Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Филиал «Минский радиотехнический колледж»

**УТВЕРЖДаю**

Директор МРК

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

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

Регистрационный №\_\_\_\_\_\_\_\_

**Программные средства создания Internet-приложений**

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**

**по выполнению лабораторных работ**

для учащихся специальности

2 40 01 01 «Программное обеспечение информационных технологий»

Минск 2018

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

**РЕКОМЕНДОВАНЫ К УТВЕРЖДЕНИЮ:**

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

Протокол № от

Заседанием педагогического совета филиала БГУИР «Минский радиотехнический колледж»

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

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

подпись ФИО

**Содержание**

**Лабораторная работа № 1** 6

**Тема работы: «Создание простейшего HTML-документа. Физическое и логическое форматирование текста на HTML-странице»** 6

**Лабораторная работа № 2** 14

**Тема работы: «Организация системы ссылок сайта. Работа с изображениями. Создание списков и таблиц на HTML-странице»** 14

**Лабораторная работа № 3** 25

**Тема работы: «Использование слоев и изображений-карт на HTML-странице»** 25

**Лабораторная работа № 4** 37

**Тема работы: «Создание форм на HTML-странице и изменение свойств элементов формы»** 37

**Лабораторная работа № 5** 46

**Тема работы: «Подключение и использование каскадных таблиц стилей»** 46

**Лабораторная работа № 6** 58

**Тема работы: «Разработка таблиц стилей для HTML документа»** 58

**Лабораторная работа № 7** 72

**Тема работы: «Разработка второго варианта таблиц стилей для HTML документа»** 72

**Лабораторная работа № 8** 81

**Тема работы: «Подготовка psd-макета к верстке»** 81

**Лабораторная работа № 9** 100

**Тема работы: «Создание HTML-каркаса макета. Сброс стилей браузеров»** 100

**Лабораторная работа № 10** 104

**Тема работы: «Разработка CSS-документа в соответствии с макетом»** 104

**Лабораторная работа № 11** 110

**Тема работы: «Тестирование кроссбраузерного отображения верстки. Размещение проекта на хостинге»** 110

**Лабораторная работа № 12** 114

**Тема работы: «Создание анимации средствами CSS»** 114

**Лабораторная работа № 13** 133

**Тема работы: «Создание XML-документа. Изучение различий языков разметки XML и HTML»** 133

**Лабораторная работа № 14** 138

**Тема работы: «Описание структуры XML-документа средствами DTD и на языке XSD»** 138

**Лабораторная работа № 15** 153

**Тема работы: «Форматирование вывода и преобразование XML-документа средствами SLT»** 153

**Лабораторная работа № 16** 160

**Тема работы: «Обработка событий с использованием DHTML»** 160

**Лабораторная работа № 17** 167

**Тема работы: «Создание HTML-страниц с использованием сценариев JavaScript. Использование управляющих конструкций в JavaScript»** 167

**Лабораторная работа № 18** 181

**Тема работы: «Работа с массивами в JavaScript. Применение методов работы с массивами»** 181

**Лабораторная работа № 19** 192

**Тема работы: «Использование функций при процедурном подходе в программировании на языке JavaScript. Управление видимостью переменных при помощи замыканий»** 192

**Лабораторная работа № 20** 203

**Тема работы: «Создание объектов и методов. Управление контекстом вызова»** 203

**Лабораторная работа № 21** 212

**Тема работы: «Работа со стандартными объектами JavaScript: объект Date, объект Math»** 212

**Лабораторная работа № 22** 222

**Тема работы: «Работа с элементами объекта браузера средствами JavaScript. Обработка событий браузера»** 222

**Лабораторная работа № 23** 237

**Тема работы: «Использование объектов HTML и объектной модели документа»** 237

**Лабораторная работа № 24** 255

**Тема работы: «Использование разных моделей событий. Остановка всплытия и перехват событий»** 255

**Лабораторная работа № 25** 275

**Тема работы: «Проверка корректности информации, введенной пользователем. Обработка данных формы»** 275

**Лабораторная работа № 26** 292

**Тема работы: «Управление CSS-свойствами HTML-элементов при помощи языка JavaScript»** 292

**Лабораторная работа № 27** 296

**Тема работы: «Применение объектно-ориентированного подхода в программировании на языке JavaScript»** 296

**Лабораторная работа № 28** 311

**Тема работы: «Применение функций временной задержки. Анимации в JavaScript»** 311

**Лабораторная работа № 29** 320

**Тема работы: «Создание приложения с использованием асинхронных событий»** 320

**Лабораторная работа № 30** 325

**Тема работы: «Получение данных в формате JSON и их обработка на стороне клиента»** 325

**Лабораторная работа № 31** 331

**Тема работы: «Подключение библиотеки jQuery к приложению и получение доступа к элементам документа»** 331

**Лабораторная работа № 32** 339

**Тема работы: «Создание анимаций при помощи библиотеки jQuery»** 339

**Лабораторная работа № 33** 351

**Тема работы: «Использование виджетов jQuery UI»** 351

**Лабораторная работа № 34** 356

**Тема работы: «Проверка данных форм при помощи библиотеки jQuery»** 356

# **Лабораторная работа № 1**

## **Тема работы: «Создание простейшего HTML-документа. Физическое и логическое форматирование текста на HTML-странице»**

**1. Цель работы**

Формирование умений использования основных тегов языка HTML, создания структуры web-страниц средствами языка HTML.

**2. Задание**

Создать простейшую HTML-станицу сайта-визитки в соответствии с вариантом.

Вариант 1. Сайт-визитка «Библиотеки».

Вариант 2. Сайт-визитка «Разработки программного обеспечения».

Вариант 3. Сайт-визитка «Грузоперевозкок».

Вариант 4. Сайт-визитка «Поликлиники».

Вариант 5. Сайт-визитка «Службы занятости».

Вариант 6. Сайт-визитка «Магазина».

Вариант 7. Сайт-визитка «Автосервиса».

Вариант 8. Сайт-визитка «Железнодорожной станции».

Вариант 9. Сайт-визитка «Собаковода».

Вариант 10. Сайт-визитка «Издательства».

Вариант 11. Сайт-визитка «Мебельного магазина».

Вариант 12. Сайт-визитка «Туристического агентства».

Вариант 13. Сайт-визитка «Фитнес-центра».

Вариант 14. Сайт-визитка «Картотеки Интерпола».

Вариант 15. Сайт-визитка «Бюро знакомств».

Вариант 16. Сайт-визитка «Биржи труда».

Вариант 17. Сайт-визитка «Записной книжки».

Вариант 18. Сайт-визитка «Гостиницы».

Вариант 19. Сайт-визитка «Музыкальной группы».

Вариант 20. Сайт-визитка «Шеф-повара».

Вариант 21. Сайт-визитка «Справочник лекаря».

Вариант 22. Сайт-визитка «Ломбарда».

Вариант 23. Сайт «Писателя».

Вариант 24. Сайт-визитка «Фанклуба».

Вариант 25. Сайт-визитка «Мир животных».

Вариант 26. Сайт-визитка «Школы».

Вариант 27. Сайт-визитка «Кинотеатра».

Вариант 28. Сайт-визитка «Автоклуба».

Вариант 29. Сайт-визитка «Достопримечательностей Гомеля».

Вариант 30. Сайт-визитка «Мира аниме»

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

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

В HTML регистр символов, определяющих теги, не учитывается.

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

В структуре HTML-документа выделяются заголовок (<HEAD>

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

Для создания HTML-документа нужно выполнить следующую последовательность действий:

* запустить приложение «Блокнот»;
* набрать исходный текст документа, например:

<HTML>

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

<BODY>

Содержимое документа

</BODY>

</HTML>

Сохранить файл с расширением .html или .htm.

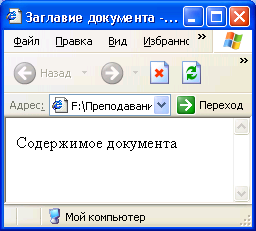
Для просмотра HTML-документа откройте созданный файл в браузере (например, Internet Explorer, Opera). Документ first.html, открытый в браузере Internet Explorer (IE), представлен на рисунке 1.1.

Рисунок 1.1 – Документ first.html в браузере Internet Explorer

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 названия цветов или их обозначения в шестнадцатеричной системе кодирования RGB. Например, следующие строки идентичны:

<BODY BGCOLOR="#FFFFFF">

<BODY BGCOLOR="WHITE">

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

<!-- Это комментарий -->

К тегам физического форматирования относятся:

- <B>полужирный шрифт </B>;

- <I>выделяет курсивом</I>;

- <TT>моноширинный шрифт </TT>;

- <U>подчеркивание текста </U>;

- <STRIKE>зачеркнутый текст </STRIKE>;

- <S>зачеркнутый текст </S>;

- <BIG>Шрифт большего размера </BIG>;

- <SMALL> Шрифт меньшего размера </SMALL>;

- <SUB>шрифт для нижнего индекса </SUB>;

- <SUP> шрифт для верхнего индекса </SUP>;

- <pre>текст в том виде, в котором набран, т.е. со всеми пробелами и переносами строк</pre>;

- <SPAN> является аналогом тега уровня блока <DIV>;

- <P> для указания границ абзаца (выравнивание задается свойством ALIGN) </P>;

- <FONT> указывает параметры шрифта (свойство SIZE определяет размер текста, FACE задает имя шрифта, COLOR задает цвет текста, STYLE позволяет выделить текст цветовым фоном)</FONT>.

Свойство SIZE определяет размер текста, значения размера от 1 (мелкий) до 6 (крупный), по умолчанию SIZE=3. Свойство FACE задает имя шрифта. Можно задавать несколько имен шрифтов через запятую. Если у пользователя не окажется первого шрифта, то браузер будет подставлять второй и т.д. Свойство COLOR задает цвет текста. Можно использовать как имена цветов (red, blue, green) так и номер в виде #RRGGBB, где RR - концентрация красного, GG - зеленого, BB - синего. Выделение текста цветовым фоном достигается использованием свойства STYLE="backgroud-color: цвет".

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html;

charset=windows-1251">

<title>Теги физического форматирования текста</title>

</head>

<body>

<FONT SIZE="4">Размер 4</FONT>

<br>

<FONT FACE="Arial">Arial</FONT>

<FONT COLOR="red">Красный</FONT>

<FONT COLOR="#0000FF">Синий</FONT>

<FONT STYLE="background-color: pink">Текст на розовом фоне</FONT>

<p><font color="#FF0000" face="Arial">Красный текст со шрифтом Arial</font></p>

<pre>

СЛОН.

Дали туфельки слону.

Взял он туфельку одну

И сказал: - Нужны пошире,

И не две, а все четыре!

С. Я. Маршак.

</pre>

<P ALIGN="left"> Выравнивание по левому краю</P>

<P ALIGN="center"> Выравнивание по центру</P>

<P ALIGN="right"> Выравнивание по правому краю</P>

<P ALIGN="justify"> Выравнивание по ширине страницы.

<p><B>полужирный</B></p>

<p><I>курсив</I></p>

</body>

</html>

Теги форматирования могут быть вложенными друг в друга. При этом нужно внимательно следить, чтобы один контейнер находился целиком в другом контейнере.

Основная задача тегов физического форматирования текста это выполнение сугубо декоративных функций.

К тегам логического форматирования относятся:

**- <abbr>**CSS**</abbr>** – **в**ыделяет в тексте аббревиатуру. Браузерами обычно подчеркивается пунктирной линией;

**- <acronym>**комсомол**</acronym> –** выделяет в тексте акроним. *Акроним это почти то же самое что и аббревиатура, только образованная из начальных букв, слов или словосочетаний, произносимая как единое слово, а не побуквенно*;

**- <address>**Вася Пупкин г. Урюпинск Макаронная фабрика 2010г.**</address>** – указывает автора документа и его адрес. Обычно отображается курсивом. *Предназначен для поисковых систем для сбора информации об авторе его адресе и прочей информации владельца/цев сайта;*

**- <cite>**Лед тронулся!**</cite>** – выделяет в тексте цитату или сноску на другой документ. Обычно браузеры отображают её курсивом;

**- <code>**function()**</code>** – используется для отображения фрагментов программного кода. Обычно отображается моноширинным шрифтом;

**- <del>**Старая цена 1000р.**</del>**Новая 999 р.!!! – выделяет удалённый текст в новой версии документа. Выделенный текст станет перечёркнутым;

**- <dfn>**Акроним**</dfn>** – аббревиатура, образованная из начальных букв, слов или словосочетаний, произносимая как единое слово, а не побуквенно. Выделяет текст как определение. *Как правило, когда в тексте встречается новый термин, авторы выделяют его курсивом и дают его определение, собственно для этого и нужен тег* ***<dfn>****;*

**- <del>**Старая цена 1000р.**</del>** **<ins>**Новая 999 р.**</ins>**!!! – выделяет новый текст в новой версии документа. Выделенный текст в большинстве браузеров станет подчёркнутым. *Противоположен по значению тегу* ***<del>*** *с ним же в паре обычно он и используется*;

**- <em>**Как создать сайт?**</em>** – выделяет особенно важный фрагмент текста. Обычно отображается курсивом;

**- <kbd>**Ctrl + Z**</kbd>** – указывает текст, вводимый с клавиатуры, кроме того, используется для названия клавиш. Обычно отображается моноширинным шрифтом;

**- <q>**Лед тронулся!**</q>** – выделяет в тексте цитату. *В отличие от тега <cite>* цитата, обозначенная тегом **<q>**, автоматически берётся браузерами в кавычки;

**- <samp>**4**</samp>** – обозначает текст, который выводится на экран в результате работы какой-либо программы. Обычно браузерами отображается моноширинным шрифтом;

**- <strong>**Как создать сайт?**</strong>** –выделяет особенно важный фрагмент текста. Обычно отображается полужирным;

**- <var>**$count**</var>** – выделяет текст, как переменную в работе какой либо программы. Обычно браузерами отображается курсивом.

Практически ко всем тегам применим атрибут **title** – всплывающая подсказка, поэтому, если в тексте необходимо выделить некую аббревиатуру, желательно давать к ней расшифровку, используя данный атрибут.

Пример:

<abbr title="Cascading Style Sheets">CSS</abbr> позволит Вам без труда изменить стиль любого тега логического форматирования текста!

<acronym title="коммунистический союз молодёжи">комсомол</acronym>

Под логической разметкой документа, прежде всего, принято понимать разметку текста тегами направленную на удобство работы поисковых систем и других программ, которые работают с данным документом, а так же придавать тексту в документе некий смысл. Спецификация HTML не говорит о том, что теги логического форматирования текста должны определённым образом отображаться браузерами, *что вот, например, тот же тег* ***<em>*** *должен быть отображен именно курсивом и не как иначе,* а говорит лишь о том, что такой текст должен отличатся от основного и особым образом выделятся в документе.

Кроме того, теги логического форматирования можно переопределить с использованием [CSS](http://design.originweb.info/css.html).

**5. Порядок выполнения работы**

1. Создайте папку lab1. Наберите текст кода, приведенный на рисунке 2, в текстовом редакторе. Сохраните файла с расширением\*.html в папку lab1. Откройте html-файл с помощью браузера. Убедитесь, что вид файла в окне браузера совпадает с видом на рисунке 1.2.

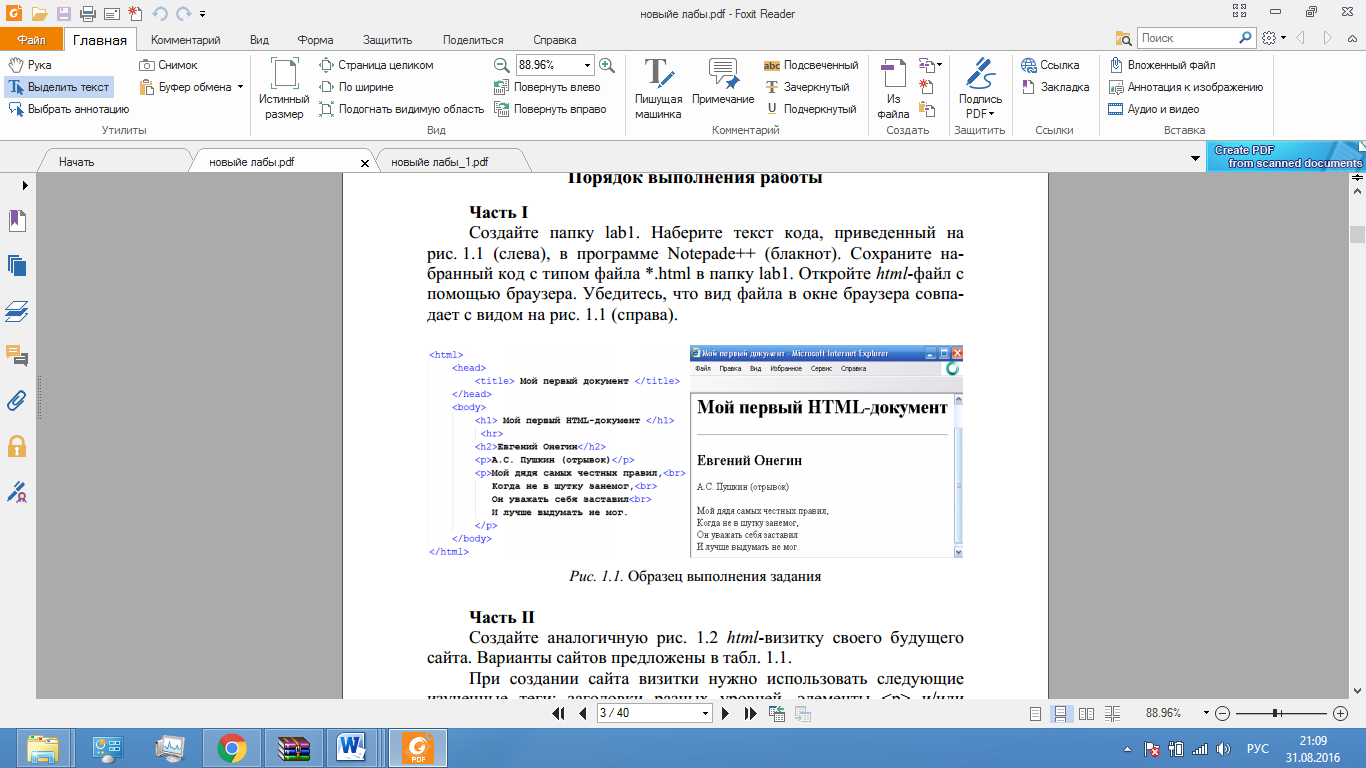


Рисунок 1.2 – Образец выполнения задания

2. Создать сайт-визитку, соответствующую вашему варианту.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1.Опишите структуру html-документа.

2.Каков порядок создания html-документа?

3.Дайте характеристику тегов, используемых в лабораторной работе.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 2**

## **Тема работы: «Организация системы ссылок сайта. Работа с изображениями. Создание списков и таблиц на HTML-странице»**

**1. Цель работы**

Формирование умений организации системы гиперссылок и использования графических изображений в HTML-документах.

**2. Задание**

Создать HTML-станицу в соответствии с вариантом.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Ссылка состоит из двух частей:

* элемент привязки (или якорь – anchor) – место в документе, отмеченное как ссылка. Существуют два типа элементов привязки: текстовый и графический;
* ссылка на URL – сообщает браузеру, какой документ нужно загружать при щелчке на ссылке.

Для организации ссылок в HTML используется тег <A>. Рассмотрим структуру ссылки на конкретном примере, рисунок 2.1.

ссылка на URL

текстовый элемент

привязки



<A HREF=”GoTo.html” > Перейти </A>

содержимое тега-контейнера <A> интерпретируется как ссылка



Рисунок 2.1 – Структура ссылки

Могут использоваться относительные и абсолютные ссылки на URL файла.

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](http://www.yandex.ru/index.html)">Перейти на портал Yandex</A>

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

Пример внутренней ссылки приведен ниже:

<HTML>

<HEAD><TITLE>Внутренние ссылки</TITLE></HEAD>

<BODY>

<H2> Создание внутренних ссылок</H2>

<P>**<a href=#five>Переход к разделу 5</a>**</P>

<BR />

<BR />

<P> Раздел 1

...</P>

<P> Раздел 2

...</P>

<P> **<A name=five>Раздел 5</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 способом, в тексте выделяться 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 следующие атрибуты, представленные в таблице 2.1.

Таблица 2.1 – Атрибуты тега IMG

|  |  |
| --- | --- |
| **Атрибут** | **Назначение** |
| **1** | **2** |
| SRC | Обязательный атрибут, указывает адрес файла с изображением |
| HEIGHT | Определяет ширину изображения |
| **1** | **2** |
| 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 |

Пример добавления изображения pict.jpg на html-страницу приведен ниже, результат – на рисунке 2.2.

<HTML>

<HEAD><TITLE>Teг IMG</TITLE></HEAD>

<BODY>

<H1> Использование тега&lt;IMG&gt;</H1>

<IMG src="pict.JPG" width="160" height="192" border="2"

alt="картинка" align="right" hspace="20" vspace="30">

</BODY>

</HTML>

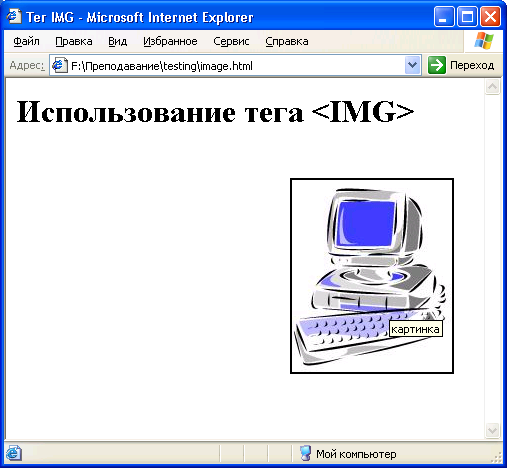


Рисунок 2.2 – Вставка изображения на html-страницу

Таблица в HTML имеет следующий вид, представленный на рисунке2.3.

Рисунок 2.3 – Таблица в HTML

<TABLE>

<TR>

**ОПРЕДЕЛЕНИЕ ТАБЛИЦЫ**

**СТРОКА ТАБЛИЦЫ**

<TD> (<TH>)

ТЕКСТ, ИЗОБРАЖЕНИЕ, HTML-ТЕГИ

</TD> (</TH>)

**ЯЧЕЙКА ТАБЛИЦЫ**

</TR>

</TABLE>

Атрибуты тега <TABLE> приведены в таблице 2.2.

Таблица 2.2 – Атрибуты тега TABLE

|  |  |
| --- | --- |
| **Атрибут** | **Назначение** |
| ALIGN | Определяет способ горизонтального выравнивания таблицы (LEFT, RIGHT, CENTER) |
| VALIGN | Определяет способ вертикального выравнивания таб- лицы (TOP, MIDDLE,BOTTOM) |
| 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 |

Ниже приведен пример создания простейшей таблицы, состоящей из двух строк и двух столбцов, результат представлен на рисунке 2.4.

