Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Севастопольский государственный университет»

Методические указания к лабораторным работам по дисциплине «Веб-технологии» для студентов дневной и заочной форм обучения направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика»

Часть 1

Севастополь
2020

УДК 004.42 (076.5)
ББК 32.973-018я73
М54

Методические указания к лабораторным работам по дисциплине «Веб-технологии» для студентов дневной и заочной форм обучения направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика», часть 1 / Сост. А.Л. Овчинников, Д.В. Степаненко, А.К. Забаштанский. – Севастополь: Изд-во СевГУ, 2020. – 120 с.

Методические указания предназначены для проведения лабораторных работ и обеспечения самостоятельной подготовки студентов по дисциплине «Веб-технологии». Целью методических указаний является облегчение изучения студентами основ веб-технологий и приобретения практических навыков создания веб-сайтов и веб-ориентированных систем.

Методические указания составлены в соответствии с требованиями программы дисциплины «Веб-технологии» для студентов направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика» и утверждены на заседании кафедры информационных систем протокол № ___ от _____ 20__ года.

Допущено научно-методической комиссией института информационных технологий и управления в технических системах в качестве методических указаний.

Рецензент: Альчаков В.В., к.т.н., доцент кафедры ИиУТС, СевГУ

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения настоящих лабораторных работ: приобретение практических навыков создания WEB-страниц с использованием языка HTML и каскадных таблиц стилей CSS, содержащих HTML-формы для реализации обратной связи с сервером; создания динамических WEB-страниц с использованием языка программирования JavaScript и библиотеки jQuery.

Выбор вариантов и график выполнения лабораторных работ

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены к каждой лабораторной работе и уточняются преподавателем.

Лабораторная работа выполняется в два этапа. На первом этапе – этапе самостоятельной подготовки – студент должен выполнить следующее:

- проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы;
- написать программный код, реализующий поставленную перед студентом задачу;
- оформить результаты первого этапа в виде заготовки отчета по лабораторной работе (**студенты-заочники** первый этап выполнения лабораторных работ оформляют в виде контрольной работы, которую сдают на кафедру до начала зачетно-экзаменационной сессии).

На втором этапе, выполняемом в лабораториях кафедры, студент должен:

- выполнить отладку кода программы;
- разработать и выполнить тестовый пример.

Выполнение лабораторной работы завершается защитой отчета. Студенты должны выполнять и защищать работы **строго по графику.** График защиты лабораторных работ студентами дневного отделения:

- лабораторная работа №1 – 3-ая неделя осеннего семестра;
- лабораторная работа №2 – 5-ая неделя осеннего семестра;
- лабораторная работа №3 – 8-ая неделя осеннего семестра;
- лабораторная работа №4 – 10-ая неделя осеннего семестра;
- лабораторная работа №5 – 12-ая неделя осеннего семестра;
- лабораторная работа №6 – 14-ая неделя осеннего семестра;

- лабораторная работа №7 – 16-ая неделя осеннего семестра.

График уточняется преподавателем, ведущим лабораторные занятия.

Описание лабораторной установки

Для выполнения лабораторных работ студенту потребуется персональный компьютер с установленной операционной системой (Windows, Linux или MacOS) и следующее программное обеспечение:

- один из современных браузеров (Google Chrome, Mozilla Firefox, Opera и тп);
- редактор программного кода (например, Sublime Text[1], Visual Studio Code[2] и тп).

Для выполнения лабораторных работ №5-7 также рекомендуется использование компилятора Prepros[3].

Требования к оформлению отчета

Отчёты по лабораторной работе оформляются каждым студентом индивидуально. Отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановку задачи; тексты разработанных программных модулей с комментариями; графическое отображение результатов проделанной работы; выводы по работе; приложения.

Постановка задачи представляет собой изложение содержания задачи и цели её решения. Также на этом этапе должны быть полностью определены исходные данные, выбраны методы решения и дана характеристика предполагаемых результатов.

В приложении могут приводиться тексты программных модулей, результаты работы программных модулей с пояснениями, тестовые примеры и наборы данных.

Содержание отчета указано в методических указаниях к каждой лабораторной работе.

1 ЛАБОРАТОРНЯ РАБОТА №1

«Исследование возможностей языка разметки гипертекстов HTML и каскадных таблиц стилей CSS»

1.1 Цель работы

Исследовать особенности структуры HTML-документа. Изучить основные понятия языка разметки гипертекстов HTML и каскадных таблиц стилей CSS. Приобрести практические навыки реализации Web-страниц с использованием гиперссылок, нумерованных и маркированных списков, графических элементов, таблиц и форм ввода.

1.2 Краткие теоретические сведения

1.2.1 Язык разметки гипертекста HTML

Язык разметки гипертекста – *HyperText Markup Language* (HTML) является языком, предназначенным для создания гипертекстовых документов. HTML-документы могут просматриваться различными типами WEB-браузеров. Когда документ создан с использованием HTML, WEB-браузер может интерпретировать HTML для выделения различных элементов документа и первичной их обработки. Использование HTML позволяет форматировать документы для их представления с использованием различных шрифтов, линий и других графических элементов на любой системе, их просматривающей.

В процессе развития HTML Консорциум Всемирной паутины W3C (англ. World Wide Web Consortium) – организация, разрабатывающая и внедряющая технологические стандарты для Всемирной паутины – опубликовала следующие варианты рекомендаций:

- RFC 1866 – HTML 2.0, одобренный как стандарт 22 сентября 1995 года;
- HTML 3.2– 14 января 1997 года;
- HTML 4.0– 18 декабря 1997 года;
- HTML 4.01– 24 декабря 1999 года;
- ISO/IEC 15445:2000 (так называемый ISO HTML, основан на HTML 4.01 Strict) – 15 мая 2000 года.
- HTML 5 – до настоящего времени находится в разработке.

Большинство документов в сети Интернет имеют стандартные элементы, такие, как заголовки, параграфы, нумерованные и маркированные списки и т.п. Используя специальные ***теги*** HTML, возможно обозначать данные элементы, обеспечивая WEB-браузеры минимальной информацией для отображения данных элементов, сохраняя в целом общую структуру и информационную полноту документов. Все что необходимо, чтобы отобразить HTML-документ – это WEB-браузер, который интерпретирует теги

HTML и воспроизводит на экране документ в виде, который ему придает разработчик.

HTML-теги могут быть условно разделены на две категории:

- теги, определяющие, как будет отображаться WEB-браузером содержимое документа;
- теги, описывающие общие свойства документа, такие как заголовок или автор документа;

Одно из преимуществ HTML заключается в том, что документ может быть просмотрен на WEB-браузерах различных типов и на различных платформах.

HTML-документы могут быть созданы при помощи любого текстового редактора или специализированных HTML-редакторов и конвертеров.

1.2.1.1 Основные понятия языка

Все теги HTML начинаются с "<" (левой угловой скобки) и заканчиваются символом ">" (правой угловой скобки). Как правило, существует стартовый тег и завершающий тег. Для примера приведем теги заголовка, определяющие текст, описывающий заголовок документа:

<TITLE> Заголовок документа </TITLE>

Завершающий тег выглядит так же, как и стартовый, но отличается от него прямым слешем перед текстом внутри угловых скобок. В данном примере тег <TITLE> говорит WEB-браузеру о начале заголовка, а тег </TITLE> – о завершении текста заголовка.

Некоторые теги, такие, как <P> (тег, определяющий абзац), не требуют завершающего тега, но его использование придает исходному тексту документа улучшенную читаемость и структурируемость.

HTML не реагирует на регистр символов, описывающих тег, и приведенный ранее пример может выглядеть следующим образом:

<title> Заголовок документа </title>

Внимание! *Дополнительные пробелы, символы табуляции и возврата каретки, добавленные в исходный текст HTML-документа для его лучшей читаемости, будут проигнорированы WEB-браузером при интерпретации документа. HTML-документ может включать вышеописанные элементы только если они помещены внутрь тегов <PRE> и </PRE>*.

Многие из открывающих HTML-тегов могут содержать дополнительные параметры, называемые *атрибутами*, которые могут иметь значения (стандартные или устанавливаемые авторами или сценариями). Пары *атрибут=значение* помещаются перед закрывающей скобкой ">" начального тэга элемента. В начальном тэге элемента может быть любое число (допустимых) пар *атрибут=значение*, разделенных пробелами. Они могут указываться в любом порядке. Например:

```
<P align=left >
```

1.2.1.2 Структура документа

Когда WEB-браузер получает документ, он определяет, как документ должен быть интерпретирован.

Для корректного отображения Web-страницы в браузере желательно указать, с каким стандартом она согласована. Для этого необходимо указать Определение типа документа (DTD – Document Type Definition) с использованием декларации DOCTYPE, которая должна предшествовать любой разметке в документе. В спецификации HTML5 декларация DOCTYPE существенно упрощена и имеет следующий вид:

<!DOCTYPE HTML>

Декларация должна находиться в самом начале HTML-документа, перед тегом <HTML>. (С примерами старых вариантов декларации DOCTYPE можно ознакомиться в документации соответствующей версии языка HTML).

Самым первый тегом, который встречается в документе, должен быть тег <HTML>.

Тег заголовочной части документа <HEAD> должен быть использован сразу после тега <HTML> и более нигде в теле документа. Данный тег содержит общее описание документа. Стартовый тег <HEAD> помещается непосредственно перед тегом <TITLE> и другими тегами, описывающими (см. ниже), а завершающий тег </HEAD> размещается сразу после окончания описания документа.

Спецификация HTML5 содержит серию новых семантических элементов, используемых для придания определенного смысла различным секциям или частям Web-страницы, таким как заголовок, нижний колонтитул, навигационная панель и т.д. В предыдущих версиях HTML для создания этих частей веб-страницы обычно применялись элементы с использованием различных стилей (идентификаторов или классов) для их идентификации.

Проблема при таком подходе состоит в отсутствии какого-либо семантического смысла, поскольку нет никаких строго определенных правил, которые специфицировали бы, какие имена классов или идентификаторы должны использоваться. Все это чрезвычайно затрудняет программному обеспечению определение того, что делает определенная область. HTML5 смягчает эти проблемы, упрощая Web-браузерам разбор семантической структуры документа.

Необходимо отметить следующий момент. Использование элементов в HTML5 по-прежнему является допустимым, однако рекомендуется в максимально возможной степени применять семантические элементы – это повысит долговечность разработки. С другой стороны, рекомендуется избегать использования новых элементов в несвойственных целях. К примеру, элемент <nav> предназначен для окружения главного навигационно-

го блока на странице и не должен использоваться для каких-либо иных групп ссылок.

Главные семантические элементы, введенные в HTML5:

- <header> – с помощью этого элемента задается заголовок для некоторой части Web-страницы, для всей страницы, для элемента <article> или для элемента <section>.
- <footer> – подобно элементу <header>, этот новый элемент задает нижний колонтитул для определенной части страницы. Нижний колонтитул не обязательно должен присутствовать в конце страницы, статьи или раздела, однако обычно он есть.
- <nav> – это контейнер для основных навигационных ссылок на Web-странице. Данный элемент не предназначен для использования со всеми группами ссылок и должен применяться только для главных навигационных блоков. Если элемент <footer> содержит навигационные ссылки, нет необходимости включать эти ссылки в элемент <nav>, поскольку элемент <footer> вполне самодостаточен.
- <article> – используется для задания на странице независимого элемента, который может быть размещен сам по себе. Примеры: новостное сообщение, заметка блога или комментарий. Как правило, такие элементы синдицируются с помощью RSS-потока.
- <section> – задает раздел документа или приложения (глава/раздел статьи или учебного пособия и т.д.). К примеру, данный раздел мог бы быть окружен элементом <section> в HTML5. Обычно элементы <section> имеют заголовок, хотя это не строгое требование. К примеру, заголовок для раздела, который вы сейчас читаете, содержал бы текст "Семантические элементы".
- <aside> – этот элемент может использоваться для разметки боковой панели или какого-либо другого контента, который должен быть отделен от окружающего его контента. Характерный пример – рекламные блоки.

Ниже приведен пример HTML документа с семантическими элементами:

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title> Пример HTML документа </title>
 </head>
 <body>
  <header>
    <h1>Пример HTML 5 документа</h1>
  </header>
```

```
<nav>
  <ul>
    <li><a href="#">Главная</a></li>
    <li><a href="#">О нас</a></li>
    <li><a href="#">Связаться с нами</a></li>
  </ul>
</nav>
<article>
  <header>
    <time datetime="2020-01-12">
      <span>Jan</span> 12
    </time>
    <h1>
      <a href="#" rel="bookmark">Заголовок статьи</a>
    </h1>
  </header>
  <p>Этот документ показывает как работает HTML 5</p>
  <section>
    <header>
      <h1>Это заголовок секции</h1>
    </header>
    <p>Это документ секции</p>
  </section>
</article>
<footer>
  <p>&copy; 2020 Имя компании.</p>
</footer>
</body>
</html>
```

1.2.1.3 Основные контейнеры заголовка

Основные контейнеры заголовка – это элементы HTML-разметки, которые наиболее часто встречаются в заголовке HTML-документа, т.е. внутри элемента разметки HEAD:

- TITLE (заглавие документа);
- BASE (база URL);
- META (метаинформация);
- LINK (общие ссылки);
- STYLE (описатели стилей);
- SCRIPT (скрипты).

Элемент разметки TITLE служит для именования WEB-страницы. Синтаксис контейнера TITLE в общем виде выглядит следующим образом:

<TITLE>название документа</TITLE>

Элемент разметки BASE служит для определения базового URL (см. п. 1.2.1.7) для гипертекстовых ссылок документа, заданных в неполной (частичной) форме.

В общем случае контейнер LINK имеет следующий вид:

```
<LINK [REL=тип_отношения] [HREF=URL]
    [TYPE=тип_содержания]>
```

Для разных типов содержания действия по интерпретации этого элемента разметки будут различными.

Элемент разметки STYLE предназначен для размещения описателей стилей (см. п. 1.2.2.1). В общем виде запись элемента STYLE выглядит так:

```
<STYLE TYPE=тип_описания_стилей>
    описание стиля/стилей
</STYLE>
```

Элемент разметки SCRIPT служит для размещения кода JavaScript (по умолчанию), VBScript или JScript в теле документа. В общем виде запись контейнера выглядит следующим образом:

```
<SCRIPT [TYPE=тип_языка_программирования]>
    JavaScript/VBScript-код
</SCRIPT>
```

или

```
<SCRIPT [TYPE=тип_языка_программирования]
    [SRC=URL]>
</SCRIPT>
```

META-тег HTML – это элемент разметки html, описывающий свойства документа как такового (метаданные). Основное предназначение META-тегов, это включение информации о документе, которая может содержать сведения об авторе, дате создания документа или авторских правах. Вся информация, находящаяся в META-теге, полезна для серверов, браузеров и поисковых роботов. Для посетителя веб-страницы информация, которую несут в себе META-теги, видна не будет.

На сегодняшний день значимость мета-тегов несколько уменьшилась из-за действий рекламных агентов и специалистов по раскрутке сайтов – в эти разделы стали помещать рекламную или недостоверную информацию, что снизило степень доверия к данному разделу сайта со стороны владельцев серверов и поисковых систем.

Назначение мета-тега определяется набором его атрибутов, которые задаются в теге <meta>. В документе может находиться любое количество тегов <meta>. Они не являются обязательными элементами, но могут быть весьма полезны. Некоторые, часто используемые META-теги приведены ниже:

Таблица 1.1 – Примеры использования тега <meta>

<meta charset="UTF−8">	Для перекодировки на стороне клиента в заголовок документа необходимо включить META-тег следующего вида (документ подготовлен в кодировке UTF-8):
<META HTTP-EQUIV="Refresh" CONTENT="n; URL=http://newadres.ru/">	Автоматическое перенаправление (через n секунд) на указанный адрес (редирект).
<META NAME="description" http-equiv="description" content="Описание содержания документа.">	Этот тег задает фразу, по которой пользователь определяет суть вашей страницы и решает, посещать ли ее. Содержимое данного META тега может использоватться поисковыми системами при ранжировании страницы. Длина содержимого тегов META "description" не должна превышать 200 символов.
<META NAME="keywords" HTTP-EQUIV="keywords" CONTENT="список ключевых слов">	В мета-теге keywords указываются ключевые слова и их синонимы, присутствующие в документе. Этот тег изначально был ориентирован на поисковые машины, но был скомпрометирован веб-мастерами, использовавшими его для поискового спама.

Внимание! *Традиционно для WINDOWS кодировкой по умолчанию является windows-1251 (cp1251), и все текстовые файлы хранятся именно в этой кодировке. Однако настоятельно рекомендуется при выполнении данных лабораторных работ для исходных файлов (HTML, JS, CSS и PHP) использовать кодировку UTF-8, которая является стандартом де-факто для Linux окружения и мультиязычных сайтов*.

1.2.1.4 HTML комментарии

Как любой язык, HTML позволяет вставлять в тело документа комментарии, которые сохраняются при передаче документа по сети, но не отображаются браузером. Синтаксис комментария:

```
<!-- Это комментарий -->
```

Комментарии могут встречаться в документе где угодно и в любом количестве.

1.2.1.5 Escape-последовательности

Некоторые символы являются управляющими символами в HTML и не могут напрямую использоваться в документе. Если возникает необхо-

димость использовать подобные символы в документе, их замняют на escape-последовательности:

- левая угловая скобка: "<" на "**<**";
- правая угловая скобка: ">" на "**>**";
- амперсанд: "&" на "**&**";
- двойные кавычки: """ на "**"**".

Существует большое количество escape-последовательностей для обозначения специальных символов, например "©" для обозначения знака ©, "®" для значка ® и тп.

Внимание! *Escape-последовательности чувствительны к регистру: НЕЛЬЗЯ использовать "<" вместо "<"!!*

1.2.1.6 Теги тела документа

Теги тела документа идентифицируют отображаемые в окне компоненты HTML-документа. Тело документа может содержать ссылки на другие документы, текст и другую форматированную информацию.

Примеры часто используемых тегов тела документа:

... – контейнер общего назначения (структурный блок)
<hN>...</hN> – заголовок N-ного уровня (N = 1...6)
... – основной текст
<pre>...</pre> – преформатирование
<a>... – гиперссылка
... – нумерованный список
... – маркированный список
... – элемент списка
<table>...</table> – контейнер таблицы
<tr>...</tr> – строка таблицы
<td>...</td> – ячейка таблицы
<img>...</img> – изображение
<form>...</form> – форма
... – выделение (курсивом)
... – усиление (полужирным шрифтом)

 – принудительный разрыв строки
<hr> – горизонтальная линия

Теги могут быть вложены, при этом форматирование внутреннего тега имеет преимущество перед внешним. При использовании вложенных тегов их нужно закрывать, начиная с самого последнего и двигаясь к первому:

```
<!-- Примеры использования вложенных тегов -->
<ol>
  <li>Элемент списка</li>
  <li>Второй элемент списка</li>
</ol>
```

```
<div>
  <h2>Заголовок второго уровня</h2>
  <p>и основной текст</p>
  внутри логического блока
</div>
```

Полный список тегов можно найти в документации на соответствующую версию языка HTML

1.2.1.7 Гипертекстовые ссылки

Гипертекстовые ссылки являются одним из ключевых элементов, делающим WEB привлекательным для пользователей. Гипертекстовые ссылки (далее – ссылки) делают набор документов связанным и структурированным, что позволяет пользователю получать необходимую ему информацию максимально быстро и удобно.

Ссылки имеют стандартный формат, что позволяет браузеру интерпретировать их и выполнять необходимые функции (вызывать методы) в зависимости от типа. Ссылки могут указывать на другой документ, специальное место данного документа или выполнять другие функции, например, запрашивать файл по FTP-протоколу для отображения его браузером.

Каждый ресурс в сети – документ HTML, изображение, программа и т.д. – имеет свой адрес, который может быть закодирован с помощью URI *(Universal Resource Identifier) – универсального идентификатор ресурса*.

Примечание: *более распространенной является аббревиатура "URL"(Uniform Resource Locator). Следует отметить, что URL образуют подмножество более общей схемы наименования URI, но в дальнейшем будем использовать аббревиатуру URL.*

Следующий пример представляет собой вызов HTML-документа `index.html` с сервера `www.is.sevsu.ru` с использованием HTTP протокола:

`http://www.is.``sevsu.ru`*`/index.html`*

Полный формат URL представлен ниже:

METHOD://SERVERNAME:PORT/PATHNAME#ANCHOR

Опишем каждый из компонентов URL.

METHOD определяет тип операции, которая будет выполняться при интерпретации данного URL. Наиболее часто используемые методы:

- file – чтение файла с локального диска, используется для отображения какого-либо файла, находящегося на машине пользователя (например: `file:/home/alex/index.html` – отображает файл `index.html` из каталога `/home/alex` на пользовательской машине);
- http – доступ к WEB-странице в сети с использованием HTTP-протокола (это наиболее часто используемый метод доступа к какому-либо HTML-документу в сети);

• ftp – запрос файла с анонимного FTP-сервера (например: `ftp://hostname/directory/filename);`

• mailto – активизирует почтовую сессию с указанным пользователем и хостом (например: `mailto: webmaster@is.sevsu.sebastopol.ru` – активизирует сессию посылки сообщения пользователю `webmaster` на машине `is.sevsu.sebastopol.ru`, если браузер поддерживает запуск электронной почты), также не требует указания слешей после двоеточия (как правило, после двоеточия сразу идет электронный адрес абонента).

SERVERNAME – необязательный параметр, описывающий полное сетевое имя машины. Если имя сервера не указано, то ссылка считается локальной, и полный путь, указанный далее в URL, вычисляется на той машине, с которой взят HTML-документ, содержащий данную ссылку. Следует отметить, что вместо символьного имени машины может быть использован IP-адрес.

PORT – номер порта TCP, на котором функционирует WEB-сервер. Если порт не указан, то "по умолчанию" используется порт 80. Данный параметр не используется в подавляющем большинстве URL.

PATHNAME – частичный или полный путь к документу, который должен вызваться в результате интерпретации URL. Различные WEB-сервера сконфигурированы по разному для интерпретации пути доступа к документу. Например, при использовании CGI-скриптов (исполняемых программ), они обычно собираются в одном или нескольких выделенных каталогах, путь к которым записан в специальных параметрах WEB-сервера. Для данных каталогов WEB-сервером выделяется специальный логический путь, который и используется в URL. Если WEB-сервер видит данный путь, то запрашиваемый файл интерпретируется как исполняемый модуль. В противном случае, запрашиваемый файл интерпретируется просто как файл данных, даже если он является исполняемым модулем. Следует отметить, что при описании пути используется UNIX-подобный синтаксис, где, в отличии от DOS и Windows, используются прямые слеши вместо обратных. Если после сетевого имени машины сразу идет имя документа, то он должен находиться в корневом каталоге на удаленной машине или (как правило) в каталоге, выделенном WEB-сервером в качестве корневого. Если же URL заканчивается сетевым именем машины, то в качестве документа запрашивается документ из корневого каталога удаленной машины с именем, установленным в настройках WEB-сервера (как правило, это `index.html`).

#ANCHOR – данный элемент является ссылкой на строку (точку) внутри HTML-документа. Большинство браузеров, встречая после имени документа данный элемент, размещают документ на экране таким образом, что указанная строка документа помещается в верхнюю строку рабочего

окна браузера. Точки, на которые ссылается #anchor, указываются в документе при помощи тега NAME, как это будет описано далее.

Для того, чтобы браузер отобразил ссылку на URL в HTML-документе, необходимо отметить URL следующим образом:

<A HREF="URL"> текст-который-будет-ссылкой</A>

Любой текст, находящийся внутри тега <A>, подсвечивается специальным образом Web-браузером. Обычно этот текст отображается подчеркнутым и выделяется синим (или другим заданным пользователем) цветом. Непосредственно сам URL, не отображается браузером в документе (может отображаться в строке статуса браузера), а используется только для выполнения предписанных им действий при активизации ссылки (обычно при щелчке мыши на подсвеченном или подчеркнутом тексте). Вот пример ссылки:

<A HREF="www.is.sevsu.ru/webprogramming/first.html">
Ваша первая Web-страница </A>

Если вместо атрибута *HREF* тега <A> использовать атрибут *NAME*, то ссылка будет работать в рамках одного и того же документа. Подобная ссылка называется *якорем* и позволяет быстро переходить от одного раздела к другому внутри документа, не используя скроллирование экрана, что бывает очень удобно. Для этого необходимо создать якорь (невидимую метку в теле документа), используя следующий синтаксис:

<A NAME="named_anchor"> Текст-который-
отобразится-в-первой-строке-браузера </A>

А для создания ссылки на этот якорь из этой же страницы в атрибуте HREF ссылки необходимо указать имя якоря после символа #:

<A HREF="#named_anchor"> Текст </A>

Якорь может быть поставлен как в том же документе, который просматривается в текущий момент, так и в другом документе. Во втором случае в ссылке на этот якорь необходимо перед «#» указать полный URL документа. Тогда браузер осуществит загрузку указанного документа и перейдет к указанному якорем разделу. Например:

<A HREF="www.is.sevsu.ru/webprogramming/second.html#anchor">

1.2.1.8 Графика в HTML-документах

Внедрение графических образов в документ позволяет пользователю видеть изображения непосредственно в контексте других элементов документа. Это наиболее часто используемая техника при проектировании документов, называемая иногда "*inline image*".

Для внедрения изображения в HTML-документ используется тег <IMG>. Синтаксис тега:

<IMG SRC="*URL*" ALT="*text*" HEIGHT=n1 WIDTH=n2
ALIGN=*top*|*middle*|*bottom*|*texttop* ISMAP>

Здесь:

SRC – обязательный параметр, имеющий такой же синтаксис, как и стандартный URL. Данный URL указывает браузеру, где находится рисунок. Рисунок должен храниться в графическом формате, поддерживаемом браузером (например, GIF или JPG).

ALT="text" – данный необязательный атрибут задает текст, который будет отображен браузером, не поддерживающим отображение графики или с отключенным отображением изображений. Обычно это короткое описание изображения, которое пользователь сможет увидеть на экране в виде всплывающей подсказки.

HEIGTH=n1 – данный необязательный атрибут используется для указания высоты рисунка в пикселях. Если данный атрибут не указан, то используется оригинальная высота рисунка. Этот параметр позволяет сжимать или растягивать изображения по вертикали, что позволяет более четко определять внешний вид документа. Однако некоторые браузеры не поддерживают данный атрибут.

WIDTH=n2 – атрибут также необязателен, как и предыдущий. Позволяет задать абсолютную ширину рисунка в пикселях.

Также следует учесть, что применение атрибутов указания ширины и высоты изображения может формировать дополнительную (и значительную) нагрузку на процессор устройства.

Большинство браузеров позволяет включать в документ фоновый рисунок, на фоне которого будет отображаться весь документ. Описание фонового рисунка включается в тег `BODY` и выглядит следующим образом:

```
<BODY BACKGROUND="picture.gif">
```

В настоящее время фоновые рисунки задаются в основном с использованием каскадных таблиц стилей (см. п. 1.2.2).

1.2.1.9 Таблицы в HTML

Таблицы в HTML организуются как набор строк и столбцов. Ячейки таблицы могут содержать любые HTML-элементы, такие, как заголовки, списки, абзацы, фигуры, графику, а также элементы форм. Таким образом, можно использовать таблицы для равномерного размещения элементов на странице.

Все элементы таблицы должны находиться внутри двух тегов <TABLE>...</TABLE>. По умолчанию таблица не имеет обрамления и разделителей. Обрамление добавляется атрибутом BORDER.

Строка таблицы задается парой тегов <TR>...</TR>. Количество строк таблицы определяется количеством встречающихся пар тегов <TR>...</TR>. Строки могут иметь атрибуты ALIGN и VALIGN, которые описывают визуальное положение содержимого строк в таблице (соответственно, расположение по горизонтали и по вертикали).

Пара тегов <TD>...</TD> описывает стандартную ячейку таблицы. Ячейка таблицы может быть описана только внутри строки таблицы. Если в строке отсутствует одна или несколько ячеек для некоторых колонок, то браузер отображает пустую ячейку. Расположение данных в ячейке по умолчанию определяется атрибутами ALIGN=left и VALIGN=middle. Данное расположение может быть изменено как на уровне описания строки, так и на уровне описания ячейки.

Ячейка заголовка таблицы определяется тегами <TH>...</TH> и имеет ширину всей таблицы; текст в данной ячейке имеет атрибут BOLD и ALIGN=center.

Теги <CAPTION>...</CAPTION> описывают название таблицы (подпись). Тег <CAPTION> может присутствовать внутри <TABLE>...</TABLE>, но снаружи описания какой-либо строки или ячейки. Атрибут ALIGN определяет, где (сверху или снизу таблицы) будет поставлена подпись. По умолчанию <CAPTION> имеет атрибут ALIGN=top, но может быть явно установлен в ALIGN=bottom. Подпись всегда центрирована в рамках ширины таблицы.

Ниже перечислены основные атрибуты, используемые при формировании таблиц в HTML:

BORDER – данный атрибут используется в теге TABLE. Если данный атрибут присутствует, граница таблицы прорисовывается для всех ячеек и для таблицы вцелом. BORDER может принимать числовое значение, определяющее ширину границы, например BORDER=3.

NOWRAP – данный атрибут говорит о том, что данные в ячейке не могут логически разбиваться на несколько строк и должны быть представлены одной строкой.

COLSPAN – указывает, какое количество ячеек будет объединено по горизонтали для указанной ячейки. По умолчанию – 1.

ROWSPAN – указывает, какое количество ячеек будет объединено по вертикали для указанной ячейки. По умолчанию – 1.

COLSPEC – данный параметр позволяет задавать фиксированную ширину колонок либо в символах, либо в процентах, например COLSPEC="20%".

CELLPADDING – данный атрибут определяет ширину пустого пространства между содержимым ячейки и ее границами, то есть задает поля внутри ячейки.

CELLSPACING – определяет ширину промежутков между ячейками в пикселях. Если этот атрибут не указан, по умолчанию задается величина, равная двум пикселям.

Примечание*: для задания различных параметров таблиц рекомендуется использование таблиц стилей.*

Пример таблицы с использованием стилей:

```
<BODY>
  <STYLE REL=STYLESHEET TYPE="text/css">
      .title {font-weight: bold; text-align: left}
      .data {font-weight: italic; text-align: center}
  </STYLE>
  <TABLE BORDER=1>
     <CAPTION> Таблица №1 </CAPTION>
      <TR CLASS=title>
          <TD ROWSPAN=2></TD>
          <TH COLSPAN=2>Среднее значение</TH>
      </TR>
     <TR CLASS=title>
         <TH>Рост, см</TH>
         <TH>Вес, кг</TH>
     </TR>
     <TR>
         <TD CLASS=title>Юноши</TD>
         <TD CLASS=data>174</TD>
         <TD CLASS=data>78</TD>
     </TR>
     <TR>
         <TD CLASS=title>Девушки</TD>
         <TD CLASS=data>165</TD>
         <TD CLASS=data>56</TD>
     </TR>
  </TABLE>
</BODY>
```

1.2.2 Каскадные таблицы стилей CSS

Каскадные таблицы стилей – CSS (Cascading Style Sheets) позволяют управлять размером и стилем шрифтов, интервалами между строками текста, отступами, цветами, используемыми для текста и фона и многими другими параметрами отображения HTML-документов.

Основным понятием CSS является *стиль* – т. е. набор правил оформления и форматирования, который может быть применен к различным элементам страницы. В стандартном HTML для присвоения какому-либо элементу определенных свойств (таких, как цвет, размер, положение на странице и т. п.) приходилось каждый раз описывать эти свойства. Кроме того, до появления таблиц стилей возможности управления внешним видом документов у авторов были весьма ограничены.

При использовании CSS для присвоения какому-либо элементу определенных характеристик необходимо один раз описать этот элемент и определить это описание как стиль, а в дальнейшем просто указывать, что элемент, который нужно оформить соответствующим образом, должен принять свойства описанного стиля.

Более того, возможно сохранить описание стиля не в тексте самой HTML-страницы, а в отдельном файле – это позволит использовать описа-

ние стиля в любом количестве Web-страниц. И еще одно, связанное с этим, преимущество – возможность изменить оформление любого количества страниц, исправив лишь описание стиля в одном (отдельном) файле.

Кроме того, CSS позволяет работать с оформлением страниц на гораздо более высоком уровне, чем стандартный HTML, избегая излишнего увеличения размера страниц графическими элементами.

1.2.2.1 Размещение информации о стилях

Как отмечалось выше, информация о стилях может располагаться либо в отдельном файле, либо в заголовочной части Web-страницы, либо непосредственно внутри тега элемента.

Расположение описания стилей в отдельном файле имеет смысл в том случае, если планируется применять эти стили к большему, чем одна, количеству страниц. Для этого нужно:

- создать обычный текстовый файл и описать в нем с помощью языка CSS необходимые стили;
- разместить этот файл на Web-сервере;
- в Web-страницах, которые будут использовать стили из этого файла, нужно будет сделать ссылку на него.

Ссылки на таблицы стилей выполняются с помощью тега <LINK>, располагающегося внутри тега <HEAD>:

<LINK REL=STYLESHEET TYPE="text/css" HREF="URL">

Первые два атрибута этого тега являются зарезервированными именами, требующимися для того, чтобы сообщить браузеру, что на этой Web-странице будет использоваться CSS. Третий параметр – HREF= «URL» – указывает на файл, который содержит описания стилей. Этот параметр должен содержать либо относительный путь к файлу – в случае, если он находится на том же сервере, что и документ, из которого к нему обращаются – или полный URL («http://...») в случае, если файл стилей находится на другом сервере.

Во втором случае размещения информации о стилях, при котором описание стилей располагается внутри Web-страницы, внутри тега <BODY> используется тег <STYLE type="text/css">... </STYLE>. Атрибут *type="text/css"* является обязательным и служит для указания браузеру использовать CSS.

И третий вариант, когда описание стиля располагается непосредственно внутри тега элемента, для которого используется. Это делается с помощью атрибута STYLE, который может применяться с большинством стандартных тегов HTML. Этот метод нежелателен, и понятно почему: он приводит к потере одного из основных преимуществ CSS – возможности отделения информации от описания оформления информации. Впрочем, если необходимо описать лишь один элемент, этот вариант расположения описания стилей также вполне применим.

1.2.2.2 Использование каскадных таблиц стилей (CSS)

Формально стиль отображения элементов разметки задается ссылкой в элементе разметки на селектор стиля. Синтаксис описания стилей в общем виде представляется следующим образом:

```
selector[, selector[, ...]]
      { attribute:value;
      [atribute:value;...] }
или
selector selector [selector ...]
       { attribute:value;
       [atribute:value;...] }
```

В первом варианте перечислены селекторы, для которых действует данное описание стиля. Второй вариант задает иерархию вложенности селекторов, для совокупности которых определен стиль. Напомним, что речь в данном случае идет об описаниях стилей в нотации text/css. Описания стилей размещаются либо внутри элемента STYLE, либо во внешнем файле.

В качестве селектора можно использовать имя элемента разметки, имя класса и идентификатор объекта на HTML-странице.

Селектор – имя элемента разметки

Когда автор хочет определить общий стиль всех страниц, он просто создает стили для всех элементов HTML-разметки, которые будут использоваться на страницах. Это дает возможность скомпоновать страницы из логических элементов, а стиль отображения элементов описать во внешнем файле.

Такой способ создания сайта позволяет автору изменять внешний вид всех страниц путем внесения изменений в файл описания стилей, а не в файлы HTML-страниц.

Внешний файл при этом может выглядеть следующим образом:

```
I, EM { color:#003366,font-style:normal }
A I { font-style:normal;font-weight:bold;
    text-decoration:line-through }
```

В первой строке этого описания перечислены селекторы-элементы, которые будут отображаться одинаково:

```
<I>Это курсив</I> и это тоже <EM>курсив</EM>
```

Последняя строка определяет стиль отображения вложенного в гипертекстовую ссылку курсива:

```
<A NAME=empty><I>SevSU</I></A>
```

В данном случае переопределение состоит в том, что текст отображается внутри гипертекстовой ссылки перечеркнутым, причем жирным шрифтом.

Селектор – имя класса

Имя класса не является каким-либо стандартным именем элемента HTML-разметки. Оно определяет описание класса элементов разметки, которые будут отображаться одинаково. Для того, чтобы отнести элемент разметки к тому или иному классу, нужно воспользоваться его атрибутом CLASS:

```
<STYLE>
.test {color:white;background-color:black;}
</STYLE>
...
<P CLASS="test">
Этот параграф мы отобразим белым цветом по
черному фону
</P>
...
<P>
Эту <A CLASS="test">гипертекстовую ссылку</A>
мы отобразим белым цветом по черному фону.
</P>
```

Таким образом, в любом элементе разметки можно сослаться на описание класса отображения. При этом совершенно необязательно, чтобы элементы разметки были однотипными. В примере к одному классу отнесены и параграф, и гипертекстовая ссылка в другом параграфе. Но можно определить классы отображения и однотипных элементов разметки:

```
a.menu { color:red;background-color:white;
        text-decoration:normal; }
a.paragraph { color:navy;
        text-decoration:underline; }
```

В данном примере класс гипертекстовых ссылок menu имеет одно описание стиля, а класс гипертекстовых ссылок paragraph – совершенно другое. При этом каждый из этих классов нельзя применить к другим элементам разметки, например, параграфу или списку. Если имя элемента разметки не задано, это означает, что класс можно отнести к любому элементу разметки – корневой класс описания стилей.

Селектор – идентификатор объекта

Каждый из элементов HTML документа можно именовать с использованием уникального *идентификатора* – это позволит обращаться к нему по имени как для задания стилей, так и для доступа при программировании.

Описание стиля в этом случае задается строкой, в которой селектор представляет собой идентификатор этого объекта с лидирующим символом "#":

```
a.mainlink { color:darkred;
        text-decoration:underline;
        font-style:italic; }
#blue { color:#003366 }
...
<A CLASS=mainlink>основная гипертекстовая
ссылка</A>
<A CLASS=mainlink ID=blue>модифицированная
гипертекстовая ссылка</A>
```

Аттрибуты CSS

В таблице 1.2 представлены некоторые аттрибуты CSS, позволяющие изменять способы отбражения элементов HTML.

Таблица 1.2 – Атрибуты CSS

Наименование аттрибута	Описание
СВОЙСТВА ШРИФТА	
font-family	Используется для указания шрифта или шрифтового семейства, которым будет отображаться элемент. P {font-family: Times New Roman, sans-serif;}
font-weight	Определяет степень жирности шрифта с помощью трех параметров: lighter, bold, bolder B {font-weight: bolder;}
font-size	Устанавливает размер шрифта. Параметр может указываться как в относительной (проценты), так и абсолютной величине (пункты, пикселы, сантиметры) H1 {font-size: 200%;} H2 {font-size: 150px;}
ЦВЕТ ЭЛЕМЕНТА И ЦВЕТ ФОНА	
color	Определяет цвет элемента I {color: yellow;}
background-color	Устанавливает цвет фона для элемента – именно для элемента, а не для страницы. Обратите внимание, что браузеры отображают это свойство по-разному:
СВОЙСТВА ТЕКСТА	
text-decoration	Устанавливает эффекты оформления шрифта, такие, как подчеркивание или зачеркнутый текст H4 {text-decoration: underline;} A {text-decoration: none;} .wrong {text-decoration: line-through;}
text-align	Определяет выравнивание элемента. P {text-align: justify} H1 {text-align: center}
text-indent	Устанавливает отступ первой строки текста. Чаще всего используется для создания параграфов с табулированной первой строкой. P {text-indent: 50pt;}
line-height	Управляет интервалами между строками текста.

	P {line-height: 50 %}
СВОЙСТВА ГРАНИЦ	
margin-left	Устанавливают значения отступов вокруг элемента. IMG { margin-right: 20pt} P { margin-left: 2cm}
margin-right margin-right margin-top	Устанавливают значения отступов вокруг элемента. IMG { margin-right: 20pt} P { margin-left: 2cm}
ЕДИНИЦЫ ИЗМЕРЕНИЯ	
Px	Пикселы
Cm	Сантиметры
Mm	Миллиметры
Pt	Пункты (типограф.)
%	Проценты

1.2.2.3 Позиционирование элементов

Позиционированием называется положение элемента в системе координат. Различают четыре типа позиционирования: нормальное, абсолютное, фиксированное и относительное. В зависимости от типа, который устанавливается через свойство position, изменяется и система координат.

Благодаря комбинации свойств position, left, top, right и bottom элемент можно накладывать один на другой, выводить в точке с определёнными координатами, фиксировать в указанном месте, определить положение одного элемента относительно другого и др. Значения этих параметров могут определяться в процентном отношении или пикселах, принимать положительные и отрицательные величины. Подобно другим свойствам CSS управление позиционированием доступно через скрипты. Таким образом, можно динамически изменять положение элементов без перезагрузки страницы, создавая анимацию и различные эффекты.

Нормальное позиционирование

Если для элемента свойство position не задано или его значение static, элемент выводится в потоке документа как обычно. Иными словами, элементы отображаются на странице в том порядке, как они идут в исходном коде HTML. Свойства left, top, right, bottom игнорируются если определены.

Абсолютное позиционирование

При абсолютном позиционировании – свойство position имеет значение absolute – элемент не существует в потоке документа и его положение задаётся относительно окна браузера или первого родительского элемента со значением position:relative. Координаты указываются относительно границ окна браузера, называемого «видимой областью» (см. рисунок 1.1).

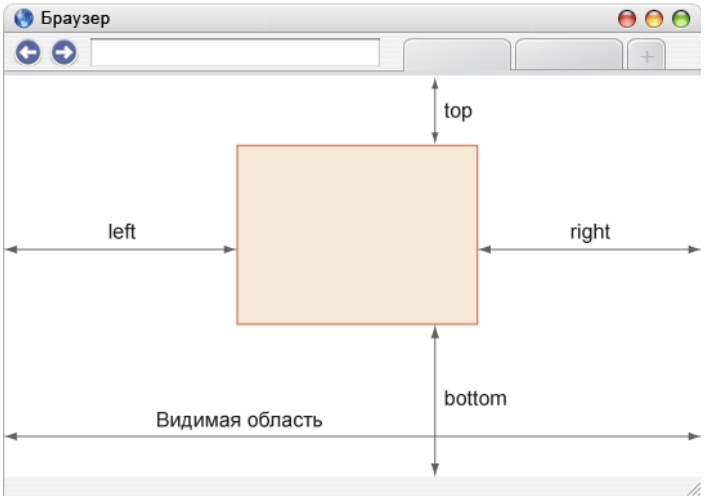

Рисунок 1.1 – Значения свойств left, right, top и bottom при абсолютном позиционировании

Фиксированное положение

Фиксированное положение слоя задаётся значением fixed свойства position и по своему действию похоже на абсолютное позиционирование. Но в отличие от него привязывается к указанной свойствами left, top, right и bottom точке на экране и не меняет своего положения при прокрутке веб-страницы. Ещё одна разница от absolute заключается в том, что при выходе фиксированного слоя за пределы видимой области справа или снизу от неё, не возникает полос прокрутки.

Применяется такой тип позиционирования для создания меню, вкладок, заголовков, в общем, любых элементов, которые должны быть закреплены на странице и всегда видны посетителю.

Относительное позиционирование

Если задать значение relative свойства position, то положение элемента устанавливается относительно его исходного места. Добавление свойств left, top, right и bottom изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения. Положительное значение left определяет сдвиг вправо от левой границы элемента, отрицательное – сдвиг влево. Положительное значение top задаёт сдвиг элемента вниз (см. рисунок 1.2), отрицательное – сдвиг вверх.

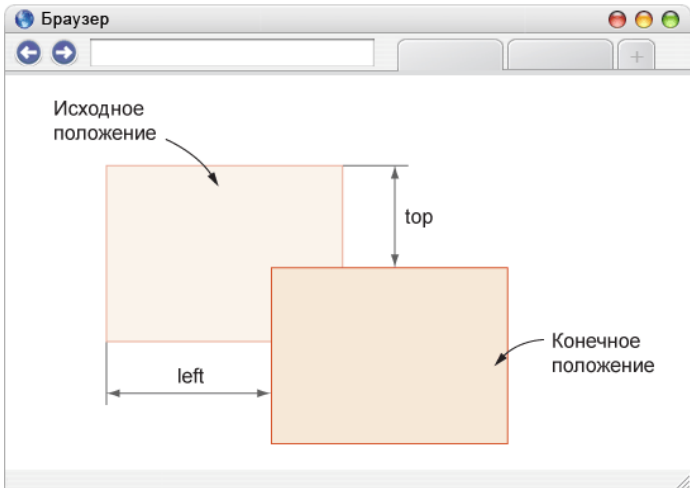

Рисунок 1.2 – Значения свойств left и top при относительном позиционировании.

1.2.2.4 Позиционирование с использованием свойства float

Часто бывает, что разработчику, использующему блочную верстку, необходимо разбить страницу или ее часть на вертикальные колонки. В этом случае можно воспользоваться методом абсолютного позиционирования, как было описано выше, но есть более правильный подход: использование свойства float.

Float говорит, что элемент, к которому оно было применено будет «обтекаем» с какой-либо стороны, в зависимости от значения.

Таблица 1.3 – Описание свойства float

Значение по умолчанию	none
Возможные значения	left | right | none
Наследуется	Нет
Применяется	Ко всем элементам, за исключением позиционированных

При использовании float желательно задавать также ширину элемента для достижения кроссбраузерности. Также, хорошей практикой считается оборачивать элементы, у которых задан float, отличный от none, в отдельный блок. Однако, необходимо учитывать следующее – высота родительского блока, содержащего элемент float будет равна 0, т.к. все элементы внутри него обтекаемы. К тому же, и сам родительский элемент станет обтекаемым, что может привести к нарушению позиционирования элементов, расположенных ниже данного. Самыми распространенными способами избежать этого являются использование свойства clear и сочетание свойств display и width.

Рассмотрим, например, следующую структуру страницы (рис. 1.3):

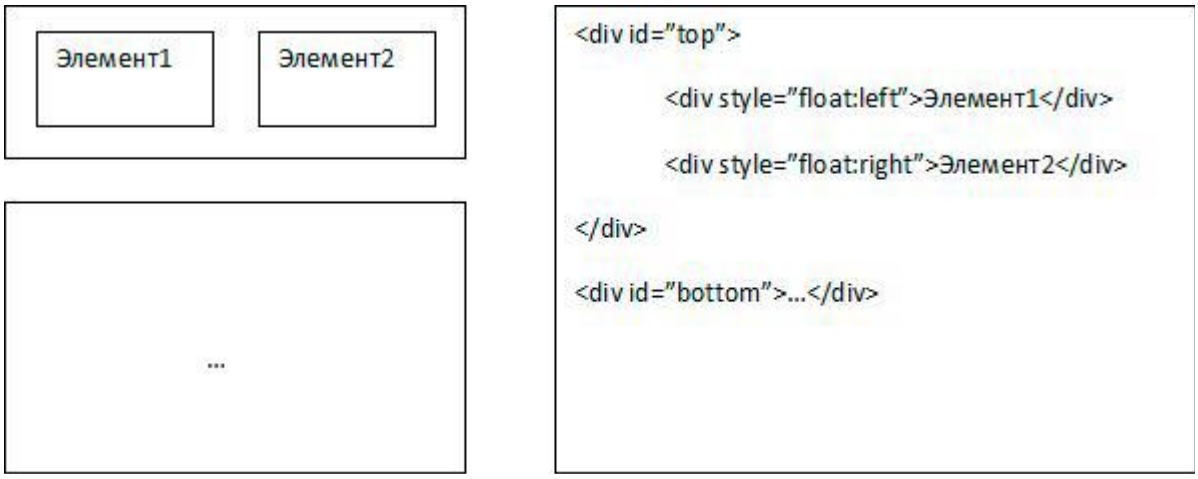

Рисунок 1.3 – Пример реализации колонок при блочной верстке

При попытке достичь такого результата с представленой верстской выйдет ситуация, описанная выше, высота блока top будет равна 0, и bottom будет обтекать top по правой стороне.

Первый способ – добавить еще один элемент div между блоками top и bottom и задать для него свойство «clear:both». В этом случае новый элемент как бы «очистит» свойство float для bottom, но top по-прежнему будет с нулевой высотой.

Второй способ решает и эту проблему. Для top задается стиль «display: inlineblock; width:100%;». Т.к. задается ширина 100%, bottom не сможет обтекать top, а на inline-block элементы не действует float.

1.2.2.5 Верстка HTML документов

Вёрсткой веб-страниц называют процесс формирования текста HTML -документа на основании графического изображения веб-страницы, либо описания ее внешнего вида. Существует несколько подходов к выполнению верстки веб-страниц, среди которых наиболее часто используются *табличная верстка* и *блочная верстка*.

Табличная верстка – условное название метода верстки HTML-документов, при котором в качестве структурной основы для расположения текстовых и графических элементов документа используются таблицы (HTML-тег <table>). В настоящее время использование этого метода не приветствуется.

Блочная верстка HTML страницы выполняется с помощью семантических элементов и тегов (см. ниже) с набором CSS стилей для них. Блочная верстка[4] имеет ряд преимуществ по сравнению с табличной верстской. Верстка блоками DIV обеспечивает более быструю загрузку страницы сайта, содержимое блоков DIV отображается по мере загрузки (в отличие от таблицы, содержимое которой не отобразится до тех пор, пока полностью не загрузится в браузер).

DIV играет роль универсального блока. Блочный элемент всегда отделен от прочих элементов страницы (контекста) пустой строкой. DIV не

несет никакой смысловой нагрузки. Часто говорят, что DIV – это раздел страницы. Применение элемента DIV имеет больший в контексте CSS. DIV позволяет применить атрибуты стиля, связанные с границей блока и отступами блока от границ старшего элемента, а также "набивку", т.е. отступ от границы блока до границы вложенного элемента.

1.2.2.6 Формы в HTML

Для организации обратной связи между пользователем и WEB-сервером в HTML используются формы.

Когда HTML-документ, содержащий форму, интерпретируется WEB-браузером, на экране браузер создает специальные экранные элементы ввода, такие, как поля ввода, checkboxes, radiobuttons, выпадающие меню, скроллируемые списки, кнопки и т.д. Когда пользователь заполняет форму и нажимает кнопку "Подтверждение" (SUBMIT – специальный тип кнопки, который задается при описании HTML-документа внутри формы), информация, введенная пользователем, посылается WEB-серверу для передачи указанному CGI-скрипту или отправки по E-mail.

Все теги формы помещаются между тегами <FORM> и </FORM>:

```
<FORM METHOD="GET | POST" ACTION="URL">
    Элементы_формы_и_другие_элементы_HTML
</FORM>
```

***Примечание**: здесь и далее «|» – означает «или». В данном случае METHOD="GET" или METHOD="POST".*

Отправка сообщения с данными из формы определяется атрибутом **METHOD**. В зависимости от используемого метода вы можете посылать результаты ввода данных в форму двумя путями:

- **GET**: Информация из формы добавляется в конец URL, который был указан в описании заголовка формы.
- **POST**: Данный метод передает всю информацию о форме в теле HTTP сообщения. Данный метод рекомендуется к использованию в большинстве случаев.

Атрибут **ACTION** описывает URL, который будет вызываться для обработки формы. Данный URL почти всегда указывает на CGI-программу, обрабатывающую данную форму. В том случае если необходимо отправить данные формы на электронный адрес, в атрибуте ACTION необходимо указать: «mailto: EMAIL», где EMAIL – электронный адрес для отправки. Например:

```
<FORM METHOD=post ACTION=mailto:yourname@your.email.address>
```

1.2.2.7 Теги формы

Тег <**TEXTAREA**> используется для того, чтобы позволить пользователю вводить многострочную текстовую информацию. Вот пример использования тега <TEXTAREA>:

```
<TEXTAREA NAME="IS" ROWS=10 COLS=50>
</TEXTAREA>
```

Атрибуты, используемые внутри тега <TEXTAREA>, описывают внешний вид и имя вводимого значения. Тег </TEXTAREA> необходим даже тогда, когда поле ввода изначально пустое. Описание основных атрибутов:

- **NAME –** имя поля ввода;
- **ROWS –** высота поля ввода в символах (количество строк ввода);
- **COLS** – ширина поля ввода в символах (количество символов в строке).

Если в поле ввода по умолчанию должен выводиться какой-либо текст, то его необходимо вставить внутри тегов <TEXTAREA> и </TEXTAREA>. Например:

```
<TEXTAREA NAME="IS" ROWS=10 COLS=60>
Севастопольский  государственный университет,
Кафедра ИС
</TEXTAREA>
```

Тег **<INPUT>** используется как для ввода одной строки текстовой информации, так и для организации CHECKBOX, RADIOBUTTON, кнопок и др. Атрибуты тега:

- **TYPE** – определяет тип поля ввода. По умолчанию это простое поле ввода для одной строки текста. Остальные типы должны быть явно указаны:
  1. **TYPE=TEXT** – данный тип поля ввода используется по умолчанию и описывает однострочное поле ввода. Используйте атрибуты MAXLENGTH и SIZE для определения максимальной длинны вводимого значения в символах и размера отображаемого поля ввода на экране (по умолчанию принимается 20 символов).
  2. **TYPE=CHECKBOX –** используется для простых логических (BOOLEAN) значений. Значение, ассоциированное с именем данного поля, которое будет передаваться в вызываемую CGI-программу, может принимать значение ON или OFF.
  3. **TYPE=HIDDEN –** поля данного типа не отображаются браузером и не дают пользователю изменять присвоенные данному полю по умолчанию значение. Это поле может использоваться для передачи в CGI-программу статической служебной информации.
  4. **TYPE=IMAGE** – данный тип поля ввода позволяет вам связывать графический рисунок с именем поля. При нажатии мышью на какую-либо часть рисунка будет немедленно вызвана ассоциированная форме CGI-программа. Значения, присвоенные перемен-

ной NAME, будут выглядеть так – создается две новых переменных: первая имеет имя, обозначенное в поле NAME с добавлением .x в конце имени. В эту переменную будет помещена X-координата точки в пикселях (считая началом координат, левый верхний угол рисунка), на которую указывал курсор мыши в момент нажатия, а переменная с именем, содержащимся в NAME и добавленным .y, будет содержать Y-координату. Все значения атрибута VALUE игнорируются. Само описание картинки осуществляется через атрибут SRC и по синтаксису совпадает с тегом <IMG>.

5. **TYPE=PASSWORD** – то же самое, что и атрибут TEXT, но вводимое пользователем значение не отображается браузером на экране (предназначается для ввода пароля).
6. **TYPE=RADIO** – данный атрибут позволяет организовать элемент для ввода одного значения из нескольких альтернатив. Для создания набора альтернатив необходимо создать несколько полей ввода с атрибутом TYPE="RADIO" с разными значениями атрибута VALUE, но с одинаковыми значениями атрибута NAME. В CGI-программу при этом будет передано значение типа NAME='VALUE', причем 'VALUE' примет значение атрибута VALUE(см. ниже) того поля ввода, которое в данный момент будет выбрано (будет активным). При выборе одного из полей ввода типа RADIO все остальные поля данного типа с тем же именем (атрибут NAME) автоматически станут невыбранными на экране. Пример:

   ```
   <FORM>
   Пол мужской <INPUT NAME="POL" TYPE= RADIO VALUE="Male">
   Пол женский<INPUT NAME="POL" TYPE= RADIO VALUE="Female">
   </FORM>
   ```

7. **TYPE=RESET –** данный тип обозначает кнопку, при нажатии которой все поля формы примут значения, описанные для них по умолчанию. Такая кнопка используется для очистки формы.
8. **TYPE=SUBMIT** – данный тип обозначает кнопку, при нажатии которой данные введенные в форму будут отправлены на сервер для обработки в соответствии с атрибутами METHOD и ACTION, указанными в заголовке формы. Атрибут VALUE в этом случае может содержать текстовую строку, которая будет высвечена на кнопке.

- **CHECKED –** означает, что CHECKBOX или RADIOBUTTON будет в установленном состоянии.
- **MAXLENGTH** – определяет количество символов, которое пользователи могут ввести в поле ввода. При превышении количества допустимых символов браузер реагирует на попытку ввода нового

символа звуковым сигналом и не дает его ввести. Не путать с атрибутом SIZE. Если MAXLENGTH больше чем SIZE, то в поле осуществляется скроллинг. По умолчанию значение MAXLENGTH неограниченно.

- **NAME** – имя поля ввода. Данное имя используется как уникальный идентификатор поля, по которому, впоследствии, вы сможете получить данные, помещенные пользователем в это поле.
- **SIZE** – определяет визуальный размер поля ввода на экране в символах.
- **SRC** – URL, указывающий на картинку (используется совместно с атрибутом IMAGE).
- **VALUE** – присваивает полю значение по умолчанию или значение, которое будет выбрано при использовании типа RADIO (для типа RADIO данный атрибут обязателен).

В настоящее время формы используются для ввода данных в веб-приложения с целью последующей обработки на стороне сервера. В результате разработчикам веб-приложений непрерывно требуются все более удобные для пользователя средства ввода (например, такие как: spinner, slider, date/time picker, color picker и т.д.). Спецификация HTML5 призвана ликвидировать некоторые пробелы, оставленные ее предшественниками в этой области, предоставив широкий ассортимент новых типов элементов ввода форм. Ниже представлены возможные значения аттрибута TYPE тега INPUT, введенные в HTML5:

color
date
datetime
datetime-local
email
month
number
range
search
tel
time
url
week

В дополнение к вышеописанным новым входным типам, спецификация HTML5 поддерживает два новых важных усовершенствования для полей формы. Первое из этих усовершенствований – функция *autofocus*, которая после рендеринга страницы дает браузеру указание автоматически установить фокус на определенном поле формы без использования JavaScript-кода. Второе усовершенствование – атрибут *placeholder*, позволяющий разработчику задать текст, который будет появляться в поле текстового элемента управления в случае отсутствия в нем необходимого контента.

1.2.2.8 Списки выбора в формах

Для организации в формах списков выбора с прокруткой и выпадающих списков выбора используют тег **<SELECT>.** Тег <**SELECT**> имеет один или более элементов выбора между стартовым тегом <**SELECT**> и завершающим <**/SELECT**>. По умолчанию, первый элемент отображается в строке выбора.

Тег <**SELECT**> поддерживает три необязательных атрибута: **MULTIPLE, NAME** и **SIZE.**

Указание атрибута **MULTIPLE** в теге <***SELECT***> указывает, что возможен выбор более чем одного значения (множественный выбор производиться при удерживании клавиш SHIFT или CTRL).

Атрибут **NAME** определяет наименование объекта.

Атрибут **SIZE** определяет число видимых пользователю пунктов списка. Если в форме установлено значение атрибута SIZE=1, то браузер выводит на экран список в виде выпадающего меню. В случае SIZE > 1 браузер представляет на экране обычный список.

Для определения каждого из пунктов списка выбора используют тег **<OPTION>**. Тег **<OPTION>** имеет атрибуты **SELECTED** и **VALUE.** Атрибут **SELECTED** используется для указания выбранных по умолчанию значений. Атрибут **VALUE** указывает на значение, возвращаемое формой после выбора пользователем данного пункта (по умолчанию значение поля соответствует тексту тега **<OPTION>).**

Ниже представлены примеры использования тега **<SELECT>:**

