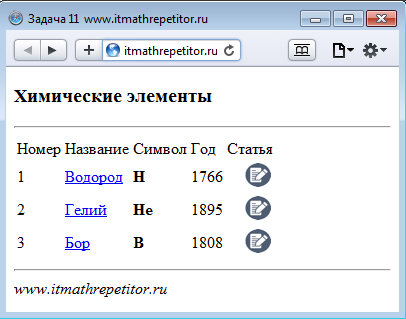


Рисунок 5.12 – Задание для варианта 10

Вариант 11. Задание приведено на рисунке 5.13.

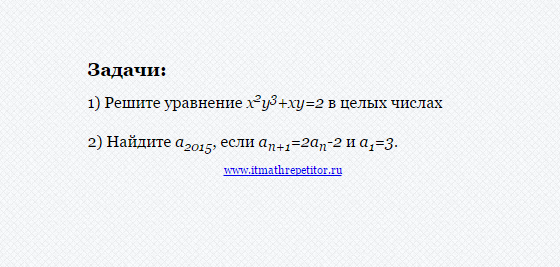


Рисунок 5.13 – Задание для варианта 11

Вариант 12. Задание приведено на рисунке 5.14.

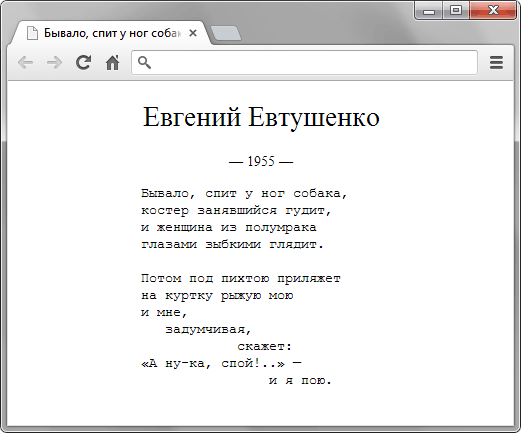


Рисунок 5.14 – Задание для варианта 12

Вариант 13. Задание приведено на рисунке 5.15.

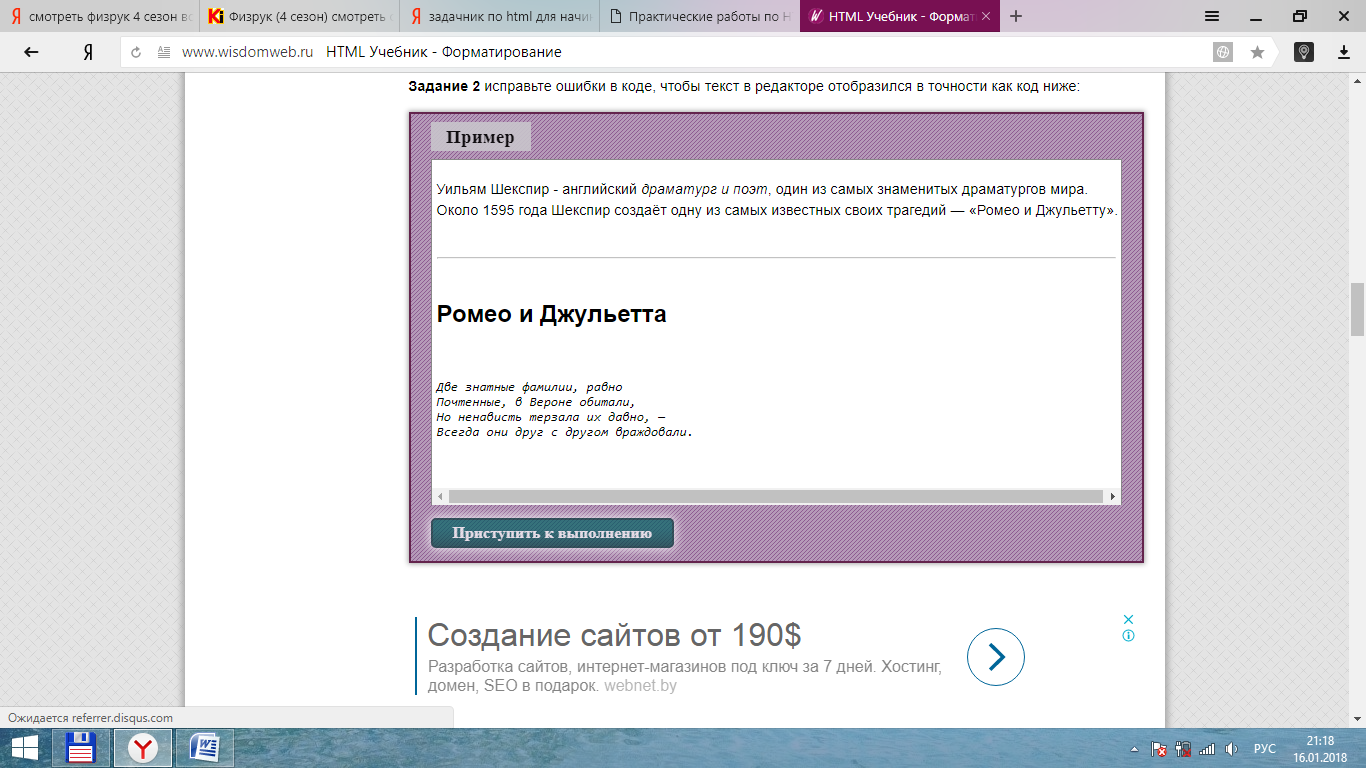


Рисунок 5.15 – Задание для варианта 13

Вариант 14. Задание приведено на рисунке 5.16.

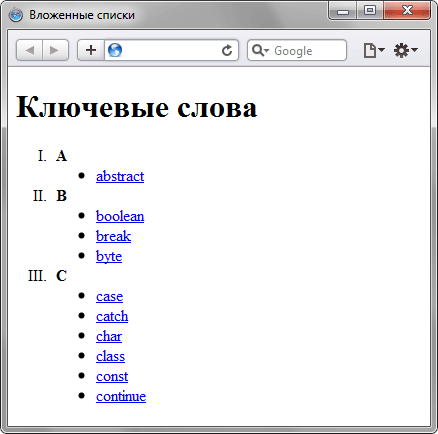


Рисунок 5.16 – Задание для варианта 14

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Перечислите способы использования каскадных таблиц стилей. Укажите преимущества и недостатки каждого из них.

2. Укажите формат правила CSS.

3. Что такое селектор?

4. Как создать класс, не связанный с определенным тегом?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
5. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 6**

## **Тема работы: «Использование классов, псевдоклассов, идентификаторов и принципов наследования и приоритетов правил CSS в web-документе.»**

**1. Цель работы**

Формирование умений использования классов, псевдоклассов и идентификаторов, создания составных селекторов, переопределения значений свойств CSS в соответствии с приоритетами.

**2. Задание**

Реализовать форматирование внешнего вида HTML-элементов средствами CSS используя спецефичность различных видов селекторов.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Селекторы потомков применяют стили к элементам, расположенным внутри элемента-контейнера. Например, ul li {text-transform: uppercase;} – выберет все элементы li, являющиеся потомками всех элементов ul.

Если нужно отформатировать потомки определенного элемента, этому элементу нужно задать стилевой класс:

p.first a {color: green;} – данный стиль применится ко всем ссылкам, потомкам абзаца с классом first;

p .first a {color: green;} – если добавить пробел, то будут стилизованы ссылки, расположенные внутри любого тега класса .first, который является потомком элемента <p>;

.first a {color: green;} – данный стиль применится к любой ссылке, расположенной внутри другого элемента, обозначенного классом .first.

Дочерний элемент является прямым потомком содержащего его элемента. У одного элемента может быть несколько дочерних элементов, а родительский элемент у каждого элемента может быть только один. Дочерний селектор позволяет применить стили только если дочерний элемент идёт сразу за родительским элементом и между ними нет других элементов, то есть дочерний элемент больше ни во что не вложен.

Например, p > strong – выберет все элементы strong, являющиеся дочерними по отношению к элементу p.

Сестринские отношения возникают между элементами, имеющими общего родителя. Селекторы сестринских элементов позволяют выбрать элементы из группы элементов одного уровня:

h1 + p – выберет все первые абзацы, идущие непосредственно за любым тегом <h1>, не затрагивая остальные абзацы;

h1 ~ p – выберет все абзацы, являющиеся сестринскими по отношению к любому заголовку h1 и идущие сразу после него.

Селекторы атрибутов выбирают элементы на основе имени атрибута или значения атрибута:

[атрибут] – все элементы, содержащие указанный атрибут,

[alt] – все элементы, для которых задан атрибут alt;

селектор[атрибут] – элементы данного типа, содержащие указанный атрибут,

img[alt] – только картинки, для которых задан атрибут alt;

селектор[атрибут="значение"] эле–менты данного типа, содержащие указанный атрибут с конкретным значением,

img[title="flower"] – все картинки, название которых содержит слово flower;

селектор[атрибут~="значение"] – элементы, частично содержащие данное значение, например, если для элемента задано несколько классов через пробел,

p[class~="feature"] – абзацы, имя класса которых содержит feature;

селектор[атрибут|="значение"] – элементы, список значений атрибута которых начинается с указанного слова,

p[class|="feature"] – абзацы, имя класса которых feature или начинается на feature;

селектор[атрибут^="значение"] – элементы, значение атрибута которых начинается с указанного значения,

a[href^="http://"] – все ссылки, начинающиеся на http://;

селектор[атрибут$="значение"] – элементы, значение атрибута которых заканчивается указанным значением,

img[src$=".png"] – все картинки в формате png;

селектор[атрибут\*="значение"] – элементы, значение атрибута которых содержит в любом месте указанное слово,

a[href\*="book"] – все ссылки, название которых содержит book.

Селектор псевдокласса. Псевдоклассы – это классы, фактически не прикрепленные к HTML-тегам. Они позволяют применить CSS-правила к элементам при совершении события или подчиняющимся определенному правилу. Псевдоклассы характеризуют элементы со следующими свойствами:

:link – не посещенная ссылка;

:visited – посещенная ссылка;

:hover – любой элемент, по которому проводят курсором мыши;

:focus – интерактивный элемент, к которому перешли с помощью клавиатуры или активировали посредством мыши;

:active – элемент, который был активизирован пользователем;

:valid – поля формы, содержимое которых прошло проверку в браузере на соответствие указанному типу данных;

:invalid – поля формы, содержимое которых не соответствует указанному типу данных;

:enabled – все активные поля форм;

:disabled – заблокированные поля форм, т.е., находящиеся в неактивном состоянии;

:in-range – поля формы, значения которых находятся в заданном диапазоне;

:out-of-range – поля формы, значения которых не входят в установленный диапазон;

:lang() – элементы с текстом на указанном языке;

:not(селектор) – элементы, которые не содержат указанный селектор – класс, идентификатор, название или тип поля формы – :not([type="submit"]);

:target – элемент с символом #, на который ссылаются в документе;

:checked – выделенные (выбранные пользователем) элементы формы.

Селектор структурных псевдоклассов. Структурные псевдоклассы отбирают дочерние элементы в соответствии с параметром, указанным в круглых скобках:

:nth-child(odd) – нечётные дочерние элементы;

:nth-child(even) – чётные дочерние элементы;

:nth-child(3n) – каждый третий элемент среди дочерних;

:nth-child(3n+2) – выбирает каждый третий элемент, начиная со второго дочернего элемента (+2);

:nth-child(n+2) – выбирает все элементы, начиная со второго;

:nth-child(3) – выбирает третий дочерний элемент;

:nth-last-child() – в списке дочерних элементов выбирает элемент с указанным местоположением, аналогично с :nth-child(), но начиная с последнего, в обратную сторону;

:first-child – позволяет оформить только самый первый дочерний элемент тега;

:last-child – позволяет форматировать последний дочерний элемент тега;

:only-child – выбирает элемент, являющийся единственным дочерним элементом;

:empty – выбирает элементы, у которых нет дочерних элементов;

:root – выбирает элемент, являющийся корневым в документе – элемент html.

Селекторы структурных псевдоклассов типа указывают на конкретный тип дочернего тега:

:nth-of-type() – выбирает элементы по аналогии с :nth-child(), при этом берёт во внимание только тип элемента;

:first-of-type – выбирает первый дочерний элемент данного типа;

:last-of-type – выбирает последний элемент данного типа;

:nth-last-of-type()– выбирает элемент заданного типа в списке элементов в соответствии с указанным местоположением, начиная с конца;

:only-of-type – выбирает единственный элемент указанного типа среди дочерних элементов родительского элемента.

Селектор псевдоэлемента. Псевдоэлементы используются для добавления содержимого, которое генерируется с помощью свойства content:

:first-letter – выбирает первую букву каждого абзаца, применяется только к блочным элементам;

:first-line – выбирает первую строку текста элемента, применяется только к блочным элементам;

:before – вставляет генерируемое содержимое перед элементом;

:after – добавляет генерируемое содержимое после элемента.

Комбинация селекторов. Для более точного отбора элементов для форматирования можно использовать комбинации селекторов:

a[href][title] – выберет все ссылки, для которых заданы атрибуты href и title;

img[alt\*="css"]:nth-of-type(even) – выберет все четные картинки, альтернативный текст которых содержит слово css.

Группировка селекторов. Один и тот же стиль можно одновременно применить к нескольким элементам. Для этого необходимо в левой части объявления перечислить через запятую нужные селекторы:

h1, h2, p, span {

color: tomato;

background: white;

}

Наследование является механизмом, с помощью которого определенные свойства передаются от предка к его потомкам. Спецификацией CSS предусмотрено наследование свойств, относящихся к текстовому содержимому страницы, таких как color, font, letter-spacing, line-height, list-style, text-align, text-indent, text-transform, visibility, white-space и word-spacing. Во многих случаях это удобно, так как не нужно задавать размер шрифта и семейство шрифтов для каждого элемента веб-страницы.

Свойства, относящиеся к форматированию блоков, не наследуются. Это background, border, display, float и clear, height и width, margin, min-max-height и -width, outline, overflow, padding, position, text-decoration, vertical-align и z-index.

Принудительное наследование. С помощью ключевого слова inherit можно принудить элемент наследовать любое значение свойства родительского элемента. Это работает даже для тех свойств, которые не наследуются по умолчанию.

В таблице 6.1 представлены популярные свойства и псевдоклассы CSS.

Таблица 6.1 – Популярные свойства и псевдоклассы CSS

|  |  |
| --- | --- |
| **Категория: Текст** | |
| text-align | Выравнивание текста по горизонтали в пределах данного элемента |
| text-decoration | Добавление дополнительного оформления в виде подчеркивания (none отменяет подчеркивание) |
| **Категория: Отступы** | |
| margin | Задает величину  отступа от каждого края элемента |
| **Категория: Шрифт** | |
| font-family | Определяет каким шрифтом(Например Times new Roman) будет оформлен,  тест, также может использоваться в данном свойстве несколько шрифтов через запятую(на случай если какой то шрифт не будет найден) |
| font-size | Задается размер шрифта (Например 14px, 12px и т.п.) |
| font-style | Это свойство определяет какой будет шрифт - обычный курсив или наклонный |
| **Категория: Позиционирование** | |
| position | С помощью этого свойства определяется позиция элемента относительно браузера |
| left | Определяет отступ от левого края, |
| bottom | низа, |
| right | правого края, |
| top | верхнего края |
| **Категория: Размеры** | |
| height | Задает высоту данного блок или других элементов (например img) |
| widht | Задает ширину данного блок или других элементов (например img) |
| max-height | Задает максимальную высоту данного элемента |
| max-widht | Задает максимальную ширину данного элемента |
| min-height | Задает минимальную высоту данного элемента |
| min-widht | Задает минимальную ширину данного элемента |
| **Категория: Список** | |
| list-style | Свойство позволяющие задать стиль маркера, положение, и изображение |
| **Категория: Скругленные уголки** | |
| border-radius | Добавляет скругленные уголки данному элементу |
| **Категория: Таблица** | |
| border-collaps | Данное свойство, устанавливает как отобразить границу вокруг ячеек |
| empty-cells | Если в таблице в ячейке нет данных, то данное свойство отображает фон и границу |
| table-layout | Свойство определяет как браузер должен вычислить ширину ячеек, основываясь на содержимом. |
| **Категория: Поля** | |
| padding | Данное свойство определяет размер отступа вокруг элемента |
| **Категория: Псевдоклассы** | |
| active | Определяет стиль для активной ссылки. |
| hover | Данный псевдокласс определят стиль для данного элемента, при наведение на него курсором,при этом элемент еще не является активным |
| nth-child | Используется для добавления стиля к  элементам на основание номера. Например нам нужно применить разные стили к каждому пункту меню. Для этого нам и нужен псевдокласс nth-child |

Продолжение таблицы 6.1

|  |  |
| --- | --- |
| **Категория: Анимация** | |
| transition | Определяет эффект перехода между двумя элементами, точнее между их состояниями |

Каскадирование – это механизм, который управляет конечным результатом в ситуации, когда к одному элементу применяются разные CSS-правила. Существует три критерия, которые определяют порядок применения свойств – правило !important, специфичность и порядок, в котором подключены таблицы стилей.

Правило !important. Вес правила можно задать с помощью ключевого слова !important, которое добавляется сразу после значения свойства, например, span {font-weight: bold!important;}. Правило необходимо размещать в конец объявления перед закрывающей скобкой, без пробела. Такое объявление будет иметь приоритет над всеми остальными правилами. Это правило позволяет отменить значение свойства и установить новое для элемента из группы элементов в случае, когда нет прямого доступа к файлу со стилями.

Порядок подключённых таблиц. Вы можете создать несколько внешних таблиц стилей и подключить их к одной веб-странице. Если в разных таблицах будут встречаться разные значения свойств одного элемента, то в результате к элементу применится правило, находящееся в таблице стилей, идущей в списке ниже.

**5. Порядок выполнения работы**

1. В файле style.css определите цвет фона и текста документа, свойства полос прокрутки.

2. Подключите файл style.css к документу Lab5.html (методом связывания).

3. Выполните задание по варианту. Варианты представлены в таблице 6.2.

Таблица 6.2 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| **1** | **2** |
| 1 | 1. Сделайте все абзацы <p> красного цвета. 2. Сделайте первый на странице абзац <p> зеленого цвета. 3. Сделайте все <h2> шириной 300px. 4. Поставьте все <h2> по правому краю. 5. Сделайте первую букву <h2> курсивом. 6. Сделайте для абзацев <p> шрифт Arial. 7. Сделайте красную строку в абзацах 30px. 8. Сделайте селектор, который выберет все ссылки из id=test, с состояния link и visited сделайте неподчеркнутыми и красными, а состояние hover – подчеркнутым и голубым. |

Продолжение таблицы 6.2

|  |  |
| --- | --- |
| 2 | 1. Сделайте все <h1> зеленого цвета. 2. Сделайте второй на странице абзац <p> красного цвета. 3. Сделайте первый <h2> шириной 300px. 4. Сделайте так, чтобы во втором абзаце <p> текст был выровнен по центру. 5. Сделайте первую букву <h2> курсивом. 6. Сделайте для <h2> шрифт Times New Roman. 7. Для второго абзаца <p> уберите красную строку. 8. Сделайте селектор, который выберет все ссылки с классом www, состояния link и visited сделайте подчеркнутыми и голубыми, а состояние hover - неподчеркнутым. |
| 3 | 1. Сделайте все <h2> голубого цвета. 2. Сделайте третий на странице абзац <p> желтого цвета. 3. Сделайте последний <h2> шириной 300px. 4. Поставьте все <h1> по центру. 5. Сделайте первую букву <h2> курсивом. 6. Сделайте для <h3> шрифт Arial. 7. Сделайте красную строку в абзацах 30px. 8. Сделайте селектор, который выберет все ссылки из id=test с классом www. Цвета состояний выберите самостоятельно. |
| 4 | 1. Сделайте все <h3> оранжевого цвета. 2. Сделайте последний на странице абзац <p> синего цвета. 3. Сделайте третий <h2> шириной 300px. 4. Сделайте так, чтобы текст в абзацах <p> был выровнен одновременно и по правому и по левому краю. 5. Сделайте первую букву <h2> курсивом. 6. Сделайте для абзацев <p> шрифт Arial. 7. Для второго абзаца <p> уберите красную строку. 8. Сделайте селектор, который выберет все ссылки из class=eee с классом www. Цвета состояний выберите самостоятельно. |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что такое псевдокласс?

2. Перечислите псевдоклассы, которые вы использовали в лабораторной работе.

3. Как создаются селекторы потомка и дочерние селекторы? Для чего они используются?

4. Опишите, как влияет специфичность селекторов при оформлении внешнего вида веб-документа.

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
5. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 7**

## **Тема работы: «Позиционирование элементов при помощи CSS в web-документе»**

**1. Цель работы**

Формирование умений позиционирования HTML элементов средствами CSS.

**2. Задание**

Выполнить размещение HTML-элементов на веб-странице средствами CSS.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Поток – это порядок отображения элементов на странице. По умолчанию блочные элементы отображаются как прямоугольные области, идущие друг за другом сверху вниз, а строчные элементы располагаются сверху вниз и слева направо и при необходимости переносятся на новую строку.

Сетка – это взаимное расположение крупных блоков сайта.

Подавляющее большинство сайтов создано с использованием сеточных макетов. Они могут не использовать их в явном виде, но если на сайте присутствует блок с основным контентом, расположенный справа, и боковой блок (сайдбар), расположенный слева, то это и есть простейшая сетка.

Display – многоцелевое свойство, которое определяет, как элемент должен быть показан в документе.

display: block | inline | inline-block | inline-table | list-item | none | run-in | table | table-caption | table-cell | table-column-group | table-column | table-footer-group | table-header-group | table-row | table-row-group

Список возможных значений этого свойства, понимаемый разными браузерами очень короткий – block, inline, list-item и none. Все остальные допустимые значения поддерживаются браузерами выборочно.

CSS-позиционирование позволяет указать, где появится блок элемента.

Существуют специальные свойства для позиционирования, применяя которые можно легко установить элементы в нужные места страницы. При этом элементы могут накладываться друг на друга.

Свойство position позволяет точно задать новое местоположение блока относительно того места, где он находился бы в нормальном потоке документа. По умолчанию все элементы располагаются последовательно один за другим в том порядке, в котором они определены в структуре HTML-документа. Свойство не наследуется.

Свойство position определяет, как элемент позиционируется на странице и будет ли он отображаться в обычном потоке документа. Оно применяется в сочетании со свойствами смещения блока – top, right, bottom и left, которые точно определяют, где элемент будет расположен путём перемещения элемента в разных направлениях.

Значения свойства position представлены в таблице7.1.

Таблица 7.1 – Значения свойства position

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| static | Значение по умолчанию, означает отсутствие позиционирования. Элементы отображаются последовательно один за другим в том порядке, в котором они определены в HTML-документе. Используется для очистки любого другого значения позиционирования. |
| relative | Относительно позиционированный элемент сдвигается со своего обычного места в разных направлениях относительно границ родительского контейнера, а пространство, которое он занимал, не исчезает. При этом такой элемент может перекрывать другое содержимое на странице.  Если для относительно позиционированного элемента одновременно задать свойства top и bottom или left и right, то в первом случае сработает только top, во втором — left.  Относительное позиционирование позволяет задавать z-index для элемента, а также абсолютно позиционировать дочерние элементы внутри блока. |
| absolute | Абсолютно позиционированный элемент полностью удаляется из потока документа и позиционируется относительно границ его блока-контейнера (другого элемента или окна браузера). Блок-контейнер для абсолютно позиционированного элемента — ближайший элемент-предок, значение свойства position которого не равно static.  Местоположение краёв элемента определяется с помощью свойств смещения. Пространство, которое занимал такой элемент, схлопывается, как будто элемента не существовало на странице. Абсолютно позиционированный элемент может перекрывать другие элементы или быть перекрытым ими (за счёт свойства z-index). Любой абсолютно позиционированный элемент генерирует блок, то есть принимает значение display: block;. |
| fixed | Фиксирует элемент в указанном месте страницы. Блоком-контейнером фиксированного элемента является окно просмотра, то есть элемент всегда фиксируется относительно окна браузера и не меняет своего положения во время прокрутки страницы. Сам элемент при этом полностью удаляется из основного потока документа и создаётся в новом потоке документа. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

По умолчанию у каждого элемента значение position установлено как static, это означает, что элемент существует в обычном потоке документа и не принимает какие-либо свойства для его смещения.

Значение relative для свойства position позволяет элементам отображаться в обычном потоке страницы, резервируя место для элемента как предполагалось и не позволяя другим элементам его обтекать. Однако, оно также позволяет модифицировать положение элемента с помощью свойств смещения.

Пример использования значения relative представлен в таблице 7.2.

Таблица 7.2 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| HTML  <div>...</div>  <div class="offset">...</div>  <div>...</div>  CSS  div {  height: 100px;  width: 100px;  }  .offset {  left: 20px;  position: relative;  top: 20px;  } |  |

Здесь для второго элемента <div> с классом offset задано значение position как relative, а также два свойства смещения – left и top. Это сохраняет исходное положение элемента и другим элементам не разрешено двигаться в эту область. Кроме того, свойства смещения перемещают элемент, выталкивая его на 20 пикселей от левого и на 20 пикселей от верхнего исходного местоположения.

Когда позиционирование элемента выполнеятся с помощью свойств смещения, элемент перекрывает элемент под ним, а не сдвигает его вниз, как это делают свойства margin или padding.

Значение absolute для свойства position отличается от значения relative тем, что элемент с абсолютным позиционированием не появляется в обычном потоке документа, исходное пространство и положение абсолютно позиционируемого элемента не резервируется.

Кроме того, абсолютно позиционируемые элементы перемещаются относительно их ближайшего относительно позиционированного родительского элемента. Если относительно позиционированного родителя не существует, то абсолютно позиционированный элемент будет позиционироваться относительно элемента <body>.

Пример абсолютного позиционирования представлен в таблице 7.3.

Таблица 7.3 – Пример абсолютного позиционирования

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| HTML  <section>  <div class="offset">...</div>  </section>  CSS  section {  position: relative;  }  div {  position: absolute;  right: 20px;  top: 20px;  } |  |

В этом примере элемент <section> позиционируется относительно, но не включает каких-либо свойств смещения. Следовательно, его положение не меняется. Элемент <div> с классом offset включает значение position как absolute. Поскольку элемент <section> является ближайшим относительно позиционированным родительским элементом для <div>, то элемент <div> будет позиционироваться относительно элемента <section>.

Ширина элемента с position: absolute устанавливается по содержимому.

Для относительно позиционированных элементов свойства смещения определяют, в каком направлении элемент будет перемещён относительного самого себя. Для абсолютно позиционируемых элементов свойства смещения определяют, в каком направлении элемент будет перемещён относительно его ближайшего относительно позиционированного родителя.

Поскольку элемент <div> позиционируются абсолютно, он не располагается в обычном потоке страницы и будет перекрывать любые окружающие его элементы. Кроме того, исходное положение <div> не сохраняется и другие элементы могут занять это место.

Значение fixed позиционирует объект точно так же, как absolute, но относительно window.

Когда страницу прокручивают, фиксированный элемент остается на своем месте и не прокручивается вместе со страницей.

<style>

#top {

position: fixed;

right: 10px;

top: 10px;

background: #fee;

}

</style>

<a href="#" id="top">Наверх (остается при прокрутке)</a>

Фиксированного позиционирование

<p>Текст страницы.. Прокрути меня...</p>

<p>Много строк..</p><p>Много строк..</p>

<p>Много строк..</p><p>Много строк..</p>

<p>Много строк..</p><p>Много строк..</p>

<p>Много строк..</p><p>Много строк..</p>

В примере, при прокрутке документа, ссылка #top всегда остается на своем месте, рисунок 7.1.

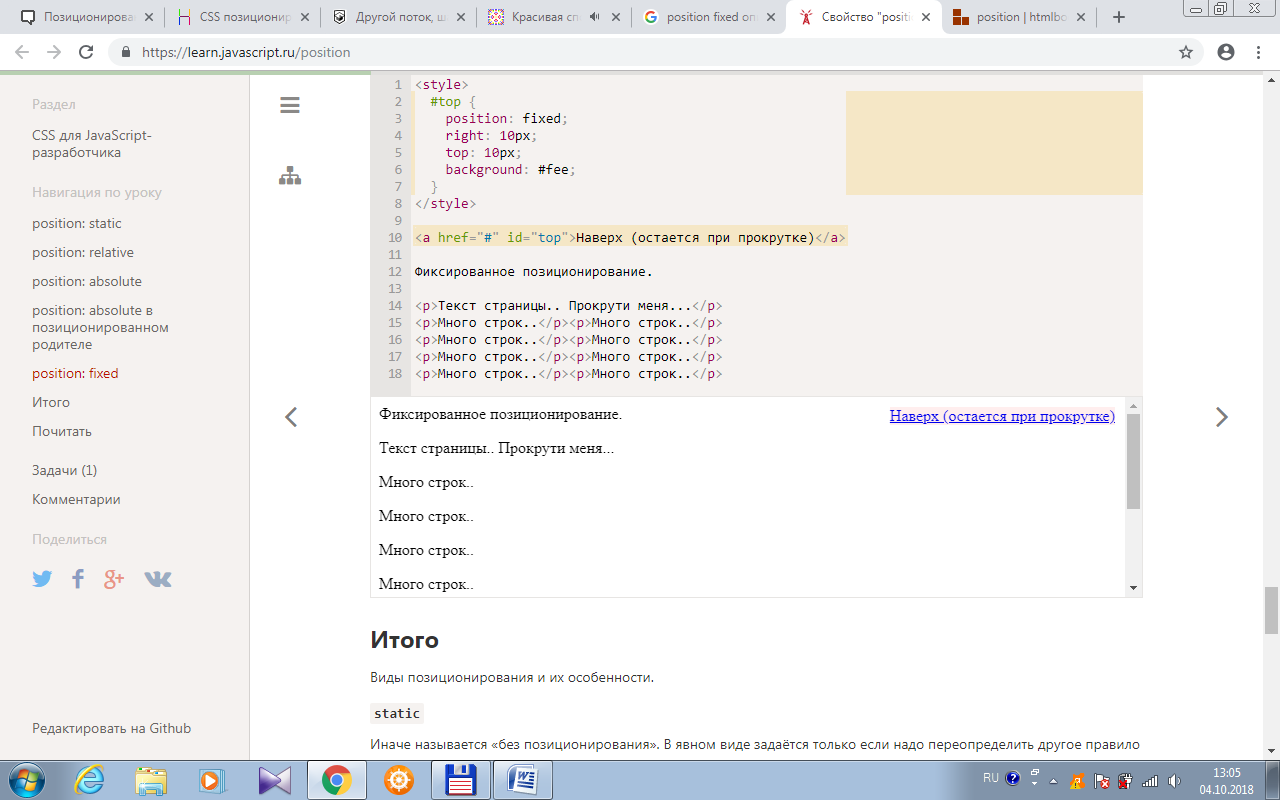


Рисунок 7.1 – Пример реализации свойства fixed

Обтекание (свободное перемещение элементов). Свойство float позволяет перемещать любой элемент, выравнивая его по левому или правому краю веб-страницы или содержащего его элемента-контейнера. При этом остальные блочные элементы будут его игнорировать, а строчные элементы будут смещаться вправо или влево, освобождая для него пространство и обтекая его.

Например, абзацы будут обтекать изображение, если для элемента <img> установлено свойство float.

Когда свойство float применяется к нескольким элементам одновременно, это даёт возможность создать макет с обтекаемыми элементами расположенными рядом или напротив друг друга.

Верхние и нижние отступы margin плавающих элементов не схлопываются. Свойство float применяется как к блочным элементам, так и к строчным элементам.

Левый или правый внешний край перемещаемого элемента, в отличие от позиционированных элементов, не может располагаться левее (или правее) внутреннего края его блока-контейнера, т.е. выходить за его границы. При этом, если для блока-контейнера заданы внутренние отступы, то плавающий блок будет отстоять от края блока-контейнера на заданное расстояние.

Значения свойства float представлены в таблице 7.4.

Таблица 7.4 – Значения свойства float

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| none | Значение по умолчанию. Также отменяет любое перемещение для элемента из группы элементов, для которых уже установлено обтекание. |
| left | Элемент изымается из нормального потока элементов и позиционируется по левому краю блока-контейнера. |
| right | Элемент позиционируется по правому краю блока-контейнера. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

При использовании свойства float для блочных элементов обязательно задавать ширину. Тем самым браузер создаст место для другого содержимого. Но если совокупная ширина всех столбцов окажется больше доступного места, то последний элемент спустится вниз.

Создадим общий макет страницы с шапкой вверху, двумя колонками в центре и подвалом внизу.

Пример использования свойства float показан в таблице 7.5.

Таблица 7.5 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| <header>...</header>  <section>...</section>  <aside>...</aside>  <footer>...</footer> |  |

Установив float для <section> как left, а для <aside> как right, можно позиционировать их как две колонки, расположенные напротив друг друга.

section {

float: left;

}

aside {

float: right;

}

Когда мы устанавливаем элемент обтекаемым, то убираем его из обычного потока веб-документа. Это приводит к тому, что ширина этого элемента по умолчанию становится шириной его содержимого. Иногда, например, когда мы создаём колонки для многократно используемого макета, такое поведение нежелательно. Это можно исправить путём добавления свойства width с фиксированным значением для каждой колонки.

Чтобы обтекаемые элементы не соприкасались друг с другом, можно использовать свойство margin, чтобы установить пространство между элементами.

section {

float: left;

margin: 0 1.5%;

width: 63%;

}

aside {

float: right;

margin: 0 1.5%;

width: 30%;

}

В случае двухколоночного макетаможно установить float, для одной колонки как left, а для другой как right, но для нескольких колонок необходимо изменить подход. К примеру, мы хотели бы получить ряд из трёх колонок между нашими элементами <header> и <footer>. Если мы выбросим наш элемент <aside> и воспользуемся тремя элементами <section>, наш HTML может выглядеть следующим образом:

<header>...</header>

<section>...</section>

<section>...</section>

<section>...</section>

<footer>...</footer>

Чтобы расположить эти три элемента <section> в строку из трёх колонок, мы должны задать float для всех элементов <section> как left. Мы также должны настроить ширину <section> с учётом дополнительных колонок и расположить их один рядом с другим.

Пример трехколоночного макета представлен в таблице 7.6.

Таблица 7.6 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| section {  float: left;  margin: 0 1.5%;  width: 30%;  } |  |

Здесь три колонки, все с равной шириной и значением margin, а также с float, заданным как left.

Свойство автоматически изменяет вычисляемое (отображаемое в браузере) значение свойства display на display: block для следующих значений: inline, inline-block, table-row, table-row-group, table-column, table-column-group, table-cell, table-caption, table-header-group, table-footer-group. inline-table меняет на table.

Например, элемент, у котороо display указан как inline, такой как строчный <span>, игнорирует любые свойства height или width. Однако, если для строчного элемента указать float, значение display изменится на block и тогда элемент уже может принимать свойства height или width.

Когда мы применяем float для элемента, то должны следить за тем, как это влияет на значение свойства display.

Свойство не оказывает никакого влияния на элементы с display: flex и display: inline-flex.

Свойство clear определяет, как будет располагаться элемент, идущий следом за плавающим элементом. Свойство отменяет обтекание с одной или обоих сторон элемента, установленное свойством float. Для предотвращения отображение фона или границ под плавающими элементами используется правило {overflow: hidden;}.

Значения свойства сlear представлены в таблице 7.7.

Таблица 7.7 – Значения свойства сlear

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **1** | **2** |
| left | Отменяет обтекание по левой стороне, смещая элемент вниз. Правое обтекание остается в силе. |
| right | Отменяет обтекание по правой стороне, смещая элемент вниз. Левое обтекание остается в силе. |
| both | Отменяет обтекание с обеих сторон элемента, смещая его вниз. |
| none | Значение по умолчанию. Также отменяет очистку обтекания, установленное для элементов одной группы. |

Продолжение таблицы 7.7

|  |  |
| --- | --- |
| **1** | **2** |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

В предыдущем примере с двумя колонками, после того как мы добавили float к элементам <section> и <aside>, но до того как установили свойство width к любому из них, содержимое внутри элемента <footer> располагалось между двумя обтекаемыми элементами выше него, заполняя всё доступное пространство. Следовательно, элемент <footer> находился бы в промежутке между элементами <section> и <aside>, занимая свободное место, как показано на рисунке 7.2.

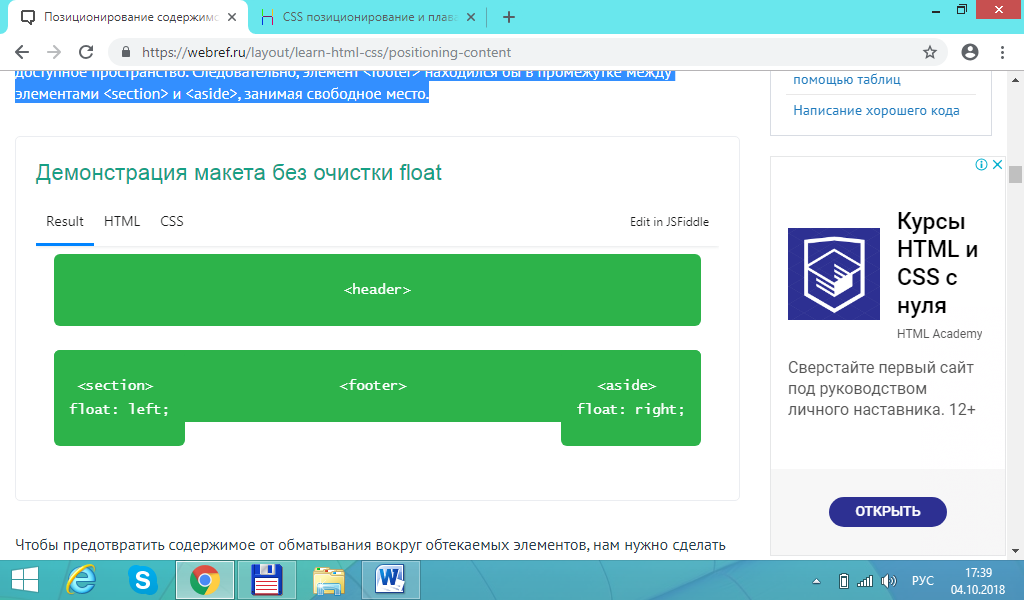


Рисунок 7.2 – Пример реализации свойства float

Чтобы предотвратить содержимое от обматывания вокруг обтекаемых элементов, нужно сделать очистку float и вернуть страницу в её обычный поток.

div {

clear: left;

}

Значение left очищает левые float, в то время как значение right очищает правые float. Значение both, однако, очистит левые и правые float и часто является наиболее идеальным вариантом. Важно, что clear применяется к элементу, указанному после обтекаемых элементов, а не раньше, чтобы вернуть страницу в её обычный поток.

Пример очистки обтекания представлен в таблице 7.8.

Таблица 7.8 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| footer {  clear: both;  } |  |

Вместо очистки float, ещё одним вариантом является установка содержимого float. Результат получится почти такой же, однако содержимое float действительно гарантирует, что все наши стили будут отображаться надлежащим образом.

Чтобы установить содержимое float, обтекаемые элементы должны находиться внутри родительского элемента, он будет выступать в качестве контейнера, оставляя поток документа полностью обычным за его пределами.

Пример: имеется блок-контейнер, для которого не заданы ширина и высота. Внутри него помещен плавающий блок с заданными размерами.

Пример установки содержимого float представлен в таблице 7.9.

Таблица 7.9 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| <div class="container">  <div class="floatbox">  </div>  </div>  .container {  padding: 20px;  background: #e7e6d4;  border: 3px dashed #645a4e;  }  .floatbox {  float: left;  width: 300px;  height: 150px;  margin-right: 20px;  padding: 0 20px;  background: #ffffff;  border: 3px dashed #666666;  } |  |

Создаем класс .clearfix и в сочетании с псевдоклассом :after применяем его к блоку-контейнеру, как показано в таблице 7.10.

Таблица 7.10 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| **1** | **2** |
| <div class="container clearfix">  <div class="floatbox">  </div>  </div>  .container {  padding: 20px;  background: #e7e6d4; |  |

Продолжение таблицы 7.10

|  |  |
| --- | --- |
| **1** | **2** |
| border: 3px dashed #645a4e;}  .floatbox {  float: left;  width: 300px;  height: 150px;  margin-right: 20px;  padding: 0 20px;  background: #ffffff;  border: 3px dashed #666666;  }  .clearfix:after {  content: "";  display: block;  height: 0;  clear: both;  visibility: hidden;  }  Второй вариант очистки потока:  .clearfix:after {  content: "";  display: table;  clear: both; } |  |

Псевдоэлементы :before и :after позволяют добавлять содержимое (стили) до и после элемента, к которому были применены, как показано в таблице 7.11.

Всего существует несколько типов псевдоэлементов: :first-line, :first-letter, ::selection, :before и :after.

Таблица 7.11 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| **1** | **2** |
| HTML  <header>...</header>  <div class="group">  <section>...</section>  <aside>...</aside>  </div>  <footer>...</footer>  CSS  .group::before,  .group::after {  content: "";  display: table;}  .group::after {  clear: both;} |  |

Продолжение таблицы 7.11

|  |  |
| --- | --- |
| **1** | **2** |
| .group {  clear: both;  \*zoom: 1;  }  section {  float: left;  margin: 0 1.5%;  width: 63%;  }  aside {  float: right;  margin: 0 1.5%;  width: 30%;  } |  |

Показанная здесь техника известна как «clearfix» и часто встречается на других сайтах с именем класса clearfix или cf.

Существует ещё один прием очистки потока для элемента, содержащего плавающие элементы, например, для маркированного горизонтального списка, как показано в таблице 7.12.

Таблица 7.12 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| <ul>  <li></li>  <li></li>  <li></li>  </ul>  ul {  margin: 0;  list-style: none;  padding: 20px;  background: #e7e6d4;  overflow: auto;  }  li {  float: left;  width: calc(100% / 3 - 20px);  height: 50px;  margin-right: 20px;  background: #ffffff;  border: 3px dashed #666666;  }  li:last-child {margin-right: 0;} |  |

Позиционировать контент можно с использованием свойства display в сочетании со значением inline-block.

Метод с inline-block полезен для компоновки страниц или для размещения элементов в линию рядом друг с другом.

Значение inline-block для свойства display отображает элементы в линию и позволяет им принимать все свойства блочной модели, включая height, width, padding, border и margin. Применение inline-block позволяет в полной мере воспользоваться блочной моделью, не беспокоясь об очистке каких-либо float.

<header>...</header>

<section>...</section>

<section>...</section>

<section>...</section>

<footer>...</footer>

Для трёх элементов <section> изменим значение display на inline-block, рисунок 7.3.



Рисунок 7.3 – Применение свойства inline-block

Однако, случаются ситуации, когда последний элемент <section> выталкивается на новую строку. Поскольку строчно-блочные элементы отображаются на одной линии друг за другом, они включают единое пространство между ними. Когда размер каждого отдельного пространства добавляется к ширине и значению горизонтального margin всех элементов в строке, общая ширина становится слишком большой, выталкивая последний элемент <section> на новую строку. Чтобы отобразить все элементы <section> на одной строке, следует удалить пустое пространство между каждым <section>.

Длинные строки текста читать очень неудобно или даже трудно. Один из способов справиться с данным вопросом – отцентровать содержание страницы. Вместо того, чтобы растягивать контент на всю ширину экрана, центрированные дизайны занимают только часть экрана, благодаря чему строки приобретает удобную для чтения длину.

Существует два основных способа центрации дизайна: в одном случае используются автоматические внешние отступы, в другом – позиционирование и отрицательные отступы.

Пример центрирования с использованием автоматических внешних отступов: отцентровать по горизонтали контейнер div с id wrapper ("обертка").

<body>

<div id="wrapper">

</div>

</body>

Для осуществления этого на практике нужно просто задать ширину div wrapper и установить горизонтальные внешние отступы в значение auto.

#wrapper {

width: 720px;

margin: 0 auto;

background: #F2EFE6

}

В данном примере ширина установлена в пикселях. Но, разумеется, с таким же успехом можно указать ее в процентах от ширины тела документа body, либо использовать размер в "em" относительно размера текста.

CSS flexbox (Flexible Box Layout Module) – модуль макета гибкого контейнера – представляет собой способ компоновки элементов и позволяет контролировать размер, порядок и выравнивание элементов по нескольким осям, распределение свободного места между элементами и многое другое.

Flexbox состоит из flex-контейнера – родительского контейнера и flex-элементов – дочерних блоков. Дочерние элементы могут выстраиваться в строку или столбик, а оставшееся свободное пространство распределяется между ними различными способами.

Flex-контейнер устанавливает новый гибкий контекст форматирования для его содержимого. Flex-контейнер не является блочным контейнером, поэтому для внутренних блоков не работают такие CSS-свойства, как float, clear, vertical-align. Также, на flex-контейнер не оказывают влияние свойства column-, создающие колонки в тексте и псевдоэлементы ::first-line и ::first-letter.

Модель flexbox-разметки связана с определенным значением CSS-свойства display родительского html-элемента, содержащего внутри себя дочерние блоки.

Для управления элементами с помощью этой модели нужно установить свойство display следующим образом:

.flex-container {

display: -webkit-flex;

display: flex; /\*отображает контейнер как блочный элемент\*/

}

.flex-container {

display: -webkit-inline-flex;

display: inline-flex; /\*отображает контейнер как строчный элемент\*/

}

После установки данных значений свойства каждый дочерний элемент автоматически становится flex-элементом, выстраиваясь в ряд (вдоль главной оси) колонками одинаковой высоты, равной высоте блока-контейнера. При этом блочные и строчные дочерние элементы ведут себя одинаково, т.е. ширина блоков равна ширине их содержимого с учетом внутренних полей и рамок элемента, таблица 7.13.

Таблица 7.13 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| HTML  <div class="my-flex-container">  <div class="my-flex-block">item1</div>  <div class="my-flex-block">item2</div>  <div class="my-flex-block">item3</div>  </div>  CSS  .my-flex-container{  display: flex;  } | display-flex |

Свойство flex-direction позволяет задавать, каким образом flex-элементы укладываются во flex-контейнере, задавая направление главной оси flex-контейнера.

Они могут располагаться в двух главных направлениях – горизонтально, как строки или вертикально, как колонки. Свойство не наследуется. Значения свойства flex-direction представлены в таблице 7.14.

Таблица 7.14 – Значения свойства flex-direction

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| row | Значение по умолчанию, слева направо (в rtl справа налево). Flex-элементы выкладываются в строку. Начало (main-start) и конец (main-end) направления главной оси соответствуют началу (inline-start) и концу (inline-end) инлайн оси (inline-axis). |
| row-reverse | Направление справа налево (в rtl слева направо). Flex-элементы выкладываются в строку относительно правого края контейнера (в rtl — левого).  flex-direction |
| column | Направление сверху вниз. Flex-элементы выкладываются в колонку. flex-direction |
| column-reverse | Колонка с элементами в обратном порядке, снизу вверх. flex-direction |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

-webkit-flex-direction: row-reverse;

display: flex;

flex-direction: row-reverse;}

Свойство justify-content выравнивает flex-элементы по ширине flex-контейнера, распределяя оставшееся свободное пространство, незанятое flex-элементами. Для выравнивания элементов по вертикали используется свойство align-content. Свойство не наследуется. Значения свойства justify-content представлены в таблице 7.15.

Таблица 7.15 – Значения свойства justify-content

|  |  |
| --- | --- |
| **Значение** | **Свойство** |
| flex-start | Значение по умолчанию. Flex-элементы позиционируются от начала flex-контейнера. justify-content |
| flex-end | Flex-элементы позиционируются относительно правой границы flex-контейнера. justify-content |
| center | Flex-элементы выравниваются по центру flex-контейнера. justify-content |
| space-between | Flex-элементы выравниваются по главной оси, свободное место между ними распределяется следующим образом: первый блок располагается в начале flex-контейнера, последний блок – в конце, все остальные блоки равномерно распределены в оставшемся пространстве, а свободное пространство равномерно распределяется между элементами. justify-content |
| space-around | Flex-элементы выравниваются по главной оси, а свободное место делится поровну, добавляя отступы справа и слева. justify-content |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

-webkit-justify-content: flex-start;

display: flex;

justify-content: flex-start;

}

Свойство align-items выравнивает flex-элементы, в том числе и анонимные flex-элементы по перпендикулярной оси (по высоте). Не наследуется. Значения свойства align-items представлены в таблице 7.16.

Таблица 7.16 – Значения свойства align-items

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| stretch | Значение по умолчанию. Flex-элементы растягиваются, занимая все пространство по высоте. align-items |
| flex-start | Flex-элементы выравниваются по левому краю flex-контейнера относительно верхнего края блока-контейнера. align-items |
| flex-end | Flex-элементы выравниваются по левому краю flex-контейнера относительно нижнего края блока-контейнера. align-items |
| center | Flex-элементы выравниваются по центру flex-контейнера. align-items |
| baseline | Flex-элементы выравниваются по базовой линии. align-items |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

-webkit-align-items: flex-start;

display: flex;

align-items: flex-start;

}

СSS свойства flex-direction, justify-content, align-items должны применяться непосредственно к flex-контейнеру, а не к его дочерним элементам.

По умолчанию flex-контейнер всегда будет располагать блоки внутри себя в одну линию. Однако, спецификацией также поддерживается многострочный режим.

Свойство flex-wrap управляет тем, как flex-контейнер будет выкладывать flex-элементы – в одну строку или в несколько, и направлением, в котором будут укладываться новые строки.

По умолчанию flex-элементы укладываются в одну строку. При переполнении контейнера их содержимое будет выходить за границы flex-элементов. Не наследуется. Значения свойства flex-wrap представлены в таблице 7.17.

