МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

Методические рекомендации к лабораторным работам для студентов направления подготовки 01.03.04 «Прикладная математика» дневной формы обучения

Могилев 2023

УДК 004.4
ББК 32.9
О75

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «24» января 2023 г., протокол № 6

Составитель ст. преподаватель Н. В. Выговская

Рецензент канд. техн. наук, доц. С. К. Крутолевич

Методические рекомендации предназначены для студентов направления подготовки 01.03.04 «Прикладная математика» дневной формы обучения.

Учебно-методическое издание

ОСНОВЫ WEB-ПРОГРАММИРОВАНИЯ

Ответственный за выпуск	А. И. Якимов
Корректор	Т. А. Рыжикова
Компьютерная верстка	Е. В. Ковалевская

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Цель методических рекомендаций к лабораторным работам по дисциплине «Основы WEB-программирования» заключается в закреплении студентами практических навыков создания и сопровождения программных систем современных ЭВМ с доступом через Интернет.

Дисциплина «Основы WEB-программирования» является неотъемлемой частью современных знаний и связана с такими дисциплинами, как «Базы данных» и «Объектно ориентированное программирование».

Выполнение заданий позволит студентам выработать практические навыки разработки сайтов и приложений для сети Интернет. Полученные при изучении дисциплины знания и навыки будут востребованы при практической разработке серверных и клиентских приложений, работающих в сети Интернет и станут востребованными при подготовке выпускной квалификационной работы.

В процессе выполнения лабораторной работы студенты ознакомятся с теоретическим материалом, методами решения задач, выполнят индивидуальное задание и оформлят отчет.

Отчет должен содержать название и цель лабораторной работы, структуру и листинг электронных документов, анализ полученных результатов и выводы. В отчете можно привести ответы на наиболее сложные вопросы, приведенные в конце каждой работы.

1 Лабораторная работа № 1. Форматирование текстов. Основные теги HTML

Цель работы

Изучить текстовое оформление страниц. Научиться вставлять изображения в Web-страницы.

Методические указания

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

Web-документ ограничивается тегами <HTML> и </HTML>, которые определяют начало и конец документа соответственно. В структуре HTML-документа выделяется заголовок (<HEAD> </HEAD>) и тело документа (<BODY> </BODY>). Заголовок может содержать заключенные в теги <TITLE> </TITLE> заглавие (или название) страницы, а также МЕТА-информацию.

Для создания HTML-документа нужно выполнить следующие действия.

1 Запустить приложение «Блокнот».

2 Набрать исходный текст документа, например:

<HTML>
<HEAD><TITLE>Заглавие документа</TITLE></HEAD>
<BODY>
Содержимое документа
</BODY>
</HTML>

3 Сохранить файл с расширением *.html* или *.htm*, например *first.html.*

Для просмотра созданного файла HTML-документа используют браузер (например, *Internet Explorer*, *Opera*, *Google Chrome или другие*).

Браузер – это специальная программа, которая позволяет искать информацию в интернете, просматривать сайты, скачивать файлы любого формата, загружать аудио и видеофайлы. Браузер отображает все элементы, представленные на странице HTML в разделе BODY.

Атрибуты тега *<BODY>* представлены в таблице 1.1.

Таблица 1.1 – Атрибуты тега <*BODY*>

Атрибут	Назначение
BACKGROUND	Указывает адрес изображения, которое следует использовать в качестве фона документа. Если вместе с этим атрибутом используется атрибут BGPROPERTIES со значением fixed, фоновое изображение не будет прокручиваться
BGCOLOR	Определяет цвет фона документа
TEXT	Определяет цвет текста в документе
LINK	Определяет цвет ссылок, которые не были посещены
ALINK	Определяет цвет активных ссылок (ссылка является активной в момент нажатия на нее)
VLINK	Определяет цвет ссылок на просмотренные документы
TOPMARGIN	Определяет ширину (в пикселях) верхнего поля документа
LEFTMARGIN	Определяет ширину (в пикселях) левого поля документа

При определении цвета для документа HTML могут использоваться название цветов или их обозначение в шестнадцатеричной системе кодирования RGB. Например, следующие строки идентичны:

<BODY BGCOLOR=”#FFFFFF”>
<BODY BGCOLOR=”WHITE”>

Для включения коммертариев в HTML-код используют последовательность символов <!--и-->. Например, <! – Это коммертарий-->

Различают теги логического и физического форматирования текста. Теги логического форматирования определяют, какую смысловую нагрузку несет выделенный фрагмент, т. е. чем он является в документе (заголовком, абзацем, цитатой и т. д.). Теги физического форматирования определяют в первую очередь, как будет выглядеть выделенный фрагмент. Теги логического форматирования представлены в таблице 1.2.

Таблица 1.2 – Теги логического форматирования

Тег	Назначение
<CITE>	Используется для выделения цитаты (обычно курсив)
<CODE>	Выделяет программный код
<EM>	Используется для выделения фрагмента текста, имеющего большое значение (обычно курсив)
<KBD>	Выделяет текст, который предлагается набрать на клавиатуре (обычно моноширинный шрифт)
<SAMP>	Используется для выделения знаков, на которые необходимо акцентировать внимание (обычно моноширинный шрифт)
<STRONG>	Используется для выделения очень важного фрагмента текста (обычно полужирный)
<VAR>	Используется для выделения имени переменной (обычно курсив)
<DFN>	Используется для выделения определения (обычно курсив)
<BLOCKQUOTE>	Выделяет цитату (обычно используется отступ от левого поля)
<P>	Выделяет отдельный абзац в тексте
<H1>…<H6>	Выделяет заголовки шесть уровней (*H1* – самый важный)

Применение тегов логического форматирования демонстрируется в приведенном ниже примере, результат которого приведен на рисунках 1.1 и 1.2.

Теги физического форматирования представлены в таблице 1.3.

Для добавления на страницу горизонтальной линии применяют тег <HR>.

Использование тегов физического форматирования демонстрируется в приведенном ниже примере, результат – на рисунке 1.3.

Для создания ссылки необходимо сообщить браузеру, что является ссылкой, а также указать адрес документа, на который следует сделать ссылку.

```
1  <HTML>
2  <HEAD>
3   <TITLE>Логическое форматирование</TITLE></HEAD>
4  <BODY>
5  <H2>
6   Теги логического форматирования</H2>
7  <P>
8   <CITE>Цитата</CITE></P>
9   <P><blockquote>Еще одна цитата</blockquote></P>
10 <P><CODE>
11  Программный код</CODE></P>
12 <P><STRONG>
13  Важный текст</STRONG></P>
14 <P><DFN>
15  Определение</DFN></P>
16  </BODY>
17  </HTML>
```

Рисунок 1.1 – Теги логического форматирования

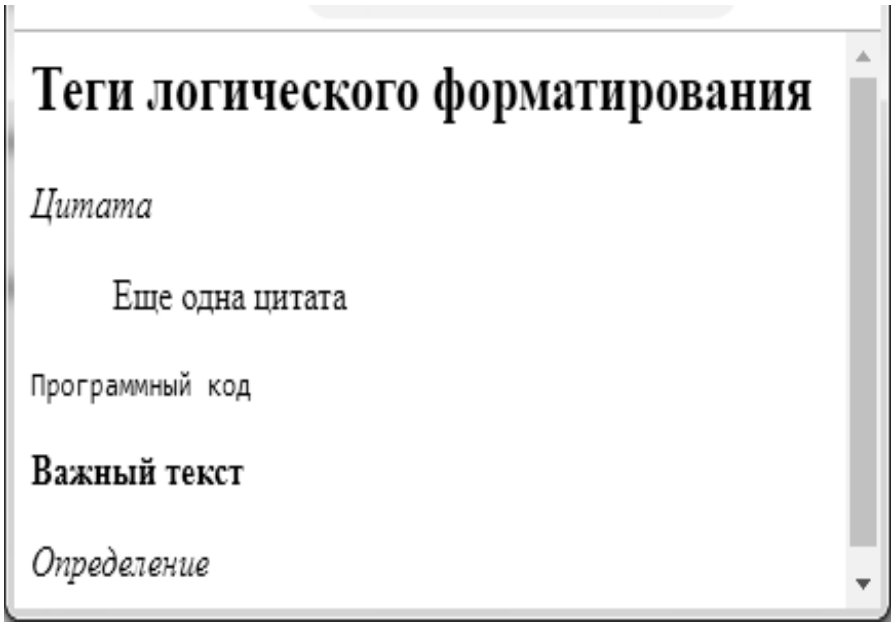

Рисунок 1.2 – Отображение тегов логического форматирования

Таблица 1.3 – Теги физического форматирования

Тег	Назначение
<DASEFONT>	Определяет цвет, размер и тип основного шрифта документа (соответственно атрибуты *color*, *size*, *face*). Данный тег не является контейнером
<FONT>	Позволяет изменять цвет, размер и тип шрифта
<I>	Выделяет текст курсивом
<B>	Выделяет текст жирным шрифтом
<U>	Подчеркивает текст
<S>,<STRIKE>	Зачеркивает тескт
<BIG>	Отображает текст увеличенным шрифтом (относительно текущего)
<SMALL>	Отображает текст уменьшенным шрифтом (относительно текущего)
<SUP>	Отображает текст со сдигом вверх (верхний индекс)
<SUB>	Отображает текст со сдигом вниз (нижний индекс)
<TT>	Отображает текст моноширинным шрифтом
 	Переход на новую строку. Используется без парного закрывающего тега
<CENTER>	Горизонтальное выравнивание текста по центру
<PRE>	Предварительное форматирование: вывод текста в том виде, в котором он представлен в исходном документе

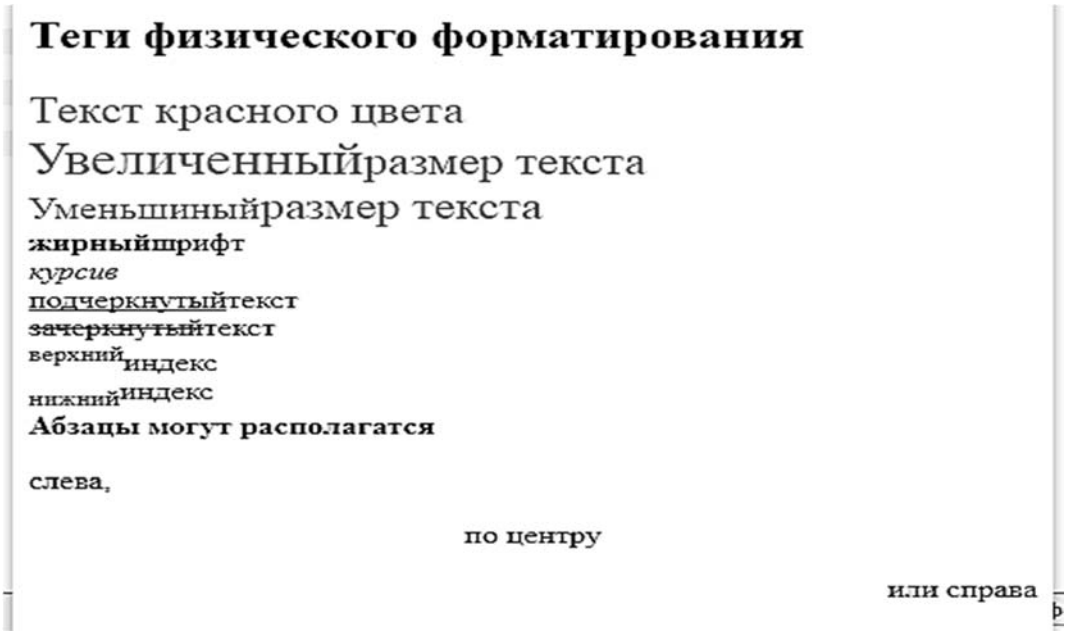

Рисунок 1.3 – Использование тегов физического форматирования

Ссылки состоят из двух частей:

1) элемент привязки (или якорь – anchor) – место в документе, отмеченное как сслылка. Существуют два типа элементов привязки: текстовый и графический;

2) ссылка на URL – сообщает браузеру, какой документ нужно загрузить при щелчке на ссылку.

Для организации ссылок в HTML используется тег <*A*>. Структура ссылки показана на рисунке 1.4.