```
<FORM>
  <SELECT  NAME=AGE SIZE=3>
       <OPTION> меньше 15
       <OPTION SELECTED> от 16 до 20
       <OPTION> от 21 до 30
       <OPTION> больше 30
  </SELECT>
</FORM>
```

```
<FORM>
  <SELECT      NAME=group      SIZE=4
MULTIPLE>
       <OPTION value=21>И-21
       <OPTION value=22 SELECTED>И-22
       <OPTION value=23 SELECTED >И-23
       <OPTION value=24>И-24
  </SELECT>
</FORM>
```

Если одновременно выбрано несколько значений, то серверу передается соответствующее количество параметров NAME=VALUE с одинаковыми значениями NAME, но разными VALUE.

Для выполнения группировки элементов списка выбора используется тег <**OPTGROUP**> **(**закрывающий тег обязателен**)**. Обычно это полезно, если пользователь должен делать выбор в длинном списке вариантов; группы связанных вариантов проще просматривать и запоминать, чем один длинный список вариантов. Пример:

```
<SELECT name="Groups">
   <OPTGROUP label="4 курс">
    <OPTION value="41">И-41
    <OPTION value="42">И-42
```

```
    <OPTION value="43">И-43
    <OPTION value="44">И-44
  </OPTGROUP>
  <OPTGROUP LABEL="5 курс">
    <OPTION value="51">И-51
    <OPTION value="52">И-52
    <OPTION value="53">И-53
    <OPTION value="54">И-54
  </OPTGROUP>
  <OPTGROUP LABEL="Магистры">
    <OPTION SELECTED value="MAG1">Маг-51
  </OPTGROUP>
</SELECT>
```

Для группировки различных полей ввода форм используется тег <**FIELDSET**>. Группировка управляющих элементов упрощает пользователям понимание назначения элементов. Корректное использование этого элемента повышает доступность документов. Тег <**LEGEND**> позволяет назначать заголовки для элемента <**FIELDSET**>.

1.2.2.9 Валидация HTML-документов

Консорциум W3C обеспечивает службу валидации страниц Web для проверки страницы на соответствие стандартам. Служба валидации W3C доступна по адресу: http://validator.w3.org.

Используя "DOCTYPE", указанный в документе, валидатор опредеяет используемую версию HTML и выполняет проверку синтаксиса документа для этой версии.

1.3 Варианты заданий

Необходимо выполнить все задания п.1.4 с учетом заданий по варианту из таблицы 1.4.

Таблица 1.4 – Варианты заданий для страницы «тест по дисциплине»

№	Наименование дисциплины	Вопрос №1	Вопрос №2	Вопрос №3
1	Безопасность жизнедеятельности	Textarea	Radio 3	Select 2
2	Высшая математика	Input	Checkbox 5	Select(OptGroup) 5
3	Инженерная графика	Radio 2	Select 4	Textarea
4	Основы дискретной математики	Checkbox 3	Select(OptGroup) 7	Input
5	Основы программирования и алгоритмические языки	Select 4	Textarea	Radio 3
6	Основы экологии	Select(OptGroup) 6	Input	Checkbox 3
7	Теория вероятностей и математическая стати-	Textarea	Radio 4	Select 2

	стика			
8	Физика	Input	Checkbox 2	Select(OptGroup) 4
9	Основы электротехники и электроники	Radio 3	Select 5	Textarea
10	Численные методы в информатике	Checkbox 4	Select(OptGroup) 5	Input
11	Методы исследования операций	Select 5	Textarea	Radio 5
12	Безопасность жизнедеятельности	Select(OptGroup) 7	Input	Checkbox 2
13	Высшая математика	Textarea	Radio 2	Select 3
14	Инженерная графика	Input	Checkbox 3	Select(OptGroup) 5
15	Основы дискретной математики	Radio 4	Select 5	Textarea
16	Основы программирования и алгоритмические языки	Checkbox 2	Select(OptGroup) 6	Input
17	Основы экологии	Select 3	Textarea	Radio 2
18	Теория вероятностей и математическая статистика	Select(OptGroup) 5	Input	Checkbox 3
19	Физика	Textarea	Radio 3	Select 3
20	Методы исследования операций	Input	Checkbox 4	Select(OptGroup) 4

Обозначения:

- Textarea – вопрос с ответом в виде произвольного текста;
- Input – вопрос с ответом в виде строки текста либо числа;
- Radio 4 – вопрос с 4 вариантами ответа. Возможность выбора только одного варианта;
- Checkbox 3 – вопрос с 3 вариантами ответа. Возможность выбора нескольких вариантов;
- Select 5 – вопрос с 5 вариантами ответа в виде выпадающего меню;
- Select(OptGroup) 7 – вопрос с 7 вариантами ответа в виде выпадающего меню (варианты сгруппированы в группы).

1.4 Порядок выполнения работы

В процессе выполнения лабораторной работы необходимо реализовать Ваш персональный Web-сайт. Рекомендуется следующее содержимое Web-страниц:

- **Главная страница**:
  - Меню, содержащее гиперссылки на все страницы сайта;
  - Фамилия Имя Отчество;
  - Фотография;
  - Группа;

- Номер и название настоящей лабораторной работы.

Необходимым является использование следующих элементов HTML: семантические элементы, блоки, гиперссылки, таблицы, элементы форматирования текста, графические элементы.

- **Обо мне**:
  - Меню, содержащее гиперссылки на все страницы сайта;
  - Автобиография в произвольной форме (возможно использование и графического материала);

Необходимым является использование следующих элементов HTML: семантические элементы, блоки, гиперссылки, стили.

- **Мои интересы**:
  - Меню, содержащее гиперссылки на все страницы сайта;
  - содержание страницы, выполненное в виде списка гиперссылок на якоря внутри этой страницы (например: мое хобби, мои любимые книги, моя любимая музыка, мои любимые фильмы и т.п.);
  - непосредственно содержимое каждого из разделов;

Необходимым является использование следующих элементов HTML: семантические элементы, блоки, заголовки различных уровней, нумерованные и маркированные списки, якоря и гиперссылки на якоря, стили.

Таблица 1.5 – Общий вид перечня дисциплин, вариант 1

ПЛАН УЧЕБНОГО ПРОЦЕССА								
№	**Дисциплина**	**Кафедра**	**Всего часов**					
			Всего	**Ауд**	**Лк**	**Лб**	**Пр**	**СРС**
1	*Экология*	БЖ	54	27	18	0	9	27
2	Высшая математика	ВМ	540	282	141	0	141	258
3	Русский язык и культура речи	НГиГ	108	54	18	0	36	54
4	Основы дискретной математики	ИС	216	139	87	0	52	77
5	Основы программирования и алгоритмические языки	ИС	405	210	105	87	18	195
6	Основы экологии	ПЭОП	54	27	18	0	9	27
7	Теория вероятностей и математическая статистика	ИС	162	72	54	18	0	90
8	Физика	Физики	324	194	106	88	0	130
9	Основы электротехники и электроники	ИС	108	72	36	18	18	36
10	Численные методы в информатике	ИС	189	89	36	36	17	100
11	Методы исследования операций	ИС	216	104	52	35	17	112

Таблица 1.6 – Общий вид перечня дисциплин, вариант 2

№	Дисциплина	Часов в неделю (Лекций, Лаб.раб, Практ. раб)											
		1 курс						2 курс					
		1 сем			2 сем			3 сем			4 сем		
1	*Экология*							**1**	**0**	**1**			
2	*Высшая математика*	**3**	**0**	**3**	**3**	**0**	**3**	**2**	**0**	**2**			
3	*Русский язык и культура речи*	**1**	**0**	**2**									
4	*Основы дискретной математики*	**2**	**0**	**1**	**3**	**0**	**2**						
5	*Основы программирования и алгоритмические языки*	**3**	**2**	**0**	**3**	**3**	**0**	**0**	**0**	**1**			
6	*Основы экологии*							**1**	**0**	**0**			
7	*Теория вероятностей и математическая статистика*							**3**	**1**	**0**			
8	*Физика*	**2**	**2**	**0**	**2**	**2**	**0**	**2**	**1**	**0**			
9	*Основы электротехники и электроники*							**2**	**1**	**1**			
10	*Численные методы в информатике*							**2**	**2**	**0**	**0**	**0**	**1**
11	*Методы исследования операций*							**1**	**1**	**0**	**2**	**1**	**1**

- **Учеба**:
  - Меню, содержащее гиперссылки на все страницы сайта;
  - Полное название университета;
  - Полное название кафедры;
  - Перечень изучаемых дисциплин с 1 по 4 семестр, выполненный в виде таблицы (см. таблицы 1.5 и 1.6 выше) в соответствии с вариантом задания. Номер варианта выбирается как остаток от деления последней цифры номера зачетной книжки на 2 плюс 1.

Необходимым является использование следующих элементов HTML: семантические элементы, блоки, таблицы, гиперссылки, стили.

- **Фотоальбом**:
  - 15 фотографий представленных в виде таблицы, по **N**[*)] фото в строке; каждое фото должно иметь всплывающую подсказку с его названием, а также подпись с его названием под фото.
  - гиперссылка на главную страницу.

[*)] **N=(П.Ц.З.К. mod 4)+3** – N определяется как остаток от деления последней цифры номера зачетной книжки на 4 плюс 3

Необходимым является использование следующих элементов HTML: семантические элементы, блоки, графические элементы, гиперссылки.

- **Контакт:**

Данная страница позволяет отправить сообщение на Ваш персональный почтовый ящик. Страница должна содержать форму ввода данных о пользователе-отправителе, поле ввода сообщения, кнопки «Отправить» и «Очистить форму». Состав данных о пользователе:

- Фамилия Имя Отчество (строка ввода);
- Пол (Radiobuttons)
- Возраст (выпадающее меню);
- E-mail(строка ввода);

Кнопка «Отправить» инициирует отправку сообщения на Ваш E-mail, а нажатие на кнопку «Очистить форму» должно приводить к очистке всех полей данных.

Необходимым является использование типов input тегов из спецификации HTML 5.

- **Тест по дисциплине (название дисциплины зависит от варианта)**

Необходимо реализовать гиперссылку на разработанный тест с соответствующей дисциплины страницы «Учеба» (варианты заданий представлены в таблице 1.4).

Данная страница должна обеспечивать ввод данных пользователя и ответов пользователя на 3 тестовых вопроса по дисциплине (типы вопросов выбираются в соответствии с вариантом задания). Состав данных о пользователе:

- Фамилия Имя Отчество (строка ввода);
- Группа (список выбора с группировкой по курсам).

Форма также должна содержать кнопки «Отправить» (инициирует отправку теста на Ваш E-mail), и «Очистить форму».

Вопросы для теста и варианты ответов (в случае необходимости) составить самостоятельно, используя знания и конспект лекций по соответствующей дисциплине.

Необходимым является использование типов input тегов из спецификации HTML 5.

Примечания:

1. Заголовок каждой HTML-страницы должен содержать название сайта и название данной страницы (например: «Персональный сайт Иванова Ивана. Главная страница.»). Все страницы должны быть выполненными в одном стиле (иметь одинаковый цвет фона или фоновое изображение и тп).

2. При выполнении верстки страниц использовать блочный метод и все семантические элементы разметки HTML 5. Приветствуется использование CSS фреймворка [5-6].

3. Для задания внешнего вида содержимого веб-страниц использовать CSS [7].

4. Необходимо выполнить валидацию разработанных страниц в соответствии со стандартами HTML 5 [8] и CSS 3 [9].

5. ПРИ ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ ЗАПРЕЩАЕТСЯ ПОЛЬЗОВАТЬСЯ ГРАФИЧЕСКИМИ HTML-РЕДАКТОРАМИ ИЛИ КОНВЕРТЕРАМИ В ФОРМАТ HTML!.

1.5 Содержание отчета

Цель работы, порядок выполнения работы, тексты разработанных HTML-документов, изображения разработанных Web-страниц, скриншоты результатов проверки кода с использованием соответствующих онлайн-сервисов, выводы по результатам работы.

1.6 Контрольные вопросы

1.6.1. Что такое HTML?
1.6.2. Для чего в HTML используются теги?
1.6.3. Расскажите о структуре HTML-документов?
1.6.4. Для чего в HTML-документах используется элемент META?
1.6.5. Как в HTML-документах реализуются заголовки частей текста?
1.6.6. Как вставить в HTML-документ преформатированный текст?
1.6.8. В чем преимущества использования каскадных таблиц стилей в HTML?
1.6.9. Где возможно размещение информации о стилях в HTML-документе?
1.6.10. Что такое URL? Расскажите о формате URL?
1.6.11. Как задаются ссылки в HTML-документе?
1.6.12. Приведите пример организации списков в HTML?
1.6.13. Как вставить графический элемент в HTML-документ?
1.6.14. Какие атрибуты используются в тегах таблиц HTML?
1.6.15. В чем состоят основные отличия HTML и XHTML?
1.6.16. Что понимают под HTML вёрсткой?
1.6.17. Что такое валидация HTML?
1.6.18. Для чего используются формы в языке HTML?
1.6.19. Какие атрибуты имеет тег <FORM> и для чего они используются?
1.6.20. В чем отличие методов отправки формы GET и POST?
1.6.21. Как реализовать элемент ввода многострочного текста в форме?
1.6.22. Как реализовать элемент ввода типа переключатель в форме?
1.6.23. Для чего используется тег <SELECT>?

1.6.24. Как выполняется группировка элементов списка выбора в выпадающих списках?
1.6.25. Приведите пример HTML-формы для ввода информации о студенте: ФИО, группа, № зачетной книжки?

2 ЛАБОРАТОРНАЯ РАБОТА №2

«Исследование возможностей программирования на стороне клиента. Основы языка JavaScript»

2.1 Цель работы

Исследовать особенности написания программ для приложений на стороне клиента. Изучить основы языка JavaScript и объектной модели браузера. Приобрести практические навыки проверки HTML-форм с использованием JavaScript.

2.2 Краткие теоретические сведения

2.2.1 Язык JavaScript

JavaScript – это прототипно-ориентированный язык программирования, наиболее часто используемый в составе страниц HTML для увеличения функциональности и возможностей взаимодействия с пользователями. С помощью JavaScript можно изменять содержимое HTML-документов, проверять введенные пользователем в форму значения без ее пересылки на сервер, выполнять сложные математические вычисления и т.п.

Скрипты JavaScript в браузере могут выполняться в результате наступления каких-либо событий, инициированных действиями пользователя.

JavaScript является реализацией спецификации ECMAScript [10].

2.2.2 Использование JavaScript в HTML

Программы, написанные на языке JavaScript, могут располагаться непосредственно в HTML-документах. Для этого в HTML используется специальный тег <*script*> и парный ему </*script*>:

```
<script type=«text/javascript»> ... программа на JavaScript ... </script>
```

Атрибут *type* определяет *MIME* тип содержимого тега *script* (например, *type/vbscript* для языка *VBScript*). В HTML 5 атрибут *type* можно опустить, он является не обязательным и принимает значение *text/javascript* по умолчанию (в дальнейшем мы будем опускать данный атрибут).

Обычно функции, составляющие программу, располагаются в секции <*head*> HTML-документа. Такие функции будут загружены раньше, чем пользователь сможет их вызвать с помощью тех или иных интерфейсных средств, располагаемых в секции <*body*>. Посмотрим, как это выглядит в HTML-документе:

```
<html>
<head>
  <title>Пример программы на JavaScript</title>
  <script>
    <!--... программа на JavaScript ... //-->
```

```
  </script>
</head>
  <body>
    ... Текст HTML-документа и вызов функций на JavaScript ...
  </body>
</html>
```

Скрипты JavaScript могут располагаться не только непосредственно в теле HTML-документа, но и во внешнем файле. Для определения места расположения файла, содержащего используемый код JavaScript, используется атрибут *SRC* тега <*script*>. Например:

```
<script src="common.js"></script>
```

Внешние скрипты часто применяются при использовании одних и тех же функций на нескольких страницах. Операторы JavaScript в теле тега <*script*> с атрибутом *SRC* игнорируются. В атрибуте *SRC* можно задать любой URL, относительный или абсолютный. Например:

```
<script src="js/func.js"></script>
<script src="http://home.netscape.com/functions/jsfuncs.js"></script>
```

Внешние файлы с кодом JavaScript не должны содержать никаких тегов HTML – они обязаны содержать только операторы и определения функций JavaScript. Внешние файлы JavaScript должны иметь расширение файла «.js».

Использование тега с атрибутом *src* дает ряд преимуществ:

- HTML-файлы становятся проще, т. к. из них можно убрать большие блоки JavaScript-кода, что помогает отделить содержимое от поведения.
- JavaScript-функцию или другой JavaScript-код, используемый несколькими HTML_файлами, можно держать в одном файле и считывать при необходимости. Это уменьшает объем занимаемой дисковой памяти и намного облегчает поддержку программного кода.
- Когда JavaScript-функции требуются нескольким страницам, размещение кода в виде отдельного файла позволяет браузеру кэшировать его и тем самым ускорять загрузку. Когда JavaScript_код совместно используется несколькими страницами, экономия времени, достигаемая за счет кэширования, явно перевешивает небольшую задержку, требуемую браузеру для открытия отдельного сетевого соединения и загрузки файла JavaScript-кода при первом запросе на его исполнение.
- Как уже было сказано выше, атрибут src принимает в качестве значения произвольный URL-адрес, поэтому JavaScript-программа или веб-страница с одного веб-сервера может воспользоваться ко-

дом (например, из библиотеки подпрограмм), предоставляемым другими веб-серверами.

Внимание! При разработке данной лабораторной работы все JavaScript сценарии необходимо размещать во внешних JavaScript файлах.

2.2.3 Основные элементы языка JavaScript

2.2.3.1 Объектная модель JavaScript

JavaScript основан на объектно-ориентированном подходе. Объект – это конструкция, обладающая *свойствами* и *методами*. Свойства являются переменными *JavaScript*. Свойства могут быть и другими объектами. Функции, связанные с объектом, называются *методами* объекта.

Объект *JavaScript* имеет свойства, ассоциированные с ним. Для обращения к свойствам объекта используется символ ″.″:

```
objectName.propertyName
```

Как отмечалось выше *метод* это функция, связанная с объектом. Метод может быть определен таким же образом, как определяется обычная функция (см. ниже). Чтобы связать функцию с существующим объектом используется следующий синтаксис:

```
object.methodname = function_name,
```

где *object* – существующий объект, *methodname* – имя, которое присваивается методу, и *function_name* – имя функции.

Объектная модель браузера в Java Script – это набор связанных между собой объектов, обеспечивающих доступ к содержимому страницы и ряду функций браузера. В составе объектной модели есть три основных объекта – объект ***window***, находящийся на вершине иерархии и представляющий собой текущее окно браузера; объект ***document*** – предназначен для программного представления самого документа и его содержимого. Последним в перечне основных объектов модели браузера является объект ***frames***, предназначенный для работы с фреймами в окне браузера.

Ниже приведена иерархия и обобщенная объектная модель браузера на рисунке 2.1. Здесь:

- *window* – объект, дающий доступ к окну браузера
- *navigator* – объект, дающий доступ к характеристикам браузера
- *frames* – объект, дающий доступ к фреймам
    - *window...*
    - *window...*
    - *...*
- *location* – объект, содержащий текущий URL
- *history* – объект, дающий доступ к истории посещенных ссылок

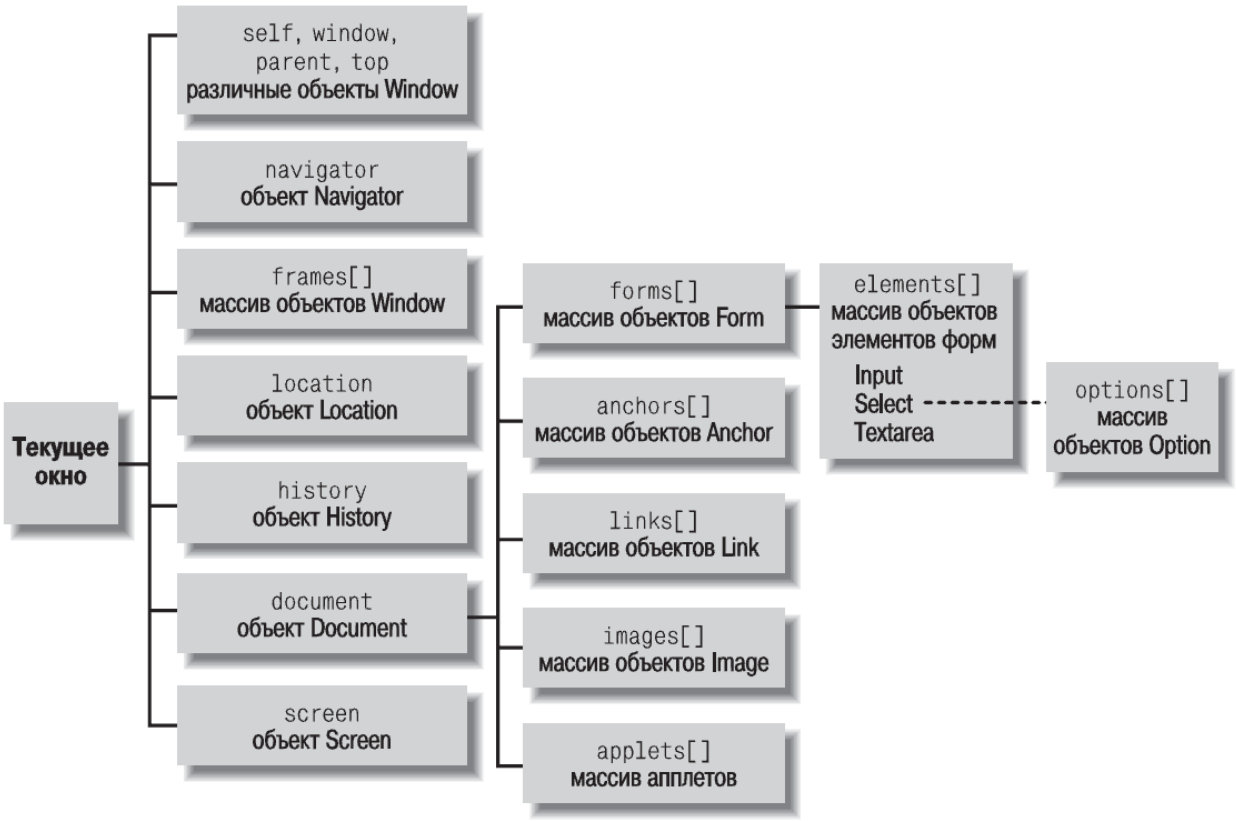

Рисунок 2.1 – Иерархия объектов клиентского JavaScript и нулевой уровень модели DOM

- *document* – объект, содержащий в себе всю HTML-страницу
  - *all* – полная коллекция всех тегов документа
  - *forms* – коллекция форм
    - *elements* – коллекция элементов формы, состоящая из объектов, которые соответствуют:
      - ✓ *button* – тегу <input type=button>
      - ✓ *checkbox* – тегу <input type=checkbox>
      - ✓ *radio* – тегу <input type=radio>
      - ✓ *text* – тегу <input type=text>
      - ✓ *textarea* – тегу <textarea>
  - *anchors* – коллекция якорей
  - *applees* – коллекция апплетов
  - *embeds* – коллекция внедренных объектов
  - *filters* – коллекция фильтров
  - *images* – коллекция изображений
  - *links* – коллекция ссылок
  - *plugins* – коллекция подключаемых модулей
  - *scripts* – коллекция блоков <script></script>
  - *selection* – коллекция выделений
  - *stylesheets* – коллекция объектов с индивидуально заданными стилями

- *screen* – объект, дающий доступ к характеристикам экрана
- *event* – объект, дающий доступ к событиям

Как видно, объектная модель обеспечивает для доступа к HTML-странице объект *document* и ряд дополнительных объектов и коллекций (массивов) для программного представления самого документа и его содержимого. Из всех методов объекта *document* пока отметим только метод *write*, выводящий указанный текст в окно браузера, и *writeln*, выводящий текст, в конце которого устанавливается символ «возврата каретки» (для того, чтобы символ «возврата каретки» отображался браузером, метод *writeln* должен быть заключен между тегами <pre></pre>, в противном случае для переноса строки можно вывести тег
):

```
<pre>
  <script>
    document.writeln («Первая строка»);
    document.writeln («Вторая строка»);
  </script>
</pre>
<script>
  document.write ("Третья строка<br>");
  document.write ("Четвертая строка");
</script>
```

Кроме того, стоит отметить метод *alert* объекта *window*. Этот метод позволяет отображать на экране браузера различные сообщения – эти сообщения появляются в диалоговом окне, содержащем кнопку OK. Пример использования метода *alert* объекта *window* представлен ниже:

```
<script>
  window.alert(«Пример вывода сообщения»);
</script>.
```

2. 2.3.2 Переменные JavaScript

Переменные языка JavaScript могут хранить значения различных типов:

- **Строки** – неизменяемая упорядоченная последовательность символов;
- **Числовые значения** – целые и действительные числа;
- **Булевы значения** – только два значения **true** или **false;**
- **Объекты** – сложные типы данных с собственными наборами методов:
  - **Массивы** – упорядоченные коллекции пронумерованных значений;
  - **Даты** – значения даты и времени.
  - **Регулярные выражения** – инструменты для сопоставления строк с заданным шаблоном.
  - **Функции –** это объекты, с которыми связан выполняемый код.

В языке JavaScript создать переменную гораздо проще, чем во многих других языках программирования. Так, например, при создании переменной нет необходимости указывать ее тип. Переменные могут быть определены как с начальными значениями, так и без них. В процессе выполнения программы уже созданные переменные можно приводить к различным типам данных.

Имена переменных могут начинаться с любой буквы (от a до z, или A-Z) либо с символа подчеркивания «_», оставшаяся часть может содержать цифры, символы подчеркивания и буквы. Следует отметить, что в именах переменных различаются символы верхнего и нижнего регистра: например MyVariable и myvariable – это две различные переменные.
Переменную можно создать одним из способов:

- при помощи оператора var и операции присваивания (=);
- при помощи операции присваивания (=).

Оператор var используют не только для создания переменной, но и для ее инициализации. Операция присваивания (=) необходима, чтобы запомнить в переменной значение, и при работе с переменными, созданными без начального значения, ее использовать не обязательно. Например:

```
var MyVariable = 35;
```

создает переменную с именем *MyVariable*, содержащую числовое значение 35. Переменная существует до тех пор, пока загружен текущий документ.

В языке JavaScript переменные можно переопределять, даже задавая другой тип данных. После выполнения оператора *MyVariable = "35";* переменная будет уже хранить строку "35". Например:

```
<script>
  var a = 35, b = "5", c;
  c = a + b;
  c = c – a;
  document.writeln("C="+c);
</script>
```

После выполнения данного фрагмента, значение переменной "c" будет соответствовать числовому значению 320. (О принципах преобразования типов переменных будет рассказано ниже).

Переменную можно определить и не используя оператор "Var", а просто достаточно присвоить значение, причем какой тип данных будет присвоен, такого типа и окажется переменная. Переменная может быть определена и без начальных значений, например, строка «var MyVariable;» создаст переменную с именем *MyVariable,* которая не имеет определенного типа данных и начального значения. Переменные, создаваемые при помощи таких описаний, известны как неопределенные (*null variable*). Например, сравнивая такую переменную со значением *null*, можно узнать, определена ли переменная. Однако, следует отметить, что "" (пустая строка) это строковый тип и ему соответствует **не** *null*-значение.

Тип переменной может быть установлен в любом месте JS-программы в отличие от других языков программирования, что дает дополнительную гибкость.

Время жизни и видимость переменной связаны с окном, в котором они созданы, и зависят от того, где они определены. Переменные, определенные внутри тела JavaScript-функции, видны только внутри этой функции.

Если Вы устанавливаете идентификатор переменной путём присвоения ей значения вне какой-либо функции, такая переменная называется *глобальной*, поскольку доступна в любом месте документа. Если Вы объявляете переменную внутри функции, она называется *локальной переменной*, поскольку доступна только внутри данной функции.

Использование var при объявлении глобальной переменной не требуется. Однако обязательно использовать var при объявлении переменой внутри функции.

Вы можете получить доступ к глобальным переменным, объявленным в одном окне или фрейме, из другого окна или фрейма, задавая имя окна или фрейма. Например, если переменная phoneNumber объявляется в документе с *FRAMESET*, Вы можете обратиться к этой переменной из дочернего фрейма так: *parent.phoneNumber*.

JavaScript-программы содержатся в документах HTML, и при загрузке нового документа в браузер любые переменные, созданные в программе, будут удалены. Чтобы сохранить какие-либо значения при загрузке нового документа в JavaScript, имеются только 2 решения:

- путем создания переменных во фреймосодержащем документе верхнего уровня;
- путем использования "cookies".

Строка представляет собой множество символов, заключенных в одинарные или двойные кавычки. Строки в JavaScript представляются объектами. Это практически самый распространенный вид объектов, поэтому в JavaScript имеется множество стандартных функций, предназначенных для управления строками. Создать объект *String* можно одним из нескольких способов:

- присваивание значения при помощи конструктора *String*();
- использование оператора var и оператора присваивания для создания и инициализации строки;
- создание и инициализация строковой переменной в операторе присваивания;
- преобразование переменной числового типа путем сложения со строковым типом (10+"" ==> "10").

Самым простым и наиболее распространенным способом создания объекта *String* является использование оператора присваивания:

```
var myVariable = "Строка первая";
```

Приведенный оператор присваивает строку "Строка первая" строковой переменной *myVariable*. Переменная *myVariable* рассматривается как строковый объект и может использовать любой из стандартных методов объекта *String* языка JavaScript. Оператор Var можно пропустить, как и говорилось ранее, он нужен в основном для читабельности программы.

Для создания строковых объектов допускается использовать конструктор *String*() с оператором *new*. В действительности, объект *String* не относится к языку JavaScript, а является встроенным объектом браузера главным образом потому, что строки создаются тогда, когда пользователь в них нуждается. Рассмотрим пример:

```
var myVariable = new String();
```

Этот оператор создает новый объект – пустую строку с именем *myVariable*. Изначально это пустая строка (""), а значение свойства *myVariable.length* равно 0. Конструктор *String*() допускает передачу заданной строки в виде аргумента для начальной инициализации:

```
var myVariable = new String("Правильное пиво");
```

Строковый объект может содержать специальные символы, управляющие форматированием строк:

- \n – символ новой строки;
- \r – символ возврата каретки;
- \f – код перехода на новую страницу;
- \xnn – представление символа в виде шестнадцатиричного ASCII-кода nn;
- \b – код клавиши [Backspace].

Эти символы будут правильно интерпретированы только в контейнерах <textarea> и при использовании функции вывода сообщения – "Alert".

Объект *String* имеет только одно свойство – *length*, значением которого является количество символов в строке, содержащейся в объекте. Методы объекта *String* можно разделить на две категории:

- методы форматирования HTML-документа;
- методы обработки строк.

Методы форматирования документа применяются для вывода строк в документ, а методы управления строками – для проверки и изменения содержимого объекта String.

В таблице ниже представлен перечень некоторых методов объекта String: (буква Ф – методы форматирования, а буква У – управления строками).

Таблица 2.1 – Методы объекта String

Метод	**Тип**	**Описание**
anchor()	У	Создает именованную метку, т.е. тег <a name> из содержимого объекта(строки)
Big()	Ф	Заключает строку в контейнер <big> . . . </big>, для отображения крупным шрифтом
blink()	Ф	Заключает строку в контейнер <blink> . . . </blink>, чтобы она отображалась мигающей.
bold()	Ф	Заключает строку в контейнер . . . , чтобы она отображалась жирным шрифтом.
charAt()	У	Возвращает символ, находящийся в заданной позиции строки
fixsed()	Ф	Заключает строку в контейнер <tt> . . . </tt>, чтобы она отображалась шрифтом постоянной ширины.
fontcolor()	Ф	Заключает строку в контейнер . . . , чтобы она отображалась определенным цветом.
fontsize()	Ф	Заключает строку в контейнер . . . , чтобы она отображалась шрифтом определенного размера.
indexOf(arg1 [,arg2])	У	Возвращает позицию в строке, где впервые встречается символ – arg1, необязательный числовой аргумент arg2 указывает начальную позицию для поиска.
italics()	Ф	Заключает строку в контейнер . . . , чтобы она отображалась курсивом.
lastindexOf(arg1 [,arg2])	У	Возвращает либо номер позиции в строке, где в последний раз встретился символ – arg1, либо -1, если символ не найден. Arg2 задает начальную позицию для поиска.
link()	У	Создает тег гиперсвязи <a href> . . . и помещает в него содержимое объекта
small()	Ф	Заключает строку в тег . . . , чтобы она отображалась шрифтом меньшего размера.
strike()	Ф	Заключает строку в контейнер <strike> . . . </strike>, чтобы она отображалась зачеркнутой.
Sub()	Ф	Заключает строку в контейнер _{. . .}, чтобы она отображалась как нижний индекс.
substring(arg1, arg2)	У	Возвращает подстроку длинной arg2, начиная с позиции arg1.
Sup()	Ф	Заключает строку в контейнер ^{. . .}, чтобы она отображалась как верхний индекс.
toLowerCase()	У	Преобразует все буквы строки в строчные
toUpperCase()	У	Преобразует все буквы строки в прописные

Операция конкатенации (+) может быть использована для объединения двух строковых значений, возвращая строку, которая является результатом объединения двух строк-операндов. Например, "my " + "string" возвращает строку "my string".

Операция-аббревиатура присвоения += также может использоваться для конкатенации строк. Например, если переменная mystring имеет значение "alpha", то выражение mystring+="bet" вычисляется в "alphabet" и это значение присваивается переменной mystring.

В примере, представленном ниже, производится подсчет количества слов в строке (предполагается, что слова разделены одним пробелом):

```
var str  = "Тестовая строка, содержащая пять слов";
var cnt = 0,startpos=0;
while(str.indexOf(" ",startpos)>=0){
   cnt++;
   // переведем стартовую позицию для поиска на следующий символ
   startpos=str.indexOf(" ",startpos)+1;
}
// увеличим счетчик, если последний пробел стоит не в конце строки
if (startpos<str.length) {
  cnt++;
}
document.write("В строке \""+str.italics()+"\" содержится "+cnt+" слов.");
```

Числовые значения могут быть либо целыми, либо числами с плавающей точкой. Числа с плавающей точкой называют действительными или вещественными. К числовым значениям применимы операции:

- умножения (*);
- деления (/);
- сложения (+);
- вычитания (-);
- увеличения (++);
- уменьшения (--);

Кроме того, используются операции умножения, деления, сложения и вычитания в сочетании с присваиванием (*=, /=, +=, -=).

Булевы, или логические, переменные содержат только литеральные значения – true и false – и используются в логических выражениях и операторах. Вместо true и false можно использовать числовые значения 1 и 0. Логические операторы сравнения представлены в таблице 2.2:

Таблица 2.2 – Операторы сравнения

Операция	**Описание**	**Примеры, возвращающие true (var1=3, var2=4)**
Равно (==)	Возвращает true, если операнды равны. Если два операнда имеют разные типы, JavaScript пытается конвертировать операнды в значения, подходящие для сравнения.	3 == var1 "3" == var1 3 == '3'
Не равно (!=)	Возвращает true, если операнды не равны. Если два операнда имеют разные типы, JavaScript пытается конвертировать операнды в значения, подходящие для сравнения.	var1 != 4 var2 != "3"
Строго равно (===)	Возвращает true, если операнды равны и одного типа.	3 === var1
Строго не равно (!==)	Возвращает true, если операнды не равны и/или не одного типа.	var1 !== "3" 3 !== '3'
Больше (>)	Возвращает true, если левый операнд больше правого операнда.	var2 > var1
Больше или равно (>=)	Возвращает true, если левый операнд больше правого операнда или равен ему.	var2 >= var1 var1 >= 3
Меньше (<)	Возвращает true, если левый операнд меньше правого операнда.	var1 < var2
Меньше или равно (<=)	Возвращает true, если левый операнд меньше правого операнда или равен ему.	var1 <= var2 var2 <= 5

В языке JavaScript наборы значений (массивы) чаще всего представляются объектом *Array*. Массивы создаются при помощи конструктора *Array*(). С помощью конструктора *Array*() не только создается сам объект array, но и могут быть присвоены начальные значения его элементам. Количество элементов массива доступно через свойство *length*.

Существует возможность добавлять элементы в массив динамически – путем присваивания определенных значений элементам массива. Допускается также "пропускать" элементы массива и задавать их в любой последовательности. Для создания нового экземпляра объекта *Array* необходимо использовать конструктор *Array*() с оператором *new*.

В следующем примере создается массив с именем *arrayImg*, содержащий два элемента, каждый из которых является объектом *String*

```
var path = "c:/images/" ,
arrayImg = new Array();
arrayImg[0] = path+"img1.gif";
arrayImg[1] = path+"img2.gif";
```

Как видно, к элементу массива следует обращаться при помощи выражения *arrayName[index],* где *arrayName* – имя массива, а *index* – числовая

переменная или число, задающее позицию элемента в массиве. Индексы элементов массива в языке JavaScript начинаются с нуля. Элементы массива могут быть любого типа, например, строками или булевыми переменными. Кроме того, при определенных условиях массив может содержать элементы различных типов данных.

При использовании конструктора *Array*() значение свойства *length* устанавливается автоматически. Поэтому после инициализации элементов массива в приведенном примере выражение *arrayImg.length* возвращает значение 2. Элементы массива также могут быть заданы как параметры конструктора:

```
var path  = "c:/images/" ,
arrayImg = new Array(path+"img1.gif", path+"img2.gif");
```

Данное выражение представляет собой сокращенную запись предыдущего примера.

Как отмечалось выше, в языке JavaScript можно создавать массив, в котором элементы имеют различный тип данных. Например:

```
var myArray = new Array(3.14, true, 85, date(), "word");
```

создает массив, элемент *myArray[0]* которого является числом с плавающей запятой, элемент *myArray[1]* – булевым значением, элемент *myArray[2]* – целочисленным значением, элемент *myArray[3]* – объектом *Date*, элемент *myArray[4]* – строкой.

Размер массива и, следовательно, значение свойства *length* массива, создаваемого конструктором *Array*(), зависят от максимального значения индекса, который применялся для задания элемента массива. Например:

```
var myArray = new Array;
myArray[20] = "Это 21 элемент массива";
```

двадцать первому элементу массива присваивается строковое значение "Это 21 элемент массива", таким образом, значение свойства *myArray.length* равно 21 независимо от того имеют ли значения элементы массива с индексом меньше 20.

Также значение свойства *length* объекта *Array* автоматически устанавливается при явном указании количества элементов в конструкторе *Array*():

```
myArray = new Array(10);
```

оператор создает массив из 10-ти элементов с индексами от 0 до 9. Значение свойства *length* массива нельзя установить путем присваивания, так как length является свойством только для чтения. Например, чтобы задать значение 10 для свойства *length* нужно только определить значение последнего, в данном случае 9-го элемента массива:

```
myArray = new Array();
myArray[9] = 0;
```

Кроме того, существует возможность задать значения элементов массива при его конструировании:

```
myArray = new Array(0,0,0,0,0,0);
```

В JavaScript существует достаточно много встроенных методов для работы с массивами, которые поддерживаются браузерами:

- pop – удаляет последний элемент в массиве и возвращает удалённый элемент.
- push – добавляет в конец массива произвольное количество элементов, которые он принимает в качестве параметров. Этот метод возвращает длину нового увеличенного массива.
- shift – удаляет нулевой элемент массива и возвращает удалённый элемент.
- unshift – добавляет в начало массива произвольное количество элементов, которые принимает в качестве параметров. Этот метод возвращает длину нового увеличенного массива.
- reverse – меняет порядок следования элементов в массиве на обратный и возвращает новый перевёрнутый массив.
- join – этот метод ‘склеивает’ элементы массива в одну строку, используя в качестве разделителя переданный аргумент, массив при этом не изменяется. Аргумент у этого метода можно опускать. В этом случае элементы массива будут соединены запятыми.
- slice – возвращает часть исходного массива. В качестве параметров принимает индекс элемента, с которого будет начинаться вырезанный массив и индекс, на единицу превосходящий индекс элемента, которым вырезанный массив будет заканчиваться вырезанном массиве. Этот метод возвращает вырезанный массив, не изменяя исходный. Второй параметр в методе slice можно опускать. В этом случае выборка элементов в новый массив продолжится до последнего элемента исходного массива.
- splice – этот метод можно использовать как для массового удаления элементов из массива, так и для массового добавления. В качестве параметров он принимает индекс элемента, с которого начинать удаление и количество удаляемых элементов. Возвращает метод массив из удалённых элементов. Для удаления элемента из массива по индексу следует передать методу splice в качестве первого параметра индекс удаляемого элемента, а в качестве второго единицу.
- concat – склеивает массивы.
- sort – используется для сортировки массива. Возвращает отсортированный массив. Стоить заметить, что массив по умолчанию сортируется в лексикографическом порядке, то есть в порядке возрастания символов в таблице ASCII. Если элементы массива числа, то этот порядок не совпадает с порядком возрастания чисел:

```
var arr = [7, 6, 5, 11, 23, 45];
arr.sort();
console.log(arr); // [11, 23, 45, 5, 6, 7]
```

Если элементы массива строки, то после сортировки они будут расположены в алфавитном порядке. Метод *sort* в качестве необязательного параметра может принимать функцию. Эта функция принимает в качестве аргументов два сравниваемых по какому-то критерию элемента массива. Она должна возвращать отрицательное число, если первый аргумент меньше второго, ноль, если аргументы равны и положительное число, если второй аргумент меньше первого.

- filter – этот метод выбирает элементы массива по определённому критерию и возвращает их в массиве. В качестве параметра он принимает функцию. Эта функция принимает в качестве параметра элемент массива, который оценивает по какому-то признаку и должна возвращать *true*, если элемент удовлетворяет этому признаку, и *false* в противном случае. Например, отберём все чётные элементы массива:

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8];
console.log(arr.filter(function (item) { // [2, 4, 6, 8]
  return item % 2 == 0; }));
```

- forEach – эта функция принимает в качестве параметра функцию, которая будет вызвана для каждого элемента массива. Выведем с помощью это функции элементы массива в столбик на консоль:

```
var arr = ['one', 'two', 'three', 'four', 'five'];
arr.forEach(function (item) {
  console.log(item); });
```

- map – создаёт новый массив, состоящий из изменённых элементов старого. В качестве параметра этот метод принимает функцию, которая изменяет элементы массива. Увеличим все элементы массива на единицу:

```
var arr = [1, 2, 6, 8, 2, 9];
arr = arr.map(function (item) {
  return ++item; });
console.log(arr); // [2, 3, 7, 9, 3, 10]
```

Многие из рассмотренных методов возвращают массив, что позволяет комбинировать их в цепочки вызовов, в которых каждый последующий вызов метода производится от результата вызова предыдущего.

2.2.3.3 Преобразование типов данных

Как было отмечено выше, JavaScript это динамически типизированный язык. Это означает, что нет необходимости специфицировать тип данных переменной при её объявлении и что типы данных при необходимости автоматически конвертируются при выполнении скрипта.

При преобразовании строк и чисел используются следующие два правила:

1. Преобразование числа в строку символов производится путем сложения числового аргумента со строковым, независимо от перестановки слагаемых. Например, если переменная *x = 123*, то преобразовать переменную и, следовательно, ее значение в строку символов можно и так: *x = x + ""*, и наоборот: *x = "" + x*. Если сложить не с пустой строкой: x = x + "456", то результатом значения переменной x станет "123456". Это же справедливо и в случае: x = "456" + x – результат: "456123".

2. Преобразование строки в число производится путем вычитания одного операнда из другого и также независимо от их позиции. Например, если переменная x = "123", то преобразовать ее в число можно если вычесть из нее значения 0: x = x – 0, и соответственно значение переменной из строкового типа преобразуется в числовой: 123. При перестановке операндов соответственно знак числового значения поменяется на противоположный. В отличие от преобразования числа в строку в действиях вычитания нельзя применять буквенные значения. Так если "JavaScript"+10 превратится в x == "JavaScript10", то операция типа x = "JavaScript" – 10 выдаст значение "NaN" (нечисловое значение) – то есть такая операция не допустима. Следует так же отметить, что при вычитании строкового значения из строкового же также происходит преобразование: x = "20" – "15", значением переменной x станет число 5.

2.2.3.4 Комментарии

В JavaScript определены два типа комментариев: однострочный и многострочный. Комментарии могут быть использованы как для добавления в программу пояснений, так и для временного исключения некоторых фрагментов программы во время отладки. Например:

```
// Текст однострочного  комментария
/* Текст
многострочного
комментария
  */.
```

2.2.3.5 Объекты JavaScript, соответствующие тегам HTML

Как отмечалось ранее, существуют объекты языка JavaScript соответствующие тегам HTML-документа. Эти объекты представлены в таблице 2.3.

Таблица 2.3 – Объекты JavaScript, соответствующие тегам HTML

Имя объекта	Краткое описание
anchor (anchors[])	Множество тегов <a name> в текущем документе

button	Кнопка, создаваемая при помощи тега <input type=button>
checkbox	Контрольный переключатель, создаваемый при помощи тега <input type=checkbox>
elements[]	Все элементы тега <form>
form (forms[])	Множество объектов тега <form> языка HTML
frame (frames[])	Фреймосодержащий документ
hidden	Скрытое текстовое поле, создаваемое при помощи тега <input type=hidden>
images (images[])	Множество изображений (тегов <img>) в текущем документе
link (links[])	Множество гиперсвязей в текущем документе
navigator	Объект, содержащий информацию о браузере, загрузившем документ
password	Поле ввода пароля, создаваемое при помощи тега <input type=password>
radio	Селекторная кнопка (radio button), создаваемая при помощи тега <input type=radio>
reset	Кнопка очистки формы, создаваемая при помощи тега <input type=reset>
select (options[])	Элементы <option> объекта <select>
submit	Кнопка передачи данных, создаваемая при помощи тега <input type=submit>
text	Поле ввода, создаваемое при помощи тега <input type=text>
textarea	Поле текста, создаваемое при помощи тега <textarea>

В примере ниже содержится два поля. Первое поле является областью текста (*textarea*). При изменении содержимого текстовой области(событие onChange) активизируется функция *sChange*(), которая выводит окно сообщения, информирующее о том, что текст изменялся:

```
<script>
  function sChange() {
    alert ("Содержимое текстовой области изменено");  }
</script>
<form>
  Измените этот текст и перейдите в другое поле формы:<br/>
  <textarea name="tarea" rews=5 cols=40 onChange="sChange()">
   Это объект textarea
   Пример текста по умолчанию
  </textarea>  <br/>
  <input type="text" size=35 name="stxt">
</form>.
```

2.2.3.6 Операторы JavaScript

Несмотря на то, что операторы JavaScript полностью сходны по синтаксису с изученными ранее операторами языка С, рассмотрим основные из них.

Оператор if используется для выполнения определённых операторов, если логическое условие *true*; не обязательный блок *else* выполняется, если условие *false*. Оператор *if* выглядит так:

```
if (condition) {   statements1 }
[else {   statements2 } ]
```

condition это может быть любое выражение JavaScript, которое вычисляется в *true* или *false*. Выполняемые операторы это любые операторы JavaScript, включая вложенные *if*. Если нужно использовать более одного оператора после операторов *if* или *else*, необходимо заключать их в фигурные скобки {}.

Пример. В следующем примере функция *checkData* возвращает true, если количество символов в объекте Text равно трём; иначе выводит диалог alert и возвращает *false*:

```
function checkData () {
  if (document.form1.threeChar.value.length == 3) {
    return true;
  } else {
    alert("Введите ровно три символа. " +
    document.form1.threeChar.value + " – введено не верно.");
    return false;
  }
}
```

Оператор switch позволяет вычислять выражение и пытается найти совпадение значения выражения с оператором label. Если совпадение найдено, программа выполняет ассоциированный оператор. Оператор switch выглядит так:

```
switch (expression){
  case label :
    statement;
    break;
  case label :
    statement;
    break;
  ...
  default : statement;}
```

Программа сначала ищет *label* (метку), совпадающую по значению с выражением, а затем выполняет ассоциированный оператор. Если совпадающий *label* не найден, программа ищет не обязательный оператор по умолчанию (*default*) и, если он найден, выполняет ассоциированный опера-

тор. Если оператор по умолчанию(*default statement*) не найден, программа продолжает выполняться с оператора, идущего после switch.

Не обязательный оператор *break*, ассоциируемый с каждым *case*, гарантирует, что программа прервёт выполнение блока switch, как только будет выполнен совпавший оператор, и продолжит выполнение с оператора, идущего после *switch*. Если break отсутствует, программа продолжит выполнение следующего оператора в блоке *switch*.

В следующем примере, если *expr* вычисляется в "Bananas", программа находит совпадение с case "Bananas" и выполняет ассоциированный оператор. Если обнаружен *break*, программа прерывает выполнение блока *switch* и выполняет оператор, идущий после *switch*. Если *break* отсутствует, оператор для case "Cherries" также будет выполнен.

```
switch (expr) {
  case "Oranges" :
    document.write("Oranges are $0.59 a pound.<br>");
    break;
  case "Apples" :
    document.write("Apples are $0.32 a pound.<br>");
    break;
  case "Bananas" :
    document.write("Bananas are $0.48 a pound.<br>");
    break;
  case "Cherries" :
    document.write("Cherries are $3.00 a pound.<br>");
    break;
  default :
    document.write("Sorry, we are out of " + i + ".<br>");
}
document.write("Is there anything else you'd like?<br>");
```

Цикл это набор команд, которые выполняются неоднократно, пока не будет выполнено специфицированное условие. JavaScript поддерживает операторы циклов *for, do while, while* и *label* (*label* сам по себе не является оператором цикла, но часто используется с этими операторами). Кроме того, Вы можете использовать операторы *break* и *continue* с операторами циклов.

Оператор *for* повторяется, пока специфицированное условие не станет *false*. JavaScript-цикл *for* похож на аналогичные циклы Java и C. Оператор *for* выглядит так:

```
for ([initialExpression]; [condition]; [incrementExpression]) {
  statements }
```

При выполнении цикла *for* происходит следующее:

1. Выполняется инициализирующее выражение initial-expression, если имеется. Это выражение обычно инициализирует один или более

счётчиков цикла, но синтаксис допускает выражения любой сложности.

2. Выражение condition вычисляется. Если condition даёт true, цикл выполняется. Если condition равно false, цикл for прерывается.
3. Выполняются statements/операторы.
4. Выполняется выражения обновления incrementExpression, и управление возвращается к шагу 2.

Например, эта функция содержит оператор for, подсчитывающий количество выбранных опций в прокручиваемом списке (объекте Select, который позволяет делать множественный выбор). Оператор for объявляет переменную i и инициализирует её значением 0 (нуль). Если i меньше количества опций объекта Select, выполняется следующий оператор if, а i увеличивается на 1 при каждом прохождении цикла.

```
<html>
<head>
 <script>
 function howMany(selectObject) {
  var numberSelected=0;
  for (var i=0; i < selectObject.options.length; i++) {
   if (selectObject.options[i].selected==true)
     numberSelected++;
  }
  return numberSelected;
 }
 </script>
</head>
<body>
 <form name="selectForm">
  <p>
   <strong>
     Выберите несколько вариантов и нажмите на кнопку:
   </strong>
  </p>
  <br/>
  <select name="musicTypes" multiple>
   <option SELECTED>R&amp;B</option>
   <option>Jazz</option>
   <option>Blues</option>
   <option>New Age</option>
   <option>Classical</option>
   <option>Opera</option>
  </select>
  <input type="button" value=" Сколько выбрано?"
   onClick="alert ('Количество выбранных выриантов: ' +
howMany(document.selectForm.musicTypes))">
</form>
</body>
</html>
```

Оператор do...while повторяется, пока специфицированное выражение не станет false. Оператор do...while выглядит так:

```
do {
  statement
} while (condition)
```

где statement выполняется как минимум один раз, так как находится перед проверяемым условием. Если condition возвращает true, цикл выполняется ещё раз. В конце каждого прохода проверяется условие. Если condition возвращает false, выполнение останавливается и управление передаётся оператору, идущему после do...while.

Оператор while выполняется, пока специфицированное условие вычисляется в true. Оператор while выглядит так:

```
while (condition) {
  statements
}
```

Если condition становится false, операторы внутри цикла перестают выполняться и управление передаётся оператору, идущему после цикла.
Проверка условия/condition выполняется перед началом каждого цикла. Если condition возвращает true, операторы выполняются и условие проверяется снова. Если condition возвращает false, выполнение прекращается и управление передаётся оператору, идущему после цикла while.