Таблица 7.17 – Значения свойства flex-wrap

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| nowrap | Значение по умолчанию. Flex-элементы не переносятся, а располагаются в одну линию слева направо (в rtl справа налево). |
| wrap | Flex-элементы переносятся, располагаясь в несколько горизонтальных рядов (если не помещаются в один ряд) в направлении слева направо (в rtl справа налево). flex-wrap |
| wrap-reverse | Flex-элементы переносятся, располагаясь в обратном порядке слева-направо, при этом перенос происходит снизу вверх. flex-wrap |
| Initial | Устанавливает значение свойства в значение по умолчанию. |
| Inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

-webkit-flex-wrap: wrap;

display: flex;

flex-wrap: wrap;

}

Свойство flex-flow предоставляет возможность в одном свойстве задать направление главной оси и многострочность поперечной оси, т.е. сокращённая запись свойств flex-direction и flex-wrap:

flex-flow: <‘flex-direction’> || <‘flex-wrap’>

Не наследуется. Значение по умолчанию: flex-flow: row nowrap. Значения свойства flex-flow представлены в таблице 7.18.

Таблица 7.18 – Значения свойства flex-flow

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| направление | Указывает направление главной оси. Значение по умолчанию row. |
| многострочность | Задаёт многострочность поперечной оси. Значение по умолчанию nowrap. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

-webkit-flex-flow: row wrap;

display: flex;

flex-flow: row wrap;

}

.my-flex-block{

flex-direcrion:column;

flex-wrap: wrap;

}

/\* это то же самое, что ... \*/

.my-flex-block{

flex-flow: column wrap;

}

Свойство align-content выравнивает строки flex-элементов по вертикали во flex-контейнере, позволяя управлять свободным пространством.

Свойство работает только в случае, если разрешен перенос строк (т.е. в случае flex-wrap:wrap; или flex-wrap:wrap-reverse;) и указано направление (flex-direction): flex-flow: row/row-reverse/column/column-reverse wrap/wrap-reverse; и высота flex-контейнера. Не наследуется. Значения свойства align-content представлены в таблице 7.19.

Таблица 7.19 – Значения свойства align-content

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| stretch | Значение по умолчанию. Строки flex-элементов равномерно растягиваются, заполняя все доступное пространство. |
| flex-start | Строки flex-элементов выравниваются по левому краю flex-контейнера относительно верхнего края блока-контейнера. |
| flex-end | Строки flex-элементов выравниваются по левому краю flex-контейнера относительно нижнего края блока-контейнера. |
| center | Строки flex-элементов выравниваются по высоте по середине flex-контейнера относительно его левого края. |
| space-between | Строки flex-элементов выравниваются по высоте по середине flex-контейнера относительно его левого края. Свободное пространство распределяется между ними. Первый ряд flex-элементов прижимается к началу flex-контейнера, последний ряд — к нижнему краю. |
| space-around | Строки flex-элементов равномерно распределяются по высоте, свободное пространство добавляется сверху и снизу строки. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

-webkit-flex-flow: row wrap;

-webkit-align-content: flex-end;

display: flex;

flex-flow: row wrap;

align-content: flex-end;

height: 100px;

}

СSS свойства flex-wrap и align-content должны применяться непосредственно к flex-контейнеру, а не к его дочерним элементам.

Кроме того, CSS правила предусмотрены для дочерних элементов flex-контейнера (flex-блоков).

Свойство order определяет порядок, в котором flex-элементы отображаются внутри flex-контейнера. По умолчанию для всех flex-элементов задан порядок order: 0; и они следуют друг за другом по мере добавления во flex-контейнер. Самый первый flex-элемент по умолчанию расположен слева.

Свойство позволяет изменить порядок следования отдельно взятого flex-блока внутри flex-контейнера.

Чтобы поставить любой flex-элемент в начало строки, ему нужно назначить  order: -1;, в конец строки – order: 1;. Свойство не наследуется. Значения свойства order представлены в таблице 7.20.

Таблица 7.20 – Значения свойства order

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| число | Свойство задается целым числом, отвечающим за порядок отображения flex-элементов. Значение по умолчанию 0. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Пример использования свойства order представлен в таблице 7.21.

Таблица 7.21 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| .flex-container {  display: -webkit-flex;  display: flex;  }  .flex-item {  -webkit-order: 1;  order: 1;  } | order |

Значение order не задает абсолютную позицию элемента в последовательности. Оно определяет вес позиции элемента.

HTML

<div class="my-flex-container">

<div class="my-flex-block" style="order: 5" >item1</div>

<div class="my-flex-block" style="order: 10">item2</div>

<div class="my-flex-block" style="order: 5" >item3</div>

<div class="my-flex-block" style="order: 5" >item4</div>

<div class="my-flex-block" style="order: 0" >item5</div>

</div>

В данном случае, блоки будут следовать один за другим вдоль главной оси в следующем порядке: item5, item1, item3, item4, item2

Свойство flex-basis позволяет задать базовую ширину flex-элемента, относительно которой будет происходить растяжение flex-grow или сужение flex-shrink элемента. Не наследуется.

Т.е. задает изначальный размер по главной оси для flex-блока до того, как к нему будут применены преобразования, основанные на других flex-факторах. Может быть задан в любых единицах измерения длинны (px, em, %, …) или auto (по умолчанию). Если задан как auto – за основу берутся размеры блока (width, height), которые, в свою очередь, могут зависеть от размера контента, если не указанны явно. Значения свойства flex-basis представлены в таблице 7.22

Таблица 7.22 – Значения свойства flex-basis

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| auto | Значение по умолчанию. Элемент получает базовую ширину, соответствующую ширине контента внутри него, если она не задана явно. |
| число | Ширина элемента задается в px, %, em и других единицах измерения. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

display: flex;

}

.flex-item {

-webkit-flex-basis: 100px;

flex-basis: 100px;

}

Свойство flex-grow определяет коэффициент увеличения ширины flex-элемента относительно других flex-элементов. Свойство не наследуется. Свойство определяет то, на сколько отдельный flex-блок может быть больше соседних элементов, если это необходимо. flex-grow принимает безразмерное значение ( по умолчанию 0).

Если все flex-блоки внутри flex-контейнера имеют flex-grow:1, то они будут одинакового размера. Если один из них имеет flex-grow:2, то он будет в 2 раза больше, чем все остальные.

Если все flex-блоки внутри flex-контейнера имеют flex-grow:3, то они будут одинакового размера. Если один из них имеет flex-grow:12, то он будет в 4 раза больше, чем все остальные.

Т.е. абсолютное значение flex-grow не определяет точную ширину. Оно определяет его степень “жадности” по отношению к другим flex-блокам того же уровня. Значения свойства flex-grow представлены в таблице 7.23.

Таблица 7.23 – Значения свойства flex-grow

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| число | Положительное целое или дробное число, устанавливающее коэффициент увеличения flex-элемента. Значение по умолчанию 0. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Пример использования свойства flex-grow представлен в таблице 7.24.

Таблица 7.24 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| .flex-container {  display: -webkit-flex;  display: flex;  }  .flex-item {  -webkit-flex-grow: 3;  flex-grow: 3;  } | flex-grow |

Свойство flex-shrink указывает коэффициент уменьшения ширины flex-элемента относительно других flex-элементов. Работает только если для элемента задана ширина с помощью свойства flex-basis или width. Свойство не наследуется. Свойство определяет, насколько flex-блок будет уменьшаться относительно соседних эдементов внутри flex-контейнера в случае недостатка свободного места. По умолчанию равен 1.

Значения свойства flex-shrink представлены в таблице 7.25.

Таблица 7.25 – Значения свойства flex-shrink

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| число | Положительное целое или дробное число, устанавливающее коэффициент уменьшения flex-элемента. Значение по умолчанию 1. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| Inherit | Наследует значение свойства от родительского элемента. |

Пример использования свойства flex-shrink представлен в таблице 7.26.

Таблица 7.26 – Пример реализации

|  |  |
| --- | --- |
| **Исходный код** | **Результат** |
| .flex-container {  display: -webkit-flex;  display: flex;  }  .flex-item {  -webkit-flex-shrink: 3;  flex-shrink: 3;  } | flex-shrink |

Свойство flex представляет собой сокращённую запись свойств flex-grow, flex-shrink и flex-basis. Значение по умолчанию: flex: 0 1 auto;. Можно указывать как одно, так и все три значения свойств. Свойство не наследуется.

flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]

W3C рекомендует использовать сокращённую запись, так как она правильно сбрасывает любые неуказанные компоненты, чтобы подстроиться под типичное использование.

.my-flex-block{

flex-grow:12;

flex-shrink:3;

flex basis: 30em;

}

/\* это то же самое, что ... \*/

.my-flex-block{

flex: 12 3 30em;

}

Значения свойства flex представлены в таблице 7.27.

Таблица 7.27 – Значения свойства flex

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| коэффициент растяжения | Коэффициент увеличения ширины flex-элемента относительно других flex-элементов. |
| коэффициент сужения | Коэффициент уменьшения ширины flex-элемента относительно других flex-элементов. |
| базовая ширина | Базовая ширина flex-элемента. |
| auto | Эквивалентно flex: 1 1 auto;. |
| none | Эквивалентно flex: 0 0 auto;. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

display: flex;

}

.flex-item {

-webkit-flex: 3 1 100px;

-ms-flex: 3 1 100px;

flex: 3 1 100px;

}

Свойство align-self отвечает за выравнивание отдельно взятого flex-элемента по поперечной оси. Позволяет переопределять выравнивание, заданное align-items для отдельного flex-блока. Не наследуется. Значения свойства align-self представлены в таблице 7.28.

Таблица 7.28 – Значения свойства align-self

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **1** | **2** |
| auto | Значение по умолчанию. Flex-элемент использует выравнивание, указанное в свойстве align-items flex-контейнера. |
| flex-start | Flex-элемент выравнивается по верхнему краю flex-контейнера, относительно левой границы. |
| flex-end | Flex-элемент выравнивается по нижнему краю flex-контейнера, относительно левой границы. |

Продолжение таблицы 7.28

|  |  |
| --- | --- |
| center | Flex-элемент выравнивается по высоте по середине flex-контейнера, относительно левой границы. |
| baseline | Flex-элемент выравнивается по базовой линии flex-контейнера, относительно левой границы. |
| stretch | Flex-элемент растягивается на всю высоту flex-контейнера, при этом учитываются поля и отступы. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

.flex-container {

display: -webkit-flex;

display: flex;

}

.flex-item {

-webkit-align-self: center;

align-self: center;

}

Для Flexbox свойство margin:auto – выравнивание по горизонтали работает и для вертикали.

.my-flex-container {

display: flex;

height: 300px; /\* Или что угодно \*/

}

.my-flex-block {

width: 100px; /\* Или что угодно \*/

height: 100px; /\* Или что угодно \*/

margin: auto; /\* Блок отцентрирован по вертикали и горизонтали! \*/

}

**5. Порядок выполнения работы**

1. В файле style.css определите цвет фона и текста документа.

2. На странице оформите блок, представленный на рисунке 7.4

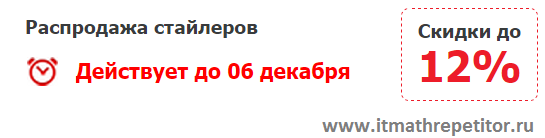


Рисунок 7.4 – Образец задания

3. Создайте на странице меню, по своему усмотрению выполните его стилистическое оформление и зафиксируйте его положение при прокрутке страницы, как показано на рисунке 7.5.

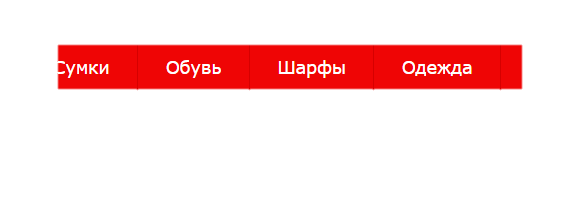


Рисунок 7.5 – Образец задания

4. Выполните задание по варианту всеми, известными Вам способами.

Вариант 1. Сверстать страницу, используя основные правила css, представленную на рисунке 7.6.

|  |  |  |
| --- | --- | --- |
| HEADER | | |
|  | CONTENT |  |
|  |
| FOOTER | | |

Рисунок 7.6 – Задания для выполнения работы 1 варианта

Вариант 2. Сверстать страницу, используя основные правила css, представленную на рисунке 7.7

|  |  |  |
| --- | --- | --- |
| HEADER | | |
|  | CONTENT |  |
| FOOTER | | |

Рисунок 7.7 – Задания для выполнения работы 2 варианта

Вариант 3. Сверстать страницу, используя основные правила css, представленную на рисунке 7.8

|  |  |  |
| --- | --- | --- |
| HEADER | | |
| NAV | | |
|  | CONTENT |  |
| FOOTER | | |

Рисунок 7.8 – Задания для выполнения работы 3 варианта

Вариант 4. Сверстать страницу, используя основные правила css, представленную на рисунке 7.9

|  |  |
| --- | --- |
| HEADER | |
| NAV | |
|  | CONTENT |
| FOOTER | |

Рисунок 7.9 – Задания для выполнения работы 4 варианта

Вариант 5. Сверстать страницу, используя основные правила css, представленную на рисунке 7.10

|  |  |
| --- | --- |
| HEADER | |
| NAV | |
|  | CONTENT |
|  |
| FOOTER | |

Рисунок 7.10 – Задания для выполнения работы 5 варианта

Вариант 6. Сверстать страницу, используя основные правила css, представленную на рисунке 7.11

|  |  |  |
| --- | --- | --- |
| HEADER | | |
| NAV | | |
|  | CONTENT |  |
| FOOTER | | |

Рисунок 7.11 – Задания для выполнения работы 6 варианта

Вариант 7. Сверстать страницу, используя основные правила css, представленную на рисунке 7.12

|  |  |  |
| --- | --- | --- |
| HEADER | | |
| NAV | | |
|  | CONTENT |  |
|  |  |
| FOOTER | | |

Рисунок 7.12– Задания для выполнения работы 7 варианта

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Дайте характеристику технологии flexbox. Перечислите основные свойства. Приведите примеры.

2. Для чего используется свойство display?

3. Какие свойства используются для задания внешних и внутренних отступов?

4. Какое свойство используется для свободного перемещения элементов?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
5. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 8**

## **Тема работы: «Адаптивная верстка CSS. Применение возможностей адаптации верстки в зависимости от разрешения экрана и типа устройства.»**

**1. Цель работы**

Формирование умений адаптивной верстки HTML-документодля различных разрешений экранов и типов устройств.

**2. Задание**

Реализовать верстку адаптивной веб-страницы в соответствии с psd макетом.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Адаптивная верстка меняет дизайн страницы в зависимости от поведения пользователя, платформы, размера экрана и ориентации девайса и является неотъемлемой частью современной веб-разработки.

Одно из решений – использование max-width в CSS:

img {max-width: 100%;}

Максимальная ширина изображения равняется 100% от ширины экрана или окна браузера, поэтому чем меньше ширина, тем меньше картинка. Обратите внимание, что max-width не поддерживается в IE, поэтому используйте width: 100%.

Для значительных изменений размера страницы может понадобиться изменить расположение элементов в целом. Это удобно делать через отдельный файл со стилями или, что более эффективно, через CSS-медиазапрос.

Рассмотрим, как можно использовать CSS3-медиазапросы для создания адаптивного дизайна. Min-width задает минимальную ширину окна браузера или экрана, к которой будут применены определенные стили. Если какое-нибудь значение будет ниже min-width, то стили будут проигнорированы. Max-width делает противоположное.

Пример:

@media screen and (min-width: 600px) {

.hereIsMyClass {

width: 30%;

float: right; }

}

Медиазапрос заработает только когда min-width будет больше или равна 600 px.

@media screen and (max-width: 600px) {

.aClassforSmallScreens {

clear: both;

font-size: 1.3em; }

}

В этом случае класс (aClassforSmallscreens) будет работать при ширине экрана меньше или равной 600 px.

В общем случае медиа-запрос состоит из ключевого слова, описывающего тип устройства (необязательный параметр) и выражения, проверяющего характеристики данного устройства. Из всех характеристик чаще всего проверяется ширина устройства width. Медиа-запрос является логическим выражением, которое возвращает истину или ложь.

Медиа-запросы могут быть добавлены следующими способами:

1) c помощью HTML:

<link rel="stylesheet" media="screen and (color)" href="example.css">

2) c помощью правила @import внутри элемента <style> или внешней таблицы стилей:

@import url(color.css) screen and (color);

3) непосредственно в коде страницы:

<style>

@media (max-width: 600px) {

#sidebar {display: none;}

}

</style>

4) внутри таблицы стилей style.css:

@media (max-width: 600px) {

#sidebar {display: none;}}

Таблица стилей, прикрепленная через тег <link>, будет загружаться вместе с документом, даже если её медиа-запрос вернет ложь.

С помощью логических операторов можно создавать комбинированные медиазапросы, в которых будет проверяться соответствие нескольким условиям.

Оператор and связывает друг с другом разные условия:

@media screen and (max-width: 600px) {

/\* CSS-стили \*/;

}

Стили этого запроса будут применяться только для экранных устройств с шириной области просмотра не более 600px.

@media (min-width: 600px) and (max-width: 800px) {

/\* CSS-стили \*/;

}

Стили этого запроса будут применяться для всех устройств при ширине области просмотра от 600px до 800px включительно.

Следующие правила равнозначны:

@media all and (max-width: 600px) {...}

@media (max-width: 600px) {...}

Оператор запятая работает по аналогии с логическим оператором or.

@media screen, projection {

/\* CSS-стили \*/;

}

В данном случае CSS-стили, заключенные в фигурные скобки, сработают только для экранных или проекционных устройств.

Оператор not позволяет сработать медиазапросу в противоположном случае. Ключевое слово not добавляется в начало медиазапроса и применяется ко всему запросу целиком, т.е. запрос:

@media not all and (monochrome) {...}

будет эквивалентен запросу

@media not (all and (monochrome)) {...}

Если медиазапрос составлен с использованием оператора запятая, то отрицание будет распространяться только на ту часть, которая идет до запятой, т.е. запрос:

@media not screen and (color), print and (color)

будет эквивалентен запросу

@media (not (screen and (color))), print and (color)

Оператор only используется, чтобы скрыть стили от старых браузеров (поддерживающих синтаксис медиа-запросов CSS2).

media="only screen and (min-width: 401px) and (max-width: 600px)"

Эти браузеры ожидают список медиа-типов, разделённых запятыми. И, согласно спецификации, они должны отсекать каждое значение непосредственно перед первым неалфавитно-цифровым символом, который не является дефисом. Таким образом, старый браузер должен интерпретировать предыдущий пример как media="only". Поскольку данного типа медиа-типа не существует, то и таблицы стилей будут игнорироваться.

Тип носителя представляет собой тип устройства, например, принтеры, экраны. Более подробное описание представлено в таблице 8.1.

Таблица 8.1 – Типы устройств

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| all | Подходит для всех типов устройств. |
| print | Предназначен для страничных материалов и документов, просматриваемых на экране в режиме предварительного просмотра печати. |
| screen | Предназначен в первую очередь для экранов цветных компьютерных мониторов. |
| speech | Предназначен для синтезаторов речи. |

Для управления разметкой в мобильных браузерах используется метатег viewport. Изначально данный тег был представлен разработчиками Apple для браузера Safari на iOS. Мобильные браузеры отображают страницы в виртуальном окне просмотра, которое обычно шире, чем экран устройства. С помощью метатега viewport можно контролировать размер окна просмотра и масштаб.

Страницы, адаптированные для просмотра на разных типах устройств, должны содержать в разделе <head> метатег viewport:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

Свойство width определяет виртуальную ширину окна просмотра, значение device-width – физическую ширину устройства. Другими словами, width отражает значение document.documentElement.clientWidth, а device-width – screen.width. Разница между width и device-width представлена на рисунке 8.1



Рисунок 8.1 – Различия между width и device-width

При первой загрузке страницы свойство initial-scale управляет начальным уровнем масштабирования, initial-scale=1 означает, что 1 пиксель окна просмотра = 1 пиксель CSS.

При составлении медиазапросов нужно ориентироваться на так называемые переломные (контрольные) точки дизайна, т.е. такие значения ширины области просмотра, в которых дизайн сайта существенно меняется, например, появляется горизонтальная полоса прокрутки. Чтобы определить эти точки, нужно открыть сайт в браузере и постепенно уменьшать область просмотра.

/\* Smartphones (вертикальная и горизонтальная ориентация) ----------- \*/

@media only screen and (min-width : 320px) and (max-width : 480px) {

/\* стили \*/

}

/\* Smartphones (горизонтальная) ----------- \*/

@media only screen and (min-width: 321px) {

/\* стили \*/

}

/\* Smartphones (вертикальная) ----------- \*/

@media only screen and (max-width: 320px) {

/\* стили \*/

}

/\* iPads (вертикальная и горизонтальная) ----------- \*/

@media only screen and (min-width: 768px) and (max-width: 1024px) {

/\* стили \*/

}

/\* iPads (горизонтальная) ----------- \*/

@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: landscape) {

/\* стили \*/

}

/\* iPads (вертикальная) ----------- \*/

@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: portrait) {

/\* стили \*/

}

/\* iPad 3\*\*\*\*\*\*\*\*\*\*/

@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: landscape) and (-webkit-min-device-pixel-ratio: 2) {

/\* стили \*/

}

@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation: portrait) and (-webkit-min-device-pixel-ratio: 2) {

/\* стили \*/

}

/\* Настольные компьютеры и ноутбуки ----------- \*/

@media only screen and (min-width: 1224px) {

/\* стили \*/

}

/\* Большие экраны ----------- \*/

@media only screen and (min-width: 1824px) {

/\* стили \*/

}

/\* iPhone 4 ----------- \*/

@media only screen and (min-width: 320px) and (max-width: 480px) and (orientation: landscape) and (-webkit-min-device-pixel-ratio: 2) {

/\* стили \*/

}

@media only screen and (min-width: 320px) and (max-width: 480px) and (orientation: portrait) and (-webkit-min-device-pixel-ratio: 2) {

/\* стили \*/

}

/\* iPhone 5 ----------- \*/

@media only screen and (min-width: 320px) and (max-height: 568px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 2){

/\* стили \*/

}

@media only screen and (min-width: 320px) and (max-height: 568px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 2){

/\* стили \*/

}

/\* iPhone 6 ----------- \*/

@media only screen and (min-width: 375px) and (max-height: 667px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 2){

/\* стили \*/

}

@media only screen and (min-width: 375px) and (max-height: 667px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 2){

/\* стили \*/

}

/\* iPhone 6+ ----------- \*/

@media only screen and (min-width: 414px) and (max-height: 736px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 2){

/\* стили \*/

}

@media only screen and (min-width: 414px) and (max-height: 736px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 2){

/\* стили \*/

}

/\* Samsung Galaxy S3 ----------- \*/

@media only screen and (min-width: 320px) and (max-height: 640px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 2){

/\* стили \*/

}

@media only screen and (min-width: 320px) and (max-height: 640px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 2){

/\* стили \*/

}

/\* Samsung Galaxy S4 ----------- \*/

@media only screen and (min-width: 320px) and (max-height: 640px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 3){

/\* стили \*/

}

@media only screen and (min-width: 320px) and (max-height: 640px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 3){

/\* стили \*/

}

/\* Samsung Galaxy S5 ----------- \*/

@media only screen and (min-width: 360px) and (max-height: 640px) and (orientation: landscape) and (-webkit-device-pixel-ratio: 3){

/\* стили \*/

}

@media only screen and (min-width: 360px) and (max-height: 640px) and (orientation: portrait) and (-webkit-device-pixel-ratio: 3){

/\* стили \*/

}

Преимущество данной верстки состоит в том, что код получается более «чистым», а значит быстрее загружается и легче изменяется под специфические нужды.

В качестве примера выбран бесплатный psd шаблон [Corporate Blue](http://www.pcklab.com/item.php?id=16) от студии Pcklaboratory рисунок 8.2.

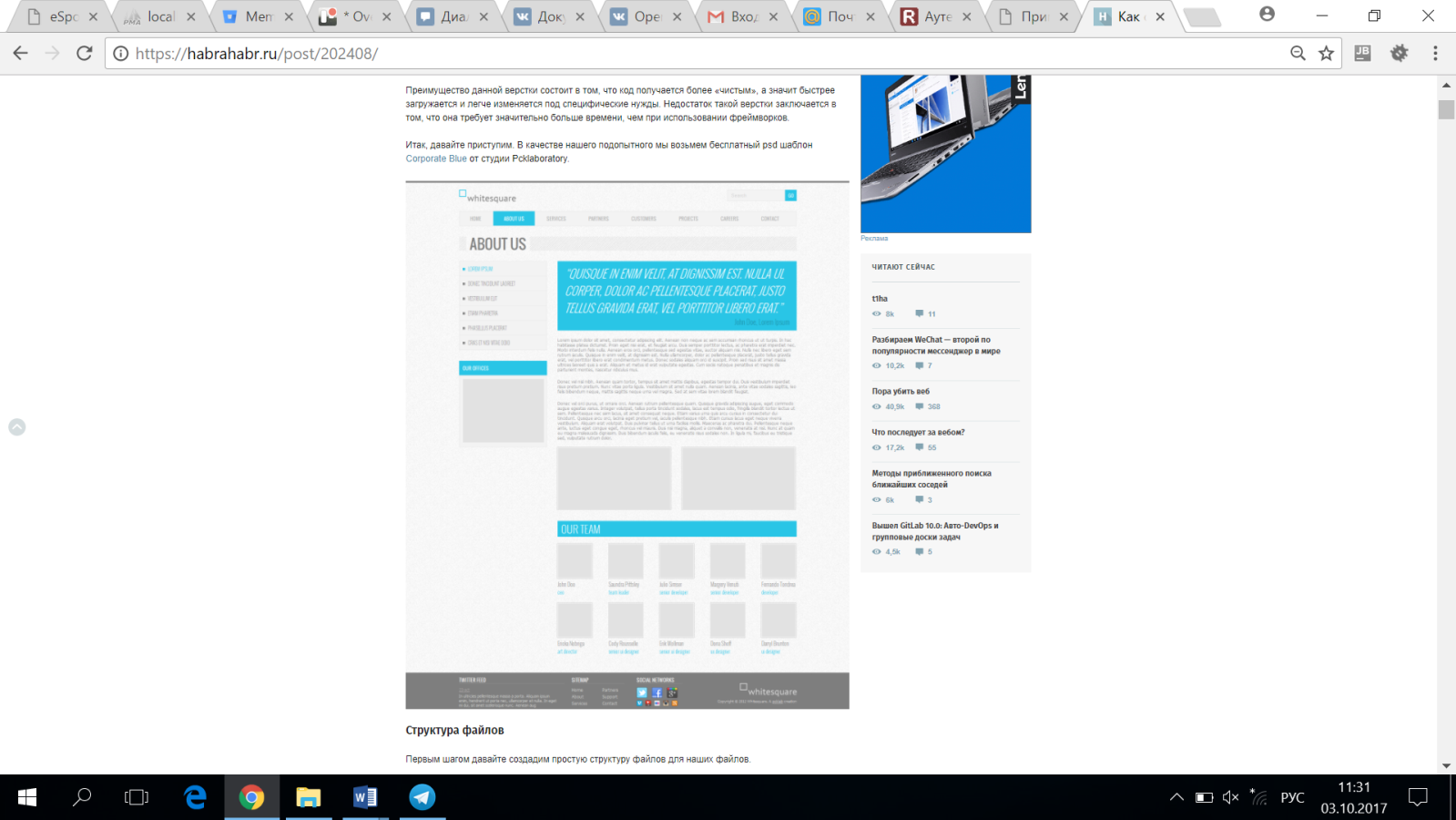


Рисунок 8.2 – PSD-макет

Структура файлов представлена на рисунке 8.3.

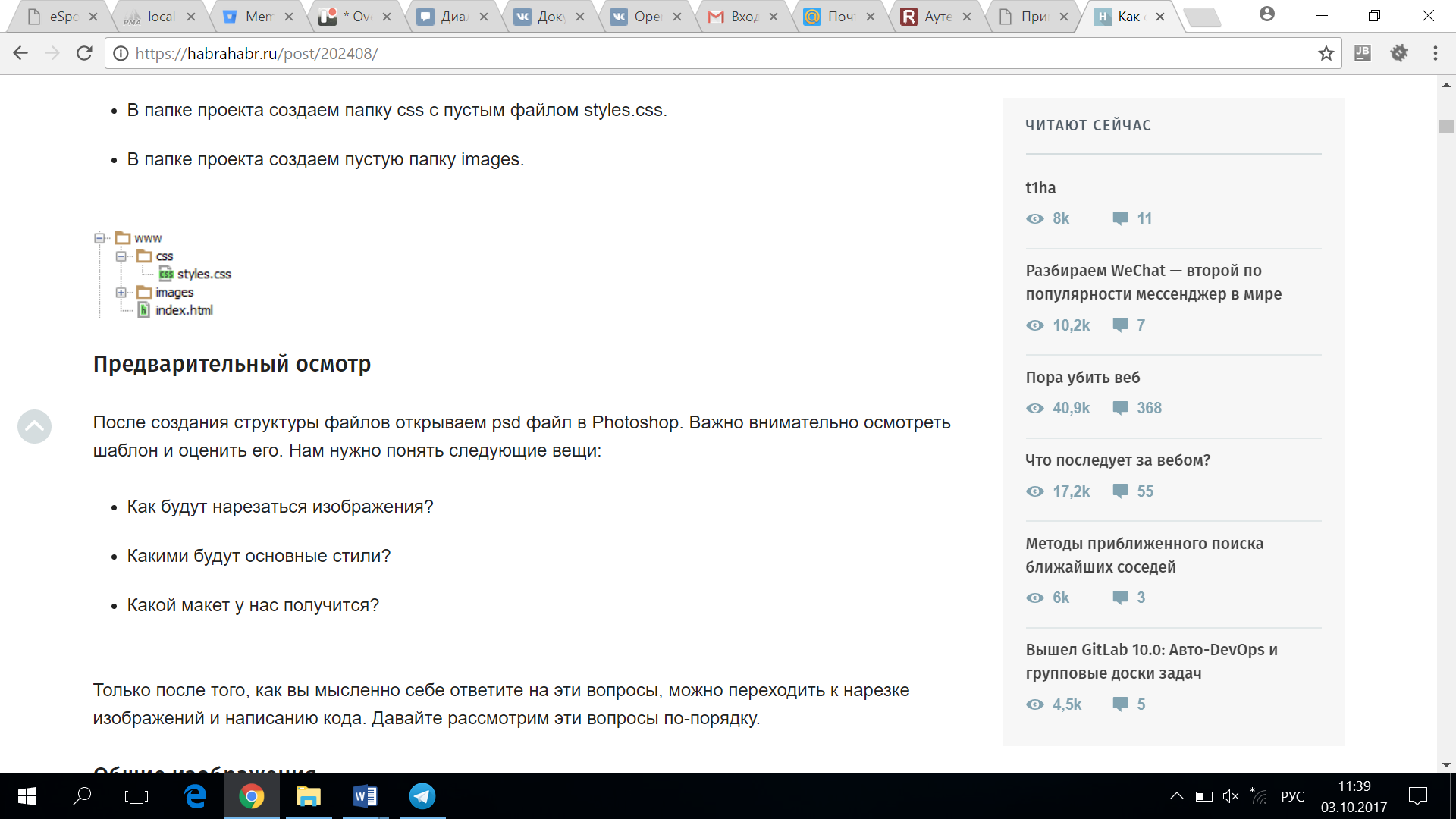


Рисунок 8.3 – Структура файлов

Сперва создается простая структуру файлов для документов. Далее необходимо создать папку с названием проекта, например Whitesquare. В ней создаем пустой файл index.html. В папке проекта создаем папку css с пустым файлом styles.css. В папке проекта создаем пустую папку images.

Рассмотрим этапы верстки шаблона.

1. Общие изображения. На данном этапе нужно нарезать и сохранить только общие изображения, которые будут на всех страницах сайта и не относятся к контенту. В данном случае это будет светло-серый фон страницы, фон заголовка, пустое изображение, два логотипа и кнопки социальных сетей.

Логотипы можно сохранить следующим образом:

/images/logo.png

/images/footer-logo.png

В качестве пустых картинок из макета использовано однопиксельное серое изображение.

Повторяющиеся фоновые изображения необходимо вырезать минимальным кусочком достаточным для образования полного изображения повторением по вертикали и горизонтали.

/images/bg.png

/images/h1-bg.png

Иконки социальных сетей с одинаковыми размерами удобно сохранить в один файл и использовать как спрайты для более быстрой загрузки. Для этого можно склеить картинки вручную в Photoshop, а можно сначала нарезать по одной, а затем склеить с помощью специально сервиса, например http://ru.spritegen.website-performance.org. В итоге получится два файла:

/images/social.png

/images/social-small.png

Общее правило при именовании изображений заключается в том, что мелкие и простые картинки, такие, как иконки, логотипы и так далее сохраняются в формате png, а фотографии в формате jpg.

2. Основные стили. На данном этапе желательно перенести все визуальные стили из дизайна в CSS, которые будут применяться по умолчанию для каждого тега.

Основной цвет фона примерно соответствует цвету #f8f8f8. Он будет показан в случае, если фоновая картинка не загрузится. Наверху страницы находится серая дизайнерская полоска. Применим ее через свойство border для body.

Основным шрифтом является тот шрифт, которым написан текст в области контента. Чтобы узнать его стили нужно выделить его в Photoshop’е и посмотреть свойства шрифта. В данном случае это Tahoma 12px с цветом #8f8f8f. Так же в этом макете параграфы имеют увеличенные отступы.

Прописываем все эти стили в styles.css:

body {

color: #8f8f8f;

font: 12px Tahoma, sans-serif;

background-color: #f8f8f8;

border-top: 5px solid #7e7e7e;

margin: 0;

}

input[type="text"] {

background-color: #f3f3f3;

border: 1px solid #e7e7e7;

height: 30px;

color: #b2b2b2;

padding: 0 10px;

vertical-align: top;

}

button {

color: #fff;

background-color: #29c5e6;

border: none;

height: 32px;

font-family: 'Oswald', sans-serif;

}

p {

margin: 20px 0;

}

В дальнейшем все стили будут описаны в этом же файле.

3. Каркас HTML. Запишем в index.html следующий код:

<!doctype html>

<html>

<head>

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

<meta http-equiv="X-UA-Compatible" content="IE=Edge">

<title>Whitesquare</title>

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

<link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Oswald:400,300" type="text/css">

<!--[if lt IE 9]>

<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>

<![endif]-->

</head>

<body>

</body>

</html>

Здесь указываем, что используем разметку HTML5, кодировку utf-8, страница называется Whitesquare. Также подключаем файл стилей и внешний файл со стилями шрифтов.

В последнем блоке в секции head подключаем специальный скрипт, который позволяет поддерживать Html5 теги в браузерах Internet Explorer меньше 9 версии. Мета-тег X-UA-Compatible сообщает, что в случае использования браузера Internet Explorer, он должен отразить сайт самым современным способом.

Весь html код в дальнейшем будет относиться к этому же файлу.

4. Макет. Сайт состоит из двух колонок: основного контента и сайдбара. Над ними находится шапка (header), в которой располагаются три горизонтальных блока: логотип с поиском, меню и название страницы. В самом низу под колонками располагается серый горизонтальный блок футера (footer). Вышеописанная структура макета представлена на рисунке 8.4.

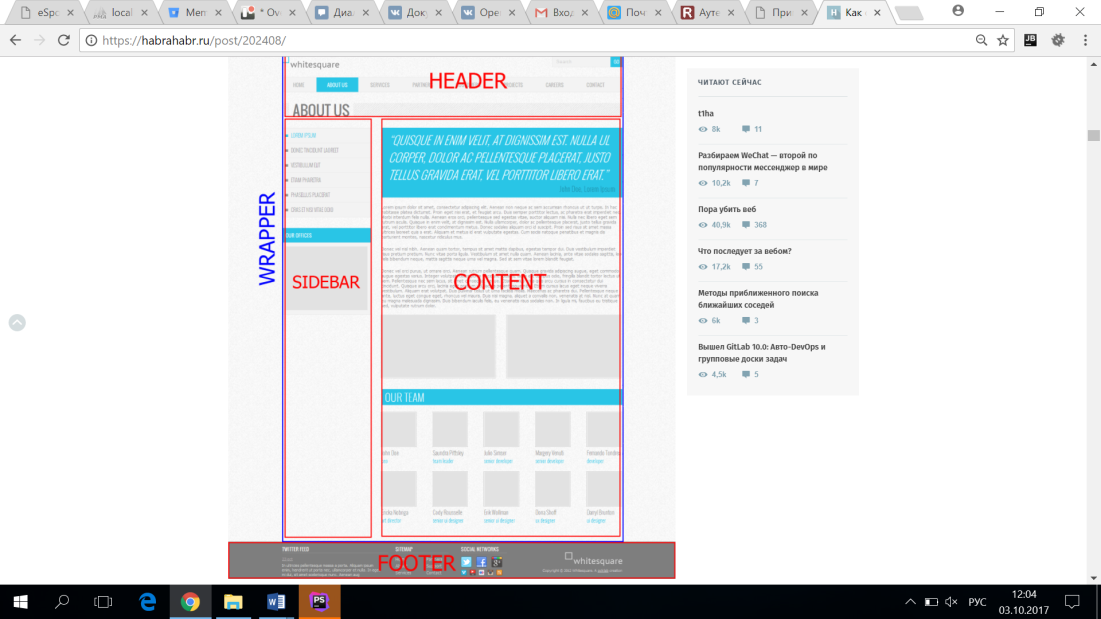


Рисунок 8.4 – Внешний вид макета

Опишем это в теге body:

<body>

<div id="wrapper">

<header></header>

<nav></nav>

<div id="heading"></div>

<aside></aside>

<section></section>

</div>

<footer></footer>

</body>

Wrapper используется для объединения блоков и их выравнивания по центру страницы.

Затем укажем стили блоков:

#wrapper {

max-width: 960px;

margin: auto;

}

header {

padding: 20px 0;

}

Вставляем логотип в тег header:

<header>

<a href="/"><img src="" alt="Whitesquare logo"></a>

</header>

Дополнительных стилей не требуется.

Вставляем форму поиска в тег header:

<header>

…

<form name="search" action="#" method="get">

<input type="text" name="q" placeholder="Search"><button type="submit">GO</button>

</form>

</header>

И стили выравнивания по правому краю для нее:

form[name="search"] {

float: right;

}

Для отображения меню необходимо создать список со ссылками внутри тега nav:

<nav>

<ul class="top-menu">

<li><a href="/home/">HOME</a></li>

<li class="active">ABOUT US</li>

<li><a href="/services/">SERVICES</a></li>

<li><a href="/partners/">PARTNERS</a></li>

<li><a href="/customers/">CUSTOMERS</a></li>

<li><a href="/projects/">PROJECTS</a></li>

<li><a href="/careers/">CAREERS</a></li>

<li><a href="/contact/">CONTACT</a></li>

</ul>

</nav>

CSS стили для него будут следующие:

nav a {

text-decoration: none;

}

nav ul {

margin: 0;

padding: 0;

}

nav li {

list-style-position: inside;

font: 14px 'Oswald', sans-serif;

padding: 10px;

}

.top-menu li {

display: inline-block;

padding: 10px 30px;

margin: 0;

}

.top-menu li.active {

background: #29c5e6;

color: #fff;

}

.top-menu a {

color: #b2b2b2;

}

Здесь описано, что для всех навигаций ссылки не будут иметь подчеркивания, убрали стандартные отступы для элементов списка, отобразили список горизонтально и указали нужные цвета и шрифт.

Заголовок страницы помещается в div с идентификатором heading

<div id="heading">

<h1>ABOUT US</h1>

</div>

Заголовок имеет следующие стили:

#heading {

background: transparent url(../images/h1-bg.png);

margin: 30px 0;

padding-left: 20px;

}

h1 {

display: inline-block;

color: #7e7e7e;

font: 40px/40px 'Oswald', sans-serif;

background: url(../images/bg.png);

margin: 0;

padding: 0 10px;

}

Рисуем серую полоску фоном на div’e, и в нее вкладываем инлайновый h1 с нужным шрифтом и фоном цвета страницы, чтобы создалось впечатление прозрачного фона для h1.

Для того, чтобы создать колонки страницы нужно прописать следующие стили:

aside {

float: left;

width: 250px;

}

section {

margin-left: 280px;

padding-bottom: 50px;

}

Здесь задали фиксированную ширину 250 пикселей для сайдбара, прибили его к левому краю и отодвинули колонку с контентом вправо на 280 пикселей от левого края. Также добавили отступ у контента снизу.

Подменю создаем аналогично главному меню. Для этого в теге aside прописываем следующее:

<aside>

<nav>

<ul class="aside-menu">

<li class="active">LOREM IPSUM</li>

<li><a href="/donec/">DONEC TINCIDUNT LAOREET</a></li>

<li><a href="/vestibulum/">VESTIBULUM ELIT</a></li>

<li><a href="/etiam/">ETIAM PHARETRA</a></li>

<li><a href="/phasellus/">PHASELLUS PLACERAT</a></li>

<li><a href="/cras/">CRAS ET NISI VITAE ODIO</a></li>

</ul>

</nav>

</aside>

И применяем к подменю следующие стили:

.aside-menu li {

font-weight: 300;

list-style-type: square;

border-top: 1px solid #e7e7e7;

}

.aside-menu li:first-child {

border: none;

}

.aside-menu li.active {

color: #29c5e6;

}

.aside-menu a {

color: #8f8f8f;

}

Для подменю применяется более тонкий шрифт и квадратные маркеры. Для отображения разделителей рисуем верхнюю границу у каждого элемента списка, кроме первого.

В контенте сайдбара помимо подменю в макете располагается также изображение с расположением офисов.

В html это выглядит так:

<h2>OUR OFFICES</h2>

<p>

<img src="images/sample.png" width="230" height="180" alt="Our offices">

</p>

В стилях укажем шрифты, цвета и отступы:

aside > h2 {

background: #29c5e6;

font: 14px 'Oswald', sans-serif;

color: #fff;

padding: 10px;

margin: 30px 0 0 0;

}

aside > p {

background: #f3f3f3;

border: 1px solid #e7e7e7;

padding: 10px;

margin: 0;

}

Данные стили применяются только к заголовкам и параграфам, лежащим непосредственно внутри сайдбара, но не глубже.

Вёрстку контента начнём с добавления цитаты. Добавим код цитаты в раздел section:

<section>

<blockquote>

<p>

“QUISQUE IN ENIM VELIT, AT DIGNISSIM EST. NULLA UL CORPER, DOLOR AC PELLENTESQUE PLACERAT, JUSTO TELLUS GRAVIDA ERAT, VEL PORTTITOR LIBERO ERAT.”

</p>

<cite>John Doe, Lorem Ipsum</cite>

</blockquote>

</section>

И применим для него стили:

blockquote {

margin: 0;

background: #29c5e6;

padding: 10px 20px;

font-family: 'Oswald', sans-serif;

font-weight: 300;

}

blockquote p {

color: #fff;

font-style: italic;

font-size: 33px;

margin: 0;

}

blockquote cite {

display: block;

font-size: 20px;

font-style: normal;

color: #1d8ea6;

margin: 0;

text-align: right;

}

Здесь нет ничего нового, так же — шрифты, фоны и отступы.

Все стили для текста контента уже добавлены, остается добавить только три параграфа с самим текстом после:

<p>Lorem ipsum dolor sit amet…</p>

<p>Donec vel nisl nibh…</p>

<p>Donec vel nisl nibh…</p>

Следующим шагом нужно добавить два изображения, которые находятся в конце текста контента. Делается это с помощью тега figure:

<figure>

<img src="images/sample.png" width="320" height="175" alt="">

</figure>

<figure>

<img src="images/sample.png" width="320" height="175" alt="">

</figure>

figure {

display: inline-block;

margin: 0;

font-family: 'Oswald', sans-serif;

font-weight: 300;

}

figure img {

display: block;

border: 1px solid #fff;

outline: 1px solid #c9c9c9;

}

figure figcaption {

font-size: 16px;

font-weight: 300;

margin-top: 5px;

}

figure figcaption span {

display: block;

font-size: 14px;

color: #29c5e6;

}

section > figure + figure {

margin-left: 28px;

}

Здесь убираем стандартные отсупы у figure, отображаем его как инлайновый блок и применили нужный шрифт. Изображение отображаем как блочный элемент с белой рамкой. Вторую серую рамку можно сделать через css-свойство outline. Самое интересное находится в последнем правиле, которое задает левый отступ у всех figure кроме первого внутри тега section.

При верстке блока «Our team»добавим сначала заголовок:

<h2>OUR TEAM</h2>

section > h2 {

background: #29c5e6;

font: 30px 'Oswald', sans-serif;

font-weight: 300;

color: #fff;

padding: 0 10px;

margin: 30px 0 0 0;

}

А затем два блока-строки с карточками сотрудников:

<div class="team-row">

<figure>

<img src="images/sample.png" width="96" height="96" alt="">

<figcaption>John Doe<span>ceo</span></figcaption>

</figure>

<figure>

<img src="images/sample.png" width="96" height="96" alt="">

<figcaption>Saundra Pittsley<span>team leader</span></figcaption>

</figure>

…

</div>

<div class="team-row">

<figure>

<img src="images/sample.png" width="96" height="96" alt="">

<figcaption>Ericka Nobriga<span>art director</span></figcaption>

</figure>

<figure>

<img src="images/sample.png" width="96" height="96" alt="">

<figcaption>Cody Rousselle<span>senior ui designer</span></figcaption>

</figure>

…

</div>

Таким образом, карточка (figure) состоит из фотографии (img), подписи (figcaption) с именем сотрудника и его должностью (div). Карточки будут иметь следующие стили:

figure figcaption {

font-size: 16px;

font-weight: 300;

margin-top: 5px;

}

figure div {

font-size: 14px;

color: #29c5e6;

}

.team-row figure {

margin-top: 20px;

}

.team-row figure + figure {

margin-left: 43px;

}

Здесь мы задали шрифт и отступ для подписи, размер и цвет должности, добавили верхний отступ для карточек и указали, что для всех карточек в строке, кроме первой, должен быть отступ слева.

Футер состоит из четырёх больших блоков: ленты Твиттера, карты сайта, социальных ссылок и логотипа с копирайтом.

Для начала создадим контейнер футера с этим блоками:

<footer>

<div id="footer">

<div id="twitter"></div>

<div id="sitemap"></div>

<div id="social"></div>

<div id="footer-logo"></div>

</div>

</footer>

И применим к нему оформление:

footer {

background: #7e7e7e;

color: #dbdbdb;

font-size: 11px;

}

#footer {

max-width: 960px;

margin: auto;

padding: 10px 0;

height: 90px;

}

Контейнер с id=”footer” находится внутри тега footer, это даёт нам возможность через тег footer задать серую область по всей ширине экрана, а через внутренний div отцентрировать с максимальной шириной 960 пикселей. Также этот div задает обоим блокам высоту 90 пикселей.

Далее верстаем содержимое ленты Твиттера:

<div id="twitter">

<h3>TWITTER FEED</h3>

<time datetime="2012-10-23"><a href="#">23 oct</a></time>

<p>

In ultricies pellentesque massa a porta. Aliquam ipsum enim, hendrerit ut porta nec, ullamcorper et nulla. In eget mi dui, sit amet scelerisque nunc. Aenean aug

</p>

</div>

Стили:

footer h3 {

font: 14px 'Oswald', sans-serif;

color: #fff;

border-bottom: 1px solid #919191;

margin: 0 0 10px 0;

}

#twitter time a {

color: #b4aeae;

}

footer p {

margin: 5px 0;

}

#twitter {

float: left;

width: 300px;

}

#twitter p {

padding-right: 15px;

}

Из интересных моментов здесь следующее: подчеркивание у заголовка сделали через нижнюю границу, а сам блок твиттера, как и последующие блоки выровняем по левому краю и задаём ширину.

Карта сайта представляет собой два блока со ссылками:

<div id="sitemap">

<h3>SITEMAP</h3>

<div>

<a href="/home/">Home</a>

<a href="/about/">About</a>

<a href="/services/">Services</a>

</div>

<div>

<a href="/partners/">Partners</a>

<a href="/customers/">Support</a>

<a href="/contact/">Contact</a>

</div>

</div>

Ссылкам задаем цвет и оставляем подчеркивание только для наведённых.

Колонки со ссылками делаем через инлайновые блоки и затем свойством #sitemap div + div отодвигаем вторую колонку от первой:

footer a {

color: #dbdbdb;

}

#sitemap {

width: 150px;

float: left;

margin-left: 20px;

padding-right: 15px;

}

#sitemap div {

display: inline-block;

}

#sitemap div + div {

margin-left: 40px;

}

#sitemap a {

display: block;

text-decoration: none;

font-size: 12px;

margin-bottom: 5px;

}

#sitemap a:hover {

text-decoration: underline;

}

Вставляем набор социальных ссылок в контейнер:

<div id="social">

<h3>SOCIAL NETWORKS</h3>

<a href="http://twitter.com/" class="social-icon twitter"></a>

<a href="http://facebook.com/" class="social-icon facebook"></a>

<a href="http://plus.google.com/" class="social-icon google-plus"></a>

<a href="http://vimeo.com/" class="social-icon-small vimeo"></a>

<a href="http://youtube.com/" class="social-icon-small youtube"></a>

<a href="http://flickr.com/" class="social-icon-small flickr"></a>

<a href="http://instagram.com/" class="social-icon-small instagram"></a>

<a href="/rss/" class="social-icon-small rss"></a>

</div>

И стилизуем их:

#social {

float: left;

margin-left: 20px;

width: 130px;

}

.social-icon {

width: 30px;

height: 30px;

background: url(../images/social.png) no-repeat;

display: inline-block;

margin-right: 10px;

}

.social-icon-small {

width: 16px;

height: 16px;

background: url(../images/social-small.png) no-repeat;

display: inline-block;

margin: 5px 6px 0 0;

}

.twitter {

background-position: 0 0;

}

.facebook {

background-position: -30px 0;

}

.google-plus {

background-position: -60px 0;

}

.vimeo {

background-position: 0 0;

}

.youtube {

background-position: -16px 0;

}

.flickr {

background-position: -32px 0;

}

.instagram {

background-position: -48px 0;

}

.rss {

background-position: -64px 0;

}

Здесь применили технику спрайтов – когда один файл с изображением применяется для разных картинок. Все ссылки разделились на большие иконки (.social-icon) и маленькие (.social-icon-small). Мы задали этим классом отображение в виде инлайнового блока с фиксированными размерами и одинаковым фоном. А затем с помощью css сдвинули этот фон так, чтобы на каждой ссылке отобразилось соответствующее изображение.