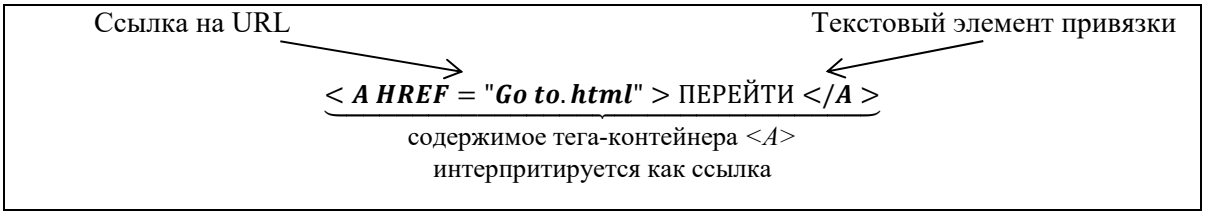

Рисунок 1.4 – Структура ссылки

Могут использоваться относительные и абсолютные ссылки на URL-файл.

Относительной называется ссылка на файл, находящийся на том же компьютере. Это означает, что URL указывается относительно компьютера и каталога, из которого браузер первоначально загружает Web-страницу.

Например:

```
<A href=”next.html”>
        Перейти на страницу next.html</A>
```

Абсолютной называется ссылка, в которой указан полный путь к файлу (компьютер, каталог, имя файла).

Например:

```
<A href=”http://www.yandex.ru/index.html”>
            Перейти на портал Yandex</A>
```

Можно создать ссылку не только на другой документ, но и конкретный раздел текущего. Такие ссылки называют внутренними ссылками или закладками. Пример внутренней ссылки:

```
<HTML>
<HEAD><TITLE>Внутренние ссылки</TITLE></HEAD>
<BODY>
 <H2> Создание внутренних ссылок</H2>
<P><a href=#five>Переход к разделу 3</a></P>
<BR><BR>
<P>Раздел1
...</P>
<P>Раздел 2
...</P>
<P><A name=five>Раздел 3</A>
...</P>
</BODY>
</HTML>
```

Таким образом, при помощи тега *<A>* создается элемент привязки, т. е. определяется место в документе, к которому нужно перейти. С помощью атрибута *NAME* элементу привязки присваивается имя *five*. Элемент привязки, созданный таким образом, в тексте выделятся не будет. Для ссылки на созданную закладку атрибуту *HREF* тега *<A>* присваивается значение следующего вида: путь_к_документу#имя_закладки. Если закладка находится в том же документе, что и ссылка, путь к документу можно не указывать.

Для внедрения графики на HTML-страницу используется тег *<IMG>*, который может включать в себя атрибуты, приведенные в таблице 1.4.

Пример добавления изображения 1.jpg на HTML-страницу показан на рисунке 1.5.

Таблица 1.4 – Атрибуты тега *<IMG>*

Атрибут	Назначение
SRC	Обязательный атрибут, указывает адрес файла с изображением
HEIGHT	Определяет ширину изображения
WIDTH	Определяет высоту изображения
HSPACE	Определяет отступ изображения по горизонтали от других объектов документа
VSPACE	Определяет отступ по вертикали от других объектов документа
ALIGN	Указывает способ выравнивания изображения в документе (LEFT, RIGHT)
NAME	Определяет имя изображения, уникальное для данного документа
ALT	Определяет альтернативный текст
BORDER	Определяет ширину рамки вокруг изображения

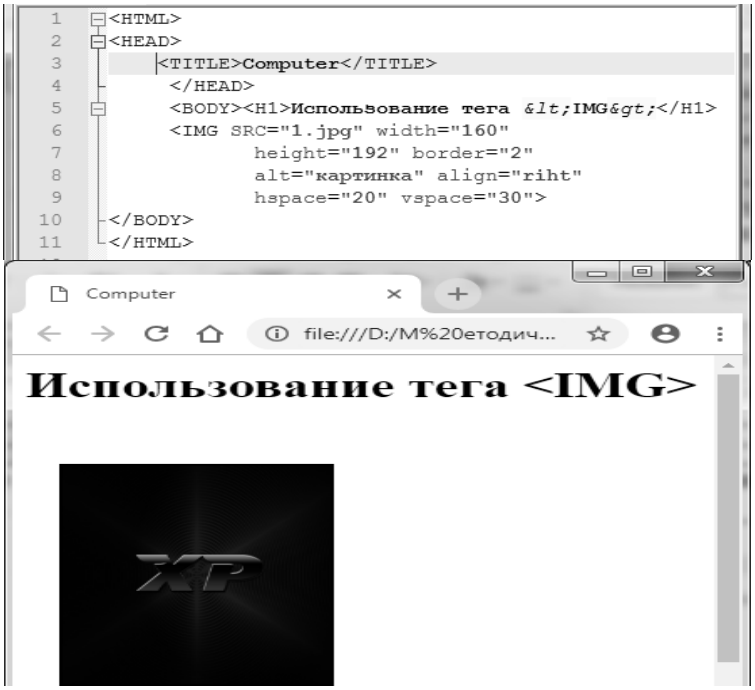

Рисунок 1.5 – Вставка изображения на HTML-страницу

Порядок выполнения работы

1 Создайте документ *Lab1_1.html*, содержащий перечь рабочих дней недели.

2 Установите название документа, цвет фона и текста.

3 Создайте документ *Lab1_2.html,* содержащий расписание занятий на неделю.

4 Организуйте систему ссылок таким образом, чтобы при выборе определенного дня недели в документе *Lab1_1.html* осуществлялся переход к расписанию занятий данного дня в документе *Lab1_2.html.*

5 Добавьте в начало документа *Lab1_2.html* ссылки на расписание определенного дня недели.

6 Установите цвет ссылок для документов *Lab1_1.html* и *Lab1_2.html.*

7 Вставьте изображение на страницу *Lab1_1.html.*

8 Установите размер изображения, ширину рамки, выравнивание, отступ по горизонтали и вертикали, а также альтернативный текст.

9 Используйте добавленное изображение в качестве графического элемента привязки ссылки на документ *Lab1_2.html.*

Контрольные вопросы

1 Какую структуру имеет HTML-документ?
2 Перечислите известные вам атрибуты тега <*BODY*>.
3 Укажите отличие использование тегов <*P*> и <*BR*>.
4 Какие теги форматирования не являются контейнерами?
5 В чем различие абсолютных и относительных ссылок?
6 Как можно создать ссылку с графическим элементом привязки?
7 Какие атрибуты тега <*IMG*> вы знаете?

2 Лабораторная работа № 2. Разработка списков на HTML

Цель работы

Формирование практических умений работы со списками HTML.

Методические указания

Существуют следующие виды списков:
– упорядоченные (пронумерованные);
– неупорядоченные (непронумерованные);
– список определений.
Для создания списков используются теги, приведенные в таблице 2.1.

Таблица 2.1 – Теги для создания списков

Тег	Назначение
<UL>	Создает неупорядоченные списки
<OL>	Создает упорядоченные списки
<LI>	Определяет пункт меню внутри элементов *OL* или *UL*
<MENU>,<DIR>	Создает неупорядоченный список, подобный *UL*
<DL>	Ограничивает список определений
<DT>	Создает термин в списке определений внутри элемента *DL*
<DD>	Создает определение термина внутри элемента *DL*

Атрибуты тега <*DL*> приведены в таблице 2.2.

Таблица 2.2 – Атрибуты тега <*DL*>.

Атрибут	Назначение
START	Определяет первое число, с которого начинается нумерация пунктов
TYPE	Определяет стиль нумерации пунктов

Значение атрибута *TYPE* для упорядоченного списка могут быть следующими:

- *1* – арабкие цифры 1,2,3…;
- *a* – строчные латинские буквы *a*, *b*, *c*…;
- *A* – прописные латинские буквы *A*, *B*, *C*...;
- i – римские цифры i, ii, iii,…;
- I – римские цифры I, II, III,…

В теге <*UL*> также можно использовать атрибут *TYPE* (со значениями *DISC*, *SQUARE* или *CIRCLE*).

Для изменения порядка нумерации элементов списка в теге <*LI*> применяется атрибут *VALUE.*

В *HTML* многоуровневых списков нет. То есть для них нет отдельной конструкции. Но она и не нужна. Для разметки многоуровневых списков достаточно того, что уже известно. В тех местах, где нужно вставить подпункты списка, используются вложенные списки аналогично любым вложенным дескрипторам.

Пример создания списков различных видов приведен на рисунках 2.1 и 2.2.

```
spiski.html
 1  <HTML>
 2  <HEAD><TITLE>создание списков</TITLE></HEAD>
 3  <BODY>
 4        <B>Ненумерованный список</B>
 5  <UL type="square">
 6  <LI>Элемент 1
 7  <LI>Элемент 2
 8  <LI>Элемент 3
 9  <LI>Элемент 4
10  </UL>
11        <B>Нумерованный список</B>
12  <OL type="I" start="3">
13  <LI>Элемент 1
14  <LI>Элемент 2
15  <LI value="1">Элемент 3
16  <LI>Элемент 4
17  </OL>
18
19        <B>Список определений</B>
20  <DL>
21         <DT>Термин 1
22         <DD>Определение 1
23             <DT>Термин 2
24             <DD>Определение 2
25  </DL>
26  </BODY>
27  <HTML>
```

Рисунок 2.1 – HTML-код использования списков

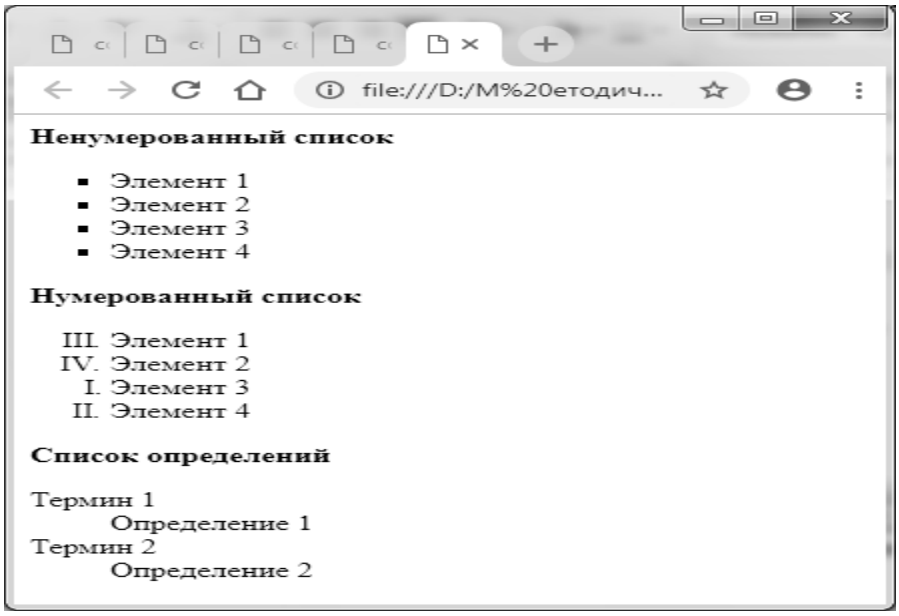

Рисунок 2.2 – Использования списков на HTML-странице

Порядок выполнения работы

Создайте документ *Lab2.html*, содержащий контрольные вопросы по данной лабораторной работе и ответы на них в виде всех типов списков.

Контрольные вопросы

1 Какие виды списков вы знаете?
2 Какие теги используются для создания упорядоченных списков?
3 Какие теги используются для создания неупорядоченных списков?
4 Как определить стиль нумерации списков?
5 Как изменить порядок нумерации списков?
6 Можно ли создать вложенные списки?
7 Для чего используются списки определений?
8 Какие теги используются для создания списков определений?

3 Лабораторная работа № 3. Разработка таблиц HTML

Цель работы

Формирование практических умений работы с таблицами HTML.

Методические указания

Таблица состоит из строк и столбцов ячеек, которые могут содержать текст и рисунки. Обычно таблицы используются для упорядочения и представления данных, однако возможности таблиц этим не ограничиваются. С помощью таблиц удобно верстать макеты страниц, расположив нужным образом фрагменты текста и изображений.

Для добавления таблицы на Web-страницу используется тег-контейнер *TABLE*. Таблица должна содержать хотя бы одну строку и колонку. Вид таблицы в HTML показан на рисунке 3.1.