Оператор break используется для прерывания выполнения цикла, либо операторов switch или label.

Когда break используется с операторами while, do-while, for или switch, он немедленно прерывает самый внутренний цикл или switch и передаёт управление следующему оператору.

Оператор continue может использоваться для рестарта оператора while, do-while, for или label.

В операторах while или for, оператор continue прерывает текущий цикл и начинает новую итерацию (проход) цикла. В отличие от break, continue не прерывает полностью выполнение цикла. В цикле while он перескакивает на condition. В цикле for он перескакивает на increment-expression.

Функции – один из фундаментальных встроенных элементов в *JavaScript*. Определение функции состоит из ключевого слова function, сопровождаемого:

- именем функции;
- списком аргументов функции, записанным в круглых скобках и отделяемых запятыми;

- *JavaScript* операторами, заключенными в фигурных скобках, {...}.

В любом месте HTML-страницы можно вызывать функции, определенные в текущей странице. Лучше всего определять функции в разделе HEAD HTML-страницы. В этом случае функции всегда будут загружаться раньше, чем будут вызваны. Функция также может быть рекурсивной, то есть может вызывать саму себя. Например, ниже представлена HTML-страница, содержащая рекурсивную функцию вычисления факториала:

```
<html>
<head>
  <script>
    function factorial(n) {
      if ((n == 0) || (n == 1)) {
                return 1;
      }
      else {
        result = (n * factorial(n-1));
        return result;
      }
    }
  </script>
</head>
<body>
  <h1>Значения факториалов от 0 до 5:</h1>
  <script>
    for (x = 0; x <= 5; x++) {
      document.write(x, "! = ", factorial(x),"<br>")
    }
  </script>
</body>
</html>
```

Возможно вызывать функцию с большим количеством аргументов, чем в ее объявлении. Для доступа к аргументам в этом случае, используется массив *arguments* . Это может быть полезно тогда, когда заранее неизвестно, сколько аргументов будет в функции. Чтобы определить реальное число аргументов в функции используется значение *arguments.length*.

Например, рассмотрим функцию, создающую списки HTML. Единственный формальный аргумент функции определяет тип списка, который создается – "U", если список ненумерованный или "O", если список нумерованный. Текст HTML-страницы, содержащей функцию *list,* представлен ниже:

```
<html>
<head>
  <script>
    function list(type) {
      // начинается список
      document.write("<" + type + "L>")
```

```
    for (var i = 1; i < list.arguments.length; i++) {
      // Повторить для каждого аргумента
      document.write("<LI>" + list.arguments[i]+"</LI>");
    }
    // заканчивается список
    document.write("</" + type + "L>")
  }
 </script>
</head>
<body>
 <h1>Нумерованный список:</h1>
 <script>
  list("O", "one", 1967, "three", "etc, etc...")
 </script>
 <h1>Ненумерованный список:</h1>
 <script>
  list("U", "one", 1967, "three", "etc, etc...")
 </script>
</body>
</html>
```

Таким образом, можно использовать любое число аргументов функции *list*, и каждый аргумент будет показан как отдельный пункт в списке, тип которого указан в первом аргументе функции.

2.3 Варианты заданий

Варианты заданий представлены в таблице ниже.

Таблица 2.4 – Варианты заданий

№	№ вопроса для провер-ки	Тип проверки
1	1	Количество слов в ответе не менее 30*
2	2	Установлены как минимум два переключателя
3	3	Количество слов в ответе не более 30*
4	3	Введенное значение является символьным(не содержит цифр)
5	2	Введено целочисленное значение
6	3	Установлены не более двух переключателей
7	1	Количество слов в ответе не более 20*
8	2	Установлены менее двух переключателей
9	3	Количество слов в ответе не менее 20*
10	1	Установлены менее трех переключателей
11	2	Хотя бы одно слово в ответе является записью целого числа*
12	2	Введенное значение – вещественное число
13	1	Хотя бы одно слово в ответе является записью веществ-го числа*
14	1	Введено целочисленное значение
15	3	Количество слов в ответе не менее 35*

16	3	Введенное значение – вещественное число
17	2	Хотя бы одно слово в ответе является записью целого числа*
18	3	Установлены не менее двух переключателей
19	1	Количество слов в ответе не более 25*
20	3	Установлены не менее трех переключателей

***** – слова разделены одним или более пробелами.

2.4 Порядок выполнения работы

1. Модифицировать страницу «Фотоальбом» (использовать HTML-страницы, разработанные при выполнении предыдущей лабораторной работы), реализовав вывод таблицы, содержащей фото, с использованием операторов циклов. Значения имен файлов фото и подписей к фото предварительно разместить в массивах ***fotos*** и ***titles***.

2. Модифицировать страницу «Мои интересы», реализовав вывод списков с использованием JavaScript-функции с переменным числом аргументов.

3. Добавить на страницах «Контакт» и «Тест по дисциплине «…»» функции проверки заполненности форм. В случае если какое-либо из полей формы осталось незаполненным при нажатии на кнопку отправить, вывести сообщение об ошибке и установить фокус на незаполненный элемент.

4. Добавить на странице «Контакт» текстовое поле «Телефон». Для полей «Фамилия Имя Отчество» и «Телефон» добавить функции специфической проверки значений. В случае если какое-либо из полей формы заполнено не верно, при нажатии на кнопку отправить, вывести сообщение об ошибке и установить фокус на неверно заполненный элемент. Формат правильных значений полей:

– Фамилия Имя Отчество – введено три слова, разделенные одним пробелом.

– Телефон – строка может состоять только из цифр; начинаться только с последовательности «+7» или «+3»; не содержит пробелов; количество цифр в строке от 9 до 11.

5. Добавить на странице «Тест по дисциплине «…»» функции специфической проверки значений полей в соответствии с вариантом задания, представленном в таблице 2.4. В случае не выполнения условия сформировать сообщение об ошибке и установить фокус на неверно заполненный элемент ввода.

6. Необходимо выполнить проверку разработанных JavaScript файлов с использованием сервиса jshint [11].

2.5 Содержание отчета

Цель работы, вариант задания, порядок выполнения работы, тексты разработанных JavaScript программ, иллюстрации полученных страниц

сайта, скриншоты результатов проверки кода с использованием соответствующих онлайн-сервисов, выводы по результатам работы.

2.6 Контрольные вопросы

2.6.1. Приведите теги HTML, внутри которых располагаются команды JavaScript?
2.6.2. Где могут располагаться скрипты JavaScript?
2.6.3. Расскажите об объектной модели браузера в JavaScript?
2.6.4. Как описываются переменные в JavaScript?
2.6.5. Расскажите о свойствах и методах строк в JavaScript?
2.6.6. Какие методы реализованы для объекта Array в языке JavaScript?
2.6.7. Как преобразуются типы данных в Java Script?
2.6.8. Расскажите о свойствах и методах массива elements?
2.6.9. Для чего используются в JavaScript объект form и массив forms?
2.6.10. Расскажите об операторах JavaScript?.

3 ЛАБОРАТОРНАЯ РАБОТА №3
«Исследование объектной модели документа (DOM) и системы событий JavaScript»

3.1 Цель работы

Исследовать структуру модели документа DOM. Изучить динамическую объектную модель документа, предоставляемую стандартом DOM и систему событий языка JavaScript, возможность хранения данных на стороне клиента. Приобрести практические навыки работы с событиями JavaScript, деревом документа, Local Storage и Cookies.

3.2 Краткие теоретические сведения

3.2.1 Объектная модель документа (DOM)

DOM (от англ. Document Object Model – «объектная модель документа») – это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями "родительский-дочерний".

Изначально различные браузеры имели собственные модели документов (DOM), несовместимые с остальными. Для обеспечения взаимной и обратной совместимости, специалисты международного консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM.

3.2.1.1 Уровни DOM

Изначальный стандарт DOM, также известный как «DOM уровень 1», был рекомендован W3C в конце 1998 года. DOM уровень 1 обеспечил полную модель для всего HTML- или XML-документа, включая способ изменения любой части документа.

DOM уровень 2 был опубликован в конце 2000 года. Консорциум W3C разделил рекомендации DOM на следующие пять категорий:

1. DOM Core (ядро DOM). Задает типовую модель в виде древовидной структуры для просмотра и изменения документа, содержащего элементы разметки.
2. DOM HTML. Определяет расширение ядра DOM для работы с HTML. Расширение DOM HTML обеспечивает возможности работы с документами HTML, используя синтаксис, типичный для традиционных объектных моделей JavaScript. В основном, это DOM

уровня 0 плюс средства работы с объектами, соответствующими элементам HTML.

3. DOM CSS. Определяет интерфейсы, необходимые для программного управления правилами CSS.
4. DOM Events (события DOM). Добавляет в DOM средства обработки событий. Такими событиями могут быть как привычные события интерфейса пользователя, например, щелчки кнопки мыши, так и специфические события, связанные с самой моделью DOM, происходящие при модификации дерева документа.
5. DOM XML. Определяет расширение ядра DOM для работы с XML. Расширение DOM XML призвано обеспечить решение специфических задач XML – для работы с разделами CDATA, инструкциями, пространством имен и т.д.

DOM уровень 3 – текущая версия спецификации DOM, опубликованная в апреле 2004 года, добавила интерфейс для сериализации документа в формат XML, и стандарт валидации структуры и содержимого документа. Так же было опубликовано ряд документов не достигших статуса спецификации, такие как: DOM Level 3 XPath, DOM Level 3 Views and Formatting, DOM Level 3 Events.

DOM уровень 4 доступен в виде снимка DOM Living Standard [12].

3.2.1.2 Объектная модель документа в соответствии с DOM

Модель DOM отображает реальную иерархическую структуру документа.

Рассмотрим, следующий HTML-документ:

```
<html>
 <head>
  <title>Пример</title>
 </head>
 <body>
   <h1>Пример простого документа</h1>
   <p>
     Начало абзаца <strong>выделенный текст</strong>
           продолжение абзаца
   </p>
   <img src="test.jpg" width="100" height="100" alt="тестовое изображение">
 </body>
</html>
```

Этот документ может быть представлен деревом, изображенным на рисунке 3.1:

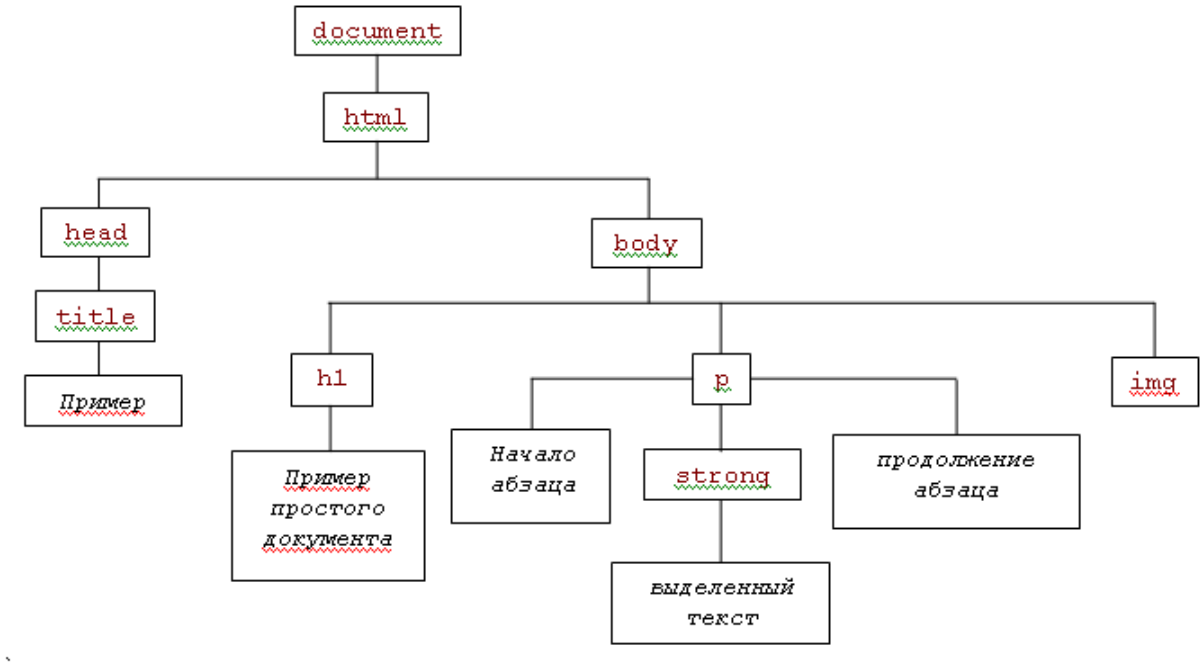

Рисунок 3.1 – Иерархия элементов документа.

Объектная модель документа представляется узлами (node), расположенными в виде иерархической структуры дерева.

Концепция объектной модели не привязана ни к какому конкретному представлению документа (HTML, XML, SGML). Она всего лишь описывает логическую организацию документа. Ее реализация в конкретной системе представления документов ставит в соответствие узлам реальные элементы.

В объектной модели документа, реализованной для HTML, в узлах могут находиться любые элементы HTML или текст, называемые узловыми элементами. Узлы в модели DOM документа HTML могут быть следующих типов:

Таблица 3.1 – Типы узлов документа

Код типа	Тип узла	Описание	Пример
1	ELEMENT	Элемент	<book>…</book>
2	ATTRIBUTE	Атрибут элемента	lang=«ru»
3	TEXT	Текстовый узел	Это текст
8	COMMENT	Комментарий	<!-- Комментарий -->
9	DOCUMENT	Узел документа	Document
10	DocumentType	Декларация типа документа	<!DOCTYPE html>

Во главе иерархии находится элемент типа DOCUMENT, который порождает узел HTML, с дочерними узлами HEAD и BODY (все типа ELEMENT).

Текстовое содержимое элемента HTML хранится в специальном текстовом узле (тип TEXT), порождаемом узлом элемента. Причем текстовых узлов может быть несколько, если содержимое элемента представлено размеченным текстом HTML. Вложенные элементы HTML и разбивают текстовое содержимое элемента на ряд текстовых узлов.

Например:

```
<p>Это <b>содержимое</b> абзаца <i>документа</i> HTML</p>
```

В объектной модели DOM этот фрагмет HTML будет представлен следующим иерархическим деревом:

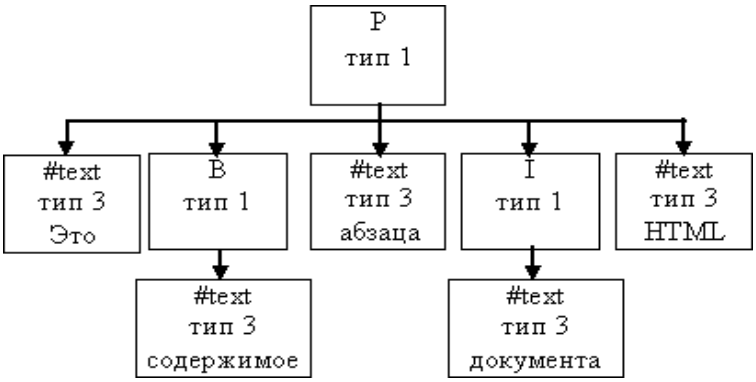

Рисунок 3.2 – Иерархия дочерних узлов элемента.

На рисунке 3.2 в текстовых узлах также представлено их содержимое. В DOM HTML содержимое, получаемое с помощью свойства *nodeValue* узла, может быть только у текстовых узлов и узлов комментария (свойство *nodeName* равно *#comment*), являющихся листьями иерархического дерева объектов документа.

В приведенных в приложении А таблицах А.1-А.3 представлены все основные методы и свойства объектов, используемые при работе с документом в соответствии со стандартом DOM 2.

Рассмотрим несколько примеров использования этих методов и свойств. Например, перемещение по объектной модели. На примере структуры, создаваемой вложенными списками, продемонстрируем, с помощью каких свойств объектов в модели DOM можно перемещаться по ее узлам:

```
<ul id="parent">
  <li id="Node1">Узел 1</li>
  <li id="Node2">Узел 2
    <ul id="inside">
      <li id="Child1">Потомок 1</li>
      <li id="Child2">Потомок 2</li>
```

```
        <li id="Child3">Потомок 3</li>
      </ul>
    </li>
     <li id="Node3">Узел 3</li>
  </ul>
```

В объектной модели документов этот фрагмент будет представлен в виде дерева, показанного на рисунке 3.3.

Элементы с именами *Node1*, *Node2* и *Node3* являются узлами-потомками элемента-родителя с именем *parent*. В семействе *childNodes* объекта *parent* хранятся ссылки на всех потомков этого элемента (*Node1*, *Node2* и *Node3*). Для получения ссылок на первого и последнего потомка узла в объектной модели предусмотрены соответственно свойства *firstChild* и *lastChild*.

Свойство *parentNode* объектов-потомков возвращает ссылку родителя объектов, поэтому значением этих свойств объектов *Node1*, *Node2* и *Node3* будет ссылка на узел *parent*.

Объекты *Node1*, *Node2* и *Node3* являются ближайшими родственниками одного поколения и открываются друг другу с помощью своих свойств *previousSibling* (предыдущий ближайший родственник) и *nextSibling* (следующий ближайший родственник). Если у элемента-узла нет соответствующих ближайших родственников, то эти свойства возвращают значение null.

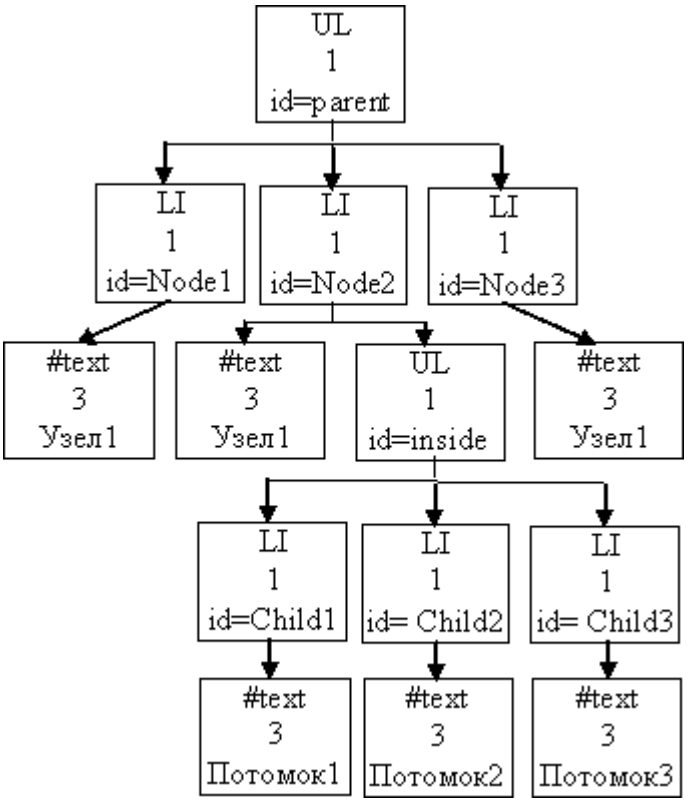

Рисунок 3.3 – Иерархия дочерних узлов элемента тега <UL>.

Для изменения, установки или получения содержимого текстового узла (узлы остальных типов не имеют текстового содержимого) в DOM используется свойство *nodeValue:*

```
// Объектная модель DOM
Node.childNodes[0].nodeValue = "Новое содержимое";
```

Также в спецификации DOM Parsing and Serialization определены свойство *innerHTML,* позволяющее изменять содержимое для любого элемента HTML:

```
// Модель DHTML
Node.innerHTML = "Новое <b>содержимое </b>";
```

Для доступа к объектной модели DOM загруженной в браузер страницы HTML прежде всего необходимо иметь ссылку на корневой элемент *#document*. Она создается автоматически и хранится в объекте *window.document*. Получить в DOM ссылку на корневой объект документа, соответствующий элементу, задаваемому тегом <*HTML*> можно единственным способом – использовать свойство *documentElement* объекта *document*. После чего, используя приведенные выше свойства узлов DOM, переместиться к требуемому элементу HTML и его содержимому.

Каждый узловой элемент порождается другим узловым элементом и может сам выступить родителем других узлов-элементов (за исключением узла *#text*). Объектная модель DOM предоставляет возможность в сценарии создать узел, соответствующий любому элементу HTML, задать значения его атрибутов, а затем встроить его в существующую модель документа, что приведет к автоматическому изменению вида документа в браузере.

Создание узла любого типа выполняется методами объекта *document* – *createElement()* создает узел типа 1, *createTextNode()* узел типа 3, *createComment()* узел типа 8 и *createDocumentFragment()* узел типа 11.

В метод *createElement()* передается строка, соответствующая открывающему тегу создаваемого элемента вместе с заданными атрибутами:

```
var newParagraph = document.createElement('<p id="par1">');
```

Если не надо задавать атрибуты вновь создаваемого элемента или используются их значения по умолчанию, то для создания нового элемента достаточно только его HTML-имя:

```
var newParagraph = document.createElement('p');
```

После создания, если потребуется, атрибуты становятся доступными как свойства созданного объекта:

```
newParagraph.style.backgroundColor = 'gray';
```

В методы createTextNode() и createComment() передается текстовая строка, которая является содержимым указанных узлов, получаемым через их свойство nodeValue:

```
var newTextElement = document.createTextNode('xxxxx');
var newComment     = document.createComment('yyyyyyyyyy');
```

У метода создания фрагмента документа *createDocumentFragment()* параметров нет:

```
var newFragment = document.createDocumentFragment();
```

При создании новых объектов и встраивания их в существующую структуру документа следует формировать объекты с правильной структурой, соответствующей их представлениям в объектной модели документа. Не соблюдение этих правил может привести к неправильно сформированному документу и, в конечном итоге, к его неправильному отображению браузером.

Рассмотрим на примере таблицы HTML процесс динамического создания элемента и встраивания его в документ HTML. Любая таблица объектной модели документа обязательно состоит, по крайней мере, из двух узлов: TABLE и TBODY. Поэтому при динамическом создании таблиц не следует забывать об этом обстоятельстве:

```
var Table  = document.createElement('TABLE');
var TBody = document.createElement('TBODY');
var Row    = document.createElement('TR');
var Cell1   = document.createElement('TD');
var Cell2   = Cell1.cloneNode();
Row.appendChild(Cell1);
Row.appendChild(Cell2);
Table.appendChild(TBody);
TBody.appendChild(Row);
document.body.appendChild(Table);
Cell1.appendChild(document.createTextNode('Ячейка 1'));
Cell2.appendChild(document.createTextNode('Ячейка 2'));
```

Процедура создания таблицы, собственно говоря, повторяет задание тегов в коде HTML документа (не пропуская тегов, вставляемых по умолчанию). Методом *appendChild*(*элемент*) любого узла осуществляется добавление к нему потомка – порождаемого этим объектом элемент HTML. В нашем примере этим методом в строку таблицы были добавлены две ячейки, в тело таблицы добавлена строка, а само тело было добавлено к объекту таблицы *Table*.

Для создания объекта, соответствующего второй ячейке таблицы использован метод *cloneNode()*, который создает объект – полную копию объекта, для которого он вызывается, включая его атрибуты и семейство *childNodes*, если в качестве параметра задано значение *true*. Если параметр

метода не задан, то используется значение по умолчанию *false*, при котором семейство ссылок на порождаемые объекты не копируется.

Для включения вновь созданной структуры в документ ее необходимо добавить к объекту *body* методом *appendChild()*.

Для манипуляции узлами используются их методы *removeNode(), replaceNode()* и *swapNode()*.

Метод *removeNode()* удаляет объект, для которого он вызван, из структуры документа. Его единственный параметр может принимать булевы значения *true* или *false*. Значение *true* предписывает удалить и все порожденные данным объектом объекты, тогда как значение *false* (умалчиваемое) удаляет только сам объект, оставляя в документе все подчиненные ему объекты.

Замену одного объекта другим можно осуществить методом *replaceNode()*, вызываемым для замещаемого объекта. Замещающий объект передается в качестве параметра метода. При замене объекта замещаемый объект удаляется из структуры документа.

Поменять местами два объекта в иерархии документа позволяет метод *swapNode()*. Меняются местами объект, метод которого вызывается, и объект, определяемый параметром метода:

```
function fncInterchange(row){
  row.swapNode(row.previousSibling);
}
```

Параметром этого метода является элемент, с которым необходимо поменять местами текущий элемент.

3.2.2 Система событий языка JavaScript и DOM Events

Еще до появления стандартов DOM, разработчики JavaScript реализовали возможность запуска функций или последовательности JavaScript операторов при наступлении определенных событий, связанных с действиями пользователя, изменением состояния документа и тп. Чтобы обеспечить (обработку) события с использованием соответствующего HTML-атрибута может быть задана функция JavaScript или группа из одного или нескольких JavaScript-операторов.

События JavaScript могут быть разделены на несколько категорий:

1) события, связанные с документами (события документа):
   - загрузка и выгрузка документов;
2) события, связанные с гиперсвязью (события гиперсвязи):
   - активизация гиперсвязи;
3) события, связанные с формой (события формы):
   - щелчки мыши на кнопках

- получение и потеря фокуса ввода и изменение содержимого полей ввода, областей текста и списков;

- выделение текста в полях ввода и областях текста;

4) события, связанные с мышью:

- помещение указателя мыши на гиперсвязь и активизация гиперсвязи;

- одинарные и двойные щелчки мыши на целевом объекте;

- перемещение указателя мыши над объектом.

В таблице 3.2 перечислены наименования событий, соответствующий HTML-атрибут и условия их возникновения:

Таблица 3.2 – События и условия их возникновения.

Имя события	Атрибут HTML	Условие возникновения события
Blur	onBlur	Потеря фокуса ввода элементом формы
Change	onChange	Изменение содержимого поля ввода иди области текста, либо выбор нового элемента списка
Click	onClick	Щелчок мыши на элементе формы или гиперсвязи
Focus	onFocus	Получение фокуса ввода элементом формы
Load	onLoad	Завершение загрузки документа
MouseOver	onMouseOver	Помещение указателя мыши на объект
MouseOut	onMouseOut	Помещение указателя мыши на объект
Select	onSelect	Выделение текста в поле ввода или области текста
Submit	onSubmit	Передача данных формы
Unload	onUnload	Выгрузка текущего документа и начало загрузки нового

Рассмотрим пример:

```
<script>
function selChange(seln) {
  selNum  = seln.group_N.selectedIndex;
  Isel       = seln.group_N.options[selNum].text;
  alert("Выбрано: "+Isel);
}
</script>
<form>
  Выберите вашу группу:
  <select name="group_N" onChange="selChange(this.form)">
    <option>И-11</option>
    <option>И-21</option>
    <option>И-22</option>
    <option>И-31</option>
  </select>
</form>
```

В этом примере объекту *select* с именем *group_N*, содержащему четыре элемента, определенные в тегах <*option*>, благодаря наличию *onChange="selChange(this.form)"*, присваивается обработчик события изменения значения объекта. В результате этого каждый раз при выборе нового элемента в объекте *select* вызывается функция *JavaScript* с именем *selChange(),* которая будет выводить содержимое выбранного пункта в окошке alert.

В стандарте DOM2 с появлением DOM Level 2 Events Specification возможности работы с событиями были существенно расширены, как появлением новых событий (например, событий клавиатуры), так и появлением возможности динамического назначения обработчиков событий.

Для добавления в страницу обработчика событий можно использовать новый метод *addEventListener()*.

Имеются три причины для использования этой функции вместо непосредственной установки HTML-аттрибута или свойства обработчика событий для объекта:

1. Возможность привязать к объекту сразу несколько обработчиков событий для одного и того же события. Когда обработчики событий привязаны таким образом, при наступлении соответствующего события происходит вызов каждого из обработчиков, но порядок их вызова произволен.

2. Возможность обработать события в течение фазы захвата (когда событие "спускается" к цели). Обработчики событий, привязанные к атрибутам типа onclick или onsubmit, вызываются только в фазе возврата.

3. Возможность привязать обработчик событий к текстовому узлу, что было невозможно до появления DOM2.