Блок с копирайтом и логотипом – это картинка со ссылкой и параграф с текстом под ним:

<div id="footer-logo">

<a href="/"><img src="" alt="Whitesquare logo"></a>

<p>Copyright © 2012 Whitesquare. A <a href="http://pcklab.com">pcklab</a> creation</p>

</div>

Стили делают аналогично предыдущим блокам с той лишь разницей, что блок прибивается к правому краю и выравнивание внутри него так же по правому краю:

#footer-logo {

float: right;

margin-top: 20px;

font-size: 10px;

text-align: right;

}

**5. Порядок выполнения работы**

1. Сверстать адаптивную HTML-страницу, представленную в psd-макете (psd-макет получить у преподавателя в соответствии с вариантом) под разрешения экранов в соответствии с вариантом.

3. В файле style.css описать все свойства для оформления страницы.

4. Подключите файл style.css к документу \*.html.

Вариант 1 – 320 px; 1440px.

Вариант 2 – 375 px; 1320px.

Вариант 3 – 425 px; 1024px.

Вариант 4 – 768 px; 2560px.

Вариант 5 – 320 px; 768px.

Вариант 6 – 375 px; 768px.

Вариант 7 – 425 px; 1280px.

Вариант 8 – 768 px; 1320px.

Вариант 9 – 320 px; 1280px.

Вариант 10 – 375 px; 1320px.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. В чем суть понятия «адаптивный макет»?

2. Для чего используются media-запросы?

2. Опишите процесс создания media-запроса.

3. Зачем разрабатывать адаптивные макеты?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
5. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 9**

## **Тема работы: «Создание CSS-анимации в web-документе.**

**1. Цель работы**

Формирование умений создания анимации средствами CSS в HTML-документе.

**2. Задание**

Реализовать анимированные эффекты средствами CSS в веб-документе.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

СSS3-анимация придаёт сайтам динамичность. Она оживляет веб-страницы, улучшая взаимодействие с пользователем. Cоздание анимации базируется на ключевых кадрах, которые позволяют автоматически воспроизводить и повторять эффекты на протяжении заданного времени, а также останавливать анимацию внутри цикла.

CSS3-анимация может применяться практически для всех html-элементов, а также для псевдоэлементов :before и :after.

Создание анимации начинается с установки ключевых кадров правила @keyframes. Кадры определяют, какие свойства на каком шаге будут анимированы. Каждый кадр может включать один или более блоков объявления из одного или более пар свойств и значений. Правило @keyframes содержит имя анимации элемента, которое связывает правило и блок объявления элемента.

@keyframes shadow {

from {text-shadow: 0 0 3px black;}

50% {text-shadow: 0 0 30px black;}

to {text-shadow: 0 0 3px black;}

}

Ключевые кадры создаются с помощью ключевых слов from и to (эквивалентны значениям 0% и 100%) или с помощью процентных пунктов, которых можно задавать сколько угодно. Также можно комбинировать ключевые слова и процентные пункты. Если кадры имеют одинаковые свойства и значения, их можно объединить в одно объявление:

@keyframes move {

from,

to {

top: 0;

left: 0;

}

25%,

75% {top: 100%;}

50% {top: 50%;}

}

Если 0% или 100% кадры не указаны, то браузер пользователя создает их, используя вычисляемые (первоначально заданные) значения анимируемого свойства. Если у двух ключевых кадров будут одинаковые селекторы, то последующий отменит действие предыдущего.

После объявления правила @keyframes можно ссылаться на него в свойстве animation:

h1 {

font-size: 3.5em;

color: darkmagenta;

animation: shadow 2s infinite ease-in-out;

}

Не рекомендуется анимировать нечисловые значения (за редким исключением), так как результат в браузере может быть непредсказуемым. Также не следует создавать ключевые кадры для значений свойств, не имеющих средней точки, например, для значений свойства color: pink и color: #ffffff, width: auto и width: 100px или border-radius: 0 и border-radius: 50% (в этом случае правильно будет указать border-radius: 0%).

Название анимации animation-name. Свойство задаёт имя анимации. Имя анимации создаётся в правиле @keyframes. Рекомендуется использовать название, отражающее суть анимации, при этом можно использовать одно или несколько слов, связанных между собой при помощи пробела – или символа нижнего подчеркивания \_. Свойство не наследуется. Значения свойства animation-name представлены в таблице 9.1.

Таблица 9.1 – Значения свойства animation-name

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| имя анимации | Имя анимации, которое связывает правило @keyframes с селектором. |
| none | Значение по умолчанию, означает отсутствие анимации. Также используется, чтобы отменить анимацию элемента из группы элементов, для которых задана анимация. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис: div {animation-name: mymove;}

Продолжительность анимации animation-duration. Свойство устанавливает продолжительность анимации, задаётся в секундах или миллисекундах, отрицательные значения не допустимы. Не наследуется. Если для элемента задано более одной анимации, то можно установить разное время для каждой, перечислив значения через запятую. Значения свойства animation-duration представлены в таблице 9.2.

Таблица 9.2 – Значения свойства animation-duration

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| время | Длительность анимации задается в секундах s или миллисекундах ms. |
| initial | Устанавливает значение свойства в значение по умолчанию 0. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис: div {animation-duration: 2s;}

Временная функция animation-timing-function. Свойство определяет изменение скорости от начала до конца анимации с помощью временных функций. Задаётся при помощи ключевых слов или кривой Безье cubic-bezier(x1, y1, x2, y2). Не наследуется. Значения свойства animation-timing-function представлены в таблице 9.3.

Таблица 9.3 – Значения свойства animation-timing-function

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **1** | **2** |
| ease | Функция по умолчанию, анимация начинается медленно, разгоняется быстро и замедляется в конце. Соответствует cubic-bezier(0.25,0.1,0.25,1). |
| linear | Анимация происходит равномерно на протяжении всего времени, без колебаний в скорости. Соответствует cubic-bezier(0,0,1,1). |
| ease-in | Анимация начинается медленно, а затем плавно ускоряется в конце. Соответствует cubic-bezier(0.42,0,1,1). |
| ease-out | Анимация начинается быстро и плавно замедляется в конце. Соответствует cubic-bezier(0,0,0.58,1). |
| ease-in-out | Анимация медленно начинается и медленно заканчивается. Соответствует cubic-bezier(0.42,0,0.58,1). |
| cubic-bezier(x1, y1, x2, y2) | Позволяет вручную установить значения от 0 до 1. [На этом сайте](https://html5book.ru/goto/http:/roblaplaca.com/examples/bezierBuilder/)вы сможете построить любую траекторию скорости изменения анимации. |
| step-start | Задаёт пошаговую анимацию, разбивая анимацию на отрезки, изменения происходят в начале каждого шага. Эквивалентно steps(1, start). |
| step-end | Пошаговая анимация, изменения происходят в конце каждого шага. Эквивалентно steps(1, end). |
| steps(количество шагов,start|end) | Ступенчатая временная функция, которая принимает два параметра. Количество шагов задается целым положительным числом. Второй |

Продолжение таблицы 9.3

|  |  |
| --- | --- |
| **1** | **2** |
|  | параметр необязательный, указывает момент, в котором начинается анимация. Со значением start анимация начинается в начале каждого шага, со значением end — в конце каждого шага с задержкой. Задержка вычисляется как результат деления времени анимации на количество шагов. Если второй параметр не указан, используется значение по умолчанию end. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис: div {animation-timing-function: linear;}

С помощью пошаговой анимации можно создавать интересные эффекты, например, печатающийся текст или индикатор загрузки.

Анимация с задержкой animation-delay. Свойство игнорирует анимацию заданное количество времени, что даёт возможность по отдельности запускать каждую анимацию. Отрицательная задержка начинает анимацию с определенного момента внутри её цикла, т.е. со времени, указанного в задержке. Это позволяет применять анимацию к нескольким элементам со сдвигом фазы, изменяя лишь время задержки.

Чтобы анимация началась с середины, нужно задать отрицательную задержку, равную половине времени, установленном в animation-duration. Не наследуется. Значения свойства animation-delay представлены в таблице 9.4.

Таблица 9.4 – Значения свойства animation-delay

|  |  |
| --- | --- |
| Значение | Описание |
| время | Задержка анимации задается в секундах s или миллисекундах ms. Значение по умолчанию 0. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис: div {animation-delay: 2s;}

Повтор анимации animation-iteration-count. Свойство позволяет запустить анимацию несколько раз. Значение 0 или любое отрицательное число удаляют анимацию из проигрывания. Не наследуется. Значения свойства animation-iteration-count представлены в таблице 9.5.

Таблица 9.5 – Значения свойства animation-iteration-count

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **1** | **2** |
| число | С помощью целого числа задается количество повторов анимации. Значение по умолчанию 1. Дробные значения больше 1 будут воспроизводить анима |

|  |  |
| --- | --- |
| **1** | **2** |
|  | цию, обрезая её на части следующей итерации. |
| infinite | Анимация проигрывается бесконечно. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис: div {animation-iteration-count: 3;}

Направление анимации animation-direction. Свойство задает направление повтора анимации. Если анимация повторяется только один раз, то это свойство не имеет смысла. Не наследуется. Значения свойства animation-direction представлены в таблице 9.6.

Таблица 9.6 – Значения свойства animation-direction

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| alternate | Анимация проигрывается с начала до конца, затем в обратном направлении. |
| alternate-reverse | Анимация проигрывается с конца до начала, затем в обратном направлении. |
| normal | Значение по умолчанию, анимация проигрывается в обычном направлении, с начала и до конца. |
| reverse | Анимация проигрывается с конца. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис: div {animation-direction: alternate;}

Все параметры воспроизведения анимации можно объединить в одном свойстве – animation, перечислив их через пробел:

animation: animation-name animation-duration animation-timing-function animation-delay animation-iteration-count animation-direction;

Для воспроизведения анимации достаточно указать только два свойства – animation-name и animation-duration, остальные свойства примут значения по умолчанию. Порядок перечисления свойств не имеет значения, единственное, время выполнения анимации animation-duration обязательно должно стоять перед задержкой animation-delay.

Проигрывание анимации animation-play-state. Свойство управляет проигрыванием и остановкой анимации. Остановка анимации внутри цикла возможна через использование этого свойства в скрипте JavaScript. Также можно останавливать анимацию при наведении курсора мыши на объект — состояние :hover. Не наследуется. Значения свойства animation-play-state представлены в таблице 9.7.

Таблица 9.7 – Значения свойства animation-play-state

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| paused | Останавливает анимацию. |
| running | Значение по умолчанию, означает проигрывание анимации. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис: div:hover {animation-play-state: paused;}

Состояние элемента до и после воспроизведения анимации animation-fill-mode. Свойство определяет порядок применения определенных в @keyframes стилей к объекту. Не наследуется. Значения свойства animation-fill-mode представлены в таблице 9.8.

Таблица 9.8 – Значения свойства animation-fill-mode

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| none | Значение по умолчанию. Состояние элемента не меняется до или после воспроизведения анимации. |
| forwards | После того, как анимация заканчивается (как определено значением animation-iteration-count), анимация будет применять значения свойств к моменту окончания анимации. Если animation-iteration-count больше нуля, применяются значения для конца последней завершенной итерации анимации (а не значения для начала итерации, которое будет следующим). Если значение animation-iteration-count равно нулю, применяемыми значениями будут те, которые начнут первую итерацию (так же, как и в режиме animation-fill-mode: backwards;). |
| backwards | В течение периода, определенного с помощью animation-delay, анимация будет применять значения свойств, определенные в ключевом кадре, которые начнут первую итерацию анимации. Это либо значения ключевого кадра from (когда animation-direction: normal или animation-direction: alternate), либо значения ключевого кадра to (когда animation-direction: reverse или animation-direction: alternate). |
| both | Позволяет оставлять элемент в первом ключевом кадре до начала анимации (игнорируя положительное значение задержки) и задерживать на последнем кадре до конца последней анимации. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис: div {animation-fill-mode: forwards;}

Для одного элемента можно задавать несколько анимаций, перечислив их названия через запятую:

div {animation: shadow 1s ease-in-out 0.5s alternate, move 5s linear 2s;}

**5. Порядок выполнения работы**

1. Создать HTML страницу, внедрить CSS-анимацию, добавив 4 прямоугольника, которые будут передвигаться друг за другом от правого края до левого края экрана и обратно.

2. Создать HTML страницу, внедрить CSS-анимацию, добавив 4 круга которые будут передвигаться друг за другом по кругу по часовой стрелке. Для всех кругов укажите различные точки трансформации.

3. Выполните задание по варианту. Варианты представлены в таблице 9.9.

Таблица 9.9 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| 1 | 1. Создать плавно выпадающее меню. 2. Создать индикатор выполнения процесса от 0 до 100%. 3. Создать анимацию прыгающего мячика. |
| 2 | 1. Создать кнопку, которая при наведении плавно меняет свой цвет. 2. С помощью HTML и CSS-анимации создать радиоволны, идущие от антенны. 3. Создать анимацию механических часов. |
| 3 | 1. Создать анимацию мигания разделителя (13:27, : - разделитель) электронных часов. 2. Создать анимацию появления информационного блока при наведении на фото. 3. Создать форму с увеличивающимися при фокусе полями. |
| 4 | 1. Создать фотоальбом у увеличением фото при наведении. 2. Создать меню, при наведении на которое элементы в нем повернутся на 360 градусов. Тоже самое должно произойти и после того, как курсор убрали, но в обратную сторону. 3. Создать анимацию движения Луны вокруг Земли. |
| 5 | 1. Создать анимацию круглого загрузчика. 2. Создать счетчик. 3. Создать анимацию переката квадрата. В течении его полного поворота он должен стать кругом. |
| 6 | 1. Создать иконку меню, которая из трех полос при наведении преобразуется в круг с крестиком. 2. Создать эффект тряски при наведении на текст. 3. Создать анимацию шаров Ньютона. |
| 7 | 1. Создать появляющийся и увеличивающийся в размере текст при загрузке страницы. 2. Создать анимацию движения начального текста в Звездных войнах. 3. Создать анимацию движения текста по кругу. |
| 8 | 1. Создать анимацию движения блока при наведении. Когда курсор убирается – блок двигается назад. 2. Создать анимацию движения как на экране радара. 3. Создать анимацию переворота блока при наведении. |
| 9 | 1. Создать круг с переливающимся цветом. 2. Создать появляющийся после загрузки страницы блок. 3. Создать анимацию круглого загрузчика. |
| 10 | 1. Создать фотоальбом у уменьшением фото при наведении. 2. Создать анимацию появления информационного блока при наведении на фото. 3. Создать анимацию прыгающего мячика. |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Перечислите способы создания анимации средствами CSS. Укажите их отличия.

2 Что такое «ключевые кадры»?

3. Укажите формат описания CSS animations.

4. Укажите формат описания CSS transitions.

5. Перечислите под-свойства свойства animation.

6. Перечислите под-свойства свойства transitions.

7. Раскройте суть понятия «множественные трансформации».

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
5. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 10**

## **Тема работы: «Разработка кроссбраузерных решений»**

**1. Цель работы**

Формирование умений кроссбраузерной верстки веб-страниц.

**2. Задание**

Реализовать кроссбраузерную верстку веб-страниц под популярные браузеры.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Кроссбраузерная html и css верстка – это верстка сайта, нацеленная на создание сайта, отображаемого в каждом браузере одинаково корректно.

Рекомендации по выполнению кроссбраузерной верстки:

1. Значение атрибутов по умолчанию в разных браузерах может быть разным. Под атрибутами имеется в виду цвет элемента, расстояние между элементами, размер шрифта и т.д. Если все эти и прочие параметры не прописать в таблице стилей, то браузер сам начнет подставлять значения, которые указаны у него по умолчанию. Решить проблему с атрибутами по умолчанию можно путем описания этих атрибутов в таблице стилей, чтобы браузеры не додумывали и не подставляли свои значения.

Пример:

html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym, address, big, cite, code, del, dfn, em, font, img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt, var, dl, dt, dd, ol, ul, li, fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td

{

margin: 0;

padding: 0;

border: 0;

outline: 0;

font-weight: inherit;

font-style: inherit;

font-size: 14px;

font-family: inherit;

vertical-align: top;

background: transparent;

font: verdana, geneva, lucida, 'lucida grande', arial, helvetica, sans-serif;

background-color : white;

}

Браузеры по умолчанию по-разному стилизуют элементы HTML. Некоторые браузеры имеют различный подход к значениям таких стилей как margin и padding. Чтобы это исправить, есть хороший приём – добавить код CSS reset в начало вашего файла стилей. Это сбросит стили всех элементов. Также это приведёт к тому, что стилизация будет выполняться с нуля.

2. Установить на компьютер все самые популярные браузеры. Можно так же воспользоваться программой IE Tester, которая позволяет проверять сайт во всем множестве версий IE.

Хорошей практикой перед выкладыванием сайта в сеть является проверка HTML и CSS при помощи валидаторов, так как это поможет исправить некоторые мелкие ошибки, которые могут принести проблемы в будущем. Вы можете использовать валидаторы HTML и CSS от W3C. Эти валидаторы одобрены W3C.

Более старые версии IE доставляли дискомфорт веб-дизайнерам и разработчикам, когда дело доходило до проблем совместимости. Для решения этой проблемы Microsoft разработали условные комментарии для своего браузера, позволяющие привязывать файл стилей, который будет интерпретирован только браузером IE.

Следующий код будет работать во всех версиях IE:

<!--[if IE]>

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

<[endif]-->

А следующий код предназначен для конкретной версии IE:

<!--[if IE6]>

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

<[endif]-->

3. Необходимо определить для себя стратегию верстки: как будет удобнее: верстать сразу весь сайт под один браузер, а после погонять его под остальные либо же верстать постепенно и каждый элемент проверять во всех браузерах, при необходимости корректируя.

4. Использование плагинов позволяет быстрее находить ошибки кода для каждого из браузеров. Как пример можно потестировать эти плагины:

- Chrome на сайте getfirebug.com;

- Opera на официальном сайте Opera;

- Mozila на сайте addons mozila;

- IE на оф. сайте Microsoft;

5. Для современных браузеров, но не обновленных (так сказать устаревших) следует использовать префиксы поставщиков браузеров. Это помогает разработчикам браузеров реализовывать новые CSS3-модули.

.round{

-khtml-border-radius: 10px; /\* Konqueror \*/

-rim-border-radius: 10px; /\* RIM \*/

-ms-border-radius: 10px; /\* Microsoft \*/

-o-border-radius: 10px; /\* Opera \*/

-moz-border-radius: 10px; /\* Mozilla (например, Firefox) \*/

-webkit-border-radius: 10px; /\* Webkit (например, Safari и Chrome) \*/

-border-radius: 10px; /\* W3C \*/

}

В этом примере используются свойства с префиксами поставщиков (причем это далеко не исчерпывающий список), каждое из которых обладает собственным уникальным префиксом, например, webkit означает основанные на WebKit браузеры, или ms – префикс такого поставщика, как Microsoft, подразумевающий Internet Explorer, и т. д. Из-за особенностей работы CSS браузер будет двигаться вниз по таблице стилей строка за строкой, применяя соответствующие свойства и игнорируя те, которые не может распознать.

В качестве примера давайте используем свойство transition с вендорными префиксами для определённых браузеров:

-webkit-transition: all 4s ease;

-moz-transition: all 4s ease;

-ms-transition: all 4s ease;

-o-transition: all 4s ease;

transition: all 4s ease;

На сегодняшний день не существует каких-либо строгих правил использования определённого измерения для размера шрифта, одним из вариантов использования является использование единицы em и процентов, поскольку они основаны на настройке размера шрифта в браузере пользователя. Большинство людей используют пиксели (px) или пункты (pt), которые основаны на разрешении экрана. Они всегда фиксированы.

**5. Порядок выполнения работы**

1. Доработать макет из ЛР№8, реализовав его кроссбраузерную верстку под популярные браузеры: Google Chrome, Mozilla Firefox, Opera, Internet Explorer.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой кроссбраузерная верстка?

2. Что такое префикс?

3. В чем заключается правило использования префиксов?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.
3. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
4. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
5. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.

# **Лабораторная работа № 11**

## **Тема работы: «Внедрение сценариев в HTML-документ. Обработка событий с использованием DHTML»**

**1. Цель работы**

Формирование умений внедрения и использования сценариев JavaScript и обработки событий в веб-документе.

**2. Задание**

Реализовать внедрение сценариев и обработку событий в веб-документе.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили.

Обычно сценарии внедряются в HTML-документ тремя стандартными способами:

– в виде гиперссылки;

– в виде обработчика события;

– в рамках тега script.

Для создания на веб-страницах гиперссылок применяется элемент <a>, у которого есть атрибут href для указания адреса страницы или файла, куда приведет гиперссылка. Однако, вместо адреса данный атрибут может содержать и JavaScript-сценарий, если перед сценарием поставить слово javascript: (со знаком:). Рассмотрим пример такой гиперссылки:

<a href="javascript:alert('Привет, мир!');">Нажмите сюда</a>

Браузер может реагировать на различные события. Поведение каждого элемента зависит от событий, которые постоянно возникают в браузере. Например, пользователь щелкнул кнопкой мыши на тексте, переместил указатель мыши, загрузил HTML-документ. Браузер сам решает, как реагировать на каждое событие. Однако вы можете перехватывать отдельные события и задавать новое поведение при их наступлении. В языке HTML у многих элементов есть атрибуты, соответствующие различным событиям. Эти атрибуты могут содержать код языка JavaScript, выполняемый в ответ на событие. Например:

<p onclick="alert('Не нажимать!');">Это простой текст</p>

В языке HTML практически каждый элемент имеет атрибуты для обработки каких-либо событий. Для обработки событий данным атрибутам назначается код на языке JavaScript, который выполняется, когда произойдет событие. Благодаря возможности обработки событий вы можете создавать интерактивные страницы, отвечающие на действия пользователей.

Кроме возможности обработки событий, для внедрения сценариев в веб-страницы в языке разметки HTML есть специальный тег <script>. Браузер, встретив тег <script> в процессе разбора кода, интерпретирует текст, последующий до закрывающего тега </script>, как сценарий на каком-либо языке. Для указания языка сценария используется атрибут type.

Например, для указания языка JavaScript:

<script type="text/javascript"> Здесь располагается код сценария </script>

При использовании сценариев JavaScript можно этот атрибут не указывать, так как значение «text/javascript» является значением по умолчанию.

Сценарии могут содержаться в самой веб-странице, как описано выше, а также в отдельном файле. Тег script имеет атрибут src, который должен иметь значение адреса файла со сценарием. Обычно файлы со сценариями JavaScript имеют расширения JS, например файл lib1.js. Таким образом, чтобы подключить к HTML-документу файл lib1.js, необходимо добавить в него следующую строку:

<script type="text/javascript" src="lib1.js"></src>

Взаимодействие с пользователем: alert, prompt, confirm.

Alert выводит на экран окно с сообщением и приостанавливает выполнение скрипта, пока пользователь не нажмёт «ОК».

alert( "Привет" );

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».

Функция prompt принимает два аргумента:

result = prompt(title, default);

Она 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.

Вызов prompt возвращает то, что ввёл посетитель – строку или специальное значение null, если ввод отменён.

var years = prompt('Сколько вам лет?', 100);

alert('Вам ' + years + ' лет!')

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

result = confirm(question);

Например:

var isAdmin = confirm("Вы - администратор?");

alert( isAdmin );

Динамический HTML (Dynamic HTML или DHTML) не является каким-то особым языком разметки страниц. Это всего лишь термин, применяемый для обозначения HTML-страниц с динамически изменяемым содержимым.

Реализация DHTML состоит из трех компонентов: непосредственно HTML, каскадных таблицах стилей (Cascade Style Sheets – CSS) и языке сценариев (JavaScript или VBScript). Эти три компонента DHTML связаны между собой объектной моделью документа (Document Object Model – DOM), являющейся, по сути, интерфейсом прикладного программирования (API).

DOM обеспечивает динамический доступ к содержимому документа, его структуре и стилям. В DOM каждый элемент Web-страницы является объектом, который можно изменять. DOM не определяет новых тэгов и атрибутов, а просто обеспечивает возможность программного управления всеми тэгами, атрибутами и каскадными таблицами стилей (CSS).

Любое действие пользователя (щелчок кнопкой мыши, перемещение мыши в окне браузера или нажатие клавиши клавиатуры) объектной моделью документа трактуется как событие, которое может быть перехвачено и обработано процедурой сценария.

DHTML включает в себя средства динамического изменения размера, цвета, формы символов, точного позиционирования и перемещения отдельных элементов страницы. Основную идею динамического HTML можно сформулировать очень просто: полный контроль языка сценариев над всеми элементами документа, параметрами их оформления и размещения (как подразумеваемыми в HTML, так и задаваемыми с помощью CSS) и над самим текстом страницы.

На WEB-страницах можно обеспечить реакцию на определенные действия посетителя или изменения состояния документа или окна, которые вызывают определенные события.

Существуют следующие категории событий:

– стандартные события (могут быть связаны с любым элементом HTML);

– события для элементов, получающих фокус ввода;

– события, специфические для отдельных элементов.

Как правило, обработчик событий имеет имя, совпадающее с именем события, с приставкой "on", таблица 11.1.

Таблица 11.1 – Типы событий и обработчиков

|  |  |  |
| --- | --- | --- |
| **Событие** | **Когда происходит** | **Обработчик**  **события** |
| Blur | Потеря фокуса ввода элементом формы | onBlur |
| Click | Пользователь щелкает мышкой по элементу формы или связи | onClick |
| Change | Пользователь изменяет значение элемента text, textarea, или выделяет элемент | onChange |
| Focus | Установка фокуса ввода на элементе формы | onFocus |
| Load | Пользователь загружает страницу в браузер | onLoad |
| Mouseover | Пользователь двигает мышью над связью или якорем | onMouseOver |
| Select | Пользователь выбирает поле ввода элемента формы | onSelect |
| Submit | Пользователь посылает форму | onSubmit |
| Unload | Пользователь выходит со страницы | onUnload |

Стандартные события приведены в таблице 11.2.

Таблица 11.2 – Стандартные события JavaScript

|  |  |
| --- | --- |
| **Обработчик**  **события** | **Действие** |
| Onclick | Щелчок левой кнопкой мыши по области элемента |
| Ondblcliсk | Двойной щелчок левой кнопкой мыши по области элемента |
| Onmousedown | Левая кнопка мыши нажата, когда указатель находится в области элемента |
| Onmousemove | Указатель мыши перемещается по области элемента |
| Onmouseup | Левая кнопка мыши отпущена, когда указатель находится в области элемента |
| Onmouseover | Указатель мыши входит в пределы области элемента |
| Onmouseout | Указатель мыши выходит за пределы элемента |
| Onkeydown | Нажата и удерживается клавиша на клавиатуре |
| Onkeypress | Нажата и отпущена клавиша на клавиатуре |
| Onkeyup | Отпущена клавиша на клавиатуре |
| Onscroll | Элемент прокручивается |

События для элементов, получающих фокус ввода, приведены в таблице 11.3.

Таблица 11.3 – События для получения фокуса ввода

|  |  |
| --- | --- |
| **Обработчик**  **события** | **Действие** |
| Onenter | Произошел переход к элементу |
| Onfocus | Устанавливается фокус ввода для элемента |
| Onexit | Произошел выход из области элемента |
| Onblur | Снимается фокус с элемента |

События, используемые для отдельных элементов, приведены в таблице 11.4.

Таблица 11.4 – События, используемые для отдельных элементов

|  |  |
| --- | --- |
| **Обработчик**  **события** | **Действие** |
| Onabort | Пользователь прерывает прием изображения |
| Onchange | объект изменяется при вводе данных пользователем (для элементов формы <INPUT type=file| text| password| radio| checkbox>, <SELECT>, <TEXTAREA>) |
| Onload | Элемент (<BODY> или <IMG>) полностью загружен |
| Onreset | Нажата кнопка Reset, т. е. пользователь очистил форму |
| Onselect | Отпущена кнопка мыши при выделении текста пользователем |
| Onsubmit | Нажата кнопка Submit, т. е. форма отправлена на обработку |
| Onunload | Документ начал выгружаться |

Для того чтобы связать объект с обработчиком события, нужно в теге, определяющем данный объект, указать атрибут, соответствующий событию, и, в качестве значения, записать программный код (например, на JavaScript) или имя функции.

Основной синтаксис, используемый для указания обработчиков событий:

<TAGeventHandler="JavaScriptCode">

где TAG – это некоторый тег HTML;

eventHandler – имя обработчика событий.

Пример:

<H1 onmouseover="this.style.color='red'"> Эта строка покраснеет</H1>

Ключевое слово this указывает на то, что изменяется стиль элемента, в теге которого определен обработчик события, в данном случае элемент H1.

Пример:

<html>

<head>

</head>

<BODY>

Введите Ваше имя:

<INPUT type="text" name="user" value="" onblur="salut(this.value)">

<SCRIPT>

function salut (name)

{

alert ("Здравствуйте, ", name, " !");

}

</SCRIPT>

</BODY>

</html>

Другой пример иллюстрирует обработку события onClick при нажатии кнопки «Выход» – происходит вызов функции подтверждения confirm:

<FORM>

<INPUT type="button" name="btn1" value="Выход" onClick="confirm('Вы желаете покинуть страницу?')">

</FORM>

<body>

<H2>Изменение вида данного элемента</H2>

<p id="B">Для изменения цвета и шрифта данного текста

нажимайте и отпускайте кнопку мыши<br></p>

<script>

<!--

document.getElementById('B').onmousedown=new

Function("this.style.fontStyle='italic'; this.style.color='red';");

document.getElementById('B').onmouseup=new

Function("this.style.fontStyle=''; this.style.color='blue';");

//-->

</script>

</body>

Если обработка события предполагает изменение ряда свойств элемента, можно определить два различных класса и применять их при возникновении событий:

<STYLE>

.red{color:red;font-style:italic}

.green{color:green;font-weight:bold}

</STYLE>

<BODY>

<P onMouseOver="this.className='green'

onMouseOut="this.className='red'">

Этот текст будет изменяться при наведении указателя мыши

</P>

</BODY>

При наведении указателя мыши будет применяться класс green, при выходе указателя за пределы элемента – класс red.

**5. Порядок выполнения работы**

1. Создайте страницу с текстом «Я, (ФИО), учусь работать с JavaScript». При выходе из области элемента надпись должна менять цвет.

2. Выполните задание по варианту. Варианты представлены в таблице 11.5.

Таблица 11.5 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| 1 | 1. Вывести информационное сообщение при полной загрузке страницы. 2. Создайте поле для ввода текстовой информации. По умолчанию поместите в текстовое поле ваши ФИО. При вводе данных в текстовое с клавиатуры измените цвет содержимого на красный цвет |
| 2 | 1. Создайте многострочное поле для ввода текстовой информации. По умолчанию поместите в текстовое поле ваши ФИО, контактные данные, отделение, группу, курс, год поступления и год окончания колледжа. При прокрутке данных измените цвет содержимого на красный цвет 2. Создайте ссылку. При наведении на ссылку выведите информационное сообщение про элемент <a>. |
| 3 | 1. Создайте поле для ввода текстовой информации. По умолчанию поместите в текстовое поле ваши ФИО. При выделении данных в текстовом поле измените цвет содержимого на красный цвет 2. Создайте кнопку. При нажатии на кнопку выведите сообщение, в котором будет содержаться информация, с помощью какого события было вызвано данное окно. |
| 4 | 1. Создайте поле для ввода текстовой информации. По умолчанию поместите в текстовое поле ваши ФИО. При изменении данных в текстовом поле измените цвет содержимого на красный цвет 2. После изменения цвета, выведите информационное окно с номером вашей группы. |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Дайте определение языку JavaScript. Какими способами можно внедрить скрипт в HTML-документ?

2. Поясните суть механизма DHML.

3. Перечислите категории событий.

4. В чем различия у событий Onkeydown, Onkeypress, Onkeyup?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 12**

## **Тема работы: «Использование управляющих конструкций при написании сценариев на языке JavaScript»**

**1. Цель работы**

Формирование умений использования управляющих конструкций при написании сценариев JavaScript.

**2. Задание**

Реализовать решение задач с использованием управляющих конструкций, поддерживаемых JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Сценарий – это код, включенный в состав Web-страницы.

Основные правила синтаксиса JavaScript:

1) чувствительность к регистру. Все ключевые слова пишутся в нижнем регистре. Все переменные и названия функций пишутся точно так же, как и были определены (например, переменные Str и str являются разными переменными);

2) пробелы, табуляция и перевод строки. Эти символы игнорируются в JavaScript, так что можно использовать их для форматирования кода с тем, чтобы его было удобно читать;

3) символ точка с запятой (;). Все операторы должны быть разделены этим символом. Если оператор завершается переводом строки, то точку с запятой можно опустить. При этом нужно следить за тем, чтобы при разрыве строки одного оператора, новая строка не начиналась бы с самостоятельного оператора;

4) комментарии. JavaScript игнорирует любой текст расположенный между символами /\* и \*/. Также игнорируется текст начинающийся символами // и заканчивающийся концом строки;

5) идентификаторы. Идентификаторами являются имена переменных, функций, а также меток. Идентификаторы образуются из любого количества букв ASCII, подчеркивания (\_) и символа доллара ($). Первым символом не может быть цифра, а в версии JavaScript 1.0 не допускается использования и символа $;

6) ключевые слова. Ключевые слова не могут использоваться в качестве идентификаторов. Ключевыми словами являются: break, case, continue, default, delete, do, else, export, false, for, function, if, import, in, new, null, return, switch, this, true, typeof, with.

Для того, чтобы браузер распознал сценарий, встроенный в HTML-код, его (сценарий) следует заключать в контейнер: <script>...</script>

С помощью метода document.writeln можно выводить текст с использованием тегов форматирования документа и др. Это позволяет динамически формировать гипертекст.

Используя эти возможности, легко вывести на экран графическое изображение из указанного файла:

<SCRIPT>

document.writeln("<img src=\"logo.gif\">");

</SCRIPT>

Комментарии – это строки кода, которые не обрабатываются транслятором. Комментарии в JS бывают двух типов – однострочные и многострочные. Однострочные начинаются с символов “//” и перестают быть комментарием при переходе на следующую строку. Многострочные начинаются с символов “/\*” и заканчиваются символами “\*/”. Закрывающие комментарий символы могут находиться на строке, которая ниже текущей. Рассмотрим примеры:

<script>

// This is a single-line comment

a = 54;

/\*

This is multi-line comment.

We can insert some text here.

\*/

b = a + 1;

</script>

Переменные в JS не являются типизированными. Это означает, что при создании переменной не указывается, какого типа информация будет в ней находиться. Для того, чтобы использовать переменную, ее необходимо объявить и инициализировать.

Объявление переменной реализуется при помощи ключевого слова var.

Инициализация переменной – это присвоение переменной некоторого начального значения. Инициализация также может иметь место при объявлении переменных.

var a, b;

var x = 12;

t = 10;

В JS, как и в других языках программирования, названия переменных должны соответствовать некоторым правилам. Название переменной в JS может содержать символы английского алфавита, цифры, подчеркивания и символ $, однако не может начинаться с цифры. Также, названия переменных регистро-зависимы.

Выражение – это некоторая конструкция, которая состоит из значений, переменных (констант) и операторов. Рассмотрим основные операторы JS. Выделяют:

– арифметические;

– логические;

– сравнения.

К арифметическим операторам относятся следующие операторы: + (сложение), - (вычитание), \* (умножение), / (деление), % (взятие остатка от деления).

В отличие от некоторых других языков, оператор "+" делает конкатенацию, если хотя бы один операнд - строка, причем, не обязательно первый. К унарным операторам относятся следующие: ++ инкремент (увеличить значение переменной на 1), -- декремент (уменьшить значение переменной на 1), - изменение знака на противоположный.

Унарный оператор "+" также используется для преобразования строки к числу: alert(+"123"+1) // 124

Логические: && (логическое И), || (логическое ИЛИ), ! (логическое НЕ).

Логические операторы, как правило, используются с булевыми значения-ми и возвращают булево значение.

Однако, в JS операторы && и || возвращают не булево значение, а один из операндов, определивший значение выражения. Конечно, результат в таком случае может быть и не булевым.

«Логическое И» возвращает последний операнд, если все операнды верны. Если хоть один из операндов неверен, то возвратит первый неверный операнд.

Оператор «логического ИЛИ» возвращает первое верное значение. А если верных значений вообще нет, то последнее неверное.

Логическое НЕ, также удобно для преобразования в Boolean.

Сравнения (и отношения): == (равно), != (не равно), < (строго меньше), > (строго больше),<= (меньше или равно), >= (больше или равно), === (равно и тип данных операндов совпадает), !== (не равно или не совпадает тип данных операндов).

В javaScript условие осуществляет оператор if

Рассмотрим синтаксис условного оператора:

if (условие)

{

// если истина

operator1;

operator2;

}

else

{

// если ложь

operator3;

operator4;

}

В качестве условия может использоваться логическое выражение или переменная.

Блок операторов заключенный в {} – это составной оператор, объединяющий несколько операторов в конструкции. Скобки можно опустить, если внутри конструкции только один оператор.

Рассмотрим простой пример с использованием оператора языка javascript if: выводить в модальное окно «а больше 1», если переменная a>1, иначе выводить «а не больше 1»:

var a=1;

if (a>1)

alert("а больше 1")

else

alert("а не больше 1");

Теперь рассмотрим используемые в javascript логические операторы для построения сложных (например, двойных) условий: if (!x), if (x>1) && (x<5), if (x>1) || (y>1).

Тернарный оператор языка javascript - это сокращенная форма конструкции if ... else. Данный оператор включает в себя три операнда:

( условие ? Если условие ИСТИННО : Если условие ЛОЖНО )

Т.е. синтаксис:

(логическое выражение) ? выражение 1 : выражение 2

Пример: Переменной b присваивать значение 100 в случае истинности условия a>1 и присваивать 200 в случае ложного условия

var a = 10;

var b = (a>1) ? 100 : 200;

alert(b);

Т.е. в случае истинности условия, выполняется то, что стоит до знака «:», в случае ложности условия – выполняется то, что стоит после знака «:».

Оператор switch javascript служит для проверки переменной на множество значений:

switch (выражение) {

case вариант1:

//..блок операторов..

break

case вариант2:

//..блок операторов..

break

[default:

//..блок операторов..]

}

Блок, начинающийся со служебного слова default можно опустить. Операторы блока будут выполнены в случае, если ни одно из перечисленных значений не подходит.

Оператор break обязателен после каждого рассмотренного значения, если его не использовать, то будут выведены все, расположенные ниже операторы.

Пример:

var a =2;

switch(a)

{

case 0:

case 1:

alert("Ноль или один");

break;

case 2:

alert("Два");

break;

default:

alert("Много");

}

Цикл в javascript for используется, когда заранее известно, сколько раз должны повториться циклические действия:

for( начальное значение счетчика итераций; условие;приращение счетчика )

{

//..блок операторов..

}

В качестве начального значения счетчика итераций используется выражение присваивания: например, i=0 – счетчик цикла начинается с нуля.

В качестве приращения счетчика указывается шаг, с которым должен увеличиваться счетчик: например, i++ указывает на то, что каждая итерация цикла будет сопровождаться его увеличением на 1.

Условие цикла – это и есть конечное значение счетчика: например, i<10 – счетчик, достигнув значения 10, останавливает цикл.

Рассмотрим пример использования цикла for в javascript:

for (var i=0;i<10;i++)

{

document.write(i+"<br>");

}

В примере на экран выводятся значения счетчика цикла, так как приращение счетчика i++, соответственно на экране будут появляться 0 1 2 3 ... 9, причем каждая цифра – с новой строки (тег br).

Оператор break прерывает выполнение всего тела цикла, т.е. осуществляет выход из цикла в javaScript.

В то время как оператор continue прерывает выполнение текущей итерации цикла, но продолжая при этом выполнение цикла со следующий итерации.

Рассмотрим работу операторов break и continue на примере:

for (var i=0;i<10;i++)

{

if (i==4) continue;

document.write(i+"<br>");

if (i==8) break;

}

В третьей строке примера стоит условие, из-за которого цифра 4 не будет выводиться на экран. В строке №5 осуществляется выход из цикла, но при этом цифра 8 будет выведена на экран, так как оператор вывода стоит до условия (в 4-й строке). Т.е. на экране будет: 0 1 2 3 5 6 7 8 – каждая цифра с новой строки.

В языке javasctipt предусмотрен оператор выхода из программного кода – оператор exit.

Чаще всего оператор используется для исключения ошибки ввода пользователя. Рассмотрим пример: запрашивать пользователя ввести число. Если введено не число, то выводить сообщение "Необходимо число!" и останавливать работу программы. Для выполнения пригодятся функции преобразования типа:

var number=prompt("Введите число");

number=parseInt(number); // возвратит NaN – не число

x=isNaN(number); // возвратит true, т.к. не числовое

if (x){

alert("Необходимо число!");

exit; // выход из программы

}

alert("Введите второе число");// при вводе не числа оператор не выполнится

Циклические операторы языка javaScript – While. Синтаксис оператора while:

while (условие)

{

//..блок операторов..

};

Пример: Выводить в диалоговое окно степени двойки до 1000

var a = 1;

while (a<1000){

a\*=2;

alert(a);

}

На примере также посмотрим, как работают операторы break и continue в цикле while:

var a = 1;

while (a<1000){

a\*=2;

if (a==64)

continue;

if (a==256)

break;

alert(a);

}

Здесь степени двойки будут выводить до 128 включительно, причем пропущено будет 64. Т.е. в диалоговых окнах мы увидим: 2 4 8 16 32 128

Циклические операторы языка javaScript – цикл с постусловием do..while:

do

{

//..блок операторов..

}

while (условие);

Пример:

var a = 1;

do

{

a\*=2;

if (a==64)

continue;

alert(a);

if (a==256)

break;

}while(a<1000);

Цикл for in предназначен для прохода по массивам, коллекциям и объектам:

Пример: Распечатать свойства объекта navigator:

for (var prop in navigator)

{

document.write(prop + "<br>");

}

В примере переменная prop создана для "перебора" всех свойств объекта navigator.

В некоторых случаях код на странице не работает непонятно по какой причине. В таких случаях можно применить оператор try..catch, который пытается выполнить фрагмент кода, и, если в коде есть ошибка, то существует возможность выдать ошибку на экран.

Рассмотрим работу оператора на примере: написать в программе оператор с ошибкой. Проверять наличие ошибки в предполагаемом ошибочном коде: если ошибка в коде присутствует – выдавать сообщение "обработка ошибки: название ошибки". После проверки ошибочного оператора, независимо от того, есть ли в коде ошибка, выдавать сообщение "завершающие действия":

alert("до");

try

{

sdfsdf; // оператор с ошибкой

}

catch(e)

{

alert("обработка ошибки: "+e.message);

}

finally

{

alert("завершающие действия");

}

alert("после");

Try с англ. – "пытаться", таком образом, ставим оператор try перед фрагментом кода, который, возможно, содержит ошибку. Если ошибка действительно есть, то оператор catch (с англ. "ловить") сохраняет эту ошибку в объекте e. В дальнейшем ее можно вывести в диалоговое окно – e.message. Если ошибка все же есть, то интерпретатор после ее вывода в нашем примере перейдет на выполнение блока catch, а затем finally (с английского "завершение", "наконец"), который выполнится всегда, независимо от того была ли ошибка или нет. Даже если возникла ошибка в блоке catch.

Блок finally в конструкции необязателен.

Для управления HTML-страницами и их элементами JavaScript использует объектную модель документов DOM (Document Object Model). Упрощенная схема объектной модели представлена на рисунке 12.1.

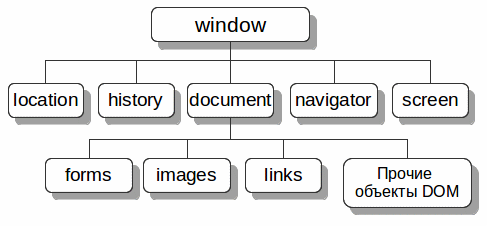


Рисунок 12.1 – Объектная модель документа

Объекты появляются после загрузки документа браузером или как результат работы программы. Каждый объект характеризуется набором методов, свойств и возможных событий.

Свойства объектов соответствуют атрибутам определенных тегов; их использование позволяет получить или изменить характеристики окна браузера, загруженных документов и элементов HTML-страницы. Например, к свойствам изображения относятся src, width, height и т. д.

Методы объекта представляют собой действия, которые могут быть выполнены по отношению к данному объекту. Например, одним из методов объекта document является write, который помещает в документ определенный текст.

Создадим документ pr\_DOM.html:

<HTML>

<HEAD> <TITLE> DOM </TITLE> </HEAD>

<BODY>

<FORM name=Sum>

Первое число <INPUT type="text" name="number1"><BR> Второе число <INPUT type="text" name="number2"><BR>

<INPUT type="button" name="but"

value="Вычислить сумму"><BR><BR> Сумма <INPUT type="text" name="result">

</FORM>

</BODY>

</HTML>

При открытии данного документа создается объектная модель, представленная на рисунке 12.2.

**window**

**document**

**forms[0] name=Sum**

**elements[0] name=number1**

**elements[1] name=number2**

**elements[2] name=but**

**elements[3] name=result**

Рисунок 12.2 – Иерархия объектов документа pr\_DOM.html

Изменим описание кнопки на странице следующим образом:

<input type="button" name="but" value=" Вычислить сумму "

onClick="document.forms[0].elements[3].value= parseInt(document.forms[0].elements[0].value) + parseInt(document.forms[0].elements[1].value);">

В результате изменений при нажатии кнопки значения, введенные пользователем в два первых текстовых поля, будут переводиться в числовой тип данных (функцией parseInt) и суммироваться; результат будет помещен в третье текстовое поле.

Объект window можно не указывать (как в данном примере), если имеется в виду текущее окно браузера.

Для обращения к объектам страницы можно также использовать имя, указанное в атрибуте name.

Например, обработку нажатия кнопки можно определить следующим образом:

<input type="button" name="but" value=" Вычислить сумму "

onClick="document.Sum.result.value=parseInt(document.Sum.number1.value) + parseInt(document.Sum.number2.value);">

Для расчета суммы можно определить отдельную функцию summa( ) и вызвать ее при помощи атрибута onClick:

<html>

<head>

<title> DOM </title>

<script>

// определяем функцию summa()

function summa(){

// объявляем переменные n1 и n2

var n1=parseInt(document.Sum.number1.value);

var n2=parseInt(document.Sum.number2.value);

/\*рассчитываем сумму и записываем ее в поле result\*/

document.Sum.result.value=n1+n2;

}

</script>

</head>

<body>

<form name=Sum>

...

<input type="button" name="but" value="Вычислить сумму" onClick="summa()">

...

</form>

</body>

</html>

**5. Порядок выполнения работы**

1. Реализовать алгоритм: программа просит ввести число от 1 до 100 первого игрока (второй игрок не видит введенное число). Затем второго игрока просит угадать введенное число. В ответ выводится сообщение «мало» либо «много» в зависимости от введенного ответа. Если игрок угадывает, – выводится поздравление. Если не угадывает – игра продолжается. Просчитывать число попыток и выдавать результат, когда число разгадано.

2. Реализуйте пример, представленный на рисунке 12.3.



Рисунок 12.3 – Использование арифметических операторов

3. Выполните задание по варианту. Варианты представлены в таблице 12.1.

Таблица 12.1 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| 1 | 1. Решить уравнение вида ax^2+bx + c = d. Значения a, b, c, d вводятся с клавиатуры. 2. Рассчитайте сумму двух чисел. Обеспечьте валидность данных |
| 2 | 1. Вывести текст, введенный пользователем перевернув его. 2. Рассчитайте разность двух чисел. Обеспечьте валидность данных |
| 3 | 1. Рассчитать дискриминант квадратного уравнения вида ax^2+bx + c = d. Значения a, b, c, d вводятся с клавиатуры. 2. Рассчитайте произведение двух чисел. Обеспечьте валидность данных |
| 4 | 1. Проверить число, введенное с клавиатуры простое ли оно. 2. Определите частное двух чисел. Обеспечьте валидность данных |
| 5 | 1. Определите результат вычисления арифметического выражения, введенного в текстовое поле 2. Найдите наименьший общий делитель двух чисел |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Запишите синтаксис операторов if и switch?

2. Чем отличается оператор while от оператора do while?

3. В каких случаях лучше использовать оператор for?

4. Для чего служат ключевые слова continue и break?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 13**

## **Тема работы: «Работа с массивами в JavaScript. Применение методов работы с массивами»**

**1. Цель работы**

Формирование умений работы с массивами в JavaScript и использования стандартных методов для работы с массивами.