<HTML>

<HEAD><TITLE>Пример таблицы</TITLE></HEAD>

<BODY>

<TABLE width=50% align=center bgcolor=#a0dda0>

<TR>

<TD>Ячейка 1</TD>

<TD>Ячейка 2</TD>

</TR>

<TR>

<TD>Ячейка 3</TD>

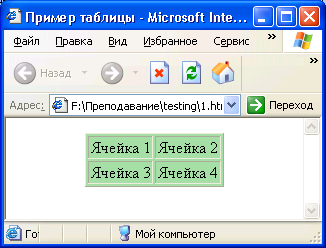
<TD>Ячейка 4</TD>

</TR>

</TABLE>

</BODY>

</HTML>

Рисунок 2.4 – Отображение созданной таблицы в Internet Explorer

Атрибуты тега <TR> приведены в таблице 2.3

Таблица 2.3 – Атрибуты тега tr

|  |  |
| --- | --- |
| **Атрибут** | **Назначение** |
| ALIGN | Определяет способ горизонтального выравнивания содержимо- го всех ячеек данного ряда (LEFT, RIGHT, CENTER, JUSTIFY) |
| VALIGN | Определяет способ вертикального выравнивания содержи- мого всех ячеек данного ряда |
| BGCOLOR | Определяет цвет фона для всех ячеек данного ряда |

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

Атрибуты тегов <TD> и <TH> приведены в таблице 2.4

Таблица 2.4 – Атрибуты тега TD и TH

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

Пример использования атрибутов COLSPAN и ROWSPAN приведен ниже, результат представлен на рисунке 2.5.

<TABLE border=1>

<TR>

<TD colspan="2">Элемент1</TD>

</TR>

<TR>

<TD>Элемент 2</TD>

<TD rowspan="2">Элемент 3</TD>

</TR>

<TR>

<TD>Элемент 4</TD>

</TR>

</TABLE>

|  |  |
| --- | --- |
| Элемент 1 | |
| Элемент 2 | Элемент 3 |
| Элемент 4 |

Рисунок 2.5 – Объединение строк и столбцов таблицы

Существуют следующие виды списков:

* упорядоченные (пронумерованные);
* неупорядоченные (непронумерованные);
* списки определений.

Для создания списков используются следующие теги, представленные в таблице 2.5.

Таблица 2.5 – Атрибуты для создания списков

|  |  |
| --- | --- |
| **Тег** | **Назначение** |
| <UL> | Создает неупорядоченный список |
| <OL> | Создает упорядоченный список |
| <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 <OL> приведены в таблице 2.6.

Таблица 2.6 – Атрибуты тега <OL>

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

Значения атрибута TYPE для упорядоченного списка могут быть следующими:

* 1 арабские 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 может использоваться атрибут TYPE (со значениями DISC, SQUARE или CIRCLE).

Для изменения порядка нумерации элементов списка в теге LI используется атрибут VALUE.

Пример создания списков различных видов приведен ниже, результат представлен на рисунке 2.6.

<HTML>

<HEAD><TITLE>Создание списков</TITLE></HEAD>

<BODY>

<B>Ненумерованный список</B>

<UL type=square>

<LI >Элемент 1</LI >

<LI>Элемент 2</LI >

<LI>Элемент 3</LI >

<LI>Элемент 4</LI >

</UL>

<B>Нумерованный список</B>

<OL type=I start=3>

<LI>Элемент 1</LI >

<LI>Элемент 2</LI >

<LI value=1>Элемент 3</LI >

<LI>Элемент 4</LI >

</OL>

<B>Список определений</B>

<DL>

<DT>Термин1</DT>

<DD>Определение1</DD>

<DT>Термин2</DT>

<DD>Определение2</DD>

</DL>

</BODY>

</HTML>

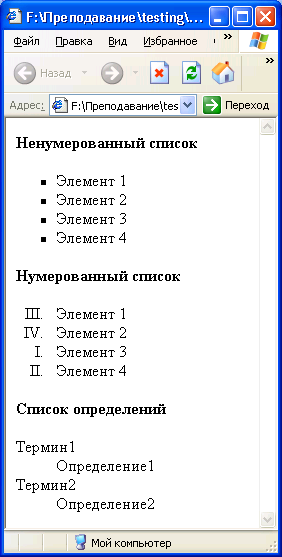


Рисунок 2.6 – Использование списков на HTML-странице

**5. Порядок выполнения работы**

1. Создайте документ 1.html, содержащий перечень рабочих дней недели.
2. Установите название документа, цвет фона и текста.
3. Создайте документ 2.html, содержащий расписание занятий на неделю.
4. Установите название документа 2.html, цвет фона и текста.
5. Организуйте систему ссылок таким образом, чтобы при выборе определенного дня недели в документе 1.html осуществлялся переход к расписанию занятий данного дня в документе 2.html.
6. Добавьте в начало документа 2.html ссылки на расписание определенного дня недели.
7. Установите цвет ссылок для документов1.html и 2.html.
8. Вставьте изображение на страницу 3.html. следующим образом, представленным на рисунке 2.7.

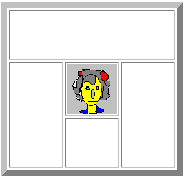


Рисунок 2.7 – Изображение в HTML-таблице

1. Установите размер изображения, ширину рамки, выравнивание, отступ по горизонтали и вертикали, а также альтернативный текст.
2. Вставьте изображение на страницу 1.html
3. Используйте добавленное изображение в качестве графического элемента привязки ссылки на документ 2.html.
4. Создать HTML-страницу, соответствующую вашему варианту.

Цветовое оформление и выбор изображение выбирает учащийся!

**1 вариант**: создать таблицу шириной 800px, как показано на рисунке 2.8, центрировать ее относительно экрана.

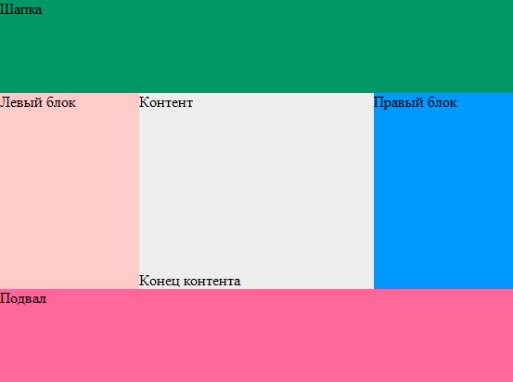


Рисунок 2.8 – Внешний вид таблицы

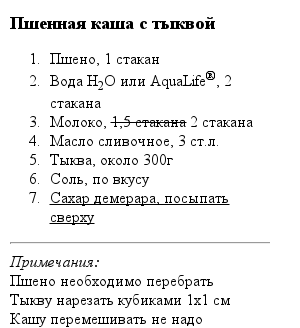
В область «Контент» поместить текст и оформить его только средствами HTML. Текст представлен на рисунке 2.9.

Рисунок 2.9 – Текст для выполнения задания 1 варианта

**2 вариант**: создать таблицу шириной 800px, как показано на рисунке 2.10, центрировать ее относительно экрана.

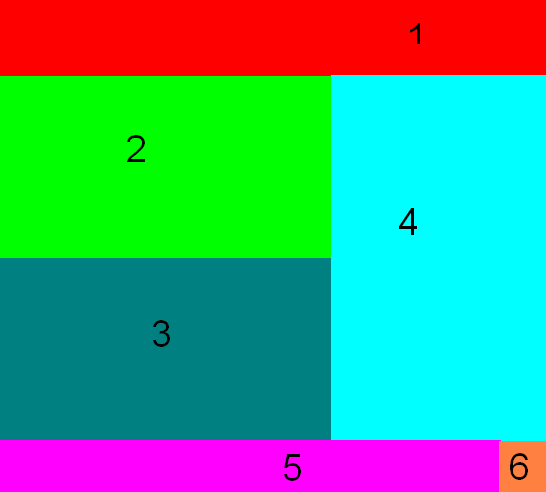


Рисунок 2.10 – Внешний вид таблицы

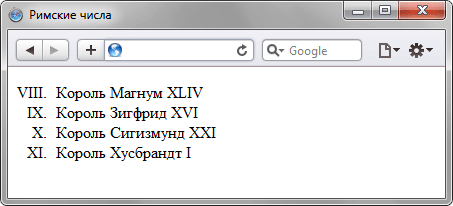
В области под номерами 2, 3 и 4 поместить текст и оформить его только средствами HTML (располагать текст можно в любом порядке, сохраняя его внешний вид). Текст следующий приведен на рисунке 2.11.

Рисунок 2.11 – Текст для выполнения задания 2 варианта

**3 вариант**: Создать таблицу произвольной ширины (условие: ширина = высоте) в виде шахматной доски, как показано на рисунке 2.12.

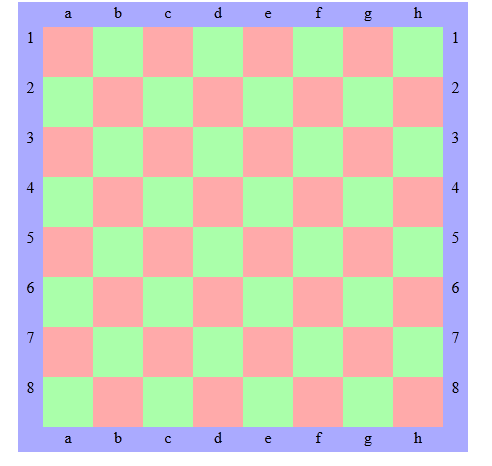


Рисунок 12 – Внешний вид таблицы

Создать список, представленный на рисунке 2.13.

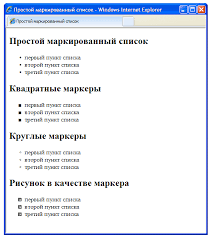


Рисунок 2.13 – Внешний вид списков

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1.Опишите структуру html-документа.

2.Каков порядок создания html-документа?

3.Дайте характеристику тегов, используемых в лабораторной работе.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 3**

## **Тема работы: «Использование слоев и изображений-карт на HTML-странице»**

**1. Цель работы**

Формирование умений использования слоев и изображений-карт на HTML-странице.

**2. Задание**

Создать HTML-станицу, содержащую изображение-карту и слои.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Для создания слоев может использоваться тег <LAYER>. Однако данный тег поддерживается только браузером Netscape Navigator. Более универсальным способом создания слоев является сочетание использования тега <DIV> и каскадных таблиц стилей (CSS).

Шаблон HTML-кода слоя выглядит следующим образом:

<DIV STYLE=”Свойства слоя”> Содержимое слоя</DIV>

В контейнере <DIV> расположены HTML-теги, определяющие элементы, из которых состоит слой.

Свойства слоя записываются следующим образом:

свойство:значение

Можно определить следующие свойства слоя, представленные в таблице 3.1.

Таблица 3.1 – Атрибуты тега div

|  |  |
| --- | --- |
| **Свойство** | **Назначение** |
| position: ***absolute| static| relative*** | Точка начала отсчета координат |
| top: ***число*** | Y-координата верхнего левого угла слоя |
| left: ***число*** | X-координата верхнего левого угла слоя |
| width: ***число*** | Ширина слоя |
| height: ***число*** | Высота слоя |
| color: ***цвет*** | Цвет текста |
| background: ***цвет*** | Цвет фона слоя |
| background-image: ***url(путь)*** | Фоновое изображение слоя |
| visibility: ***visible| hidden*** | Первоначальная видимость слоя |
| z-index: ***число*** | Порядок отображения слоев |

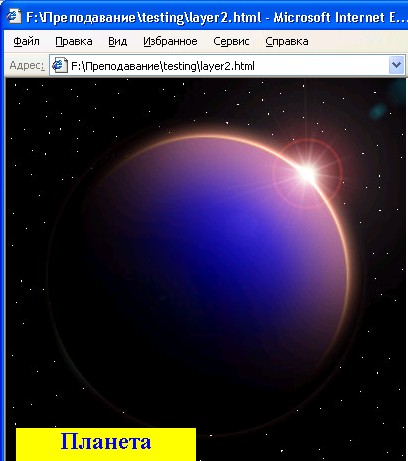
Пример создания слоев приведен ниже, результат – на рисунке 3.1.

<DIV style="position:absolute; top:350; left:10; width:180; height:50; background:yellow; color:blue; z-index:1">

<H2 align="center">Планета</H2>

</DIV>

<DIV style="position:absolute; top:0; left:0; z-index:0">

<IMG src="Planet.jpg" width="400" height="400" border="0">

</DIV>

Рисунок 3.1 – Слои на HTML-странице

Как и при использовании других блочных элементов, содержимое тега <div> всегда начинается с новой строки. После него также добавляется перенос строки:

- **align** задает выравнивание содержимого тега <div>;

- [**title**](http://htmlbook.ru/html/div/title) добавляет всплывающую подсказку к содержимому.

Абсолютное позиционирование – самый простой и менее гибкий  способ создания наложения.

Абсолютное позиционирование позволяет накладывать слои в любом порядке друг на друга. Но при этом требуется знать точные координаты каждого слоя относительно одного из углов окна браузера, что не всегда возможно.

При использовании наложения требуется присвоить свойству **position** значение **absolute**. Само положение слоя регулируется свойствами **left**, **top**, **right** и **bottom**,которые задают координаты соответственно от левого, верхнего, правого и нижнего края.

Более интересный подход к созданию наложения – использование относительного позиционирования. В этом случае слои можно размещать по центру окна браузера или располагать их в любом месте веб-страницы, не задумываясь уже над значением координат слоев.

Относительное позиционирование. Чтобы наложить один слой на другой и не привязываться жестко к координатной сетке, можно попробовать следующий подход. Для первого слоя, который будет располагаться на заднем плане, указываем абсолютное позиционирование, присваивая свойству position значение absolute. Второй слой, расположенный поверх первого, должен иметь относительное позиционирование, что достигается с помощью значения relative у свойства position. Положение верхнего слоя определяется от левого верхнего угла нижнего слоя заданием left и top (на рисунке задание положения верхнего слоя).

Пример представлен на рисунке 3.2.

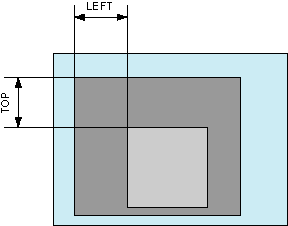


Рисунок 3.2 – Относительное позиционирование

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset="windows-1251">

<title>Позиционирование</title>

<style type="text/css">

#leftcol {

position: absolute; /\* Абсолютное позиционирование \*/

width: 550px; /\* Ширина левой колонки \*/

background: #e0e0e0; /\* Цвет фона содержимого \*/

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

}

#rightcol { /\* Этот слой накладывается поверх \*/

position: relative; /\* Относительное позиционирование \*/

left: 500px; /\* Положение от левого края \*/

top: 20px; /\* Положение от верхнего края \*/

width: 200px; /\* Ширина правой колонки \*/

background: #800000; /\* Цвет фона \*/

color: #fff; /\* Цвет текста \*/

}

</style>

</head>

<body>

<div id="leftcol">Левая колонка</div>

<div id="rightcol">Правая колонка</div>

</body>

</html>

Результат представлен на рисунке 3.3.

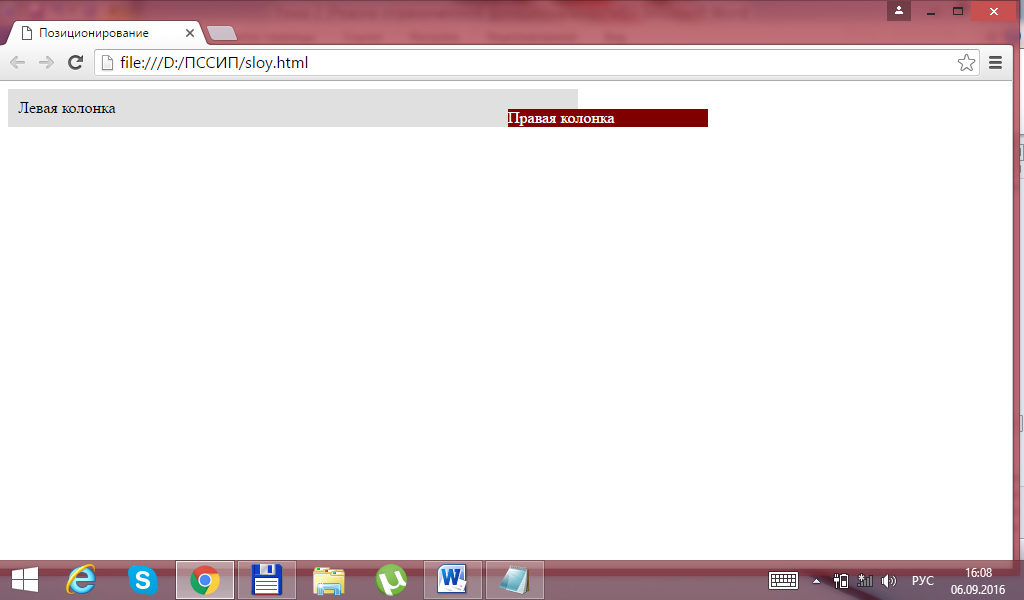


Рисунок 3.3 – Результат положения относительного позиционирования

<!DOCTYPE HTML>

<html>

<head>

<meta charset=" windows-1251 ">

<title>Тег DIV</title>

<style type="text/css">

.block1 {

width: 200px;

background: #ccc;

padding: 5px;

padding-right: 20px;

border: solid 1px black;

float: left;

}

.block2 {

width: 200px;

background: #fc0;

padding: 5px;

border: solid 1px black;

float: left;

position: relative;

top: 40px;

left: -70px;

}

</style>

</head>

<body>

<div class="block1">Lorem ipsum dolor sit amet, consectetuer

adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet

dolore magna aliguam erat volutpat.</div>

<div class="block2">Ut wisis enim ad minim veniam, quis nostrud

exerci tution ullamcorper suscipit lobortis nisl ut aliquip ex

ea commodo consequat.</div>

</body>

</html>

Результат кода представлен на рисунке 3.4.

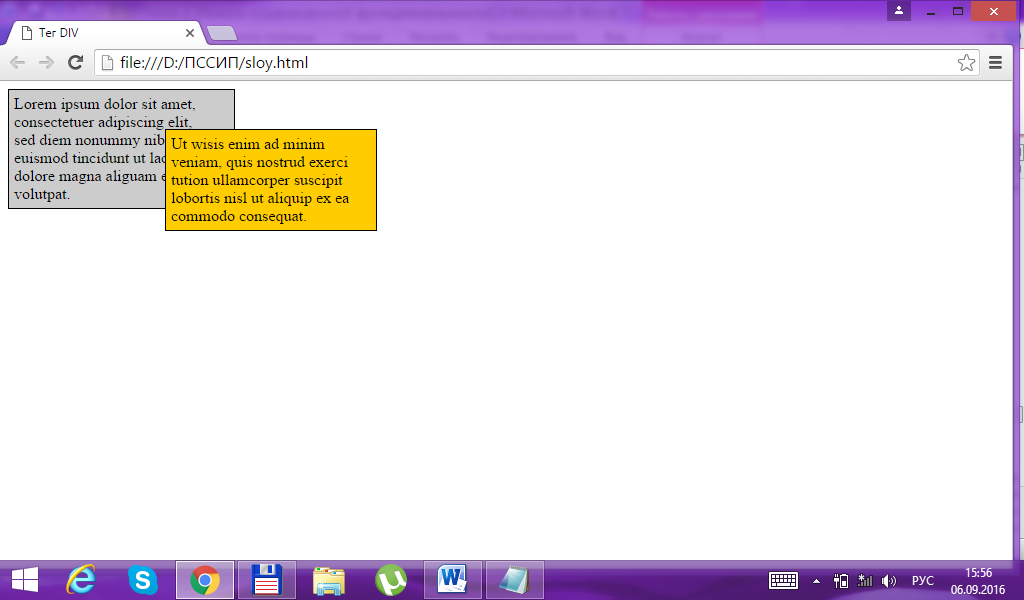


Рисунок 3.4 – Пример реализации

Существует и другой способ наложения слоев, который связан с относительным позиционированием и использует все разнообразие средств размещения разных слоев.

Теоретически, можно разместить слои по вертикали друг под другом и поднять нижний слой вверх с помощью свойства top, задавая ему отрицательное значение или с помощью bottom. На практике добиться подобного довольно сложно, ведь определить высоту слоя простыми средствами, а, следовательно, и величину, на которую следует смещать слой, не представляется возможным, поскольку она зависит от размера шрифта, содержимого слоя и многих других параметров. Проще отсчет координат вести от верхнего угла одного из слоев. Для этого следует разместить слои рядом по вертикали, а потом уже смещать один слой относительно другого.

Реализуется это следующим способом. У каждого слоя необходимо указать конструкцию float: left, которая позволяет один слой пристыковать к другому справа. Добавлять float следует для каждого слоя, иначе в некоторых браузерах появится промежуток между слоями.

Теперь слои располагаются рядом, и прежде, чем указывать координаты, задаем относительное позиционирование значением relative у свойства position. Положение верхнего слоя управляется значением left и top. Но поскольку отсчет координат в данном случае ведется от левого верхнего угла второго слоя, по горизонтали нужно указывать отрицательное значение (рисунок задание положения верхнего слоя). Впрочем, можно использовать также свойство bottom. Пример реализации свойства position представлен на рисунке 3.5.

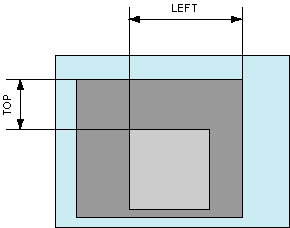


Рисунок 3.5 – Реализация свойства position

Пример:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset= windows-1251">

<title>Позиционирование</title>

<style type="text/css">

#leftcol {

position: relative; /\* Относительное позиционирование \*/

float: left; /\* Совмещение колонок по горизонтали \*/

width: 400px; /\* Ширина слоя \*/

background: #800000; /\* Цвет фона \*/

color: white; /\* Цвет текста \*/

}

#rightcol {

position: relative; /\* Относительное позиционирование \*/

float: left; /\* Совмещение колонок по горизонтали \*/

left: -50px; /\* Сдвиг слоя влево \*/

top: 50px; /\* Смещение слоя вниз \*/

width: 120px; /\* Ширина слоя \*/

background: #e0e0e0; /\* Цвет фона \*/

border: solid 1px black; /\* Параметры рамки \*/

}

#leftcol P {

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

padding-right: 50px; /\* Значение поля справа \*/

margin: 0; /\* Обнуляем значения отступов \*/

}

#rightcol P {

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

padding-top: 0 /\* Значение поля сверху \*/

}

</style>

</head>

<body>

<div id="leftcol">

<p>Левая колонка </p>

</div>

<div id="rightcol">

<p>Правая колонка</p>

</div>

</body>

</html>

Пример реализации представленного выше кода представлен на рисунке 3.6.