Синтаксис использования метода addEventListener() следующий:

объект.addEventListener(событие, обработчик, фазаЗахвата),

где:

объект – задает узел, к которому привязывается приемник события;

событие – соответствует событию, которое должно отслеживаться;

обработчик – задает функцию, которая должна вызываться, когда происходит соответствующее событие;

фазаЗахвата – является логическим значением, указывающим, когда должен вызываться обработчик событий – в фазе захвата (true) или в фазе возврата (false).

Когда в приложении возникает событие DOM, оно не просто срабатывает один раз в месте своего происхождения; оно отправляется в путь, состоящий из трёх фаз. Событие движется от корня документа к цели (фаза перехвата/захвата), затем срабатывает для цели события (фаза цели) и движется назад к корню документа (фаза всплытия/возврата).

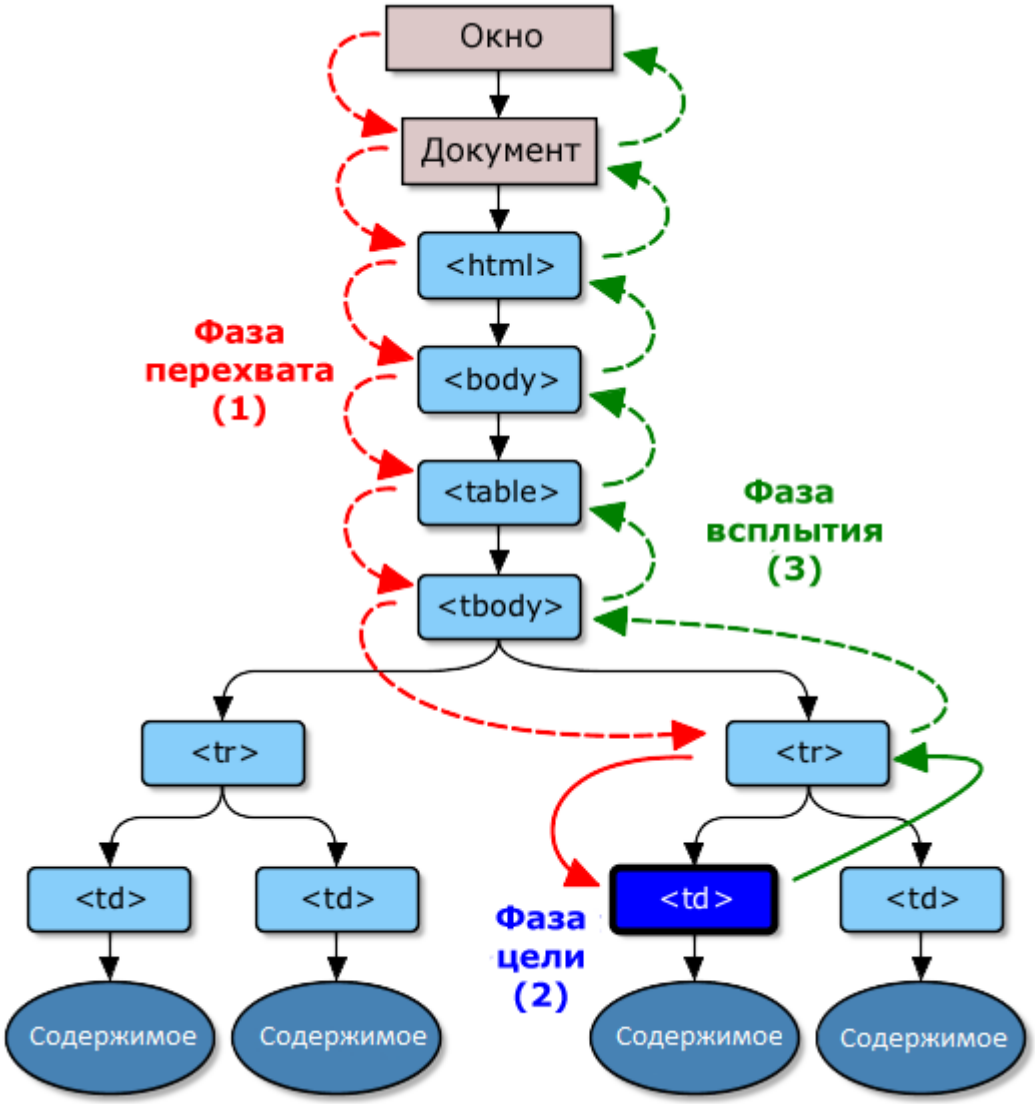

Рисунок 3.4 – Продвижения события

Фаза перехвата.

Первой фазой является фаза перехвата/захвата. Событие начинает своё путешествие в корне документа, проходит каждый слой дерева документа по направлению к цели, срабатывая для каждого узла по пути, пока её не достигнет. Задача фазы перехвата – наметить траекторию распространения, по которой событие будет двигаться в обратном направлении в фазе всплытия.

Как упоминалось ранее, обработчик можно вызвать в фазе перехвата, если в качестве третьего параметра для *addEventListener* указать true.

```
var form = document.querySelector('form');

form.addEventListener('click', function(event) {
    event.stopPropagation();
}, true);
```

Фаза цели.

Момент, когда событие достигает конечного объекта, известен как фаза цели. Событие срабатывает для целевого узла перед тем, как изменить направление своего продвижения и вернуться назад к самому внешнему уровню документа.

В случае с вложенными элементами, события мыши и указателя мыши всегда нацелены на наиболее глубоко расположенный вложенный элемент. Если обработчик вызывается для события click элемента , и пользователь кликает по элементу <p> внутри , этот <p> становится целью события. Тот факт, что события «всплывают» значит, что можно вызывать обработчик для кликов по (или любому другому родительскому элементу) и получать функцию обратного вызова при прохождении события.

Фаза всплытия.

После того, как событие сработало для цели, оно на этом не останавливается. Оно всплывает наверх (или же распространяется) по дереву документа, пока не достигнет его корня. Это значит, что то же самое событие срабатывает для родительского узла целевого элемента, затем для его родительского узла, и это продолжается пока не остаётся родительских элементов, для которых может сработать событие.

Всплытие события – это очень полезное явление. Оно делает необязательным установку обработчика события для конкретного элемента, для которого происходит событие; вместо этого мы можем установить обработчик для элемента выше по дереву документа и подождать пока событие его достигнет. Если бы события не всплывали, нам, возможно, в некоторых случаях пришлось бы устанавливать обработчики для множества разных элементов для гарантии, что событие не останется незамеченным.

Например, чтобы назначить функцию *changeColor()* на роль обработчика событий *mouseover* в фазе захвата для абзаца с id, равным *myText,* необходимо выполнить вызов:

```
document.getElementByld('myText').addEventListener("mouseover",
changeColor, true);
```

а чтобы добавить обработчик событий *swapImage()* в фазе возврата:

```
document.getElementByld('myText').addEventListener ("mouseover", swapI-
mage, false);
```

Следует отметить, что обработчики событий удаляются с помощью *removeEventListener()* с теми же аргументами, что и при их добавлении.

3.2.2.1 Объект Event

Браузеры, обеспечивающие поддержку модели событий DOM2, передают обработчикам событий в виде первого аргумента объект Event(событие), содержащий дополнительную информацию о произошедшем событии.

Объект Event создаётся, когда соответствующее событие происходит впервые; он сопровождает событие в его «путешествии» по дереву документа. Объект событие передаётся в качестве первого параметра функции, которую мы прописываем для приемника события как функцию обратного вызова. Этот объект можно использовать, чтобы получить доступ к огромному количеству информации о случившемся событии:

- type (строка). Это имя события.
- target (узел). Это узел DOM, который породил событие.
- currentTarget (узел). Это узел DOM, для которого на текущий момент работает обработчик события.
- bubbles (булево значение). Показывает является ли событие «всплывающим»(имеется ли фаза всплытия/возврата).
- preventDefault (функция). Позволяет предотвратить любое поведение, установленное по умолчанию, со стороны браузера в отношении события (например, предотвращение загрузки новой страницы вследствие события click элемента <a>).
- stopPropagation (функция). Предотвращает запуск следующих обработчиков дальше по цепочке событий, однако не предотвращает запуск дополнительных обработчиков события с тем же именем для текущего узла. (Мы поговорим об этом позже.)
- stopImmediatePropagation (функция). Предотвращает запуск обработчиков для любых узлов дальше по цепочке событий, а также дополнительных обработчиков события с тем же именем для текущего узла.
- cancelable (булево значение). Указывает на то, можно ли с помощью метода event.preventDefault предотвратить запуск действий по умолчанию в ответ на событие.
- defaultPrevented (булево значение). Указывает был ли вызван метод preventDefault для объекта событие.
- isTrusted (булево значение). Событие называется доверенным, если оно исходит от самого устройства, а не синтезируется в JavaScript.
- eventPhase (число). Это число указывает фазу, в которой на данный момент находится событие: ни в какой (0), перехвата (1), цели (2) или всплытия (3).
- timestamp (число). Это дата, когда произошло событие.

Объект event может принимать множество других свойств, однако они зависят от конкретного типа события. Например, для событий мыши

объекта event применяются свойства clientX и clientY чтобы определить размещение указателя в области просмотра.

3.2.2.2 События мыши и клавиатуры

События мыши в DOM2, соответствуют событиям в (X)HTML. Они представлены в таблице. В соответствии со спецификациями DOM2 не все события допускают фазу возврата и не все действия по умолчанию могут быть отменены:

Таблица 3.3 – События мыши

Событие	Фаза возврата	Возможность отмены
click	Да	Да
mousedown	Да	Да
mouseup	Да	Да
mouseover	Да	Да
mousemove	Да	Нет
mouseout	Да	Да

Когда происходит событие мыши, браузер наполняет объект Event дополнительной информацией:

Таблица 3.4 – Описание событий мыши

Свойство	Описание
altKey	Логическое значение, указывающее, была ли нажата клавиша ALT
button	Числовое значение, соответствующее использовавшейся кнопке мыши (обычно 0 соответствует левой, 1 – средней, а 2 – правой кнопкам)
clientX	Горизонтальная координата точки, в которой произошло событие, относительно окна содержимого браузера
clientY	Вертикальная координата точки, в которой произошло событие, относительно окна содержимого браузера
ctrlKey	Логическое значение, указывающее, была ли нажата клавиша CTRL
detail	Указывает число щелчков мыши (если таковые были вообще)
metaKey	Логическое значение, указывающее, была ли нажата клавиша META

relatedTarget	Ссылка на узел, связанный с событием; например, для mouseove г она указывает на узел, по которому указатель мыши движется, а для mouseout – на узел, который указатель мыши покидает
screenX	Горизонтальная координата точки, в которой произошло событие, относительно всего экрана
screenY	Вертикальная координата точки, в которой произошло событие, относительно всего экрана
shiftKey	Логическое значение, указывающее, была ли нажата клавиша SHIFT

В таблице ниже указаны связанные с клавиатурой события для браузеров с поддержкой DOM3.

Таблица 3.5 – События связанные с клавиатурой

Событие	Фаза возврата	Возможность отмены
keyup	Да	Да
keydown	Да	Да

Связанные с клавиатурой свойства объекта Event приводятся в таблице ниже:

Таблица 3.6 – Описание событий, связанных с клавиатурой

Свойство	Описание
altKey	Логическое значение, указывающее, была ли нажата клавиша ALT
charCode	Для печатаемых символов является числовым значением, указывающим значение Unicode нажатой клавиши
ctrlKey	Логическое значение, указывающее, была ли нажата клавиша CTRL
isChar	Логическое значение, указывающее, был ли при нажатии клавиши сгенерирован символ (полезное свойство, поскольку некоторые комбинации клавиш, например CTRL-ALT, символов не генерируют)
keyCode	Для непечатаемых символов является числовым значением, указывающим значение Unicode нажатой клавиши
metaKey	Логическое значение, указывающее, была ли нажата клавиша META
shiftKey	Логическое значение, указывающее, была ли нажата клавиша SHIFT

3.2.3 Объект Date

Для работы в JavaScript с датой и временем может быть использован объект Date. Этот объект имеет множество методов, предназначенных для получения такой информации. Кроме того, объекты Date можно создавать и изменять, например, путем сложения или вычитания значений дат получать новую дату. Для создания объекта Date применяется синтаксис:

```
dateObj = new Date(parameters);
```

где dateObj – переменная, в которую будет записан новый объект Date. Аргумент parameters может принимать следующие значения:

- пустой параметр, например date() в данном случае – системные дата и время.
- строку, представляющую дату и время в виде: "месяц, день, год, время", например "March 1, 2000, 17:00:00", время представлено в 24-часовом формате;
- значения года, месяца, дня, часа, минут, секунд. Например, строка "00,4,1,12,30,0" означает 1 апреля 2000 года, 12:30.
- целочисленные значения только для года, месяца и дня, например "00,5,1" означает 1 мая 2000 года, сразу после полночи, так как значения времени равны нулю.

Как уже говорилось ранее, данный объект имеет множество методов (свойств объект Date не имеет). Методы объекта Date приведены в таблице А.4 в приложении А.

В данном примере приведен HTML-документ, в заголовке которого выводится текущие дата и время.

```
<html>
<head>
<script>
function showh() {
  var theDate = new Date();
  document.writeln("<table cellpadding=5 width=100% border=0>" +
   "<tr><td width=95% bgcolor=gray align=left>" +
   "<font color=white>Date: " + theDate +
   "</font></td></tr></table><p>");
}
showh();
</script>
</head>
</html>
```

Для отображения часов на веб-странице возможно использование рекурсивной функции вызывающей себя с определенной задержкой. Для реализации задержки может использоваться функция setTimeout (синтаксис: setTimeout(expression, msec), где expression – выражение JavaScript или имя функции, msec – числовое значение задержки в миллисекундах). В

примере ниже 100 раз с интервалом 10 секунд происходит изменение цифры элемента формы:

```
<script>
i=0;
go();
function go() {
 document.gobarform.elements[0].value =i;
 if (i<100){
  setTimeout("go()",10000);
  i=i+1;
 }
}
</script>.
```

3.2.4 Хранение данных на стороне клиента

3.2.4.1 Cookie

Cookie – это небольшая порция текстовой информации, которую сервер передает браузеру. Браузер будет хранить эту информацию на стороне пользователя и передавать ее серверу с каждым запросом как часть HTTP заголовка. Основные области применения cookie приведены ниже:

- Аутентификации пользователя. Например, когда пользователь вводит логин и пароль на сайте, в cookie сохраняется его уникальный идентификатор, который используется в дальнейшем для идентификации пользователя.
- Хранения персональных предпочтений и настроек пользователя. Например, в cookie можно сохранять предпочитаемый язык пользователя.

Значение cookie может быть установлено сервером путем отправления специального HTTP заголовка

```
Set-Cookie: name=value.
```

После этого браузер создаст cookie с именем name и значением value. Также cookie могут устанавливаться на стороне клиента с помощью языка JavaScript. Для этого используется объект *document.cookie*. Кроме пары имя/значение, cookie может содержать срок действия, путь и доменное имя. Срок хранения cookie истекает в следующих случаях:

- В конце сеанса (например, когда браузер закрывается), если cookie не являются постоянными (так называемые сессионные cookie).
- Дата истечения была указана, и срок хранения вышел.

Браузер удалил cookie по запросу пользователя.

3.2.4.2 Объект localStorage

Для работы в JavaScript с временными данными пользователя может быть использован объект localStorage. Объект localStorage (также именуемый "HTML5-хранилище" или "WEB-хранилище") – это способ для WEB-страниц хранить пары ключ/значение на компьютере пользователя сайта. Указанные данные сохраняются после ухода с сайта и закрытия вкладки браузера. В отличие от Cookies эти данные не передаются на удаленный WEB-сервер. Объект localStorage доступен во всех современных браузерах. Максимальный размер сохраняемых данных – 5 Мб.

HTML5-хранилище базируется на именах пар ключ/значение. Данные могут быть любого типа, который поддерживает JavaScript, включая строки, логические, целые числа или числа с плавающей запятой. Однако в действительности данные хранятся в виде строки. Если вы сохраняете и извлекаете не строки, то надо будет использовать такие функции как parseInt() или parseFloat(), чтобы перевести полученные данные в корректные типы JavaScript. Для работы с HTML5-хранилищем используется следующие методы:

localStorage.getItem(ключ) – Получить значение

localStorage.setItem(ключ, данные) – Установить значение. Данный метод вернет NULL для несуществующего ключа.

localStorage.removeItem(ключ) – Удалить значение

localStorage.clear() – Очистка всего хранилища.

3.3 Варианты заданий

Исходным материалом служит проект из предыдущей лабораторной работы. Необходимо выполнить задания, перечисленные в п.3.4 с учетом указанных вариантов (берутся по правилу деления последней цифры номера зачетной книжки на их количество, уточняются преподавателем).

3.4 Порядок выполнения работы

1. Реализовать интерактивное графическое меню сайта, в соответствии со следующими требованиями:

- при наведении мыши на соответствующий пункт меню изображение, соответствующее этому пункту меняется на другое (в том случае если этот пункт не активен, т.е. не загружена соответствующая ему страница);
- сделать пункт «Мои интересы» в виде выпадающего меню, каждый элемент которого ведет на соответствующую ему подсекцию данной страницы. Выпадающее меню реализовать с помощью тегов <ul> и <li>. Данный список должен раскрываться в зависимости от варианта задания:

0	Выпадающее меню раскрывается по клику
1	Выпадающее меню раскрывается по наведению

2. Реализовать отображение в области меню сайта текущих даты и времени (обновление времени 1 раз в секунду). Формат даты определяется как остаток от деления последней цифры зачетной книжки на 4:

0	ЧЧ.ММ.ГГ День недели
1	ЧЧ Месяц ГГ
2	ЧЧ.ММ.ГГГГ День недели
3	ЧЧ Месяц ГГГГ

(«День недели» – полная запись дня недели на русском языке, «Месяц» – полная запись названия месяца на русском языке)

3. На странице «Контакт» добавить поле «Дата рождения», для которого реализовать всплывающий снизу элемент «календарь». Примерный вид элемента изображен на рисунке ниже:

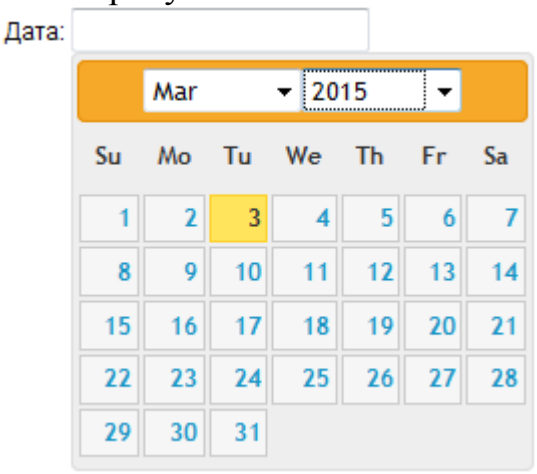

Рисунок 3.5– Элемент «календарь»

При этом элемент должен давать возможность менять дату, месяц и год. Верстка элемента должна быть выполнена с помощью тегов и <select> (для выбора месяца и года). Варианты задания элемента «календарь» приведены ниже:

Вариант	Язык интерфейса	Формат даты
0	English	месяц/день/год
1	Русский	день.месяц.год
2	Русский	месяц/день/год
3	English	день.месяц.год

Внимание! Пользоваться готовыми решениями для реализации календаря запрещается!

4. Реализовать динамическую проверку корректности заполнения пользователем формы на странице «Контакт» таким образом, чтобы при потере фокуса заполняемого поля осуществлялась проверка корректности его заполнения. В случае если поле заполнено корректно, оно должно быть подсвечено зеленым цветом, иначе оно должно быть подсвечено красным, а после данного поля должна появиться надпись, поясняющая характер ошибки. После исправления пользователем ошибки, надпись должна ис-

чезнуть. Если все поля формы заполнены корректно, должна стать активной кнопка «Отправить».

5. Реализовать открытие в динамически формируемом новом окне (блоке DIV) соответствующих больших фото при щелчке мыши по маленьким фото на странице «Фотоальбом».

6. Добавить страницу «История просмотра». На данной странице реализовать отображение двух таблиц:

- «История текущего сеанса» – в данной таблице отображается количество посещений каждой страницы за время текущего сеанса. Реализовать хранение этих данных в Local Storage.

- «История за все время» – в данной таблице отображается количество посещений каждой страницы за все время. Реализовать хранение этих данных в хранилище Cookies. Для выполнения этого задания необходимо создать два JavaScript метода: getCookie (name) и setCookie(name, value, exlpiration_days).

Для реализации страницы «История просмотра» необходимо добавить на каждую страницу вызов JavaScript функции, которая будет сохранять информацию о просмотре страницы в Local Storage и Cookies.

Примечание: для доступа к элементам страницы, назначения событий необходимо использовать функции DOM.

3.5 Содержание отчета

Цель работы, порядок выполнения работы, тексты разработанных JavaScript программ, выводы по результатам работы.

3.6 Контрольные вопросы

3.6.1. Назовите основные события в JavaScript?
3.6.2. Перечислите атрибуты HTML соответствующие возникающим событиям.
3.6.3. Как можно осуществить вызов обработчика события?
3.6.4. Что такое DOM? Какие стандарты DOM вы знаете?
3.6.5. Какие виды связей существуют между объектами DOM?
3.6.6. Какими функциями можно обратиться к первому/последнему ребенку текущего элемента?
3.6.7.Перечислите методы и свойства объекта document.
3.6.8. Как можно обратиться к родительскому узлу текущего элемента?
3.6.9. Какими функциями можно обратиться к элементам, находящимся на одном уровне с текущим элементом?
3.6.10. Какие параметры могут передаваться в конструктор при создании нового объекта класса Date?.

4 ЛАБОРАТОРНАЯ РАБОТА №4

«Исследование возможностей библиотеки jQuery»

4.1 Цель работы

Исследовать эффективность применения библиотек при разработке клиентских приложений на примере библиотеки jQuery. Изучить возможность программирования на клиентской стороне с использованием библиотеки jQuery. Приобрести практические навыки использования библиотеки jQuery для обработки форм, модификации содержимого HTML-страницы, создания эффектов анимации.

4.2 Краткие теоретические сведения

4.2.1 Основные понятия

JQuery [13] – кросс-браузерная JavaScript библиотека, предназначенная для упрощения разработки сценариев на стороне клиента. На сегодняшний момент, JQuery является одной из самых популярным JavaScript библиотек.

JQuery является свободным программным обеспечением с открытым исходным кодом, распространяемая под лицензией MIT License.

Ключевые возможности jQuery:

- Простой и мощный выбор элементов DOM с помощью кросс-браузерной библиотеки Sizzle (часть проекта JQuery);
- Возможность обхода и модификаций узлов DOM (включая поддержку CSS 1-3);
- Возможность создания и обработки событий;
- Возможность использования эффектов и анимации;
- Упрощение работы с AJAX;
- Встроенный парсер JSON;
- Возможность создания собственных расширений с помощью подключаемых модулей.

Использование JQuery позволяет отделить логику WEB-приложения на клиентской стороне от HTML-кода, подобно тому, как CSS отделяет визуализацию от HTML.

4.2.2 Использование jQuery

Для использования jQuery возможно скачать библиотеку с сайта http://jquery.com и разместить в доступном для web-приложения каталоге, после чего подключить с использованием тега script.

Существуют два варианта файлов jQuery: для использования в готовых приложениях (production) и для разработки (development). Вариант для разработки содержит комментарии и структурированный код. В сокращенной версии нет комментариев и код в ней не структурирован, но она имеет

меньший размер файла, и поэтому страницы с ней будут загружаться быстрее.

Кроме того, в настоящий момент существует две основные ветки версий jQuery 1.x и 2.x. Их отличия заключаются лишь в том, что в версиях 2.x перестали поддерживаться браузеры Internet Explorer (IE) версий 8 и ниже. Это позволило уменьшить размер jQuery более чем на 10%, а так же немного ускорить работу библиотеки. Но так как старые версии IE все еще используются пользователями, рекомендуется использовать ветку 1.x.

Например:

```
<script src="js/jquery-1.11.0.min." type="text/javascript"></script>
```

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. Например:

```
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
```

или

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></script>
```

4.2.3 Функция jQuery() и событие ready

Одной из основных функций в библиотеке jQuery является функция jQuery(). Она вызывается намного чаще других и поэтому для нее существует короткая форма записи – $(). Функция jQuery() позволяет производить поиск элементов на странице, создавать новые элементы по заданному HTML тексту и т.п. Возвращает эта функция *объект jQuery* – javascript-объект, который содержит коллекцию выбранных элементов страницы, а так же обладает множеством методов jQuery для работы с ними.

Выполнение любого JavaScript-кода обычно происходит после того, как часть страницы, с которой этот скрипт будет работать, уже загружена. В JavaScript для этого используются событие onload, которое происходит по окончанию загрузки всей страницы:

```
window.onload = function(){
  // вызов нужных функций скрипта
}
```

Однако onload происходит после того, как страница сформирована полностью, включая загрузку всех изображений, флеш-баннеров и видеороликов. В то время как структура дерева DOM (элементов страницы), с которой обычно и работает скрипт, оказывается готова гораздо раньше. В результате скрипт запускается значительно позднее, чем мог бы. На этот случай в jQuery есть метод ready, вызов которого осуществляется в момент готовности дерева DOM:

```
$(document).ready( function(){
  // вызов нужных функций скрипта
});
```

В сокращенном виде данная функция может быть вызвана так:

```
$().ready(function(){
  // вызов нужных функций скрипта
});
```

или так:

```
$(function(){
  // вызов нужных функций скрипта
});
```

4.2.4 Выбор элементов и цепочки вызовов jQuery

Базовая команда библиотеки выглядит следующим образом:

$(селектор).действие()

где:

- $() – вызов функции jQuery();
- (селектор) – селектор для выбора элементов в HTML коде страницы; селекторы jQuery основаны на селекторах CSS, но расширяют возможности CSS по выбору элементов (с полным перечнем селекторов, поддерживаемых jQuery можно ознакомиться в [13, раздел selectors]);
- действие() – это методы jQuery, которые должны быть выполнены над выбранными элементами.

Примеры:

```
$("p").css("fontSize","20px");//Установим размер шрифта всех абзацев равным 20 пикселям
$(".el3").hide();//Спрячем все элементы с class=el3
$("#el2,.el3").css("fontWeight","bold");//Сделаем жирным шрифт элементов с id=el2 и class=el3
$("#bigIt").html(<p>Новое содержимое!</p>); //Изменим все html-содержимое элемента с id = bigIt, на заданное в методе html.
$("[href]").css("fontSize","20px"); //Установим размер шрифта всех элементов имеющих атрибут href равным 20 пикселям
```

Важной особенностью большинства методов jQuery, является возможность связывать их в цепочки. Методы, манипулирующие элементами документа, обычно возвращают объекты jQuery для дальнейшего использования, что позволяет делать несколько вызовов подряд:

```
$("#bigIt").empty().attr("class", "noContent");
```

// в результате, у элемента с идентификатором bigIt будет удалено все содержимое, после чего ему будет установлен класс noContent.

Такие цепочки могут состоять из гораздо большего числа методов. Для улучшения читабельности кода, каждый вызов метода в цепочке рекомендуется писать с новой строки:

```
$("div")                  // найдем все div-элементы
  .parent()               // найдем их родительские элементы
  .css("heigth", "10px")  // установим последним высоту в 10 пикселей
  .fadeTo(0, 0.5)         // установим им  прозрачность в 50%
```

```
.addClass("divOwner");  // добавим им  класс divOwner
```

В jQuery также предусмотрены функции позволяющие изменять набор элементов, выбранный с помощью селекторов. Например, расширить выборку можно с использованием метода add():

```
// добавить к  четным изображениям изображения, содержащие заданные подстроки в src, а также и все элементы label
var labelElems = document.getElementsByTagName("label");
var jq = $('img[src*=sevsu]');
$('img:even').add('img[src*=is]').add(jq)
      .add(labelElems).css("border", "thick double red");
```

Сократить выборку с применением фильтра возможно с использованием метода filter:

```
// удалить из выборки элементы с указанным src или индексом 4
$('img').filter(function (index) {
    return this.getAttribute("src") == "peony.png" || index == 4;
}).css("border", "thick solid red")
```

При выборке элементов с использованием селекторов, даже если найден только один элемент, результатом всё равно будет jQuery-коллекция. Для получения одного элемента из jQuery-коллекции есть несколько способов:

1. Метод get(индекс), работает так же, как прямой доступ:

```
alert( $('body').get(0) ); // BODY
```

Если элемента с таким номером нет – вызов get возвратит undefined.

2. Метод eq(индекс) возвращает коллекцию из одного элемента – с данным номером. Он отличается от метода get(индекс) и прямого обращения по индексу тем, что возвращает именно jQuery-коллекцию с одним элементом, а не сам элемент.