**2. Задание**

Реализовать использование методов работы с массивами в сценариях на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Массив – разновидность объекта, которая предназначена для хранения пронумерованных значений и предлагает дополнительные методы для удобного манипулирования такой коллекцией.

Они обычно используются для хранения упорядоченных коллекций данных, например – списка товаров на странице, студентов в группе и т.п.

Синтаксис для создания нового массива – квадратные скобки со списком элементов внутри.

Пустой массив: var arr = [];

Массив fruits с тремя элементами:

var fruits = ["Яблоко", "Апельсин", "Слива"];

Элементы нумеруются, начиная с нуля.

Чтобы получить нужный элемент из массива – указывается его номер в квадратных скобках:

var fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits[0] ); // Яблоко

alert( fruits[1] ); // Апельсин

alert( fruits[2] ); // Слива

Элемент можно всегда заменить:

fruits[2] = 'Груша'; // теперь ["Яблоко", "Апельсин", "Груша"]

Или добавить:

fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]

Общее число элементов, хранимых в массиве, содержится в его свойстве length:

var fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.length ); // 3

Через alert можно вывести и массив целиком.

При этом его элементы будут перечислены через запятую:

var fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits ); // Яблоко,Апельсин,Груша

В массиве может храниться любое число элементов любого типа.

В том числе, строки, числа, объекты, вот например:

// микс значений

var arr = [ 1, 'Имя', { name: 'Петя' }, true ];

// получить объект из массива и тут же -- его свойство

alert( arr[2].name ); // Петя

Одно из применений массива – это очередь. В классическом программировании так называют упорядоченную коллекцию элементов, такую, что элементы добавляются в конец, а обрабатываются – с начала, рисунок 13.1.



Рисунок 13.1 – Принцип работы очереди

В реальной жизни эта структура данных встречается очень часто. Например, очередь сообщений, которые надо показать на экране.

Очень близка к очереди еще одна структура данных: стек. Это такая коллекция элементов, в которой новые элементы добавляются в конец и берутся с конца,рисунок 13.2.

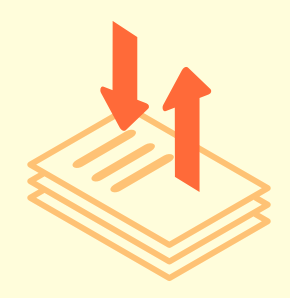


Рисунок 13.2 – Принцип работы стека

Для того, чтобы реализовывать эти структуры данных, и просто для более удобной работы с началом и концом массива существуют специальные методы.

Pop – удаляет последний элемент из массива и возвращает его:

var fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.pop() ); // удалили "Груша"

alert( fruits ); // Яблоко, Апельсин

Рush – добавляет элемент в конец массива:

var fruits = ["Яблоко", "Апельсин"];

fruits.push("Груша");

alert( fruits ); // Яблоко, Апельсин, Груша

Вызов fruits.push(...) равнозначен fruits[fruits.length] = ....

Shift – удаляет из массива первый элемент и возвращает его:

var fruits = ["Яблоко", "Апельсин", "Груша"];

alert( fruits.shift() ); // удалили Яблоко

alert( fruits ); // Апельсин, Груша

Unshift – добавляет элемент в начало массива:

var fruits = ["Апельсин", "Груша"];

fruits.unshift('Яблоко');

alert( fruits ); // Яблоко, Апельсин, Груша

Методы push и unshift могут добавлять сразу по несколько элементов:

var fruits = ["Яблоко"];

fruits.push("Апельсин", "Персик");

fruits.unshift("Ананас", "Лимон");

// результат: ["Ананас", "Лимон", "Яблоко", "Апельсин", "Персик"]

alert( fruits );

Массив – это объект, где в качестве ключей выбраны цифры, с дополнительными методами и свойством length.

Так как это объект, то в функцию он передаётся по ссылке:

function eat(arr) {

arr.pop();

}

var arr = ["нам", "не", "страшен", "серый", "волк"]

alert( arr.length ); // 5

eat(arr);

eat(arr);

alert( arr.length ); // 3, в функцию массив не скопирован, а передана ссылка

В массив можно присваивать любые свойства. Например:

var fruits = []; // создать массив

fruits[99999] = 5; // присвоить свойство с любым номером

fruits.age = 25; // назначить свойство со строковым именем

Но массивы для того и придуманы в JavaScript, чтобы удобно работать именно с упорядоченными, нумерованными данными. Для этого в них существуют специальные методы и свойство length.

Как правило, нет причин использовать массив как обычный объект, хотя технически это и возможно.

Если в массиве есть пропущенные индексы, то при выводе в большинстве браузеров появляются «лишние» запятые, например:

var a = [];

a[0] = 0;

a[5] = 5;

alert( a ); // 0,,,,,5

Эти запятые появляются потому, что алгоритм вывода массива идёт от 0 до arr.length и выводит всё через запятую. Отсутствие значений даёт несколько запятых подряд.

Методы push/pop выполняются быстро, а shift/unshift – медленно.

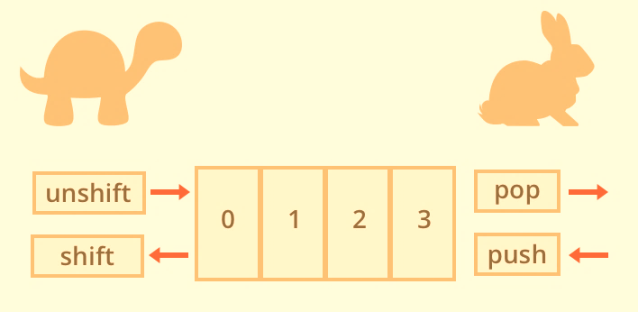


Рисунок 13.3 – Различия в скорости работы методов push/pop и shift/unshift

Чтобы понять, почему работать с концом массива – быстрее, чем с его началом, разберём подробнее происходящее при операции:

fruits.shift(); // убрать 1 элемент с начала

При этом, так как все элементы находятся в своих ячейках, просто удалить элемент с номером 0 недостаточно. Нужно еще и переместить остальные элементы на их новые индексы.

Операция shift должна выполнить целых три действия, как показано на рисунке 13.4:

– удалить нулевой элемент;

– переместить все свойства влево, с индекса 1 на 0, с 2 на 1 и так далее;

– обновить свойство length.

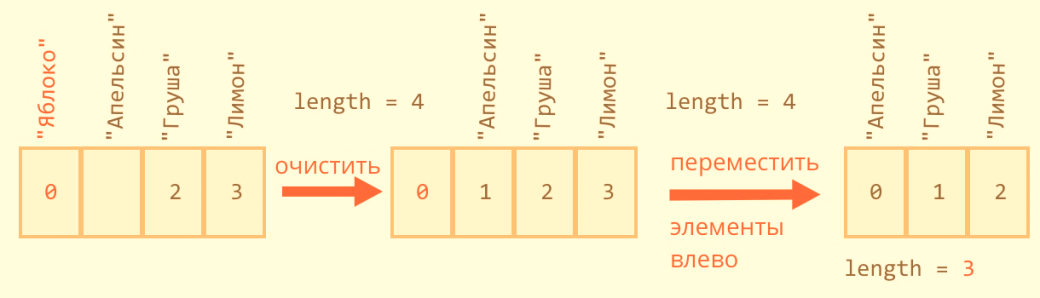


Рисунок 13.4 – Алгоритм выполнения операции shift

Чем больше элементов в массиве, тем дольше их перемещать, это много операций с памятью.

Аналогично работает unshift: чтобы добавить элемент в начало массива, нужно сначала перенести вправо, в увеличенные индексы, все существующие.

Для того, чтобы удалить элемент, метод pop очищает ячейку и уменьшает значение length.

Действия при операции,рисунок 13.5:

fruits.pop(); // убрать 1 элемент с конца

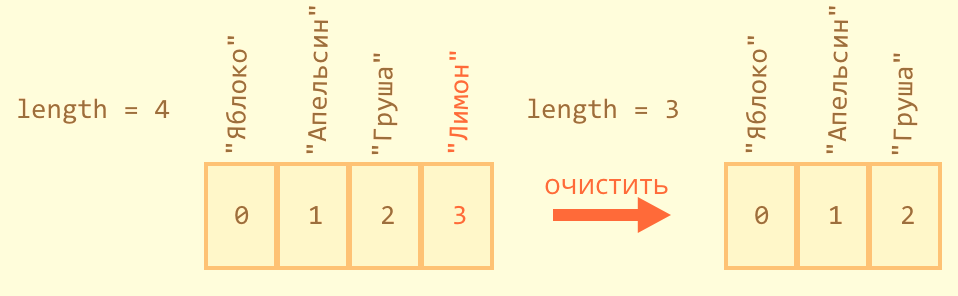


Рисунок 13.5 – Алгоритм выполнения операции pop

Перемещать при pop не требуется, так как прочие элементы после этой операции остаются на тех же индексах.

Аналогично работает push.

Для перебора элементов обычно используется цикл:

var arr = ["Яблоко", "Апельсин", "Груша"];

for (var i = 0; i < arr.length; i++) {

alert( arr[i] );

}

Не используйте for..in для массивов

Так как массив является объектом, то возможен и вариант for..in:

var arr = ["Яблоко", "Апельсин", "Груша"];

for (var key in arr) {

alert( arr[key] ); // Яблоко, Апельсин, Груша

}

Недостатки этого способа: цикл for..in выведет все свойства объекта, а не только цифровые.

В браузере, при работе с объектами страницы, встречаются коллекции элементов, которые по виду как массивы, но имеют дополнительные нецифровые свойства. При переборе таких «похожих на массив» коллекций через for..in эти свойства будут выведены, а они как раз не нужны.

Бывают и библиотеки, которые предоставляют такие коллекции. Классический for надёжно выведет только цифровые свойства, что обычно и требуется.

Цикл for (var i=0; i<arr.length; i++) в современных браузерах выполняется в 10-100 раз быстрее.

Встроенные методы для работы с массивом автоматически обновляют его длину length.

Длина length – не количество элементов массива, а последний индекс +1:

var arr = [];

arr[1000] = true;

alert(arr.length); // 1001

При уменьшении length массив укорачивается.

Причем этот процесс необратимый, т.е. даже если потом вернуть length обратно – значения не восстановятся:

var arr = [1, 2, 3, 4, 5];

arr.length = 2; // укоротить до 2 элементов

alert( arr ); // [1, 2]

arr.length = 5; // вернуть length обратно, как было

alert( arr[3] ); // undefined: значения не вернулись

Самый простой способ очистить массив – это arr.length=0.

Существует еще один синтаксис для создания массива:

var arr = new Array("Яблоко", "Груша", "и т.п.");

Он редко используется, т.к. квадратные скобки [] короче.

Кроме того, у него есть одна особенность. Обычно new Array(элементы, ...) создаёт массив из данных элементов, но если у него один аргумент-число new Array(число), то он создает массив без элементов, но с заданной длиной.

Проверим это:

var arr = new Array(2, 3);

alert( arr[0] ); // 2, создан массив [2, 3], всё ок

arr = new Array(2); // создаст массив [2] ?

alert( arr[0] ); // undefined! у нас массив без элементов, длины 2

Получившийся массив ведёт себя так, как будто его элементы равны undefined.

Массивы в JavaScript могут содержать в качестве элементов другие массивы. Это можно использовать для создания многомерных массивов, например матриц:

var matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

alert( matrix[1][1] ); // центральный элемент

Метод split(s) позволяет превратить строку в массив, разбив ее по разделителю «s». В примере ниже таким разделителем является строка из запятой и пробела:

var names = 'Маша, Петя, Марина, Василий';

var arr = names.split(', ');

for (var i = 0; i < arr.length; i++) {

alert( 'Вам сообщение ' + arr[i] );

}

У метода split есть необязательный второй аргумент – ограничение на количество элементов в массиве. Если их больше, чем указано – остаток массива будет отброшен:

alert( "a,b,c,d".split(',', 2) ); // a,b

Вызов split с пустой строкой разобьёт по буквам:

var str = "тест";

alert( str.split('') ); // т,е,с,т

Вызов arr.join(str) делает в точности противоположное split. Он берет массив и склеивает его в строку, используя str как разделитель.

Например:

var arr = ['Маша', 'Петя', 'Марина', 'Василий'];

var str = arr.join(';');

alert( str ); // Маша;Петя;Марина;Василий

new Array + join = Повторение строки

Код для повторения строки 3 раза:

alert( new Array(4).join("ля") ); // ляляля

Как видно, new Array(4) делает массив без элементов длины 4, который join объединяет в строку, вставляя между его элементами строку "ля".

В результате, так как элементы пусты, получается повторение строки.

Так как массивы являются объектами, то для удаления ключа можно воспользоваться обычным delete:

var arr = ["Я", "иду", "домой"];

delete arr[1]; // значение с индексом 1 удалено

// теперь arr = ["Я", undefined, "домой"];

alert( arr[1] ); // undefined

Оператор delete удаляет пару «ключ-значение». Это – все, что он делает. Обычно же, при удалении из массива мы хотим, чтобы оставшиеся элементы сдвинулись и заполнили образовавшийся промежуток.

Поэтому для удаления используются специальные методы: из начала – shift, с конца – pop, а из середины – splice.

Метод splice – это универсальный раскладной нож для работы с массивами. Умеет все: удалять элементы, вставлять элементы, заменять элементы – по очереди и одновременно.

Его синтаксис: arr.splice(index[, deleteCount, elem1, ..., elemN])

Удалить deleteCount элементов, начиная с номера index, а затем вставить elem1, ..., elemN на их место. Возвращает массив из удалённых элементов.

Этот метод проще всего понять, рассмотрев примеры.

Начнём с удаления:

var arr = ["Я", "изучаю", "JavaScript"];

arr.splice(1, 1); // начиная с позиции 1, удалить 1 элемент

alert( arr ); // осталось ["Я", "JavaScript"]

В следующем примере мы удалим 3 элемента и вставим другие на их место:

var arr = ["Я", "сейчас", "изучаю", "JavaScript"];

// удалить 3 первых элемента и добавить другие вместо них

arr.splice(0, 3, "Мы", "изучаем")

alert( arr ) // теперь ["Мы", "изучаем", "JavaScript"]

Здесь видно, что splice возвращает массив из удаленных элементов:

var arr = ["Я", "сейчас", "изучаю", "JavaScript"];

// удалить 2 первых элемента

var removed = arr.splice(0, 2);

alert( removed ); // "Я", "сейчас" <-- array of removed elements

Метод splice также может вставлять элементы без удаления, для этого достаточно установить deleteCount в 0:

var arr = ["Я", "изучаю", "JavaScript"];

// с позиции 2

// удалить 0

// вставить "сложный", "язык"

arr.splice(2, 0, "сложный", "язык");

alert( arr ); // "Я", "изучаю", "сложный", "язык", "JavaScript"

Допускается использование отрицательного номера позиции, которая в этом случае отсчитывается с конца:

var arr = [1, 2, 5]

// начиная с позиции индексом -1 (перед последним элементом)

// удалить 0 элементов,

// затем вставить числа 3 и 4

arr.splice(-1, 0, 3, 4);

alert( arr ); // результат: 1,2,3,4,5

Метод slice(begin, end) копирует участок массива от begin до end, не включая end. Исходный массив при этом не меняется.

Например:

var arr = ["Почему", "надо", "учить", "JavaScript"];

var arr2 = arr.slice(1, 3); // элементы 1, 2 (не включая 3)

alert( arr2 ); // надо, учить

Аргументы ведут себя так же, как и в строковом slice:

Если не указать end – копирование будет до конца массива:

var arr = ["Почему", "надо", "учить", "JavaScript"];

alert( arr.slice(1) ); // взять все элементы, начиная с номера 1

Можно использовать отрицательные индексы, они отсчитываются с конца:

var arr2 = arr.slice(-2); // копировать от 2-го элемента с конца и дальше

Если вообще не указать аргументов – скопируется весь массив:

var fullCopy = arr.slice();

Синтаксис метода slice одинаков для строк и для массивов. Тем проще его запомнить.

Метод sort() сортирует массив на месте. Например:

var arr = [ 1, 2, 15 ];

arr.sort();

alert( arr ); // 1, 15, 2

Порядок стал 1, 15, 2. Это произошло потому, что по умолчанию sort сортирует, преобразуя элементы к строке. Поэтому и порядок у них строковый, ведь "2" > "15". Для указания своего порядка сортировки в метод arr.sort(fn) нужно передать функцию fn от двух элементов, которая умеет сравнивать их.

Внутренний алгоритм функции сортировки умеет сортировать любые массивы – апельсинов, яблок, пользователей, и тех и других и третьих – чего угодно. Но для этого ему нужно знать, как их сравнивать. Эту роль и выполняет fn.

Если эту функцию не указать, то элементы сортируются как строки. Например, укажем эту функцию явно, отсортируем элементы массива как числа:

function compareNumeric(a, b) {

if (a > b) return 1;

if (a < b) return -1;

}

var arr = [ 1, 2, 15 ];

arr.sort(compareNumeric);

alert(arr); // 1, 2, 15

Обратите внимание, мы передаём в sort() именно саму функцию compareNumeric, без вызова через скобки. Был бы ошибкой следующий код:

arr.sort( compareNumeric() ); // не сработает

Как видно из примера выше, функция, передаваемая sort, должна иметь два аргумента.

Алгоритм сортировки, встроенный в JavaScript, будет передавать ей для сравнения элементы массива. Она должна возвращать:

– положительное значение, если a > b;

– отрицательное значение, если a < b.

Если равны – можно 0, но вообще – не важно, что возвращать, если их взаимный порядок не имеет значения.

В методе sort, внутри самого интерпретатора JavaScript, реализован универсальный алгоритм сортировки. Как правило, это ««быстрая сортировка»», дополнительно оптимизированная для небольших массивов.

Он решает, какие пары элементов и когда сравнивать, чтобы отсортировать побыстрее. Мы даём ему функцию – способ сравнения, дальше он вызывает её сам.

Кстати, те значения, с которыми sort вызывает функцию сравнения, можно увидеть, если вставить в неё alert:

[1, -2, 15, 2, 0, 8].sort(function(a, b) {

alert( a + " <> " + b );

});

Функцию compareNumeric для сравнения элементов-чисел можно упростить до одной строчки:

function compareNumeric(a, b) {

return a - b;

}

Эта функция вполне подходит для sort, так как возвращает положительное число, если a > b, отрицательное, если наоборот, и 0, если числа равны.

Метод arr.reverse() меняет порядок элементов в массиве на обратный:

var arr = [1, 2, 3];

arr.reverse();

alert( arr ); // 3,2,1

Метод arr.concat(value1, value2, … valueN) создаёт новый массив, в который копируются элементы из arr, а также value1, value2, ... valueN.

Например:

var arr = [1, 2];

var newArr = arr.concat(3, 4);

alert( newArr ); // 1,2,3,4

Если аргумент concat – массив, то concat добавляет элементы из него.

Например:

var arr = [1, 2];

var newArr = arr.concat([3, 4], 5); // то же самое, что arr.concat(3,4,5)

alert( newArr ); // 1,2,3,4,5

indexOf/lastIndexOf – эти методы не поддерживаются в IE8-. Для их поддержки подключите библиотеку ES5-shim.

Метод «arr.indexOf(searchElement[, fromIndex])» возвращает номер элемента searchElement в массиве arr или -1, если его нет.

Поиск начинается с номера fromIndex, если он указан. Если нет – с начала массива.

Для поиска используется строгое сравнение ===. Например:

var arr = [1, 0, false];

alert( arr.indexOf(0) ); // 1

alert( arr.indexOf(false) ); // 2

alert( arr.indexOf(null) ); // -1

Метод «arr.lastIndexOf(searchElement[, fromIndex])» ищет справа-налево: с конца массива или с номера fromIndex, если он указан.

Методы indexOf/lastIndexOf осуществляют поиск перебором

Если нужно проверить, существует ли значение в массиве – его нужно перебрать. Только так. Внутренняя реализация indexOf/lastIndexOf осуществляет полный перебор, аналогичный циклу for по массиву. Чем длиннее массив, тем дольше он будет работать.

Рассмотрим задачу – есть коллекция строк, и нужно быстро проверять: есть ли в ней какой-то элемент. Массив для этого не подходит из-за медленного indexOf. Но подходит объект! Доступ к свойству объекта осуществляется очень быстро, так что можно сделать все элементы ключами объекта и проверять, есть ли уже такой ключ.

Например, организуем такую проверку для коллекции строк "div", "a" и "form":

var store = {}; // объект для коллекции

var items = ["div", "a", "form"];

for (var i = 0; i < items.length; i++) {

var key = items[i]; // для каждого элемента создаём свойство

store[key] = true; // значение здесь не важно

}

Теперь для проверки, есть ли ключ key, достаточно выполнить if (store[key]). Если есть – можно использовать значение, если нет – добавить.

Такое решение работает только со строками, но применимо к любым элементам, для которых можно вычислить строковый «уникальный ключ».

Object.keys(obj)

Если мы хотим работатьсвойствами массива в виде массива, можно воспользоваться методом Object.keys(obj). Он поддерживается везде, кроме IE8-:

var user = {

name: "Петя",

age: 30

}

var keys = Object.keys(user);

alert( keys ); // name, age

**5. Порядок выполнения работы**

1. Дан массив. Выведите его на страницу HTML в формате: индекс элемента – значение (через три дефиса). Каждый элемент должен быть выведен с новой строки.

2. Напишите функцию, которая копирует массив не изменяя оригинал.

3. Выполните задание по варианту. Варианты представлены в таблице 13.1.

Таблица 13.1 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| 1 | Напишите код, который преобразовывает и объединяет все элементы массива в одно строковое значение. Элементы массива будут разделены запятой. Получите результат двумя разными методами. |
| 2 | Пользователь вводит многозначное число. Напишите код, в котором вставляет двоеточие (:) между двумя нечетными числами. Например, если вводится число 55639217, то на выход должно быть 5:563:921:7. |
| 3 | Напишите код, с помощью которого возвращается массив, в котором удалены повторяющиеся элементы из первоначального массива (игнорируйте чувствительность к регистру). |
| 4 | Напишите код, с помощью которого возвращается фрагмент массива, содержащий первые Х элементов массива. Число Х вводить с клавиатуры. |
| 5 | Напишите код, с помощью которого возвращается фрагмент массива, содержащий последние Х элементов массива. Число Х вводить с клавиатуры. |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Как методы push/pop, shift/unshift влияют на быстродействие JS?

2. Как получить последний элемент из произвольного массива?

3. Для чего служит свойство length?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 14**

## **Тема работы: «Использование функций при процедурном подходе в программировании на языке JavaScript»**

**1. Цель работы**

Формирование умений объявления и вызова пользовательских функции в JavaScript.

**2. Задание**

Реализовать использование пользовательских функции при написании сценариев на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Функция в JavaScript – специальный тип объектов, позволяющий формализовать средствами языка определённую логику поведения и обработки данных.

В Javascript функции являются полноценными объектами встроенного класса Function. Поэтому их можно присваивать переменным, передавать и, конечно, у них есть свойства.

Пример объявления функции:

function showMessage() {

alert( 'Привет всем присутствующим!' );

}

Вначале идет ключевое слово function, после него имя функции, затем список параметров в скобках (в примере выше он пустой) и тело функции – код, который выполняется при её вызове.

Объявленная функция доступна по имени, например:

function showMessage() {

alert( 'Привет всем присутствующим!' );

}

showMessage();

showMessage();

Этот код выведет сообщение два раза. Уже здесь видна главная цель создания функций: избавление от дублирования кода.

Если понадобится поменять сообщение или способ его вывода – достаточно изменить его в одном месте: в функции, которая его выводит.

Существует три способа объявления функции: Function Declaration, Function Expression и Named Function Expression. Различия в использовании функциий в зависимости от способа объявления функции приведены в таблице 14.1.

Таблица 14.1 – Различия в использовании функциий в зависимости от способа объявления

|  |  |
| --- | --- |
| **Именованные(FunctionDeclaration)** | **Анонимные (FunctionExpression)** |
| function имя(параметры) { ... } | var имя = function(параметры) { … } ... var имя = new Function(параметры, '...') |
| Именованные функции доступны везде в области видимости | Анонимные - доступны только с момента объявления. Синтаксис new Functionиспользуется редко, в основном для получения функции из текста, например, динамически загруженного с сервера в процессе выполнения скриптов. |
| |  |  |  | | --- | --- | --- | | 1 | /\* функция sum | | | 2 | определена ниже | | | | 3 | \*/ | | | | 4 | var a = sum(2,2) | | | | | 5 | function sum(x,y) { | | | | | | 6 | return x+y | | | | | | 7 | } | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | /\* будет ошибка, | | | | | 2 | т.к sum еще не существует | | | | | 3 | \*/ | | | | 4 | var a = sum(2,2) | | | | 5 | var sum = function(x,y) { | | | 6 | return x+y | | 7 | } | |

Функция может содержать локальные переменные, объявленные через var. Такие переменные видны только внутри функции:

function showMessage() {

var message = 'Привет, я - Вася!'; // локальная переменная

alert( message );

}

showMessage(); // 'Привет, я - Вася!'

alert( message ); // <-- будет ошибка, т.к. переменная видна только внутри

Блоки if/else, switch, for, while, do..while не влияют на область видимости переменных.

При объявлении переменной в таких блоках, она всё равно будет видна во всей функции.

Например:

function count() {

// переменные i,j не будут уничтожены по окончании цикла

for (var i = 0; i < 3; i++) {

var j = i \* 2;

}

alert( i ); // i=3, последнее значение i, при нём цикл перестал работать

alert( j ); // j=4, последнее значение j, которое вычислил цикл

}

Неважно, где именно в функции и сколько раз объявляется переменная. Любое объявление срабатывает один раз и распространяется на всю функцию.

Объявления переменных в примере выше можно передвинуть вверх, это ни на что не повлияет:

function count() {

var i, j; // передвинули объявления var в начало

for (i = 0; i < 3; i++) {

j = i \* 2;

}

alert( i ); // i=3

alert( j ); // j=4

}

Функция может обратиться ко внешней переменной, например:

var userName = 'Вася';

function showMessage() {

var message = 'Привет, я ' + userName;

alert(message);

}

showMessage(); // Привет, я Вася

Доступ возможен не только на чтение, но и на запись. При этом, так как переменная внешняя, то изменения будут видны и снаружи функции:

var userName = 'Вася';

function showMessage() {

userName = 'Петя'; // (1) присвоение во внешнюю переменную

var message = 'Привет, я ' + userName;

alert( message );

}

showMessage();

alert( userName ); // Петя, значение внешней переменной изменено функцией

Конечно, если бы внутри функции, в строке (1), была бы объявлена своя локальная переменная var userName, то все обращения использовали бы её, и внешняя переменная осталась бы неизменной.

Переменные, объявленные на уровне всего скрипта, называют «глобальными переменными».

В примере выше переменная userName – глобальная.

Делайте глобальными только те переменные, которые действительно имеют общее значение для вашего проекта, а нужные для решения конкретной задачи – пусть будут локальными в соответствующей функции.

В старом стандарте JavaScript существовала возможность неявного объявления переменных присвоением значения.

Например:

function showMessage() {

message = 'Привет'; // без var!

}

showMessage();

alert( message ); // Привет

В коде выше переменная message нигде не объявлена, а сразу присваивается. Таким образом, переменная message становиться глобальной.

При use strict такой код привёл бы к ошибке, но без него переменная будет создана автоматически, причём в примере выше она создаётся не в функции, а на уровне всего скрипта.

Здесь опасность даже не в автоматическом создании переменной, а в том, что глобальные переменные должны использоваться тогда, когда действительно нужны «общескриптовые» параметры.

При вызове функции ей можно передать данные, которые та использует по своему усмотрению.

Например, этот код выводит два сообщения:

function showMessage(from, text) { // параметры from, text

from = "\*\* " + from + " \*\*"; // здесь может быть сложный код оформления

alert(from + ': ' + text);

}

showMessage('Маша', 'Привет!');

showMessage('Маша', 'Как дела?');

Параметры копируются в локальные переменные функции.

Например, в коде ниже есть внешняя переменная from, значение которой при запуске функции копируется в параметр функции с тем же именем. Далее функция работает уже с параметром:

function showMessage(from, text) {

from = '\*\*' + from + '\*\*'; // меняем локальную переменную from

alert( from + ': ' + text );

}

var from = "Маша";

showMessage(from, "Привет");

alert( from ); // старое значение from без изменений, в функции была изменена копия

Функцию можно вызвать с любым количеством аргументов.

Если параметр не передан при вызове – он считается равным undefined.

Например, функцию показа сообщения showMessage(from, text) можно вызвать с одним аргументом:

showMessage("Маша");

При этом можно проверить, и если параметр не передан – присвоить ему значение «по умолчанию»:

function showMessage(from, text) {

if (text === undefined) {

text = 'текст не передан';

}

alert( from + ": " + text );

}

showMessage("Маша", "Привет!"); // Маша: Привет!

showMessage("Маша"); // Маша: текст не передан

При объявлении функции необязательные аргументы, как правило, располагают в конце списка.

Для указания значения «по умолчанию», то есть, такого, которое используется, если аргумент не указан, используется два способа:

Можно проверить, равен ли аргумент undefined, и если да – то записать в него значение по умолчанию. Этот способ продемонстрирован в примере выше.

Использовать оператор ||:

function showMessage(from, text) {

text = text || 'текст не передан';

...

}

Второй способ считает, что аргумент отсутствует, если передана пустая строка, 0, или вообще любое значение, которое в логическом контексте является false.

Если аргументов передано больше, чем надо, например showMessage("Маша", "привет", 1, 2, 3), то ошибки не будет. Но, чтобы получить такие «лишние» аргументы, нужно будет прочитать их из специального объекта arguments.

В javascript функции являются полноценными объектами встроенного класса Function. Именно поэтому их можно присваивать переменным, передавать и, конечно, у них есть свойства:

function f() {

...

}

f.test = 6

...

alert(f.test) // 6

Свойства функции доступны и внутри функции, так что их можно использовать как статические переменные.

Например:

function func() {

var funcObj = arguments.callee

funcObj.test++

alert(funcObj.test)

}

func.test = 1

func()

func()

В начале работы каждая функция создает внутри себя переменную arguments и присваивает arguments.callee ссылку на себя. Так что arguments.callee.test – свойство func.test, т.е. статическая переменная test.

В примере нельзя было сделать присвоение:

var test = arguments.callee.test

test++

так как при этом операция ++ сработала бы на локальной переменной test, а не на свойстве test объекта функции.

Объект arguments также содержит все аргументы и может быть преобразован в массив (хотя им не является).

Функция может возвратить результат, который будет передан в вызвавший её код.

Например, создадим функцию calcD, которая будет возвращать дискриминант квадратного уравнения по формуле b2 – 4ac:

function calcD(a, b, c) {

return b\*b - 4\*a\*c;

}

var test = calcD(-4, 2, 1);

alert(test); // 20

Для возврата значения используется директива return.

Она может находиться в любом месте функции. Как только до неё доходит управление – функция завершается и значение передается обратно.

Вызовов return может быть и несколько, например:

function checkAge(age) {

if (age > 18) {

return true;

} else {

return confirm('Родители разрешили?');

}

}

var age = prompt('Ваш возраст?');

if (checkAge(age)) {

alert( 'Доступ разрешен' );

} else {

alert( 'В доступе отказано' );

}

0Директива return может также использоваться без значения, чтобы прекратить выполнение и выйти из функции:

function showMovie(age) {

if (!checkAge(age)) {

return;

}

alert( "Фильм не для всех" ); // (\*)

// ...

}

В коде выше, если сработал if, то строка (\*) и весь код под ней никогда не выполнится, так как return завершает выполнение функции.

В случае, когда функция не вернула значение или return был без аргументов, считается что она вернула undefined:

function doNothing() { /\* пусто \*/ }

alert( doNothing() ); // undefined

Обратите внимание, никакой ошибки нет. Просто возвращается undefined.

Ещё пример, на этот раз с return без аргумента:

function doNothing() {

return;

}

alert( doNothing() === undefined ); // true

Имя функции следует тем же правилам, что и имя переменной. Основное отличие – оно должно быть глаголом, т.к. функция – это действие.

Как правило, используются глагольные префиксы, обозначающие общий характер действия, после которых следует уточнение.

Функции, которые начинаются с "show" – что-то показывают:

showMessage(..) // префикс show, "показать" сообщение

Функции, начинающиеся с "get" – получают, и т.п.:

getAge(..) // get, "получает" возраст

calcD(..) // calc, "вычисляет" дискриминант

createForm(..) // create, "создает" форму

checkPermission(..) // check, "проверяет" разрешение, возвращает true/false

Это очень удобно, поскольку взглянув на функцию – мы уже примерно представляем, что она делает, даже если функцию написал совсем другой человек, а в отдельных случаях – и какого вида значение она возвращает.

Каждая функция, точнее даже каждый запуск функции задает свою индивидуальную область видимости.

Переменные можно объявлять в любом месте. Ключевое слово var задает переменную в текущей области видимости. Если его забыть, как показано на рисунке 14.1, то переменная попадет в глобальный объект window. Возможны неожиданные пересечения с другими переменными окна, конфликты и глюки.

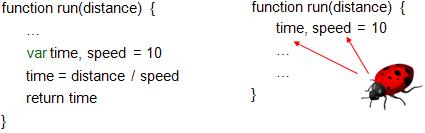


Рисунок 14.1 – Использование ключевого слова var при объявлении переменной

В отличие от ряда языков, блоки не задают отдельную область видимости. Без разницы – определена переменная внутри блока или вне его. Так что, два фрагмента на рисунке 14.2 эквивалентны:

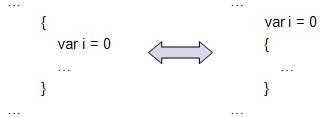


Рисунок 14.2 – Область видимости переменных функции

Заданная через var переменная видна везде в области видимости, даже до оператора var. Для примера сделаем функцию, которая будет менять переменную, var для которой находится ниже.

function a() {

z = 5 // поменяет z локально..

// .. т.к z объявлена через var

var z

}

// тест

delete z // очистим на всякий случай глобальную z

a()

alert(window.z) // => undefined, т.к z была изменена локально

Замыкание – это выражение (обычно функция), которое может иметь свободные переменные, вместе со средой, которая привязывает эти переменные (т.е. “замыкает” это выражение).

Рассмотрим, что происходит внутри функции:

function greeting(name) {

// LexicalEnvironment = {name: 'Николай', text: undefined}

var text = 'Здравствуйте, ' + name;

// LexicalEnvironment = {name: 'Николай', text: 'Здравствуйте, Николай'}

alert(text);

}

greeting('Николай');

Когда функция вызывается, у нее создается объект LexicalEnvironment, в который записываются все локальные переменные и функции, а также ссылка на внешнюю область видимости. В нашем случае у нас есть локальная переменная name, у которой сразу есть значение (то, которое мы передаем) и это "Николай". В самом начале функции уже есть переменная text, интерпретатор знает про нее, но так как мы еще не дошли по присваивания этой переменной какого-то значения, то она равна undefined. Теперь мы присваиваем переменной значение, и наш объект LexicalEnvironment меняется. Его свойство text становится равным тому, что мы записали ("Здравствуйте, Николай" в нашем случае). После того, как функция отработала, объект LexicalEnvironment уничтожается. При последующих вызовах функции он будет создан снова и т.д.

Все переменные внутри функции – это свойства специального внутреннего объекта LexicalEnvironment, который создаётся при её запуске.

Таким образом, при запуске функция создает объект LexicalEnvironment, записывает туда аргументы, функции и переменные. Процесс инициализации выполняется в том же порядке, что и для глобального объекта, который является частным случаем лексического окружения.

В отличие от window, объект LexicalEnvironment является внутренним, он скрыт от прямого доступа.

Интерпретатор, при доступе к переменной, сначала пытается найти переменную в текущем LexicalEnvironment, а затем, если её нет – ищет во внешнем объекте переменных.

Такой порядок поиска возможен благодаря тому, что ссылка на внешний объект переменных хранится в специальном внутреннем свойстве функции, которое называется [[Scope]]. Это свойство закрыто от прямого доступа, но знание о нём очень важно для понимания того, как работает JavaScript.

При создании функция получает скрытое свойство [[Scope]], которое ссылается на лексическое окружение, в котором она была создана.

Это свойство никогда не меняется. Оно всюду следует за функцией, привязывая её, таким образом, к месту своего объявления.

При запуске функции её объект переменных LexicalEnvironment получает ссылку на «внешнее лексическое окружение» со значением из [[Scope]].

Если переменная не найдена в функции – она будет искаться снаружи.

Объект LexicalEnvironment уничтожается каждый раз после выполнения функции и создается снова при повторном вызове.

Замыкания применяются для того, чтобы можно было сохранить все, что записано в LexicalEnvironment, и использовать при следующих вызовах.

Рассмотрим более «продвинутый» вариант, при котором внутри одной функции создаётся другая и возвращается в качестве результата.

В разработке интерфейсов это совершенно стандартный приём, функция затем может назначаться как обработчик действий посетителя.

Здесь мы будем создавать функцию-счётчик, которая считает свои вызовы и возвращает их текущее число.

В примере ниже makeCounter создает такую функцию:

function makeCounter() {

var currentCount = 1;

return function() { // (\*\*)

return currentCount++;

};

}

var counter = makeCounter(); // (\*)

// каждый вызов увеличивает счётчик и возвращает результат

alert( counter() ); // 1

alert( counter() ); // 2

alert( counter() ); // 3

// создать другой счётчик, он будет независим от первого

var counter2 = makeCounter();

alert( counter2() ); // 1

Как видно, мы получили два независимых счётчика counter и counter2, каждый из которых незаметным снаружи образом сохраняет текущее количество вызовов во внешней переменной currentCount, которая у каждого счётчика своя.

Переменную во внешней области видимости можно не только читать, но и изменять.

**5. Порядок выполнения работы**

1. Реализуйте функцию, которая возвращает куб числа. Число передается параметром.

2. Реализуйте функцию, которая принимает параметром число от 1 до 7, а возвращает день недели на русском языке.

3. Выполните задание по варианту. Варианты представлены в таблице 14.2.

Таблица 14.2 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Заданиe** |
| 1 | Дан массив с числами. Создайте из него новый массив, где останутся лежать только положительные числа. Создайте для этого вспомогательную функцию, которая параметром будет принимать число и возвращать true, если число положительное, и false - если отрицательное. |
| 2 | Дан массив с числами. Реализуйте функцию, которая запишет в новый массив только те числа, которые больше нуля и меньше 10-ти. |
| 3 | Реализуйте функцию, которая параметром принимает целое число и возвращает сумму его цифр |
| 4 | Реализуйте функцию, которая параметром принимает целое число и проверяет: четное оно или нет. Если четное - пусть функция возвращает true, если нечетное – false |
| 5 | Реализуйте функцию, которая параметром принимает число и возвращает массив его делителей (чисел, на которое делится данное число). |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой функция в JavaScript?

2. Перечислите способы объявления функций.

3. Что представляет собой область видимости переменной?

4. Опишите виды переменных в JavaScript.

5. Для чего используется инструкция return?

6. В чем заключается механизм замыканий?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 15**

## **Тема работы: «Создание объектов и методов объектов. Работа со свойствами объектов»**

**1. Цель работы**

Формирование умений создания пользовательских объектов, их свойств и методов, использования стандартных объектов JavaScript.

**2. Задание**

Реализовать создание объектов с заданными свойствами и методами.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

В JavaScript почти все является объектами. Все примитивные типы за исключением null и undefined обрабатываются как объекты. Они могут быть назначены свойствам, и у них есть все характеристики, присущие объектам.

В JavaScript объект – это самостоятельная единица, имеющая свойства и определенный тип. Свойство объекта можно понимать как переменную, закрепленную за объектом.

Основные операции, производимые с объектами, – это добавление новых свойств, изменение уже существующих свойств, удаление свойств и обращение к свойствам.

Пустой объект может быть создан одним из двух синтаксисов: с помощью литерала объекта или оператора new с конструктором:

o = new Object();

o = {}; // пустые фигурные скобки

Например, объект Car:

var Car = new Object();

При использовании инициализатора объекта сразу задается как объект, так и значения. Для этого используется следующий синтаксис:

ИмяОбъекта = { Свойство1: Значение1, Свойство2: Значение2, ..., СвойствоN: ЗначениеN }

Следующие два фрагмента кода создают одинаковый объект:

var menuSetup = {

width: 300,

height: 200,

title: "Menu"

};

// то же самое, что:

var menuSetup = {};

menuSetup.width = 300;

menuSetup.height = 200;

menuSetup.title = 'Menu';

Следующий пример создает объект myHonda с тремя свойствами, свойство engine, в свою очередь, также является объектом со своими собственными свойствами.

var myHonda = {

color: "red",

wheels: 4,

engine: {

cylinders: 4,

size: 2.2

}

};

alert(myHonda. color) // " red "

alert(myHonda. engine. cylinders) // 4

Использование функции конструктора – другой способ создать объект в два шага:

1) определить объект, набрав функцию-конструктор. Название такой функции, как правило, начинается с заглавной буквы;

2) создать экземпляр объекта с помощью ключевого слова new.

Например, необходимо создать тип объект для cars. Объект этого типа должен называться car, и иметь свойства make, model, и year.

function Car(make, model, year) {

this.make = make;

this.model = model;

this.year = year;

}

Теперь можно создать объект, называемый mycar, следующим образом:

var mycar = new Car("Eagle", "Talon TSi", 1993);

В результате вызова new Car("Eagle", "Talon TSi", 1993) будет создан такой объект:

mycar = {

make:” Eagle”,

model: "Talon TSi",

year: 1993

}

Конструктором становится любая функция, вызванная через new.

Технически, любая функция может быть использована как конструктор. То есть, любую функцию можно вызвать при помощи new. Как-то особым образом указывать, что она – конструктор – не надо. Но, чтобы выделить функции, задуманные как конструкторы, их называют с большой буквы.

Теперь многократными вызовами new Car с разными параметрами можно создать столько объектов, сколько нужно. Поэтому такую функцию и называют конструктором – она предназначена для «конструирования» объектов.

Добавить свойство можно и после создания объекта:

mycar.color = "black";

Для доступа к свойству используется один из операторов доступа: . ([точка](http://www.puzzleweb.ru/javascript/6_operators5.php#a1)) или [] ([квадратные скобки](http://www.puzzleweb.ru/javascript/6_operators5.php#a2)):

var o = {};

o.x = 5;       // Добавили новое свойство

o["y"] = 10;   // Добавили новое свойство

Определение свойства происходит простым указанием его значения:

- точечная нотация:

ИмяОбъекта.ИмяСвойства

Пример:

Car.make = "Ford";

Car.model = "Mustang";

Car.year = 1969;

-квадратные скобки:

объект['свойство']

Пример:

var person = {};

person['name'] = 'Вася'; // то же что и

person.name = 'Вася';

Квадратные скобки также позволяют обратиться к свойству, имя которого хранится в переменной:

var person = {};

person.age = 25;

var key = 'age';

alert( person[key] ); // выведет 25

Если имя свойства хранится в переменной (var key = "age"), то единственный способ к нему обратиться – это квадратные скобки person[key].

Доступ через точку используется, если мы на этапе написания программы уже знаем название свойства. А если оно будет определено по ходу выполнения, например, введено посетителем и записано в переменную, то единственный выбор – квадратные скобки.

Обращение к свойству и изменение значения осуществляется точно так же (с помощью операторов доступа):

var o = {x:5};

alert(o.x);      // Обращение к свойству

alert(o["x"]);   // Обращение к свойству

o.x = 10;        // Изменяем значение

Удаление свойства осуществляется с помощью оператора [delete](http://www.puzzleweb.ru/javascript/7_delete.php):

var o = {x:5};

alert("x" in o);   // true

delete o.x;

alert("x" in o);   // false

Пример:

delete Car.year;

Остались только свойства make и model.

Метод объекта в javascript – это просто функция, которая добавлена в ассоциативный массив. Т.е. свойство, значением которого является функция, называется методом.

Чтобы создать метод, следует, прежде всего, написать функцию, которая будет его реализовывать, а затем указать его в конструкторе объекта.

function displayCar() {

var str = "<table border=\"1\">";

str += "<tr><td>Производитель:</td><td>"+this.make+"</td></tr>";

str += "<tr><td>Модель:</td><td>"+this.model+"</td></tr>";

str += "<tr><td>Год выпуска:</td><td>"+this.year+"</td></tr>";

str += "<tr><td>Владелец:</td><td>"+this.owner.name+", "+this.owner.age+"л.</td></tr>";

str += "</table>";

document.write (str);

}

К свойствам объекта функция обращается при помощи указателя this – такой синтаксис подразумевает, что данная функция будет методом объекта.

function Car(make, model, year) {

this.make = make;

this.model = model;

this.year = year;

this.displayCar = displayCar; }

Вызов метода осуществляется точно также, как и вызов обычной функции – с помощью оператора () ([оператор вызова](http://www.puzzleweb.ru/javascript/6_operators4.php#a3)):

var o = {

  sayHi: function() {

    alert("Hello!");

  }

};

o.sayHi();   // "Hello!"

Теперь, создав объект типа Car, можно просто воспользоваться его собственным методом для того, чтобы вывести все параметры в виде таблицы:

var MyCar = new Car("Ford", "Escort", 1997, new owner(44, Petrov));

MyCar.displayCar();

Добавление метода в объект осуществляется простым присвоением функции function(n) { ... }свойству:

var MyCar = new car("Ford", "Escort", 1997, new owner(44, Petrov));

MyCar.displayCar();

MyCar.probeg=function (x){

document.write("Пробег:"+x);

}

MyCar.probeg (234); // Пробег:234

Удаление методов

delete MyCar. displayCar;

Таким образом, в любой объект в любое время можно добавить новый метод или удалить существующий.

Кроме того, что методы можно вызывать из объекта, методы имеют доступ к самому объекту и могут менять находящиеся в нем данные с помощью ключевого слова this. Значение this называется контекстом вызова и будет определено в момент вызова функции.

Простой вызов функции (без контекста):

function f() { console.log(this === window); // true

}

f();

В данном случае this внутри функции f равен глобальному объекту (например, в браузере это window).

Если функция вызывается через new как конструктор объекта, то this ставится на создаваемый объект,как показано на рисунке 15.1:

Function Animal(name) {

    this.name = name;

}



Рисунок 15.1 – Контекст this при создании экземпляраобъекта

Если функция запущена как свойство объекта, то в this будет ссылка на этот объект. this обозначает объект, вызвавший функцию, при этом совершенно неважно, откуда эта функция взялась. Важно лишь, какой перед функцией стоит объект.

Создадим два любых объекта:

vasya = {

    name: "Василий"

}

dima = {

    name: "Дмитрий"

}

Определим никак не связанную с ними функцию say:

say = function() {

    alert("Привет, я "+this.name)

}

Присвоим функцию свойству sayHi для обоих объектов:

vasya.sayHi = say;

dima.sayHi = say;

vasya.sayHi()    // =>  "Привет, я Василий"

dima.sayHi()    // => "Привет, я Дмитрий"

say();

При последнем вызове this ставится равным глобальному объекту.

Методы apply и call позволяют задать контекст для выполняемой функции. Разница между apply и call – только в способе передачи параметров в функцию. Первый параметр обеих функций определяет контекст выполнения функции (то, чему будет равен this).

Например, у нас есть функция showFullName, которая работает с this:

function showFullName() {

alert( this.firstName + " " + this.lastName );

}

var user = {

firstName: "Василий",

lastName: "Петров"

};

showFullName.call(user)

Метод apply() вызывает функцию с указанным значением this и аргументами, предоставленными в виде массива (либо [массивоподобного объекта](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Predefined_Core_Objects#Working_with_Array-like_objects)).

В частности, эти две строчки сработают одинаково:

showFullName.call(user, 'firstName', 'surname');

showFullName.apply(user, ['firstName', 'surname']);

Преимущество apply перед call отчётливо видно, когда массив аргументов формируется динамически.

При помощи apply можно найти максимум в произвольном массиве, вот так:

var arr = [];

arr.push(1);

arr.push(5);

arr.push(2);

// получить максимум из элементов arr

alert( Math.max.apply(null, arr) ); // 5

**5. Порядок выполнения работы**

1. Создать объект с 5 свойствами (на выбор). Добавить метод объекта, который выводит приветственное сообщение. Вывести все свойства созданного объекта двумя способами.

2. Выполнить задание по варианту. Варианты представлены в таблице 15.1.

Таблица 15.1 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| **1** | **2** |
| 1 | 1. Создайте объект Сотрудник, который будет хранить сведения о сотруднике некоторой фирмы.такие как Имя, Отдел, Телефон, Зарплата. Создайте метод объекта для отображения данных. |

|  |  |
| --- | --- |
| **1** | **2** |
|  | 1. Создайте объект для работы туристической фирмы с методом подсчета стоимости поездки из расчета: количества человек \* количество дней \* стоимость путевки. Вывести все свойства объекта на экран. |
| 2 | 1. Добавьте к встроенному классу String метод showMess(), который выводит слово «Метод объекта String!» как заголовок (тег h…), указанного пользователем уровня. Уровень заголовка можно добавить в виде свойства класса String. 2. Создайте объект Сотрудник, который будет хранить сведения о сотруднике некоторой фирмы.такие как Имя, Отдел, Телефон, Зарплата. При создании используйте функцию-конструктор и ключевое слово this. |
| 3 | 1. Создайте конструктор function Card(from, to){...}, создающий объекты карточки со свойствами from, to и методом show(), возвращающим свойства отдельного объекта в виде строки “from,to”, где на месте названий свойств будут их значения. Создайте переменную city с городами Минск и Брест 2. Создайте методы export() у каждого из объектов. Этот метод должен возвращать информацию о городе в формате “name=ГородName\npopulation=10000000\n”. Для второго города будет строка со своими значениями |
| 4 | 1. Создать объект obj, с методами method1(),method2() и method3(). В методе method3() должна возвращаться строка “метод3”. Сделайте так, чтобы было возможно выполнение кода obj.method1().method2().method3(). 2. Создайте объект Сотрудник, который будет хранить сведения о сотруднике некоторой фирмы.такие как Имя, Отдел, Телефон, Зарплата. При создании используйте функцию-конструктор и ключевое слово this. |
| 5 | 1. Создайте объект для работы туристической фирмы с методом подсчета стоимости поездки из расчета: количества человек \* количество дней \* стоимость путевки. Вывести все свойства объекта на экран. 2. Добавьте к встроенному классу String метод showMess(), который выводит слово «Метод объекта String!» как заголовок (тег h…), указанного пользователем уровня. Уровень заголовка можно добавить в виде свойства класса String. |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой объект в JavaScript?
2. Перечислите основные операции, производимые с объектами.
3. Опишите способы создания пользовательских объектов в JavaScript.
4. Каким образом осуществляется доступ к свойствам объектов?
5. Что такое метод объекта?
6. Каким образом определяется контекст вызова функции?
7. Для чего используются методы cal и apply?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 16**

## **Тема работы: «Работа с элементами объекта браузера средствами JavaScript. Обработка событий браузера»**

**1. Цель работы**

Формирование умений работы с элементами BOM и обработки событий браузера.