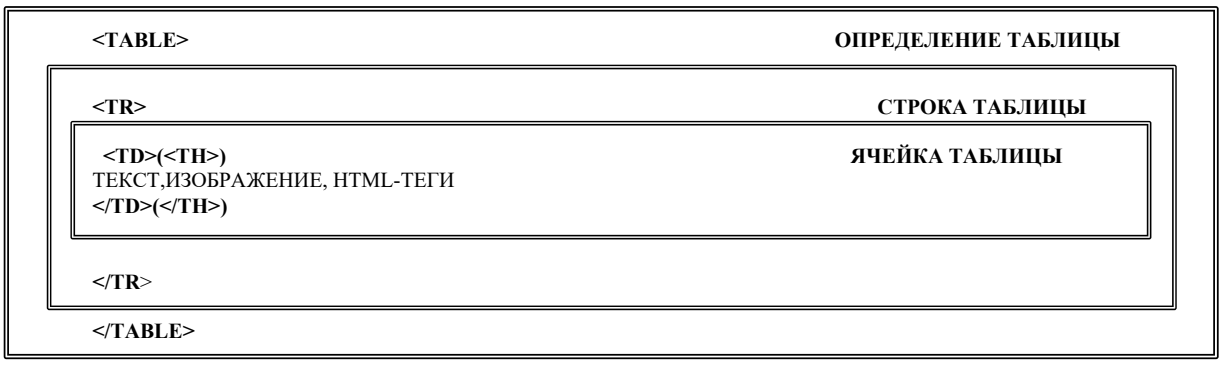

Рисунок 3.1 – Таблица в HTML

Атрибуты тега <*TABLE*> приведены в таблице 3.1.

Пример создания простейшей таблицы, состоящей из двух строк и двух столбцов приведен на рисунке 3.2.

Атрибуты тега <*TR*> приведены в таблице 3.2.

Тег <*TH*> используется для определения заголовков в первой строке или в первом стобце таблицы.

Атрибуты тегов <*TD*> и <*TH*> приведены в таблице 3.3.

Пример использования атрибутов *COLSPAN* и *ROWSPAN* приведен на рисунке 3.3.

Таблица 3.1 – Арибуты тега <*TABLE*>

Атрибут	Назначение
ALIGN	Определяет способ горизонтального выравнивания таблицы (LEFT, RIGHT, CENTER)
VALING	Определяет способ вертикального выравнивания таблицы (LEFT, RIGHT, CENTER)
BORDER	Определяет толщину рамки таблицы
CELLPADDING	Определяет расстояние между рамкой ячейки таблицы и ее содержимым
CELLSPACING	Определяет расстояние между границами соседних ячеек
WIDTH	Определяет ширину таблицы
HEIGHT	Определяет высоту таблицы
BGCOLOR	Определяет цвет фона ячеек таблицы
BACKGROUND	Заполняет фон таблицы изображением

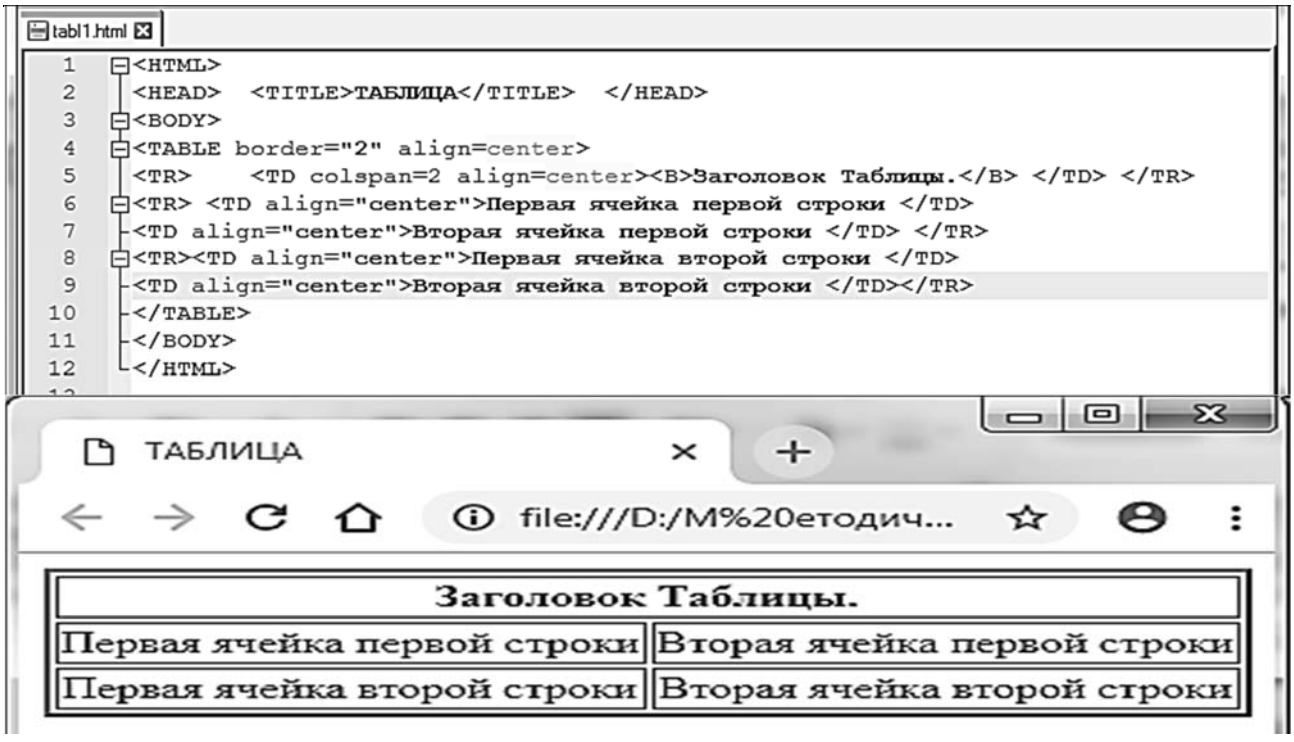

Рисунок 3.2 – Пример создания простейшей таблицы

Таблица 3.2 – Атрибуты тега <TR>

Атрибут	Назначение
ALIGN	Определяет способ горизонтального выравнивания содержимого всех ячеек данного ряда (LEFT, RIGHT, CENTER, JUSTIFY)
VALIGN	Определяет способ вертикального выравнивания содержимого всех ячеек данного ряда
BGCOLOR	Определяет цвет фона для всех ячеек данного ряда

Таблица 3.3 – Атрибуты тегов <TD> и <TH>

Атрибут	Назначение
ALIGN	Определяет способ горизонтального выравнивания содержимого ячейки
VALING	Определяет способ вертикального выравнивания содержимого ячейки
WIDTH	Определяет ширину ячейки
HEIGHT	Определяет высоту ячейки
COLSPAN	Определяет количество столбцов, объединенных данной ячейки
ROWSPAN	Определяет количество рядов, объединенных данной ячейкой
NOWRAP	Определяет автоматический перенос слов в пределах текущей ячейки
BGCOLOR	Определяет цвет фона ячейки
BACKGROUND	Заполняет фон ячейки изображением

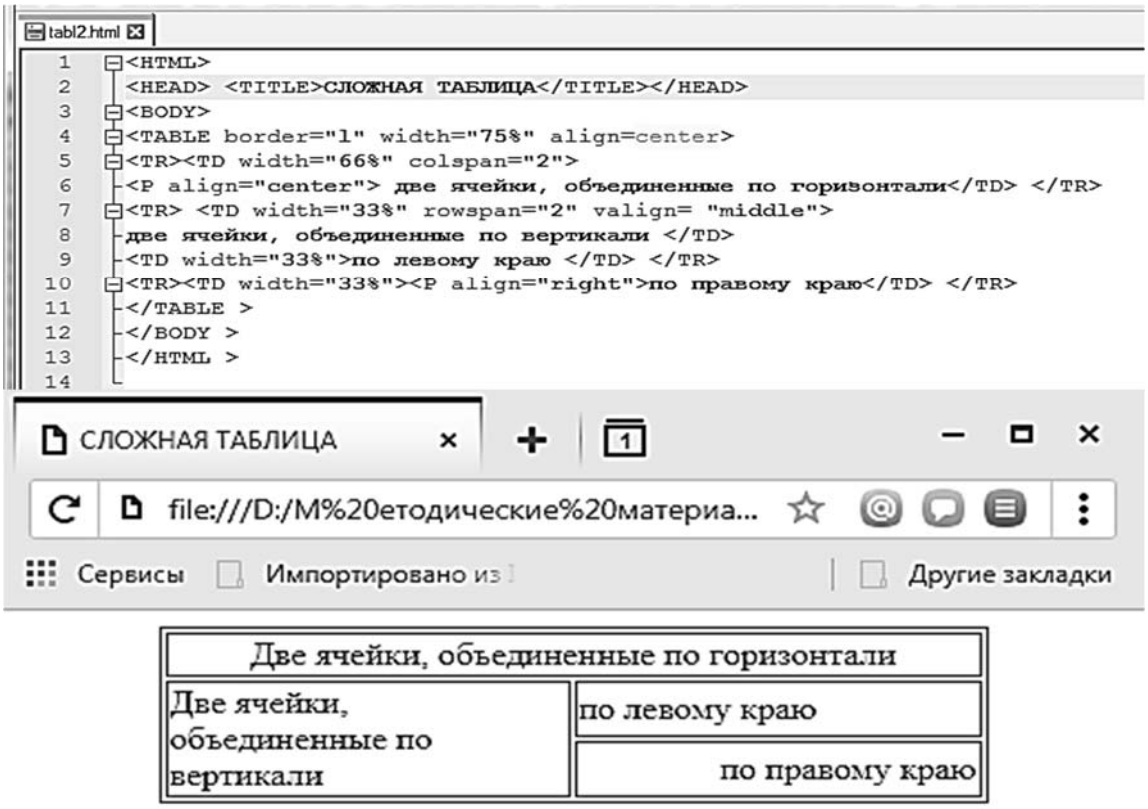

Рисунок 3.3 – Пример использования атрибутов *COLSPAN* и *ROWSPAN*

Порядок выполнения работы

1 Создайте документ *Lab3.html*, содержащий таблицу, соответствующую заданию, выданному преподавателем.

2 Установите название документа, цвет фона и текста.

3 Определите выравнивание таблицы по горизонтали и вертикали.

4 Определите ширину и высоту таблицы.

5 Определите цвет фона ячеек таблицы.

Контрольные вопросы

1 Какие теги создания таблицы вы знаете?
2 Чем отличаются теги <*TD*>и <*TH*>?
3 Какие атрибуты могут использоваться для тегов <*TR*> и <*TD*>?
4 Как можно определить выравнивание таблицы?

4 Лабораторная работа № 4. Разработка и форматирование форм на HTML

Цель работы

Формирование практических навыков создания форм на HTML-страницах.

Методические указания

Формы используются для получения оаределенной информации от пользователя с целью ее последующей обработки.

Для создания форм используются парные теги <*FORM*><*/FORM*>.

Тег <*FORM*> может содержать атрибуты, приведенные в таблице 4.1.

Для определения элементов форм могут использоваться теги, приведённые в таблице 4.2.

Таблица 4.1 – Атрибуты тега <*FORM*>

Атрибут	Назначение
NAME	Имя формы
ACTION	URL, по которому следует передать введенную информацию для последующей обработки
METHOD	Метод передачи данных из формы

Таблица 4.2 – Теги для определения элементов форм

Тег	Назначение
<TEXTAREA></TEXTAREA>	Многосточное текствое поле ввода
<INPUT>	Поля ввода
<SELECT><SELECT>	Меню – список

Тег <*TEXTAREA*> имеет атрибуты, приведенные в таблице 4.3.

Таблица 4.3 – Атрибуты тега <*TEXTAREA*>

Атрибут	Назначение
NAME	Имя поля ввода
ROWS	Число строк в поле ввода
COLS	Ширина поля ввода в символах

Тег <*INPUT*> имеет атрибуты, приведенные в таблице 4.4.

Таблица 4.4 – Атрибуты тега <*INPUT*>

Атрибут	Назначение
CHECKED	Элемент форм CHECKBOX или RADIO будет отменен
SIZE	Размер поля ввода в символах
MAXLENGTH	Количество символов, которое можно ввести в поле ввода
NAME	Имя поля ввода
SRC	Указывает путь к изображению (используется вместе со значением IMAGE атрибута TYPE)
VALUE	Определяет текст по умолчанию для поля ввода текста или пароля. Для флажка или переключателя указывает значение, возвращаемое серверу в случае выбора флажка или переключателя
TYPE	Определяет тип поля ввода; по умолчанию создается однострочное текстовое поле ввода

Возможные значения тега *TYPE* приведены в таблице 4.5.