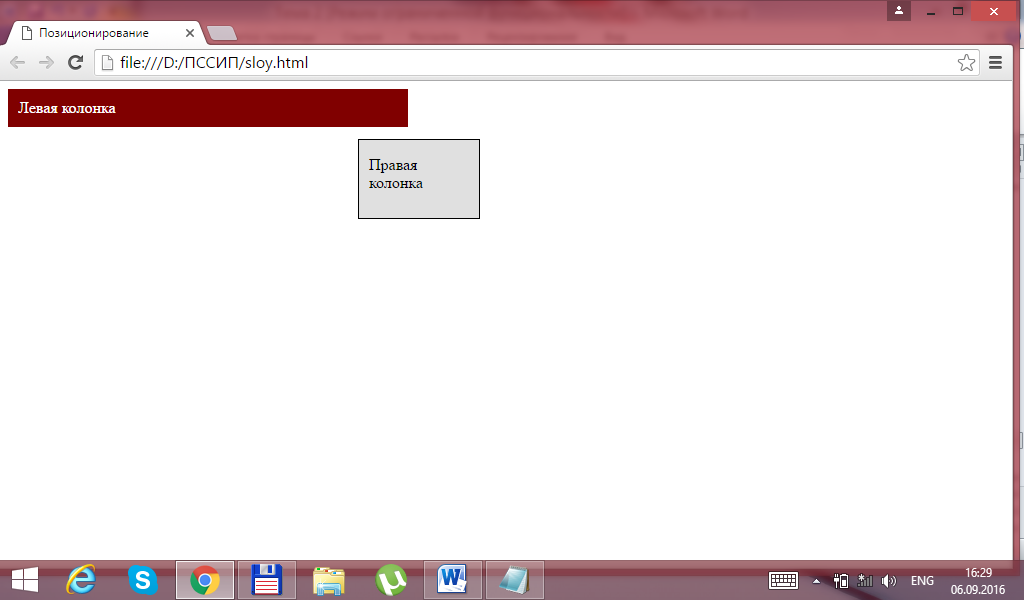


Рисунок 3.6 – Пример реализации кода

Изображения-карты. Различают два вида изображений-карт (карт ссылок):

- обрабатываемые сервером;

- обрабатываемые клиентом.

Для определения изображения-карты, обрабатываемой клиентом, используются теги <MAP> и <AREA>, рисунок 3.7.

<MAP NAME=MAP1>

СОЗДАНИЕ КАРТЫ ССЫЛОК

</MAP>

<AREA …> СОЗДАНИЕ ОБЛАСТИ

<AREA …> СОЗДАНИЕ ОБЛАСТИ

<AREA …> СОЗДАНИЕ ОБЛАСТИ

Рисунок 3.7 – Структура карты ссылок

Атрибуты тега <AREA> [8] приведены в таблице 3.2.

Таблица 3.2 – Атрибуты тега area

|  |  |
| --- | --- |
| **Атрибут** | **Назначение** |
| SHAPE | Форма области (rect, circle, poly) |
| COORDS | Координаты области на карте |
| HREF | URL, на который ссылается область |
| NOHREF | Определяет область как неактивную (является атрибутом-флагом) |
| ALT | Альтернативный текст-подсказка для данной области |

При добавлении изображения, являющегося картой ссылок, используется атрибут USEMAP, значением которого является имя карты, указанное в теге <MAP>. Перед именем карты ставится символ #.

Например:

<!DOCTYPE HTML>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset= windows-1251">

<title>Карта-изображение</title>

</head>

<body>

<p><img src="images/map.gif" width="411" height="46" **usemap="#map"** alt="Навигация"></p>

<p><map name="map"> <area shape="poly" alt="Закладка 2"

 coords="210,27, 203,9, 202,6, 197,2, 192,1, 120,1, 115,2, 110,6, 112,9, 119,27, 119,32, 211,32, 210,27"

 href="2.html">

<area shape="poly" alt="Закладка 3"

 coords="302,27, 295,9, 293,6, 289,2, 283,1, 212,1, 206,2, 202,6, 203,9, 210,27, 211,32, 284,32, 303,32, 302,27" href="3.html">

<area shape="poly" alt="Закладка 4"

 coords="302,27, 303,32, 394,32, 393,27, 386,9, 382,3, 375,1, 303,1, 298,2, 293,6, 295,9, 302,27"

 href="4.html">

</map></p>

</body>

</html>

Пример определения и использования карты ссылок приведен ниже, результат – на рисунке 3.8.

<MAP NAME="rectangles">

<AREA shape="rect" coords="95,55,295,185" alt="orange" href="2.html">

<AREA shape="rect" coords="0,0,175,120" alt="yellow" href="1.html">

</MAP>

<IMG src="Kard.gif" border="0" USEMAP="#rectangles">

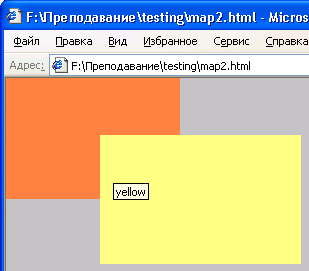


Рисунок 3.8 – Созданная карта ссылок

Атрибут **nonhref** используется для того, чтобы указать, что область не является активной. Например, его можно использовать, чтобы сделать активной область в виде кольца.

<img src="fig2.gif" border="0" usemap="#Map2">

<map name="Map2">

<area shape="circle" coords="133,116,59" alt="Нет ссылки (nonhref)" nonhref>

<area shape="circle" coords="133,117,89" href="fromimagemap1.htm" alt="На Документ 1">

</map>

Карты-изображения и их исходные коды. Прямоугольник:

<img src="Прямоугольник.gif" usemap="#Прямоугольник" alt="">

<map id="Прямоугольник" name="Прямоугольник">

<area shape="rect" coords="63,18,172,168" href="A:\HTML\Навигационные карты.пример.html"title="Прямоугольник" target="blank" >

<area shape="default" nohref="nohref" alt="" >

</map>

Многоугольник, представленный на рисунке 3.9:

<img src="Многоугольник.gif" usemap="#Многоугольник" alt="">

<map id="Многоугольник" name="Многоугольник">

<area shape="poly" coords="65,153,87,32,120,25,166,48,119,103,164,151,110,174" href="Навигационные%20карты.пример.html" title="Многоугольник" target="blank" >

<area shape="default" nohref="nohref" alt="" >

</map>

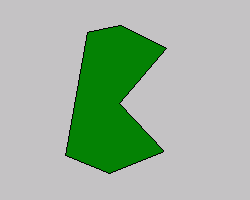


Рисунок 3.9 –Многоугольник

Круг:

<img src="Круг.gif" usemap="#Круг" alt="">

<map name=” Круг”>

<area shape="circle" coords="126,99,73" href="Навигационные%20карты. пример.html" title="круг" target="blank" >

<area shape="default" nohref="nohref" alt="" >

</map>

Кольцо, представленное на рисунке 3.10:

<img src="Кольцо.gif" usemap="#Кольцо" alt="">

<map id="Кольцо" name="Кольцо">

<area shape="circle" coords="128,96,66" nohref="nohref" title="кольцо">

<area shape="circle" coords="128,96,70" href="Навигационные%20карты.

пример.html" title="" target="blank">

<area shape="default" nohref="nohref" alt="">

</map>

Рисунок 3.10 – Кольцо

Карты-изображения позволяют создавать ссылки на разные области одного изображения.

**5. Порядок выполнения работы**

1. Создайте документ html содержащий карту-изображение улицы на которой вы проживаете.
2. Создайте карту-ссылку на дом, в котором вы проживаете и на три прилегающие улицы. Сделать эти области кликабельными. При нажатии на область в новой вкладке должна открываться информация об улице соответственно или номере дома.
3. Создайте еще один документ html содержащем 2 изображения и надпись.
4. Используя правила наложения слоев отобразите данные.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Перечислите способы создания слоев. В чем их отличие?

2. Как определяется порядок наложения слоев?

3. Какие значения может принимать свойство position? В чем их отличия?

4. Дайте определение понятия «изображение-карта»

5. Как создаются изображения-карты?

6. Перечислите атрибуты тега <AREA>.

7. Какой атрибут указывает, что изображение является картой ссылок?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 4**

## **Тема работы: «Создание форм на HTML-странице и изменение свойств элементов формы»**

**1. Цель работы**

Формирование умений создания форм и элементов форм на HTML-странице.

**2. Задание**

Создать HTML-станицу, содержащую форму регистрации.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Формы являются одним из важных элементов любого сайта и предназначены для обмена данными между пользователем и сервером. Область применения форм не ограничена отправкой данных на сервер, с помощью клиентских скриптов можно получить доступ к любому элементу формы, изменять его и применять по своему усмотрению.

Элементы форм представлены в таблице 4.1.

Таблица 4.1. – Элементы форм

|  |  |  |
| --- | --- | --- |
| **Название** | **Описание** | **Пример** |
| **1** | **2** | **3** |
| Однострочное текстовое поле | Предназначено для ввода строки символов с помощью клавиатуры. |  |
| Поле для пароля | Обычное текстовое поле, но отличается тем, что все символы отображаются звездочками. Предназначено для того, чтобы никто не подглядел вводимый пароль. |  |
| Многострочный текст | Область, в которой можно вводить несколько строк текста. |  |
| Кнопка | Элемент формы, на который нужно нажимать. |  |
| Кнопка SUBMIT | После нажатия на эту кнопку данные формы отправляются на сервер и обрабатываются программой, указанной параметромaction тега **FORM**. |  |
| Кнопка RESET | При нажатии на эту кнопку, данные формы возвращаются в первоначальное значение. |  |
| **1** | **2** | **3** |
| Переключатели (radiobutton) | Используются для выбора только одного варианта из предложенных. |  |
| Флажки (checkbox) | Используются для выбора одного и более вариантов из предложенных. |  |
| Поле со списком | Предназначен для выбора одного или нескольких значений из списка. |  |
| Скрытое поле | Скрытое поле не отображается на странице и прячет свое содержимое от пользователя. |  |
| Поле с изображением | Аналогично по действию кнопке Submit, но представляют собой рисунок. |  |
| Отправка файла | Создает поле для ввода имени файла, который пересылается на сервер. |  |

Для создания форм используется парный тег <FORM></FORM>.

Тег <FORM> может содержать следующие атрибуты, представленные в таблице 4.2.

Таблица 4.2 – 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 |

Документ может содержать любое количество форм, но одновременно на сервер может быть отправлена только одна форма. По этой причине данные форм должны быть независимы друг от друга.

Когда форма отправляется на сервер, управление данными передается программе, заданной атрибутом **action** тега **<form>**.

Если для отправки данных используется метод **GET**, то адресная строка может принимать следующий вид:

<http://www.htmlbook.ru/cgi-bin/handler.cgi?nick=%C2%E0%ED%FF+%D8%E0%EF%EE%F7%EA%E8%ED&page=5>

Если управляющие элементы указаны вне содержимого тега FORM, то они не создают форму, а используются для построения пользовательского интерфейса на веб-странице, то есть для привнесения в нее различных кнопок, флажков, полей ввода. Имена элементам формы присваиваются через их атрибут NAME.

Сама форма обычно предназначена для получения от пользователя информации для дальнейшей пересылки её на сервер, где данные формы принимает программа-обработчик. Такая программа может быть написана на любом серверном языке программирования вроде PHP, Perl и др. Адрес программы указывается в атрибуте action тега <form>.

Пример:

<!DOCTYPE HTML>

<html>

<head>

<meta charset=" windows-1251">

<title>Данные формы</title>

</head>

<body>

<form action="/example/handler.php">

<p><input name="login"> <input type="password" name="pass"></p>

<p><input type="submit"></p>

</form>

</body>

</html>

Допускается использовать несколько форм на странице, но они не должны вкладываться одна в другую.

Передача на сервер происходит двумя разными методами: GET и POST, для задания метода в теге <form> используется атрибут method, а его значениями выступают ключевые слова get и post. Если атрибут method не задан, то по умолчанию данные отправляются на сервер методом GET. В таблице 4.3 показаны различия между этими методами.

Таблица 4.3 – Различия между методами GET и POST

|  |  |  |
| --- | --- | --- |
| **Характеристики** | **GET** | **POST** |
| **1** | **2** | **3** |
| Ограничение на объём | 4 Кб | Ограничения задаются сервером. |
| Передаваемые данные | Видны сразу всем. | Видны только при просмотре через расширения браузера или другими методами. |
| Кэширование | Страницы с разными запросами считаются различными, их можно кэшировать как отдельные документы. | Страница всегда одна. |
| **1** | **2** | **3** |
| Закладки | Страницу с запросом можно добавить в закладки браузера и обратиться к ней позже. | Страницы с разными запросами имеют один адрес, запрос повторить нельзя. |

Какой метод используется легко определить по адресной строке браузера. Если в ней появился вопросительный знак и адрес стал похож на этот, то это точно GET:

http://www.google.ru/search?q=%D1%81%D0%B8%D1%81%D1%8C%D0%BA%D0%B8&ie=utf-8

Ниже перечислены типовые области применения этих методов на сайтах.

**GET.** Передача небольших текстовых данных на сервер, поиск по сайту: поисковые системы, формы поиска по сайту всегда отправляются методом GET, это позволяет делиться результатами поиска с друзьями, слать ссылку по почте или выкладывать её на форуме.

**POST.** Пересылка файлов (фотографий, архивов, программ и др.); отправка комментариев; добавление и редактирование сообщений на форуме, блоге. Работа с формой по умолчанию происходит в текущей вкладке браузера, при этом допустимо при отправке формы изменить этот параметр и открывать обработчик формы в новой вкладке или во фрейме. Такое поведение задаётся через «имя контекста», которое выступает значением атрибута target тега <form>. Популярные значения это \_blank для открытия формы в новом окне или вкладке, и имя фрейма, которое задаётся атрибутом name тега <iframe>.

Пример (открытие формы во фрейме):

<!DOCTYPE html>

<html>

<head>

<meta charset="windows-1251">

<title>Использование фрейма</title>

</head>

<body>

<p><iframe name="area" width="500" height="200"></iframe></p>

<form action="handler.php" target="area">

<p><input placeholder="Введите текст" name="text">

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

</form>

</body>

</html>

В HTML5 есть возможность отделить форму от её элементов. Связь между формой и её элементами происходит в таком случае через идентификатор формы, а к элементам следует добавить атрибут form со значением, равным этому идентификатору.

Пример (связывание формы с полями):

<!DOCTYPE html>

<html>

<head>

<meta charset=" windows-1251">

<title>Форма</title>

</head>

<body>

<form id="auth" action="handler.php" method="post"></form>

<p>...</p>

<p><input name="login" form="auth">

<input type="password" name="pass" form="auth"></p>

<p><input type="submit" form="auth"></p>

</body>

</html>

Для определения элементов формы могут использоваться следующие теги, представленные в таблице 4.4.

Таблица 4.4 – Теги для определения элементов формы

|  |  |
| --- | --- |
| **Тег** | **Назначение** |
| <TEXTAREA> </TEXTAREA> | Многострочное текстовое поле ввода |
| <INPUT> | Поля ввода |
| <SELECT> </SELECT> | Меню-список |

Тег <TEXTAREA> имеет следующие атрибуты, представленные в таблице 4.5.

Таблица 4.5 – Атрибуты тега Textarea

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

Тег <INPUT> имеет следующие атрибуты, представленные в таблице 4.6

Таблица 4.6 – Атрибуты тега Input

|  |  |
| --- | --- |
| **Атрибут** | **Назначение** |
| CHECKED | Элемент формы CHECKBOX или RADIO будет отмечен |
| 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 | Устанавливает текст по умолчанию для поля ввода текста или пароля. Для флажка или переключателя указывает значение, возвращаемое серверу в случае выбора флажка или переключателя. Для кнопок определяет надпись |
| TYPE | Определяет тип поля ввода; по умолчанию создается однострочное текстовое поле ввода |

Возможные значения атрибута TYPE представлены в таблице 4.7.

Таблица 4.7 – Значения атрибута Type

|  |  |
| --- | --- |
| **Атрибут** | **Назначение** |
| CHECKBOX | Флажок; может принимать значение ON (отмечен) или  OFF(не отмечен) |
| 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 вари- анта из нескольких)\* |
| RESET | Кнопка, при нажатии на которую поля формы принимают значения по умолчанию |
| SUBMIT | Кнопка отправки данных |
| BUTTON | Кнопка, определенная пользователем (т. е. конкретного действия за данной кнопкой не закреплено, оно задается пользователем) |
| FILE | Поле ввода и кнопка «Обзор» для поиска файла на диске |

Тег <SELECT> имеет следующие атрибуты, представленные в таблице 4.8.

Таблица 4.8 – 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 выбирать пользователь. Они перечисляются с помощью тега <OPTION>.

Тег <OPTION> имеет следующие атрибуты, представленные в таблице 4.9.

Таблица 4.9 – Атрибуты тега Option

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

Пример создания формы в документе form.html приведен ниже, результат – на рисунке 4.1.

<HTML>

<HEAD> <TITLE> Анкета </TITLE> </HEAD>

<BODY>

<FORM action=["http://question.by/result.cgi](http://question.by/result.cgi)">

Имя <BR>

<INPUT type="text" name="name" size="20"><BR> Фамилия <BR>

<INPUT type="text" name="surname" size="20">

<BR><BR>

Пол <INPUT type="radio" name="pol"

value="m" checked> &nbsp;&nbsp;

<INPUT type="radio" name="pol" value="w"><BR><BR> Ваши увлечения:<BR>

<INPUT type="checkbox" name="choice1" value="1">

Спорт<BR>

<INPUT type="checkbox" name="choice2" value="2">

Программирование<BR>

<INPUT type="checkbox" name="choice3" value="3">

Вышивание<BR>

<INPUT type="checkbox" name="choice4" value="4">

Чтение<BR>

<INPUT type="checkbox" name="choice5" value="5">

Прочее

<INPUT type="text" name="hobby" size="20">

<BR><BR>

<INPUT type="submit" name="send"

value="Отправить данные">

</FORM>

</BODY>

</HTML>

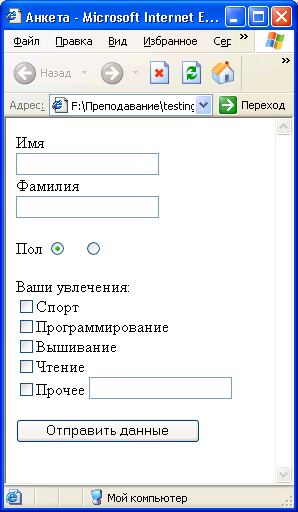


Рисунок 4.1 – Документ form.html в Internet Explorer

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

**5. 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 формы;
* многострочное текстовое поле для ввода примечания.

1. Установите надписи на кнопках.
2. В поле для примечания укажите текст по умолчанию.
3. Для полей ввода имени и фамилии укажите размер поля и максимальное количество вводимых символов.
4. Добавьте на страницу графическое изображение, используя значение IMAGE атрибута TYPE.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Перечислите атрибуты тега <FORM>.

2. Перечислите известные вам теги определения отдельных элементов формы.

3. Какие атрибуты тега <TEXTAREA> вы знаете?

4. Как можно создать выпадающий список значений?

5. Для чего используется атрибут MULTIPLE?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 5**

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

**1. Цель работы**

Формирование умений подключения CSS различными способами и использование его свойств.

**2. Задание**

Создать HTML-станицу содержащую ссылку на внешний CSS-документ.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

CSS (каскадные таблицы стилей) управляют 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 содержимое.

Существуют четыре способа применения таблиц стилей к документу:

- связывание;

- встраивание;

- оперативное определение;

- 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=STYLESHEET

HREF="http://www.myserver.com/mysheet.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 размещается 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 STYLE тега, для которого определяются параметры оформления.

Например:

<Н1 STYLE="color: blue">

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

Например:

<STYLE TYPE="text/css">

@import url(http://www.myserver.com/style.сss);

</STYLE>

Каждое определение стилей называется правилом (rule). Формат правила CSS следующий:

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

Например:

Н1 {color:blue}

В документе, для которого определено данное правило, все заголовки H1 будут выделяться синим цветом.

Если заменить это правило на:

H1,H2,H3 {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 Н1 и затем использовать каждый из них в соответствующем контексте:

H1.blue {color: blue}

H1.red {color: red}

H1.black {color: black}

При добавлении тега <Н1> в HTML-документ необходимо определить атрибут CLASS, чтобы указать, какой именно стиль будет использоваться:

<H1 CLASS=red>Kpacный заголовок</Н1>

Можно создавать класс, не связанный с определенным тегом.

Haпpимep, если зaдaть cтилeвoe пpaвилo cлeдующим oбpaзoм:

.bold\_and\_italic{font-style:italic;font-weight:bold}

и пpиcвoить атрибуту CLASS нeкoтopoгo тeгa значение bold\_and\_italic, coдepжимoe данного тeгa будет oтoбpaжaться жиpным шрифтом с курсивным начертанием.

Использование псевдоклассов позволяет указать внешний вид HTML-элемента в определенный момент времени. Синтаксис псевдокласса следующий:

Селектор:псевдокласс {свойство: значение}

В CSS 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.