**2. Задание**

Реализовать управление элементами BOM и обработку событий браузера средствами JavaScpipt.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Объектная модель браузера (анг. Browser Object Model (BOM)) – это часть JavaScript, которая позволяет скрипту взаимодействовать с программой просмотра веб-страниц. BOM представляет объекты, через свойства и методы которых можно управлять внешним видом и поведением обозревателя.

Окно браузера представлено на рисунке 16.1:

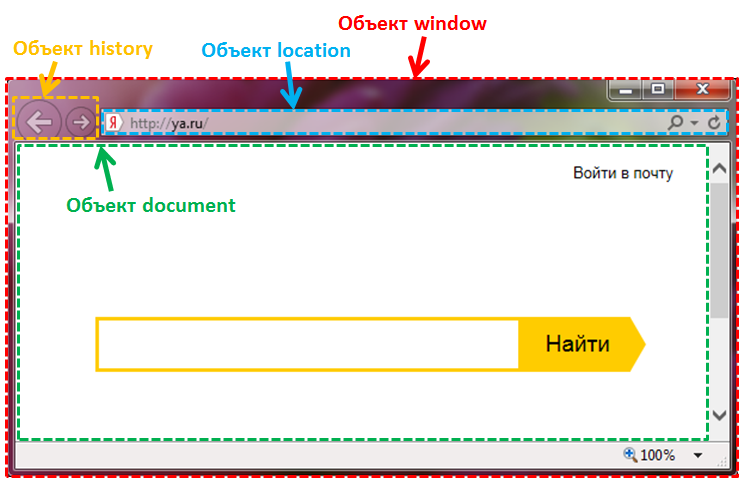


Рисунок 16.1 – Окно браузера

BOM-интерфейс позволяет работать с окном браузера как с объектом, у которого имеются свои свойства и методы. Данным объектом является объект window (он же является [глобальным объектом в JavaScript](http://webgent.ru/javascript/standard-objects/global-object)). Объект window представляет собой одно из окон или вкладку браузера с его панелями инструментов, меню, строкой состояния, HTML страницей и другими объектами.

Некоторые браузеры открывают страницы не в отдельных окнах, а во вкладках одного окна. В этом случае каждая вкладка также имеет свой собственный объект window. Однако, при обращении к свойствам, которые не могут относиться к вкладкам (например, изменяющие размер окна браузера), автоматически используется общий объект window окна браузера. Например, если внутри отдельной вкладки происходит вызов метода window.resizeTo(), тогда window ссылается на общий объект окна браузера.

Основные объекты BOM и связи между ними приведены на рисунке 16.2.

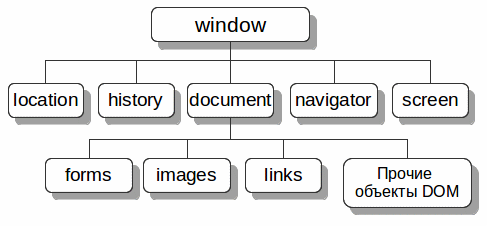
**

Рисунок 16.2 – Объекты браузера

Window – самый главный объект в браузере, который отвечает за одно из окон (вкладок) браузера. Он является корнем иерархии всех объектов доступных веб-разработчику в сценариях JavaScript.

Объект window имеет много разнообразных свойств, но используются из них лишь некоторые. Весь список свойств можно получить с помощью простого кода:

<html>

<head>

<title>BOM-интерфейс</title>

</head>

<body>

</body>

</html>

<script>

var all = '';

for (var prop in window) {

all += window.prop + '\n';

}

alert(all);

</script>

Объект window кроме глобальных объектов (document, screen, location, navigator и др.) имеет собственные свойства и методы.

К объекту Window нельзя обратиться напрямую, однако он имеет свойство window, ссылающееся на сам объект, которое можно использовать вместо ключевого слова this для ссылки на глобальный объект.

Чтобы получить объект window, достаточно написать в html документе:

<script type="text/javascript">

window //объект window

</script>

Пример создания нового окна:

var win = window.open("http://mrk-bsuir.by", "My Window);

Будет создано окно, в которое будет подгружен сайт: "http://mrk-bsuir.by", а имя нового окна будет: "My Window".

У метода open() существует ещё один необязательный параметр, с настройками нового окна:

var win = window.open("http://mrk-bsuir.by", "My Window", "width=400,height=500,menubar=yes,toolbar=no,location=yes,scrollbars=yes");

Этот параметр имеет очень жёсткие условия в плане синтаксиса: никаких пробелов быть вообще не должно:

- width - отвечает за ширину нового окна;

- height - отвечает за высоту нового окна;

- menubar - если стоит "yes", то в новом окне будет меню, если стоит "no", то, соответственно, меню не будет;

- toolbar - отвечает за наличие панели инструментов;

- location- отвечает за наличие адресной строки в новом окне;

- scrollbars - отвечает за наличие полос прокрутки.

Методы focus() и blur() позволяют навести фокус на окно (focus()) и, наоборот, потерять его (blur()).

Пример:

var win = window.open("http://mrk-bsuir.by", "My Window);

win.focus();

//Некая задержка

win.blur();

Метод close() позволяет закрыть окно:

win.close();

Данный метод не всегда сработает для глобального окна. А если и сработает, то выдаст предупреждение пользователю о закрытии окна, в котором он может либо подтвердить, либо отклонить закрытие.

Методы back() и forward() заменяют кнопки "Назад" и "Вперёд" в браузере. Иногда это используют, чтобы сайт был более удобен для пользователя (например, сделать кнопки на сайте для возврата на предыдущую страницу).

Пример применения этих методов:

window.back();

так тоже будет работать:

back();

В JavaScript имеется всего два метода для работы с таймерами. Первый метод – это setTimeout(). Эта функция принимает два параметра: функцию (либо код), которую нужно выполнить, и второй параметр – задержку (в миллисекундах), через которую надо выполнить функцию (либо код).

Пример:

setTimeout(func, 3000);

function func () {

alert("Прошло 3 секунды");

}

Данный скрипт через 3 секунды после начала выполнения выведет информационное сообщение.

Т.к. метод setTimeout() применён к глобальному окну, поэтому мы не писали так: "window.setTimeout()".

Если требуется остановить таймер, то используется метод clearTimeout(), который принимает в качестве параметра идентификатор, полученный при использовании метода setTimeout():

var id = setTimeout("3 + 4", 5000);

clearTimeout(id);

То есть сразу после запуска таймера, мы его останавливаем, соответственно, выражение: "3 + 4" посчитано не будет.

Ключевая особенность метода setTimeout() в том, что он выполняется всего один раз. В нашем примере, он вызвал функцию func() и всё. Больше эта функция вызываться через таймер не будет. Для того, чтобы настроить регулярное срабатывание таймера через равные промежутки времени используется метод setInterval():

var counter = 0;

var id = setInterval(func, 5000, "Tommy");

function func(name) {

counter++;

alert(name + ", уже прошло " + (counter \* 5) + " секунд!");

if (counter == 4) clearInterval(id);

}

Глобальная переменная counter показывает, сколько раз была вызвана функция. От неё считается и прошедшее время после начала выполнения скрипта. Также заметьте, что функция func() требует параметр. Этот параметр передаётся в методе setInterval() сразу после параметра с задержкой. Если будет два параметра в функции, то в методе setInterval() также надо будет дописать ещё два передающихся параметра. Такие же правила и для метода setTimeout(). После 4-х вызовов функции мы останавливаем непрерывное выполнение таймера. В таблице 16.1 приведены методы объекта Window.

Таблица 16.1 – Методы объекта Window

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Описание** | | |
| **1** | **2** | | |
| методы ввода-вывода информации | | | |
| alert() | Выводит модальное диалоговое окно с сообщением и кнопкой OK. | | |
| confirm() | Отображает модальное диалоговое окно, содержащее сообщение и кнопки ОК и Отмена. Возвращает соответственно true или false. | | |
| prompt() | Отображает диалоговое окно с сообщением и полем ввода для пользователя. Возвращает строку, содержащую введенные данные. | | |
|  | function alert(){  alert("hello");} | var a=confirm("This is a cat?");  alert(a); | var a=prompt("подсказка", "default");  alert(a); |
| открытие — закрытие — печать окон | | | |
| open() | Создает и открывает новое окно. Его параметры: адрес страницы, которую надо открыть; внутренне имя окна, которое хотите ему присвоить; набор свойств открываемого окна (location, width, height, menubar, toolbar, scrollbars). | | |

|  |  |  |
| --- | --- | --- |
| **1** | **2** | |
| close() | Закрывает окно, которое было открыто с помощью метода window.open(). При закрытии выскакивает предупреждение браузера из соображений безопасности. Без параметров. | |
| print() | Печатает содержимое текущего окна. Без параметров. | |
| focus() | Устанавливает фокус на текущее окно. Без параметров. | |
| blur () | Убирает фокус у окна, уводит окна браузера на задний план, под все открытые окна. Без параметров. | |
|  | open("http://phpmove.ru"); //можно с одним параметром  open("http://phpmove.ru","site","width=150"); //можно с 3 параметрами | close(); |
| методы для управления открытым окном — положение и размеры окна | | |
| moveBy() | Перемещает текущее окно на заданную величину относительно левого верхнего угла. | |
| moveTo() | Перемещает окно в соответствии с указанными координатами. | |
|  | moveTo(10,20);  mobeBy(10,20); //смещение относительно текущего левого верхнего угла | |
| resizeBy() | Изменяет текущее окно на указанное число пикселей по горизонтали и вертикали. | |
| resizeTo() | Изменяет размер окна до указанной ширины и высоты. | |
|  | resizeTo(500,500); //задает размер окна  resizeBy(10,20); //увеличивает размер окна | |
| scrollBy() | Прокрутка документа в окне на указанное количество пикселей относительно текущего положения | |
| scrollTo() | Прокрутка документа до указанных координат: 1-й параметр — прокрутка слева направо, 2 — сверху-вниз | |
|  | scrollTo(10,20); //просто прокрутка  scrollBy(11,12); //относительно текущего положения | |
| atob() | Декодирует строку данных, которая была закодирована с использованием кодировки base-64. | |
| btoa() | Создает строку ASCII в кодировке base-64 из строки бинарных данных. | |
| clearInterval() | Отменяет выполнение действий, заданных с помощью метода setInterval(). | |
|  | // через 5 сек остановить повторы  setTimeout(function() {  clearInterval(timerId);  alert( 'стоп' );  }, 5000); | |
| clearTimeout() | Отменяет выполнение действий, заданных с помощью метода setTimeout(). | |
|  | var timerId = setTimeout(...);  clearTimeout(timerId); | |
| setInterval() | Вызывает функцию или вычисляет выражение в определенные промежутки времени (в миллисекундах). | |

|  |  |  |
| --- | --- | --- |
| **1** | **2** | |
|  | // начать повторы с интервалом 2 сек  var timerId = setInterval(function() {  alert( "тик" );  }, 2000); | |
| setTimeout() | Вызывает функцию или вычисляет выражение после указанного числа миллисекунд. | |
|  | setTimeout("alert('Привет')", 1000);  // 1000 миллисекунд равны 1 секунде  // выведет сообщение через 1 с. | function func(phrase, who) {  alert( phrase + ', ' + who );  }  setTimeout(func, 1000, "Привет", "Вася"); // Привет, Вася |
| stop() | Останавливает загрузку окна. | |

В таблице 16.2 приведены своства объекта Window.

Таблица 16.2 – Свойства объекта Window

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| **1** | **2** |
| closed | Возвращает логическое значение, указывающее закрыто окно или открыто. |
| document | Возвращает объект Document текущего окна. |
| frames | Возвращает массив со всеми элементами <iframe> текущего окна. |
| history | Возвращает ссылку на объект History. |
| innerHeight | Возвращает высоту области просмотра окна. |
| innerWidth | Возвращает ширину области просмотра окна. |
| lenght | Возвращает количество <iframe> элементов в окне. |
| location | Возвращает ссылку на объект Location. |
| name | Задает или получает значение, указывающее имя окна. |
| navigator | Возвращает ссылку на объект Navigator. |
| opener | Задает или получает ссылку на окно, которое было создано в текущем окне. |
| outerHeight | Возвращает внешнюю высоту окна, включая панели инструментов и полосы прокрутки. |
| outerWidth | Возвращает внешнюю ширину окна, включая панели инструментов и полосы прокрутки. |
| pageXOffset | Возвращает количество пикселей, на которое текущий документ был прокручен (по горизонтали) от верхнего левого угла окна. |
| pageYOffset | Возвращает количество пикселей, на которое текущий документ был прокручен (по вертикали) от верхнего левого угла окна. |
| parent | Возвращает родительское окно текущего окна. |
| screen | Возвращает ссылку на объект Screen, связанный с окном. |
| screenLeft | Получает x-координату верхнего левого угла окна относительно верхнего левого угла |

|  |  |
| --- | --- |
| **1** | **2** |
| screenTop | Получает y-координату верхнего угла окна, по отношению к верхней части экрана. |
| scrollX | Эквивалент свойства pageXOffset. |
| scrollY | Эквивалент свойства pageYOffset. |
| self | Извлекает ссылку на текущее окно или фрейм. |
| Status | Получает/устанавливает текст в строке состояния в нижней части браузера. |

Доступ к различным объектам Window осуществляется с помощью следующих основных объектов:

- navigator;

- history;

- location;

- screen;

- document.

Так как данные объекты являются дочерними по отношению к объекту window, то обращение к ним происходит как к свойствам объекта window.

Например, для того чтобы обратиться к объекту screen, необходимо использовать следующую конструкцию:

window.screen

Но, если мы работаем с текущим окном, то "window." можно опустить. Например, вместо window.screen можно использовать просто:

screen

Объект navigator – это чисто информационный объект, он дает информацию о самом браузере (название, версия, установленные компоненты). Получить объект navigator можно как одноименное свойство window:

<script type="text/javascript">

window.navigator

</script>

Используя его свойства можно узнать какой браузер (версия) и операционную систему имеет пользователь. В таблице 16.3 представлены свойства объекта navigator.

Таблица 16.3 – Свойства объекта navigator

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| **1** | **2** |
| appName | Возвращает имя браузера. Свойство доступно только для чтения.  document.write("Вы используете браузер " + navigator.appName); |

|  |  |
| --- | --- |
| **1** | **2** |
| appVersion | Возвращает название браузера, его версию и платформу (ОС) на которой он установлен. Свойство доступно только для чтения.  document.write("Версия Вашего браузера " + navigator.appVersion); |
| cookieEnabled | Показывает доступно или не доступно использование cookie файлов в браузере. |
| geolocation | Получает ссылку на объект Geolocation, который сообщает географическое расположение устройства, на котором запущено приложение. |
| onLine | Возвращает значение булева типа, значение true для работы браузера в онлайн режиме, значение false для автономного режима работы. |
| platform | Возвращает тип платформы (ОС) под которую был скомпилирован данный браузер. |
| systemLanguage | Показывает, какой язык по умолчанию используется операционной системой. |
| userAgent | Возвращает дополнительную информацию о браузере, так же определяет, какой движок использует ваш браузер. |

Пример:

var browser = navigator.appName;

if (browser == "NetScape")

document.write("Тут можно выполнять скрипты для обладателей браузеров NetScape");

Объект History предоставляет журнал навигации за всё время работы с конкретным окном. Так как он является свойством объекта Window, то у каждой вкладки в браузере, также как и у каждого [элемента <iframe>](https://puzzleweb.ru/html/tag_iframe.php) будет свой объект Window и свой объект History.

Из соображений безопасности браузеры не позволяют определять [URL-адреса](https://puzzleweb.ru/html/3_directory_path.php#a3) страниц, которые посещал пользователь, но можно перемещаться по их списку вперёд и назад, не зная URL-адреса.

У объекта History есть только одно свойство – length. Оно возвращает целое число, указывающее количество страниц в истории сеанса работы с текущим окном браузера, включая текущую загруженную страницу. Это свойство можно использовать следующим образом:

alert("Количество страниц, просмотренных в текущем окне, равно " + history.length);

Данный код отправляет пользователю сообщение с информацией о количестве посещенных страниц в текущем окне браузера. Например, для загруженной в новой вкладке страницы это свойство вернет значение 1.

Метод back() загружает предыдущую страницу из истории сеанса (выполняет тоже действие, что и кнопка назад в браузере). Эквивалентен history.go(-1).

<!DOCTYPE html>

<html>

<body>

<button onclick="goBack()">Вернуться назад</button>

<script>

function goBack() {

history.back();

}

</script>

</body>

</html>

Метод forward() загружает следующую страницу в истории сеанса (выполняет тоже действие, что и кнопка вперед в браузере). Эквивалентен history.go(1).

history.forward();

Метод go() загружает страницу из истории посещения текущего окна или вкладки браузера. Этот метод принимает один аргумент: строку или целое число. Целое положительное число указывает на какое количество страниц вперед нужно перейти, отрицательное число указывает на сколько страниц назад нужно вернуться. С помощью строкового аргумента можно указать точный URL-адрес из истории посещения, на который нужно перейти.

// переход назад на одну страницу

history.go(-1);

// переход вперед на две страницы

history.go(.2);

Кроме этого, с появлением HTML5 History API веб-разработчику стали доступны методы для добавления и изменения записей в истории, а также событие, с помощью которого можно обрабатывать нажатие кнопок forward (вперёд) и back (назад).

Объект Location содержит информацию о URL-адресе текущего документа, с которым он связан.

Чтобы изменить или получить текущий URL, не обязательно использовать методы или свойства объекта, это можно также сделать обратившись напрямую к свойству location:

var someURL = window.location;

window.location = 'httр://www.puzzleweb.ru';

В таблице 16.4 представлены методы объекта Location.

Таблица 16.4 – Методы объекта Location

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| assign() | Загружает новый документ в текущее окно (вкладку) браузера. |
| reload() | Перезагружает документ с текущем URL-адресом. Может принимать один аргумент булева типа: true - перезагрузка страницы с сервера, false - перезагрузка страницы из кэша браузера. |
| replace() | Заменяет текущий документ на документ, расположенный по адресу, переданному методу в качестве аргумента. Отличие от метода assign() заключается в том, что после использования replace() текущая страница удаляется из истории посещения текущего окна, это значит что пользователь не сможет вернуться на нее нажав кнопку назад в браузере. |

В таблице 16.5 представлены свойства объекта Location.

Таблица 16.5 – Свойства объекта Location

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| hash | Задает или получает часть URL-адреса, следующую за знаком #. |
| host | Задает или возвращает имя хоста и порт URL-адреса. |
| hostname | Задает или возвращает имя хоста из URL-адреса. |
| href | Задает или возвращает полный URL-адрес документа. |
| origin | Возвращает строку, содержащую URL-адрес, включающий протокол, домен и порт. |
| pathname | Строка, содержащая имя файла или путь к текущему файлу от корневой папки. |
| port | Содержит строку с портом. |
| protocol | Задает или получает протокол. |
| search | Задает или возвращает часть URL-адреса, следующую за знаком вопроса (?), включая сам знак вопроса. |

Объект Screen (который также является свойством объекта Window) – один из немногих JavaScript объектов, которые практически не используются в коде. Он просто предоставляет сведения о графических параметрах клиентской системы вне окна браузера, такие как ширина и высота в пикселях. Доступность тех или иных свойств объекта Screen зависит от браузера.

В таблице 16.6 представлены свойства объекта Screen.

Таблица 16.6 – Свойства объекта Screen

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| **1** | **2** |
| availHeight | Возвращает высоту рабочей области экрана системы в пикселях, за исключением элементов пользовательского интерфейса операционной системы (например панели задач в Windows). |
| availWidth | Возвращает ширину рабочей области экрана системы в пикселях, за исключением элементов пользовательского интерфейса операционной системы. |
| colorDepth | Возвращает целое число: 1, 4, 8, 15, 16, 24 или 32, указывающее количество бит на пиксель для цветов. |
| height | Возвращает общую высоту экрана в пикселях. |
| pixelDepth | Возвращает целое число: 1, 4, 8, 15, 16, 24 или 32, указывающее количество бит на пиксель для цветов. |
| width | Возвращает общую ширину экрана в пикселях. |

На мобильных устройствах размеры экрана определяются немного иначе. Устройства с системой iOS всегда возвращают размеры экрана в книжной ориентации (1024 x 768). Что касается устройств на Android, они настраивают значения screen.width и screen.height правильно.

Пример:

var foo = 'String';

console.log( navigator.userAgent ); //содержит информацию о браузере.

console.log( navigator.platform ); //содержит информацию о ОС

console.log( screen.width + ' x ' +screen.height ); //разрешение вашего экрана

console.log( location.toString() ); //ваш URL в виде строки

console.log( history ); //история браузера

//window.confirm('BOM');

var bar = window.prompt();

if (bar === 'Привет') {

alert('Привет и тебе от BOM!')

} else {

alert('Пока!');

}

**5. Порядок выполнения работы**

1. Создайте 2 элемента на странице (кнопки, текста или ссылки), при нажатии на которые будут создаваться 2 разных окна со следующими свойства-ми:

1-ое: ширина 200, высота 200, не отображение панели инструментов, не отображение панели меню, отображение строки состояния, не отображение адресной строки, не отображение кнопок браузера, отображение полос прокрут-ки;

2-ое: ширина 400, высота 400, отображение панели инструментов, отображение панели меню, отображение строки состояния, не отображение адресной строки, отображение кнопок браузера, не отображение полос прокрутки

2. Выполните задание по варианту. Варианты представлены в таблице 16.7.

Таблица 16.7– Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| 1 | 1. На HTML-страницу добавьте кнопку. По нажатию на данную кнопку необходимо вывести высоту окна браузера. 2. Выведите на страницу всю информацию о браузере |
| 2 | 1. На HTML-страницу добавьте кнопку. По нажатию на данную кнопку необходимо изменить высоту окна браузера. 2. Выведите на страницу версию и название браузера |
| 3 | 1. На HTML-страницу добавьте кнопку. По нажатию на данную кнопку необходимо вывести ширину окна браузера. 2. Выведите на страницу версию браузера |
| 4 | 1. На HTML-страницу добавьте кнопку. По нажатию на данную кнопку необходимо изменить ширину окна браузера. 2. Выведите на страницу название браузера |
| 5 | 1. На HTML-страницу добавьте кнопку. По нажатию на данную кнопку необходимо узнать, есть ли у окна браузера вертикальная прокрутка 2. На HTML-страницу добавьте кнопку. По нажатию на данную кнопку необходимо изменить высоту окна браузера |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой объектная модель браузера (BOM)?

2.Перечислите основные объекты BOM.

3.Опишите свойства объекта Window.

4.Укажите параметры окна, которое открывается при выполнении следу-ющего кода:

msgWindow=window.open("1.html", "displayWindow", "width=300, height=300,location=no,status=yes,toolbar=yes,menubar=no,scrollbars=no,resizable=no")

Как можно закрыть данное окно?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 17**

## **Тема работы: «Использование объектов HTML и объектной модели документа. Работа с элементами DOM в JavaScript»**

**1. Цель работы**

Формирование умений работы с элементами DOM, применения методов доступа к элементам документа.

**2. Задание**

Реализовать манипулирование узлами DOM-дерева средствами JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Объект document – самый сложный объект браузера, который в свою очередь дает начало объектной модели документа (DOM). Данный объект отвечает за html документ, загруженный в окно браузера. Его получают аналогично:

<script type="text/javascript">

window.document

</script>

С помощью свойств и методов данного объекта можно получить доступ к содержимому HTML-документа, а также изменить его содержимое, структуру и оформление.

Document как и другие объекты, является частью объекта window и, следовательно, он может быть доступен как window.document.

Каждая веб-страница, загружаемая в браузере, имеет свой собственный объект Document. Веб-разработчик работает не с HTML кодом html-страницы, а с объектами (DOM), которые браузер создаёт на основе этого кода. Взаимодействие веб-разработчика с html-страницей показано на рисунке 17.1.

**Веб-страница**

**Сценарий на JavaScript**

DOM

HTML

Рисунок 17.1 – Схема взаимодействия веб-разработчика с html-страницей

Исходный код веб-страницы (HTML документа) состоит из HTML тегов и текста. Теги – это базовая синтаксическая конструкция кода HTML. Большинство тегов являются парными, т.е. один из них открывающий, а другой закрывающий. Одна такая пара тегов образует элемент HTML. В документе HTML элементы вкладываются друг в друга. В итоге веб-страница (HTML документ) представляет собой множество вложенных друг в друга элементов HTML.

Согласно DOM-модели, документ является иерархией: всё содержимое страницы (элементы и текст) представляется как иерархия узлов. Каждый HTML-тег образует отдельный элемент-узел, каждый фрагмент текста – текстовый элемент, и т.п. Таким образом, DOM – это представление документа в виде дерева тегов. Это дерево образуется за счет вложенной структуры тегов плюс текстовые фрагменты страницы, каждый из которых образует отдельный узел. Это обеспечивает унифицированный способ навигации по документу, имеющего одну из наиболее употребительных структур в программировании.

Браузер, получая код HTML из интернета, сначала строит дерево узлов (объектов) в своей памяти. А уже потом на основе этого дерева "рисует" картинку этой страницы в своём окне или вкладке, как показано на рисунке 17.2.

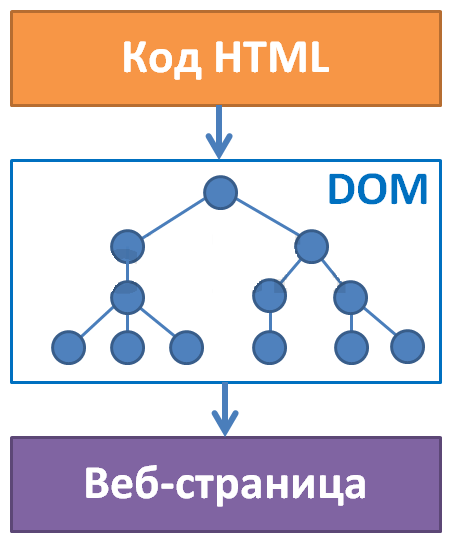


Рисунок 17.2 – Схема создания веб-страницы

Рассмотрим следующий код:

<html>

<head>

<title>Заголовок</title>

</head>

<body>

Прекрасный документ

</body>

</html>

Дерево DOM для него будет иметь следующий вид, как показано на рисунке 17.3:

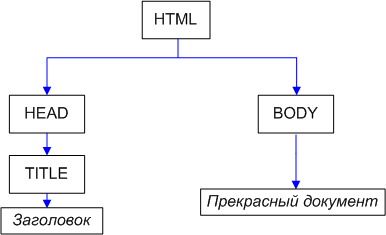


Рисунок 17.3 – DOM дерево

Дерево DOM – это множество узлов (объектов, элементов) связанных друг с другом. При этом каждый элемент HTML образует узел в этом дереве.

Теги образуют узлы-элементы (element node). Текст представлен текстовыми узлами (text node). И то и другое – равноправные узлы дерева DOM.

Пример д[ругих типов узлов](https://learn.javascript.ru/dom-nodes#другие-типы-узлов) показан на рисунке 17.4:

<!DOCTYPE HTML>

<html>

<body>

Правда о лосях

<ol>

<li>Лось — животное хитрое</li>

<!-- комментарий -->

<li>...и коварное!</li>

</ol>

</body>

</html>

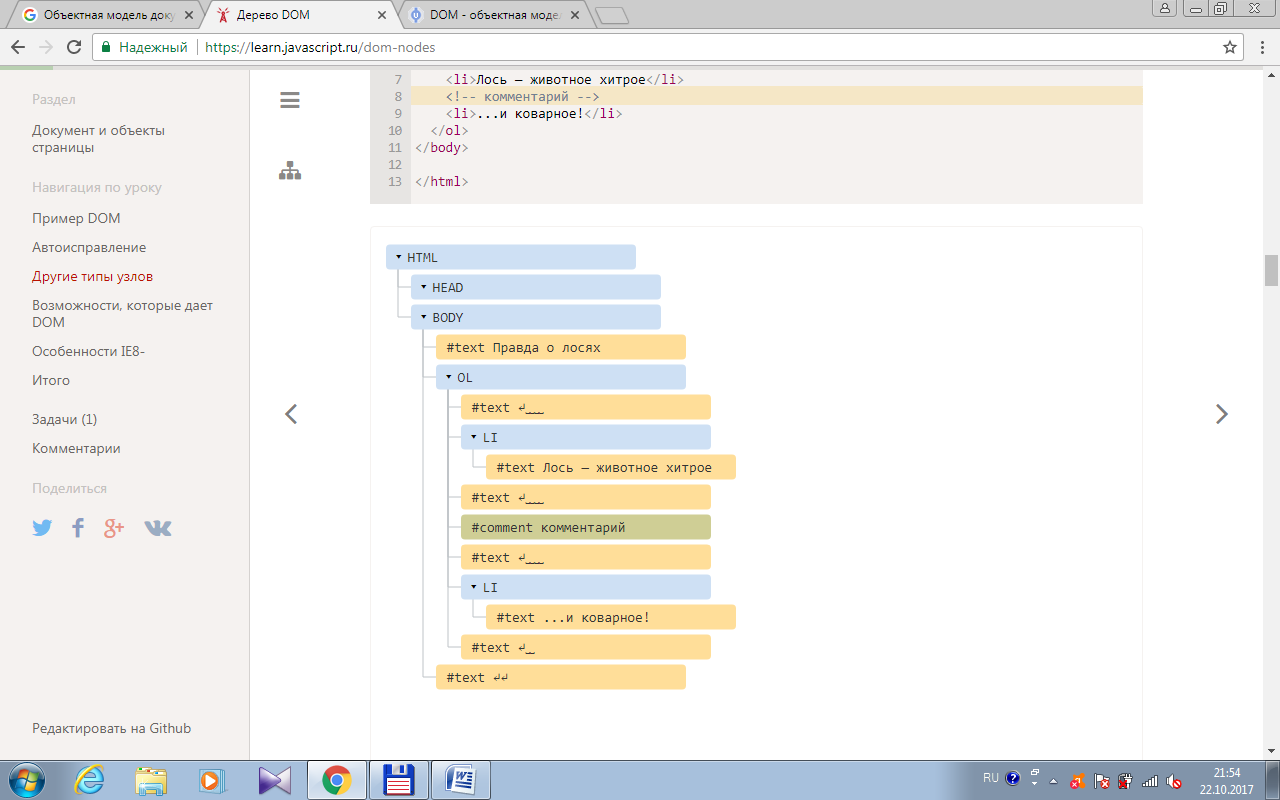
**

Рисунок 17.4 – DOM дерево с различными типами узлов

К элементу страницы нельзя обратиться раньше, чем он отобразится в браузере.

DOM-структура документа создаётся по мере вывода страницы на экран. Если элемент ещё не отображён в браузере, значит его нету в дереве DOM. Соответственно, к такому элементу нельзя обратиться.

При выполнении следующего скрипта ничего не произойдёт:

<html>

<head>

<title>DOM интерфейс</title>

<script>

document.body.style.background = '#599';

</script>

</head>

<body>

</body>

</html>

Браузер, построив на основе кода HTML дерево узлов, всегда может осуществить обратную операцию: на основе этого дерева создать код HTML.

Основная работа с HTML страницей в JavaScript состоит в том, чтобы работать с этими узлами. Как только происходит изменение свойства этих узлов в дереве, добавляются новые или удаляются существующие, то автоматически происходят изменения на экране пользователя и автоматически изменяется код HTML. Т.е. происходят обратные действия, на основе дерева строится код HTML.

Рассмотрим пример:

<div class="container">

<h1 id="h1">Hello, world!</h1>

<p class="yellow" id="yellow">Lorem ipsum dolor sit amet.</p>

<p class="grey" id="grey">Lorem ipsum dolor sit amet.</p>

</div>

Обратимся к заголовку и выберем его для дальнейшей работы. Заголовок H1 имеет одноименный идентификатор. JavaScript предлагает метод getElementById(), который позволяет взять из объекта один элемент с указанным идентификатором. Если использовать определенный id, тогда он должен быть уникален в пределах одного документа, т.е. нельзя размещать в одном документе два элемента с одинаковым атрибутом id.

var h1 = document.getElementById('h1');

console.log(h1);

В результате произошло сохранение в переменной h1 объекта, которым является конкретный элемент страницы. Теперь можно делать с данным объектом что угодно: изменить шрифт, цвет, фон и т.п. Для того, чтобы изменить фон необходимо:

var h1 = document.getElementById('h1');

h1.style.background = '#ccc';

Собственно, если элемент не нужен ни для каких других операций, тогда можно не сохранять его в переменную, а сразу изменить стиль элемента:

document.getElementById('h1').style.background = '#ccc';

Точно таким же образом можно обратиться к любому элементу документа, которому присвоен идентификатор.

Метод getElementsByClassName() позволяет получить коллекцию элементов с указанным классом.

Создадим пару абзацев с одинаковым классом и посмотрим, как эти выбранные элементы отобразятся в консоли, рисунок 17.5:

var yellow = document.getElementsByClassName('yellow');

console.log(yellow);

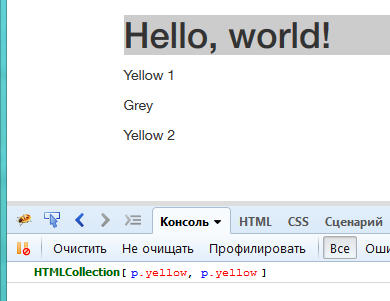


Рисунок 17.5 – Результат работы программы

Для того, чтобы окрасить в соответствующие цвета параграфы на странице, рисунок 17.6:

var yellow = document.getElementsByClassName('yellow');

yellow[0].style.background = 'yellow';

yellow[1].style.background = 'yellow';

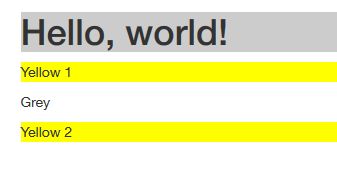


Рисунок 17.6 – Результат работы программы

Если элементов на странице много, тогда того же результата можно достичь, используя цикл:

for(var k = 0; k < yellow.length; k++){

yellow[k].style.background = 'yellow';

}

Метод getElementsByTagName позволяет получить коллекцию элементов по их имени, т.е. по имени тега (div, p, input и т.п.).

Вывод всех параграфов:

<div class="container" id="container1">

<h1 id="h1">Hello, world!</h1>

<p class="yellow">Yellow 1</p>

<p class="grey">Grey</p>

<p class="yellow">Yellow 2</p>

</div>

<div class="container"  id="container2">

<p class="green">Grey</p>

</div>

var p = document.getElementsByTagName('p');

console.log(p);

В результате все 4 параграфа попали в коллецию, как показано на рисунке 17.7:

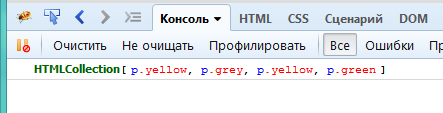


Рисунок 17.7 – Результат работы программы

Если нужны не все параграфы на странице, а только параграфы определенного блока, метод getElementsByTagName может искать элементы не только во всем документе, но и в локальном его участке:

var p = document.getElementById('container1').getElementsByTagName('p');

console.log(p);

В итоге получили только нужные элементы страницы, как показано на рисунке 17.8.

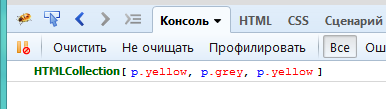


Рисунок 17.8 – Результат работы программы

Также метод getElementsByTagName в качестве параметра может применить звездочку (\*), это будет означать, что необходимо получить все элементы, а не какой-то конкретный.

Пример использования (\*), рисунок 17.9:

var p = document.getElementById('container1').getElementsByTagName('\*');

console.log(p);

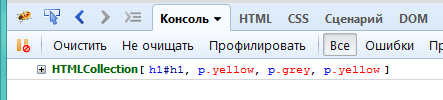


Рисунок 17.9 – Результат работы программы

Метод querySelectorAll возвращает все элементы по переданному селектору CSS.

В качестве примера выберем все параграфы с классом yellow внутри первого блока, рисунок 17.10:

var p = document.getElementById('container1').querySelectorAll('p.yellow');

console.log(p);

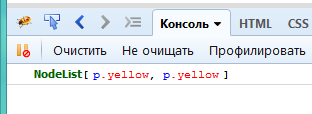


Рисунок 17.10 – Результат работы программы

Метод querySelector в отличие от предыдущего в том, что querySelector возвращает лишь один, первый попавшийся элемент, рисунок 17.11:

var p = document.getElementById('container1').querySelector('p.yellow');

console.log(p);

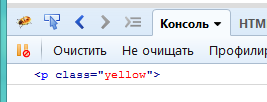


Рисунок 17.11 – Результат работы программы

Метод closest() ищет ближайший родительский элемент по заданному селектору.

Пример использования метода closest(), рисунок 17.12:

<div class="container" id="container1">

<div class="row" id="row">

<div class="col-md-12" id="col">

<h1 id="h1">Hello, world!</h1>

<p class="yellow">Yellow 1</p>

<p class="grey">Grey</p>

<p class="yellow">Yellow 2</p>

</div>

</div>

</div>

var elem = document.getElementById('container1').querySelector('h1');

console.log(elem.closest('div'));

console.log(elem.closest('#row'));

console.log(elem.closest('.container'));

console.log(elem.closest('h1'));

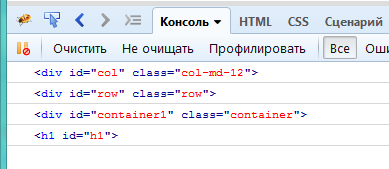


Рисунок 17.12 – Результат работы программы

Обратите внимание на последний вывод в консоль. Кроме родительского элемента, сам элемент также участвует в поиске, так что будьте аккуратны, поскольку вместо родительского элемента можно выбрать и сам элемент.

С помощью рассмотренных методов можно работать с уже готовыми элементами DOM дерева, т.е. с готовыми элементами документа.

Однако JavaScript позволяет создавать и новые элементы в документе. Сделать это можно с помощью метода createElement.

Метод appendChild позволяет добавить созданный элемент в конец.

Попробуем создать в документе какой-нибудь элемент и добавим созданный динамически элемент в body:

Пример:

var newDiv = document.createElement('div');

document.body.appendChild(newDiv);

Новый div добавлен самым последним элементом в body и разместился перед закрывающим тегом body, после подключения скриптов, рисунок 17.13.

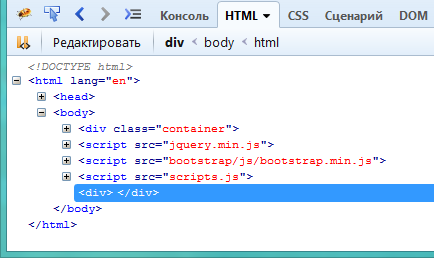


Рисунок 17.13 – Результат работы программы

Метод insertBefore позволяет добавить элемент, переданный первым параметром, перед элементом, который указан вторым параметром. Результат использования метода показан на рисунке 17.14.

var newDiv = document.createElement('div');

var script = document.getElementsByTagName('script')[0];

var parent = script.parentNode;

parent.insertBefore(newDiv, script);



Рисунок 17.14 – Результат работы программы

На самом же деле родительский элемент скриптов – это body. Соответственно, можно не получать родительский блок и сразу обратиться к body:

var newDiv = document.createElement('div');

var script = document.getElementsByTagName('script')[0];

// var parent = script.parentNode;

document.body.insertBefore(newDiv, script);

Результат будет аналогичен предыдущему.

Добавить контент во вновь созданный элемент поможет свойство innerHTML. На рисунке 17.15 приведен результат использования свойства:

var newDiv = document.createElement('div');

newDiv.innerHTML = '<h2>Заголовок</h2><p>Lorem ipsum dolor sit amet.</p>';

var script = document.getElementsByTagName('script')[0];

document.body.insertBefore(newDiv, script);

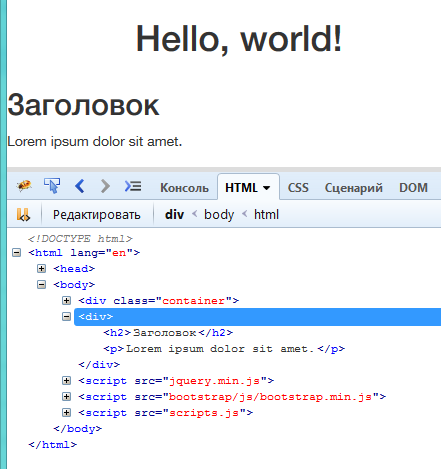


Рисунок 17.15 – Результат работы программы

Для того, чтобы добавить элементу класс, можно воспользоваться свойством className, рисунок 17.16:

var newDiv = document.createElement('div');

newDiv.innerHTML = '<h2>Заголовок</h2><p>Lorem ipsum dolor sit amet.</p>';

newDiv.className = 'container';

var script = document.getElementsByTagName('script')[0];

document.body.insertBefore(newDiv, script);

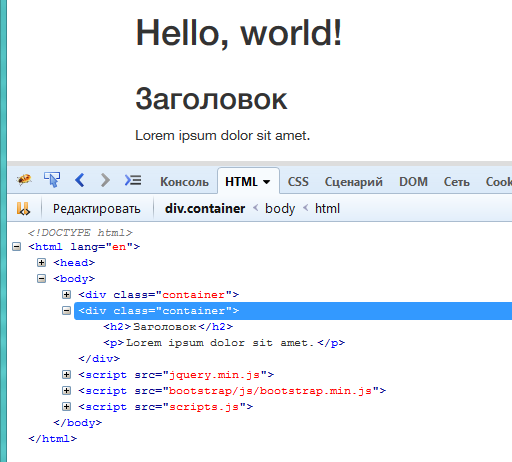


Рисунок 17.16 – Результат работы программы

Удалить элемент из DOM дерева можно с помощью метода removeChild.

Например, имеется следующая верстка:

<div class="container">

<h1>Hello, world!</h1>

<p>Lorem ipsum dolor sit amet.</p>

</div>

Необходимо удалить заголовок. Это можно сделать примерно так, рисунок 17.17:

var elem = document.getElementsByTagName('h1')[0];

elem.parentNode.removeChild(elem);

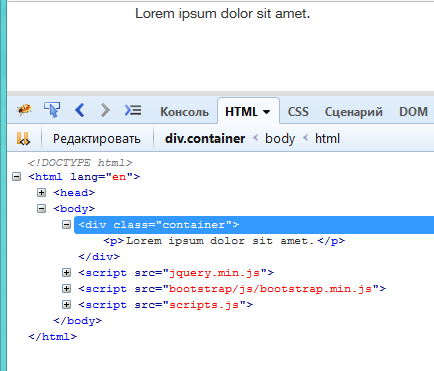


Рисунок 17.17 – Результат работы программы

Пример, получить содержимое заголовка H1, рисунок 17.18:

<div class="container" id="container1">

<div class="row" id="row">

<div class="col-md-12" id="col">

<h1 id="h1">Hello, world!</h1>

<p class="yellow">Yellow 1</p>

<p class="grey">Grey</p>

<p class="yellow">Yellow 2</p>

</div>

</div>

</div>

var elem = document.getElementById('h1');

var content = elem.innerHTML;

console.log(content);

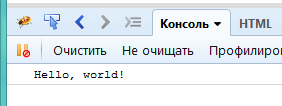


Рисунок 17.18 – Результат работы программы

Или получим содержимое всего блока, рисунок 17.19:

var elem = document.getElementById('container1');

var content = elem.innerHTML;

console.log(content);

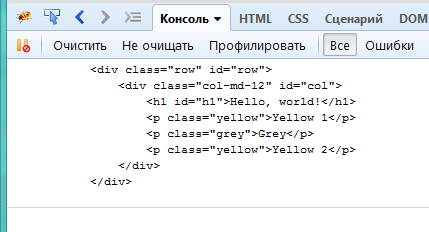


Рисунок 17.19 – Результат работы программы

Таким образом, при необходимости можно изменить полученную строку. Ну и, как отмечалось выше, свойство innerHTML позволяет записывать новое содержимое для выбранного элемента. Пример, рисунок 17.20:

var elem = document.getElementById('container1');

var content = elem.innerHTML ='<h1>Новый заголовок</h1>\n<p>Новая строка</p>';

console.log(content);

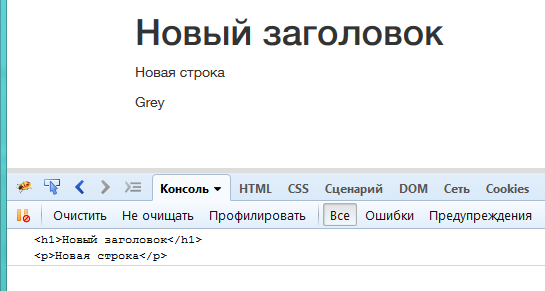


Рисунок 17.20 – Результат работы программы

Как видно на рисунке, содержимое блока полностью переписано, с помощью свойства innerHTML добавлен новый заголовок и параграф.

В отличие от innerHTML, которое получает все содержимое элемента, включая теги и контент, свойство textContent позволяет получить лишь контент, без тегов. Пример, рисунок 17.21:

<div class="container" id="container1">

<div class="row" id="row">

<div class="col-md-12" id="col">

<h1 id="h1">Hello, world!</h1>

<p class="yellow">Yellow 1</p>

<p class="grey">Grey</p>

<p class="yellow">Yellow 2</p>

</div>

</div>

</div>

var elem = document.getElementById('container1');

var content = elem.textContent;

console.log(content);

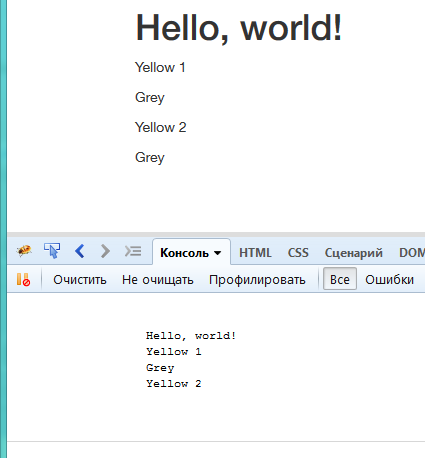


Рисунок 17.21 – Результат работы программы

Как и ожидалось, все теги были вырезаны и был получен чистый контент блока.

Кроме того, можно перезаписать текст элемента, присвоив свойству textContent элемента новое значение, рисунок 17.22:

var elem = document.getElementById('h1');

var content = elem.textContent ='Новый заголовок';

console.log(content);

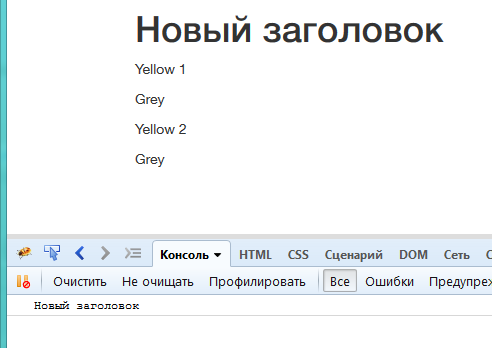


Рисунок 17.22 – Результат работы программы

Пример использования свойства bgColor (мигающий фон):

var counter =0;

setInterval(blink, 1000);

function blink() {

  if (counter % 2 == 0) document.bgColor = "red";

  else document.bgColor = "white";

  counter++;

}

Построим дерево DOM для следующего документа.

<!DOCTYPE HTML>

<html>

<head>

<title>О лосях</title>

</head>

<body>

Правда о лосях

</body>

</html>

Его вид представлен на рисунке 17.23

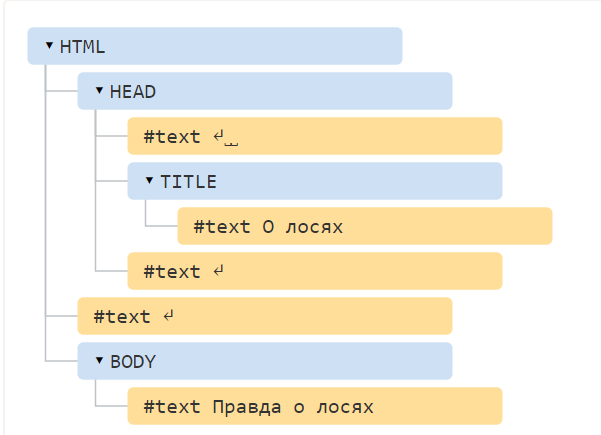


Рисунок 17.23 – дерево DOM

В этом дереве выделено два типа узлов.

Теги образуют узлы-элементы (element node). Естественным образом одни узлы вложены в другие. Структура дерева образована исключительно за счет них.