Таблица 4.5 – Значения тега *TYPE*

Атрибут	Назначение
CHECBOX	Флажок; может принимать значение *ON* (отменен) или *OFF* (не отменен)
HIDDEN	Скрытое поле
IMAGE	Изображение
TEXT	Однострочное поле ввода
PASSWORD	Модифицированное текстовое поле (при вводе текста вместо символов отображаются звездочки)
RADIO	Переключатель (используется для выбора одного варианта из нескольких)*
RESET	Кнопка, при нажатии на которую поля форм принимают значение по умолчанию
SUBMIT	Кнопка отправки данных
BUTTON	Кнопка, определенная пользователем (то есть конкретного действия за данной кнопкой не закреплено, оно задается пользователем)
FILE	Поле ввода и кнопка «Обзор» для поиска файла на диске
Примечание – * – если переключателям даны одинаковые имена (в атрибуте *NAME*) или они объединяются в группы, из группы переключателей можно выбрать только один	

Тег <*SELECT*> имеет атрибуты, приведенные в таблице 4.6.

Таблица 4.6 – Атрибуты тега <*SELECT*>

Атрибут	Назначение
MULTIPLE	Дает возможность выбора нескольких пунктов меню при удержании клавиши Ctrl
NAME	Определяет имя меню
SIZE	Определяет количество видимых пунктов меню

Между тегами **<*SELECT*></*SELECT*>** находятся значения, которые может выбрать пользователь. Они перечисляются с помощью тега **<*OPTION*>.**

Тег **<*OPTION*>** имеет атрибуты, приведенные в таблице 4.7.

Таблица 4.7 – Атрибуты тега <*OPTION*>

Атрибут	Назначение
VALUE	Значение, присваиваемое выбранному элементу списка и отсылаемое серверу
SELECTED	Означает, что данный элемент списка будет выбран

Примеры создания формы в документах *Forma1* и *Forma2* показан на рисунках 4.1 и 4.2.

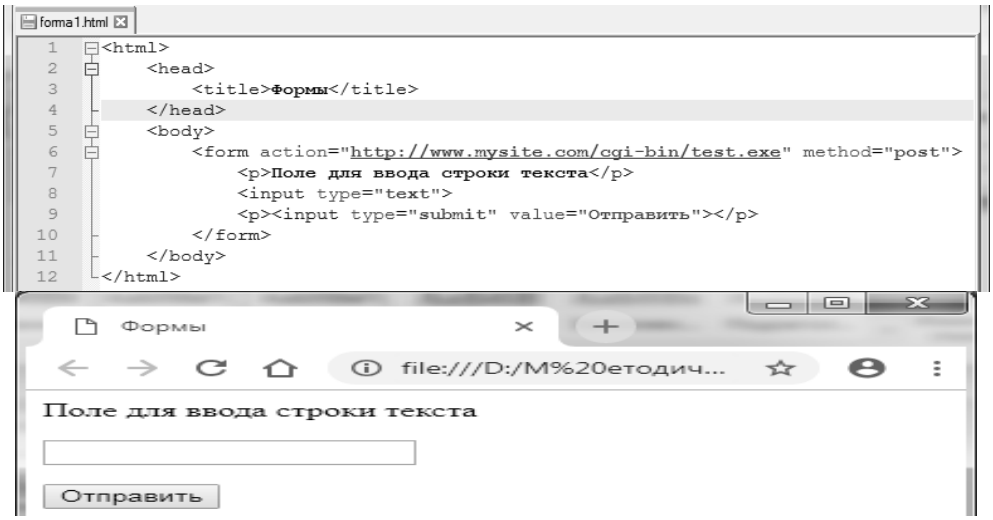

Рисунок 4.1 – Документ *Forma1*

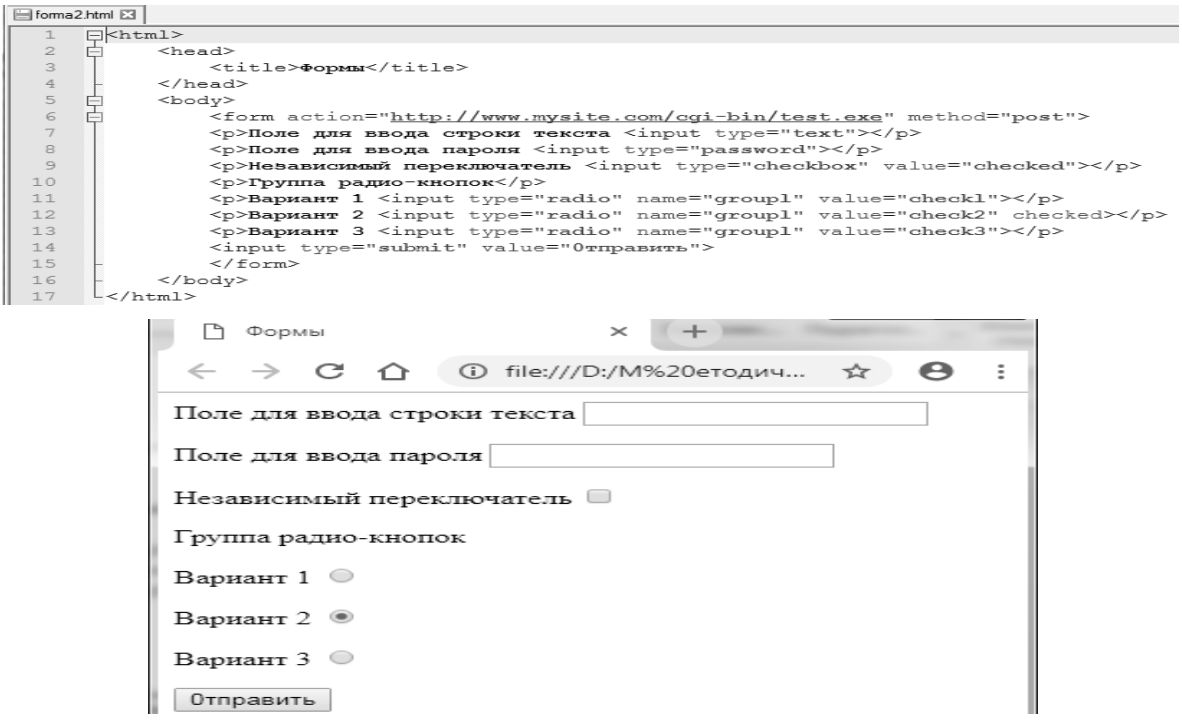

Рисунок 4.2 – Документ *Forma2*

При отправке данных создаются пары *параметр=значение*, где параметру соотвествует имя элемента (атрибут *NAME*), а значению – данные, введенные пользователем (в частности атрибут *VALUE*).

Порядок выполнения работы

1 Создайте документ *Lab4.html*.
2 Установите название документа, цвет фона и текста.
3 Создайте регистрационную форму, содержащую:
– текстовое поле для ввода имени;
– текстовое поле для ввода фамилии;
– поле для ввода пароля;
– меню-список для выбора страны проживания;
– группу выпадающих списков для указания даты рождения;
– переключатель для указания пола;
– переключатели-флажки для указания интересов;
– кнопку очистки формы;
– кнопку отправки данных форм;
– многострочное текстовое поле для ввода примечаний.
4 Установите надписи на кнопках.
5 В поле для примечаний укажите текст по умолчанию.
6 Для полей ввода имени и фамилии укажите размер поля и максимальное количество вводимых символов.
7 Добавьте на страницу графическое изображение, используя значение *IMAGE* атрибута *TYPE*.

Контрольные вопросы

1 Для чего используются формы?
2 Какой тег используется для создания формы?
3 Перечислите атрибуты тега <*FORM*>.
4 Перечислите известные теги определения отдельных элементов формы.
5 Какие атрибуты тега <*TEXTAREA*> вы знаете?
6 Перечислите атрибуты тега <*INPUT*>.
7 Перечислите возможные значения атрибута *TYPE* тега <*INPUT*>.
8 Как можно создать выпадающий список значений?
9 Для чего используется атрибут *MULTIPLE*?
10 При помощи какого тега определяется отдельный элемент меню списка?

5 Лабораторная работа № 5. Селекторы CSS и позиционирование

Цель работы

Ознакомление с языками HTML и CSS, а также получение практических навыков применения каскадных таблиц стилей для формирования отображения страниц HTML.

Методические указания

CSS (каскадные таблицы стилей) управляет внешним видом документа. Использование CSS позволяет отделить содержание документа от его оформления, т. е. сначала определяется, как будет выглядеть тот или иной элемент документа (например, заголовок, абзац и т. д.), а затем вводится его содержание.

Существуют четыре способа применения таблиц стилей к документу:

1) связывание;
2) встраивание;
3) оперативное определение;
4) импорт.

Связывание – это установка связи HTML-документа с таблицей стилей, хранящейся в отдельном файле с расширением css. Для связывания используется тег <*LINK*>.

Например:

```
<LINK REL=STYLESHEE
HREF="http://www.meserver.com/mesheet.css"
        TYPE="text/css">
```

Таблицу стилей можно определять не только в отдельном файле, но и в документе, в котором она будет применяться. Включение таблицы стилей в документ называется встраиванием (используется тег-контейнер *<STYLE>)*. Описание стилей размешается между тегами *<HTML> и <BODY>*.

Оперативное определение стиля используется, если нужно определить свойства для конкретного фрагмента HTML-документа, отличные от установленных по умолчанию для всего документа. Новые свойства указываются в атрибуте тега *STYLE*, для которого определяются параметры оформления.

Например:

```
<H1 STYLE="color:blue">
```

Для *импорта* таблиц стиля в HTML-файл используется слово *@import.* В данном случае импортируется только содержимое текстового файла, поэтому для того, чтобы этот текст интерпретировался как таблица стилей, *@import* нужно поместить в контейнер *<STYLE>*.

Например:

```
<STYLE TYPE=”text/css”>
@import url(http://www.meserver.com/style.css);
</STYLE>
```

Каждое определение стилей называется правилом (rule).
Формат правила SCC следующие:

селектор{свойство1:значение1;свойство2:значение2;…}
Например:

```
H1{color:blue}
```

В документе, для которого определяется данное правило, все заголовки *H*1 будут выделяться синим цветом.

Если заменить это правило на *H*1, *H*2, *H*3 *{color*:*blue}*, синим цветом будут выделяться заголовки первого, второго и третьего уровней.

Класс определяет разновидность стиля, к которому можно обращаться в определенном теге, используя атрибут *CLASS*. Например, можно определить три разновидности стиля *H*1 и затем использовать каждый из них в соответствующем контексте:

```
H1.blue{color:blue}
H1.red{color:red}
H2.black{color:black}
```

При добавлении тега <*H1*> в HTML-документ определяется атрибут *CLASS*, чтобы указать, какой именно стиль будет использоваться:

```
<H1 CLASS=red>Красный заголовок</H1>
```

Можно создавать класс, не связанный с определенным тегом. Например, если задать стилевое правило следующим образом:

```
.bold_and_italic{form-style:italic;font-weight:bold}
```

и присвоить атрибуту *CLASS* некоторого тега значение *bold_and_italic*, содержимое данного тега будет отображаться жирным шрифтом с курсивным начертанием. Использование псевдоклассов позволяет указать внешний вид HTML-элемента в определенный момент времени. Синтаксис псевдоклассов следующий: *Селектор:псевдокласс{свойство:значение}*.

В SCC определены псевдоклассы для гиперссылок.
Например:

```
/*непосещенная гиперссылка*/
A:link{color:blue}
/*активная гиперссылка*/
A:active{color:red}
/*посещенная гиперссылка*/
A:visited{color:yellow}
/*свойства гиперссылки при наведении курсора*/
A:hover{color:green}
```

Таким образом, непосещенная гиперссылка будет выделена синим цветом, активная – красным, посещенная – желтым, а при наведении курсора мыши цвет ссылки будет изменяться на зеленый.

Пример применения CSS в документе *Style1.html* приведен на рисунке 5.1.