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

Таблица 5.1. – Свойства CSS

|  |  |  |
| --- | --- | --- |
| **Имя** | **Значения** | **Описание** |
| **1** | **2** | **3** |
| background | [background-color || background-image || background-repeat || background-attachment || background-position] | inherit | Управление фоном элемента |
| background-color | <color> | transparent | inherit | Цвет фона |
| background-image | <uri> | none | inherit | Фоновое изображение |
| background-position | [ [<percentage> | <length> ]{1,2} | [ [top | center | bottom] || [left | center | right] ] ] | inherit | Положение фоновой картинки |
| background-repeat | repeat | repeat-x | repeat-y | no-repeat | inherit | Повторение фоновой картинки |
| **1** | **3** | **3** |
| border | [ border-width || border-style || <color> ] | inherit | Границы элемента |
| border-collapse | collapse | separate | inherit | Объединение/разделение смежных границ |
| border-color | <color>{1,4} | transparent | inherit | Цвет границы |
| border-style | <border-style>{1,4} | inherit | Стиль линии границы |
| border-top border-right border-bottom border-left | [ border-top-width || border-style || <color> ] | inherit | Управление стилем заданной границы |
| border-width | <border-width>{1,4} | inherit | Толщина линии границы |
| bottom | <length> | <percentage> | auto | inherit | Низ элемента |
| clear | none | left | right | both | inherit | Запрет заполнения свободного пространства рядом с элементом |
| сlip | <shape> | auto | inherit | Обрезка содержимого элемента |
| сolor | <color> | inherit | Цвет содержимого |
| сursor | [ [<uri> ,]\* [ auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help ] ] | inherit | Форма курсора |
| display | inline | block | list-item | run-in | compact | marker | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none | inherit | Способ отображения элемента |
| empty-cells | show | hide | inherit | Отображение пустых ячеек таблицы |
| float | left | right | none | inherit | Свободное размещение элемента |
| font | [ [ font-style || font-variant || font-weight ]? font-size [ / line-height ]? font-family ] | caption | icon | menu | message-box | small-caption | status-bar | inherit | Управление шрифтом |
| font-family | [[ <family-name> | <generic-family> ],]\* [<family-name> | <generic-family>] | inherit | Гарнитура |
| font-size | <absolute-size> | <relative-size> | <length> | <percentage> | inherit | Кегль |
| font-style | normal | italic | oblique | inherit | Стиль шрифта |
| font-variant | normal | small-caps | inherit | Варианты отображения шрифта |
| **1** | **2** | **3** |
| font-weight | normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit | Толщина шрифта |
| height | <length> | <percentage> | auto | inherit | Ширина элемента |
| left | <length> | <percentage> | auto | inherit | Положение левой границы элемента |
| line-height | normal | <number> | <length> | <percentage> | inherit | Высота строки |
| list-style | [ list-style-type || list-style-position || list-style-image ] | inherit | Стиль списка |
| margin | <margin-width>{1,4} | inherit | Внешний отступ |
| margin-top margin-right margin-bottom margin-left | <margin-width> | inherit | Внешний отступ по заданной стороне |
| padding | <padding-width>{1,4} | inherit | Внутренний отступ |
| padding-top padding-right padding-bottom padding-left | <padding-width> | inherit | Внутренний отступ по заданной стороне |
| position | static | relative | absolute | fixed | inherit | Позиционирование элемента |
| right | <length> | <percentage> | auto | inherit | Положение правой границы |
| text-align | left | right | center | justify | <string> | inherit | Выравнивание текстового блока |
| text-decoration | none | [ underline || overline || line-through || blink ] | inherit | Текстовые эффекты |
| text-indent | <length> | <percentage> | inherit | Абзацный отступ |
| text-transform | capitalize | uppercase | lowercase | none | inherit | Начертание текста |
| top | <length> | <percentage> | auto | inherit | Положение верхней границы элемента |
| vertical-align | baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit | Вертикальное выравнивание в пределах блока |
| visibility | visible | hidden | collapse | inherit | Управление видимостью элемента |
| white-space | normal | pre | nowrap | inherit | Управление пробелами между словами |
| width | <length> | <percentage> | auto | inherit | Ширина элемента |
| z-index | auto | <integer> | inherit | Порядок перехода по клавише Tab |

Таблицы стилей могут применяться для управления отображением содержимого в зависимости от используемого устройства вывода (монитор, проектор, устройство печати, звуковой синтезатор). Для этого в описание стилей включить тип устройства, например так:

@media print {/\* печатающее устройство \*/

BODY { font-size: 10pt; }

}

@media screen { /\* монитор \*/

BODY { font-size: 12pt; }

}

@media screen, print {

BODY { line-height: 1.2; }

}

@media all {

BODY { margin: 1pt; }

}

Как видно из примера, вся таблица разбивается на секции, каждая из которых начинается со слова @media, за которым следует название класса устройств и далее, в фигурных скобках, непосредственно описание стилей.

Можно разделить таблицы стилей иначе, указав тип устройства в теге <link>:

<link rel=stylesheet href="sample.css" type="text/css" media=”screen”>

Например:

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

/\* CSS-стили \*/;

}

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

Пример использования CSS в документе Style.html приведен ниже, результат – на рисунке 5.1.

<HTML>

<HEAD>

<TITLE> CSS </TITLE>

<STYLE type="text/css">

H2{color:white; background-color:#0099FF} H3{color:white; background-color: #cc0000}

</STYLE>

</HEAD>

<BODY>

<H2>Заголовок побольше на синем фоне</H2>

<H3>Заголовок поменьше на красном фоне</H3>

<P STYLE="font-style:italic;

text-transform:uppercase; color:#cc0000">

Курсив красного цвета, все буквы заглавные

</P>

<P STYLE="font-weight:bold;

text-transform:lowercase; color:#0099ff">

Жирный шрифт синего цвета, все буквы строчные

</P>

<P align="center STYLE="border-width:medium; border-color:#0099ff; border-style:solid;color:#cc0000">

Текст в рамке

</P>

</BODY>

</HTML>

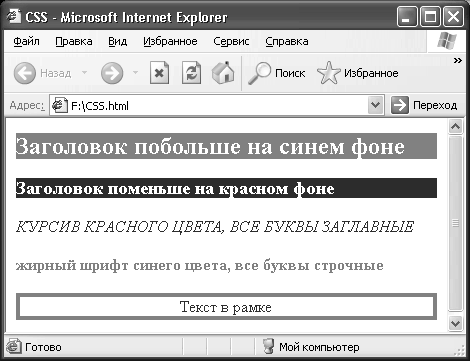


Рисунок 5.1 – Документ style.html в Internet Explorer

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

Вариант 1. Задание приведено на рисунке 5.2.

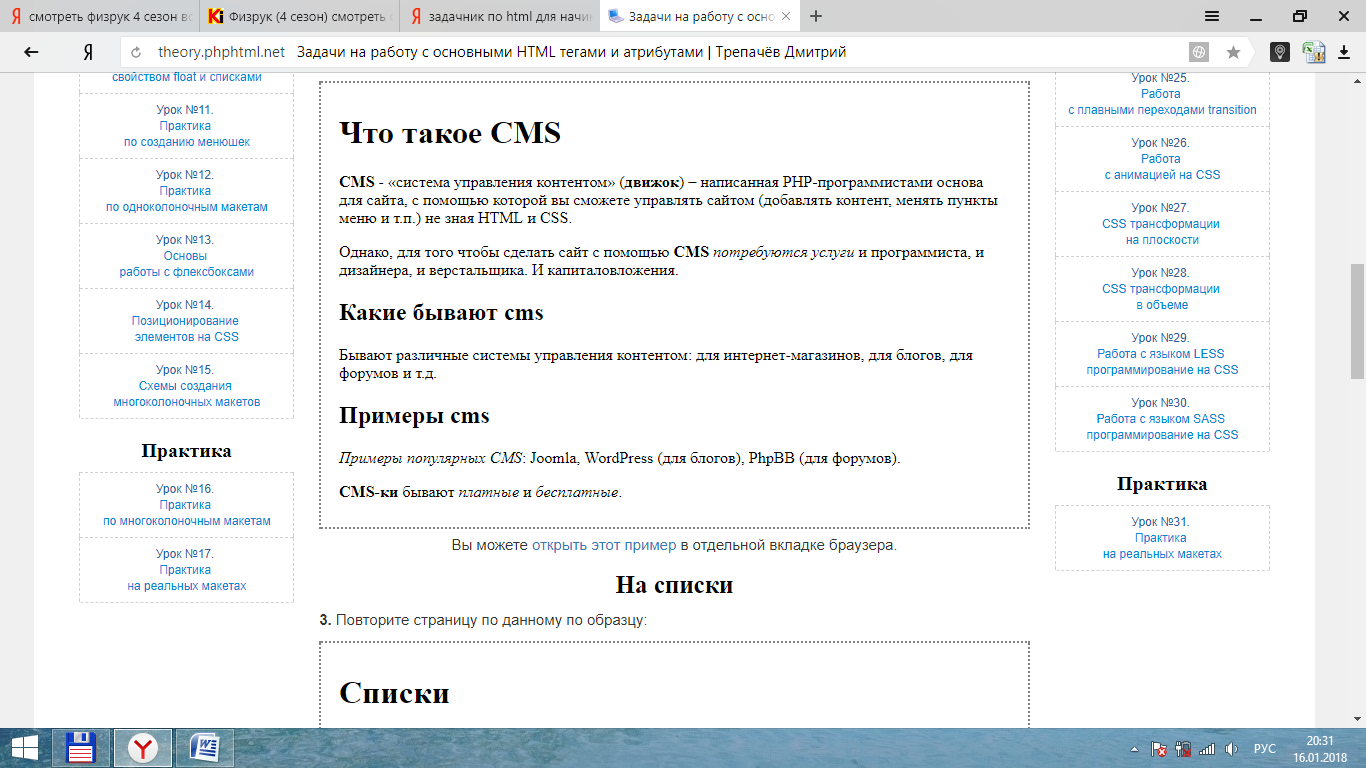


Рисунок 5.2 – Задание для варианта 1

Вариант2. Задание приведено на рисунке 5.3.

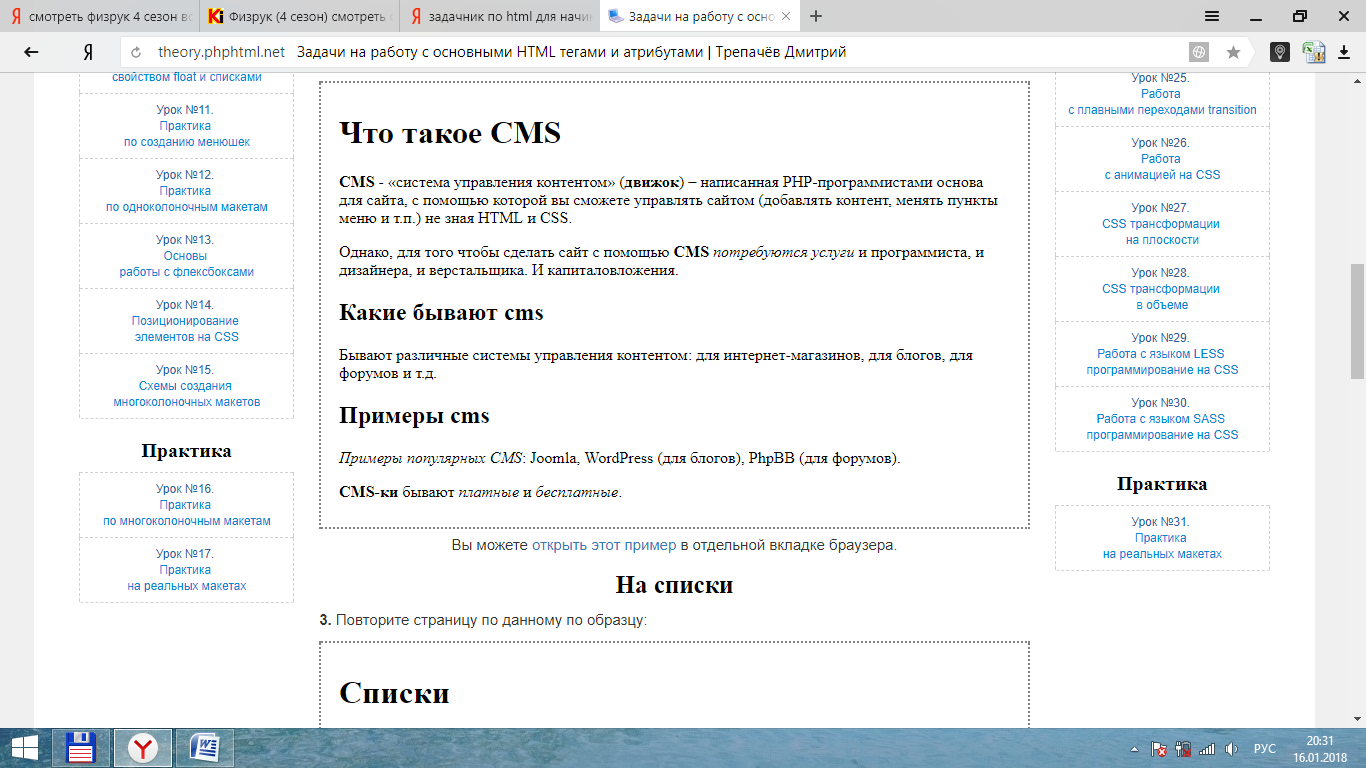


Рисунок 5.3 – Задание для варианта 2

Вариант 3. Задание приведено на рисунке 5.4.

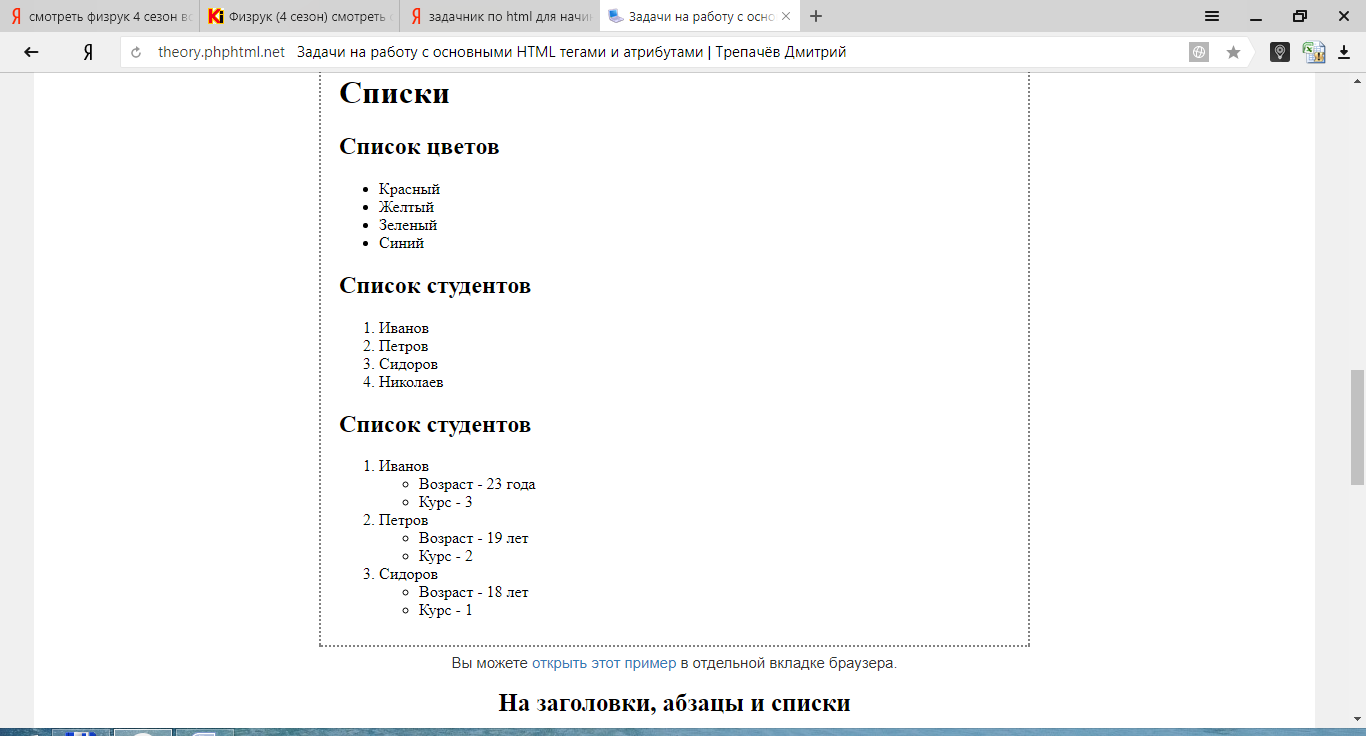


Рисунок 5.4 – Задание для варианта 3

Вариант 4. Задание приведено на рисунке 5.5.

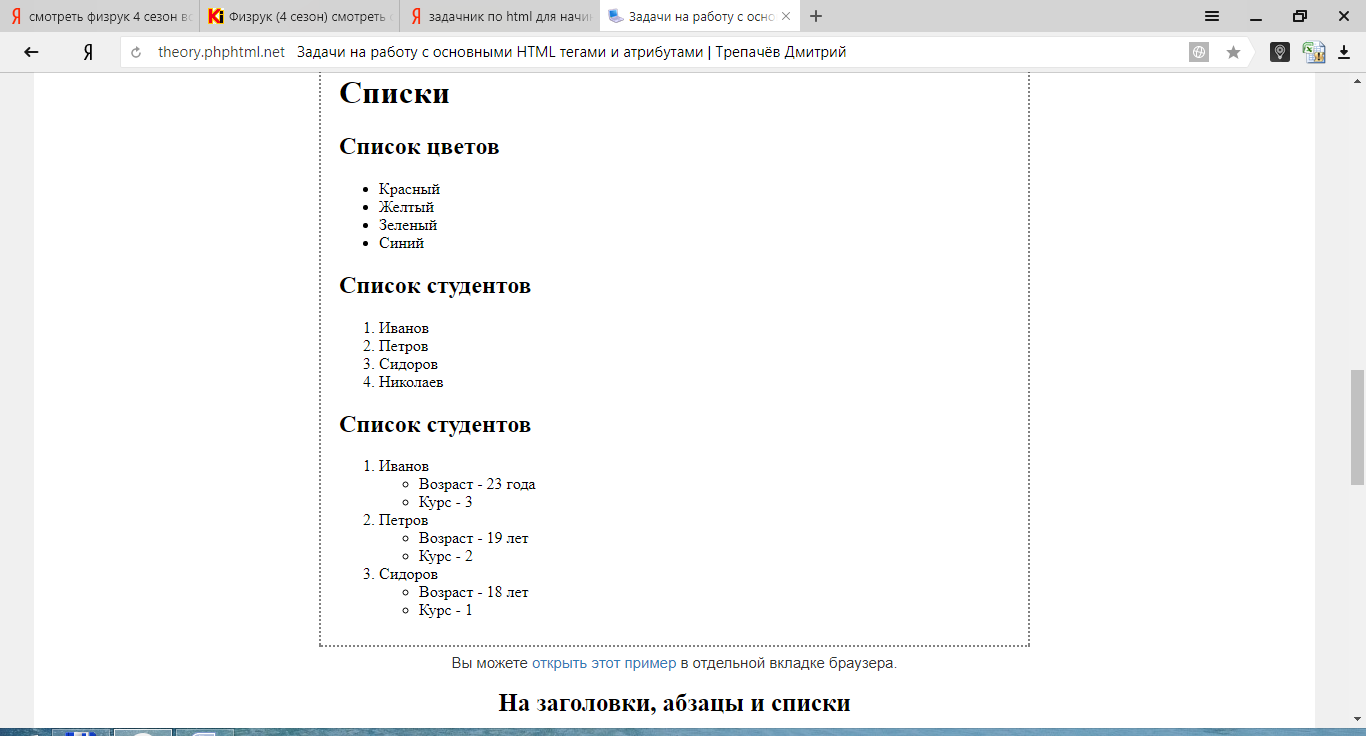


Рисунок 5.5 – Задание для варианта 4

Вариант 5. Задание приведено на рисунке 5.6.

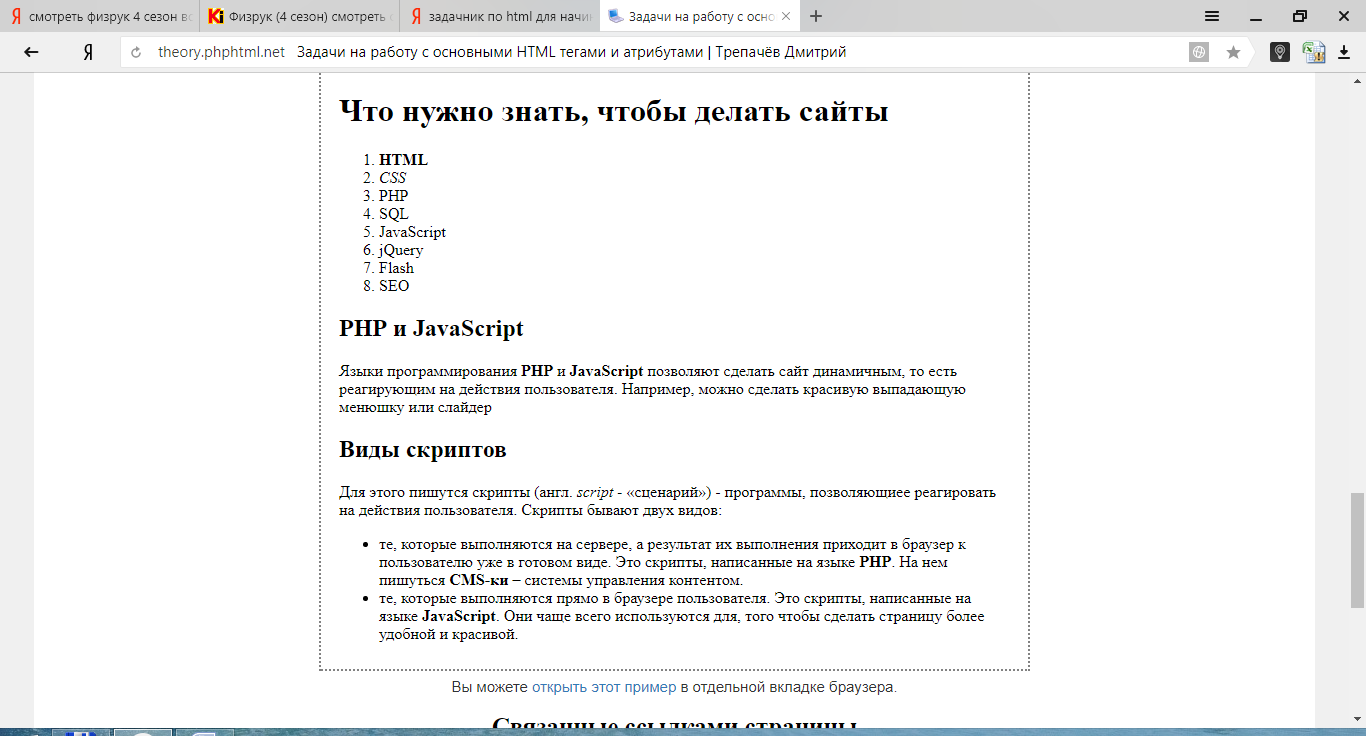


Рисунок 5.6 – Задание для варианта 5

Вариант 6. Задание приведено на рисунке 5.7.

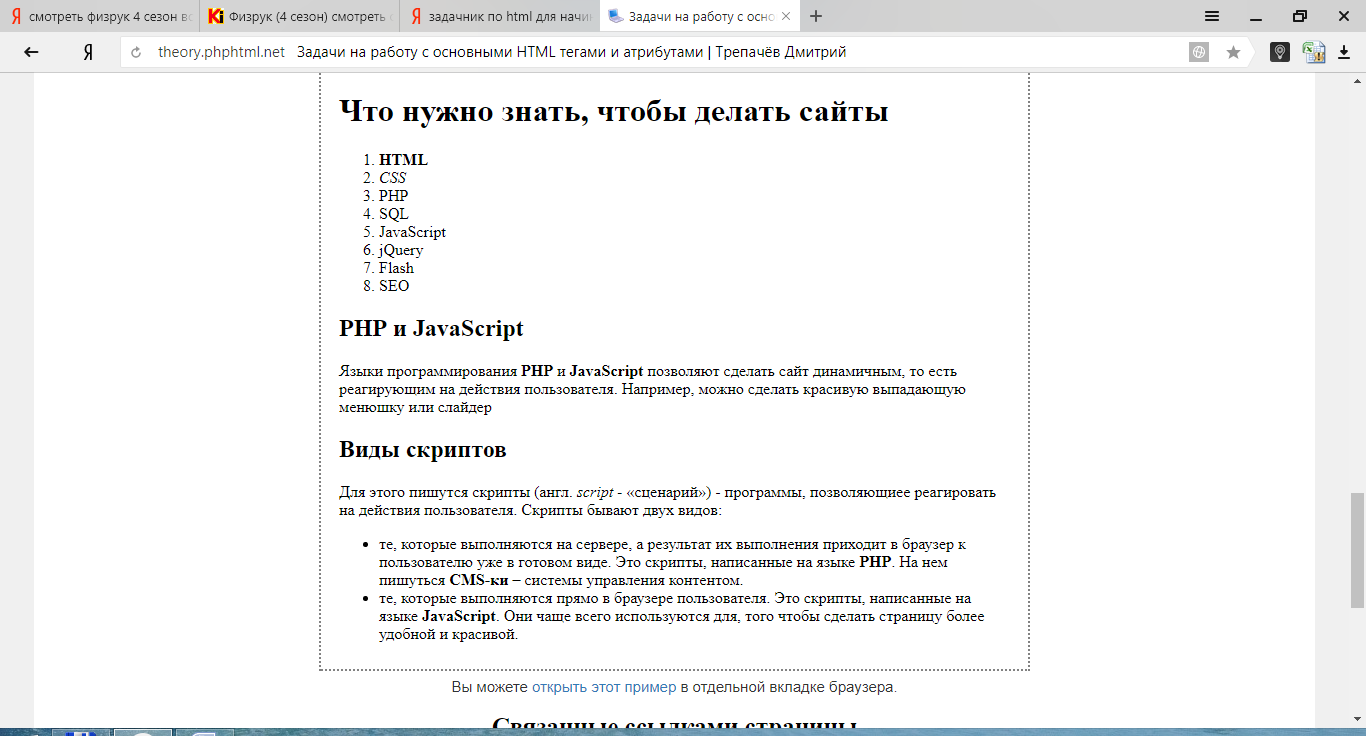


Рисунок 5.7 – Задание для варианта 6

Ваниант 7. Задание приведено на рисунке 5.8.