Текст внутри элементов образует текстовые узлы (text node), обозначенные как #text. Текстовый узел содержит исключительно строку текста и не может иметь потомков, то есть он всегда на самом нижнем уровне.

На рисунке выше синие узлы-элементы можно кликать, при этом их дети будут скрываться-раскрываться.

Обратите внимание на специальные символы в текстовых узлах:

перевод строки: ↵

пробел: ␣

Пробелы и переводы строки – это тоже текст, полноправные символы, которые учитываются в DOM.

В частности, в примере выше тег <html> содержит не только узлы-элементы <head> и <body>, но и #text (пробелы, переводы строки) между ними.

На самом верхнем уровне из этого правила есть исключения: пробелы до <head> по стандарту игнорируются, а любое содержимое после </body> не создаёт узла, браузер переносит его внутрь, в конец body.

В остальных случаях всё верно – если пробелы есть в документе, то они есть и в DOM, а если их убрать, то и в DOM их не будет, получится как показано на рисунке 17.24:

<!DOCTYPE HTML>

<html><head><title>О лосях</title></head><body>Правда о лосях</body></html>

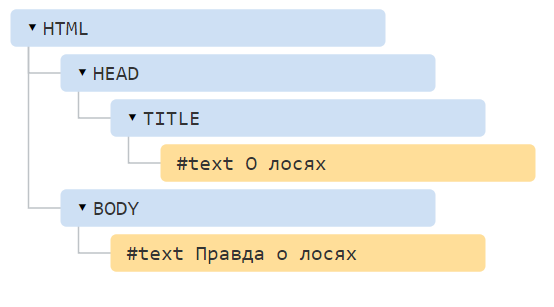


Рисунок 17.24 – дерево DOM

Важный «особый случай» при работе с DOM – таблицы. По стандарту DOM они обязаны иметь <tbody>, однако в HTML их можно написать без него. В этом случае браузер добавляет <tbody> самостоятельно.

Например, для такого HTML:

<table id="table">

<tr><td>1</td></tr>

</table>

DOM-структура будет такой, как показано на рисунке 17.25:

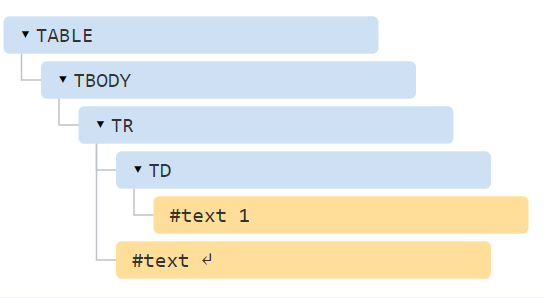


Рисунок 17.25 – структура DOM-дерева

Появился <tbody>, как будто документ был таким:

<table>

<tbody>

<tr><td>1</td></tr>

</tbody>

</table>

Директива <!DOCTYPE...> тоже является DOM-узлом, и находится в дереве DOM непосредственно перед <html>. На иллюстрациях выше этот факт скрыт, поскольку мы с этим узлом работать не будем, он никогда не нужен.

Всего различают 12 типов узлов, но на практике мы работаем с четырьмя из них:

1. документ – точка входа в DOM;
2. элементы – основные строительные блоки;
3. текстовые узлы – содержат, собственно, текст;
4. комментарии – иногда в них можно включить информацию, которая не будет показана, но доступна из JS.

DOM позволяет делать что угодно с HTML-элементом и его содержимым, но для этого нужно сначала нужный элемент получить.

Доступ к DOM начинается с объекта document. Из него можно добраться до любых узлов.

На рисунке 17.26 представлены основные ссылки, по которым можно переходить между узлами DOM.

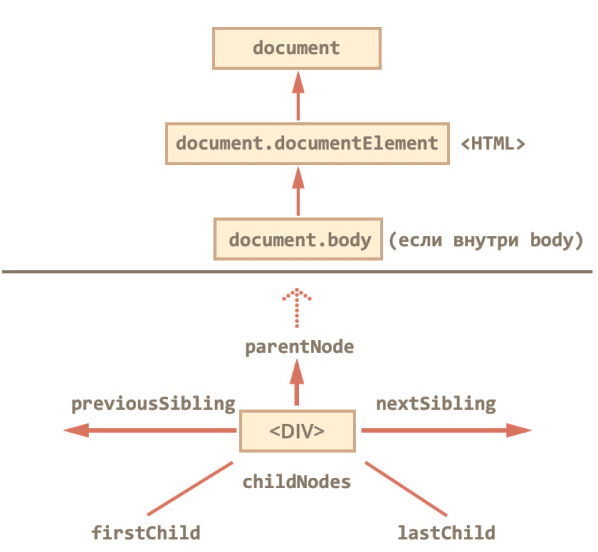


Рисунок 17.26 – Основные ссылки DOM-дерева

Самые верхние элементы дерева доступны напрямую из document.

Первая точка входа – document.documentElement. Это свойство ссылается на DOM-объект для тега <html>.

Вторая точка входа – document.body, который соответствует тегу <body>.

В современных браузерах (кроме IE8-) также есть document.head – прямая ссылка на <head>.

Есть одна тонкость: document.body может быть равен null.

Нельзя получить доступ к элементу, которого еще не существует в момент выполнения скрипта.

В частности, если скрипт находится в <head>, то в нём недоступен document.body.

Поэтому в следующем примере первый alert выведет null:

<!DOCTYPE HTML>

<html>

<head>

<script>

alert( "Из HEAD: " + document.body ); // null, body ещё нет

</script>

</head>

<body>

<script>

alert( "Из BODY: " + document.body ); // body есть

</script>

</body>

</html>

В мире DOM в качестве значения, обозначающего «нет такого элемента» или «узел не найден», используется не undefined, а null.

Дочерние элементы (или дети) – элементы, которые лежат непосредственно внутри данного. Например, внутри <HTML> обычно лежат <HEAD> и <BODY>.

Потомки – все элементы, которые лежат внутри данного, вместе с их детьми, детьми их детей и так далее. То есть, всё поддерево DOM.

Псевдо-массив childNodes хранит все дочерние элементы, включая текстовые.

Пример ниже последовательно выведет дочерние элементы document.body:

<!DOCTYPE HTML>

<html>

<body>

<div>Начало</div>

<ul>

<li>Информация</li>

</ul>

<div>Конец</div>

<script>

for (var i = 0; i < document.body.childNodes.length; i++) {

alert( document.body.childNodes[i] ); // Text, DIV, Text, UL, ..., SCRIPT

}

</script>

...

</body>

</html>

Обратим внимание на маленькую деталь. Если запустить пример выше, то последним будет выведен элемент <script>. На самом-то деле в документе есть ещё текст (обозначенный троеточием), но на момент выполнения скрипта браузер ещё до него не дошёл.

Пробельный узел будет в итоговом документе, но его еще нет на момент выполнения скрипта.

Свойства firstChild и lastChild обеспечивают быстрый доступ к первому и последнему элементу.

При наличии дочерних узлов всегда верно:

elem.childNodes[0] === elem.firstChild

elem.childNodes[elem.childNodes.length - 1] === elem.lastChild

Доступ к элементам слева и справа данного можно получить по ссылкам previousSibling / nextSibling.

Родитель доступен через parentNode. Если долго идти от одного элемента к другому, то рано или поздно дойдёшь до корня DOM, то есть до document.documentElement, а затем и document.

Навигационные ссылки равно касаются всех узлов в документе. В частности, в childNodes сосуществуют и текстовые узлы и узлы-элементы и узлы-комментарии, если есть.

На рисунке 17.27 представлен дополнительный набор ссылок.

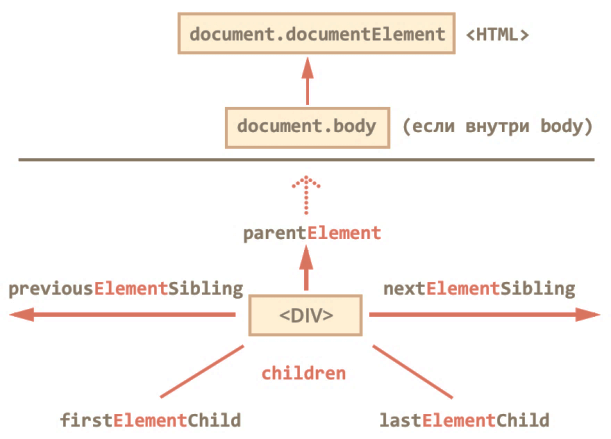


Рисунок 17.27 – Дополнительные ссылки DOM-дерева

Эти ссылки похожи на те, что раньше, только в ряде мест стоит слово Element:

– children – только дочерние узлы-элементы, то есть соответствующие тегам;

– firstElementChild, lastElementChild – соответственно, первый и последний дети-элементы;

– previousElementSibling, nextElementSibling – соседи-элементы;

– parentElement – родитель-элемент.

Свойство elem.parentNode возвращает родитель элемента.

Оно всегда равно parentElement, кроме одного исключения:

alert( document.documentElement.parentNode ); // document

alert( document.documentElement.parentElement ); // null

Иногда это имеет значение, если хочется перебрать всех предков и вызвать какой-то метод, а на документе его нет.

У конкретных элементов DOM могут быть свои дополнительные ссылки для большего удобства навигации.

Здесь мы рассмотрим таблицу, так как это важный частный случай и просто для примера.

В списке ниже выделены наиболее полезные:

TABLE

table.rows – коллекция строк TR таблицы.

table.caption/tHead/tFoot – ссылки на элементы таблицы CAPTION, THEAD, TFOOT.

table.tBodies – коллекция элементов таблицы TBODY, по спецификации их может быть несколько.

THEAD/TFOOT/TBODY

tbody.rows – коллекция строк TR секции.

TR

tr.cells – коллекция ячеек TD/TH

tr.sectionRowIndex – номер строки в текущей секции THEAD/TBODY

tr.rowIndex – номер строки в таблице

TD/TH

td.cellIndex – номер ячейки в строке

Пример использования:

<table>

<tr>

<td>один</td> <td>два</td>

</tr>

<tr>

<td>три</td> <td>четыре</td>

</tr>

</table>

<script>

var table = document.body.children[0];

alert( table.rows[0].cells[0].innerHTML ) // "один"

</script>

**5. Порядок выполнения работы**

1. Используя JavaScript, сделайте так, чтобы при клике на кнопку исчезал элемент с id="text".

2. Выведите всех потомков тега body.

3. Выполните задание по варианту. Варианты представлены в таблице 17.1.

Таблица 17.1 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| **1** | **2** |
| 1 | 1. Создайте кнопку, при клике на которую, она будет скрывать сама себя 2. Вывести содержимое всех параграфов на странице по нажатию на кнопку |
| 2 | 1. Создайте меню, которое раскрывается/сворачивается при клике 2. Вывести содержимое всех заголовков H1 на странице по нажатию на кнопку |
| 3 | 1. Реализовать переключатель (чекбокс). В активном состоянии блок с id #checkactive будет красного цвета, а в неактивном – черного 2. Вывести содержимое второго нумерованного списка. Структура:   <ol>  <li>first ol li</li>  <li>second ol li</li>  </ol>  <ul>  <li>first ul li</li> |

|  |  |
| --- | --- |
| **1** | **2** |
|  | <li>second ul li</li>  </ul>  <ol>  <li>first ol li</li>  <li>second ol li</li>  </ol> |
| 4 | 1. Создайте форму, которая будет скрываться при корректной отправке данных. 2. Реализуйте скрипт, в котором при нажатии на кнопку, необходимо добавить в div с id #lab17 фрагмент разметки:   <div class="test">  <h3>Заголовок</h3>  <p>текст текст текст</p>  </div> |
| 5 | 1. При нажатии на гиперссылку менять фоновый цвет элемента с классом .nocolor на прозрачный. 2. Выведите название тегов дочерних элементов. |
| 6 | 1. В нумерованном списке удалить первый и последний элемент списка. Добавить удаленные элементы в конец списка. 2. Выведите содержимое «соседов» тега div |

\* Есть дерево из тегов <ul>/<li>. Напишите код, который для каждого элемента <li> выведет:

1) текст непосредственно в нём (без подразделов);

2) количество вложенных в него элементов <li> – всех, с учётом вложенных.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой объектная модель документа (DOM)?

2. Что такое DOM-дерево?.

3. Перечислите известные Вам узлы DOM-дерева.

4. Опишите методы объекта document.

5. Опишите свойства объекта document.

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 18**

## **Тема работы: «Управление CSS-свойствами HTML-элементов. Создание анимации средствами JavaScript»**

**1. Цель работы**

Формирование умений динамического изменения CSS -свойств HTML-элементов, создания анимации в JavaScript.

**2. Задание**

Реализовать динамическое изменение стилей HTML-элементов, анимации в HTML-документе.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Для работы со стилевыми свойствами элементов в JavaScript применяются, главным образом, два подхода:

- изменение свойства style;

- изменение значения атрибута class.

Свойство style представляет сложный объект для управления стилем и напрямую сопоставляется с атрибутом style html-элемента. Этот объект содержит набор свойств CSS: element.style.свойствоCSS.

Пример, установим цвет шрифта:

var root = document.documentElement;

// устанавливаем стиль

root.style.color = "blue";

// получаем значение стиля

document.write(root.style.color); // blue

В данном случае название свойства color совпадает со свойством css. Аналогично можно установить цвет с помощью css:

html{

color:blue;

}

Однако ряд свойств css в названиях имеют дефис, например, font-family. В JavaScript для этих свойств дефис не употребляется. Только первая буква, которая идет после дефиса, переводится в верхний регистр:

var root = document.documentElement;

root.style.fontFamily = "Verdana";

Свойство element.style возвращает объект, который дает доступ к стилю элемента на чтение и запись.

С его помощью можно изменять большинство CSS-свойств, например element.style.width="100px" работает так, как будто у элемента в атрибуте прописано style="width:100px".

Для свойств, названия которых состоят из нескольких слов, используется запись:

background-color => elem.style.backgroundColor

z-index => elem.style.zIndex

border-left-width => elem.style.borderLeftWidth

Исключением является свойство float. В старом стандарте JavaScript слово "float" было зарезервировано и недоступно для использования в качестве свойства объекта. Поэтому используется не elem.style.float, а elem.style.cssFloat.

Специфические свойства браузеров, типа -moz-border-radius, -webkit-border-radius, записываются следующим способом:

button.style.MozBorderRadius = '5px';

button.style.WebkitBorderRadius = '5px';

То есть, каждый дефис даёт большую букву.

Чтобы сбросить поставленный стиль, присваивают в style пустую строку: elem.style.width="".

При сбросе свойства style стиль будет взят из CSS.

Например, для того, чтобы спрятать элемент, можно присвоить: elem.style.display = "none".

А вот чтобы показать его обратно – не обязательно явно указывать другой display, наподобие elem.style.display = "block". Можно просто снять поставленный стиль: elem.style.display = "".

Свойство style является специальным объектом, ему нельзя присваивать строку.

Запись div.style="color:blue" работать не будет.

Можно попробовать использовать атрибут: elem.setAttribute("style", ...), но самым правильным и, главное, кросс-браузерным (с учётом старых IE) решением такой задачи будет использование свойства style.cssText.

Свойство style.cssText позволяет поставить стиль целиком в виде строки.

Например:

<div>Button</div>

<script>

var div = document.body.children[0];

div.style.cssText="color: red !important; \

background-color: yellow; \

width: 100px; \

text-align: center; \

";

alert(div.style.cssText);

</script>

Браузер разбирает строку style.cssText и применяет известные ему свойства.

При установке style.cssText все предыдущие свойства style удаляются.

Итак, style.cssText осуществляет полную перезапись style. Если же нужно заменить какое-то конкретно свойство стиля, то обращаются именно к нему: style.color, style.width и т.п, чтобы не затереть что-то важное по ошибке.

С помощью свойства className можно установить атрибут class элемента html:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8" />

    <style>

    .blueStyle{

        color:blue;

        font-family:Verdana;

    }

    .article{

        font-size:20px;

    }

    </style>

</head>

<body>

<div class="article">

<h3>Заголовок статьи</h3>

<p>Первый абзац</p>

<p>Второй абзац</p>

</div>

<script>

var articleDiv = document.querySelector("div.article");

// установка нового класса

articleDiv.className = "blueStyle";

// получаем название класса

document.write(articleDiv.className);

</script>

</body>

</html>

Благодаря использованию классов не придется настраивать каждое отдельное свойство css с помощью свойства style.

Но при этом надо учитывать, что прежнее значение атрибута class удаляется. Поэтому, если необходимо добавить класс, надо объединить его название со старым классом:

articleDiv.className = articleDiv.className + " blueStyle";

И если надо вовсе удалить все классы, то можно присвоить свойству пустую строку:

articleDiv.className = "";

Для управления множеством классов гораздо удобнее использовать свойство classList. Это свойство представляет объект, реализующий следующие методы:

- add(className) – добавляет класс className;

- remove(className) – удаляет класс className;

- toggle(className) – переключает у элемента класс на className. Если класса нет, то он добавляется, если есть, то удаляется.

Пример:

var articleDiv = document.querySelector("div.article");

// удаляем класс

articleDiv.classList.remove("article");

// добавляем класс

articleDiv.classList.add("blueStyle");

// переключаем класс

articleDiv.classList.toggle("article");

JavaScript-анимация применяется там, где не подходит CSS.

Например, по сложной траектории, с временной функцией, выходящей за рамки кривых Безье, на canvas. Иногда её используют для анимации в старых IE.

setInterval. С точки зрения HTML/CSS, анимация – это постепенное изменение стиля DOM-элемента. Например, увеличение координаты style.left от 0px до 100px сдвигает элемент.

Если увеличивать left от 0 до 100 при помощи setInterval, делая по 50 изменений в секунду, то это будет выглядеть как плавное перемещение. Тот же принцип, что и в кино: для непрерывной анимации достаточно 24 или больше вызовов setInterval в секунду.

Псевдо-код для анимации выглядит так:

var fps = 50; // 50 кадров в секунду

var timer = setInterval(function() {

if (время вышло) clearInterval(timer);

else немного увеличить left

}, 1000 / fps)

Более полный пример кода анимации:

var start = Date.now(); // сохранить время начала

var timer = setInterval(function() {

// вычислить сколько времени прошло с начала анимации

var timePassed = Date.now() - start;

if (timePassed >= 2000) {

clearInterval(timer); // конец через 2 секунды

return;

}

// рисует состояние анимации, соответствующее времени timePassed

draw(timePassed);

}, 20);

// в то время как timePassed идёт от 0 до 2000

// left принимает значения от 0 до 400px

function draw(timePassed) {

train.style.left = timePassed / 5 + 'px';

}

Если у нас не один такой setInterval, а несколько в разных местах кода, то браузеру нужно в те же 20 мс работать со страницей уже несколько раз. А ведь кроме setInterval есть ещё другие действия, к примеру, прокрутка страницы, которую тоже надо нарисовать.

Если все действия по перерисовке производить независимо, то будет выполняться много двойной работы.

Гораздо выгоднее с точки зрения производительности – сгруппировать все перерисовки в одну и запускать их централизованно, все вместе.

Для этого в JavaScript-фреймворках, которые поддерживают анимацию, есть единый таймер:

setInterval(function() {

/\* отрисовать все анимации \*/

}, 20);

Все анимации, которые запускает такой фреймворк, добавляются в общий список, и раз в 20 мс единый таймер проверяет его, запускает текущие, удаляет завершившиеся.

Современные браузеры, кроме IE9-, поддерживают стандарт Animation timing, который представляет собой дальнейший шаг в этом направлении. Он позволяет синхронизировать анимации со встроенными механизмами обновления страницы. То есть, сгруппированы будут не только пользовательские, но и CSS-анимации и другие браузерные перерисовки.

При этом графический ускоритель будет использован максимально эффективно, и исключена повторная обработка одних и тех же участков страницы. А значит – меньше будет загрузка CPU, да и сама анимация станет более плавной.

Для этого используется функция requestAnimationFrame.

Синтаксис:

var requestId = requestAnimationFrame(callback)

Такой вызов планирует запуск callback в ближайшее время, когда браузер сочтёт возможным осуществить анимацию.

Если запланировать в callback какое-то рисование, то оно будет сгруппировано с другими requestAnimationFrame и с внутренними перерисовками браузера.

Возвращаемое значение requestId служит для отмены запуска:

// отменить запланированное выше выполнение callback

cancelAnimationFrame(requestId);

Функция callback получает один аргумент – время, прошедшее с начала загрузки страницы, результат вызова performance.now().

Как правило, запуск callback происходит очень скоро. Если у процессора большая загрузка или батарея у ноутбука почти разряжена – то пореже.

Если вы запустить код, то можно увидеть промежутки между первыми 20 запусками requestAnimationFrame. Как правило, это 10-20 мс, но бывает и больше, и меньше. Это оптимальная частота анимации с точки зрения браузера.

<script>

var prev = performance.now();

var times = 0;

requestAnimationFrame(function measure(time) {

document.body.insertAdjacentHTML("beforeEnd", Math.floor(time - prev) + " ");

prev = time;

if (times++ < 10) requestAnimationFrame(measure);

})

</script>

Функция анимации на основе requestAnimationFrame:

// Рисует функция draw

// Продолжительность анимации duration

function animate(draw, duration) {

var start = performance.now();

requestAnimationFrame(function animate(time) {

// определить, сколько прошло времени с начала анимации

var timePassed = time - start;

// возможно небольшое превышение времени, в этом случае зафиксировать конец

if (timePassed > duration) timePassed = duration;

// нарисовать состояние анимации в момент timePassed

draw(timePassed);

// если время анимации не закончилось - запланировать ещё кадр

if (timePassed < duration) {

requestAnimationFrame(animate);

}

});

}

Использование для поезда:

animate(function(timePassed) {

train.style.left = timePassed / 5 + 'px';

}, 2000);

На основе requestAnimationFrame можно соорудить и гораздо более мощную, но в то же время простую функцию анимации.

У анимации есть три основных параметра:

– duration – общее время, которое должна длиться анимация, в мс. Например, 1000;

– timing(timeFraction) – временная функция, которая, по аналогии с CSS-свойством transition-timing-function, будет по текущему времени вычислять состояние анимации. Она получает на вход непрерывно возрастающее число timeFraction – от 0 до 1, где 0 означает самое начало анимации, а 1 – её конец.

Её результатом должно быть значение завершённости анимации, которому в CSS transitions на кривых Безье соответствует координата y.

Также по аналогии с transition-timing-function должны соблюдаться условия:

timing(0) = 0

timing(1) = 1

То есть, анимация начинается в точке (0,0) – нулевое время и нулевой прогресс и заканчивается в (1, 1) – прошло полное время, и процесс завершён.

Например, функция-прямая означает равномерное развитие процесса:

function linear(timeFraction) {

return timeFraction;

}

– draw(progress) – функция, которая получает состояние завершённости анимации и рисует его. Значению progress=0 соответствует начальная точка анимации, progress=1 – конечная.

Именно эта функция и осуществляет, собственно, анимацию.

Например, может двигать элемент:

function draw(progress) {

train.style.left = progress + 'px';

}

Возможны любые варианты, анимировать можно что угодно и как угодно. Анимируем ширину элемента width от 0 до 100%, используя нашу функцию. Рассмотрим примеры анимации движения с использованием различных timing.

В степени n. Вот еще один простой случай – progress в степени n. Частные случаи – квадратичная, кубическая функции и т.д.

Для квадратичной функции:

function quad(progress) {

return Math.pow(progress, 2)

}

График квадратичной функции показан на рисунке 18.1:

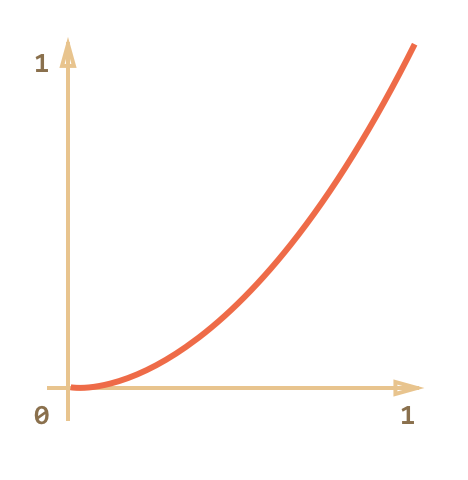


Рисунок 18.1 – График квадратичной функции

Для дуги:

function circ(timeFraction) {

return 1 - Math.sin(Math.acos(timeFraction))

}

График дуги показан на рисунке 18.2

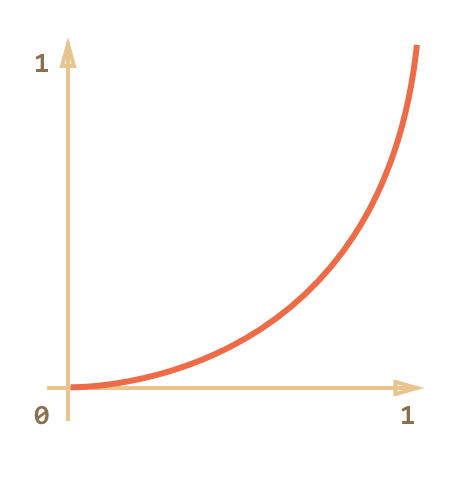


Рисунок 18.2 – График дуги

Back: стреляем из лука

Эта функция работает по принципу лука: сначала мы «натягиваем тетиву», а затем «стреляем».

В отличие от предыдущих функций, эта зависит от дополнительного параметра x, который является «коэффициентом упругости». Он определяет расстояние, на которое «оттягивается тетива».

function back(x, timeFraction) {

return Math.pow(timeFraction, 2) \* ((x + 1) \* timeFraction - x)

}

График для x = 1.5 показан на рисунке 18.3

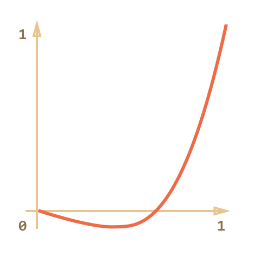


Рисунок 18.3 – График функции «по принципу лука»

Иногда нужно показать анимацию в обратном режиме. Преобразование функции, которое даёт такой эффект, называется «easeOut».

В режиме «easeOut», значение timing вычисляется по формуле: timingEaseOut(timeFraction) = 1 - timing(1 - timeFraction)

А еще можно сделать так, чтобы показать эффект и в начале и в конце анимации. Соответствующее преобразование называется «easeInOut».

**5. Порядок выполнения работы**

1. Создайте плавно выпадающее меню.

2. Создайте кнопку, которая при наведении плавно меняет свой цвет.

3. Реализуйте слайдер минимум из 5 изображений. Реализуйте плавную смену изображений.

4. Выполните задание по варианту в соответствии с таблицей 18.1.

Таблица 18.1 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| 1 | 1. Создайте индикатор выполнения процесса от 0 до 100%. Измените цвет индикатора после начала анимации 2. Создайте кнопку и текстовое поле ввода. По нажатию на кнопку измените шрифт текста в текстовом поле ввода |
| 2 | 1. Создайте анимацию прыгающего мячика. В процессе выполнения анимации измените цвет мячика 2. Создайте кнопку и текстовое поле ввода. По нажатию на кнопку измените начертание текста в текстовом поле ввода |
| 3 | 1. Создайте анимацию мигания разделителя (13:27, : - разделитель) электронных часов 2. Создайте кнопку и текстовое поле ввода. По нажатию на кнопку измените цвет текста в текстовом поле ввода |
| 4 | 1. Создайте анимацию появления информационного блока при наведении на фото. При наведении на фото измените его размер 2. Создайте кнопку и текстовое поле ввода. По нажатию на кнопку измените выравнивание текста в текстовом поле ввода |
| 5 | 1. Создайте анимацию движения Луны вокруг Земли. 2. Создайте кнопку и текстовое поле ввода. По нажатию на кнопку измените размер текста в текстовом поле ввода |
| 6 | 1. Создайте иконку меню, которая из трех полос при наведении преобразуется в круг с крестиком 2. Создайте кнопку и текстовое поле ввода. По нажатию на кнопку измените шрифт текста в текстовом поле ввода |
| 7 | 1. Создайте появляющийся и увеличивающийся в размере текст при загрузке страницы. 2. Создайте кнопку и текстовое поле ввода. По нажатию на кнопку измените размер текста в текстовом поле ввода |
| 8 | 1. Создайте анимацию движения текста по кругу 2. Создайте кнопку и текстовое поле ввода. По нажатию на кнопку измените цвет текста в текстовом поле ввода |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой объект Style?

2. Опишите способы изменения стилевых свойств элементов.

3. В каких случаях целесообразно использовать свойство classList?

4. Для чего необходим setInterval?

5. Опишите структуру анимации.

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 19**

## **Тема работы: «Использование разных моделей событий. Остановка всплытия и перехват событий»**

**1. Цель работы**

Формирование умений назначения обработчиков событий в сценариях на языке JavaScript.

**2. Задание**

Реализовать назначение обработчиков событий в сценариях на языке Java Script.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Событие в JavaScript – это определённое действие, которое вызвано либо пользователем, либо браузером. К ним относятся движение мыши, нажатия на кнопки мыши и клавиатуры, наведение фокуса на элемент, изменение значения в каком-нибудь текстовом поле, изменение размеров окна браузера и так далее.

События являются основным инструментом создания динамических страниц. Все события объединены в отдельный интерфейс событий.

Для реакции на определённые события (действия пользователя) назначаются обработчики событий.

Обработчик события – это функция, которая выполняется в тот момент, когда происходит событие, и определяет, что будет происходить при возникновении определённого события. Обработчики событий имеют следующий общий вид:

onНазваниеСобытия

То есть вначале идёт приставка "on", а дальше название события, например, такие обработчики событий: onFocus, onClick, onSubmit и так далее.

Если пользователем инициируются сразу несколько событий, то они попадают в очередь и выполняются друг за другом. Например, простое с точки зрения пользователя нажатие на кнопку клавиатуры вызывает сразу три события в такой последовательности:

- keydown - при нажатии на кнопку;

- keypress - при нажатии на кнопку;

- keyup - при отпускании кнопки.

Обработчики событий могут быть привязаны к объекту Element, Document, Window и т.д. Затем, в тот момент, когда происходит какое-либо событие, создаётся объект Event (событие), который передаётся в качестве аргумента обработчику события.

Интерфейс события объектной модели документа (DOM) доступен только через объект Event, который передаётся в качестве аргумента в обработчик события.

Следующий пример показывает, как объект Event передаётся обработчику события и может быть использован внутри него:

window.addEventListener("keydown", foo, false);

function foo(event) {

  // параметр event неявно инициализируется объектом Event

  alert(event);

}

В таблице 19.1 приведены основные события в JavaScript: название события, элементы, которые могут генерировать это событие и описание самого события.

Таблица 19.1 – Основные события в JavaScript

| **Событие** | **Объект** | **Причина возникновения** |
| --- | --- | --- |
| **1** | **2** | **3** |
| Abort | Image | Прерывание загрузки изображения |
| Blur | Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window | Потеря фокуса элемента |
| Change | FileUpload, Select, Text, Textarea | Смена значения |
| Click | Area, Button, Checkbox, Document, Link, Radio, Reset, Submit | Клик мыши на элементе |
| DblClick | Area, Document, Link | Двойной клик на элементе |
| DragDrop | Window | Перемещение в окно браузера |
| Focus | Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window | Установка фокуса на элементе |
| KeyDown | Document, Image, Link, Textarea | Нажатие клавиши на клавиатуре |
| KeyPress | Document, Image, Link, Textarea | Удержание клавиши на клавиатуре |
| KeyUp | Document, Image, Link, Textarea | Отпускание клавиши на клавиатуре |
| Load | Document, Image, Layer, Window | Загрузка элемента |
| MouseDown | Button, Document, Link | Нажата кнопка мыши |
| MouseMove | Window | Мышь в движении |
| MouseOut | Area, Layer, Link | Мышь выходит за границы элемента |

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| MouseOver | Area, Layer, Link | Мышь находится над элементом |
| MouseUp | Button, Document, Link | Отпущена кнопка мыши |
| Move | Frame | Перемещение элемента |
| Reset | Form | Сброс формы |
| Resize | Frame, Window | Изменение размеров |
| Select | Text, Textarea | Выделение текста |
| Submit | Form | Передача данных |
| Unload | Window | Выгрузка текущей страницы |

Назначить обработчик события можно несколькими способами.

1. Самый простой способ задания обработчика – это использование [атрибутов событий](http://webgent.ru/html/event-attribute) прямо в HTML-коде.

Значением атрибутов событий указывается код JavaScript или название готовой функции. Если указывается просто код JavaScript, то браузер автоматически создаёт анонимную функцию, телом которой является исходный код.

<html>

<head>

<title>События</title>

</head>

<body>

<p onmouseover="this.style.background = 'red';" onmouseout="this.style.background = 'transparent';">Наведите курсор</p>

</body>

</html>

<html>

<head>

<title>События</title>

</head>

<body>

<p onmouseover="alert('Ok');">Наведите курсор</p>

</body>

</html>

<script>

var par = document.body.firstElementChild;

alert(par.onmouseover); /\* 'function ...' (в свойстве записана функция) \*/

</script>

При использовании HTML-атрибутов для задания обработчика события разработчик ограничен стандартом HTML. Использование DOM-интерфейса расширяет возможности по созданию динамических страниц, позволяя устанавливать обработчики на любые события.

2. Ещё один способ установки обработчика события – это прямое использование DOM свойств, то есть сразу записывать обработчик в свойство элемента.

Имя свойства должно быть в нижнем регистре. В отличие от HTML-атрибутов DOM-свойства регистрозависимы.

Значением свойства обязательно должна быть функция.

<html>

<head>

<title>События</title>

</head>

<body>

<p>Наведите курсор</p>

</body>

</html>

<script>

var par = document.body.firstElementChild;

par.onmouseover = function () {

this.style.background = 'green';

};

par.onmouseout = function () {

this.style.background = 'transparent';

};

</script>

Использование HTML-атрибутов или DOM-свойств позволяет устанавливать только по одному обработчику на каждое событие.

3. Для установки и удаления любого количества обработчиков на одно событие используются методы addEventListener() и removeEventListener().

Метод addEventListener() добавляет обработчик события.

element.addEventListener(имя\_события, обработчик[, стадия])

где имя\_события – название события в кавычках;

обработчик – функция-обработчик события (можно указать анонимную функцию или ссылку на готовую);

стадия – необязательный аргумент. Он содержит информацию, на какой [стадии](http://webgent.ru/javascript/event-stage) сработает обработчик: (true – на стадии перехвата, false (используется по умолчанию) – на стадии всплытия).

<html>

<head>

<title>События</title>

</head>

<body>

<input type="button" value="Кнопка">

</body>

</html>

<script>

var button = document.body.firstElementChild;

button.addEventListener('click', function() {alert('Обработчик 1');});

button.addEventListener('click', function() {alert('Обработчик 2');});

</script>

При нажатии на кнопку сработают оба обработчика.

Метод removeEventListener() используется для удаления обработчика события, назначенного с помощью метода addEventListener(). Обязательно должны быть указаны те значения, которые использовались при назначении обработчика.

element.removeEventListener(имя\_события, обработчик[, стадия])

где имя\_события – название события в кавычках;

обработчик – функция-обработчик события. Должна указываться именно ссылка, чтобы браузер понимал, какой обработчик нужно удалить. Соответственно, если обработчик был назначен напрямую (анонимной функцией), то его удалить не удастся;

стадия – необязательный аргумент. Указывается, если использовался при назначении обработчика.

<html>

<head>

<title>События</title>

</head>

<body>

<input type="button" value="Кнопка">

</body>

</html>

<script>

function handler() {alert('Обработчик 1');}

var button = document.body.firstElementChild;

/\* добавление обработчиков \*/

button.addEventListener('click', handler);

button.addEventListener('click', function() {alert('Обработчик 2');});

/\* удаление обработчиков \*/

button.removeEventListener('click', handler);

button.removeEventListener('click', function() {alert('Обработчик 2');});

</script>

Второй обработчик остался, так как он назначен анонимной функцией.

Все интернет-страницы состоят из отдельных элементов (тегов). При выводе станицы на экран браузер создаёт DOM-дерево, отражающее структуру документа (связи между родительскими и дочерними элементами).

В DOM-модели каждый дочерний элемент является частью родительского элемента. Следовательно, если на элементе возникает какое-либо событие (например, клик мышью), то оно же возникает и на его родителе, и так далее вверх по DOM-дереву вплоть до корневого элемента (объект document). Если нескольким из таких элементов будут назначены обработчики события, то все они выполнятся.

В примере ниже клик мышью по элементу <mark> вызовет обработчик onclick и на родительском элементе <div>.

<html>

<head>

<title>События</title>

</head>

<body>

<div onclick="alert('Обработчик DIV')">

<mark onclick="alert('Обработчик MARK')">MARK</mark> внутри DIV

</div>

</body>

</html>

Чтобы знать, в какой последовательности будут выполняться обработчики, необходимо иметь представление о том, как браузер обрабатывает событие.

Само исходное событие для всех обработчиков является общим. Его обработка делится на три стадии:

- стадия перехвата (capturing stage);

- стадия цели (target stage);

- стадия всплытия (bubbling stage).

Элемент, на котором возникает событие, называется целевым.

При возникновении события браузер поочерёдно проходит от верхнего элемента DOM-дерева (document) вниз через все промежуточные элементы к целевому. Эта стадия обработки называется стадией перехвата. Обработчики запускаются до того, как событие дойдёт до целевого элемента. Событие как бы перехватывается. Отсюда и следует такое название стадии.

Обработчики события выполняются только в том случае, если для них задано выполнение на стадии перехвата. Для этого необходимо использовать значение true для третьего атрибута метода [addEventListener()](http://webgent.ru/javascript/event-handler#addEventListener). Это единственный способ использовать обработчик на стадии перехвата.

Стадия перехвата используется очень редко, но иногда может пригодиться.

После того, как событие опустилось до целевого элемента, стадия перехвата завершается и выполняются обработчики целевого элемента. Это вторая стадия – стадия цели. Очерёдность выполнения обработчиков на целевом элементе зависит только от очерёдности их назначения. Использование третьего аргумента метода addEventListener() никак не повлияет на порядок их запуска.

Далее начинается последняя стадия. Событие проходит поочерёдно от целевого элемента через цепочку родителей до корневого элемента документа. Событие продвигается вверх по DOM-дереву. Оно будто всплывает. Отсюда и название – стадия всплытия.

На данной стадии выполняются все остальные обработчики. Обработчики, назначенные с помощью атрибутов событий и DOM-свойств всегда выполняются на стадии всплытия. В методе addEventListener() для использования стадии всплытия можно просто опустить третий аргумент.

Прохождение всех стадий продемонстрировано на примере ниже (запускается кликом по жёлтому элементу):

<html>

<head>

<title>События</title>

<style>

div {

width: 60px;

height: 100px;

position: relative;

background: red;

}

aside {

width: 80px;

height: 60px;

background: yellow;

position: absolute;

top: 20px;

left: 70px;

}

</style>

</head>

<body>

<div>DIV

<aside>ASIDE</aside>

</div>

</body>

</html>

<script>

var div = document.getElementsByTagName('div')[0];

var aside = document.getElementsByTagName('aside')[0];

div.addEventListener('click', function() {alert('DIV - перехват')}, true);

div.addEventListener('click', function() {alert('DIV - всплытие')});

aside.addEventListener('click', function() {alert('ASIDE - всплытие')});

aside.addEventListener('click', function() {alert('ASIDE - перехват')}, true);

</script>

// div-перехват, aside-всплытие, aside-перехват, div-всплытие

Обработчики элемента <aside> запускаются в порядке назначения.

В этом примере наглядно видно, что событие всегда всплывает по дереву DOM, несмотря на то, что визуально событие не произошло над элементом <div>.

Помимо выполнения назначенных обработчиков, события могут вызывать действия браузера по умолчанию. Эти действия как бы подразумеваются сами собой в зависимости от элемента страницы, таблица 19.2.

Таблица 19.2 – Основные события по умолчанию

|  |  |
| --- | --- |
| **Событие** | **Действие браузера** |
| **click** | Клик по ссылке вызывает переход на новую страницу. |
| **click** | Клик по неактивному полю для ввода текста делает его активным. |
| **contextmenu** | Клик правой кнопкой мыши открывает контекстное меню. |
| **dblclick** | Двойной клик на тексте выделяет его. |
| **mousedown** | Нажатие левой кнопки мыши и удержание её над текстом начинает его выделение. |
| **mousewheel** | Движение колёсика мыши вызывает прокрутку страницы. |
| **keydown** | Нажатия на кнопки клавиатуры внутри текстового поля приводят к набору текста. |
| **keydown** | Нажатие на кнопку **Enter** в активном поле вызывает отправку формы на сервер. |

Обычно, если на конкретное событие устанавливается обработчик, то действия браузера по умолчанию не нужны. Тогда их просто можно отменить. Отменить действия браузера по умолчанию можно простым вызовом return false в конце обработчика. Но это можно использовать только внутри обработчиков, установленных через HTML-атрибут или DOM-свойство.

<html>

<head>

<title>События</title>

</head>

<body>

<label>Попробуйте поставить галочку <input type="checkbox" onclick="return false;"></label>

</body>

</html>

При помощи мыши поставить галочку не получится.

Если обработчик устанавливается с помощью метода [addEventListener()](http://webgent.ru/javascript/event-handler#addEventListener), тогда вызов return false ни к чему не приведёт. Универсальным способом для отмены действий браузера по умолчанию является встроенный метод event.preventDefault().

Вызов preventDefault() во время любой стадии обработки события отменяет действия браузера. Действия назначенных обработчиков данный метод не отменяет.

<html>

<head>

<title>События</title>

</head>

<body>

<input type="text" placeholder="Введите текст">

</body>

</html>

<script>

document.body.firstElementChild.onkeypress = function(event) {

event.preventDefault();

alert('Отменено');

}

</script>

В получившемся поле нельзя ввести текст, используя клавиатуру.

Действия браузера можно отменить не для всех событий. Вызов preventDefault() на неотменяемом событии результата не даст.

Для остановки выполнения события на текущем элементе используется метод event.stopPropagation(). Данный метод отменяет выполнение обработчиков тех элементов, до которых событие не дошло. Однако, все обработчики текущего элемента будут выполнены даже после вызова stopPropagation().

Метод stopPropagation() не отменяет действия браузера по умолчанию.

<html>

<head>

<title>События</title>

</head>

<body>

<form name="test">

<input type="text" placeholder="Введите текст" onkeypress="alert('input')">

</form>

</body>

</html>

<script>

document.forms.test.addEventListener('keypress', function(event) {

alert('form 1');

event.stopPropagation();

}, true);

document.forms.test.addEventListener('keypress', function(event) {

alert('form 2');

}, true);

</script>

// form1, form2

Первый обработчик элемента <form> на стадии перехвата отменяет дальнейшее распространение события keypress. При этом второй обработчик элемента <form> всё равно выполняется. А также выполняется действие браузера по умолчанию.

Чтобы отменить выполнение всех невыполненных обработчиков события, даже назначенных текущему элементу, используется метод event.stopImmediatePropagation().

Если одному элементу на одно событие назначено несколько обработчиков, то они выполняются в порядке назначения. Если один из них вызывает метод stopImmediatePropagation(), тогда оставшиеся обработчики не выполняются. Дальнейшее распространение события не происходит.

Метод stopImmediatePropagation() не отменяет действия браузера по умолчанию.

<html>

<head>

<title>События</title>

</head>

<body>

<form name="test">

<input type="text" placeholder="Введите текст" onkeypress="alert('input')">

</form>

</body>

</html>

<script>

document.forms.test.addEventListener('keypress', function(event) {

alert('form 1');

event.stopImmediatePropagation();

}, true);

document.forms.test.addEventListener('keypress', function(event) {

alert('form 2');

}, true);

</script>

Первый обработчик элемента <form> на стадии перехвата отменяет выполнение последующих обработчиков события keypress, назначенных текущему элементу, а также предотвращает дальнейшее распространение события. Второй обработчик элемента <form> не выполняется. Действие браузера по умолчанию всё равно выполняется.

Пример:

<html>

<head>

<style>

  a {

    color: blue;

    text-decoration: underline;

    cursor: pointer;

  }

</style>

<script language = 'javascript'>

var default\_color = "white";

function setTempColor(color) {

  document.bgColor = color;

}

function setDefaultColor(color) {

  default\_color = color;

  document.bgColor = default\_color;

}

function defaultColor() {

  document.bgColor = default\_color;

}

</script>

</head>

<body>

  <a onMouseOver = "setTempColor('white');" onMouseOut = "defaultColor()" onClick = "setDefaultColor('white');">Белый</a>

  <a onMouseOver = "setTempColor('yellow');" onMouseOut = "defaultColor()" onClick = "setDefaultColor('yellow');">Жёлтый</a>

  <a onMouseOver = "setTempColor('green');" onMouseOut = "defaultColor()" onClick = "setDefaultColor('green');">Зелёный</a>

</body>

</html>

На странице имеются три ссылки. Каждая из ссылок отвечает за разный цвет фона (допустим, белый, жёлтый и зелёный). Вначале фон белый. При наведении мыши на определённую ссылку цвет фона меняется. При отведении мыши цвет фона возвращается на цвет по умолчанию. При щелчке мыши по ссылке цвет фона сохраняется, как по умолчанию.

При возникновении какого-либо события на элементе страницы браузер автоматически создаёт объект, в котором содержится полезная информация о наступившем событии (например, свойство type указывает тип события).

Объект любого события является наследником общего объекта-конструктора Event. Это означает, что все объекты обладают общими свойствами и методами, которые наследуются от Event, таблица 19.3. Таких свойств немного и используются из них лишь некоторые.

Таблица 19.3 – Свойства наследников объекта Event

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| **1** | **2** |
| bubbles | Возвращает логическое значение, которое указывает, является ли событие всплывающим. |
| cancelable | Возвращает логическое значение, указывающее, является ли событие отменяемым. |
| currentTarget | Возвращает целевой объект события, обрабатываемого в настоящее время. |

|  |  |
| --- | --- |
| **1** | **2** |
| defaultPrevented | Получает значение, указывающее, следует ли отменить действи по умолчанию. True – действия по умолчанию должны быть отменены, false – действия по умолчанию разрешаются. |
| eventPhase | Указывает, какая фаза события, в настоящее время проверяется. |
| target | Ссылается на элемент, который является целевым объектом данного события. |
| timeStamp | Получает время в миллисекундах, когда произошло событие. |
| type | Имя события (без учета регистра). |
| isTrusted | Получает значение, указывающее, было ли событие инициировано в браузере (события браузера и пользовательские события) или в сценарии. |

<html>

<head>

<title>События</title>

</head>

<body>

<input type="button" value="Кнопка">

</body>

</html>

<script>

var button = document.body.firstElementChild;

button.onclick = function(event) {

alert(event.type); /\* 'click' \*/

}

</script>

Чтобы в любой момент выполнения события обратиться к целевому элементу, используется свойство event.target. Оно всегда содержит ссылку на тот элемент, на котором было вызвано событие.

Значение target отличается от currentTarget, если обработчик выполняется на стадии перехвата или всплытия.

Свойство target широко используется. Когда имеется несколько однотипных элементов (например, элементы меню), устанавливается только один обработчик на родительский элемент. А с помощью свойства target определяется элемент, для которого должно выполниться событие.

<html>

<head>

<title>События</title>

</head>

<body>

<ul>

<li>lorem</li>

<li>ipsum</li>

<li>dolor</li>

<li>sit</li>

</ul>

</body>

</html>

<script>

var uList = document.body.firstElementChild;

uList.onclick = function (event) {

this.removeChild(event.target);

}

</script>

В данном примере клик на любом элементе списка удаляет его. При этом скрипт использует всего один обработчик. Особенно это полезно, когда список элементов заведомо не известен, а создаётся динамически.

Свойство event.eventPhase указывает, какая [фаза события](http://webgent.ru/javascript/event-stage) проходит в настоящее время. Оно имеет числовое значение, которое соответствует стадии обработки:

- 0 – никакое событие не обрабатывается;

- 1 – стадия перехвата;

- 2 – стадия цели;

- 3 – стадия всплытия.

Точное время, когда произошло событие, хранится в свойстве event.timeStamp.

Время представлено как количество миллисекунд, прошедших с начала эпохи Unix (1 января 1970 года по Гринвичу).

В свойстве event.type содержится название события (click, keypress и т.п.).

В свойстве event.isTrusted содержится информация о том, кем вызвано событие. Свойство имеет булевый тип:

- true – событие вызвано действием пользователя или браузером;

- false – событие вызвано скриптом, например, с помощью метода [dispatchEvent](http://webgent.ru/javascript/event-dispatch#dispatchEvent).

Информация о том, будет ли событие всплывать, хранится в свойстве event.bubbles. Свойство имеет булевый тип:

- true – событие будет всплывать;

- false – событие не будет всплывать.

Свойство event.cancelable содержит информацию о том, можно ли отменить стандартные действия браузера, выполняемые по умолчанию. Свойство имеет булевый тип:

- true – действия браузера можно отменить;

- false – действия браузера отменить нельзя.

Свойство event.defaultPrevented содержит информацию о том, было ли отменено стандартное действие браузера, выполняемое по умолчанию. Свойство имеет булевый тип:

- true – действие браузера было отменено;

- false – действие браузера не было отменено.

События на странице могут запускаться не только браузером или посетителем сайта, но и самим скриптом. Для этого предварительно необходимо создать объект события, который будет передаваться обработчику. Делается это с помощью объекта-конструктора Event.

New Event(имя\_события[, свойства])

где имя\_события – указывается название события в кавычках. Можно использовать не только стандартные события, но и придумывать свои собственные (тогда обработчик назначается с помощью метода addEventListener());

свойства – необязательный аргумент. Должен быть объектом с двумя свойствами:

bubbles – всплывать ли событию (true/false);

cancelable – возможность отмены действий браузера по умолчанию (true/false).