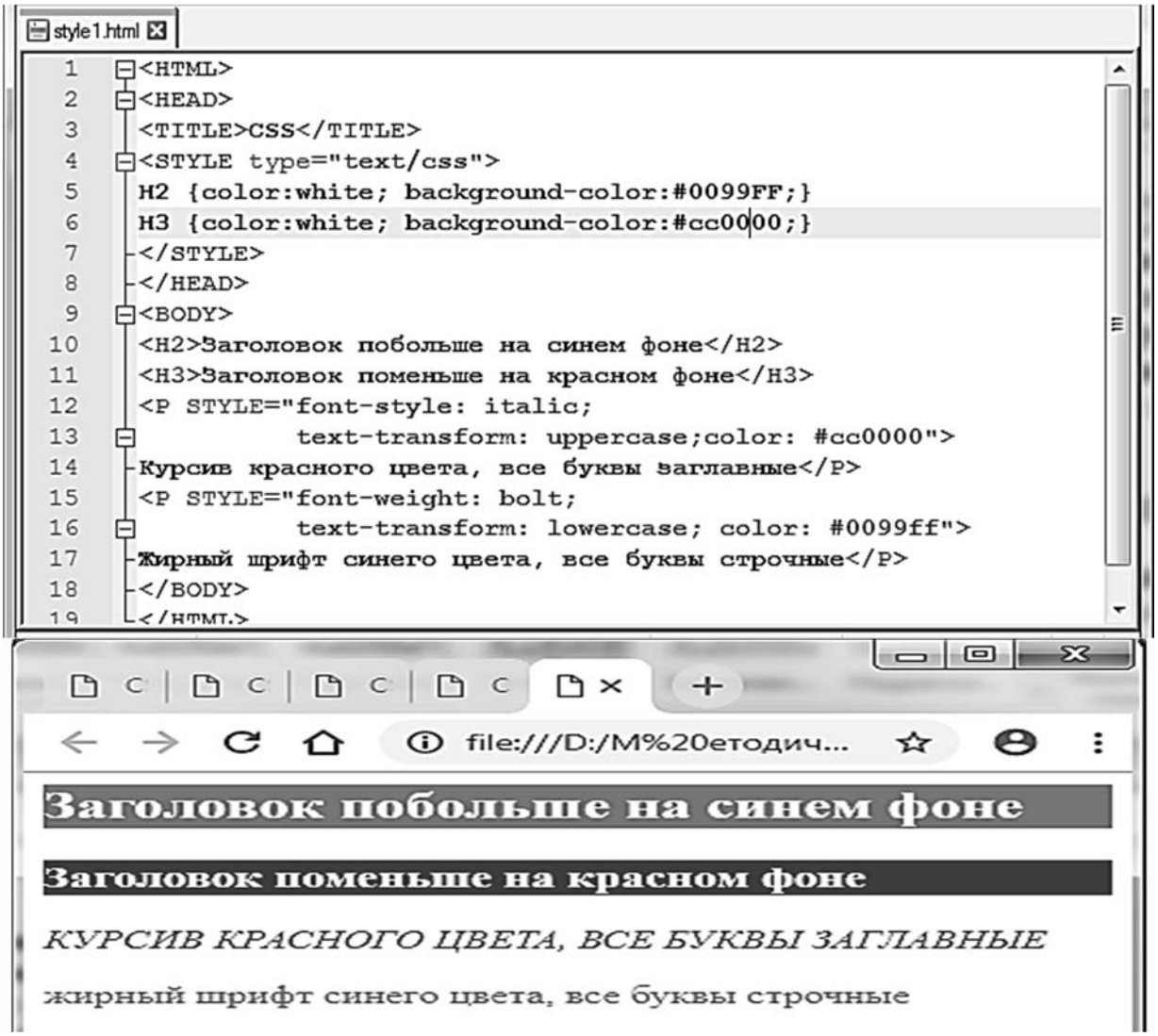

Рисунок 5.1 – Использование CSS в документе *Style1.html*

Порядок выполнения работы

1 Создайте документ *lab5.html*, определите его название.

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

3 Подключите файл *style.css* к документу *lab5.html* (методом связывания).

4 Получите вариант задания у преподавателя.

5 Используя тег *<STYLE>*, определите стиль отдельных тегов в соответствии с вариантом, а также необходимые классы.

Примечание – При выполнении данной работы нельзя использовать теги физического форматирования.

6 Наберите текст задания, используя теги логического форматирования.

7 Для изменения свойств отдельных фрагментов текста используйте метод оперативного определения.

8 В файле *style.css* определите цвет непосещенной, активной, посещенной ссылок, а также цвет ссылки при наведении курсора мыши.

9 Создайте документ *lab5_1.html* и скопируйте в него содержимое документа *lab5.html.*

10 В документе *lab5_1.html* для подключения файла, содержащего стилевые правила, используйте импортирование.

11 Добавьте в документ *lab5.html* ссылку на *lab5_1.html.*

12 Измените вид курсора для элемента *H1* в документе *lab5.html.*

13 Установите фоновое изображение для документа *lab5_1.html*, определите свойства данного фонового изображения (*background-attachment* и т. д.).

14 Добавьте на страницу *lab5.html* изображение. Создайте в *style.css* стилевое правило для элемента IMG, определенного размера изображения и его положения на странице.

Контрольные вопросы

1 Для чего используется CSS?

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

3 Укажите формат правила CSS.

4 Что такое селектор?

5 Что такое классы в CSS? Укажите их назначение.

6 Как определить и применить класс?

7 Как создать класс, не связанный с определенным тегом?

8 Что такое псевдокласс?

6 Лабораторная работа № 6. Основные положения JavaScript

Цель работы

Научиться использовать JavaScript при создании Web-страниц.

Методические указания

Программа (сценарий) на языке JavaScript представляет собой последовательность операторов с точкой с запятой (;) между ними. Если каждый оператор размещается на одной строке, то разделитель можно не писать. Один оператор может располагаться на нескольких строках.

В программах на JavaScript можно использовать комментарии. Для того чтобы задать комментарий, располагающийся на одной строке, достаточно перед его текстом поставить две косые черты (//). Если же поясняющий текст занимает несколько строк, то его следует заключать между символами /* и */. В JavaScript строчные и прописные буквы алфавита считаются разными символами. Любой язык программирования оперирует с постоянными и переменными величинами. В JavaScript это литералы и переменные.

Простейшие данные, с которыми может оперировать программа, называются *литералами*. Литералы не могут изменяться. Литералы целого типа могут быть заданы в десятичном (по основанию 10), шестнадцатеричном (по основанию 16) или восьмеричном (по основанию 8) представлении. Шестнадцатеричные числа включают цифры 0–9 и буквы *a*, *b*, *c*, *d*, *e*, *f*. Шестнадцатеричные числа записываются с символами *0x* перед числом, например, *0x25*, *0xa1*, *0xff*. Запись вещественного литерала отличается от записи вещественного числа в математике тем, что вместо запятой, отделяющей целую часть от дробной, указывается точка, например, 123.34, –22.56. Кроме того, для записи вещественных чисел можно использовать так называемую экспоненциальную форму.

Кроме целых и вещественных значений, в языке JavaScript могут встречаться так называемые логические значения. Существуют только два логических значения: истина и ложь. Первое представляется литералом *true*, второе – *false*. В некоторых реализациях JavaScript может быть использована единица в качестве *true* и ноль в качестве *false*.

Строковый литерал представляется последовательностью символов, заключенной в одинарные или двойные кавычки. Примером строкового литерала может быть строка "результат" или 'результат'.

Элемент, используемый для хранения данных, называется *переменной*.

Тип переменной зависит от хранимых в ней данных, при изменении типа данных меняется тип переменной. Определить переменную можно с помощью оператора *var*, например: *var test1*.

В данном случае тип переменной *test1* не определен и станет известен только после присвоения переменной некоторого значения. Оператор var можно использовать и для инициализации переменной, например, конструкцией *var test2=276* определяется переменная *test2*, и ей присваивается значение *276.*

Значение переменной изменяется в результате выполнения оператора присваивания. Оператор присваивания может быть использован в любом месте программы и способен изменить не только значение, но и тип переменной. Оператор присваивания выглядит так: *a = b*, где *a* – переменная, которой необходимо задать некоторое значение; *b* – выражение, определяющее новое значение переменной.

Пусть в сценарии описаны следующие переменные:

```
var n=3725
 var x=2.75
 var p=true
 var s="Выполнение завершено"
```

n и *x* имеют тип *number*, тип переменной *p* логический, переменная *s* имеет тип *string*. В JavaScript определен тип function для всех стандартных функций и функций, определяемых пользователем. Объекты JavaScript имеют тип данных *object*. Переменные типа *object* часто называют просто объектами, они могут хранить объекты.

Выражения строятся из литералов, переменных, знаков операций, скобок. В зависимости от типа вычисленного значения выражения можно разделить на арифметические, логические и строковые. Арифметические выражения получаются при выполнении операций.

Операции отношения применимы к операндам любого типа. Результат операции – логическое значение *true*, если сравнение верно, и *false* – в противном случае.

Приоритет операций определяет порядок, в котором выполняются операции в выражении.

Сценарии, написанные на языке JavaScript, могут располагаться непосредственно в HTML-документе между тегами <*script*> и </*script*>.

Одним из параметров тега <*script*> является *language*, который определяет используемый язык сценариев. Для языка JavaScript значение параметра равно "JavaScript". Если применяется язык сценариев VBScript, то значение параметра должно быть равным "VBScript". В случае использования языка JavaScript параметр *language* можно опускать, т. к. этот язык выбирается браузером по умолчанию.

Обычно браузеры, не поддерживающие какие-либо теги HTML, эти теги просто игнорируют. Попытка браузера проанализировать содержимое неподдерживаемых тегов может привести к неверному отображению страницы. Чтобы избежать такой ситуации, рекомендуется помещать операторы языка JavaScript в теги комментария <!-- ... -->. Для правильной работы интерпретатора перед закрывающим тегом комментария --> следует поставить символы //.

Таким образом, для размещения сценария в HTML-документе следует написать следующее:

```
<script language="JavaScript">
</script>
```

Документ может содержать несколько тегов <*script*>. Все они последовательно обрабатываются интерпретатором JavaScript. В следующем примере в раздел <*body*> (в тело) HTML-документа вставлены операторы языка JavaScript.

Необходимо написать сценарий, определяющий площадь прямоугольного треугольника по заданным катетам. Сценарий разместим в разделе <*body*> HTML-документа (листинг). Первый сценарий в документе :

```
<HTML>
 <HEAD>
<title>Первый сценарий в документе</title>
</HEAD>
```

```
<BODY>
<P>Страница, содержащая сценарий.</P>
<script>
<!--
var a=8; h=10 /*Инициализируются две переменные*/
document.write ("Площадь прямоугольного треугольника равна ", a*h/2,".") /*Для формирования вывода используется метод write объекта document*/
//-->
</script>
<P>Конец формирования страницы, содержащей сценарий</P>
</BODY>
</HTML>
```

Порядок выполнения работы

1 Проверьте примеры из лабораторной работы.

2 Составьте сценарий, в котором вычисляется площадь круга по заданному радиусу.

3 Составить сценарий, вычисляющий гипотенузу по заданным катетам.

Контрольные вопросы

1 Укажите способы использования кода JavaScript в составе HTML-страницы.

2 Как в JavaScript определяются комментарии?

3 Что называется литералами?

4 Чем отличается запись вещественного литерала от записи вещественного числа в математике?

5 Что располагается между тегами <script> и </script>?

7 Лабораторная работа № 7. Функция и обработка событий JavaScript

Цель работы

Научиться использовать функции при создании Web-страниц.

Методические указания

Основным элементом языка JavaScript является функция. Описание функции имеет вид *function F* (*V*) {*S*}, где *F* – идентификатор функции, задающий имя, по которому можно обращаться к функции; *V* – список параметров функции, разделяемых запятыми; *S* – тело функции, в нем задаются действия, которые нужно выполнить, чтобы получить результат. Необязательный оператор *return* определяет возвращаемое функцией значение. Обычно все определения и функции задаются в разделе <*head*> документа.

Это обеспечивает интерпретацию и сохранение в памяти всех функций при загрузке документа в браузер.

Пример – нахождение площади треугольника. В предыдущих примерах пользователю не предоставлялась возможность вводить значения и в зависимости от них получать результат. Интерактивные документы можно создавать, используя формы. Предположим, что мы хотим создать форму, в которой поля *Основание* и *Высота* служат для ввода соответствующих значений. Кроме того, в форме следует создать кнопку *Вычислить.* При щелчке мышью по этой кнопке нужно получить значение площади треугольника. Действие пользователя (например, щелчок кнопкой мыши) вызывает событие. События в основном связаны с действиями, производимыми пользователем с элементами форм HTML. Обычно перехват и обработка события задаются в параметрах элементов форм. Имя параметра обработки события начинается с приставки *on*, за которой следует имя самого события. Например, параметр обработки события *click* будет выглядеть как *onclick* (листинг 1).