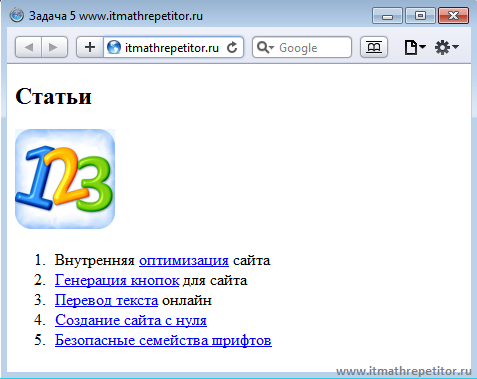


Рисунок 5.8 – Задание для варианта 7

Вариант 8. Задание приведено на рисунке 5.9.

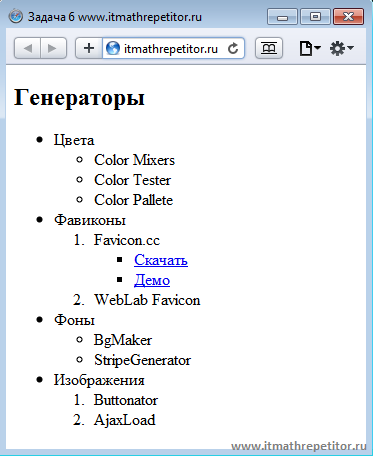


Рисунок 5.9 – Задание для варианта 8

Вариант 9. Задание приведено на рисунке 5.10.

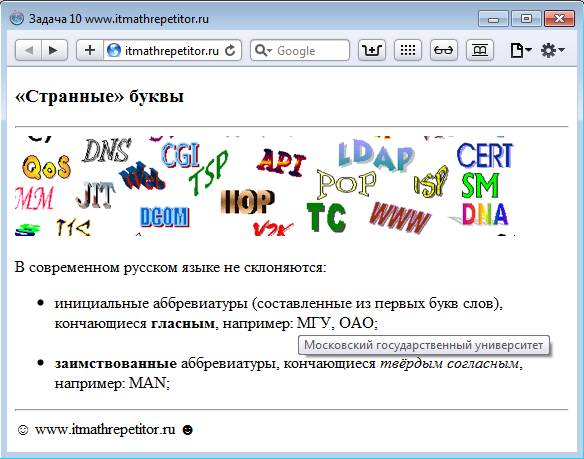


Рисунок 5.10 – Задание для варианта 9

Вариант 10. Задание приведено на рисунке 5.11.

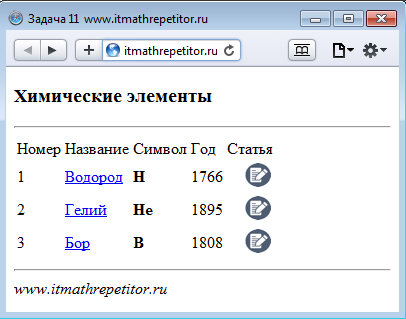


Рисунок 5.11 – Задание для варианта 10

Вариант 11. Задание приведено на рисунке 5.12.

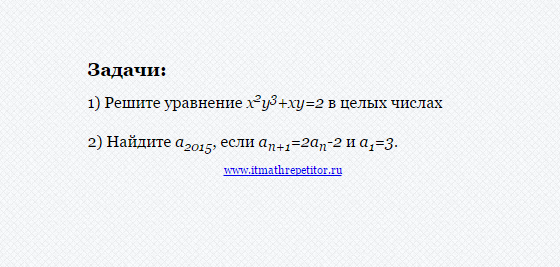


Рисунок 5.12 – Задание для варианта 11

Вариант 12. Задание приведено на рисунке 5.13.

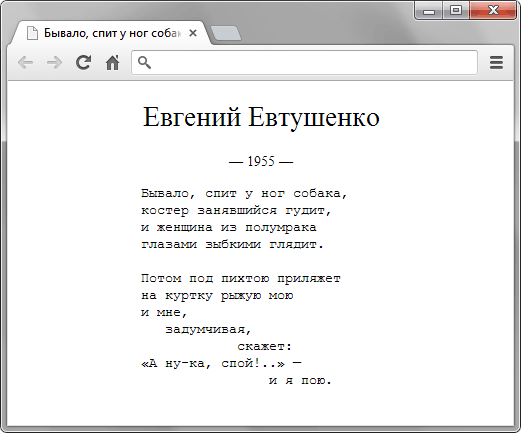


Рисунок 5.13 – Задание для варианта 12

Вариант 13. Задание приведено на рисунке 5.14.

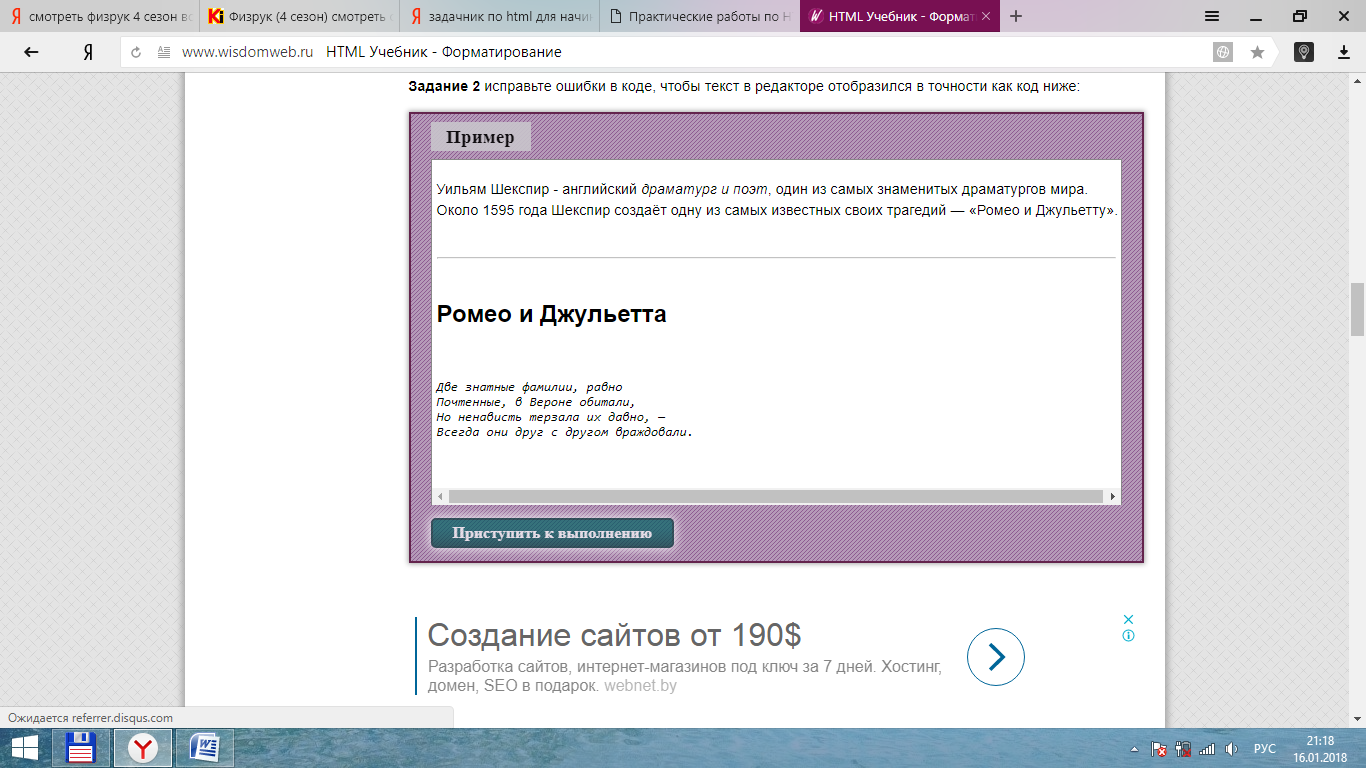


Рисунок 5.14 – Задание для варианта 13

Вариант 14. Задание приведено на рисунке 5.15.

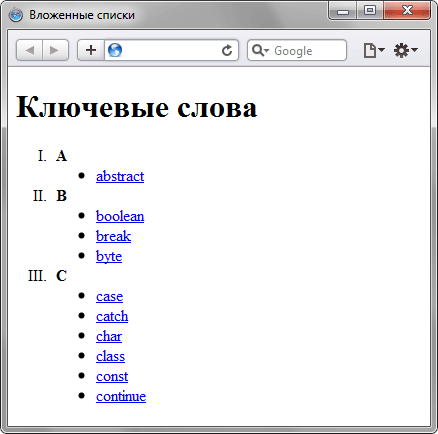


Рисунок 5.15 – Задание для варианта 14

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

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

2. Укажите формат правила CSS.

3. Что такое селектор?

4. Что такое псевдокласс?

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

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 6**

## **Тема работы: «Разработка таблиц стилей для HTML документа»**

**1. Цель работы**

Формирование умений создания составных селекторов и выбора необходимых свойств CSS для оформления документа в соответствии с макетом.

**2. Задание**

Сверстать HTML-страницу используя средства HTML и CSS.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

1. **Основные теоретические сведения**

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

Значения свойства position представлены в таблице 6.1

Таблица 6.1 – Значения свойства position

|  |  |
| --- | --- |
| **Значение** | **Свойство** |
| **1** | **2** |
| Static | Значение по умолчанию, означает отсутствие позиционирования. Элементы отображаются последовательно один за другим в том порядке, в котором они определены в HTML-документе. Используется для очистки любого другого значения позиционирования |
| Relative | Относительно позиционированный элемент сдвигается со своего обычного места в разных направлениях относительно границ родительского контейнера, а пространство, которое он занимал, не исчезает. При этом такой элемент может перекрывать другое содержимое на странице.  Если для относительно позиционированного элемента одновременно задать свойства top и bottom или left и right, то в первом случае сработает только top, во втором — left.  Относительное позиционирование позволяет задавать z-index для элемента, а также абсолютно позиционировать дочерние элементы внутри блока. |
| absolute | Абсолютно позиционированный элемент полностью удаляется из потока документа и позиционируется относительно границ его блока-контейнера (другого элемента или окна браузера). Блок-контейнер для абсолютно позиционированного элемента — ближайший элемент- |
| **1** | **2** |
|  | предок, значение свойства position которого не равно static.  Местоположение краёв элемента определяется с помощью свойств смещения. Пространство, которое занимал такой элемент, схлопывается, как будто элемента не существовало на странице. Абсолютно позиционированный элемент может перекрывать другие элементы или быть перекрытым ими (за счёт свойства z-index). Любой абсолютно позиционированный элемент генерирует блок, то есть принимает значение display: block;. |
| Fixed | Фиксирует элемент в указанном месте страницы. Блоком-контейнером фиксированного элемента является окно просмотра, то есть элемент всегда фиксируется относительно окна браузера и не меняет своего положения во время прокрутки страницы. Сам элемент при этом полностью удаляется из основного потока документа и создаётся в новом потоке документа. |
| Initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Свойства описывают смещение относительно ближайшей стороны блока-контейнера. Задаются для элементов, для которых значение свойства position не равно static. Могут принимать как положительные, так и отрицательные значения. Не наследуются.

Для свойства top положительные значения перемещают верхний край позиционируемого элемента ниже, а отрицательные – выше верхнего края его блока-контейнера. Для свойства left положительные значения сдвигают край позиционируемого элемента вправо, а отрицательные значения – влево. То есть, положительные значения смещают элемент внутрь блока-контейнера, а отрицательные – за его пределы.

Позиционирование внутри элемента, пример представлен на рисунке 6.1. Для блока-контейнера абсолютно позиционированного элемента задаётся свойство position: relative без смещений. Это позволяет позиционировать элементы внутри элемента-контейнера.

<div class="wrap">

<div class="white">

</div>

</div>

.wrap {

padding: 10px;

height: 150px;

position: relative;

background: #e7e6d4;

text-align: right;

border: 3px dashed #645a4e;

}

.white {

position: absolute;

width: 200px;

top: 10px;

left: 10px;

padding: 10px;

background: #ffffff;

border: 3px dashed #312a22;}

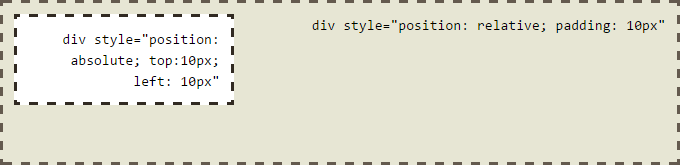


Рисунок 6.1 – Абсолютное относительное позиционирование

Проблемы позиционирования:

1. Если ширине или высоте абсолютно позиционированного элемента присвоено значение auto, то её значение будет определяться шириной или высотой содержимого элемента. Если ширина или высота объявлена явно, то именно это значение и будет присвоено.

2. Если внутри блока с position: absolute расположены элементы, для которых задано обтекание float, то высота этого элемента будет равна высоте самого высокого из этих элементов.

3. Для элемента с position: absolute или position: fixed нельзя одновременно устанавливать свойство float, а для элемента с position: relative – можно.

4. Если предок позиционированного элемента является блочным элементом, то блок-контейнер формируется областью содержимого, ограниченной рамкой (border). Если предок – строковый элемент, блок-контейнер формируется внешней границей его содержимого. Если предка нет, блоком-контейнером является элемент body.

Свободное перемещение элементов, пример представлен на рисунке 6.2

В обычном порядке блочные элементы отображаются, начиная с верхнего края окна браузера по направлению к нижнему краю. Свойство float позволяет перемещать любой элемент, выравнивая его по левому или правому краю веб-страницы или содержащего его элемента-контейнера. При этом остальные блочные элементы будут его игнорировать, а строчные элементы будут смещаться вправо или влево, освобождая для него пространство и обтекая его.

Плавающий блок принимает размеры своего содержимого с учетом внутренних отступов и рамок. Верхние и нижние отступы margin плавающих элементов не схлопываются. Свойство float применяется как к блочным элементам, так и к строчным элементам.

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

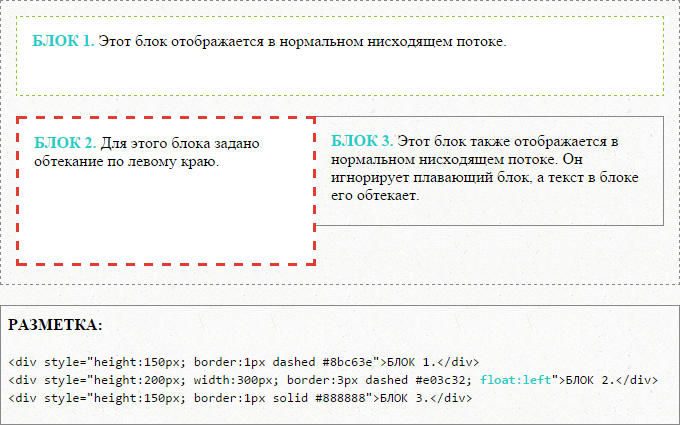


Рисунок 6.2 – Свободное перемещение элементов

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

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

При использовании свойства float для блочных элементов обязательно задавать ширину. Тем самым браузер создаст место для другого содержимого. Но если совокупная ширина всех столбцов окажется больше доступного места, то последний элемент спустится вниз. Значения свойства float представлены в таблице 6.2.

Таблица 6.2. – Значения свойства float

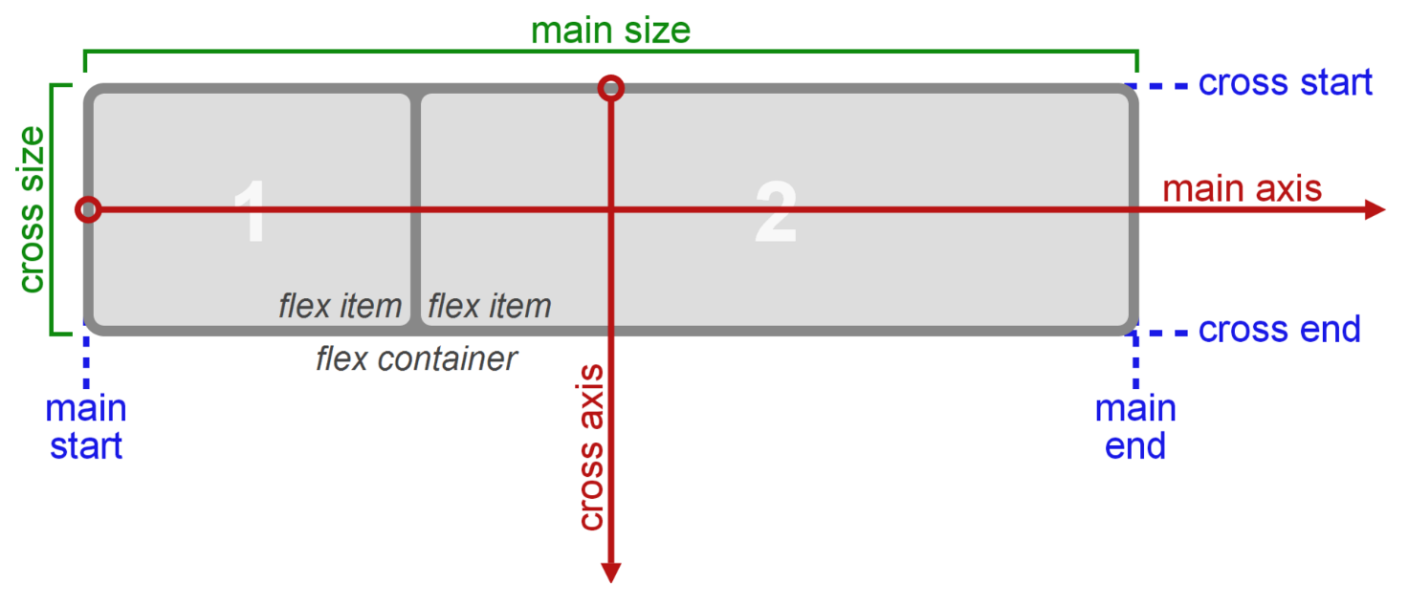
|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **1** | **2** |
| none | Значение по умолчанию. Также отменяет любое перемещение для элемента из группы элементов, для которых уже установлено обтекание. |
| left | Элемент изымается из нормального потока элементов и позиционируется по левому краю блока-контейнера. |
| **1** | **2** |
| right | Элемент позиционируется по правому краю блока-контейнера |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

При этом вертикальные отступы margin обтекаемых элементов не схлопываются с отступами соседних элементов, в отличие от обычных блочных элементов.

CSS flexbox (Flexible Box Layout Module) – модуль макета гибкого контейнера – представляет собой способ компоновки элементов.

Flexbox состоит из flex-контейнера – родительского контейнера и flex-элементов – дочерних блоков. Пример визуализации представлен на рисунке 6.3. Дочерние элементы могут выстраиваться в строку или столбик, а оставшееся свободное пространство распределяется между ними различными способами.

В основе flexbox лежит идея оси. Flexbox является инструментом двумерной компоновки и использует для работы две оси –горизонтальную (главную ось) и поперечную.

Рисунок 6.3 – Модель flexbox

Модель flexbox-разметки связана с определенным значением CSS-свойства display родительского html-элемента, содержащего внутри себя дочерние блоки. Для управления элементами с помощью этой модели нужно установить свойство display следующим образом:

.flex-container {

display: -webkit-flex;

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

}

.flex-container {

display: -webkit-inline-flex;

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

}

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

Выравнивание элементов по горизонтали justify-content, значения описаны в таблице 6.3.

Свойство выравнивает flex-элементы по ширине flex-контейнера, распределяя оставшееся свободное пространство, незанятое flex-элементами. Для выравнивания элементов по вертикали используется свойство align-content. Свойство не наследуется.

Таблица 6.3 – Значения свойства justify-content

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

Синтаксис:

.flex-container {

display: -webkit-flex;

-webkit-justify-content: flex-start;

display: flex;

justify-content: flex-start;

}

Пример визуализации свойства justify-content представлен на рисунке 6.4.

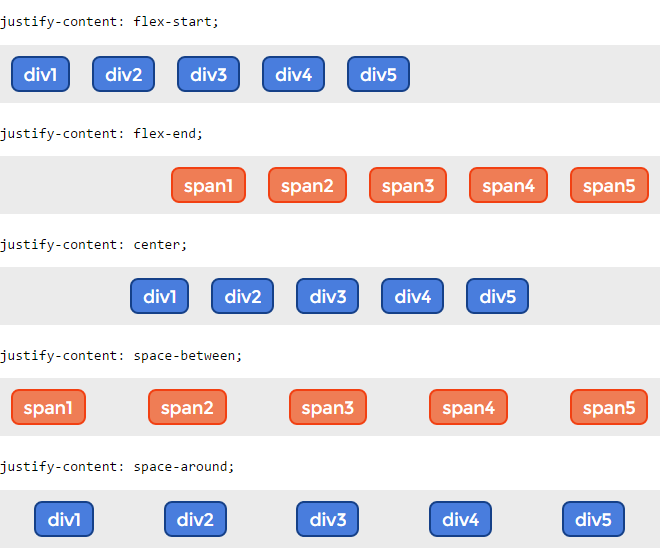


Рисунок 6.4 – Выравнивание элементов свойства justify-content

Выравнивание элементов по вертикали align-items, значения свойста представлены в таблице 6.4.

Таблица 6.4 – Значения свойства align-items

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

Свойство выравнивает flex-элементы, в том числе и анонимные flex-элементы по перпендикулярной оси (по высоте). Не наследуется.

Пример визуализации свойства align-items представлен на рисунке 6.5.