```
// DOM-элемент для первой ссылки
$('a').get(0);
// jQuery-объект из одного элемента: первой ссылки
$('a').eq(0);
```

Во втором случае вызов eq создает новую jQuery-коллекцию, добавляет в нее нулевой элемент и возвращает его. Это удобно, если мы хотим дальше работать с этим элементом, используя методы jQuery. Если элемента с таким номером нет – eq возвратит пустую коллекцию.

Полный список методов работы с выборкой элементов доступен в [13, раздел traversing].

4.2.5 Управление элементами и манипулирование DOM

jQuery содержит большое количество методов для работы с элементами, включая получение, добавление и изменение атрибутов, работу с

классами и свойствами CSS, получение и изменение текста внутри элемента и содержимого HTML.

Так attr(name) – получает значение атрибута с указанным именем для первого элемента в объекте jQuery, а attr(name, value) – устанавливает указанное значение атрибута с указанным именем для всех элементов в объекте jQuery. При необходимости установки нескольких свойств может быть использован объект JavaScript: свойства этого объекта интерпретируются как имена атрибутов, а значения свойств – как значения атрибутов. Например, в следующем фрагменте устанавливаются свойства src и style для всех изображений документа:

```
$(document).ready(function () {
   $('img').attr({
     src: 'noimage.png',
     style: 'border: thick solid red'
   });
});
```

Для управления классами элементов в jQuery могут быть использованы такие функции как:

- *addClass(name name)* – добавляет ко всем элементам в объекте jQuery указанный класс.
- *removeClass(name name)* – удаляет классы в объекте jQuery.
- *toggleClass(name)* – добавляет или удаляет класс (в зависимости от их наличия).

Определить наличие класса у элемента можно при помощи метода hasClass. В примере ниже демонстрируется использование данных методов:

```
$('img').addClass("redBorder");
$('img:even').removeClass("redBorder").addClass("blueBorder");
console.log("All elements: " + $('img').hasClass('redBorder'));
$('img').each(function (index, elem) {
  console.log("Element "+index+" ("+elem.src +") has class redBorder: "+
  $(elem).hasClass('redBorder'));
});
```

В данном примере для перебора элементов набора был использован метод each. Его синтаксис похож на forEach массива:

```
.each( function(index, item))
```

Он выполняет для каждого элемента коллекции функцию-аргумент, и передаёт ей номер index и очередной элемент item.

jQuery также предоставляет набор методов, которые упрощают работу с CSS:

Таблица 4.1 – jQuery методы для манипулирования CSS свойствами

css(name)	Получает значение указанного свойства первого элемента в объекте jQuery
css(name, value)	Устанавливает значение для указанного свойства для всех элементов в объекте jQuery
css(map)	Устанавливает несколько свойств для всех элементов в объекте jQuery при помощи объекта
css(name, function)	Устанавливает значение для указанного свойства для всех элементов в объекте jQuery с использованием функции

В примере ниже используется динамическое изменение CSS свойств с использованием *css(name, function).* Аргументами функции являются индекс элемента и текущее значение свойства. Переменная this указывает на HTMLElement объект для текущего элемента, а также возвращается значение, которое необходимо установить:

```
$('label').css("border", function (index, currentValue) {
    if ($(this).hasClass('redBorder')) {
      return "thick solid red";
    } else if (index % 2 == 1) {
      return "thick double blue";
    }
 });
```

Еще одной из важных возможностей, предоставляемых jQuery, является изменение структуры самого HTML документа, известное как *манипулирование* DOM. С помощью функций, предоставляемых jQuery, возможно изменять структуру, включая вставку элементов в качестве дочерних, родительских или сестринских элементов, создавать новые элементы, перемещать элементы из одной части документа в другую, полностью удалять элементы.

Так, например, можно создать новые элементы, если добавить строку HTML фрагмента в $() функцию. jQuery разбирает строку и создает соответствующие DOM объекты:

```
var newElems = $('<div class="dcell"><img src="lily.png"/></div>');
```

Созданные объекты могут быть впоследствии добавлены в DOM-дерево документа с использованием таких функций как *append, prepend, appendTo, prependTo*:

```
$('#row1').append(newElems);
```

Помимо добавления существует возможность также и замены элементов (функции *replaceWith*, *replaceAll*), удаления (функции *empty, remove*). В примере ниже выбираются элементы img, чьи src атрибуты содержат is и avt, берутся их родительские элементы и удаляются:

```
$(document).ready(function () {
   $('img[src*=is], img[src*=avt]').parent().remove();
});
```

Для получения/изменения содержимого текстового узла элемента, html-кода элемента или значения поля формы используются следующие функции:

- text() – задает или возвращает текстовое содержимое выбранных элементов;
- html() – задает или возвращает содержание отдельных элементов (включая HTML разметку);
- val() – задает или возвращает значение поля формы.

В следующем примере демонстрируется, их использование:

```
var html = $('div.test').html();
console.log(html);
$("div.test').html("Новое содержимое <b>блока</b>");
```

Полный список методов, управления элементами и манипуляции DOM доступен в [13, раздел manipulation].

4.2.6 Обработка событий

jQuery облегчает использование обработчиков событий JavaScript и расширяет их функциональность. Общий вид определения обработчиков jQuery:

```
$(селектор).обработчик_события(function(){код_обработчика_события});
```

Пример:

```
$("#but1").click(function(){
   // обработчик события, который будет
   // выводить сообщение при нажатии на кнопку с id=but1
   alert("Вы нажали на кнопку с id=but1")
});
```

Большинство DOM-событий имеют эквивалентный jQuery – метод:

Таблица 4.2 – Методы jQuery

События мыши	События клавиатуры	События форм	События окна/документа
Click	keypress	Submit	load
Dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

Но jQuery также содержит множество методов управления событиями. Например, метод *one()* позволяет создавать обработчики, которые срабатывают только один раз, метод *toggle()* позволяет переключаться между различными обработчиками событий при щелчке мышью и тп. Например:

```
$( "#target" ).toggle(function() {
  alert( "Первый обработчик .toggle() вызван." );
}, function() {
  alert( "Второй обработчик .toggle() вызван." );
});
```

Полный список событий и функций управления представлен в [13, раздел events].

4.2.7 Использование эффектов и анимации

В библиотеку jQuery включены различные эффекты и возможность создания и управления анимацией. Существуют эффекты, предназначенные для того, чтобы показать или скрыть элементы, передвигать элементы, изменять различные свойства элементов. В таблице ниже представлены базовые функции отображения и скрытия элементов:

Таблица 4.3 – Базовые функции отображения и скрытия элементов

Метод	Описание
hide()	Прячет все элементы в объекте jQuery
hide(time) hide(time, easing)	Прячет элементы в объекте jQuery на определенный период времени, опционально можно указать скорость анимации
hide(time, function) hide(time, easing, function)	Прячет элементы в объекте jQuery на определенный период времени, опционально можно указать скорость анимации и функцию, которая будет вызвана после завершения эффекта (анимации)
show()	Показывает все элементы в объекте jQuery
show(time)	Показывает элементы в объекте jQuery на определенный период времени, опционально можно указать скорость анимации
show(time, function) show(time, easing, function)	Показывает элементы в объекте jQuery на определенный период времени, опционально можно указать скорость анимации и функцию, которая будет вызвана после завершения эффекта

Метод	Описание
toggle()	Меняет (переключает) видимость всех элементов в объекте jQuery
toggle(time) toggle(time, easing)	Переключает видимость всех элементов в объекте jQuery на определенный период времени, опционально можно указать скорость анимации
toggle(time, function) toggle(time, easing, function)	Переключает видимость всех элементов в объекте jQuery на определенный период времени, опционально можно указать скорость анимации и функцию, которая будет вызвана после завершения эффекта
toggle(boolean)	Переключает видимость всех элементов в объекте jQuery в соответствии с аргументом

Для создания эффектов скольжения элементов могут быть использованы такие функции как slideDown, slideUp, а для создания эффектов появления/исчезновения (изменения прозрачности) такие методы как fadeIn и fadeOut.

jQuery не ограничивает нас использованием базовых эффектов, скольжения и изменения прозрачности. Также можно создавать и собственные эффекты, например, изменяя CSS-свойства элемента с использованием функции animate:

```
$("#button").click(function (e) {
        $('h1').animate({
          height: $('h1').height() + $('form').height() + 10,
          width: ($('form').width())
});
```

Для анимации обычно указываются только конечные значения. Начальными значениями для анимации является текущее значение свойств, для которых создается анимация.

Полный список функций эффектов и управления анимацией представлен в [13, раздел effects].

4.3 Варианты заданий

Выполнить задания п.4.4 на основе ранее созданного сайта.

4.4 Порядок выполнения работы

1. Модифицировать JavaScript, разработанные при выполнении предыдущих лабораторных работ, используя везде, где возможно, библиотеку jQuery для выполнения поставленных задач.

2. Модифицировать страницу «Фотоальбом», добавив возможность просмотра увеличенных версий изображений. С использованием возможностей библиотеки jQuery реализовать просмотр фото, как показано ниже:

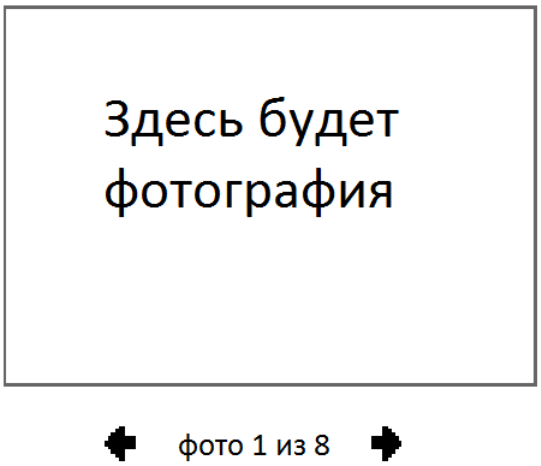

Рисунок 4.1 – Внешний вид просмотра фото

При щелчке по стрелкам должна происходить анимированная смена изображений. Изменяться должны также и тексты всплывающих сообщений (alt, title).

3. Реализовать возможность отображения всплывающего блока Popover, который показывается рядом с элементом, если на него наводится указатель мыши и который исчезает через M секунд после того, как указатель мыши покидает элемент.

Примерный макет разрабатываемого функционала изображен на рисунке ниже:

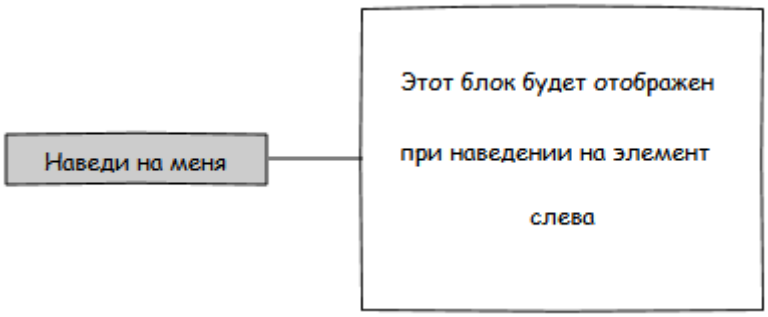

Рисунок 4.2 – Внешний вид элемента Popover

Использовать разработанный код для отображения подсказок уточняющих формат ввода пользователю в формах.

3.4. Реализовать возможность отображения всплывающего модального окна, которое показывается в центре экрана. При этом задний фон во-

круг модального окна должен размываться. Примерный макет разрабатываемого функционала изображен на рисунке ниже:

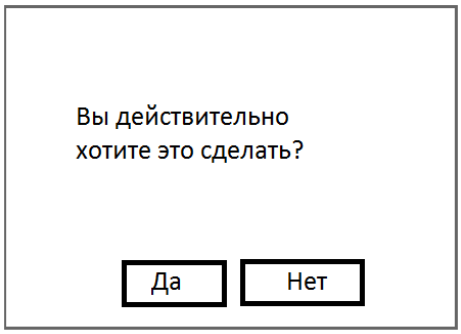

Рисунок 4.3 – Внешний вид элемента «Модальное окно».

4.5 Содержание отчета

Цель работы, порядок выполнения работы, тексты переработанных HTML-документов и разработанных JavaScript, скриншоты разработанных Web-страниц, выводы по результатам работы

4.6 Контрольные вопросы

4.6.1. Как выбирать элементы, используя class или id?
4.6.2. Как проверить имеет ли элемент специфический класс?
4.6.3. Как проверить существование элемента?
4.6.4. Как определить состояние переключаемого элемента (видим он или нет)?
4.6.5. Как разрешать/запрещать использование элемента?
4.6.6. Как отметить/снять отметку с элемента checkbox?
4.6.7. Как получать значения выбранной опции элемента select?.

5 ЛАБОРАТОРНАЯ РАБОТА №5

«Исследование возможностей ускорения разработки веб-приложений с использованием HAML»

5.1 Цель работы

Исследовать влияние препроцессоров на разработку клиентских приложений. Изучить основные возможности языка HAML. Приобрести практические навыки реализации веб-страниц с использованием данной технологии.

5.2 Краткие теоретические сведения

5.2.1 Препроцессорный язык разметки гипертекста HAML

Препроцессор – это программа, которая использует свой синтаксис (метаязык), а затем преобразовывает его в другой язык (например, в HTML код, если это препроцессор HTML). Препроцессоры позволяют автоматизировать ряд задач (например, закрытие тегов или расстановка скобок/отступов), которые в ином случае требуют времени и внимания программиста, а кроме того увеличить читабельность HTML кода и уменьшить его объем, что приводит к ускорению разработки представлений веб-приложения. Примерами препроцессоров для HTML являются HAML [14-16] и Jade[17].

Аббревиатура HAML может применяться для названия как препроцессора, так и самого языка разметки (HTML Abstraction Markup Language). Для того, чтобы убедиться в эффективности использования HAML достаточно сравнить небольшие эквивалентные фрагменты HTML и HAML файлов:

```
<div id='content'>
 <div class='left column'>
   <h2>Добро пожаловать!</h2>
   <p>Рады приветствовать Вас!</p>
 </div>
 <div class="right column">
   Новости
 </div>
</div>
```

```
#content
 .left.column
   %h2 Добро пожаловать!
   %p Рады приветствовать Вас!
 .right.column
    Новости
```

5.2.2 Элементы языка

5.2.2.1 Теги и структура документа

Все теги HAML начинаются с символа "%", стоящего в начале строки. Разделения тегов на открывающий и закрывающий нет, поскольку за-

крывающий тег генерируется препроцессором автоматически. Следом за символом "%" следует название тега. Например, следующий текст HAML:

```
%html
%head
%body
%div
```

преобразуется в код вида:

```
<html></html>
<head></head>
<body></body>
<div></div>
```

Для создания структурированного HTML кода (с использованием вложения тегов) необходимо **использовать отступы**. Это могут быть пробелы или знаки табуляции, однако на протяжении всего документа следует придерживаться одного стандарта создания отступов. Изменим наш пример:

```
%html
    %head
    %body
        %div
```

Тогда на выходе HAML препроцессора получим HTML код:

```
<html>
    <head></head>
    <body>
        <div></div>
    </body>
</html>
```

5.2.2.2 Текст и комментарии

Если тег содержит текстовую информацию, то ее следует написать на текущей строке через пробел или на следующей строке с отступом. Например:

```
%p текст
```

будет преобразовано в:

```
<p>текст</p>
```

тогда как

```
%p
    текст
```

будет преобразовано в:

```
<p>
    текст
</p>
```

Текст может быть любым, если он отличается от интерпретируемых HAML конструкций (в том числе и тегами HTML) – он просто будет перенесен в сгенерированный документ. Однако при необходимости интерпретируемый код можно экранировать символом “\” в начале строки – такая строка также будет перенесена без изменений.

Для комментирования текста используется символ “/”; все, что расположено в строке после него, будет преобразовано в комментарий в документе HTML. Если необходим многострочный комментарий, то символ “/” ставится на новой строчке, а текст располагается ниже и с отступами:

```
%div
        / текст
/
        комментарий
        многострочный
```

Результатом будет следующее:

```
<div>
        <!-- текст -->
</div>
<!--
        комментарий
        многострочный
-->
```

Если же комментарий нужно оставить только в документе HAML, то используется конструкция вида:

```
-# текст
```

5.2.2.3 Атрибуты, классы и id

Есть несколько способов объявления атрибутов в HAML. Объявить их можно как с помощью фигурных скобок {}, так и с помощью обычных (), первый подход применяется в Ruby, второй в HTML синтаксисе. Например, любая из строк:

```
%img{:src => "shay.jpg", :alt => "Shay Howe"}
%img{src: "shay.jpg", alt: "Shay Howe"}
%img(src="shay.jpg" alt="Shay Howe")
```

преобразуется в строку:

```
<img alt='Shay Howe' src='shay.jpg'>
```

Атрибуты :class и :id могут использоваться точно так же, однако в HAML для их использования существует сокращенный синтаксис. Для задания класса можно указать его имя сразу после имени элемента через точку, а для задания id используется символ “#”:

```
%p.one
%div#main
```

Результат:

```
<p class='one'></p>
<div id='main'></div>
```

Если необходимо указать несколько классов у одного тега, то их можно перечислять подряд через точку. Следует отметить, что в связи с тем, что тег DIV является самым распространенным средством верстки со-

временных сайтов, для элементов этого типа нет необходимости указывать сокращенное название %div. То есть, конструкция вида:

```
.one#main
```

будет преобразована как:

```
<div class='one' id='main'></div>
```

5.2.2.4 Переменные и массивы

Переменные имеют крайне простой синтаксис объявления. Сперва следует символ «-», затем собственно сама переменная и ее значение. Например:

```
- name = 'Значение переменной'
%p
        = name
```

преобразуется в:

```
<p>
        Значение переменной
</p>
```

Объявление массива и присвоение значения аналогично переменным:

```
- @array_1 = ['Item1', 'Item2', 'Item3'];
 - @array_2 = {'key1' => 'Item1', 'key2' => 'Item2', 'key3' => 'Item3'};
```

5.2.2.5 Операторы

Для упрощения процесса написания кода и придания последнему дополнительной гибкости, в HAML поддерживаются стандартные операторы условий и циклов. Например, условный оператор будет выглядеть так:

```
- if name == 1
        -# первая ветвь условия
- elsif name == 2
        -# вторая ветвь условия
- else
        -# действия в остальных случаях
```

Использование циклов позволяет существенно упростить верстку однотипного кода, сокращая затраты времени программиста. Например, на основе объявленного в предыдущем пункте массива следующим образом:

```
%ul
        -@array_2.each do |key, value|
                %li #{key}:#{value}
```

можно создать такой список:

```
<ul>
        <li>key1:Item1</li>
        <li>key2:Item2</li>
        <li>key3:Item3</li>
</ul>
```

В качестве примера использования операторов рассмотрим создание блока постраничной навигации:

```
.nav.nav-pagination
        %ol.pages
                - (1..6).each do |i|
                        %li.page
                                - if i == 3
                                        %span.current #{i}
                                - else
                                        %a{:href => '#'} #{i}
```

После компиляции будет получен следующий результат:

```
<div class="nav nav-pagination">
        <ol class="pages">
                <li class="page">
                        <a href="#">1</a>
                </li>
                <li class="page">
                        <a href="#">2</a>
                </li>
                <li class="page">
                        <span class="current">3</span>
                </li>
                <li class="page">
                        <a href="#">4</a>
                </li>
                <li class="page">
                        <a href="#">5</a>
                </li>
                <li class="page">
                        <a href="#">6</a>
                </li>
        </ol>
 </div>
```

5.2.2.6 Вставка кода из других файлов и другие особенности

HAML позволяет вставлять блоки кода из внешних файлов в текущий код. Например, необходимо создать множество страниц с однотипными блоками типа меню или постраничной навигации. Крайне удобно расположить код таких блоков в отдельных файлах, а на страницах в необходимых местах лишь вызывать подключение требуемого блока. Следует отметить также, что HAML позволяет передать переменную со значением в подключаемый файл, что делает данный механизм еще удобнее и пластичнее.

5.2.2.7 Установка и использование HAML

Для использования HAML необходимо использовать один из компиляторов, преобразующих HAML код в HTML. Существуют различные версии HAML компилляторов для различных языков программирования (Ruby, PHP и тп).

В настоящей и последуюших двух лабораторных работах рекомендуется использовать универсальный компилятор Prepos [3].

Эта программа компилирует различные типы файлов, в том числе, Haml, Sass, Less и CoffeeScript.

Prepos позволяет преобразовавать файлы в реальном режиме времени (сразу же после изменения), размещать скомпилированный или минифицированный файл, либо в той же папке (например, изменяя расширение исходного файла с haml на html), либо в указанной.

Кроме того, Prepros способен автоматически перезагружать открытую страницу в браузере, когда какой-то файл был изменен и сохранен.

5.3 Варианты заданий

Работа выполняется на основе проекта, полученого в ходе выполнения предыдущих лабораторных работ. Небходимо доработать проект в соответствии с указаними, данными в п.5.4 данной лабораторной работы.

5.4 Порядок выполнения работы

1. Необходимо с использованием языка HAML создать шаблон страниц пользовательской части сайта, содержащий общие блоки информации, такие как меню, ‘шапка’, ‘подвал’ и др. Шаблон разместить в отдельном файле.

2. Реализовать страницы «Главная», «Обо мне», «Контакты», «Учеба» с использованием HAML, поместить результаты в отдельные файлы.

3. Реализовать страницу «Фотоальбом» с использованием языка HAML. При реализации страницы воспользоваться циклическими операторами языка HAML для автоматизации процесса верстки фотоальбома. Информацию об изображениях хранить с использованием массива.

5.5 Содержание отчета

Цель работы, порядок выполнения работы, тексты разработанных HAML-документов, изображения разработанных Web-страниц, выводы по результатам работы.

5.6 Контрольные вопросы

5.6.1. Для чего используют HTML препроцессоры?
5.6.2. Что такое HAML?
5.6.3. Расскажите о структуре HAML-документов?
5.6.4. Как описать тег в HAML? Приведите примеры.
5.6.5. Как реализуются комментарии в HAML?
5.6.6. Атрибут и класс – опишите методы реализации в HAML.
5.6.7. Однострочный и многострочный текст – в чем разница при оформлении?

5.6.8. Как реализован условный оператор в HAML?
5.6.9. Приведите пример работы с массивами в HAML.
5.6.10. Как реализовать циклические действия в HAML?
5.6.11. Каким образом можно использовать внешние файлы в HAML?

6 ЛАБОРАТОРНАЯ РАБОТА №6

«Исследование возможностей ускорения разработки клиентских приложений с использованием SASS»

6.1 Цель работы

Исследовать влияние препроцессоров на разработку клиентских приложений. Изучить возможности метаязыка SASS для упрощения разработки файлов каскадных таблиц стилей. Приобрести практические навыки использования SASS/SCSS при реализации веб-страниц.

6.2 Краткие теоретические сведения

6.2.1 Препроцессорные языки обработки CSS

При разработке больших Web-приложений файл стилей зачастую также существенно увеличивается. При этом появляются повторяющиеся участки кода, а структура CSS-документа становится довольно сложной и часто запутанной. Кроме того, классический CSS не поддерживает такие возможности программирования, как переменные, циклы и др., что усложняет написание кода и делает этот процесс достаточно долгим.

Решением стали метаязыки, существенно упрощающие работу с CSS и добавляющие недостающий функционал. Первым появился SASS (Syntactically Awesome StyleSheets) [18], а затем LESS [19]. Характерной особенностью LESS был синтаксис, основанный на CSS. В дальнейшем LESS оказал своё влияние на препроцессор SASS, в котором теперь используется похожий синтаксис – данную модификацию назвали SCSS (SASSy CSS). Так что в настоящий момент LESS и SASS во многом похожи, различаясь лишь в деталях.

Оба препроцессора включают в себя переменные, функции, повторяемые компоненты и другие улучшения синтаксиса, которые позволяют сэкономить время и упростить процесс написания кода. Кроме того, существуют и другие препроцессоры CSS. Ниже будет рассмотрен язык SASS.

6.2.2 Элементы языка SASS

6.2.2.1 SASS и SCSS

SCSS – "диалект" языка SASS. Отличие SCSS от SASS заключается в том, что SCSS больше похож на обычный CSS код. То есть, при форматировании кода используются фигурные скобки и в конце строк ставится символ “;”, в то время как в SASS ничего подобного нет. Например:

SASS	*SCSS*
$blue: #3bbfce *$margin: 16px* *.content-navigation*	*$blue: #3bbfce;* *$margin: 16px;* *.content-navigation {*

```
  border-color: $blue
  color: darken($blue, 9%)

.border
  padding: $margin / 2
  margin: $margin / 2
  border-color: $blue
```

```
  border-color: $blue;
  color: darken($blue, 9%);
}
.border {
  padding: $margin / 2;
  margin: $margin / 2;
  border-color: $blue;
}
```

Код SASS хранится в файлах с расширением .sass; SCSS – в файлах типа .scss. Также следует отметить, что обычный CSS-код вполне вписывается в SCSS синтаксис, однако не совместим с SASS.

6.2.2.2 Использование переменных

Одним из существенных недостатков CSS является то, что в нем не предусмотрены переменные. Например, в описаниях стилей есть несколько классов с одинаковым значением одного из параметров. Тогда, если возникнет необходимость изменить значение этого параметра, разработчик должен будет пересмотреть все CSS-файлы и вручную (или автоматической заменой) поменять старое значение на новое там, где это нужно. Данный недостаток был устранен в метаязыках SASS/SCSS, где реализована возможность использования переменных. Объявление переменной начинается со знака «$», например:

```
$yellow: #fce473;
```

Теперь в коде при необходимости можно просто обращаться к этой переменной:

```
.quote { border-left: 5px solid $yellow; }
```

Впоследствии данная строка будет преобразована в такой код CSS:

```
.quote { border-left: 5px solid #fce473; }
```

Как видно из примера, при компиляции на место переменной будет произведена обычная подстановка значения. Таким образом, можно в начале файла определить все характеристики стиля приложения, а затем при необходимости менять их “в одну строчку”. Например, так:

```
// Определяем значения цвета
$yellow: #fce473;
$green: #32cd32;

// Определяем основной цвет
$primary-color: $green;

// Используем
.quote { border-left: 5px solid $primary-color; }
.button { background: $primary-color; }
```

Теперь для изменения цветового стиля достаточно внести изменения в одну строку, описывающую основной цвет.

Переменной в SCSS можно задать любое значение, применяемое в CSS.

6.2.2.3 Управляющие конструкции языка

Управляющих конструкций в языке SCSS всего четыре, и начинаются они с символа *«@»*. Рассмотрим их подробнее.

Первая – это условный оператор, имеет следующий синтаксис:

```
@if (<условие>) {
  // команды выполняются, если <условие> истинно
}
@else {
  // команды выполняются, если <условие> ложно
}
```

Альтернатива *@else* не является обязательной и может быть пропущена, как и круглые скобки условия. В условии применимы стандартные операторы сравнения (==, !=, >, <, >=, <=).

Циклические операции задаются двумя командами: *@for* и *@while*. Формат параметрического цикла:

```
@for <переменная> from <старт> through <стоп> {
  // тело цикла
}
```

Параметры <старт> и <стоп> – это начальное и конечное значения переменной цикла. Они могут быть как положительными, так и отрицательными числами. Если начальное значение больше конечного, то цикл будет работать с уменьшением значения переменной.

Для цикла @while нужны два параметра: переменная и шаг. Пока величина шага удовлетворяет условию, происходит преобразование переменной. При каждой итерации шаг увеличивается. Начальное значение переменной задается заранее. Формат цикла:

```
@while <условие>{
  // тело цикла с изменением шага
}
```

Циклы удобно применять для CSS-спрайтов, в которых используется одна фоновая картинка, содержащая несколько изображений. Тогда задаются ограничения размера элемента, чтобы часть фона была скрыта, а затем при необходимости просто сдвигается фон и показывается нужная его часть.

Отдельно следует отметить команду *@each*, которая позволяет для каждого значения из определенного списка применить заданный набор команд. Синтаксис следующий:

```
@each <переменная> in <значение1>, < значение2>... {
  // тело цикла
}
```

Переменная последовательно принимает значения из списка после ключевого слова *in*. Чтобы воспользоваться ее текущим значением, можно использовать конструкцию *#{<переменная>}*.

В примере ниже для каждого из названий современных браузеров создается CSS класс и задаётся соответствующая фоновая картинка.