Листинг 1. Реакция на событие Click.

```
<HTML>
<HEAD>
<title>Обработка значений из формы</title>
<script language="JavaScript">
<!--//
function care (a, h)
{
var s=(a*h)/2;
document.write ("Площадь прямоугольного треугольника равна ",s);
return s
}
//-->
</script>
</HEAD>
<BODY>
<P>Пример сценария со значениями из формы</P>
<FORM name="form1">
Основание: <input type="text" size=5 name="st1"><hr>
Высота: <input type="text" size=5 name="st2"><hr>
<input type="button" value=Вычислить
onClick="care(document.form1.st1.value, document.form1.st2.value)"> /*По клику мыши на кнопке в функцию care передаются два параметра - содержимое полей ввода*/
</FORM>
</BODY>
</HTML>
```

При интерпретации HTML-страницы браузером создаются объекты JavaScript. Взаимосвязь объектов между собой представляет иерархическую структуру. На самом верхнем уровне иерархии находится объект *windows*, представляющий окно браузера. Объект *windows* является «предком» или «родителем» всех остальных объектов. Каждая страница, кроме объекта *windows*, имеет объект *document*. Свойства объекта *document* определяются

содержимым самого документа: цвет фона, цвет шрифта и т. д. Для получения значения основания треугольника, введенного в первом поле формы, должна быть выполнена конструкция *document.forml.stl.value*, т. е. (при этом читаем с конца) используем данные *value* из поля ввода с именем *st1*, находящегося на форме *form1* объекта document.

Пример вычисления площади квадрата. Напишем сценарий, определяющий площадь квадрата по заданной стороне. Площадь должна вычисляться в тот момент, когда изменилось значение его стороны. Пусть форма содержит два текстовых поля: одно для длины стороны квадрата, другое для вычисленной площади. Кнопка Обновить очищает поля формы. Площадь квадрата вычисляется при возникновении события change, которое происходит в тот момент, когда значение элемента формы с именем num1 изменилось и элемент потерял фокус. HTML-код представлен в листинге 2.

Листинг 2. Реакция на событие Change.

```
<HTML>
<HEAD>
<title>Обработка события Change - изменение значения элемента</title>
<script>
function srec(obj)
{obj.res.value=obj.num1.value* obj.num1.value}
</script>
</HEAD>
<BODY>
<P>Вычисление площади квадрата</P>
<FORM name="form1">
Сторона: <input type="text" size=7 name="num1"
onChange="srec(form1)">
<hr>
Площадь: <input type="text" size=7 name="res"><hr>
<input type="reset" value=Обновить>
</FORM>
</BODY>
</HTML>
```

Событие *Focus* возникает в момент, когда пользователь переходит к элементу формы с помощью клавиши *<Tab>* или щелчка мыши. Событие «потеря фокуса» (*Blur*) происходит в тот момент, когда элемент формы теряет фокус. Событие *select* вызывается выбором части или всего текста в текстовом поле. Например, щелкнув дважды мышью по полю, выделяют поле, наступит событие *select*, обработка которого приведет к вычислению требуемого значения. В языке JavaScript определены некоторые стандартные объекты и функции, использовать которые можно без предварительного описания. Одним из стандартных объектов является объект *Math*. В свойствах упомянутого объекта хранятся основные математические константы, а его методы можно использовать для вызова основных математических функций. Выражение $y = log\ x$ запишется как $y = Math.log(x)$.

Порядок выполнения работы

1 Проверьте примеры из лабораторной работы.

2 На плоскости заданы координаты трех точек. Напишите сценарий, который вычисляет площадь треугольника (использовать событие *Focus*).

3 Напишите сценарий, который для точки, заданной координатами на плоскости, определяет расстояние до начала координат (использовать событие *Select*).

4 Напишите сценарий, который обменивает местами значения двух введенных переменных (использовать событие *Blur*).

Контрольные вопросы

1 Какой вид имеет описание функции на языке JavaScript?

2 В каком разделе HTML-документа задаются определения и функции?

3 Что понимается под определением «событие»?

4 Перечислите категории событий.

5 Перечислите известные вам события, получающие фокус ввода.

8 Лабораторная работа № 8. Организация ветвлений и циклов на JavaScript. Объекты JS

Цель работы

Освоение методов организации ветвлений и циклов на JavaScript при создании Web-страниц.

Методические указания

При составлении программы часто необходимо выполнение различных действий в зависимости от результатов проверки некоторых условий. Для организации ветвлений можно воспользоваться условным оператором, который имеет вид: *if B {S1} else {S2}*, где *B* – выражение логического типа; *S*1 и *S*2 – операторы. Выполнение условного оператора осуществляется следующим образом. Вычисляется значение выражения *B*. Если оно истинно, то выполняются операторы *S*1, если ложно – операторы *S*2. Если последовательность операторов *S*1 или *S*2 состоит лишь из одного оператора, то фигурные скобки можно опустить. Возможна сокращенная форма условного оператора: *if B {S}*, где *B* – выражение логического типа; *S* – последовательность операторов. Выполнение краткого условного оператора осуществляется так: вычисляется значение выражения *B*, и если оно истинно, то выполняются операторы *S*.

Пример 1 – Для трех заданных значений *a*, *b*, *c* необходимо написать сценарий, определяющий максимальное значение.

Поступим следующим образом. Сначала максимальным значением *m* будем считать значение *a*, далее значение *b* сравним *c* максимальным. Если окажется, что *b* больше *m*, то максимальным становится *b*. И, наконец, значение *c* сравнивается с максимальным значением из предыдущих значений *a* и *b*. Если *c* больше *m*, то максимальным становится *c*. Оператор присваивания *obj.res.value=m* обеспечивает запись вычисленного максимального значения в соответствующее поле формы. Функция *Number*(*s*) преобразует объект *s*, заданный в качестве параметра, в число. Полностью сценарий может быть записан так, как представлено в листинге 1.

Листинг 1. Вычисление максимального значения из трех заданных.

```
<HTML>
 <HEAD>
 <TITLE>Вычисление максимального значения</TITLE>
 <script language="JavaScript">
 <!-- //
function maxval (obj )
 {
 var a = Number(obj.num1.value);
 var b = Number(obj.num2.value);
 var c = Number(obj.num3.value);
 var m=a
 if (b > m) m=b
 if (c > m) m=c
 obj.res.value=m }
 //-->
 </script>
 </HEAD>
 <BODY>
 <H4>Вычисление максимального значения</H4>
 <FORM name="form1">
 Число 1: <input type="text" size=8 name="num1"><hr>
 Число 2: <input type="text" size=8 name="num2"><hr>
 Число 3: <input type="text" size=8 name="num3"><hr>
 Максимальное значение равно
 <input type="button" value=Определить onClick="maxval(form1)">
 <input type="text" size=8 name="res"><hr>
 <input type="reset">
 </FORM>
 </BODY>
 </HTML>
```

Решим рассмотренную задачу другим способом. Вспомним, что стандартный объект *Math* имеет метод *max*, который определяет наибольшее

значение двух аргументов. Опишем функцию *maxval1*, которая определяет максимальное значение из трех заданных значений и использует объект *Math*.

```
function maxval1 (obj )
 {
 var a = Number(obj.num1.value);
 var b = Number(obj.num2.value);
 var c = Number(obj.num3.value);
 obj.res.value=Math.max(Math.max(a,b),c)
 }
```

Если бы требовалось определить максимальное из четырех заданных значений *a*, *b*, *c*, *d*, то можно было бы воспользоваться формулой

Math.max(Math.max(a,b), Math.max(c,d)).

Организация циклов в JavaScript. Для успешного решения широкого круга задач требуется многократно повторить некоторую последовательность действий, записанную в программе один раз. В том случае, когда число повторений последовательности действий нам неизвестно либо число повторений зависит от некоторых условий, можно воспользоваться оператором цикла вида: *while* (*B*) {*s*}
где *B* – выражение логического типа; *s* – операторы, называемые телом цикла. Операторы *s* в фигурных скобках выполняются до тех пор, пока условие *B* не станет ложным.

Пример 2 – Нахождение общего делителя. Необходимо написать программу, которая для двух заданных чисел определяет наибольший общий делитель.

При решении задачи воспользуемся алгоритмом Евклида. Если значение *m* равно нулю, то наибольший общий делитель чисел *n* и *m* равен *n*: *НОД*(*n*, *0*) = *n*.

В остальных случаях верно следующее соотношение: *НОД*(*n*, *m*) = = *НОД*(*m*, *n*%*m*). В функции *nod* переменная p используется для получения остатка от деления чисел *n* и *m* (листинг 2). Выполнение цикла продолжается до тех пор, пока значение *p* не станет равным нулю. Последнее вычисленное значение *m* равно наибольшему общему делителю.

Листинг 2. Наибольший общий делитель двух чисел.

```
<HTML>
 <HEAD>
 <TITLE>Наибольший общий делитель двух чисел</TITLE>
 <script language="JavaScript">
 <!-- //
 function nod(obj)
 { var n=obj.num1.value
 var m=obj.num2.value
 var p = n%m
 while (p!=0)
 { n=m
 m=p
 p=n%m
 }
```

```
obj.res.value=m
}
//-->
</script>
</HEAD>
<BODY>
Наибольший общий делитель двух заданных чисел
<FORM name="form1">
Введите число <input type="text" name="num1" size="8"><br>
Введите число <input type="text" name="num2" size="8"><br>
<input type="button" value="Вычислить" onClick="nod(form1)"><br>
Наибольший общий делитель <input type="text" name="res"
size="8"><hr>
<input type="reset" value="Отменить">
</FORM>
</BODY>
</HTML>
```

Если число повторений заранее известно, то можно воспользоваться следующим оператором цикла, который часто называют оператором цикла арифметического типа. Синтаксис этого оператора таков: *for* (*A*; *B*; *I*){*S*}. Выражение *A* служит для инициализации параметра цикла и вычисляется один раз в начале выполнения цикла. Выражение *B* (условие продолжения) управляет работой цикла. Если значение выражения ложно, то выполнение цикла завершается, если истинно, то выполняется оператор *S*, составляющий тело цикла. Выражение *I* служит для изменения значения параметра цикла. После выполнения тела цикла *S* вычисляется значение выражения I, затем опять вычисляется значение выражения *B* и т. д. Цикл может прекратить свою работу в результате выполнения оператора *break* в теле цикла.

Пример 3 – Написать программу, определяющую, является ли заданное число *n* совершенным. Совершенным называется число *n*, равное сумме своих делителей, не считая самого числа. Например, число 6 является совершенным, т. к. верно 6 = 1 + 2 + 3, где 1, 2, 3 – делители числа 6. Число 28 также является совершенным, справедливо равенство 28 = 1 + 2 + 4 + 7 + 14. При решении задачи воспользуемся только функцией *sumdei* (листинг 3).

Листинг 3. Итерационные методы. Совершенные числа

```
 <HTML>
<HEAD>
<TITLE>Итерационные методы. Совершенные числа</TITLE>
<script language="JavaScript">
<!-- //
function sumdel(n)
{ var s=1;
for (var i=2; i<=n/2; i++)
{ if (n % i == 0) s += i }
 return s
}
function sov(obj)
```

```
{ var n=obj.numb.value;
var s=""
if (n==sumdel(n)) s="совершенное"
else s="не является совершенным"
return s
}
//-->
</script>
</HEAD>
<BODY>
<P> Итерационные методы. Совершенные числа</P>
<FORM name="form0">
Введите натуральное число: <input type="text" size=8 name="numb">
<input type="button" value=Выполнить
onClick="this.form.res.value=sov(form0)"><hr>
Данное число: <input type="text" size=24 name="res"><hr>
<input type="reset" value=Отменить>
</FORM>
</BODY>
</HTML>
```

Следует обратить внимание на значение параметра обработки события. В данном случае это оператор присваивания, в правой части которого вызов функции *sov*.

Порядок выполнения работы

1 Проверьте примеры из лабораторной работы.

2 Вводится последовательность из пяти чисел. Напишите сценарий, в котором определяется число максимальных элементов.

3 Напишите программу, которая определяет, можно ли построить треугольник с заданными длинами сторон.

4 Точка на плоскости задается своими координатами. Определите, какой из четвертей прямоугольной системы координат принадлежит заданная точка.

5 Напишите программу, которая «переворачивает» заданное натуральное число.

6 Напишите сценарий, в котором определяется количество «счастливых» шестизначных автобусных билетов, т. е. таких, в номерах которых сумма первых трех цифр равна сумме трех последних.

7 Напишите программу, определяющую все делители заданного натурального числа.

Контрольные вопросы

1 В каких случаях эффективно использовать механизм ветвления JavaScript в HTML-документах?

2 В каких случаях эффективно использовать механизм организации циклов JavaScript в HTML-документах?

3 Какой оператор используется для организации ветвлений?
4 Какой вид могут иметь операторы условного перехода?
5 Когда используется оператор цикла вида *while* (*B*) {*s*}?