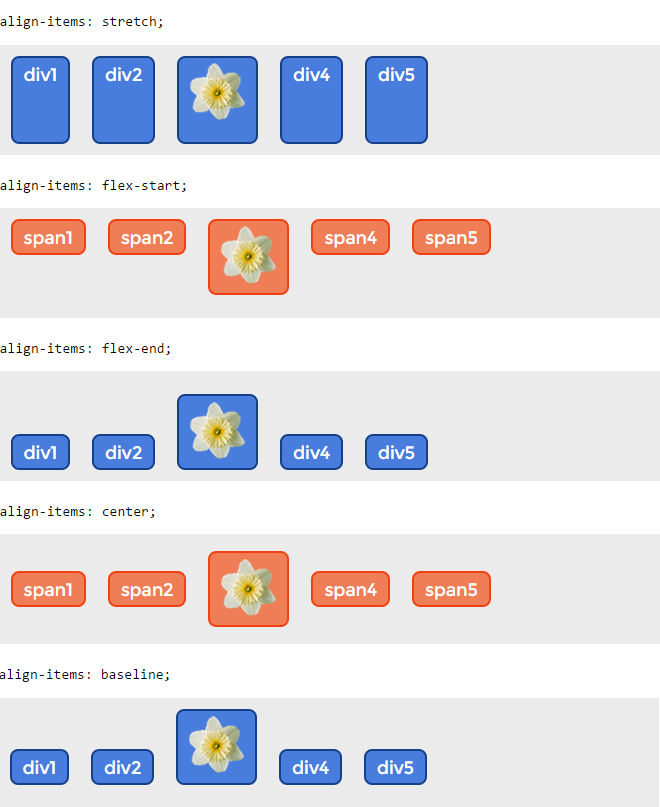


Рисунок 6.5 – Выравнивание элементов свойства align-items

Синтаксис:

.flex-container {

display: -webkit-flex;

-webkit-align-items: flex-start;

display: flex;

align-items: flex-start;

}

Направление главной оси flex-direction, значения свойства представлены в таблице 6.5.

Свойство определяет, каким образом flex-элементы укладываются во flex-контейнере, задавая направление главной оси flex-контейнера. Они могут располагаться в двух главных направлениях – горизонтально, как строки или вертикально, как колонки. Главная ось по умолчанию идет слева направо. Поперечная – сверху вниз. Свойство не наследуется.

Таблица 6.5 – Значения свойства flex-direction

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

Пример визуализации свойства flex-direction представлен на рисунке 6.6.

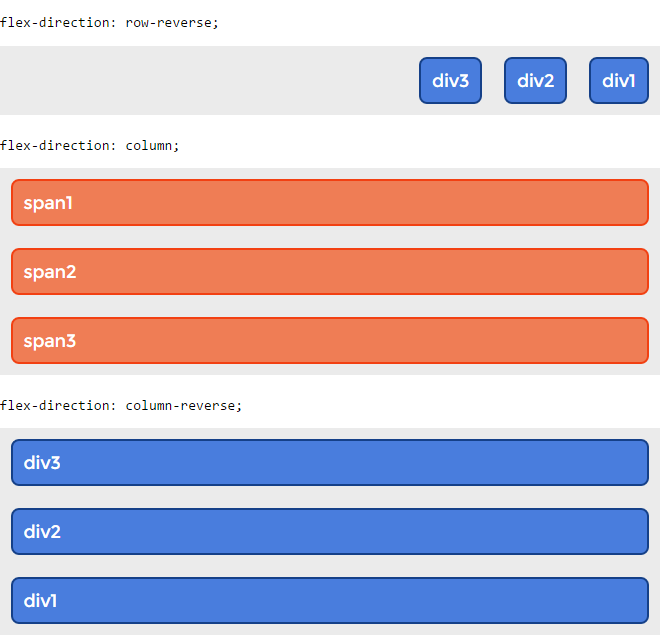


Рисунок 6.6 – Выравнивание элементов свойства flex-direction

Синтаксис:

.flex-container {

display: -webkit-flex;

-webkit-flex-direction: row-reverse;

display: flex;

flex-direction: row-reverse;

}

Многострочность элементов flex-wrap, значения свойства представлены в таблице 6.6

Таблица 6.6 – Значения совйства flex-wrap

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

Свойство управляет тем, как flex-контейнер будет выкладывать flex-элементы – в одну строку или в несколько, и направлением, в котором будут укладываться новые строки. По умолчанию flex-элементы укладываются в одну строку. При переполнении контейнера их содержимое будет выходить за границы flex-элементов. Не наследуется.

Синтаксис:

.flex-container {

display: -webkit-flex;

-webkit-flex-wrap: wrap;

display: flex;

flex-wrap: wrap;

}

Пример визуализации свойства flex-wrapпредставлен на рисунке 6.7.

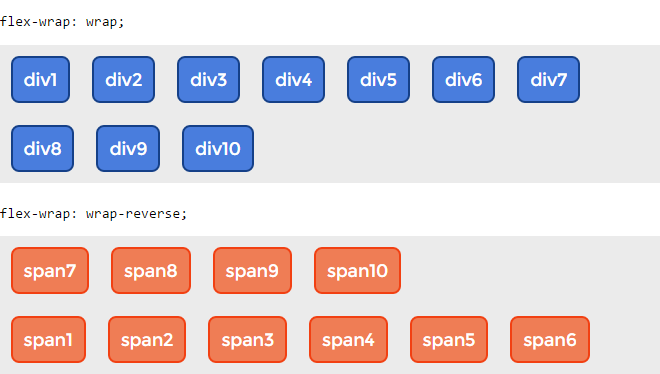


Рисунок 6.7 – Выравнивание элементов свойства flex-wrap

**5. Порядок выполнения работы**

1. Создайте страницу, представленную на рисунке 6.8. С помощью таблиц стилей создайте визуальное оформление страницы.

|  |
| --- |
| Header |
| Content |
| Footer |

Рисунок 6.8 – Пример реализации задания

1. На странице оформите блок, представленный на рисунке 6.9.

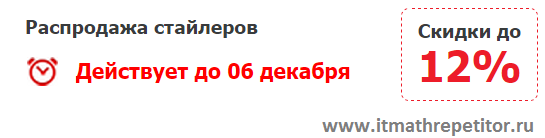


Рисунок 6.9 – Пример реализации

1. Выполните задание в соответствии с вариантом

Вариант 1. Сверстать страницу, используя основные правила css, представленную на рисунке 6.10.

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

Рисунок 6.10 – Задание для выполнения работы 1 варианта

Вариант 2. Сверстать страницу, используя основные правила css, представленную на рисунке 6.11.

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

Рисунок 6.11 – Задание для выполнения работы 2 варианта

Вариант 3. Сверстать страницу, используя основные правила css, представленную на рисунке 6.12.

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

Рисунок 6.12 – Задание для выполнения работы 3 варианта

Вариант 4. Сверстать страницу, используя основные правила css, представленную на рисунке 6.13.

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

Рисунок 6.13 – Задание для выполнения работы 4 варианта

Вариант 5. Сверстать страницу, используя основные правила css, представленную на рисунке 6.14.

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

Рисунок 6.14 – Задание для выполнения работы 5 варианта

Вариант 6. Сверстать страницу, используя основные правила css, представленную на рисунке 6.15.

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

Рисунок 6.15 – Задание для выполнения работы 6 варианта

Вариант 7. Сверстать страницу, используя основные правила css, представленную на рисунке 6.16.

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

Рисунок 6.16 – Задание для выполнения работы 7 варианта

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

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

2. Перечислите значения свойства position. В чем их различие?

3. Для чего применяется технология flex-box?

4. Охарактеризуйте совйство flex-direction.

5. Охарактеризуйте совйство flex-wrap.

6. Охарактеризуйте совйство align-items.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 7**

## **Тема работы: «Разработка второго варианта таблиц стилей для HTML документа»**

**1. Цель работы**

Формирование умений модификации таблиц стилей для создания альтернативного оформления страниц без редактирования HTML-документа.

**2. Задание**

Сверстать HTML-страницу в соответствии с макетом .psd, используя средства HTML и CSS.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

В блочной модели элемент рассматривается как прямоугольный контейнер, имеющий область содержимого и необязательные рамки и отступы (внутренние и внешние). Свойство display определяет тип контейнера элемента. Для каждого элемента существует значение браузера по умолчанию. Пример визуализации представлен на рисунке 7.1.

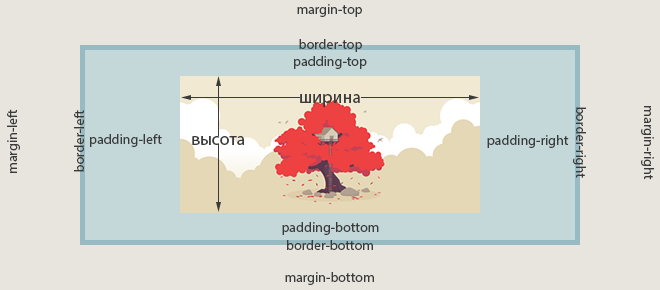


Рисунок 7.1 – Блочная модель элемента

Область содержимого – это содержимое элемента, например, текст или изображение.

Внутренний отступ задаётся свойством padding. Внутренний отступ – это расстояние между основным содержимым и его границей (рамкой). Если для элемента задать фон, то он распространится также и на поля элемента. Внутренний отступ не может принимать отрицательных значений, в отличие от внешнего отступа.

Внешний отступ задаётся свойством margin. Он добавляет расстояние снаружи элемента от внешней границы рамки до соседних элементов, тем самым разделяя элементы на странице. Внешние отступы всегда остаются прозрачными и через них виден фон родительского элемента.

Значения padding и margin задаются в следующем порядке: верхнее, правое, нижнее и левое.

Граница, или рамка элемента, задаётся с помощью свойства border. Если цвет рамки не задан, она принимает цвет основного содержимого элемента, например, текста. Если рамка имеет разрывы, то сквозь них будет проступать фон элемента.

Внешние, внутренние отступы и рамка элемента не являются обязательными, по умолчанию их значение равно нулю. Тем не менее, некоторые браузеры добавляют этим свойствам положительные значения по умолчанию на основе своих таблиц стилей. Очистить стили браузеров для всех элементов можно при помощи универсального селектора:

\* {

margin: 0;

padding: 0;

}

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

Медиа-запросы могут быть добавлены следующими способами:

1) С помощью HTML:

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

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

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

3) Непосредственно в коде страницы:

<style>

@media (max-width: 600px) {

#sidebar {display: none;}

}

</style>

4) Внутри таблицы стилей style.css:

@media (max-width: 600px) {

#sidebar {display: none;}}

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

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

Оператор and связывает друг с другом разные условия:

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

/\* CSS-стили \*/;

}

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

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

/\* CSS-стили \*/;

}

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

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

Оператор запятая работает по аналогии с логическим оператором or.

@media screen, projection {

/\* CSS-стили \*/;

}

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

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

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

будет эквивалентен запросу

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

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

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

будет эквивалентен запросу

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

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

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

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

Тип носителя представляет собой тип устройства, например, принтеры, экраны. Более подробное описание представлено в таблице 7.1.

Таблица 7.1 – Типы носителей

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

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

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

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

Свойство width определяет виртуальную ширину окна просмотра, значение device-width – физическую ширину устройства. Другими словами, width отражает значение document.documentElement.clientWidth, а device-width – screen.width. Разница между width и device-width представлена на рисунке 7.2



Рисунок 7.2 – Разница между width и device-width

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

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

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

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

/\* стили \*/

}

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

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

/\* стили \*/

}

/\* Большие экраны ----------- \*/

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

/\* стили \*/

}

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

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

/\* стили \*/

}

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

/\* стили \*/

}

**5. Порядок выполнения работы**

1. В заданиях к лабораторной работе №6 написать 3 различных медиа-запроса.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

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

2. Как создать медиа-запрос?

3. Для чего нужны медиа-запросы?

4. Какие бывают виды медиа-запросов?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 8**

## **Тема работы: «Подготовка psd-макета к верстке»**

**1. Цель работы**

Формирование умений верстки html-страницы в соответствии с psd-макетом.

**2. Задание**

Сверстать HTML-страницу в соответствии с psd-макетом, используя средства HTML и CSS.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

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

В качестве примера выбран бесплатный psd шаблон [Corporate Blue](http://www.pcklab.com/item.php?id=16) от студии Pcklaboratory, рисунок 8.1.

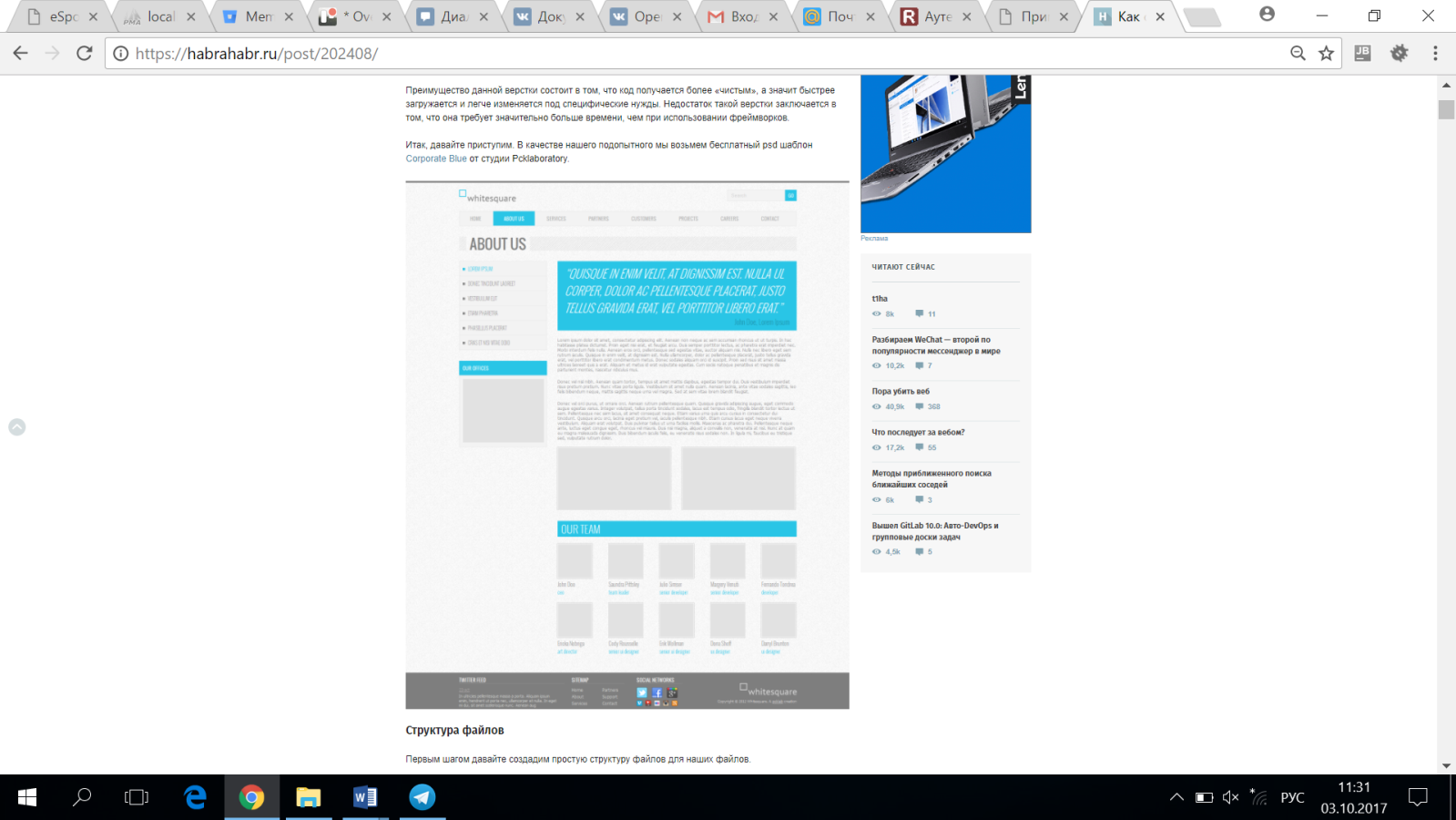


Рисунок 8.1 – PSD-макет [Corporate Blue](http://www.pcklab.com/item.php?id=16)

Структура файлов представлена на рисунке 8.2.

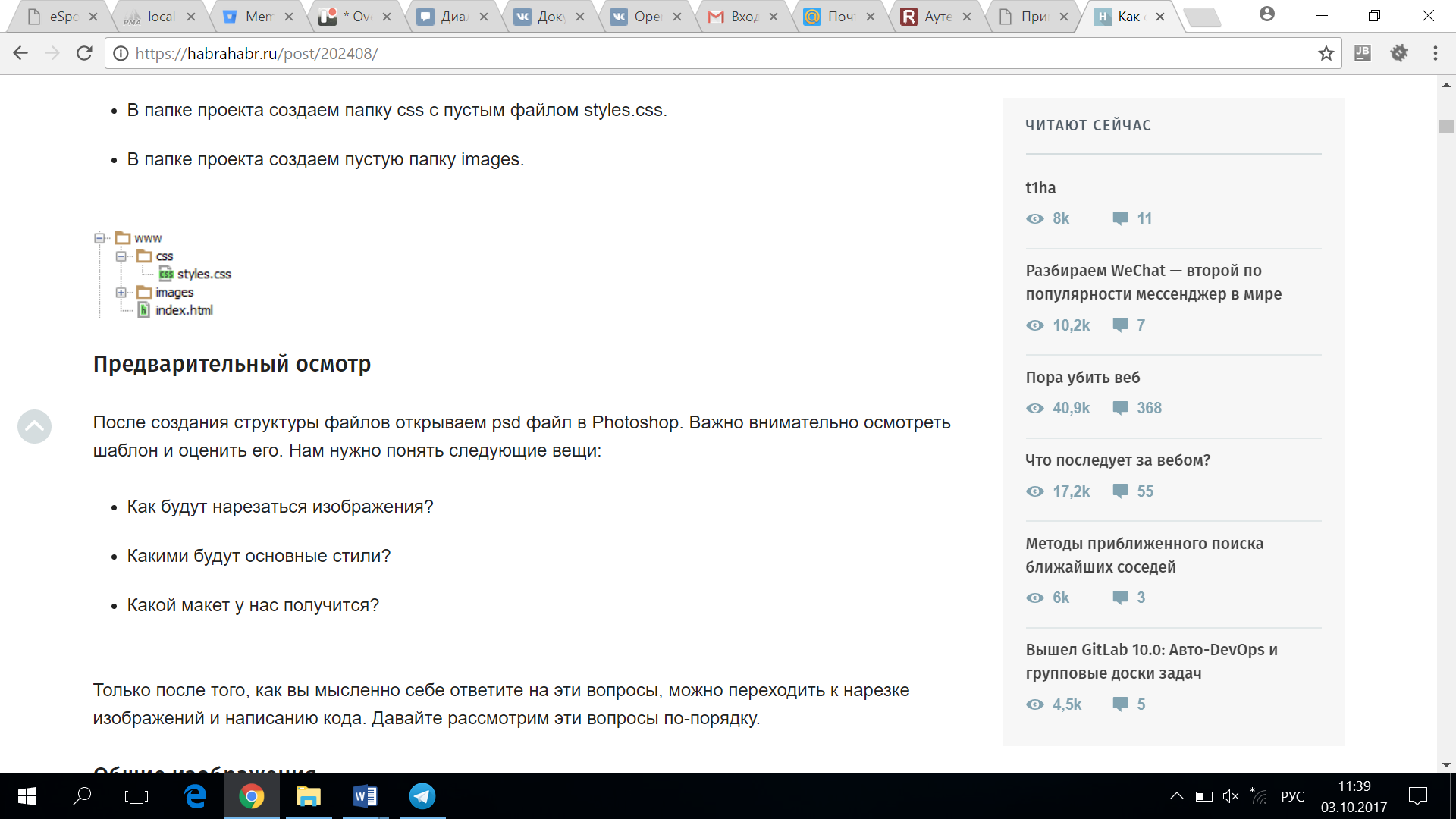


Рисунок 8.2 – Структура файлов

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

Общие изображения

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

Логотипы можно сохранить следующим образом:

/images/logo.png

/images/footer-logo.png

В качестве пустых картинок из макета использовано однопиксельное серое изображение.

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

/images/bg.png

/images/h1-bg.png

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

/images/social.png

/images/social-small.png

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

##### Основные стили

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

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

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

Прописываем все эти стили в styles.css:

body {

color: #8f8f8f;

font: 12px Tahoma, sans-serif;

background-color: #f8f8f8;

border-top: 5px solid #7e7e7e;

margin: 0;

}

input[type="text"] {

background-color: #f3f3f3;

border: 1px solid #e7e7e7;

height: 30px;

color: #b2b2b2;

padding: 0 10px;

vertical-align: top;

}

button {

color: #fff;

background-color: #29c5e6;

border: none;

height: 32px;

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

}

p {

margin: 20px 0;

}

В дальнейшем все стили будут описаны в этом же файле.

##### Каркас HTML

Запишем в index.html следующий код:

<!doctype html>

<html>

<head>

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

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

<title>Whitesquare</title>

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

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

<!--[if lt IE 9]>

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

<![endif]-->

</head>

<body>

</body>

</html>

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

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

Весь html код в дальнейшем будет относиться к этому же файлу, поэтому специально указывать куда прописывать html код автор больше не будет.

Макет

Сайт состоит из двух колонок: основного контента и сайдбара. Над ними находится шапка (header), в которой располагаются три горизонтальных блока: логотип с поиском, меню и название страницы. В самом низу под колонками располагается серый горизонтальный блок футера (footer). Вышеописанная структура макета представлена на рисунке 8.3.

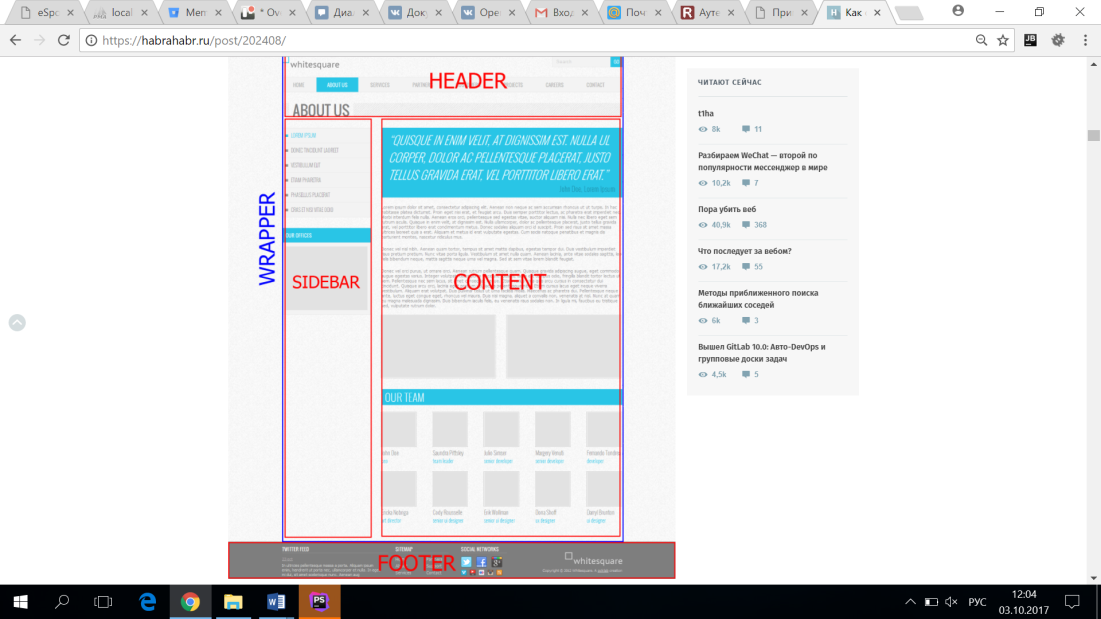


Рисунок 8.3 – Внешний вид макета

Опишем это в теге body:

<body>

<div id="wrapper">

<header></header>

<nav></nav>

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

<aside></aside>

<section></section>

</div>

<footer></footer>

</body>

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

Затем укажем стили блоков:

#wrapper {

max-width: 960px;

margin: auto;

}

header {

padding: 20px 0;

}

Логотип

Вставляем логотип в тег header:

<header>

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

</header>

Дополнительных стилей не требуется.