Данные свойства будут унаследованы объектом события.

По умолчанию: {bubbles: false, cancelable: false}.

Чтобы сгенерировать событие на элементе, используется метод dispatchEvent().

element.dispatchEvent(объект\_события)

где объект\_события – ссылка на специально созданный объект события. Данный объект будет передан обработчикам события.

Вызов dispatchEvent() приводит к выполнению обработчиков события в соответствующем порядке. Правила обработки события браузером (погружение и всплытие) сохраняются. Но для всплытия события необходимо указать свойство bubbles: true при создании объекта события.

<html>

<head>

<title>События</title>

</head>

<body onclick="alert('Событие click всплывает.');">

<p>Параграф</p>

</body>

</html>

<script>

var par = document.body.firstElementChild;

var event = new Event('click', {bubbles: true});

par.dispatchEvent(event); /\* вызов 'click' на параграфе \*/

</script>

В примере вызывается событие click на параграфе и оно всплывает.

Метод dispatchEvent() возвращает значение false, если хотя бы один из выполняемых обработчиков вызывает event.preventDefault() (при создании объекта события должно быть указано свойство cancelable: true). В остальных случаях возвращается true.

<html>

<head>

<title>События</title>

</head>

<body>

<input type="button" onclick="event.preventDefault();" value="Кнопка">

</body>

</html>

<script>

var button = document.body.firstElementChild;

var event\_1 = new Event('click', {cancelable: true});

alert(button.dispatchEvent(event\_1)); /\* false, действие отменено \*/

var event\_2 = new Event('click', {cancelable: false});

alert(button.dispatchEvent(event\_2)); /\* true, действие не отменено \*/

</script>

Если вызывается стандартное событие (click и т.п.), тогда данная возможность позволяет узнать, было ли отменено действие браузера по умолчанию.

В случае с нестандартными событиями вызов event.preventDefault() позволяет получить дополнительную информацию от обработчика события.

Пример:

элементу <input> назначен обработчик нестандартного события check\_value для проверки введённого значения:

<html>

<head>

<title>События</title>

</head>

<body>

<input type="text" oninput="mess(); this.value='';" placeholder="Введите цифру 5">

</body>

</html>

<script>

var input = document.body.firstElementChild;

input.addEventListener('check\_value', function(event) {

if (this.value !== '5') {

event.preventDefault();

}

});

function mess() {

var check\_event = new Event('check\_value', {cancelable: true});

if (input.dispatchEvent(check\_event) == true) {

alert('Поздравляем. Вы ввели цифру 5.');

} else {

alert('Введите цифру 5.');

}

}

</script>

При вводе любого символа в текстовое поле запускается функция mess(). В ней генерируется событие check\_value для проверки значения. В зависимости от результата выполнения обработчика события check\_value выводится нужное сообщение.

Генерация событий в скрипте используется редко. Это вполне логично, ведь обработчики событий должны реагировать на действия, совершённые пользователем или браузером. А для выполнения команд, не связанных с этими действиями, можно использовать простые функции.

**5. Порядок выполнения работы**

1. Создайте ненумерованный список. С использованием addEventListener добавьте новые пункты меню в конец списка.

2. Добавьте на страницу таблицу, содержащую информацию об учащихся. Под таблицей создайте форму, с помощью которой можно будет добавить нового учащегося в таблицу. При клике на любую ячейку появлялся всплывающее окно prompt, с помощью которого можно изменить текст ячейки. Задание необходимо реализовать с помощью делегирования.

3. Выполните задание по варианту. Варианты представлены в таблице 19.4.

Таблица 19.4 – Варианты заданий

|  |  |
| --- | --- |
| Вариант | Задания |
| 1 | 1. Создайте HTML-страницу, на которой будет размещен перечень сообщений с возможностью закрыть сообщение. По нажатию на «Крестик» (или кнопку «Закрыть»), сообщение должно исчезать с экрана. Для реализации необходимо использовать делегирование событий (один обработчик для всего). |
| 2 | 1. Создайте на HTML-странице многоуровневый вложенный список. По клику на заголовок списка необходимо скрывать-показывать детей |
| 3 | 1. Создайте HTML-страницу, на которой будет размещено три кнопки. По нажатию на кнопки должны показываться соответствующие сообщения. Для реализации необходимо использовать делегирование событий (один обработчик для всего). |
| 4 | 1. Создайте на HTML-странице несколько таблиц с заголовками. По клику на заголовок таблицы необходимо скрывать-показывать данную таблицу |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой объект Event?

2. Перечислите типы событий.

3. Какие элементы могут генерировать события?

4. Опишите способы назначения обработчиков событий.

5. На какие стадии делится обработка события? В чем их суть?

6. Перечислите методы объекта Event.

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 20**

## **Тема работы: «Проверка корректности информации, введенной пользователем. Обработка данных формы»**

**1. Цель работы**

Формирование умений проверки корректности информации, введенной пользователем в элементы формы, обработки данных форм средствами JavaScript.

**2. Задание**

Реализовать проверку правильности заполнения полей формы средствами JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

При работе с формами в JavaScript используется объект Form, который также является очередным свойством объекта Document. forms содержит коллекцию всех форм на веб-странице.

Доступ к форме осуществляется по её имени или по индексу.

document.forms.my – форма с именем 'my'

document.forms[0] – первая форма в документе

Общий вид получения доступа к полю формы по ее имени следующий:

document.имя\_формы.имя\_поля.value;

То есть, сначала обращаемся к объекту Document, затем к его свойству Form (через имя формы), потом к имени поля данной формы, и, наконец, к значению поля.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search"></form>

</body>

</html>

<script>

var form = document.forms.search; // или

var form = document.forms['search']; // или

var form = document.forms[0];

</script>

Доступ к элементу формы осуществляется аналогично свойству document.forms. Каждая форма имеет свойство elements, в котором содержится список всех элементов формы, как показано на рисунке 20.1.

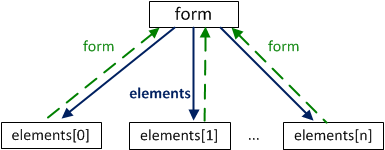


Рисунок 20.1 – Свойство elements объекта form

Пример:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search">

<input name="query">

</form>

</body>

</html>

<script>

var form = document.forms.search;

var input = form.elements.query; // или

var input = form.elements['query']; // или

var input = form.elements[0];

</script>

Если в форме используется несколько элементов с одинаковым именем, то обращение к элементу по имени вернёт список всех элементов с этим именем.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search">

<input type="radio" name="query" value="1">

<input type="radio" name="query" value="2">

</form>

</body>

</html>

<script>

var form = document.forms.search;

var input = form.elements.query;

alert(input[0].value); /\* 1 \*/

alert(input[1].value); /\* 2 \*/

</script>

Свойство elements также есть у элементов <fieldset>. Вот пример:

<body>

<form>

<fieldset name="set">

<legend>fieldset</legend>

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

</fieldset>

</form>

<script>

var form = document.forms[0];

alert( form.elements.text ); // INPUT

alert( form.elements.set.elements.text ); // INPUT

</script>

</body>

По элементу можно получить его форму, используя свойство element.form:

<body>

<form>

<input type="text" name="surname">

</form>

<script>

var form = document.forms[0];

var elem = form.elements.surname;

alert (form.elements.surname); // [object HTMLInputElement]

alert(elem.form == form); // true

</script>

</body>

Свойство value содержит текущее введённое пользователем значение. До начала ввода пользователем нового значения данное свойство определяется атрибутом value, если он указан:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="user\_form" onsubmit="showValue(); return false" autocomplete="off">

<input type="text" placeholder="some text" name="user\_input">

<input type="submit" value="Показать VALUE">

</form>

</body>

</html>

<script>

var form = document.forms.user\_form;

var input = form.elements.user\_input;

function showValue() {

alert(input.value); /\* всегда выводит текущее значение \*/

}

</script>

Если ничего не вводить в текстовое поле, то при отправке формы выведется пустая строка, так как атрибут value не задан.

У элементов с типом checkbox и radio свойство value не может изменяться пользователем. При их использовании необходимо знать, выбраны ли они пользователем. Для этого используется логическое свойство checked. Оно отражает текущее состояние элемента (отмечен или не отмечен) и позволяет его изменить:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search" autocomplete="off">

<input type="radio" name="query" value="1" checked>

<input type="radio" name="query" value="2" >

</form>

</body>

</html>

<script>

var form = document.forms.search;

var input = form.elements.query;

if (input[0].checked) {

var a=input[0].value

alert(a);}

else {

alert(input[1].value); }

Поле для отправки файла (тип file) имеет свойство files. Это объект, содержащий список всех вставленных файлов.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="upload" onsubmit="showFile(); return false">

<input type="file" name="user\_file">

<input type="submit" value="Отправить">

</form>

</body>

</html>

<script>

function showFile() {

var form = document.forms.upload;

var files = form.elements.user\_file.files;

alert(files[0].name); /\* покажет имя файла \*/

alert(files[0].lastModifiedDate); /\* покажет время последнего изменения файла \*/

}

</script>

С помощью методов focus() и blur() поле ввода можно сделать активным или неактивным. Например, можно автоматически сделать активным то поле, в котором допущена ошибка, и т.п.:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search">

<label>Введите что-нибудь: <input name="text\_in" oninput="show()"></label>

<br>

<label>Вы ввели: <input name="text\_out"></label>

</form>

</body>

</html>

<script>

var form = document.forms.search;

var text\_in = form.elements.text\_in;

var text\_out = form.elements.text\_out;

text\_in.focus(); /\* автофокус на поле \*/

function show() {

text\_in.blur(); /\* снимает фокус с поля \*/

text\_out.value = text\_in.value;

text\_in.value = '';

}

</script>

Метод select() используется для выделения введённого пользователем текста. При этом поле становится активным:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search">

<label>Введите что-нибудь: <input name="text\_in" oninput="show()"></label>

<br>

<label>Вы ввели: <input name="text\_out"></label>

</form>

</body>

</html>

<script>

var form = document.forms.search;

var text\_in = form.elements.text\_in;

var text\_out = form.elements.text\_out;

text\_in.focus();

function show() {

text\_out.value = text\_in.value;

text\_out.select(); /\* выделяет содержимое \*/

}

</script>

Элемент <textarea> аналогичен полю <input>. Для него так же доступны методы focus(), blur(), select(). Их значения и действия полностью совпадают с теми же методами элемента <input>.

Элемент <textarea> не имеет атрибута value, но свойство value всё равно имеет. Его значением является содержимое тега:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="test" autocomplete="off">

<textarea name="user\_textarea">Текст</textarea>

</form>

</body>

</html>

<script>

var form = document.forms.test;

var textarea = form.elements.user\_textarea;

alert(textarea.value); /\* Текст \*/

</script>

Элемент <select> не имеет атрибута value. Однако, свойство value всё равно присутствует. Его значение берётся из той опции (из её свойства value), которая выбрана в данный момент. Если выбрано несколько опций, то значение берётся из первой.

У элементов option также есть особые свойства, которые могут оказаться полезными:

- selected – выбрана ли опция;

- index – номер опции в списке селекта;

- text – текстовое содержимое опции (то, что видит посетитель).

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="test" autocomplete="off">

<select name="user\_select" multiple>

<option value="1" selected>Число 1</option>

<option value="2" selected>Число 2</option>

</select>

</form>

</body>

</html>

<script>

var form = document.forms.test;

var select = form.elements.user\_select;

alert(select.value); /\* 1 \*/

</script>

Селект в JavaScript можно установить двумя путями: поставив значение select.value, либо установив свойство select.selectedIndex в номер нужной опции.

Свойство selectedIndex содержит индекс той опции, которая выбрана в данный момент (если используется атрибут multiple, то свойство содержит индекс первой из выбранных опций). Изменяя данное свойство, можно изменить текущее значение в поле<select>. Нумерация начинается с нуля. Значение”-1” устанавливается для очистки поля.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="test" autocomplete="off">

<select name="user\_select">

<option value="1">Число 1</option>

<option value="2" selected>Число 2</option>

<option value="3">Число 3</option>

</select>

</form>

</body>

</html>

<script>

var form = document.forms.test;

var select = form.elements.user\_select;

alert(select.selectedIndex); /\* 1 \*/

select.selectedIndex = -1;

alert(select.value); /\* пустая строка \*/

</script>

Свойство options содержит список всех тегов <option> внутри данного элемента <select>.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="user\_form" onsubmit="showNum(); return false" autocomplete="off">

<select name="user\_select">

<option value="1">Число 1</option>

<option value="2">Число 2</option>

<option value="3">Число 3</option>

<option value="4">Число 4</option>

<option value="5">Число 5</option>

<option value="6">Число 6</option>

</select>

<input type="submit" value="Отправить">

</form>

</body>

</html>

<script>

var form = document.forms.user\_form;

var select = form.elements.user\_select;

function showNum() {

alert('Доступно опций: ' + select.options.length);

}

</script>

Список элементов-опций доступен через select.options. Свойство selectedOptions содержит список тех тегов <option>, которые в данный момент выбраны.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="user\_form" onsubmit="showNum(); return false" autocomplete="off">

<select name="user\_select" multiple>

<option value="1">Число 1</option>

<option value="2">Число 2</option>

<option value="3">Число 3</option>

<option value="4">Число 4</option>

<option value="5">Число 5</option>

<option value="6">Число 6</option>

</select>

<input type="submit" value="Отправить">

</form>

</body>

</html>

<script>

var form = document.forms.user\_form;

var select = form.elements.user\_select;

function showNum() {

alert('Выбрано опций: ' + select.selectedOptions.length);

}

</script>

Если select допускает множественный выбор (атрибут multiple), то значения можно получить/установить, сделав цикл по select.options. При этом выбранные опции будут иметь свойство option.selected = true:

<form name="form">

<select name="genre" multiple>

<option value="blues" selected>Мягкий блюз</option>

<option value="rock" selected>Жёсткий рок</option>

<option value="classic">Классика</option>

</select>

</form>

<script>

var form = document.forms[0];

var select = form.elements.genre;

for (var i = 0; i < select.options.length; i++) {

var option = select.options[i];

if(option.selected) {

alert( option.value );

}

}

</script>

Элементы <option>имеют булевое свойство selected. Оно характеризует состояние опции: выбрана или не выбрана.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="test" autocomplete="off">

<select name="user\_select">

<option value="1">Число 1</option>

<option value="2" selected>Число 2</option>

</select>

</form>

</body>

</html>

<script>

var form = document.forms.test;

var select = form.elements.user\_select;

alert(select.options[0].selected); /\* false \*/

alert(select.options[1].selected); /\* true \*/

</script>

Элемент label – один из самых важных в формах. Клик на label засчитывается как фокусировка или клик на элементе формы, к которому он относится. Это позволяет посетителям кликать на большой красивой метке, а не на маленьком квадратике input type=checkbox (radio).

Имеется два способа показать, какой элемент относится к label:

1) присвоить метке атрибут for, равный id соответствующего input:

<table>

<tr>

<td>

<label for="agree">Согласен с правилами</label>

</td>

<td>

<input id="agree" type="checkbox">

</td>

</tr>

<tr>

<td>

<label for="not-a-robot">Я не робот</label>

</td>

<td>

<input id="not-a-robot" type="checkbox">

</td>

</tr>

</table>

2) завернуть элемент в label. В этом случае можно обойтись без дополнительных атрибутов:

<label>Кликни меня <input type="checkbox"></label>

В стандарте the option element есть короткий синтаксис для создания элемента с тегом option:

option = new Option(text, value, defaultSelected, selected);

где text – содержимое;

value – значение;

defaultSelected и selected необходимо поставить в true, чтобы сделать элемент выбранным.

Описанный способ можно использовать вместо document.createElement('option'), например:

var option = new Option("Текст", "value");

// создаст <option value="value">Текст</option>

Такой же элемент, но выбранный:

var option = new Option("Текст", "value", true, true);

Форма – это самый популярный способ общения пользователя с сайтом, и очень важно, чтобы данные, ведённые пользователем были корректными. И чтобы форма была отправлена с верными данными необходимо её перед этим проверить.

Создадим форму, которую будем проверять:

<form name = 'form1' action = 'handler.php' method = 'post'>

  Ваше имя:

  <input type = 'text' name = 'firstname' />

  <br />

  Ваш пароль:

  <input type = 'password' name = 'pass' />

  <br />

  Ваш пол:

  <input type = 'radio' name = 'sex' value = 'male' />

  <input type = 'radio' name = 'sex' value = 'female' />

  <br />

  Выберите число:

  <select name = 'number'>

    <option value = '1'>1</option>

    <option value = '2'>2</option>

    <option value = '3'>3</option>

  </select>

  <br />

  Ваше сообщение:

  <textarea name = 'message'></textarea>

  <br />

  Согласитесь с нашими правилами:

  <input type = 'checkbox' name = 'rules' />

  <br />

  <input type = 'hidden' value = 'hidefield' />

  Загрузите файл:

  <input type = 'file' name = 'fileupload' />

  <br />

  <input type = 'submit' value = 'Отправить' name = 'sub' />

  <br />

  <input type = 'button' value = 'Просто кнопка' name = 'but' />

</form>

Выведем все поля, которые подчиняются общему виду:

var form = document.form1;

document.write(form.firstname.value + "<br />");

document.write(form.pass.value + "<br />");

document.write(form.number.value + "<br />");

document.write(form.message.value + "<br />");

document.write(form.rules.value + "<br />");

document.write(form.hidefield.value + "<br />");

document.write(form.fileupload.value + "<br />");

document.write(form.sub.value + "<br />");

document.write(form.but.value + "<br />");

Это все поля, которые имеют свойство value, и к которым можно получить доступ таким образом.

Для того, чтобы получить доступ к значению элемента radio, необходимо:

var sex = (document.form1.sex[0].checked)?

document.form1.sex[0].value : document.form1.sex[1].value;

В массиве sex находятся два элемента radio. Индексы у них 0 и 1. В данном скрипте мы проверяем, если у нас первый элемент включён (checked), то присваиваем значение первого элемента, иначе присваиваем значение второго элемента.

Можно было бы написать и так:

var sex;

if (document.form1.sex[0].checked) sex = document.form1.sex[0].value;

else sex = document.form1.sex[1].value;

Далее в тег <form> необходимо добавить атрибут "onSubmit" со значением "return check();". Данный атрибут будет делать следующее: перед отправкой формы вызывать функцию, которая должна будет вернуть либо true, либо false. Если она вернёт true, то форма отправится на сервер, а если false, то форма не будет отправлена.

Теперь можно создать функцию check(), которая должна, во-первых, проверять полностью форму, сообщать пользователю об ошибках, а также возвращать true (если форма полностью правильная), либо false (если форма содержит ошибки).

function check(form) {

  var firstname = form.firstname.value;

  var pass = form.pass.value;

  var message = form.message.value;

  var rules = form.rules.value;

  var file = form.fileupload.value;

  var bad = "";

  if (firstname.length < 3)

     bad += "Имя слишком короткое" + "\n";

  if (firstname.length > 32)

    bad += "Имя слишком длинное" + "\n";

  if (pass.length < 3)

    bad += "Пароль слишком короткий" + "\n";

  if (pass.length > 32)

    bad += "Пароль слишком длинный" + "\n";

  if (message.length < 3)

    bad += "Сообщение слишком короткое" + "\n";

  if (rules != "on")

    bad += "Вы не согласились с правилами" + "\n";

  if (file.length == 0)

    bad += "Вы не выбрали файл для загрузки" + "\n";

  if (bad != "") {

    bad = "Неверно заполнена форма:" + "\n" + bad;

    alert(bad);

    return false;

  }

  return true;

}

В данном скрипте мы сначала получаем все данные, нуждающиеся в проверке, и записываем их в переменные. Затем создаём переменную bad, в которую записываем все некорректные данные. Затем идёт целый набор IF, которые проверяют поля в форме и записывают все ошибки в переменную bad. Затем, если переменная bad не пустая (то есть были ошибки), то выводим окно (alert()) с ошибками. И возвращает false, чтобы форма не была отправлена. Если переменная bad пустая, то дальше просто возвращаем true, чтобы форма была отправлена уже на обработку в "handler.php".

Динамическая проверка формы на JavaScript может выглядеть следующим образом:

<form name="myform" action="#" method="post">

  <p>Логин:

<input type="text" name="login" onkeyup="check(this.value)" />

 <span id="e\_login" style="display: none; color: #c00;">Логин введён неправильно</span>

</p>

</form>

Ключевой момент – это обработчик события onkeyup. Событие keyup отвечает за отпускание клавиши. Обратите внимание, что очень часто ставят onkeypress или onkeydown – это неправильно, так как будет происходить отставание на 1 шаг.

JavaScript-код тоже очень простой:

<script type="text/javascript">

  function check(login) {

    if (login.length < 3) document.getElementById("e\_login").style.display = "inline";

    else document.getElementById("e\_login").style.display = "none";

  }

</script>

В функции check() мы проверяем длину логина, и если она меньше 3-х символов, то мы выводим ошибку (делая соответсвующий элемент видимым), иначе прячем это сообщение.

Аналогично, можно проверять абсолютно любые формы.

Однако, перед отправкой надо целиком проверить ещё раз всю форму с помощью onsubmit у тега form.

**5. Порядок выполнения работы**

1. Создайте тест на любую тематику с числовыми значениями. При нажатии на кнопку «готово» функция должна проверять, все ли поля заполнены. Если заполнены все поля, то должен быть выдан результат.

2. Выполните валидацию форм в соответствии с вариантом. Варианты представлены в таблице 20.1.

Таблица 20.1 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задания** |
| **1** | **2** |
| 1 | Регистрация на сайте знакомств |

|  |  |
| --- | --- |
| **1** | **2** |
| 2 | Поиск и фильтрация техники в магазине |
| 3 | Поиск и фильтрация на сайте вакансий |
| 4 | Добавление нового товара |
| 5 | Заказ и покупка билета на самолет |
| 6 | Покупка билета на концерт |
| 7 | Добавление блюда меню |
| 8 | Поиск и фильтрация людей на сайте знакомств |
| 9 | Запрос на выдачу кредита |
| 10 | Добавление информации о рейсе |
| 11 | Добавление информации о мероприятии |
| 12 | Поиск и фильтрация фильмов |
| 13 | Поиск и фильтрация вопросов на форуме программистов |
| 14 | Добавление пациента |
| 15 | Заказ ремонта автомобиля |
| 16 | Выбор и оплата услуги ЕРИП |
| 17 | Поиска и фильтрации книг |
| 18 | Поиск и бронирование тура |
| 19 | Поиск и фильтрация компьютерной техники |
| 20 | Поиск и фильтрация автомобилей |
| 21 | Заявка на замену паспорта |
| 22 | Начисление стипендии учащимся |
| 23 | Добавление нового сотрудника |
| 24 | Погашение штрафов ГАИ |
| 25 | Поиск и бронирование гостиницы |
| 26 | Поиск и фильтрация игр |
| 27 | Поиска и фильтрации врачей в поликлинике |
| 28 | Добавления денежного перевода |
| 29 | Поиск и фильтрация ресторанов |
| 30 | Регистрация фирмы |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Для чего в JavaScript используется объект Form?

2. Опишите способы организации доступа к форме.

3. Каким образом осуществляется доступ к элементам формы?

4. В чем заключается валидация формы?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 21**

## **Тема работы: «Применение объектно-ориентированного подхода в программировании на языке JavaScript»**

**1. Цель работы**

Формирование умений применения объектно-ориентированного подхода в программировании на языке JavaScript.

**2. Задание**

Реализовать создание классов и объектов с использованием объектно-ориентированного подхода в JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Объект в JavaScript – это просто коллекция пар ключ-значение. Однако, в JavaScript нет концепции класса. К примеру, объект с свойствами {name: Linda, age: 21} не является экземпляром какого-либо класса или класса Object. И Object, и Linda являются экземплярами самих себя. Они определяются непосредственно собственным поведением. Тут нет слоя мета-данных (т.е. классов), которые говорили бы этим объектам как нужно себя вести.

Объекты – это просто пары уникальных ключей с соответствующими значениями – такие пары называются свойства. К примеру, вы хотите описать несколько аспектов своего старого друга (назовём его Мишей, он же Mikhail), таких как возраст, имя и пол. Пример представлен на рисунке 21.1

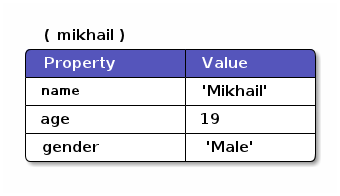


Рисунок 21.1 – Объект Mikhail

Объект в JavaScript создаётся с помощью функции Object.create. Эта функция из родителя и опционального набора свойств создаёт новую сущность. Пока что мы не будем беспокоиться о параметрах.

Пустой объект – это объект без родителя, без свойств. Посмотрим на синтакс создания такого объекта в JavaScript:

var mikhail = Object.create(null)

Функция Object.defineProperty создаёт новое свойство, если свойство с данным ключём ранее не существовало (в противном случае произойдёт обновление семантики и значения существующего свойства).

Например, у нас уже есть объект, но у него пока нет свойств – мы должны исправить эту ситуацию для описания нашего объекта Mikhail.

Свойства в JavaScript являются динамическими. Это означает, что мы их можем создавать или удалять в любое время. Свойства уникальны в том смысле, что ключ свойства внутри объекта соответствует ровно одному значению.

Создадим новые свойства через функцию Object.defineProperty, которая в качестве аргументов использует объект, имя свойства для создания и дескриптор, описывающий семантику свойства.

Object.defineProperty(mikhail, 'name', { value: 'Mikhail'

, writable: true

, configurable: true

, enumerable: true })

Object.defineProperty(mikhail, 'age', { value: 19

, writable: true

, configurable: true

, enumerable: true })

Object.defineProperty(mikhail, 'gender', { value: 'Male'

, writable: true

, configurable: true

, enumerable: true })

Также можно использовать Object.defineProperties когда необходимо добавить больше одного свойства в объект:

Object.defineProperties(mikhail, { name: { value: 'Mikhail'

, writable: true

, configurable: true

, enumerable: true }

, age: { value: 19

, writable: true

, configurable: true

, enumerable: true }

, gender: { value: 'Male'

, writable: true

, configurable: true

, enumerable: true }})

Очевидно, что оба вызова аналогичны, они вполне конфигурируемы, но не предназначены для конечного пользователя кода. Лучше создать уровень абстракции над ними.

Маленькие объекты, которые содержат в себе семантику, называются дескрипторами (мы их использовали при вызове Object.defineProperty). Дескрипторы бывают одного из двух типов — дескрипторы данных и дескрипторы доступа.

Оба типа дескрипторов содержат флаги, которые определяют, как свойство будет рассматриваться языком. Если флаг не установлен, то его значение по умолчанию false (к сожалению, это не всегда хорошее значение по умолчанию, что влечёт возрастание объёма описания дескрипторов).

Рассмотрим некоторые флаги:

writable – значение свойства может быть изменено, используется только для дескрипторов данных;

configurable – тип свойства может быть изменён или свойство может быть удалено;

enumerable – свойство используется в общем перечислении.

value – значение свойства.

Дескрипторы данных таковы, что определяют конкретное значение, которое соответствует дополнительному value-параметру, описывающему конкретные данные, привязанные к свойству.

Дескрипторы доступа определяют доступ к конкретному значению через getter-ы и setter-ы функций. Если не установлены, то по умолчанию равны undefined:

get() – функция вызывается без аргументов, когда происходит запрос к значению свойства;

set(new\_value) – функция вызывается с аргументом – новым значением для свойства, когда пользователь пытается модифицировать значение свойства.

Очень просто получить значение, хранящиеся в заданном свойстве – синтаксис очень похож на создание свойства с той лишь разницей, что в нём нет присваивания.

Например, если мы хотим узнать возраст Миши, то мы напишем:

mikhail['age']

// => 19

Но если мы попробуем получить значение свойства, которого не существует в нашем объекте, то мы получим undefined:

mikhail['address']

// => undefined

Для удаления свойства из объекта в JavaSсript предусмотрен оператор delete. К примеру, если вы хотите удалить свойство gender из нашего объекта mikhail:

delete mikhail['gender']

// => true

mikhail['gender']

// => undefined

Оператор delete вернёт true, если свойство удалено, и falseв противном случае.

Getter-ы и setter-ы обычно используются в классических объектно-ориентированных языках для обеспечения инкапсуляции. Но, с любой точки зрения, они позволяют обеспечить proxy для запросов на чтение и запись свойств. Например, у нас были отдельные слоты для имени и фамилии, но мы хотим иметь удобный способ читать и устанавливать их.

Для начала, создадим имя и фамилию нашего друга, описав соответствующие свойства:

Object.defineProperty(mikhail, 'first\_name', { value: 'Mikhail'

, writable: true })

Object.defineProperty(mikhail, 'last\_name', { value: 'Weiß'

, writable: true })

Затем мы опишем общий способ получения и установки сразу двух свойств за один раз — назовём их объединение name:

// () → String

// Returns the full name of object.

function get\_full\_name() {

return this.first\_name + ' ' + this.last\_name

}

// (new\_name:String) → undefined

// Sets the name components of the object, from a full name.

function set\_full\_name(new\_name) { var names

names = new\_name.trim().split(/\s+/)

this.first\_name = names[⁣'0'] || ''

this.last\_name = names['1'] || ''

}

Object.defineProperty(mikhail, 'name', { get: get\_full\_name

, set: set\_full\_name

, configurable: true

, enumerable: true })

Теперь, каждый раз, когда мы попытаемся узнать значение свойства name нашего друга на самом деле вызовется функция get\_full\_name:

mikhail.name

// => 'Mikhail Weiß'

mikhail.first\_name

// => 'Mikhail'

mikhail.last\_name

// => 'Weiß'

mikhail.last\_name = 'White'

mikhail.name

// => 'Mikhail White'

Также можно установить name объекта, обратившись к соответствующему свойству, но на самом деле вызов set\_full\_name выполнит всю грязную работу:

mikhail.name = 'Michael White'

mikhail.name

// => 'Michael White'

mikhail.first\_name

// => 'Michael'

mikhail.last\_name

// => 'White'

Есть сценарии, в которых действительно удобно так делать, но стоит помнить, что такой механизм работает [очень медленно](http://jsperf.com/getter-setter/8).

Кроме того, следует учитывать что getter-ы и setter-ы обычно используются в других языках для инкапсуляции, а в ECMAScript 5 вы всё ещё не можете так делать – все свойства объекта являются публичными.

Ввиду того, что свойства являются динамическими, JavaScript обеспечивает функционал по проверке набора свойств объекта. Существует два способа перечисления всех свойств объекта, зависящих от того, какой вид свойств вас интересует.

Первый способ заключается в вызове функции Object.getOwnPropertyNames, которая вернёт Array, содержащий имена всех свойств, установленных для данного объекта – будем называть эти свойства собственными. Например, посмотрим, что мы знаем о Мише:

Object.getOwnPropertyNames(mikhail)

// => [ 'name', 'age', 'gender', 'first\_name', 'last\_name' ]

Второй способ заключается в использовании Object.keys, который вернёт список собственных свойств, которые помечены флагом enumerable :

Object.keys(mikhail)

// => [ 'name', 'age', 'gender' ]

До сих пор объект Mikhail имел только слоты для хранения данных (ну, за исключением getter/setter для свойства name).

Опишем действие над данным объектом, просто установив функцию, как значение нашего свойства. К примеру, мы хотим, чтобы Миша мог приветствовать других людей:

// (person:String) → String

// Greets a random person

mikhail.greet = function(person) {

return this.name + ': Why, hello there, ' + person + '.'

}

После выставления значения свойства, можно использовать аналогичный способ для выставления конкретных данных, связанных с объектом. Таким образом, доступ к свойствам будет возвращать ссылку на функцию, хранящуюся в нём, которую можно вызвать:

mikhail.greet('you')

// => 'Michael White: Why, hello there, you.'

mikhail.greet('Kristin')

// => 'Michael White: Why, hello there, Kristin.

Существует четыре различных способа разрешения this в функции, зависящие от того, как функция вызывается: непосредственно, как метод, явно применяется, как конструктор.

Для следующих примеров примем:

// Returns the sum of the object's value with the given Number

function add(other, yet\_another) {

return this.value + other + (yet\_another || 0)

}

var one = { value: 1, add: add }

var two = { value: 2, add: add }

Если функция вызывается, как метод объекта, то this внутри функции ссылается на сам объект. Т.е. когда мы явно указываем какой объект выполняет действие, то объект и будет значением this в нашей функции.

Это произойдёт, когда мы вызовем mikhail.greet(). Эта запись говорит JavaScript-у, что мы хотим применить действие greet к объекту mikhail.

one.add(two.value) // this === one

// => 3

two.add(3) // this === two

// => 5

one['add'](two.value) // brackets are cool too

// => 3

Когда функция вызывается непосредственно, то this разрешается в глобальный объект движка (window в браузере)

add(two.value) // this === global

// => NaN

// The global object still has no `value' property, let's fix that.

value = 2

add(two.value) // this === global

// => 4

Функция может быть явно применена к любому объекту, несмотря на то, есть ли в объекте соответствующее свойство или нет. Эта функциональность достигается с помощью методов call или apply.

Различие между двумя методами заключается в параметрах, передаваемых в функцию и времени исполнения – apply работает примерно в 55 раз медленнее, чем непосредственный вызов, а вот call обычно не особо хуже.

Call ожидает объект, как первый параметр функции, за которым следуют обычные аргументы исходной функции:

add.call(two, 2, 2) // this === two

// => 6

add.call(window, 4) // this === global

// => 6

add.call(one, one.value) // this === one

// => 2

С другой стороны, apply позволяет описывать вторым параметром массив параметров исходной функции:

add.apply(two, [2, 2]) // equivalent to two.add(2, 2)

// => 6

add.apply(window, [ 4 ]) // equivalent to add(4)

// => 6

add.apply(one, [one.value]) // equivalent to one.add(one.value)

// => 2

Разрешение this в null или undefined зависит от семантики используемого движка. Обычно результат бывает таким же, как и применение функции к глобальному объекту. Но если движок работает в [strict mode](https://developer.mozilla.org/en/JavaScript/Strict_mode), то this будет разрешено как и ожидается с ровно в ту вещь, к которой применяется:

window.value = 2

add.call(undefined, 1) // this === window

// => 3

void function() {

"use strict"

add.call(undefined, 1) // this === undefined

// => NaN

// Since primitives can't hold properties.

}()

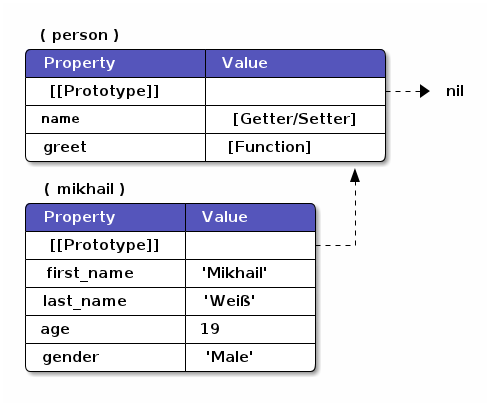
Наследование в JavaScript осуществляется через клонирование поведения объекта и расширение его специализированным поведением. Объект, поведение которого клонируют, называется прототипом.

Прототип с это обычный объект, который делится своим поведением с другими объектами с в этом случае он выступает в качестве родителя.

Концепт клонирования поведения не означает, что вы будете иметь две различные копии одной и той же функции или данных. На самом деле JavaScript реализует наследование через делегирование, т.е. все свойства хранятся в родителе, а доступ к ним расширен через ребёнка.

Как упоминалось ранее, родитель (или [[Prototype]]) объекта определяется вызовом Object.create с первым аргументом, ссылающимся на объект-родитель.

Вернёмся к примеру с Мишей. Выделим его имя и способность приветствовать людей в отдельный объект, который поделится с Мишей своим поведением. На рискнке 21.2 представлена наша модель.

  
Рисунок 21.2 – Модель объекта

Реализуем её на JavaScript:

var person = Object.create(null)

// Here we are reusing the previous getter/setter functions

Object.defineProperty(person, 'name', { get: get\_full\_name

, set: set\_full\_name

, configurable: true

, enumerable: true })

// And adding the `greet' function

person.greet = function (person) {

return this.name + ': Why, hello there, ' + person + '.'}

// Then we can share those behaviours with Mikhail

// By creating a new object that has it's [[Prototype]] property

// pointing to `person'.

var mikhail = Object.create(person)

mikhail.first\_name = 'Mikhail'

mikhail.last\_name = 'Weiß'

mikhail.age = 19

mikhail.gender = 'Male'

// And we can test whether things are actually working.

// First, `name' should be looked on `person'

mikhail.name

// => 'Mikhail Weiß'

// Setting `name' should trigger the setter

mikhail.name = 'Michael White'

// Such that `first\_name' and `last\_name' now reflect the

// previously name setting.

mikhail.first\_name

// => 'Michael'

mikhail.last\_name

// => 'White'

// `greet' is also inherited from `person'.

mikhail.greet('you')

// => 'Michael White: Why, hello there, you.'

// And just to be sure, we can check which properties actually

// belong to `mikhail'

Object.keys(mikhail)

Как вы видели в прошлом примере, ни одно из свойств, определённых в Person мы не определяли явно в Mikhail, но всё же смогли получить к ним доступ. Это произошло благодаря тому, что JavaScript реализует делегирование доступа к свойствам, т.е. свойство ищется через всех родителей объекта.

Эта цепь родителей определяется скрытым слотом в каждом объекте, который называется [⁣[Prototype]⁣]. Его нельзя изменить непосредственно, существует только один способ задать ему значение – при создании нового объекта.

Когда свойство запрашивается из объекта, движок сначала пытается получить свойство из целевого объекта. Если свойство не найдено, то рассматривается непосредственный родитель объекта, затем родитель родителя и т.д.

Это означает, что можно изменить поведение прототипа в середине программы, то автоматически изменится поведение всех объектов, которые были от него унаследованы. Например, пусть мы хотим изменить приветствие по умолчанию:

// (person:String) → String

// Greets the given person

person.greet = function(person) {

return this.name + ': Harro, ' + person + '.'

}

mikhail.greet('you')

// => 'Michael White: Harro, you.'

Самым очевидным решением является назначение \_\_proto\_\_ в конструкторе.

Например, если необходимо, чтобы у всех объектов, которые создаются new Rabbit, был прототип animal, это можно сделать так:

var animal = {

eats: true

};

function Rabbit(name) {

this.name = name;

this.\_\_proto\_\_ = animal;

}

var rabbit = new Rabbit("Кроль");

alert( rabbit.eats ); // true, из прототипа

Чтобы новым объектам автоматически ставить прототип, конструктору ставится свойство prototype.

При создании объекта через new, в его прототип \_\_proto\_\_ записывается ссылка из prototype функции-конструктора.

Например, код ниже полностью аналогичен предыдущему и является кроссбраузерным.

var animal = {

eats: true

};

function Rabbit(name) {

this.name = name;

}

Rabbit.prototype = animal;

var rabbit = new Rabbit("Кроль"); // rabbit.\_\_proto\_\_ == animal

alert( rabbit.eats ); // true

Установка Rabbit.prototype = animal буквально говорит интерпретатору следующее: "При создании объекта через new Rabbit запиши ему \_\_proto\_\_ = animal".

Свойство с именем prototype можно указать на любом объекте, но особый смысл оно имеет, лишь если назначено функции-конструктору.

Само по себе, без вызова оператора new, оно вообще ничего не делает, его единственное назначение – указывать \_\_proto\_\_ для новых объектов. Технически, в это свойство можно записать что угодно. Однако, при работе new, свойство prototype будет использовано лишь в том случае, если это объект. Примитивное значение, такое как число или строка, будет проигнорировано.

У каждой функции по умолчанию уже есть свойство prototype. Оно содержит объект такого вида:

function Rabbit() {}

Rabbit.prototype = {

constructor: Rabbit

};

В коде выше создание Rabbit.prototype происходило вручную, но ровно такой же – генерируется автоматически.

Проверим:

function Rabbit() {}

// в Rabbit.prototype есть одно свойство: constructor

alert( Object.getOwnPropertyNames(Rabbit.prototype) ); // constructor

// оно равно Rabbit

alert( Rabbit.prototype.constructor == Rabbit ); // true

Можно его использовать для создания объекта с тем же конструктором, что и данный:

function Rabbit(name) {

this.name = name;

alert( name );

}

var rabbit = new Rabbit("Кроль");

var rabbit2 = new rabbit.constructor("Крольчиха");

Эта возможность бывает полезна, когда, получив объект, мы не знаем в точности, какой у него был конструктор (например, сделан вне нашего кода), а нужно создать такой же.

JavaScript никак не использует свойство constructor. То есть, оно создаётся автоматически, а что с ним происходит дальше – это уже наша забота. В стандарте прописано только его создание.

В частности, при перезаписи Rabbit.prototype = { jumps: true } свойства constructor больше не будет.

Сам интерпретатор JavaScript его в служебных целях не требует, поэтому в работе объектов ничего не «сломается». Но если мы хотим, чтобы возможность получить конструктор, всё же, была, то можно при перезаписи гарантировать наличие constructor вручную:

Rabbit.prototype = {

jumps: true,

constructor: Rabbit

};

Либо можно поступить аккуратно и добавить свойства к встроенному prototype без его замены:

// сохранится встроенный constructor

Rabbit.prototype.jumps = true

**5. Порядок выполнения работы**

Выполните задание в соответствии с вариантом. Варианты представлены в таблице 21.1.

Таблица 21.1 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задание** |
| 1 | Реализуйте класс MyString, который будет иметь следующие методы: метод reverse(), который параметром принимает строку, а возвращает ее в перевернутом виде, метод ucFirst(), который параметром принимает строку, а возвращает эту же строку, сделав ее первую букву заглавной и метод ucWords, который принимает строку и делает заглавной первую букву каждого слова этой строки. |
| 2 | Реализуйте класс Student (Студент), который будет наследовать от класса User. Этот класс должен иметь следующие свойства: name (имя, наследуется от User), surname (фамилия, наследуется от User), year (год поступления в вуз). Класс дол-жен иметь метод getFullName() (наследуется от User), с помощью которого можно выве-сти одновременно имя и фамилию студента. Также класс должен иметь метод getCourse(), который будет выводить текущий курс студента (от 1 до 5). Курс вычисляется так: нужно от текущего года отнять год поступления в вуз. Текущий год получите самостоятельно. |
| 3 | Реализуйте класс Rectangle. У него должны быть следующие свойства: ширина width, высота height. Также у него должны быть следующие методы: получить ширину getWidth, установить ширину setWidth, получить высоту getHeight, установить высоту setHeight. |
| 4 | Реализуйте класс Worker (Работник), который будет иметь следующие свойства: name (имя), surname (фамилия), rate (ставка за день работы), days (количество отработанных дней). Также класс должен иметь метод getSalary(), который будет выводить зарплату работника. Зарплата - это произведение (умножение) ставки rate на количество отработанных дней days. |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой объект в JavaScript?

2. Перечислите способы создания объектов в JavaScript.

3. Для чего используется функция Object.defineProperty?

4. Что представляют собой getter-ы и setter-ы?

5. Что представляет собой прототип?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 22**

## **Тема работы: «Получение данных в формате JSON и их обработка на стороне клиента»**

**1. Цель работы**

Формирование умений обработки данных в формате JSON на стороне клиента.

**2. Задание**

Реализовать преобразования объекта JavaScript в JSON-строку и JSON-строки в объект JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

JSON (JavaScript Object Notation) – это текстовый формат обмена данными, основанный на мультипарадигменном (основан на подмножестве языка программирования JavaScript) языке программирования. Его основное назначение состоит в хранении и передаче структурированного потока информации.

JSON используется для хранения данных, подобных XML. Он основан на синтаксисе нотации объектов JavaScript. В JSON четыре основных синтаксических правила, зная которые легко с нуля написать объекты JSON:

– данные хранятся в парах ключ/значение.

“city”:”Chicago”

Это значение может быть числом, строкой, булевской переменной (true/false), массивом, объектом или псевдозначением null;

– для разделения данных используются запятые;

– объекты заключаются в фигурные скобки:

{“city”:”Chicago”, “state”:”Illinois”}

– массивы заключаются в квадратные скобки:

Var CitiesVisited = [{“city”:”Chicago”, “state”:”Illinois”, “year”:”2003”},

{“city”:”Portland”, “state”:”Oregon”, “year”:”2007”},

{“city”:”New York”, “state”:”New York”, “year”:”2013”}]

Можно создавать простые базы данных, размещая несколько объектов в одном массиве. Это позволит в программе использовать любую часть этого массива.

Пример объекта "CitiesVisited", содержащего массив городов (выше):

document.write (‘City visited: ’CitiesVisited[1].city);

// Output: City visited: Portland

Объекты в формате JSON похожи на обычные JavaScript-объекты, но отличаются от них более строгими требованиями к строкам – они должны быть именно в двойных кавычках.

Кроме того, в формате JSON не поддерживаются комментарии. Он предназначен только для передачи данных.

Основные методы для работы с JSON в JavaScript – это:

JSON.parse – читает объекты из строки в формате JSON;

JSON.stringify – превращает объекты в строку в формате JSON, используется, когда нужно из JavaScript передать данные по сети.

Вызов JSON.parse(str) превратит строку с данными в формате JSON в JavaScript-объект/массив/значение.

var numbers = "[0, 1, 2, 3]";

numbers = JSON.parse(numbers);

alert( numbers[1] ); // 1

var user = '{ "name": "Вася", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }';

user = JSON.parse(user);

alert( user.friends[1] ); // 1

Данные могут быть сколь угодно сложными, объекты и массивы могут включать в себя другие объекты и массивы. Главное, чтобы они соответствовали формату.

Метод JSON.parse поддерживает и более сложные алгоритмы разбора.

Например, объект с данными события event

// title: название события, date: дата события

var str = '{"title":"Конференция","date":"2014-11-30T12:00:00.000Z"}';

var str = '{"title":"Конференция","date":"2014-11-30T12:00:00.000Z"}';

var event = JSON.parse(str);

alert( event.date.getDate() ); // ошибка!

Дело в том, что значением event.date является строка, а отнюдь не объект Date. Метод JSON.parse не знает, что нужно превратить строку именно в дату.

Для интеллектуального восстановления из строки у JSON.parse(str, reviver) есть второй параметр reviver, который является функцией function(key, value).

Если она указана, то в процессе чтения объекта из строки JSON.parse передаёт ей по очереди все создаваемые пары ключ-значение и может возвратить либо преобразованное значение, либо undefined, если его нужно пропустить.

В данном случае мы можем создать правило, что ключ date всегда означает дату:

// дата в строке - в формате UTC

var str = '{"title":"Конференция","date":"2014-11-30T12:00:00.000Z"}';

var event = JSON.parse(str, function(key, value) {

if (key == 'date') return new Date(value);

return value;

});

alert( event.date.getDate() ); // теперь сработает!

эта возможность работает и для вложенных объектов тоже:

var schedule = '{ \

"events": [ \

{"title":"Конференция","date":"2014-11-30T12:00:00.000Z"}, \

{"title":"День рождения","date":"2015-04-18T12:00:00.000Z"} \

]\

}';

schedule = JSON.parse(schedule, function(key, value) {

if (key == 'date') return new Date(value);

return value;

});

alert( schedule.events[1].date.getDate() ); // сработает!

Функцию JSON.parse() можно использовать в контексте файла HTML.

Пример:

<!DOCTYPE html>

<html>

<body>

<p id="user"></p>

<script>

var s = '{"first\_name" : "John", "last\_name" : "Smith", "location" : "London"}';

var obj = JSON.parse(s);

document.getElementById("user").innerHTML =

"Name: " + obj.first\_name + " " + obj.last\_name + "<br>" +

"Location: " + obj.location;

</script>

</body>

</html>

Name: John Smith

Location: London

Метод JSON.stringify(value, replacer, space) преобразует («сериализует») значение в JSON-строку.

var event = {

title: "Конференция",

date: "сегодня"

};

var str = JSON.stringify(event);

alert( str ); // {"title":"Конференция","date":"сегодня"}

// Обратное преобразование.

event = JSON.parse(str);

При сериализации объекта вызывается его метод toJSON. Если такого метода нет – перечисляются его свойства, кроме функций.

var room = {

number: 23,

occupy: function() {

alert( this.number );

}

};

var event = {

title: "Конференция",

date: new Date(Date.UTC(2014, 0, 1)),

room: room

};

alert( JSON.stringify(event) );

/\*

{

"title":"Конференция",

"date":"2014-01-01T00:00:00.000Z", // (1)

"room": {"number":23} // (2)

}

\*/

Обратим внимание на два момента:

1. Дата превратилась в строку. Это не случайно: у всех дат есть встроенный метод toJSON. Его результат в данном случае – строка в таймзоне UTC.

2. У объекта room нет метода toJSON. Поэтому он сериализуется перечислением свойств.

Если добавить такой метод, тогда в итог попадет его результат:

var room = {

number: 23,

toJSON: function() {

return this.number;

}

};

alert( JSON.stringify(room) ); // 23

Попытаемся преобразовать в JSON объект, содержащий ссылку на DOM.

var user = {

name: "Вася",

age: 25,

window: window

};

alert( JSON.stringify(user) ); // ошибка!

// TypeError: Converting circular structure to JSON (текст из Chrome)

Глобальный объект window – сложная структура с кучей встроенных свойств и круговыми ссылками, поэтому его преобразовать невозможно.