9 Лабораторная работа № 9. Селекторы и методы jQuery

Цель работы

Приобретение навыков создания HTML-документов с использованием jQuery, в который включены действия над отобранными элементами.

Методические указания

Для создания интересных интерактивных сайтов разработчики используют библиотеку JavaScript, такую как jQuery, чтобы автоматизировать решение наиболее типичных задач и упростить решение более сложных. Библиотека jQuery имеет весьма широкие функциональные возможности, согласованную и симметричную архитектуру (большая часть концепций заимствована из HTML и каскадных таблиц стилей CSS). Библиотека имеет расширяемую архитектуру и позволяет решать следующие задачи:

– доступ к элементам документа;
– изменение внешнего вида страницы;
– изменение содержимого документа;
– отклик на действия пользователя;
– воспроизведение анимационных эффектов в документе;
– извлечение информации со стороны сервера без полного обновления страницы;
– упрощение решения типичных задач программирования на JavaScript.

Чтобы обеспечить широкий круг возможностей, обозначенных выше, и остаться при этом достаточно компактной, библиотека jQuery использует несколько стратегий:

– использование знаний о CSS;.
– поддержка расширений;
– абстрактный способ обхода несовместимостей браузеров;
– работа с наборами;
– возможность выполнения множества операций в одной строке.

Большинство примеров использования jQuery состоит из трех частей: собственно документ HTML, файлы CSS, где содержатся определения стилей, и файлы JavaScript, выполняющие операции над документом. Программный код будет находиться в файле JavaScript, который подключается к документу HTML с помощью тега *<script src="alice.js" type="text/javascript"></script>*.

Для получения простого и быстрого доступа к элементам или группам элементов в объектной модели документа (Document Object Model, DOM) библиотека jQuery предоставляет мощные селекторы каскадных таблиц стилей

(Cascading Style Sheets, CSS). Множества элементов, отбираемых методами с помощью селекторов, всегда упаковываются в объекты jQuery. Эти объекты jQuery просты в обращении при выполнении операции над элементами страницы. К этим объектам можно легко привязать обработчиков событий, добавлять эффекты, а также объединять в цепочки несколько операций или эффектов.

Независимо от того, какого типа селектор будет использоваться, вызов любых операций из библиотеки jQuery всегда начинается со знака доллара и круглых скобок: *$()*. Почти все, что может использоваться в таблицах стилей, точно так же может обертываться в кавычки и помещаться между круглыми скобками. Селекторы составляются из трех основных строительных блоков: *имени тега*, *идентификатора* (*ID*) *и класса*. Они могут использоваться самостоятельно или в комбинации с другими селекторами. В таблице 9.1 показано, как выглядит каждый из этих трех селекторов.

Таблица 9.1 – Селекторы CSS и jQuery

Селектор	CSS	jQuery	Описание
Имя тега	*P*	*$('P')*	Выбирает все параграфы в документе
Идентификатор (ID)	*#some-id*	*$('#some-id')*	Выбирает единственный элемент документа с идентификатором *some-id*
Класс	*.some-class*	$('.some-class')	Выбирает все элементы документа, имеющие класс *some-class*

Метод .text() возвращает или изменяет текстовое содержимое выбранных элементов страницы. Функция имеет три варианта использования:

1) .text():stringv:1.0 – возвращает текст, содержащийся в выбранном элементе. Если таких элементов несколько, метод возвратит строку, в которой будет содержимое всех элементов, расположенное через пробел.

2) .text(newText):jQueryv:1.0 – заменяет все содержимое у выбранных элементов, на текст newText.

3) .text(function(index, value)):jQueryv:1.4 – заменяет все содержимое у выбранных элементов на возвращенный пользовательской функцией текст. Функция вызывается отдельно для каждого из выбранных элементов. При вызове ей передаются следующие параметры: index – позиция элемента в наборе, value – текущий текст элемента.

Замечание – Если попытаться с помощью метода text() поместить в элемент другие элементы с помощью html-текста, то jQuery будет экранировать все теги, и в результате на странице появится html-текст вместо html-элементов. Для вставки html-элементов нужно использовать метод .html().

Примеры использования.

$(".topBlock").text() – вернет текстовое содержимое всех элементов с классом topBlock (одной строкой).

$(".topBlock").text("Новье!") – заменит содержимое всех элементов с классом *topBlock* на текст «Новье!».

Пример 1 – Поместить в элемент с классом demo-container текст:

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
  <div class="demo-container">
    <div class="demo-box">Контейнер для демонстраций</div>
    <ul>
      <li>Первый</li>
      <li>Второй</li>
    </ul>
  </div>
<script>
  $('div.demo-container').text('<p> А вот и текст! </p>');
</script>
</body>
</html>
```

Метод .hide() и .show() – без каких-либо параметров, можно рассматривать как сокращенный вариант метода *.css*(*'display'*, *'string'*), где *string* это соответствующее значение свойства *display*. Эффект от применения этих методов – выбранный набор элементов, немедленно спрячется или отобразится без всякой анимации. Метод *.hide*() устанавливает линейный атрибут стиля выбранному набору элементов в значение *display*: *none*. Сложная часть здесь в том, что он запоминает значение свойства *display* (обычно *block* или *inline*), прежде чем изменит его на значение none. В свою очередь, метод *.show*() восстанавливает выбранному набору элементов любое видимое свойство *display*, которое у них было установлено, до применения *display*: *none*.

Пример 2 – На основе этих двух методов построить простейшее выпадающее меню на JQuery. Для начала понадобится листинг 1.

Листинг1.

```
<div id="menu">
        <ul>
          <li>
            <a href="#">первый пункт</a>
            <ul>
              <li><a href="#">пункт 1</a></li>
              <li><a href="#">пункт 2</a></li>
              <li><a href="#">пункт 3</a></li>
            </ul>
          </li>
          <li>
            <a href="#">второй пункт</a>
             <ul>
              <li><a href="#">пункт 1</a></li>
```

```
            <li><a href="#">пункт 2</a></li>
            <li><a href="#">пункт 3</a></li>
          </ul>
        </li>
        <li>
          <a href="#">третий пункт</a>
          <ul>
            <li><a href="#">пункт 1</a></li>
            <li><a href="#">пункт 2</a></li>
            <li><a href="#">пункт 3</a></li>
          </ul>
        </li>
        <li>
          <a href="#">четвертый пункт</a>
          <ul>
            <li><a href="#">пункт 1</a></li>
            <li><a href="#">пункт 2</a></li>
            <li><a href="#">пункт 3</a></li>
          </ul>
        </li>
      </ul>
    </div>
```

Далее добавим CSS-стили:

```
#menu {
    margin: 20px 0;
    clear: both;
    font-size: 12px;
}
ul {
    list-style: none;
    display: block;
}
ul li {
    list-style: none;
    float: left;
    display: block;
}
ul li a {
    text-decoration: none;
    color: #fff;
    background: #055788 url(menu.png) repeat-x;
    padding: 7px;
    border: 1px #2E5C09 solid;
}
ul li ul{
    display: none;
    float: none;
    margin: 5px 0 0 0;
    padding: 0;
}
ul li ul li{
    display: block;
```

```
    float: none;
}
ul li ul li a{
    background:  #000;
    display: block;
    padding: 7px;
    border: 1px #2E5C09 solid;
}
ul li ul li a:hover{
    display: block;
    background: #63B024;
}
```

Теперь перейдем непосредственно к коду:

```
$(document).ready(function(){
        $("#menu ul li").hover(function(){
            $(this).find('ul').show();
        }, function(){
            $(this).find('ul').hide();
        })
    })
```

Как видно из вышеприведенного примера, использовалось событие *.hover*(), которое включает в себя двух обработчиков событий: наведение мыши на элемент и отведение мыши от элемента. Когда пользователь подводит мышь, отображается выпадающее меню, Как только мышь перемещается в другое место, выпадающее меню снова исчезает (рисунок 9.1).

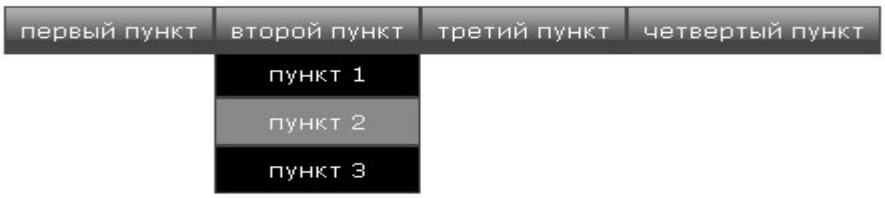

Рисунок 9.1 – Простое выпадающее меню

Когда применяется скорость к методам *.show*() или *.hide*(), они становятся анимированными в течение некоторого периода времени. Например, метод *.hide*(*'speed'*) будет уменьшать высоту элемента, ширину и прозрачность, до тех пор, пока все три значения не станут равными 0, т. е. пока не вступит в силу CSS-правило *display*: *none*. Метод *.show*(*'speed*') будет увеличивать высоту элемента сверху вниз, ширину слева направо и прозрачность от 0 до 1, пока все содержимое элемента не станет полностью видимым. Можем использовать одну из трех скоростей: *slow*, *normal* и *fast*. Использование *.show*(*'slow'*) выполнит эффект появления в течение 0,6 с, *.show*(*'normal'*) – 0,4 с. и *show*(*'fast'*) – 0,2 с. Для еще большей точности, можно указать количество миллисекунд, например, *.show*(*850*). В отличие от имен скоростей числовые значения можно не заключать в кавычки.

Пример 3 – Включим скорость в пример 1 (листинги 2 и 3, рисунок 9.2.).

Листинг 2. Использование затухания.

```
$(document).ready(function(){
        $("#menu ul li").hover(function(){
            $(this).find('ul').show('slow');
        }, function(){
            $(this).find('ul').hide();
        })
    })
```

Листинг 3. Выпадающее меню появлялось путем постепенного изменения прозрачности с использованием метода *.fadeIn*(*'slow'*):

```
$(document).ready(function(){
        $("#menu ul li").hover(function(){
      $(this).find('ul').fadeIn('slow');
            $(this).find('ul').show();
        }, function(){
            $(this).find('ul').hide();
        })
    })
```

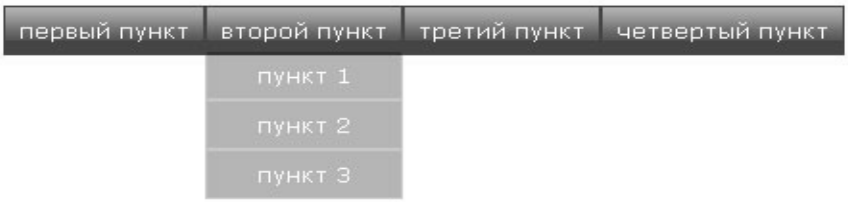

Рисунок 9.2 – Выпадающее меню

Метод *attr* **–** получение доступа к свойству первого совпавшего элемента (имя – *attr*(*имя*); тип – *Возвращает*: *Объект*). Используя этот метод можно легко получить значение свойства первого совпавшего элемента. Если элемент не имеет указанного атрибут а, то возвращается *undefined* (не определено). Атрибутами могут быть: *title*, *alt*, *src*, *href*, *width*, *style* и т. д. (*attr*(*свойства*) *Возвращает*: *jQuery*).

Метод *attr* устанавливает атрибуты всех элементов набора, используя при этом объект, который содержит пары ключ/значение (*attr*(*ключ*, *значение*) *Возвращает*: *jQuery*).

Метод *attr* изменяет значение единственного свойства для каждого совпавшего элемента (*attr*(*ключ*, *функция*) *Возвращает*: *jQuery*). Но вместо непосредственно значения указывается функция, которая возвращает значение как свой результат

(*removeAttr*(*имя*) *Возвращает*: *jQuery* – удаляет указанный атрибут из каждого совпавшего элемента.

(*attr*(*name*)) **–** обеспечивает доступ к значению указанного атрибута первого элемента в наборе.

Пример 4.

```
var a=$("i").attr("title");
 $("div").text(a);
```

Данная инструкция найдет первый элемент в тегах *i*, найдет атрибут *title* этого элемента и добавит его значение в *div*.

Метод *attr*(*name*) получает значение заданного атрибута соответствующего элемента набора jQuery либо первого элемента в наборе jQuery (если их несколько). Возвращает значение *undefined*, если у элемента указанный атрибут отсутствует или в наборе нет элементов. Возвращаемое значение: (*строка*) *Значение искомого атрибута* или *undefined*. *Параметры*: *name* – (*строка*) *Имя атрибута*, значение которого необходимо получить.

Примечание – Чтобы получить значения атрибутов для каждого элемента в наборе jQuery, можно использовать методы .each() или .map().а также собственные нестандартные имена атрибутов. Библиотека jQuery предоставляет дополнительные нормализованные имена атрибутов для кросс-браузерной работы команды .attr(): class (от ClassName), float, cssFloat, styleFloat, for (от HtmlFor), maxlength, readonly. Удалить атрибут средствами jQuery можно с помощью функции .removeAttr().

Пример 5 – Получить значение атрибута *alt* первого изображения на странице и, используя его, установить значение атрибута *title*.