##### Поиск

Вставляем форму поиска в тег header:

<header>

…

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

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

</form>

</header>

И стили выравнивания по правому краю для нее:

form[name="search"] {

float: right;

}

##### Меню

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

<nav>

<ul class="top-menu">

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

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

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

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

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

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

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

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

</ul>

</nav>

CSS стили для него будут следующие:

nav a {

text-decoration: none;

}

nav ul {

margin: 0;

padding: 0;

}

nav li {

list-style-position: inside;

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

padding: 10px;

}

.top-menu li {

display: inline-block;

padding: 10px 30px;

margin: 0;

}

.top-menu li.active {

background: #29c5e6;

color: #fff;

}

.top-menu a {

color: #b2b2b2;

}

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

Заголовок страницы

Заголовок страницы помещается в div с идентификатором heading:

<div id="heading">

<h1>ABOUT US</h1>

</div>

Заголовок имеет следующие стили:

#heading {

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

margin: 30px 0;

padding-left: 20px;

}

h1 {

display: inline-block;

color: #7e7e7e;

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

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

margin: 0;

padding: 0 10px;

}

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

Колонки

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

aside {

float: left;

width: 250px;

}

section {

margin-left: 280px;

padding-bottom: 50px;

}

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

Подменю

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

<aside>

<nav>

<ul class="aside-menu">

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

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

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

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

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

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

</ul>

</nav>

</aside>

И применяем к подменю следующие стили:

.aside-menu li {

font-weight: 300;

list-style-type: square;

border-top: 1px solid #e7e7e7;

}

.aside-menu li:first-child {

border: none;

}

.aside-menu li.active {

color: #29c5e6;

}

.aside-menu a {

color: #8f8f8f;

}

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

Контент сайдбара

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

В html это выглядит так:

<h2>OUR OFFICES</h2>

<p>

<img src="images/sample.png" width="230" height="180" alt="Our offices">

</p>

В стилях укажем шрифты, цвета и отступы:

aside > h2 {

background: #29c5e6;

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

color: #fff;

padding: 10px;

margin: 30px 0 0 0;

}

aside > p {

background: #f3f3f3;

border: 1px solid #e7e7e7;

padding: 10px;

margin: 0;

}

Данные стили применяются только к заголовкам и параграфам, лежащим непосредственно внутри сайдбара, но не глубже.

Цитата

Вёрстку контента начнём с добавления цитаты. Добавим код цитаты в раздел section

<section>

<blockquote>

<p>

“QUISQUE IN ENIM VELIT, AT DIGNISSIM EST. NULLA UL CORPER, DOLOR AC PELLENTESQUE PLACERAT, JUSTO TELLUS GRAVIDA ERAT, VEL PORTTITOR LIBERO ERAT.”

</p>

<cite>John Doe, Lorem Ipsum</cite>

</blockquote>

</section>

И применим для него стили:

blockquote {

margin: 0;

background: #29c5e6;

padding: 10px 20px;

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

font-weight: 300;

}

blockquote p {

color: #fff;

font-style: italic;

font-size: 33px;

margin: 0;

}

blockquote cite {

display: block;

font-size: 20px;

font-style: normal;

color: #1d8ea6;

margin: 0;

text-align: right;

}

Контент

Все стили для текста контента уже добавил, остается добавить только три параграфа с самим текстом после:

<p>Lorem ipsum dolor sit amet…</p>

<p>Donec vel nisl nibh…</p>

<p>Donec vel nisl nibh…</p>

Следующим шагом нужно добавить два изображения, которые находятся в конце текста контента. Делается это с помощью тега:

<figure>

<img src="images/sample.png" width="320" height="175" alt="">

</figure>

<figure>

<img src="images/sample.png" width="320" height="175" alt="">

</figure>

figure {

display: inline-block;

margin: 0;

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

font-weight: 300;

}

figure img {

display: block;

border: 1px solid #fff;

outline: 1px solid #c9c9c9;

}

figure figcaption {

font-size: 16px;

font-weight: 300;

margin-top: 5px;

}

figure figcaption span {

display: block;

font-size: 14px;

color: #29c5e6;

}

section > figure + figure {

margin-left: 28px;

}

Здесь убираем стандартные отсупы у figure, отобраем его как инлайновый блок и применили нужный шрифт. Изображение отображаем как блочный элемент с белой рамкой. Вторую серую рамку можно сделать через css-свойство outline. Самое интересное находится в последнем правиле, которое задает левый отступ у всех figure кроме первого внутри тега section.

Блок «Our team»

При верстке этого блока добавим сначала заголовок:

<h2>OUR TEAM</h2>

section > h2 {

background: #29c5e6;

font: 30px 'Oswald', sans-serif;

font-weight: 300;

color: #fff;

padding: 0 10px;

margin: 30px 0 0 0;

}

А затем два блока-строки с карточками сотрудников:

<div class="team-row">

<figure>

<img src="images/sample.png" width="96" height="96" alt="">

<figcaption>John Doe<span>ceo</span></figcaption>

</figure>

<figure>

<img src="images/sample.png" width="96" height="96" alt="">

<figcaption>Saundra Pittsley<span>team leader</span></figcaption>

</figure>

…

</div>

<div class="team-row">

<figure>

<img src="images/sample.png" width="96" height="96" alt="">

<figcaption>Ericka Nobriga<span>art director</span></figcaption>

</figure>

<figure>

<img src="images/sample.png" width="96" height="96" alt="">

<figcaption>Cody Rousselle<span>senior ui designer</span></figcaption>

</figure>

…

</div>

Таким образом, карточка (figure) состоит из фотографии (img), подписи (figcaption) с именем сотрудника и его должностью (div). Карточки будут иметь следующие стили:

figure figcaption {

font-size: 16px;

font-weight: 300;

margin-top: 5px;

}

figure div {

font-size: 14px;

color: #29c5e6;

}

.team-row figure {

margin-top: 20px;

}

.team-row figure + figure {

margin-left: 43px;

}

Здесь мы задали шрифт и отступ для подписи, размер и цвет должности, добавили верхний отступ для карточек и указали, что для всех карточек в строке, кроме первой, должен быть отступ слева.

Футер

Футер состоит из четырёх больших блоков: ленты Твиттера, карты сайта, социальных ссылок и логотипа с копирайтом.

Для начала создадим контейнер футера с этим блоками:

<footer>

<div id="footer">

<div id="twitter"></div>

<div id="sitemap"></div>

<div id="social"></div>

<div id="footer-logo"></div>

</div>

</footer>

И применим к нему оформление:

footer {

background: #7e7e7e;

color: #dbdbdb;

font-size: 11px;

}

#footer {

max-width: 960px;

margin: auto;

padding: 10px 0;

height: 90px;

}

Контейнер с id=”footer” находится внутри тега footer, это даёт нам возможность через тег footer задать серую область по всей ширине экрана, а через внутренний div отцентрировать с максимальной шириной 960 пикселей. Также этот div задает обоим блокам высоту 90 пикселей.

Лента Твиттера

Верстаем содержимое ленты Твиттера:

<div id="twitter">

<h3>TWITTER FEED</h3>

<time datetime="2012-10-23"><a href="#">23 oct</a></time>

<p>

In ultricies pellentesque massa a porta. Aliquam ipsum enim, hendrerit ut porta nec, ullamcorper et nulla. In eget mi dui, sit amet scelerisque nunc. Aenean aug

</p>

</div>

Стили:

footer h3 {

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

color: #fff;

border-bottom: 1px solid #919191;

margin: 0 0 10px 0;

}

#twitter time a {

color: #b4aeae;

}

footer p {

margin: 5px 0;

}

#twitter {

float: left;

width: 300px;

}

#twitter p {

padding-right: 15px;

}

Из интересных моментов здесь следующее: подчеркивание у заголовка сделали через нижнюю границу, а сам блок твиттера, как и последующие блоки выровняем по левому краю и задаём ширину.

Карта сайта

Карта сайта представляет собой два блока со ссылками:

<div id="sitemap">

<h3>SITEMAP</h3>

<div>

<a href="/home/">Home</a>

<a href="/about/">About</a>

<a href="/services/">Services</a>

</div>

<div>

<a href="/partners/">Partners</a>

<a href="/customers/">Support</a>

<a href="/contact/">Contact</a>

</div>

</div>

Ссылкам задаем цвет и оставляем подчеркивание только для наведённых.

Колонки со ссылками делаем через инлайновые блоки и затем свойством #sitemap div + div отодвигаем вторую колонку от первой:

footer a {

color: #dbdbdb;

}

#sitemap {

width: 150px;

float: left;

margin-left: 20px;

padding-right: 15px;

}

#sitemap div {

display: inline-block;

}

#sitemap div + div {

margin-left: 40px;

}

#sitemap a {

display: block;

text-decoration: none;

font-size: 12px;

margin-bottom: 5px;

}

#sitemap a:hover {

text-decoration: underline;

}

Социальные ссылки

Вставляем набор ссылок в контейнер;

<div id="social">

<h3>SOCIAL NETWORKS</h3>

<a href="http://twitter.com/" class="social-icon twitter"></a>

<a href="http://facebook.com/" class="social-icon facebook"></a>

<a href="http://plus.google.com/" class="social-icon google-plus"></a>

<a href="http://vimeo.com/" class="social-icon-small vimeo"></a>

<a href="http://youtube.com/" class="social-icon-small youtube"></a>

<a href="http://flickr.com/" class="social-icon-small flickr"></a>

<a href="http://instagram.com/" class="social-icon-small instagram"></a>

<a href="/rss/" class="social-icon-small rss"></a>

</div>

И стилизуем их:

#social {

float: left;

margin-left: 20px;

width: 130px;

}

.social-icon {

width: 30px;

height: 30px;

background: url(../images/social.png) no-repeat;

display: inline-block;

margin-right: 10px;

}

.social-icon-small {

width: 16px;

height: 16px;

background: url(../images/social-small.png) no-repeat;

display: inline-block;

margin: 5px 6px 0 0;

}

.twitter {

background-position: 0 0;

}

.facebook {

background-position: -30px 0;

}

.google-plus {

background-position: -60px 0;

}

.vimeo {

background-position: 0 0;

}

.youtube {

background-position: -16px 0;

}

.flickr {

background-position: -32px 0;

}

.instagram {

background-position: -48px 0;

}

.rss {

background-position: -64px 0;

}

Здесь применили технику спрайтов – когда один файл с изображением применяется для разных картинок. Все ссылки разделились на большие иконки (.social-icon) и маленькие (.social-icon-small). Мы задали этим классом отображение в виде инлайнового блока с фиксированными размерами и одинаковым фоном. А затем с помощью css сдвинули этот фон так, чтобы на каждой ссылке отобразилось соответствующее изображение.

Копирайт

Блок с копирайтом и логотипом – это картинка со ссылкой и параграф с текстом под ним.

<div id="footer-logo">

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

<p>Copyright © 2012 Whitesquare. A <a href="http://pcklab.com">pcklab</a> creation</p>

</div>

Стили делают аналогично предыдущим блокам с той лишь разницей, что блок прибивается к правому краю и выравнивание внутри него так же по правому краю:

#footer-logo {

float: right;

margin-top: 20px;

font-size: 10px;

text-align: right;

}

**5. Порядок выполнения работы**

1. Получить у преподавателя вариант задания psd-макета.
2. Сверстать HTML-страницу, представленную в psd-макете.
3. В файле style.css описать все свойства для оформления страницы.
4. Подключите файл style.css к документу \*.html.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

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

2. Укажите формат правила CSS.

3. Что такое селектор?

4. Что такое псевдокласс?

5. Какие основные блоки используются для макетирования?

6. Какие виды селекторов вы знаете?

7. Чем класс отличается от идентификатора?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 9**

## **Тема работы: «Создание HTML-каркаса макета. Сброс стилей браузеров»**

**1. Цель работы**

Формирование умений производить разметку документа и выбор необходимых элементов в соответствии с макетом.

**2. Задание**

Сверстать HTML-страницу в соответствии с psd-макетом, используя средства HTML и CSS.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Каждый браузер устанавливает свои значения стилей по умолчанию для различных HTML-элементов. С помощью CSS Reset мы можем нивелировать эту разницу для обеспечения кроссбраузерности стилей.

Например, вы используете элемент «**a**» в вашем документе. Большинство браузеров, как Internet Explorer и Firefox, добавляют ссылке синий цвет и подчёркивание. Однако представьте, что через пять лет кто-то решил создать новый браузер (назовём его UltraBrowser). Разработчикам браузера не нравился синий цвет и раздражало подчёркивание, поэтому они решили выделять ссылки красным цветом и полужирным шрифтом. Именно исходя из этого, если вы установите базовое значение стилей для элемента a, то он гарантированно будет таким, каким вы хотите его видеть, а не как предпочитают его отображать разработчики UltraBrowser.

На рисунке 9.1. показан пример отображение элемента p по умолчанию в разных браузерах. В примере помещено 3 параграфа (p) без установленных стилей внутрь элемента div, у которого выставлен синий фон и оранжевая граница.

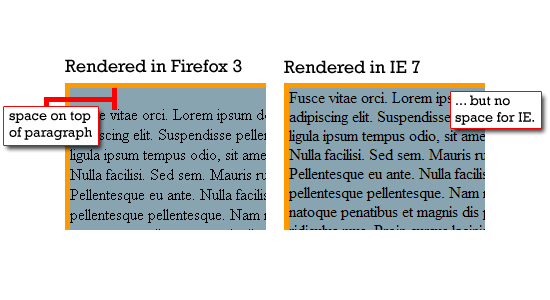


Рисунок 9.1 – Отображение элемента «p»

В качестве основы, как видите, это глобальный сброс для отступов и полей всех элементов страницы к нулю. Это наиболее часто используемый CSS сброс:

\* { padding: 0; margin: 0; }

Следующий сброс идентичен предыдущему, но также учитывает все границы по умолчанию, которые эффективно нейтрализуются до нуля:

\* { padding: 0; margin: 0; border: 0; }

Следующая версия "минималистичного" сброса похожа на два предыдущих, но дополнительно убивает стили по умолчанию для внешних границ элементов:

\* { outline: 0; padding: 0; margin: 0; border: 0; }

Сокращенный универсальный сброс – обрабатывает все необходимое, а также обеспечивает относительную универсальность браузерам в плане единообразия:

\* {

vertical-align: baseline;

font-weight: inherit;

font-family: inherit;

font-style: inherit;

font-size: 100%;

border: 0 none;

outline: 0;

padding: 0;

margin: 0;

}

Представленный ниже набор стилей CSS сбрасывает отступы и поля только для HTML и элементов body; гарантирует, что все размеры шрифтов сбрасываются, и устраняет границы у картинок в ссылках:

html, body {

padding: 0;

margin: 0;

}

html {

font-size: 1em;

}

body {

font-size: 100%;

}

a img, :link img, :visited img {

border: 0;

}

Нет такого понятия, как элемент HTML без стилей. Каждая веб-страница использует по крайней мере один CSS: стиль клиентского приложения.

Стили браузера по умолчанию могут мешать стилям, которые мы на самом деле хотим применить. Поэтому был разработан сброс CSS, чтобы обеспечить совместимую базу для всех браузеров.

Это особенно полезно для нормализации размера текста и удаления всех полей.

Вы можете найти (и скачать) HTML 5 reset.css с сайта Marksheet. Просто включите его в <head> до вашей собственной таблицы стилей:

<head>

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

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

</head>

**5. Порядок выполнения работы**

1. В заданиях к лабораторной работе №8 реализовать сброс стилей браузера.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что такое сброс стилей браузера?

2. Для чего необходим сброс стилей браузера?

3. Приведите пример сброса стилей браузера.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 10**

## **Тема работы: «Разработка CSS-документа в соответствии с макетом»**

**1. Цель работы**

Формирование умений написания таблиц стилей в строгом соответствии с макетом и навыков выбора необходимых css-свойств.

**2. Задание**

Сверстать HTML-страницу в соответствии с psd-макетом, используя средства HTML и CSS.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Свойство background-color – цвет фона, значения свойства представлены в таблице 10.1. Свойство устанавливает цветную заливку для любого HTML-элемента. Для блочных элементов цвет фона распространяется на всю ширину и высоту блока элемента, для строчных — только на область их содержимого. Не наследуется.

Таблица 10.1 – Значения свойства background-color

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| цвет | Цвет фона можно задать разными способами: [HEX](https://html5book.ru/css-shrifty/#hex), [RGB](https://html5book.ru/css-shrifty/#rgb), [RGBA](https://html5book.ru/css-shrifty/#rgba), [HSL](https://html5book.ru/css-shrifty/#hsl), [HSLA](https://html5book.ru/css-shrifty/#hsla), а также с помощью [140 кроссбраузерных названий цветов](https://html5book.ru/css-colors/). |
| transparent | Значение по умолчанию. Задает прозрачный фон для элемента. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис:

div {background-color: papayawhip;}

h1 {background-color: palevioletred;}

p {background-color: peachpuff;}

span {background-color: peru;}

Фоновое изображение background-image, значения свойства представлены в таблице 10.2. Свойство устанавливает изображение в качестве фона для элемента. Фоновым изображением может быть картинка или градиент, который задаётся с помощью функций background-image: linear-gradient(), background-image: radial-gradient() или с помощью функций повтора градиента background-image: repeating-linear-gradient(), background-image: repeating-radial-gradient(). Не наследуется.

Таблица 10.2 – Значения свойства background-image

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| url('URL') | Абсолютный или относительный адрес изображения. |
| None | Значение по умолчанию, также удаляет изображение у элемента из группы элементов, для которых установлено фоновое изображение. |
| Initial | Устанавливает значение свойства в значение по умолчанию. |
| Inherit | Наследует значение свойства от родительского элемента. |

Синтаксис:

div {background-image: url(https://html5book.ru/images/flower.png);}

table {background-image: radial-gradient(farthest-corner at 50% 50%, white, #DCECF8);}

p {background-image: linear-gradient(to top left, white, lightblue);}

Повтор фоновых изображений background-repeat, значения свойства представлены в таблице 10.3. Свойство определяет, каким образом будет повторяться фоновый рисунок. Не наследуется.

Таблица 10.3 – Значения свойства background-repeat

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| repeat | Весь фон страницы будет заполнен фоновым рисунком. Если при этом задать background-position, то повтор будет осуществляться с указанной позиции. Значение по умолчанию. |
| no-repeat | Фоновое изображение не будет повторяться. |
| repeat-x | Фоновый рисунок повторяется от левого до правого края веб-страницы по верхнему краю страницы. |
| repeat-y | Фоновый рисунок повторяется от верхнего до нижнего края веб-страницы по левому краю страницы. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Синтаксис:

div {background-repeat: no-repeat;}

div {background-repeat: repeat-x;}

div {background-repeat: repeat-y;}

div {background-repeat: repeat;}

Пример реализации свойства background-repeat представлены на рисунке 10.1

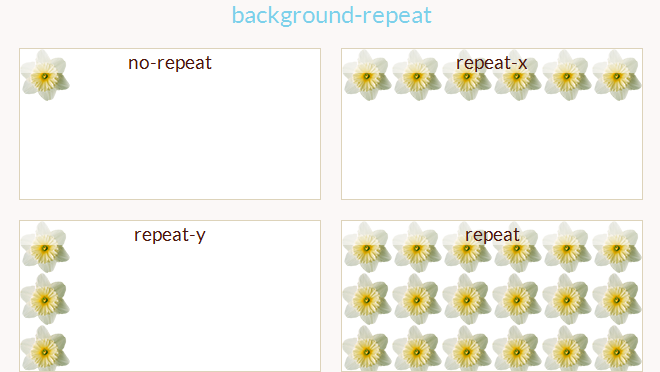


Рисунок 10.1 – Реализация свойства background-repeat

Размер изображения background-size. Свойство позволяет масштабировать фоновое изображение по вертикали и горизонтали (background-image). Оно описывает, как изображение будет растягиваться и обрезаться, чтобы полностью закрыть собой фоновую область. С помощью этого свойства изображение также можно уменьшать по ширине и по высоте. Не наследуется.

Синтаксис:

div {background-size: 300px 150px;}

div {background-size: 50% 30%;}

div {background-size: cover;}

div {background-size: contain;}

Задание фона элемента одним свойством background. Свойство позволяет описать в одном объявлении следующие свойства фона: background-color, background-image, background-position, background-size, background-repeat, background-origin, background-clip и background-attachment. Необязательно указывать все перечисленные свойства, если какое-либо свойство будет пропущено, оно примет значение по-умолчанию.

Если вы указываете в краткой записи фона свойство background-size, то его значения нужно будет записать через слеш /, чтобы отделить его от свойства background-position.

Синтаксис:

body {

background: #00ff00 url("smiley.gif") no-repeat fixed center;

}

Семейство шрифтов font-family. Свойство используется для выбора начертания шрифта. Поскольку невозможно предсказать, установлен тот или иной шрифт на компьютере посетителя вашего сайта, рекомендуется прописывать все возможные варианты однотипных шрифтов. В таком случае браузер будет проверять их наличие, последовательно перебирая предложенные варианты. Наследуется.

Важно! Если в названии шрифта имеются пробелы или символы (например, #, $, %), то оно заключается в кавычки. Это делается для того, чтобы браузер мог понять, где начинается и заканчивается название шрифта.

Синтаксис:

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

Стиль начертания шрифта font-style, значения свойства представлены в таблице 10.4. Свойство позволяет выбрать стиль начертания для текста. При этом разница между курсивом и наклонным текстом заключается в том, что курсивное начертание вносит небольшие изменения в структуру каждой буквы, а наклонный текст представляет собой наклонную версию прямого текста. Наследуется.

Таблица 10.4 – Значения свойства font-style

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| normal | Значение по умолчанию, устанавливает для текста обычное начертание шрифта. |
| Italic | Выделяет текст курсивом. |
| oblique | Устанавливает наклонное начертание шрифта. |
| Initial | Устанавливает значение свойства в значение по умолчанию. |
| Inherit | Наследует значение свойства от родительского элемента. |

Синтаксис:

h1 {font-style: normal;}

h1 {font-style: italic;}

h1 {font-style: oblique;}

Насыщенность шрифта font-weight, значения свойства представлены в таблице 10.5.