```
@each $browser in ie, chrome, firefox, opera {
  .#{$browser} {
    background: url(../img/#{$browser}.png) no-repeat;
  }
}
```

В результате будет создано 4ре класса с названиями, указанными в списке значений, и у каждого класса будет свое фоновое изображение с уникальным именем графического файла.

```
.ie {
  background: url(../img/ie.png) no-repeat;
}
.chrome {
  background: url(../img/chrome.png) no-repeat;
}
.firefox {
  background: url(../img/firefox.png) no-repeat;
}
.opera {
  background: url(../img/opera.png) no-repeat;
}
```

6.2.2.4 Вложения селекторов

Допустим, что один из селекторов в CSS необходимо применить к элементу, который располагается внутри другого элемента – в таком случае в коде стилей необходимо указать *иерархию* селекторов:

```
.title {}
.title strong {}
.title em {}
```

Все просто и наглядно, однако при увеличении глубины иерархии код также значительно увеличивается. Кроме того, обычно описывают не один контекстный элемент, а целую группу, для каждого из которых необходимо продублировать описание всей иерархии селекторов.

Язык SCSS написание подобного кода делает более наглядным и простым за счет *вложений*:

```
.title {
  strong {}
  em {}
}
```

Данный текст будет скомпилирован в указанный выше фрагмент CSS, однако читаемость и наглядность его значительно лучше. Глубина вложений не ограничена одним уровнем, так что возможно компактное построение любых цепочек селекторов.

Также вложения можно применять с псевдоклассами и псевдоэлементами, но в этом случае перед их именами необходим символ «&». Например, так:

```
// SCSS
.parent {
  &:hover {}
}
```

==>

```
// CSS
.parent:hover {}
```

6.2.2.5 Примеси в SCSS

Примеси (или миксины, от англ. mixin) – это аналог функций в языках программирования. Иными словами, это фрагмент кода, который можно вставлять несколько раз. При этом появляется ряд дополнительных преимуществ:

- в качестве значений можно передавать переменные, что придаст примесям гибкость;

- существуют библиотеки примесей (Bourbon [20], Compass и др.), в которых реализовано множество готовых типовых задач;

- примеси можно вынести в отдельный файл и загружать их по мере необходимости, используя команду *@import* – при этом код сокращается и упрощается.

Для создания примеси используется команда *@mixin*, после которой через пробел идёт произвольное имя и список параметров в скобках. Внутри в фигурных скобках описываются стилевые правила. А в том месте кода, где нужно «вызвать» примесь, добавляется строчка *@include* с её именем. Наличие списка параметров позволяет использовать примеси для одинаковых по смыслу участков кода с отличиями в нескольких лишь значениях. К тому же, можно задать параметрам значения по умолчанию, что убережет от лишних ошибок. Например, такой код и вызовы:

```
@mixin border-radius($radius: 1px) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
}
```

```
.box{
  @include border-radius();
}
```

```
.box{
  @include border-radius(3px);
}
```

скомпилируются следующим образом:

```
.box{
  -webkit-border-radius: 1px;
     -moz-border-radius: 1px;
      -ms-border-radius: 1px;
          border-radius: 1px; }
```

```
.box{
  -webkit-border-radius: 3px;
     -moz-border-radius: 3px;
      -ms-border-radius: 3px;
          border-radius: 3px; }
```

Как видно из примера, примеси также удобно использовать для обработки *вендорных* свойств (работающих только в одном браузере).

6.2.2.6 Расширения в SCSS

Некоторые селекторы используют одни и те же стилевые правила. Чтобы не повторять одинаковый код несколько раз, в CSS применяется группирование для удобства представления и сокращения кода. В этом случае селекторы перечисляются друг за другом через запятую, при этом стиль применяется к каждому селектору в группе. В SASS/SCSS для этих целей применяется команда *@extend* – она автоматически группирует те селекторы, в которых добавляется *@extend*. Вначале пишется селектор, содержащий одинаковые стилевые правила для всей группы, затем к любому другому селектору добавляем *@extend* и через пробел имя первого селектора. В примере ниже показано применение этой команды на практике:

```
.browser {
  min-height: 16px;
  padding-left: 20px;
  background-repeat: no-repeat;
}
.ie {
  @extend .browser;
  background-image: url(../img/browser_ie.png);
}
.cr {
  @extend .browser;
  background-image: url(../img/browser_cr.png);
}
.fx {
  @extend .browser;
  background-image: url(../img/browser_fx.png);
}
```

В результате будет получен следующий код:

```
.browser, .ie, .cr, .fx {
  min-height: 16px;
  padding-left: 20px;
  background-repeat: no-repeat;
}
.ie {
  background-image: url(../img/browser_ie.png);
}
.cr {
  background-image: url(../img/browser_cr.png);
}
.fx {
  background-image: url(../img/browser_fx.png);
}
```

Бывает, что необходимо вставить повторяющийся фрагмент кода в набор селекторов без создания нового класса. Тогда вместо точки перед обобщающим селектором ставится символ «%». В этом случае класс создаваться не будет, а такая операция называется ***заполнением***.

```
%browser {
.....
```

Следует отметить, что механизм работы расширений похож на примеси, но это не одно и то же – есть ряд различий:

- у расширений нет параметров, как у примесей;
- расширения группируют селекторы, а примеси – нет;
- применение примесей приводит к повторению кода при каждом вызове, расширение же записывает общие свойства только один раз, то есть более эффективно.

6.2.2.7 Другие особенности языка

В SASS/SCSS реализована поддержка математических операций над параметрами и свойствами, например, так:

```
.block {
.....
  width: $block_width - ( 5px * 2 ) - ( 1px * 2 );
}
```

Или так:

```
.some {
  $color: #010203;
  color: $color;
  border-color: $color - #010101;
  &:hover { color: #010203 * 2; }
}
```

Если к этому добавить широкие возможности, предоставляемые наличием в SASS/SCSS переменных, то в итоге получится очень мощный и гибкий инструмент формирования стилей Web-приложения.

6.3 Варианты заданий

В процессе выполнения лабораторной работы необходимо реализовать файл стилей заданного вариантом задания раздела сайта с использованием SASS/SCSS. Необходимо использовать возможности языка, указанные в таблице 6.1.

Таблица 6.1 – Варианты задания

№ вар.	Синтаксис	Раздел сайта	Возможности языка
1	Sass	администратора	Переменные, примеси, расширения, заполнения
2	Scss	администратора	Переменные, операторы, вложения, математика
3	Sass	администратора	Переменные, операторы, примеси, заполнения
4	Scss	администратора	Вложения, расширения, заполнения, математика
5	Sass	администратора	Операторы, примеси, расширения, математика
6	Scss	пользователя	Операторы, вложения, заполнения, математика
7	Sass	пользователя	Переменные, примеси, расширения, математика
8	Scss	пользователя	Переменные, операторы, вложения, заполнения
9	Sass	пользователя	Переменные, операторы, вложения, математика
10	Scss	пользователя	Вложения, примеси, расширения, заполнения
11	Scss	администратора	Переменные, примеси, расширения, заполнения
12	Sass	администратора	Переменные, операторы, вложения, математика
13	Scss	администратора	Переменные, операторы, примеси, заполнения
14	Sass	администратора	Вложения, расширения, заполнения, математика
15	Scss	администратора	Операторы, примеси, расширения, математика
16	Sass	пользователя	Операторы, вложения, заполнения, математика
17	Scss	пользователя	Переменные, примеси, расширения, математика
18	Sass	пользователя	Переменные, операторы, вложения, заполнения
19	Scss	пользователя	Переменные, операторы, вложения, математика
20	Sass	пользователя	Вложения, примеси, расширения, заполнения

6.4 Порядок выполнения работы

Используя результаты работ, выполненных ранее, модифицировать файл стилей заданного вариантом задания раздела сайта с использованием SASS/SCSS.

6.5 Содержание отчета

Цель работы, порядок выполнения работы, содержимое файлов стилей и их использования, выводы по результатам работы.

6.6 Контрольные вопросы

6.6.1. Что такое метаязык?
6.6.2. Что такое SASS/SCSS? В чем отличия?
6.6.3. Для чего используют SASS/SCSS?
6.6.4. Расскажите о переменных в SASS/SCSS. Приведите пример.
6.6.5. Какие операторы SASS/SCSS Вы знаете?
6.6.6. Что понимают под вложением селекторов?
6.6.7. Что такое примеси? Приведите пример.
6.6.8. Что такое расширение? Заполнение? В чем отличия?
6.6.9. В чем отличия между примесями и расширениями?

7 ЛАБОРАТОРНАЯ РАБОТА №7

«Исследование возможностей ускорения разработки клиентских приложений с использованием CoffeeScript»

7.1 Цель работы

Исследовать влияние препроцессоров на разработку клиентских приложений. Изучить основные возможности языка CoffeeScript. Приобрести практические навыки написания клиентских приложений с использованием CoffeeScript.

7.2 Краткие теоретические сведения

7.2.1 Язык программирования CoffeeScript

CoffeeScript – это язык программирования, транслируемый в JavaScript. По сути своей CoffeeScript является синтаксической надстройкой над языком JavaScript, применяемой для того, чтобы улучшить читаемость кода и уменьшить его размер. В качестве примера:

CoffeeScript	*JavaScript*
# Присвоение: *number = 42* *opposite = true* *# Условия:* *number = -42 if opposite* *# Функции:* *square = (x) -> x * x* *# Массивы:* *list = [1, 2, 3, 4, 5]* *# Объекты:* *math =* *root: Math.sqrt* *square: square* *cube: (x) -> x * square x* *# Многоточие:* *race = (winner, runners...) ->* *print winner, runners* *# Существование:* *alert "I knew it!" if elvis?* *# Итерации по массивам:* *cubes = (math.cube num for num in list)*	*var cubes, list, math, num, number, opposite, race, square,* *__slice = [].slice;* *number = 42;* *opposite = true;* *if (opposite) {* *number = -42;* *}* *square = function(x) {* *return x * x;* *};* *list = [1, 2, 3, 4, 5];* *math = {* *root: Math.sqrt,* *square: square,* *cube: function(x) {* *return x * square(x);* *}* *};* *race = function() {* *var runners, winner;* *winner = arguments[0], runners = 2 <= arguments.length* *? __slice.call(arguments, 1) : [];* *return print(winner, runners);* *};* *if (typeof elvis !== "undefined" && elvis !== null) {* *alert("I knew it!");* *}*

	cubes = (function() { var _i, _len, _results; _results = []; for (_i = 0, _len = list.length; _i < _len; _i++) { num = list[_i]; _results.push(math.cube(num)); } return _results; })();

JavaScript-код, получаемый трансляцией из CoffeeScript, полностью проходит проверку JavaScript Lint.

7.2.2 Элементы языка

7.2.2.1 Использование переменных и массивов

Объявление переменных в CoffeeScript несколько отличается от классического JavaScript:

hasBody = true

Это преобразуется в:

var hasBody = true;

Следует отметить, что нет необходимости ставить символ “;” в конце строки. Кроме того, компилятор CoffeeScript сам следит за необходимостью объявления и областью видимости переменных, так что у программиста нет доступа к ключевому слову *var*. Поэтому необходимо внимательно следить за переменными, поскольку совпадение имен локальной и глобальной переменных приведет к сокрытию значения последней.

Использование массивов и объектов очень похоже на JavaScript. Например:

CoffeeScript	*JavaScript*
song = ["do", "re", "mi", "fa", "so"] singers = {Jagger: "Rock", Elvis: "Roll"}	*var bitlist, kids, singers, song; song = ["do", "re", "mi", "fa", "so"]; singers = { Jagger: "Rock", Elvis: "Roll" };*

Вместо скобок можно использовать отступы, как в HAML и SASS. Кроме того, стало проще использовать зарезервированные слова в качестве ключей объекта – при компиляции они будут автоматически “обернуты” в кавычки.

7.2.2.2 Управляющие конструкции языка

При использовании условного оператора можно опускать круглые и фигурные скобки. Также оператор можно использовать «в одну строку», что при компиляции может быть распознано как тернарный оператор. Вложенные операторы обозначаются отступами:

CoffeeScript	*JavaScript*
mood = greatlyImproved if singing	*var date, mood;*

if happy and knowsIt * clapsHands()* * chaChaCha()* *else* * showIt()* *date = if friday then sue else jill*	*if (singing) {* * mood = greatlyImproved;* *}* *if (happy && knowsIt) {* * clapsHands();* * chaChaCha();* *} else {* * showIt();* *}* *date = friday ? sue : jill;*

Отдельно следует отметить символ “?”, означающий проверку параметра на *null* и *undefined* одновременно. Также существует оператор *unless*, по смыслу соответствующий инверсии условного оператора.

Итеративная работа с данными в CoffeeScript выполняется при помощи конструкции *for .. in*. В случае работы с полями объекта можно воспользоваться конструкцией *for .. on*. Если необходимо отфильтровать элементы по какому-либо признаку, используется ключевое слово *when*. Ключевое слово *by* задает шаг цикла. Например:

```
# Вывод массива
arr = [1, 2, 3]
for val in arr
  console.log(val);
```

```
# Выбрать всех кошек из массива питомцев (массив объектов)
for pet in pets when pet.type is "cat"
  myPet = pet
  console.log myPet
```

В CoffeeScript есть возможность пользоваться циклом *while*, а также его сокращенными формами *until* (*while not*) и *loop* (*while true*). Этот цикл может использоваться как выражение, возвращающее значение. Применяется реже цикла *for*. Например, так: *buy() while supply > demand*.

7.2.2.3 Функции в CoffeeScript

Функции определяются опциональным список параметров в скобках, символом «–>» и телом функции. В отличие от JavaScript, функции могут иметь значения по умолчанию для своих аргументов. Значение по умолчанию заменяется передачей ненулевого аргумента.

CoffeeScript	*JavaScript*
fill = (container, liquid = "coffee")-> "Filling the #{container} with #{liquid}..."	*var fill;* *fill = function(container, liquid) {* * if (liquid == null) {* * liquid = "coffee";* * }* * return "Filling the " + container + " with " + liquid + "...";* *};*

Следует обратить внимание на синтаксис подстановки переменных: *#{<Имя переменной>}*. Также ключевое слово *this* заменяется символом «@».

7.2.2.4 Классы в CoffeeScript

Язык CoffeeScript реализует механизм создания классов и иерархий из них. Так, класс может быть определен:

```
class Animal
  constructor: (@name, @breed) ->
    @.introduce = ->
      console.log "Hi, I am #{@.name}, one hell of a #{@.breed}"
```

В данном контексте знак «@» показывает, что значение будет записано в свойство класса. Реализация наследования выглядит следующим образом:

```
class Dog extends Animal

Dog.prototype.bark = ->
     console.log "#{@name} barks!"
```

Создание объекта и работа с ним:

```
newHusky = new Dog("Lassy", "Labrador Husky")
         newHusky.introduce()
newHusky.bark().
```

7.2.2.5 Другие особенности языка

Так как при использовании оператора «==» возможно появление множества ошибок, CoffeeScript преобразует оператор «==» в «===», генерируя, таким образом, более защищенный код. Существует также ряд операторов, имеющих разницу в написании на CoffeeScript и JavaScript; ниже приведен перечень основных из них и их аналоги.

CoffeeScript	**JavaScript**
`is, ==`	`===`
`isnt, !=`	`!==`
`not`	`!`
`and`	`&&`
`or`	`||`
`true, yes, on`	`true`
`false, no, off`	`false`
`@, this`	`this`
`of`	`in`
`in`	*нет эквивалента*

Ещё одной полезной особенностью CoffeeScript является возможность создания и использования диапазонов чисел:

```
someValues = [0..10]
countdown = (num for num in [10..1])
```

Более подробно о возможностях языка CoffeeScript можно узнать из документации, представленной на официальном сайте [21], и материалов на обучающих ресурсах [22].

7.3 Варианты заданий

Работа выполняется на основе проекта, полученого в ходе выполнения предыдущих лабораторных работ. Небходимо доработать проект в соответствии с указаними, данными в п.7.4 данной лабораторной работы.

7.4 Порядок выполнения работы

В процессе выполнения лабораторной работы необходимо переработать клиентский код страниц «Контакт», «Тест по дисциплине», «Фотоальбом» и «Мои интересы», реализовав его на языке CoffeeScript.

7.5 Содержание отчета

Цель работы, порядок выполнения работы, листинг кода, реализованного на CoffeeScript, выводы по результатам работы.

7.6 Контрольные вопросы

7.6.1. Что такое CoffeeScript?
7.6.2. Для чего используется CoffeeScript?
7.6.3. Дает ли CoffeeScript преимущества в скорости выполнения клиентских приложений?
7.6.4. Расскажите о переменных и массивах в языке CoffeeScript. Приведите пример.
7.6.5. Какие операторы применяются в CoffeeScript? Как и для чего?
7.6.6. Расскажите о функциях, приведите иллюстрирующий пример кода.
7.6.7. Расскажите о классах в CoffeeScript.
7.6.8. Возможно ли реализовать наследование при определении класса в CoffeeScript? Приведите пример.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Sublime Text [Электронный ресурс] // https://www.sublimetext.com: официальный сайт Sublime HQ Pty Ltd, 2007-2020. Дата обновления 2020. URL: https://www.sublimetext.com/3 (дата обращения: 01.06.2020).
2. Visual Studio Code [Электронный ресурс] // https://code.visualstudio.com: официальный сайт Visual Studio Code, Microsoft, 2018-2020. Дата обновления 2020. URL: https://code.visualstudio.com/docs (дата обращения: 01.06.2020).
3. Prepros [Электронный ресурс] // https://prepros.io: официальный сайт Prepros, 2020. Дата обновления 2020. URL: https://prepros.io/downloads (дата обращения: 01.06.2020).
4. Верстка блоками DIV [Электронный ресурс] // http://www.linedmk.com: Веб-компас сайтостроителей, 2008-2020. Дата обновления 2020. URL: https://www.linedmk.com/Verstka-blokami-DIV-art32.html (дата обращения: 01.06.2020).
5. Bootstrap [Электронный ресурс] // https://getbootstrap.com: Bootstrap team, 2011-2020. Дата обновления 2019. URL: https://getbootstrap.com (дата обращения: 01.06.2020).
6. Yet Another Multicolumn Layout – YAML 4 CSS Framework [Электронный ресурс] // http://www.yaml.de: Dirk Jesse, 2005-2013. Дата обновления 2019. URL: http://www.yaml.de (дата обращения: 01.06.2020).
7. Спецификация CSS [Электронный ресурс] // https://www.w3.org: W3C, 1994-2020. Дата обновления 2020. URL: https://www.w3.org/Style/CSS/Overview.ru.html (дата обращения: 01.06.2020).
8. Проверка на соответствие стандарту HTML 5 [Электронный ресурс] // https://www.w3.org: W3C, 1994-2020. Дата обновления 2020. URL: http://validator.w3.org (дата обращения: 01.06.2020).
9. Проверка на соответствие стандарту CSS 3 [Электронный ресурс] // https://www.w3.org: W3C, 1994-2020. Дата обновления 2020. URL: https://jigsaw.w3.org/css-validator (дата обращения: 01.06.2020).
10. Спецификация ECMAScript [Электронный ресурс] // http://www.ecma-international.org: Ecma International, 1986-2020. Дата обновления 2020. URL: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf (дата обращения: 01.06.2020).
11. Проверка качества JavaScript файлов [Электронный ресурс] // https://jshint.com: JSHint, 2010-2020. Дата обновления 2020. URL: https://jshint.com (дата обращения: 01.06.2020).

12. Рабочее описание стандарта DOM 4 [Электронный ресурс] // https://www.w3.org: WHATWG, 2018. Дата обновления 2020. URL: https://dom.spec.whatwg.org (дата обращения: 01.06.2020).
13. Библиотека JQuery [Электронный ресурс] // https://jquery.com: The jQuery Foundation, 2020. Дата обновления 2020. URL: https://jquery.com (дата обращения: 01.06.2020).
14. Документация HAML [Электронный ресурс] // http://haml.info (http://haml.ru): официальный сайт HAML. 2006-2017. Дата обновления 2017. URL: http://haml.ru/documentation (дата обращения: 01.06.2020).
15. Мурашев О. Используем Haml для генерации HTML [Электронный ресурс] // http://omurashov.ru/: личный блог Мурашева Олега. 2014. Дата обновления: 2017. URL: http://omurashov.ru/use-haml (дата обращения: 01.06.2020).
16. Мурашев О. HAML. Сборник рецептов [Электронный ресурс] // http://omurashov.ru/: личный блог Мурашева Олега. 2014. Дата обновления: 2017. URL: http://omurashov.ru/haml-cookbook (дата обращения: 01.06.2020).
17. Дорошев А. Jade - препроцессор HTML и шаблонизатор [Электронный ресурс] // http://www.reclamare.ua: «Веб-студия Reclamare». 2011-2020. Дата обновления: 2020. URL: http://www.reclamare.ua/blog/jade-preprocessor-html-i-shablonizator (дата обращения: 01.06.2020).
18. Catlin H. Документация: Основы SASS [Электронный ресурс] // http://sass-scss.ru (http://sass-lang.com): официальный сайт SASS. H. Catlin, N. Weizenbaum, C. Eppstein и др. 2006-2018. Дата обновления 2018. URL: http://sass-scss.ru/documentation (дата обращения: 01.06.2020).
19. Using LESS [Электронный ресурс] // http://lesscss.org: официальный сайт LESS. 2009-2016. Дата обновления 2016. URL: http://lesscss.org/usage (дата обращения: 01.06.2020).
20. Документация библиотеки Bourbon [Электронный ресурс] // http://bourbon.io: официальный сайт библиотеки Bourbon. thoughtbot, inc. 2011-2020. Дата обновления 2020. URL: http://bourbon.io/docs (дата обращения: 01.06.2020).
21. CoffeeScript [Электронный ресурс] // http://coffeescript.org (http://cidocs.ru/coffeescript): официальный сайт CoffeeScript. 2009-2020. Дата обновления 2020. URL: http://coffeescript.org (дата обращения: 01.06.2020).
22. Коржнев С. Знакомство с CoffeeScript [Электронный ресурс] // https://habrahabr.ru: ресурс для IT-специалистов. 2006-2020. Дата обновления 2016. URL: https://habrahabr.ru/post/179031 (дата обращения: 01.06.2020)

ПРИЛОЖЕНИЕ А
(справочное)
ХАРАКТЕРИСТИКИ ЭЛЕМЕНТОВ МОДЕЛИ DOM

Таблица А.1 – Методы объекта document

Метод/свойство	Параметры	Возвращаемое значение
getElementById()	Строка – значение атрибута id элемента	Узел DOM
getElementsByName()	Строка – значение атрибута name элемента	Семейство узлов DOM
getElementsByTagName()	Строка – наименование тега элемента	Семейство узлов DOM
getElementsByClassName()	Строка – класс тега элемента	Семейство узлов DOM
createElement()	Строка – наименование тега элемента или открывающий тег элемента с атрибутами	Узел DOM
createTextNode()	Строка – содержимое текстового узла	Узел DOM
createComment()	Строка – содержимое комментария	Узел DOM
createAttribute()	Строка – название атрибута	Объект-атрибут DOM

Таблица А.2 – Свойства узлов DOM

Свойство	Возвращаемое значение
firstChild	Первый дочерний узел
lastChild	Последний дочерний узел
previousSibling	Предыдущий ближайший узел одного уровня
nextSibling	Следующий ближайший узел одного уровня
childNodes	Семейство непосредственно порожденных узлов
nodeName	Наименование узла
nodeType	Тип узла
nodeValue	Значение узла (содержимое текстового узла, для остальных узлов null)
textContent	Текстовое содержимое узла
parentNode	Родительский узел

Таблица А.3 – Методы узлов DOM

Метод	Параметры	Возвращаемое значение
appendChild()	Новый дочерний узел	Добавленный узел
cloneNode()	Истина – клонировать и дочерние узлы, ложь – клонировать без дочерних	Новый (клонированный) узел
insertBefore()	Новый дочерний узел; необязательный параметр – дочерний узел, перед которым требуется вставить новый	Новый узел
replaceChild()	Новый дочерний узел и подлежащий замене дочерний узел	Замещенный узел
replaceNode()	Новый узел для замены существующего	Замещенный узел
removeChild()	Дочерний узел, подлежащий удалению	Удаленный узел
removeNode()	Истина – удалить и дочерние элементы, ложь – дочерние оставить	Удаленный узел
swapNode()	Узел, с которым исходный меняется местами	Узел, с которым осуществлялась замены
hasChildNodes()	Имеет ли узел дочерние – параметров нет	Истина, если да, ложь – в противном случае
getAttributeNode()	Строка – название атрибута элемента	Атрибут в виде объекта IHTMLDOMAttribute
setAttributeNode()	Узел-атрибут	Атрибут в виде объекта IHTMLDOMAttribute
getElementsByTagName()	Строка – наименование тега элемента	Семейство узлов DOM

Таблица А.4 – Методы объекта Date

Метод	Описание метода
getDate()	Возвращает день месяца из объекта в пределах от 1 до 31
getDay()	Возвращает день недели из объекта: 0 – вс, 1 – пн, 2 – вт, 3 – ср, 4 – чт, 5 – пт, 6 – сб.
getHours()	Возвращает время из объекта в пределах от 0 до 23
getMinutes()	Возвращает значение минут из объекта в пределах от 0 до 59
getMonth()	Возвращает значение месяца из объекта в пределах от 0 до 11
getSeconds()	Возвращает значение секунд из объекта в пределах от 0 до 59
getTime()	Возвращает количество миллисекунд, прошедшее с 00:00:00 1 января 1970 года.
getTimeZoneoffset()	Возвращает значение, соответствующее разности во времени (в минутах)
getYear()	Возвращает значение года из объекта
Date.parse(arg)	Возвращает количество миллисекунд, прошедшее с 00:00:00 1 января 1970 года. Arg – строковый аргумент.
setDate(day)	С помощью данного метода устанавливается день месяца в объекте от 1 до 31
setHours(hours)	С помощью данного метода устанавливается часы в объекте от 0 до 23
setMinutes(minutes)	С помощью данного метода устанавливаются минуты в объекте от 0 до 59
setMonth(month)	С помощью данного метода устанавливается месяц в объекте от 1 до 12
setSeconds(seconds)	С помощью данного метода устанавливаются секунды в объекте от 0 до 59
setTime(timestring)	С помощью данного метода устанавливается значение времени в объекте.
setYear(year)	С помощью данного метода устанавливается год в объекте year должно быть больше 1900.
toGMTString()	Преобразует дату в строковый объект в формате GMT.
toString()	Преобразует содержимое объекта Date в строковый объект.
toLocaleString()	Преобразует содержимое объекта Date в строку в соответствии с местным временем.
Date.UTC(year, month, day [,hours] [,mins] [,secs])	Возвращает количество миллисекунд в объекте Date, прошедших с 00:00:00 1 января 1970 года по среднему гринвичскому времени.

Заказ № _____ от «_____» _____2020г. Тираж _____экз.
Изд-во СевГУ