```
var title = $("img").attr("alt") + " -> Увеличить";
$("img:first").attr("title", title);
```

Пример 6 – Сохранить в массив значения атрибутов *id* всех элементов <*div*> в документе.

```
var arr = new Array();
$("div").each(function(){
   arr[] = $(this).attr("id");
});
```

attr(properties) – установит атрибуты во всех отобранных элементах.

Пример 7 – Данная инструкция найдет все картинки и установит им соответствующие атрибуты.

```
$("img").attr({src:"images/pict.gif", alt:"рисунок"});
```

attr(*key*,*value*) – установит значение *value* атрибута *key* для всех отобранных элементов.

Пример 8 – Данная инструкция установит для всех кнопок значение *"disabled"* атрибута *"disabled"*.

```
$("button").attr("disabled", "disabled").
```

removeAttr(*name*) – удалит указанный атрибут у всех элементов.

Пример 9 – Данная инструкция удалит атрибут *"alt"* у всех картинок.

```
$("img").removeAttr("alt");
```

Все эффекты анимации в jQuery выполняются вокруг функции *animate.* Данная функция берет один или несколько CSS свойств элемента и изменяет их исходного до конечного за *N*-е количество итераций (количество итераций зависит от указанного времени, но не реже одной итерации в 13ms.). Функция *animate* понимает следующие параметры:

– *params* – описание CSS-свойств элемента, до которых будет происходить анимация (то есть если *div* с высотой 100*px*, то *animate*({*height:200*}) плавно изменит высоту до 200*px*);

– *duration* – скорость анимации – задается в миллисекундах или с помощью ключевых слов: *“fast” = 200ms*, *“normal” = 400ms* или *“slow” = 600ms*;

– *easing* – указывает, какую функцию будем использовать для наращивания значений, на выбор *“linear”* или *“swing”*;

– *callback* – функция, которая будет вызвана после окончания анимации.

Альтернативный способ инициализации:

– *params* – описание CSS-свойств элемента, до которых будет происходить анимация;

– *options* – объект настроек;

– *complete* – аналогичен описанному *callback*-параметру;

– *step* – дополнительная *callback* функция – обеспечивает пошаговое изменение параметров;

– *queue* – флаг очереди, если выставить в *false*, то данная анимация будет игнорировать очередь и запустится сразу.

Вспомогательные функции:

– *show*( ) – отображает выбранные элементы;

– *hide*( ) – скрывает выбранные элементы;

– *toggle*( ) – переключатель между *show/hide*;

– *slideDown*(*speed*, *callback*) – выдвигает объект(ы) вниз – увеличивает высоту от 0 до 100 %;

– *slideUp*(*speed*, *callback*) – задвигает объект(ы) вверх – уменьшает высоту от 100 % до 0;

– *slideToggle*(*speed*, *callback*) – переключатель между *slideDown/slideUp;*

– *fadeIn*(*speed*, *callback*) – отображает выбранные элементы – изменяет прозрачность элементов;

– *fadeOut*(*speed*, *callback*) – скрывает выбранные элементы – изменяет прозрачность элементов;

– *fadeTo*(*speed*, *opacity*, *callback*) – изменяет прозрачность элементов до указанного значения.

Пример 10 – Самые простые методы *hide* и *show* обходятся без функции *animate*, т. к. манипулируют лишь атрибутом *display* (демо):

```
// вызов метода
$('#my').hide();
// аналогичен
$('#my').css({display:"none"});
// но если задать скорость анимации либо callback функцию,
// то будут изменяться значения height и width
```

Метод *Append* добавляет контент внутрь каждого элемента набора. Добавляемый контент следует за уже существующим *append* (*контент*). Данный метод подобен применению *appendChild*. Аргументы: контент *Строка*, *Элемент*, *jQuery*, *Контент*, который необходимо добавить.

Пример 11 – Добавляет код HTML ко всем параграфам.

```
$("p").append("<strong>Hello</strong>");
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <script src="http://code.jquery.com/jquery-latest.js"></script>
 <script>
 $(document).ready(function(){
   $("p").append("<strong>Hello</strong>");
 });
 </script>
 <style>p { background:yellow; }</style>
</head>

<body>
 <p>I would like to say: </p>
</body>
</html>
```

Те же действия можно применить в одну строчку с помощью так называемых «цепных функций».

```
$(document).ready(function(){
$('#example_id').hide(2000).show(2000);
});
```

Функции, описанные выше, позволяют добавлять стили из CSS при помощи классов. Однако функция *css()* позволяет получить доступ непосредственно к таблицам CSS.

Порядок выполнения работы

1 Проверьте примеры из лабораторной работы.

2 Отберите в *jquery* набор все картинки, кроме первой, и скройте их с помощью функции *hide*() за 5 с. Для формирования набора используйте фильтр *not.*

3 Сделайте выборку для картинки второго мотоцикла *moto2.jpg*.

4 С помощью метода *css* добавьте для картинки рамку *1px solid #333333*.

5 Примените анимацию для увеличения рамки до значения в 5*px* за 5 с.

После выполнения анимации плавно скройте шапку с помощью функции *fadeOut*() за 5 с. Используйте цепные функции. Свойство для увеличения размера рамки в обычном *css* записывается как *border-width*.

Контрольные вопросы

1 В каких случаях разработчики используют библиотеки jQuery?
2 Какие задачи позволяют решать библиотеки jQuery?
3 Какие стратегии использует библиотека jQuery?
4 Как осуществляется вызов любых операций из библиотеки jQuery?
5 Какие преимущества дает использование библиотеками jQuery селекторов каскадных таблиц стилей?
6 Назовите основные методы библиотеки jQuery.

10 Лабораторная работа № 10. События jQuery

Цель работы

Изучить события JQuery и научиться использовать события на Web-страницах.

Методические указания

События являются фундаментом для большинства Web-приложений JavaScript. Используя события, код может реагировать на то, что происходит в браузере, например, на то, что пользователь нажал кнопку, ввёл текст в поле ввода или отправил форму.

Событие в JavaScript (и jQuery) генерируется, когда что-то происходит с элементом на странице. В список общих событий входят:

– *click* – генерируется, когда пользователь нажимает на элементе кнопку мыши;

– *dblclick* – генерируется, когда пользователь делает двойной щелчок кнопкой мыши на элементе;

– *mouseover* – генерируется, когда пользователь перемещает указатель мыши на элемент;

– *load* – генерируется после того, как элемент, например, изображение, полностью загружен;

– *submit* – генерируется, когда происходит отправка формы (данное событие генерируется только для элементов form).

Для работы с событиями в jQuery нужно создать функцию, называемую *обработчиком события*, которая будет работать с событием, когда оно произойдёт. Затем вызывается метод jQuery *bind*, который привязывает функцию-обработчика события к определённому событию для выбранного элемента (или элементов). Существует много методов jQuery для привязки событий к обработчикам, но метод *bind*() является основным. Он принимает имя

события и имя функции как аргументы и привязывает обработчика к событию для выбранного элемента (или элементов):

```
$(selector).bind( eventName, functionName );
```

Затем, когда происходит событие, ваша функция-обработчик автоматически запускается и событие обрабатывается так, как требуется.

Пример 1 – Создать простого обработчика события и привязать его к событию *click* кнопки формы, т. е. при нажатии на кнопку должно выводиться в окне сообщение с надписью *"Всем – привет!"*:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$( init );
function init() {
  $('#myButton').bind( 'click', sayHello );
}
function sayHello() {
  alert( "Всем - привет!" );
}
</script>
</head>
<body>
<div>
<input type="button" id="myButton" value="Нажми меня" />
</div>
</body>
</html>
```

jQuery имеет много коротких методов для связывания часто используемых событий с функциями-обработчиками:

– *click*(*functionName*) – эквивалентно вызову *bind*(*'click'*, *functionName*);
– *dblclick*(*functionName*) – эквивалентно вызову *bind*(*'dblclick'*, *functionName*);
– *load*(*functionName*) – эквивалентно вызову *bind*(*'load'*, *functionName*).

Например, можно переписать вызов метода *bind*() следующим образом:

```
$('#myButton').click( sayHello );
```

Когда событие вызывает функцию-обработчика, можно получить доступ к элементу как к объекту DOM из функции-обработчика с помощью ключевого слова *this*. Это означает, что можно получить больше информации об элементе, для которого сгенерировано событие, можно манипулировать данным элементом и т. д.

Пример 2 – Создать пульсацию кнопки (плавно затухает и плавно высвечивается снова) при нажатии на неё. Чтобы выполнить поставленную задачу, обработчик события получает доступ к объекту нажатой кнопки с помощью *this*, оборачивает его объектом jQuery, а затем вызывает методы jQuery *fadeOut*() *и fadeIn*() для организации пульсирования кнопки:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$( init );
function init() {
  $('#myButton').bind( 'click', pulsate );
}
function pulsate() {
  $(this).fadeOut();
  $(this).fadeIn();
}
</script>
</head>
<body>
<div>
<input type="button" id="myButton" value="Нажми меня" />
</div>
</body>
</html>
```

Порядок выполнения работы

1 Проверьте примеры из лабораторной работы.
2 По указанию преподавателя выполните задание с тестовой страницей.

Контрольные вопросы

1 Что входит в список общих событий в JavaScript (и jQuery)?
2 Что нужно создать для работы с событиями в jQuery?
3 Что привязывает функцию-обработчика события к определенному событию для выбранного элемента?

Варианты заданий

1 Напишите обработчика события, который делает следующее. При клике мышью на любом из названий мотоциклов в этом блоке:

```
<ol id="moto_models">
      <li>Харлей Дэвидсон</li>
```

*
Кроссовый мотоцикл
*
*
Гоночный мотоцикл
*
*
Концептуальный мотоцикл
*
*
*

В таблице с *id="moto_table"* должна появляться новая строчка, в которой плавно появляется текст из элемента, по которому кликнули.

2 Напишите обработчика события, который будет подсвечивать строку таблицы (*id="moto_table"*), при наведении на нее курсора мыши, цветом *#1F233C*, а текст будет делать белым.

3 С помощью события *hover*() и функции *css*() сделайте так, чтобы при наведении на кнопку «Отправить заявку», ее свойства менялись:

а) фоновый цвет должен меняться на *#32375D*;

б) цвет текста – на белый.

Когда же курсор мыши убирается с элемента, то все должно возвратиться на свои места.

4 С помощью *toggle*() сделайте так, чтобы при первом щелчке строка таблицы подсвечивалась, а при повторном принимала обычное положение (здесь уже *css*() применять необязательно, можно использовать классы).

5 Напишите скрипт, который делает следующее. При щелчке на любой из картинок четырех мотоциклов вокруг должна появляться *сплошная рамка* в *4px.* При этом если в момент щелчка мышью по картинке была зажата клавиша *shift*, то цвет рамки должен быть *#cc0000*, а если не была зажата, то цвет рамки должен быть *#333333*. При повторном щелчке по той же самой картинке рамка должна исчезать.

6 Сделайте так, чтобы во время нажатия кнопки отправки формы с *id="#my_button"*, появлялось сообщение с вопросом: «Вы подтверждаете правильность ввода данных?» В этом окошке должны быть кнопки *Ок* и *Отмена.* Если человек нажмет *Ок*, то форма отправляется как положено, а если нажмет *Отмена*, то форма не отправляется. Для задания вопроса используйте метод *confirm*().

Список литературы

1 **Лисьев, Г. А.** Программное обеспечение компьютерных сетей и web-серверов : учебное пособие / Г. А. Лисьев, П. Ю. Романов, Ю. И. Аскерко. – Москва : ИНФРА-М, 2020. – 145 с.

2 **Прохоренок, Н. А**. HTML, JavaScript, PHP и MySQL. Джентельменский набор Web-мастера / Н. А. Прохоренок. – 2-е изд. – Санкт-Петербург: БХВ-Петербург, 2009. – 880 с.

3 **Каслдайн, Э.** Изучаем jQuery / Э. Каслдайн, К. Шарки. – 2-е изд. – Санкт-Петербург: Питер, 2012. – 400 с.: ил.