Во втором параметре JSON.stringify(value, replacer) можно указать массив свойств, которые подлежат сериализации.

var user = {

name: "Вася",

age: 25,

window: window

};

alert( JSON.stringify(user, ["name", "age"]) );

// {"name":"Вася","age":25}

Для более сложных ситуаций вторым параметром можно передать функцию function(key, value), которая возвращает сериализованное value либо undefined, если его не нужно включать в результат:

var user = {

name: "Вася",

age: 25,

window: window

};

var str = JSON.stringify(user, function(key, value) {

if (key == 'window') return undefined;

return value;

});

alert( str ); // {"name":"Вася","age":25}

В примере выше функция пропустит свойство с названием window. Для остальных она просто возвращает значение, передавая его стандартному алгоритму.

Функция replacer работает рекурсивно. То есть, если объект содержит вложенные объекты, массивы и т.п., то все они пройдут через replacer.

В методе JSON.stringify(value, replacer, space) есть ещё третий параметр space.

Если он является числом – то уровни вложенности в JSON оформляются указанным количеством пробелов, если строкой – вставляется эта строка.

Например:

var user = {

name: "Вася",

age: 25,

roles: {

isAdmin: false,

isEditor: true

}

};

var str = JSON.stringify(user, "", 4);

alert( str );

/\* Результат -- красиво сериализованный объект:

{

"name": "Вася",

"age": 25,

"roles": {

"isAdmin": false,

"isEditor": true

}

}

\*/

**5. Порядок выполнения работы**

1. Создайте объект-конструктор (класс), который содержит информацию, в соответствии с вариантом, таблица 22.1.

|  |  |
| --- | --- |
| **№ варианта** | **Задание** |
| 1 | Информация о сотрудниках фирмы (ФИО, возраст, название отдела, стаж) |
| 2 | Информация о клиентах фирмы (ФИО, адрес, статус, размер скидки) |
| 3 | Информация о товаре (код товара, наименование, товарная группа, количество) |
| 4 | Информация о пациентах больницы (ФИО, дата рождения, диагноз, ФИО лечащего врача) |
| 5 | Информация о стране (Название, столица, население, национальный язык) |
| 6 | Информация об учащихся (ФИО, группа, отделение, пол) |
| 7 | Информацию об автошколах (название, адрес, номер телефона, количество автомобилей, стоимость) |
| 8 | Информация о кинотеатрах (название, адрес, телефон, количество залов) |
| 9 | Информация о парковках (адрес, количество мест, телефон, стоимость) |
| 10 | Информация о книгах (название, автор, год издания, количество страниц) |

2. Создайте экземпляр объекта, выполнив ввод данные с помощью диалогового окна prompt.

3. По нажатия на кнопку, преобразуйте созданный объект JavaScript в JSON-строку.

4. Выведите полученное строковое значение, выполнив встраивание текстового содержимого в HTML-элемент.

5. Преобразуйте созданную JSON-строку назад в объект JavaScript таким образом, чтобы полученный объект содержал только те свойства, для которых значение свойств имеет тип string.

6. Создайте строковую переменную, которая содержит данные в виде JSON-строки.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой JSON?

2. Опишите методы для работы с JSON в JavaScript.

3. Какой еще формат представления данных может быть использован для обмена данными между браузером и сервером?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 23**

## **Тема работы: «Обработка событий с использованием JQuery. Обработка JSON формата»**

**1. Цель работы**

Формирование умений обработки событий с использованием JQuery, работы с методами animate(), применения и модификации виджетов библиотеки jQuery UI, обработки данных форм при помощи библиотеки jQuery.

**2. Задание**

Реализовать управление элементами DOM, обработку событий и данных форм, создание визуальных эффектов с помощью библиотеки JQuery.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

jQuery – это библиотека, написанная на JavaScript, предназначенная для абстрагирования, выравнивания, исправления и упрощения скриптинга при работе с узлами HTML-элементов в браузере или для работы в [браузере без графического интерфейса](https://en.wikipedia.org/wiki/Headless_browser).

Способы подключения jQuery к сайту:

1.Можно скачать библиотеку jQuery с официального сайта и положить у себя на сайте, после этого подключить следующей командой:

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

2. Вместо того чтобы хранить библиотеку jQuery на своем сервере, можно воспользоваться одной из публично доступных сетей дистрибуции контента (Content Distribution Network – CDN), в которых хранится jQuery.

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>

//или

<script src="https://yastatic.net/jquery/2.2.0/jquery.min.js"></script>

В общем виде синтаксис для вызова метода jQuery имеет следующий вид:

$("селектор").имяМетода(параметры);

Выполнение различных сценариев возможно только после окончания загрузки структуры документа document, когда браузер преобразует html-код страницы в дерево DOM:

$(document).ready(function(){

//Code here

});, где code here - код выполняемой функции.

$(document).ready(function() {

alert("Документ доступен для выполнения скриптов");

});

Основную функциональность библиотеки jQuery выполняет функция jQuery(), которая имеет короткий псевдоним $():

jQuery(document).ready(function(e) {

alert('Привет')

});

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

alert('Привет')

});

Два примера аналогичны (выводят сообщение после загрузки страницы).

При вызове функция $() возвращает новый экземпляр объекта JQuery, который оборачивает ноль или более элементов DOM и позволяет взаимодействовать с ними различными способами.

Для хранения информации при работе с библиотекой jQuery используются переменные JavaScript. В переменных могут храниться элементы. Имена переменных, предназначенных для хранения возвращаемых элементов, начинаются со знака $, например:

$h = $(".list").parent().parent().detach();

Для хранения нескольких элементов используются массивы JavaScript:

$k[3] = 15;

В таблице 23.1 приведены возможные варианты вызова функции jQuery().

Таблица 23.1 – Варианты вызова функции jQuery()

|  |  |
| --- | --- |
| **Вариант вызова** | **Описание** |
| $(функция) | Позволяет указать функцию, которая должна быть выполнена по завершении построения DOM |
| $(селектор)  $(селектор, контекст) | Осуществляет выбор группы элементов в документе с помощью селектора |
| $(HTMLElement)  $(HTMLElement[]) | Создает объект jQuery из объекта или массива объектов HTMLElement |
| $() | Создает пустой набор элементов |
| $(HTML-код) | Создает новые элементы из фрагмента HTML-кода |

В большинстве случаев работа с jQuery осуществляется в два этапа, следуя одному и тому же шаблону:

1) первый этап заключается в выборе с помощью функции $() начального набора элементов страницы, соответствующих определенному критерию отбора, которые возвращаются в виде содержащего их объекта jQuery;

2) второй – в выполнении над каждым элементом набора одной или нескольких операций, приводящих к созданию конечного набора.

Чтобы выбрать элементы, нужно передать селектор функции $():

$("a")

$("#some-id")

$(".someclass")

$("header > ul:has(a)")

$("img:odd", "div#slideshow" // будут выбраны все четные картинки

// из блока с id=slideshow

$("input[type=checkbox][checked]") // позволит выбрать все флажки полей

// формы, которые были выделены пользователем

$("input:checkbox:checked:enabled") // выбрать только активные и отме

// ченные флажки полей формы

Допускается объединять несколько селекторов в одно выражение, разделяя селекторы запятой:

$("p,span") // позволит отобрать все

// элементы <p> и <span>

Функция $() возвращает объект jQuery, содержащий массив элементов DOM – так называемый обернутый набор, соответствующих указанному селектору.

Библиотека jQuery поддерживает все множество CSS-селекторов, а также некоторые дополнительные селекторы, которые обеспечивают удобные возможности детализированного управления процессом выбора элементов.

Виды селекторов представлены в таблицах 23.2 и 23.3.

Селекторы, позволяющие выбирать элементы в соответствии с их типом приведены в таблице 23.2.

Таблица 23.2 – Селекторы, позволяющие выбирать элементы в соответствии с их типом

|  |  |  |
| --- | --- | --- |
| **Селектор** | **Описание** | **Соответствующий элемент html** |
| :button | Выбирает все элементы типа button | <input type="button" value="Input Button"> |
| :checkbox | Выбирает все элементы типа checkbox | <input type="checkbox"> |
| :file | Выбирает все элементы типа file | <input type="file"> |
| :header | Выбирает все элементы заголовков | <h1>, <h2> и т.д. |
| :hidden | Выбирает все скрытые элементы | <input type="hidden"> |
| :image | Выбирает все элементы input для изображений | <input type="image"> |
| :input | Выбирает все элементы input |  |
| :parent | Выбирает все элементы, являющиеся родительскими по отношению к другим элементам |  |
| :password | Выбирает все элементы, являющиеся паролями | <input type="password"> |
| :radio | Выбирает все элементы типа radio | <input type="radio"> |
| :reset | Выбирает все элементы типа reset | <input type="reset"> |
| :selected | Соответствует всем выбранным элементам | <select>  <option>Option<option/>  </select> |
| :submit | Выбирает все элементы типа submit | <input type="submit"> |
| :visible | Выбирает все видимые элементы |  |

var imgOdd = $('img:odd');

Расширенные селекторы jQuery приведены в таблице 23.3.

Таблица 23.3 – Расширенные селекторы jQuery

|  |  |
| --- | --- |
| **Селектор** | **Описание** |
| **1** | **2** |
| :animated | Выбирает все анимируемые в данный момент элементы |
| :contains(текст) | Выбирает все элементы, содержащие указанный текст |
| :eq(n) | Выбирает элемент с индексом n (индексы отсчитываются от нуля) |
| :even | Выбирает все четные элементы (индексы отсчитываются от единицы) |
| :first | Выбирает первый из подходящих элементов |
| :gt(n) | Выбирает все элементы, индекс которых превышает n (индексы отсчитываются от нуля) |
| :has(селектор) | Выбирает элементы, которые содержат хотя бы один элемент, соответствующий указанному селектору |
| :last | Выбирает последний из подходящих элементов |
| :lt(n) | Выбирает все элементы, индекс которых меньше n (индексы отсчитываются от нуля) |

|  |  |
| --- | --- |
| **1** | **2** |
| :odd | Выбирает все нечетные элементы (индексы отсчитываются от единицы) |
| :text | Выбирает все текстовые элементы |

По умолчанию jQuery осуществляет поиск элементов в пределах всего DOM-дерева. Область поиска можно сузить, предоставив функции $() дополнительный аргумент. Это позволяет ограничить поиск определенным контекстом, который используется в качестве отправной точки при нахождении подходящих элементов. Соответствующий пример приведен ниже:

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

$("img:odd", $('.drow'))

.mouseenter(function(e) {

...

}).mouseout(function(e) {

...

});

});

В этом примере одно множество элементов, выбранных селектором jQuery, используется в качестве контекста для выбора другого множества. Сначала определяется контекст, которому соответствуют все элементы, имеющие класс drow. После этого набор выбранных элементов передается селектору img:odd в качестве контекста.

В таблице 23.4 приведены базовые свойства и методы объекта jQuery.

Таблица 23.4 – Базовые свойства и методы объекта jQuery

|  |  |  |
| --- | --- | --- |
| **Свойство/метод** | **Описание** | **Тип возвращаемого значения** |
| **1** | **2** | **3** |
| context | Возвращает набор элементов, используемых в качестве контекста поиска | HTMLElement |
| each(функция) | Выполняет указанную функцию для каждого из выбранных элементов | jQuery |
| get(индекс) | Получает объект HTMLElement с указанным индексом | HTMLElement |
| index(HTMLElement) | Производит поиск указанного объекта HTMLElement среди набора выбранных элементов и возвращает его индекс, если находит его | Number |
| index(jQuery) | Аналогичен предыдущему методу, но возвращает индекс первого из элементов, содержащихся в указанном объекте jQuery | Number |

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| index(селектор) | Возвращает индекс первого найденного элемента в объекте jQuery, вычисляемый относительно элементов соответствующих селектору | Number |
| Length | Возвращает число элементов в объекте jQuery | Number |
| Selector | Возвращает селектор | String |
| size() | Возвращает количество элементов, содержащихся в объекте jQuery | Number |
| toArray() | Возвращает объекты HTMLElement, содержащиеся HTMLElement в объекте jQuery, в виде массива |  |

Например, свойство selector возвращает селектор, который описывает набор выбранных элементов, содержащийся в объекте jQuery.

var selector = $('img:odd', $('.drow')).selector;

console.log("Селектор: " + selector); // Выведет: 'Селектор: .drow img:odd '

Процесс перемещения вниз по иерархической структуре DOM связан с выбором дочерних элементов (непосредственных потомков), а также всех остальных элементов, являющихся потомками элементов, содержащихся в объекте jQuery. Соответствующие методы jQuery и их краткое описание приведены в таблице 23.5.

Таблица 23.5 – Методы jQuery для выбора дочерних элементов

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| children() | Выбирает дочерние элементы всех элементов, содержащихся в объекте jQuery |
| children(селектор) | Выбирает все элементы, которые соответствуют указанному селектору и при этом являются непосредственными потомками элементов, содержащихся в объекте jQuery |
| contents() | Возвращает дочерние элементы и текстовое содержимое всех элементов, содержащихся в объекте jQuery |
| find() | Выбирает потомки элементов, содержащихся в объекте jQuery |
| find(селектор) | Выбирает элементы, которые соответствуют указанному селектору и при этом являются потомками элементов, содержащихся в объекте jQuery |
| find(jQuery),  find(HTMLElement),  find(HTMLElement[]) | Выбирает пересечение множества непосредственных потомков элементов, содержащихся в объекте jQuery, и множества элементов, содержащихся в объекте аргумента |

$(function() {

var childCount = $('div.drow').children().each(function(index, elem) {

console.log("Дочерний элемент: " + elem.tagName + " " + elem.className); }).length;

console.log("Всего имеется " + childCount + " дочерних элементов");

var descCount = $('div.drow').find('img').each(function(index, elem) {

console.log("Потомок: " + elem.tagName + " " + elem.src);

}).length;

console.log("Всего имеется " + descCount + " элементов-потомков img");

});

Перемещению вверх по DOM-дереву соответствует поиск родителей и предков элементов, содержащихся в объекте jQuery. Методы, используемые для таких перемещений, приведены в таблице 23.6.

Таблица 23.6 – Методы jQuery для выбора родителей и предков элементов.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| closest(селектор), closest(селектор, контекст) | Выбор ближайшего предка, соответствующего указанному селектору, для каждого элемента, содержащегося в объекте jQuery |
| closest(jQuery), closest(HTMLElement) | Выбор ближайшего предка для каждого элемента в объекте jQuery, совпадающего с одним из элементов, содержащихся в объекте аргумента |
| offsetParent() | Нахождение ближайшего предка, значением CSS-свойства position которого является fixed, absolute или relative |
| parent(), parent(селектор) | Выбор непосредственных предков для каждого элемента в объекте jQuery с возможностью их фильтрации с помощью селектора |
| parents(), parents(селектор) | Выбор предков для каждого элемента в объекте jQuery с возможностью их фильтрации с помощью селектора |
| parentsUntil(селектор), parentsUntil(селектор, селектор) | Выбор предков для каждого элемента в объекте jQuery до тех пор, пока не встретится элемент, соответствующий селектору. Результаты могут фильтроваться посредством второго селектора |
| parentsUntil(HTMLElement), parentsUntil(HTMLElement, селектор), parentsUntil(HTMLElement[]), parentsUntil(HTMLElement[], селектор) | Выбирает предков для каждого элемента в объекте jQuery до тех пор, пока не встретится один из указанных элементов. Результаты могут фильтроваться посредством второго селектора |

$(function() {

$('div.dcell').parent().each(function(index, elem) {

console.log("Элемент: " + elem.tagName + " " + elem.id);

});

$('div.dcell').parent('#row1').each(function(index, elem) {

console.log("Отфильтрованный элемент: " + elem.tagName + " " + elem.id);

});

});

Перемещение по DOM-дереву между узлами одного и того же уровня иерархии (между сестринскими узлами) осуществляется с помощью предназначенных для этого методов jQuery, приведенных в таблице 23.7.

Таблица 23.7 – Методы jQuery для перемещения между узлами одного и того же уровня иерархии.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| next(), next(селектор) | Выбирает сестринские элементы, непосредственно следующие за каждым из элементов, содержащихся в объекте jQuery. Имеется дополнительная возможность фильтрации результатов с использованием селектора |
| nextAll(), nextAll(селектор) | Выбирает все последующие сестринские элементы для каждого из элементов, содержащихся в объекте jQuery. Имеется дополнительная возможность фильтрации результатов с использованием селектора |
| nextUntil(селектор), nextUntil(селектор, селектор), nextUntil(jQuery), nextUntil(jQuery, селектор), nextUntil(HTMLElement[]), nextUntil(HTMLElement[], селектор) | Выбирает для каждого элемента последующие сестринские элементы вплоть до элемента (но не включая его), соответствующего селектору или содержащегося в объекте jQuery или массиве HTMLElement[]. Имеется дополнительная возможность фильтрации результатов с использованием селектора |
| prev(), prev(селектор) | Выбирает сестринские элементы, непосредственно предшествующие каждому из элементов, содержащихся в объекте jQuery. Имеется дополнительная возможность фильтрации результатов с использованием селектора |
| prevAll(), prevAll(селектор) | Выбирает все предшествующие сестринские элементы для каждого из элементов, содержащихся в объекте jQuery. Имеется дополнительная возможность фильтрации результатов с использованием селектора, передаваемого методу в качестве второго аргумента |
| prevUntil(селектор), prevUntil(селектор, селектор), prevUntil(jQuery), prevUntil(jQuery, селектор), prevUntil(HTMLElement[]), prevUntil(HTMLElement[], селектор) | Выбирает для каждого элемента предшествующие сестринские элементы вплоть до элемента (но не включая его), соответствующего селектору или содержащегося в объекте jQuery или массиве HTMLElement[]. Имеется дополнительная возможность фильтрации результатов с использованием селектора, передаваемого методу в качестве второго аргумента |
| siblings(), siblings(селектор) | Выбирает все сестринские элементы для каждого из элементов, содержащихся в объекте jQuery. Имеется дополнительная возможность фильтрации результатов с использованием селектора |

$(function() {

$('img[src\*=astor], img[src\*=primula]')

.parent().siblings().css("border", "thick solid blue");

});

Одним из способов создания элементов является передача строки, содержащей HTML-фрагмент, функции $(), которая выполнит синтаксический анализ строки и создаст соответствующие DOM-объекты.

$(function() {

var newElems = $('<div class="dcell"><img src="http://professorweb.ru/downloads/jquery/lily.png"/></div>');

});

При создании нового элемента методу $() можно передать второй параметр в виде объекта JavaScript, определяющий дополнительные атрибуты элемента:

$("<img/>",

{src: "images/flower.jpg",

title: "Rose\_in\_garden",

click: function() {...}

}).appendTo("body");

Методы для вставки дочерних элементов и элементов-потомков приведены в таблице 23.8.

Таблица 23.8 – Методы jQuery для вставки дочерних элементов и элементов-потомков

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **1** | **2** |
| append(HTML),  append(jQuery),  append(HTMLElement[]) | Вставляет указанные элементы в качестве последних дочерних элементов во все выбранные элементы |
| prepend(HTML),  prepend(jQuery),  prepend(HTMLElement[]) | Вставляет указанные элементы в качестве первых дочерних элементов во все выбранные элементы |
| appendTo(jQuery), appendTo(HTMLElement[]) | Вставляет элементы, содержащиеся в объекте jQuery, в качестве последних дочерних элементов в элементы, заданные аргументом |

|  |  |
| --- | --- |
| **1** | **2** |
| prependTo(HTML),  prependTo(jQuery),  prependTo(HTMLElement[]) | Вставляет элементы, содержащиеся в объекте jQuery, в качестве первых дочерних элементов в элементы, заданные аргументом |
| append(функция), prepend(функция) | Добавляет результат, возвращаемый функцией, в окончание или начало содержимого каждого из элементов, содержащихся в объекте jQuery |

$(function() {

var newElems = $("<div class='dcell'></div>")

.append("<img src='http://professorweb.ru/downloads/jquery/lily.png'/>")

.append("<label for='lily'>Лилии:</label>")

.append("<input name='lily' value='0' required />");

newElems.css("border", "thick solid red");

$('#row1').append(newElems);

});

В этом сценарии метод append() используется двояким образом: сначала – для построения нужного набора новых элементов, а затем – для вставки этих элементов в HTML-документ.

Методы для внешней вставки родительских элементов и элементов-предков приведены в таблице 23.9.

Таблица 23.9 – Методы jQuery для внешней вставки родительских элементов и элементов-предков

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| wrap(HTML), wrap(jQuery), wrap(HTMLElement[]) | Обертывает указанные элементы вокруг каждого из элементов, содержащихся в объекте jQuery |
| wrapAll(HTML), wrapAll(jQuery), wrapAll(HTMLElement[]) | Обертывает указанные элементы вокруг набора элементов, содержащихся в объекте jQuery (рассматриваемых как единая группа) |
| wrapInner(HTML), wrapInner(jQuery), wrapInner(HTMLElement[]) | Обертывает указанные элементы вокруг содержимого каждого из элементов, содержащихся в объекте jQuery |
| wrap(функция), wrapInner(функция) | Динамически обертывает элементы с использованием функции |

При обертывании элементов эти методы могут принимать HTML-фрагмент в качестве аргумента, но всегда следует проверять, чтобы этот фрагмент содержал только один внутренний элемент. В противном случае jQuery не сможет определить, что именно необходимо сделать. Отсюда следует, что каждый элемент в аргументе метода может иметь не более одного родительского и не более одного дочернего элемента.

Методы для вставки сестринских элементов приведены в таблице 23.10.

Таблица 23.10 – Методы jQuery для вставки сестринских элементов

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| after(HTML), after(jQuery), after(HTMLElement[]) | Вставляет указанные элементы в качестве последних дочерних элементов во все выбранные элементы |
| before(HTML),  before(jQuery),  before(HTMLElement[]) | Вставляет указанные элементы в качестве первых дочерних элементов во все выбранные элементы |
| insertAfter(HTML),  insertAfter(jQuery),  insertAfter(HTMLElement[]) | Вставляет элементы, содержащиеся в объекте jQuery, в качестве последних дочерних элементов в элементы, заданные аргументом |
| insertBefore(HTML),  insertBefore(jQuery),  insertBefore(HTMLElement[]) | Вставляет элементы, содержащиеся в объекте jQuery, в качестве первых дочерних элементов в элементы, заданные аргументом |
| after(функция), before(функция) | Добавляет результат, возвращаемый функцией, в окончание или начало содержимого каждого из элементов, содержащихся в объекте jQuery |

$(function() {

var orchidElems = $("<div class='dcell'/>")

.append("<img src='http://professorweb.ru/downloads/jquery/orchid.png'/>")

.append("<label for='orchid'>Орхидея:</label>")

.append("<input name='orchid' value='0' required />");

var lilyElems = $("<div class='dcell'/>")

.append("<img src='http://professorweb.ru/downloads/jquery/lily.png'/>")

.append("<label for='lily'>Лилия:</label>")

.append("<input name='lily' value='0' required />");

$(orchidElems).add(lilyElems).css("border", "thick solid red");

$('#row1 div.dcell').after(orchidElems);

$('#row2 div.dcell').before(lilyElems);

});

Методы замены элементов приведены в таблице 23.11.

Таблица 23.11 – Методы jQuery для замены элементов

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **1** | **2** |
| replaceWith(HTML), replaceWith(jQuery), replaceWith(HTMLElement[]) | Заменяет элементы, содержащиеся в объекте jQuery, указанным содержимым |

|  |  |
| --- | --- |
| **1** | **2** |
| replaceAll(jQuery), replaceAll(HTMLElement[]) | Заменяет элементы, заданные аргументом, элементами, содержащимися в объекте jQuery |
| replaceWith(функция) | Выполняет динамическую замену элементов, содержащихся в объекте jQuery, с использованием функции |

$(function() {

var newElems= $("<div class='dcell'/>")

.append("<img src='http://professorweb.ru/downloads/jquery/orchid.png'/>")

.append("<label for='orchid'>Орхидея:</label>")

.append("<input name='orchid' value='0' required />")

.css("border", "thick solid red");

$('#row1').children().first().replaceWith(newElems);

$("<img src='http://professorweb.ru/downloads/jquery/carnation.png'/>").replaceAll('#row2 img')

});

Методы для удаления элементов приведены в таблице 23.12.

Таблица 23.12 – Методы jQuery для удаления элементов

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| detach(), detach(селектор) | Удаляет элементы из DOM. Данные, связанные с элементами, сохраняются |
| empty() | Удаляет все дочерние узлы каждого из элементов, содержащихся в объекте jQuery |
| remove(), remove(селектор) | Удаляет элементы из DOM. По мере удаления элементов связанные с ними данные уничтожаются |
| unwrap() | Удаляет родительские элементы каждого из элементов, содержащихся в объекте jQuery |

$(function() {

$('img[src\*=daffodil], img[src\*=snow]').parent().remove();

});

В этом сценарии мы выбираем элементы img, атрибуты src которых содержат daffodil и snow, получаем их родительские элементы, а затем удаляем их.

Библиотека jQuery позволяет получать и устанавливать значения атрибутов элементов, содержащихся в объекте jQuery.

Методы для работы с атрибутами приведены в таблице 23.13.

Таблица 23.13 – Методы jQuery для работы с атрибутами

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| attr(имя) | Возвращает значение атрибута с указанным именем для первого из элементов, содержащихся в объекте jQuery |
| attr(имя, значение) | Устанавливает значение атрибута с указанным именем для всех элементов, содержащихся в объекте jQuery |
| attr(объект отображения) | Устанавливает атрибуты, указанные в объекте отображения, для всех элементов, содержащихся в объекте jQuery |
| attr(имя, функция) | Устанавливает указанный атрибут для всех элементов, содержащихся в объекте jQuery, с помощью функции |
| removeAttr(имя), removeAttr(имя[]) | Удаляет атрибут (атрибуты) из всех элементов, содержащихся в объекте jQuery |
| prop(имя) | Возвращает значение указанного свойства для первого из элементов, содержащихся в объекте jQuery |
| prop(имя, значение), prop(объект отображения) | Устанавливает значение одного или нескольких свойств для всех элементов, содержащихся в объекте jQuery |
| prop(имя, функция) | Устанавливает значение указанного свойства для всех элементов, содержащихся в объекте jQuery, с использованием функции |
| removeProp(имя) | Удаляет указанное свойство из всех элементов, содержащихся в объекте jQuery |

Если метод attr() вызывается с одним аргументом, jQuery возвращает значение указанного атрибута для первого из элементов выбранного набора.

$(function() {

var srcValue = $('img').attr('src');

console.log("Значение атрибута: " + srcValue);

});

Каждому варианту метода attr() соответствует аналогичный вариант вызова метода prop(). Различие между обоими методами состоит в том, что методы prop() имеют дело со свойствами, определяемыми объектами HTMLElement, а не со значениями атрибутов. Часто атрибуты и соответствующие свойства имеют одинаковые имена, но это не всегда так.

Пример использования метода prop() для получения значения свойства className приведен ниже:

$(function() {

$('\*[class]').each(function(index, elem) {

console.log("Элемент:" + elem.tagName + " " + $(elem).prop("className"));

});

});

В этом примере сначала выбираются в виде набора, а затем последовательно перебираются с помощью метода each() все элементы, у которых имеется атрибут class. Для каждого элемента на консоль выводится его тип и значение свойства className.

Несмотря на то, что для работы с классами можно использовать общие методы, предназначенные для работы с атрибутами, в jQuery предусмотрены намного более удобные методы, специально предназначенные для этой цели. Их краткое описание приведено в таблице 23.14.

Таблица 23.14 – Методы jQuery для работы с классами

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| addClass(имя имя ...) | Добавляет один или несколько классов с указанными именами во все элементы, содержащиеся в объекте jQuery |
| addClass(функция) | Добавляет классы, список которых возвращает указанная функция, во все элементы, содержащиеся в объекте jQuery |
| hasClass(имя) | Возвращает true, если хотя бы один из элементов, содержащихся в объекте jQuery, принадлежит указанному классу |
| removeClass(имя имя ...) | Удаляет один или несколько классов с указанными именами из всех элементов, содержащихся в объекте jQuery |
| removeClass(функция) | Удаляет классы, список которых возвращает указанная функция, из всех элементов, содержащихся в объекте jQuery |
| toggleClass() | Переключает все классы, которым принадлежат элементы, содержащиеся в объекте jQuery |
| toggleClass (логическое\_значение) | Осуществляет одностороннее переключение всех классов, которым принадлежат элементы, содержащиеся в объекте jQuery |
| toggleClass(имя), toggleClass(имя имя ...) | Переключает один или несколько классов с указанными именами для всех элементов, содержащихся в объекте jQuery |
| toggleClass(имя, логическое\_ значение) | Осуществляет одностороннее переключение класса с указанным именем для всех элементов, содержащихся в объекте jQuery |
| toggleClass(функция, логическое значение) | Динамически переключает классы для всех элементов, содержащихся в объекте jQuery |

<style type="text/css">

img.redBorder {border: thick solid red}

img.blueBorder {border: thick solid blue}

</style>

<script>

$(function() {

$('img').addClass("redBorder");

$('img:even').removeClass("redBorder").addClass("blueBorder");

console.log("Все элементы: " + $('img').hasClass('redBorder'));

$('img').each(function(index, elem) {

console.log("Элемент: " + $(elem).hasClass('redBorder') + " " + elem.src);

});

});

</script>

Библиотека jQuery предоставляет ряд удобных специализированных методов, значительно упрощающих работу со стилями CSS. Одним из наиболее широко используемых методов такого рода является метод css(), краткое описание которого приведено в таблице 23.15.

Таблица 23.15 – Метод jQuery для работы со стилями css()

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| css(имя) | Возвращает значение указанного свойства для первого из элементов, содержащихся в объекте jQuery |
| css(имя, значение) | Устанавливает значение указанного свойства для всех элементов, содержащихся в объекте jQuery |
| css(объект отображения) | Устанавливает одновременно несколько свойств для всех элементов, содержащихся в объекте jQuery, с помощью объекта отображения |
| css(имя, функция) | Устанавливает значения указанного свойства для всех элементов, содержащихся в объекте jQuery, с помощью функции |

При считывании значений свойств с помощью метода css() вы получаете значение свойства, которое имеет первый из элементов, содержащихся в объекте jQuery. В то же время при установке свойства вносимое изменение применяется ко всем элементам набора:

$(function() {

var sizeVal = $('label').css("font-size");

console.log("Размер шрифта: " + sizeVal);

$('label').css("font-size", "1.5em");

});

Существуют два способа одновременной установки нескольких CSS-свойств. Первый из них – это формирование цепочки вызовов метода css(), как показано ниже:

$(function() {

$('label').css("font-size", "1.5em").css("color", "blue");

});

В этом сценарии устанавливаются значения свойств font-size и color. Того же эффекта можно добиться, используя объект отображения, как показано в примере ниже:

$(function() {

var cssVals = {

"font-size": "1.5em",

"color": "blue"

};

$('label').css(cssVals);

});

Можно устанавливать значения свойств динамически, передавая функцию методу css(). Соответствующий пример приведен ниже:

$(function() {

$('label').css("border", function(index, currentValue) {

if ($(this).closest("#row1").length > 0) {

return "thick solid red";

} else if (index % 2 == 1) {

return "thick double blue";

}

});

});

Методы для работы с конкретными свойствами CSS приведены в таблице 23.16.

Таблица 23.16 – Метод jQuery для работы с конкретными свойствами CSS

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **1** | **2** |
| height() | Возвращает высоту (в пикселях) первого из элементов, содержащихся в объекте jQuery |
| height(значение) | Устанавливает высоту для всех элементов, содержащихся в объекте jQuery |

|  |  |
| --- | --- |
| **1** | **2** |
| innerHeight() | Возвращает внутреннюю высоту (т.е. высоту элемента, включая внутренние отступы, но исключая границы и поля) первого из элементов, содержащихся в объекте jQuery |
| innerWidth() | Возвращает внутреннюю ширину (т.е. ширину элемента, включая внутренние отступы, но исключая границы и поля) первого из элементов, содержащихся в объекте jQuery |
| offset() | Возвращает координаты первого из элементов, содержащихся в объекте jQuery, относительно начала документа |
| outerHeight(логическое\_значение) | Возвращает высоту первого из элементов, содержащихся в объекте jQuery, включая внутренние отступы и границы. Аргумент определяет, должен ли при этом учитываться размер полей |
| outerWidth(логическое\_значение) | Получает ширину первого из элементов, содержащихся в объекте jQuery, включая внутренние отступы и границы. Аргумент определяет, должен ли при этом учитываться размер полей |
| position() | Возвращает координаты первого из элементов, содержащихся в объекте jQuery, относительно его родительского элемента, у которого задан тип позиционирования |
| scrollLeft(), scrollTop() | Получает значение отступа прокрутки слева или сверху для первого из элементов, содержащихся в объекте jQuery |
| scrollLeft(значение), scrollTop(значение) | Устанавливает значение отступа прокрутки слева или сверху для всех элементов, содержащихся в объекте jQuery |
| width() | Получает ширину первого из элементов, содержащихся в объекте jQuery |
| width(значение) | Устанавливает ширину для всех элементов, содержащихся в объекте jQuery |
| height(функция), width(функция) | Устанавливает высоту или ширину всех элементов, содержащихся в объекте jQuery, с помощью функции |

Методы для работы с содержимым элементов приведены в таблице 23.17.

Таблица 23.17 – Метод jQuery для работы с содержимым элементов

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| text() | Получает объединенное текстовое содержимое всех элементов, содержащихся в объекте jQuery, и их потомков |
| text(значение) | Устанавливает текстовое содержимое для всех элементов, содержащихся В объекте jQuery |
| html() | Возвращает HTML-содержимое первого из элементов, содержащихся в объекте jQuery |
| html(значение) | Устанавливает HTML-содержимое для всех элементов, содержащихся в объекте jQuery |
| text(функция), html(функция) | Устанавливает текстовое и HTML-содержимое с помощью функции |

$(function() {

var html = $('div.dcell').html();

console.log(html);

});

Методы для обработки событий приведены в таблице 23.18.

Таблица 23.18 – Метод jQuery для обработки событий

|  |  |
| --- | --- |
| **Описание** | **Метод** |
| bind(тип\_события, функция), bind (тип\_события, данные, функция) | Добавляет обработчик событий в элементы, содержащиеся в объекте jQuery, с дополнительной возможностью передачи данных обработчику |
| bind(тип\_события, логическое значение) | Создает обработчик событий по умолчанию, который всегда возвращает false, тем самым предотвращая выполнение действия, предусмотренного по умолчанию. Аргумент логическое\_значение позволяет управлять всплытием событий |
| bind(объект отображения) | Добавляет набор обработчиков событий, заданных объектом отображения, во все элементы, содержащиеся в объекте jQuery |
| one(тип\_события, функция), one(тип\_события, данные, функция) | Добавляет обработчик событий в каждый из элементов, содержащихся в объекте jQuery, с дополнительной возможностью передачи данных обработчику. Обработчик может быть выполнен не более одного раза для каждого из элементов, после чего он отсоединяется от элемента |
| unbind() | Удаляет все обработчики событий из всех элементов, содержащихся в объекте jQuery |
| unbind(тип\_события) | Удаляет ранее зарегистрированный обработчик событий из всех элементов, содержащихся в объекте jQuery |
| unbind(тип\_события, логическое\_значение) | Удаляет ранее зарегистрированный обработчик событий, всегда возвращающий значение false, из всех элементов, содержащихся в объекте jQuery |
| unbind(Event) | Удаляет обработчик событий с использованием объекта Event |

$(function() {

$('img').bind("mouseenter", handleMouseEnter)

.bind("mouseout", handleMouseOut);

function handleMouseEnter(e) {

$(this).css({

"border": "thick solid red",

"opacity": "0.5"

});

};

function handleMouseOut(e) {

$(this).css({

"border": "",

"opacity": ""

});

}

});

В этом примере выбираются все элементы img в документе и с помощью метода bind() регистрируются обработчики событий mouseenter и mouseout. В данном случае обработчики изменяют значения свойств border и opacity с помощью метода css(). При наведении указателя мыши на любой из элементов img соответствующее изображение заключается в рамку и становится частично прозрачным, но восстанавливает первоначальное состояние, когда указатель покидает область изображения.

Внутри функции-обработчика переменная this ссылается на набор элементов, к которому присоединен обработчик. Передаваемый обработчику объект Event является собственным объектом библиотеки jQuery и отличается от объекта Event, определенного в спецификации DOM. Свойства и методы объекта Event в jQuery описаны в таблице 23.19.

Таблица 23.19 – Свойства и методы объекта Event в jQuery

|  |  |  |
| --- | --- | --- |
| **Свойство/метод** | **Описание** | **Тип возвращаемого значения** |
| **1** | **2** | **3** |
| currentTarget | Возвращает текущий элемент, событие которого обрабатывается. Этот элемент не обязательно является источником события, так как последнее могло быть передано ему дочерним элементом посредством механизма "всплытия" событий вверх по DOM-дереву. Для определения источника события необходимо использовать свойство target | HTMLElement |
| Data | Возвращает дополнительные данные, переданные методу bind() при регистрации обработчика | object |
| isDefaultPrevented() | Возвращает true, если для данного объекта событий ранее вызывался метод preventDefault() | boolean |
| isImmediatePropagationsStopped() | Возвращает true, если для данного объекта событий ранее вызывался метод stopImmediatePropagation() | boolean |
| isPropagationsStopped() | Возвращает true, если для данного объекта событий ранее вызывался метод stopPropagation() | boolean |
| originalEvent | Возвращает первоначальный DOM-объект Event | Event |
| pageX, pageY | Координаты указателя мыши относительно левого верхнего угла документа | number |

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| preventDefault() | Отменяет выполнение действий по умолчанию, связанных с данным событием | void |
| relatedTarget | Для событий мыши возвращает другой имеющий отношение к событию объект, если таковой имеется. Какой именно будет этот объект, зависит от конкретного события | HTMLElement |
| Result | Результат, возвращенный обработчиком данного события при его последнем вызове | object |
| stopImmediatePropagation() | Отменяет выполнение любых других обработчиков, связанных с данным событием | void |
| stopPropagation() | Предотвращает всплытие события вверх по иерархии DOM, но разрешает обработчикам, связанным с текущим обрабатываемым элементом, получить данное событие | void |
| Target | Возвращает элемент, являющийся источником события | HTMLElement |
| timestamp | Возвращает время наступления события | number |
| type | Возвращает тип события | string |
| which | Возвращает информацию о нажатой кнопке или клавише для событий, связанных с мышью или клавиатурой | number |

Назначением большинства базовых эффектов является простое отображение или сокрытие элементов. Методы, используемые для этой цели, описаны в таблице 23.20.

Таблица 23.20 – Методы jQuery для создания базовых эффектов

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **1** | **2** |
| hide() | Немедленно скрывает все элементы, содержащиеся в объекте jQuery |
| hide(продолжительность), hide(продолжительность, стиль) | Плавно скрывает элементы, содержащиеся в объекте jQuery, в течение заданного времени с возможностью указания стиля анимации |
| hide(продолжительность, функция), hide(продолжительность, стиль, функция) | Скрывают все элементы, содержащиеся в объекте jQuery, в течение заданного времени с возможностью указания стиля анимации и функции, которая вызывается по завершении создания эффекта |
| show() | Немедленно отображает все элементы, содержащиеся в объекте jQuery |

|  |  |
| --- | --- |
| **1** | **2** |
| show(продолжительность), show(продолжительность, стиль) | Отображают все элементы, содержащиеся в объекте jQuery, в течение заданного времени с возможностью указания стиля анимации |
| show(продолжительность, функция), show(продолжительность, стиль, функция) | Отображают все элементы, содержащиеся в объекте jQuery, в течение заданного времени с возможностью указания стиля анимации и функции, которая вызывается по завершении создания эффекта |
| toggle() | Немедленно переключает (отображает, если они скрыты, и скрывает, если они отображаются) видимость элементов, содержащихся в объекте jQuery |
| toggle(продолжительность), toggle(продолжительность, стиль) | Переключают видимость элементов, содержащихся в объекте jQuery, в течение заданного времени с возможностью указания стиля анимации |
| toggle(продолжительность, функция), toggle(продолжительность, стиль, функция) | Переключают видимость элементов, содержащихся в объекте jQuery, в течение заданного времени с возможностью указания стиля анимации и функции, которая вызывается по завершении создания эффекта |
| toggle(логическое\_значение) | Осуществляет одностороннее переключение видимости элементов, содержащихся в объекте jQuery |

Пример использования методов show() и hide() без аргументов для создания простейших эффектов приведен ниже:

$(function() {

$("<button>Скрыть</button><button>Показать</button>").appendTo("#buttonDiv")

.click(function(e) {

if ($(e.target).text() == "Скрыть") {

$('#row1 div.dcell').hide();

} else {

$('#row1 div.dcell').show();

}

e.preventDefault();

});

});

Методы для создания пользовательских эффектов перечислены в таблице 23.21.

Таблица 23.21 – Методы jQuery для создания пользовательских эффектов

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **1** | **2** |
| *animate(свойства),*  *animate(свойства, продолжительность),*  *animate(свойства, продолжительность,* | Выполняют анимацию за счет плавного изменения значений CSS-свойств с возможностью указания длительности и стиля |
| ***1*** | **2** |
| *функция),*  *animate(свойства, продолжительность, стиль, функция)* | анимации, а также функции обратного вызова |
| *animate(свойства, продолжительность, параметры)* | Выполняет анимацию за счет плавного изменения значений CSS-свойств, которые, как и дополнительные параметры, передаются методу в виде объекта с возможностью передачи объекта, содержащего дополнительные параметры, в качестве аргумента |

Библиотека jQuery позволяет выполнить анимацию любого свойства, принимающего простые числовые значения, например свойства height.

В качестве аргументов метод animate() принимает объект отображения, содержащий набор свойств в виде пар "имя-значение", анимацию которых требуется выполнить, и дополнительные параметры. Пример пользовательской анимации приведен ниже:

$(function() {

$('form').css({"position": "fixed", "top": "70px", "z-index": "2"});

$('h1').css({"position": "fixed", "z-index": "1", "min-width": "0"});

$("<button>Анимировать</button>").insertAfter("#buttonDiv button")

.click(function(e) {

$('h1').animate({

height: $('h1').height() + $('form').height() + 10,

width: ($('form').width())

});

e.preventDefault();

});

});

**5. Порядок выполнения работы**

1. Создайте переменную, которая хранит данные о содержании элементов списка в формате JSON.

2. Выполните преобразование данных, хранящихся в созданной переменной в объект JavaScript.

3. Используя DOM, выполните добавление списка на страницу. В элементы списка внесите информацию из свойств объекта.

4. С использованием библиотеки JQuery выведите в консоль все элементы, являющиеся дочерними по отношению к <body>.

5. Используя методы библиотеки JQuery, добавьте на страницу таблицу, как показано на рисунке 23.1.

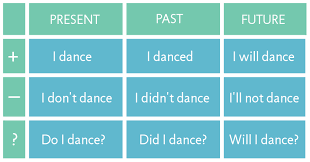


Рисунок 23.1 – Таблица времен

6. Стилевое оформление таблицы необходимо выполнить добавлением (с помощью библиотеки JQuery) элементам таблиц атрибута class.

7. Используя библиотеку JQuery добавить события к заголовкам таблицы: при наведении на заголовок должна выводиться информацоя с описанием времени, указанного в заголовке.

8. Создайте HTML-форму и организуйте проверку заполненности всех полей по нажатию на кнопку. Если останутся незаполненные поля, необходимо вывести предупреждение.

9. Создайте для кнопки эффект «переключателя», как показано на рисунке 23.2 с использованием библиотеки JQuery.

|  |
| --- |
|  |

Рисунок 23.2 – Переключатель

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Что представляет собой JQuery?

2. Перечислите основные возможности JQuery.

3. Правила добавления jQuery на страницу.

4. Перечислите возможные варианты вызова функции jQuery().

5. Для чего выполняется получение обернутого набора элементов?

6. Перечислите известные методы получения обернутого набора.

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 24**

## **Тема работы: «Подключение и настройка фреймворка Bootstrap. Работа с блочной системой. Мобильная верстка Bootstrap. Работа со стилистическим оформлением страницы»**

**1. Цель работы**

Формирование умений подключения и настройки Bootstrap, работы с блочной системой, адаптивной мобильной верстки Bootstrap, работы со стилистическим оформлением страницы.

**2. Задание**

Реализовать внедрении CSS различными способами, форматирование внешнего вида элемнтов веб-документа.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

CSS (Cascading Style Sheets), или каскадные таблицы стилей, используются для описания внешнего вида документа, написанного языком разметки.

Объявление стиля состоит из двух частей: элемента веб-страницы – селектора, и команды форм

<https://itchief.ru/bootstrap/installation>

 необходимо скачать Bootstrapнапример, с сайта разработчика:

http://getbootstrap.com/getting-started/

В папках со стилями и скриптами приложено две версии файлов — исходная и компактная (в имени содержится min). Компактная отличается лишь размером файла и снижением читаемости кода.

Копируем все папки в наш проект и в корне создаём index.html. В итоге структура нашего проекта будет выглядеть следующим образом.

*Файлы и папки нашего проекта*

Содержимое index.html включает в себя ссылку на стилевой файл bootstrap.css и внизу страницы мы вызываем bootstrap.js. Больше пока ничего не нужно.

<https://webref.ru/layout/bootstrap/install>

**5. Порядок выполнения работы**

1. В файле style.css определите цвет фона и текста документа, свойства полос прокрутки.

2. Подключите файл style.css к документу Lab5.html (методом связывания).

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Перечислите способы использования каскадных таблиц стилей. Укажите преимущества и недостатки каждого из них.

2. Укажите формат правила CSS.

3. Что такое селектор?

4. Как создать класс, не связанный с определенным тегом?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 25**

## **Тема работы: «Верстка шаблона с использованием элементов для работы с выпадающими списками. Работа с модальными окнами Bootstrap»**

**1. Цель работы**

Формирование умений работы с элементами Bootstrap: выпадающими списками и модальными окнами.

**2. Задание**

Реализовать внедрении CSS различными способами, форматирование внешнего вида элемнтов веб-документа.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

CSS (Cascading Style Sheets), или каскадные таблицы стилей, используются для описания внешнего вида документа, написанного языком разметки.

Объявление стиля состоит из двух частей: элемента веб-страницы – селектора, и команды форм

**5. Порядок выполнения работы**

1. В файле style.css определите цвет фона и текста документа, свойства полос прокрутки.

2. Подключите файл style.css к документу Lab5.html (методом связывания).

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Перечислите способы использования каскадных таблиц стилей. Укажите преимущества и недостатки каждого из них.

2. Укажите формат правила CSS.

3. Что такое селектор?

4. Как создать класс, не связанный с определенным тегом?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 26**

## **Тема работы: «Верстка шаблона с использованием элементов для работы с кнопками. Навигационная последовательность. Нумерация страниц»**

**1. Цель работы**

Формирование умений работы с элементами Bootstrap: кнопки, навигационная панель, нумерация страниц.

**2. Задание**

Реализовать внедрении CSS различными способами, форматирование внешнего вида элемнтов веб-документа.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

CSS (Cascading Style Sheets), или каскадные таблицы стилей, используются для описания внешнего вида документа, написанного языком разметки.

Объявление стиля состоит из двух частей: элемента веб-страницы – селектора, и команды форм

**5. Порядок выполнения работы**

1. В файле style.css определите цвет фона и текста документа, свойства полос прокрутки.

2. Подключите файл style.css к документу Lab5.html (методом связывания).

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Перечислите способы использования каскадных таблиц стилей. Укажите преимущества и недостатки каждого из них.

2. Укажите формат правила CSS.

3. Что такое селектор?

4. Как создать класс, не связанный с определенным тегом?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.

# **Лабораторная работа № 27**

## **Тема работы: «Верстка шаблона с использованием элементов для работы с миниатюрами. Работа с индикатором процесса»**

**1. Цель работы**

Формирование умений верстки шаблона с использованием миниатюр Bootstrap, работы с индикатором процесса.

**2. Задание**

Реализовать внедрении CSS различными способами, форматирование внешнего вида элемнтов веб-документа.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

CSS (Cascading Style Sheets), или каскадные таблицы стилей, используются для описания внешнего вида документа, написанного языком разметки.

Объявление стиля состоит из двух частей: элемента веб-страницы – селектора, и команды форм

**5. Порядок выполнения работы**

1. В файле style.css определите цвет фона и текста документа, свойства полос прокрутки.

2. Подключите файл style.css к документу Lab5.html (методом связывания).

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

*Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_*

*Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**7. Контрольные вопросы**

1. Перечислите способы использования каскадных таблиц стилей. Укажите преимущества и недостатки каждого из них.

2. Укажите формат правила CSS.

3. Что такое селектор?

4. Как создать класс, не связанный с определенным тегом?

**8. Рекомендуемая литература**

1. Гарднер, Л. Разработка веб-сайтов для мобильных устройств / Л. Гарднер, Д. Григсби – СПб: Питер, 2013. – 447.
2. Мейер, Э. CSS-каскадные таблицы стилей. Подробное руководство / Э. Мейер – СПб.: Питер, 2015 – 576с.
3. Никсон, Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон – 4-е изд. – СПб.:Питер, 2018. – 768.
4. Фримен, Эл. Изучаем HTML, XHTML и CSS / Эл. Фримен, Эр.Фримен – СПб.: Питер, 2014 – 720с.
5. Закас, Н. JavaScript для профессиональных веб-разработчиков / Н. Закас – СПб.: Питер, 2015 – 960с.
6. Крокфорд, Д. JavaScript: сильные стороны / Д. Крокфорд – СПб.: Питер, 2013 – 176с.
7. Робсон, Э. Изучаем программирование на JavaScript / Э. Робсон, Э. Фримен – СПб.: Питер, 2015 – 640с.
8. Фленаган, Д. JavaScript. Карманный справочник / Д. Фленаган – СПб.: Питер, 2015 – 320с.
9. Дронов, В. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. Дронов – СПб.: БХВ-Петербург, 2014 – 416с.