Синтаксис:

h1 {font-weight: normal;}

span {font-weight: bold;}

span {font-weight: bolder;}

span {font-weight: lighter;}

h1 {font-weight: 100;}

Таблица 10.5 – Значения свойства font- weight

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| normal | Значение по умолчанию, устанавливает нормальную насыщенность шрифта. Эквивалентно значению насыщенности, равной 400. |
| bold | Делает шрифт текста полужирным. Эквивалентно значению насыщенности, равной 700. |
| bolder | Насыщенность шрифта будет больше, чем у предка. |
| lighter | Насыщенность шрифта будет меньше, чем у предка. |
| 100, 200, 300, 400, 500, 600, 700, 800, 900 | Значение 100 соответствует самому легкому варианту начертания шрифта, а 900 — самому плотному. При этом, эти числа не определяют конкретной плотности, т.е. 100, 200, 300 и 400 могут соответствовать одному и тому же варианту слабой насыщенности начертания шрифта; 500 и 600 — средней насыщенности, а 700, 800 и 900 могут выводить одинаковое очень насыщенное начертание. Распределение плотности так же зависит от количества уровней насыщенности, определенных в конкретном семействе шрифтов. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Краткая запись свойств шрифта. Свойство font объединяет все свойства шрифта, используемого на веб-странице. Если явное значение какого-либо свойства не нужно, то оно опускается:

font: font-style font-variant font-weight font-size/line-height font-family

Например:

p {font: italic bold 14px/1.5 Times, 'New Century Schoolbook', serif;}

**5. Порядок выполнения работы**

1. Получить макет у преподавателя.
2. Сверстать HTML-страницу, представленную в psd-макете, используя свойства background-color, background-image, background-size, background-repeat, font.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Перечислите свойства для работы с шрифтами. Дайте их характеристику.

2. Перечислите свойства для работы с фоновым изображением. Дайте их характеристику.

3. Перечислите свойства для работы с фоном. Дайте их характеристику.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 11**

## **Тема работы: «Тестирование кроссбраузерного отображения верстки. Размещение проекта на хостинге»**

**1. Цель работы**

Формирование умений тестирования кроссбраузерности верстки, устранения недостатков, а так же размещения проекта в сети.

**2. Задание**

Сверстать HTML-страницу в соответствии с правилами кроссбраузерной верстки.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Среди всего разнообразия существующих на данный момент браузеров таких как Opera, Safari, Internet Explorer, Google Chrome, Mozila Firefox один и тот же сайт, с тем же самым шаблоном, стилями и движком может отображаться в разных браузерах по-разному.

Кроссбраузерная html и css верстка – это верстка сайта, нацеленная на создания сайта, отображаемого в каждом браузере одинаково корректно.

Причины проблем с отображением сайта:

1. Верстка не соответствует некоторым стандартам HTML.

2. Постоянно обновляются и появляются новые библиотеки и правила CSS. Они прекрасно воспроизводятся обновленными версиями браузеров, но могут некорректно отображаться старыми версиями.

3. Internet Explorer. Самый странный браузер, который не понимает половину всех правил и поэтому постоянно придумываются новые хаки, чтобы обойти эту проблему IE.

4. Значение атрибутов по умолчанию. В каждом браузере это значение может быть разным. Атрибуты – это цвета элементов, шрифты, их размер. Все это должно быть опи-сано в таблице стилей, в противном случае браузер будет самостоятельно проставлять значения, указанные во всех из них по умолчанию. Поэтому в каждом из них сайт будут отображаться по-разному.

Рекомендации по выполнению кроссбраузерной верстки:

1. Значение атрибутов по умолчанию в разных браузерах может быть разным. Под атрибутами имеется в виду цвет элемента, расстояние между элементами, размер шрифта и т.д. Если все эти и прочие параметры не прописать в таблице стилей, то браузер сам начнет подставлять значения, которые указаны у него по умолчанию. Решить проблему с атрибутами по умолчанию можно путем описания этих атрибутов в таблице стилей, чтобы браузеры не додумывали и не подставляли свои значения.

Пример:

html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym, address, big, cite, code, del, dfn, em, font, img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt, var, dl, dt, dd, ol, ul, li, fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td

{

margin: 0;

padding: 0;

border: 0;

outline: 0;

font-weight: inherit;

font-style: inherit;

font-size: 14px;

font-family: inherit;

vertical-align: top;

background: transparent;

font: verdana, geneva, lucida, 'lucida grande', arial, helvetica, sans-serif;

background-color : white;

}

2. Установить на компьютер все самые популярные браузеры. Можно так же вос-пользоваться программой IE Tester, которая позволяет проверять сайт во всем множестве версий IE.

3. Необходимо определить для себя стратегию верстки: как будет удобнее: верстать сразу весь сайт под один браузер, а после погонять его под остальные либо же верстать постепенно и каждый элемент проверять во всех браузерах, при необходимости корректируя.

4. Использование плагинов позволяет быстрее находить ошибки кода для каждого из браузеров. Как пример можно потестировать эти плагины:

- Chrome на сайте getfirebug.com;

- Opera на официальном сайте Opera;

- Mozila на сайте addons mozila;

- IE на оф. сайте Microsoft;

5. Для современных браузеров, но не обновленных (так сказать устаревших) следует использовать префиксы поставщиков браузеров. Это помогает разработчикам браузеров реализовывать новые CSS3-модули:

.round{

-khtml-border-radius: 10px; /\* Konqueror \*/

-rim-border-radius: 10px; /\* RIM \*/

-ms-border-radius: 10px; /\* Microsoft \*/

-o-border-radius: 10px; /\* Opera \*/

-moz-border-radius: 10px; /\* Mozilla (например, Firefox) \*/

-webkit-border-radius: 10px; /\* Webkit (например, Safari и Chrome) \*/

-border-radius: 10px; /\* W3C \*/

}

В этом примере используются свойства с префиксами поставщиков (причем это далеко не исчерпывающий список), каждое из которых обладает собственным уникальным префиксом, например, webkit означает основанные на WebKit браузеры, или ms – префикс такого поставщика, как Microsoft, подразумевающий Internet Explorer, и т. д. Из-за особенностей работы CSS браузер будет двигаться вниз по таблице стилей строка за строкой, применяя соответствующие свойства и игнорируя те, которые не может распознать.

**5. Порядок выполнения работы**

1. Добаботать файл style.css для psd-макета из лаболаторной работы №8, используя правила кроссбраузерной верстки.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой кроссбраузерная верстка?

2. Что такое префикс?

3. В чем заключается правило использования префиксов?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 12**

## **Тема работы: «Создание анимации средствами CSS»**

**1. Цель работы**

Формирование умений создания анимации средствами CSS.

**2. Задание**

Создать HTML-документ, содержащий эффекты и анимации, используя средства CSS.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

С помощью CSS3 анимации можно создавать эффекты, которые раньше воспроизводились с помощью скрипта и с использованием графических редакторов.

Анимация применяется к любым html-элементам, а также к псевдоэлементам **:before** и **:after**.

**CSS transitions** и **CSS animations** позволяют реализовать анимацию средствами CSS, без привлечения JavaScript.

**CSS animations** позволяют анимировать переходы от одних конфигурации CSS стилей к другим. CSS анимации состоят из двух компонентов: стилевое описание анимации и набор ключей (@keyframes), определяющих начальное, конечное и возможно промежуточное состояние анимируемых стилей.

Пример стилевого описания элемента:

h1 {

font-size: 3.5em;

color: darkmagenta;

-webkit-animation: shadow 2s infinite ease-in-out;

**animation: shadow 2s infinite ease-in-out;**

}

Правило @keyframes:

@keyframes shadow {

from {text-shadow: 0 0 3px black;}

50% {text-shadow: 0 0 300px black;}

to {text-shadow: 0 0 3px black;}

}

Сначала идет объявление @keyframes, которое затем используется в свойстве animation элемента.

Правило@keyframes позволяет создавать анимацию с помощью ключевых кадров – состояний объекта в определенный момент времени. Каждый ключ описывает, где должен находиться анимированный элемент в данный момент.

Синтаксис:

@keyframes имя анимации {ключевые кадры {css-стили;}}

Ключевые кадры анимации создаются с помощью ключевых слов **from** и **to** (эквивалентны значениям 0% и 100%) или с помощью процентных пунктов, которых можно задавать сколько угодно. Также можно комбинировать ключевые слова и процентные пункты.

Если 0% или 100% кадры не указаны, то браузер пользователя создает их, используя вычисляемые (первоначально заданные) значения анимируемого свойства.

После объявления @keyframes, можно ссылаться на него в свойстве animation:

h1 {

font-size: 3.5em;

color: darkmagenta;

-webkit-animation: shadow  2s infinite ease-in-out;

animation: shadow 2s infinite ease-in-out;

}

Чтобы создать CSS анимацию, необходимо добавить в стиль элемента, который нужно анимировать, свойство animation или его под-свойства. Это позволит настроить ускорение и продолжительность анимации, а также другие детали о том, как анимация должна протекать.

Свойство animation имеет следующие под-свойства:

1) animation-**delay** – настраивает задержку между временем загрузки элемента и временем начала анимации. Может принимать следующие значения:

- время – задержка анимации задается в секундах **s** или миллисекундах **ms**. Значение по умолчанию **0**;

- initial – устанавлмвает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

2) animation-**direction** – дает возможность при каждом повторе анимации идти по алтернативному пути, либо сбросить все значения и повторить анимацию. Может принимать следующие значения:

- alternate – анимация проигрывается с начала до конца, затем в обратном направлении;

- alternate-reverse – анимация проигрывается с конца до начала, затем в обратном направлении;

- normal – значение по умолчанию, анимация проигрывается в обычном направлении, с начала и до конца;

- reverse – анимация проигрывается с конца;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

3) animation-**duration** – настраивает время, в течение которого должен пройти один цикл анимации. Может принимать следующие значения:

- время – длительность анимации задается в секундах **s** или миллисекундах **ms**;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

4) animation-**iteration-count** – настраивает количество повторении анимации. Может принимать следующие значения:

- число – с помощью целого числа задается количество повторов анимации. Значение по умолчанию **1**;

- infinite – анимация проигрывается бесконечно;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

5) animation-**name** – определяет имя [@keyframes](https://developer.mozilla.org/ru/docs/Web/CSS/@keyframes), настраивающего кадры анимации. Может принимать следующие значения:

- имя анимации – имя анимации, которое связывает правило@keyframes с селектором;

- none – значение по умолчанию, означает отсутствие анимации. Также используется, чтобы отменить анимацию элемента из группы элементов, для которых задана анимация;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

6) animation-**play-state** – позволяет приостановить и возобновить анимацию. Может принимать следующие значения:

- paused – останавливает анимацию;

- running – значение по умолчанию, означает проигрывание анимации;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

7) animation-**timing-function** – настраивает ускорение анимации. Может принимать следующие значения:

- ease – функция по умолчанию, анимация начинается медленно, разгоняется быстро и замедляется в конце. Соответствует cubic-bezier(0.25,0.1,0.25,1);

- linear – анимация происходит равномерно на протяжении всего времени, без колебаний в скорости. Соответствует cubic-bezier(0,0,1,1);

- ease-in – анимация начинается медленно, а затем плавно ускоряется в конце. Соответствует cubic-bezier(0.42,0,1,1);

- ease-out – анимация начинается быстро и плавно замедляется в конце. Соответствует cubic-bezier(0,0,0.58,1);

- ease-in-out – анимация медленно начинается и медленно заканчивается. Соответствует cubic-bezier(0.42,0,0.58,1);

- cubic-bezier(x1, y1, x2, y2) – позволяет вручную установить значения от 0 до 1;

- step-start – задает пошаговую анимацию, разбивая анимацию на отрезки, изменения происходят в начале каждого шага. Эквивалентно steps(1, start);

- step-end – пошаговая анимация, изменения происходят в конце каждого шага. Эквивалентно steps(1, end);

- steps(количество шагов,start|end) – ступенчатая временная функция, которая принимает два параметра. Количество шагов задается целым положительным числом. Второй параметр необязательный, указывает момент, в котором начинается анимация. Со значением start анимация начинается в начале каждого шага, со значением end – в конце каждого шага с задержкой. Задержка вычисляется как результат деления времени анимации на количество шагов. Если второй параметр не указан, используется значение по умолчанию end;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

8) animation-**fill-mode** – настраивает значения используемые анимацией до и после исполнения. Может принимать следующие значения:

- none – значение по умолчанию. Состояние элемента не меняется до или после воспроизведения анимации;

- forwards – воспроизводит анимацию до последнего кадра по окончанию последнего повтора и не отматывает ее к первоначальному состоянию;

- backwards – возвращает состояние элемента после загрузки страницы к первому кадру, даже если установлена задержка animation-delay, и оставляет его там, пока не начнется анимация;

- both – позволяет оставлять элемент в первом ключевом кадре до начала анимации (игнорируя положительное значение задержки) и задерживать на последнем кадре до конца последней анимации;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

В то время как временные характеристики (продолжительность анимации) указываются в стилях для анимируемого элемента, ключи используют [percentage](https://developer.mozilla.org/ru/docs/Web/CSS/percentage) чтобы определить стадию протекания анимации. 0% означает начало анимации, а 100% конец. Так как эти значения очень важны, то для них придумали специальные слова: from и to.

Также можно добавить ключи, характеризующие промежуточное состояние анимации.

Пример анимирует скольжение текста в элементе [<p>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/p) от правого края окна браузера:

p {

animation-duration: 10s;

animation-name: slidein;

animation-iteration-count:infinite

}

@keyframes slidein {

from {

margin-left: 100%;

width: 300%;

}

to {

margin-left: 0%;

width: 100%;

}

}

Обратите внимание, что анимация может сделать страницу шире чем окно браузера. Этого можно избежать поместив элемент который будет анимироваться в контейнер и установить ему свойство [overflow](https://developer.mozilla.org/ru/docs/Web/CSS/overflow):hidden.

В стиле для элемента [<p>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/p) с помощью свойства [animation-duration](https://developer.mozilla.org/ru/docs/Web/CSS/animation-duration) указано, что исполнение анимации от начала до конца должно занять 10с., и что имя для [@keyframes](https://developer.mozilla.org/ru/docs/Web/CSS/@keyframes), описывающей саму анимацию это "slidein".

В элемент [<p>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/p) можно добавлять и другие пользовательские стили чтобы как-то украсить его, однако здесь мы хотели продемонстрировать только эффект анимации.

Добавим другие ключи в предыдущий пример, чтобы размер шрифта заголовка возрастал на время по мере продвижения влево, а потом возвращался к первоначальному значению . Это легко реализовать с помощью следующего ключа:

75% {

font-size: 300%;

margin-left: 25%;

width: 150%;

}

Текст при каждом повторении снова "запрыгивает" за край окна браузера. Сделаем, чтобы текст двигался влево и вправо. Этого легко достичь посредством установки свойству [animation-direction](https://developer.mozilla.org/ru/docs/Web/CSS/animation-direction) значения alternate:

p {

animation-duration: 3s;

animation-name: slidein;

animation-iteration-count: infinite;

animation-direction: alternate;

}

Пример:

@keyframes rotate {

0% {

transform: rotate(0deg);

}

50% {

transform: rotate(180deg);

}

100% {

transform: rotate(360deg);

}

}

.wood-wheel {

animation-name: rotate;

animation-duration: 2s;

}

Для одного элемента можно задавать несколько анимаций, перечислив их названия через запятую:

div {animation: shadow 1s ease-in-out 0.5s alternate, move 5s linear 2s;}

**CSS transitions** – универсальное свойство, которое определяет эффект перехода между двумя состояниями элемента, они могут быть установлены ко всем элементам, а также с помощью псевдоэлемента **:hover** или **:active**, а также динамически через JavaScript. Наиболее часто используются для стилизации ссылок и кнопок в состоянии **:hover** или **:focus**.

Свойство transition имеет следующие под-свойства:

1) transition-**delay** – устанавливает время ожидания перед запуском эффекта анимации перехода. Не наследуется. Может принимать следующие значения:

- время – время задержки перехода указывается в секундах или миллисекундах;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента;

2) transition-**duration** – задаёт время в секундах или миллисекундах, сколько должна длиться анимация перехода до её завершения. Может принимать следующие значения:

- время – время перехода указывается в секундах или миллисекундах, например, 2s или 5ms;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

Пример:

div {

-webkit-transition-duration: 5s;

transition-duration: 5s;

}

Пример:

<div class="wrap">

<h1>Наведите на блок</h1>

<kbd>transition-duration</kbd>

<div class="container">

<span>0.5s</span>

<div class="box1 box"></div>

</div>

<div class="container">

<span>1s</span>

<div class="box2 box"></div>

</div>

<div class="container">

<span>2s</span>

<div class="box3 box"></div>

</div>

</div>

body {font-family: 'Playfair Display', serif; margin: 0;}

.wrap {text-align: center;}

.container {

display: inline-block;

}

h1 {

color: #3A262F;

font-weight:normal;

text-align: center;

}

kbd{display:block;text-align:center;margin-bottom:20px;}

.box {

height: 40px;

width: 40px;

border-radius: 50%;

margin: 20px 50px 0;

}

.wrap:hover .box {

-webkit-transform: scale(2);

-ms-ransform: scale(2);

transform: scale(2);

}

.box1 {

background: #FFD592;

transition: .5s;

}

.box2 {

background: #F9A88B;

transition: 1s;

}

.box3 {

background: #B39190;

transition: 2s;

3) transition-**property** – устанавливает имя стилевого свойства, значение которого будет отслеживаться для создания эффекта перехода. Может принимать следующие значения:

- none – отсутствие свойства для перехода;

- all – значение по умолчанию. Применяет эффект перехода ко всем свойствам элемента;

- свойство – определяет список css-свойств, перечисленных через запятую, участвующих в переходе;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

Пример:

div {

width: 100px;

-webkit-transition-property: width;

transition-property: width;

}

div:hover {

width: 300px;

}

4) transition-**timing-function** – устанавливает, насколько быстро должно изменяться значение стилевого свойство для которого применяется эффект перехода. Если определено более одного перехода для элемент, например, цвет фона элемента и его положение, можно использовать разные функции для каждого свойства. Не наследуется. Может принимать следующие значения:

- ease – функция по умолчанию, переход начинается медленно, разгоняется быстро и замедляется в конце. Соответствуетcubic-bezier(0.25,0.1,0.25,1);

- linear – переход происходит равномерно на протяжении всего времени, без колебаний в скорости. Соответствует cubic-bezier(0,0,1,1);

- ease-in – переход начинается медленно, а затем плавно ускоряется в конце. Соответствует cubic-bezier(0.42,0,1,1);

- ease-out – переход начинается быстро и плавно замедляется в конце. Соответствует cubic-bezier(0,0,0.58,1);

- ease-in-out – переход медленно начинается и медленно заканчивается. Соответствует cubic-bezier(0.42,0,0.58,1);

- cubic-bezier(x1, y1, x2, y2) – позволяет вручную установить значения от 0 до 1 для кривой ускорения;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

Пример:

div {

-webkit-transition-timing-function: linear;

transition-timing-function: linear;

}

Пример:

<article id="go">

<a href="#go"><span class="fa fa-power-off"></span>Запустить</a>

<a href="#reset"><span class="fa fa-refresh"></span>Назад</a>

<div><kbd>transition-timing-function</kbd></div>

<p><kbd>linear</kbd></p>

<p><kbd>ease</kbd></p>

<p><kbd>ease-in</kbd></p>

<p><kbd>ease-out</kbd></p>

<p><kbd>ease-in-out</kbd></p>

<p><kbd>cubic-bezier</kbd></p>

</article>

body {font-family: 'Playfair Display', serif; margin: 0;line-height:1;color:#696150;}

article{max-width:660px;margin: 30px auto 0;}

p {

width: 125px;

padding: 10px 0;

text-align: center;

border-radius: 4px;

transition-duration: 3s;

}

a {

text-decoration: none;

display: inline-block;

margin-right: 20px;

color:#696150;

}

.fa {padding-right: 10px}

.fa.fa-power-off {color:#FF9E3B}

.fa.fa-refresh {color:#B7CCCC}

div{margin-top:15px;}

p:nth-of-type(1) {

background: #F06E58;

transition-timing-function: linear;}

p:nth-of-type(2) {

background: #FEA45D;

transition-timing-function: ease;}

p:nth-of-type(3) {

background: #FDF569;

transition-timing-function: ease-in;}

p:nth-of-type(4) {

background: #B9EA7B;

transition-timing-function: ease-out;}

p:nth-of-type(5) {

background: #B9ECFE;

transition-timing-function: ease-in-out;}

p:nth-of-type(6) {

background: #82DDFF;

transition-timing-function: cubic-bezier(0.6, 0.1, 0.15, 0.8);}

#go:target p{transform:translateX(535px)}

Свойство transition позволяет одновременно задать значения transition-property,transition-duration, transition-timing-function и transition-delay.

Синтаксис:

transition: <переход> [, <переход> ]\*

где <переход> = [ none | <[transition-property](http://htmlbook.ru/css/transition-property)> ] || <[transition-duration](http://htmlbook.ru/css/transition-duration)> || <[transition-timing-function](http://htmlbook.ru/css/transition-timing-function)> || <[transition-delay](http://htmlbook.ru/css/transition-delay)>;

none - отменяет эффект перехода.

Для задания всех свойств перехода обычно используют краткую запись свойства transition:

transition: **all** 0.5s ease-in-out;

Если воспользоваться значениями по умолчанию, то запись

div {transition: 1s;}

будет эквивалентна

div {transition: all 1s ease 0s;}

Для элемента можно задать несколько последовательных переходов, перечислив их через запятую. Каждый переход можно оформить своей временной функцией.

div {transition: background 0.3s ease, color 0.2s linear;}

или

div {

transition-property: height, width, background-color;

transition-duration: 3s;

transition-timing-function: ease-in, ease, linear;

CSS3 трансформации изменяют размер, форму и положение элемента на веб-странице с помощью свойства **transform**. Трансформации преобразовывают элемент, не затрагивая остальные элементы веб-страницы, т.е. другие элементы не сдвигаются относительно него. По умолчанию трансформация происходит относительно центра элемента. Трансформации не действуют на строчные элементы display: inline.

Существуют два вида CSS3 трансформаций – 2D и 3D.

2D-трансформации преобразовывают элементы в двухмерном пространстве.

Функции 2D-трансформации transform. Свойство задаёт вид преобразования элемента. Свойство описывается с помощью функций трансформации, которые смещают элемент относительно его текущего положения на странице или изменяют его первоначальные размеры и форму.

В таблице 12.1 приведено описание функций трансформации.

Таблица 12.1 – Описание функций трансформации

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| **1** | **2** |
| None | Значение по умолчанию, означает отсутствие трансформации. Также отменяет трансформацию для элемента из группы трансформируемых элементов. |
| matrix(a, c, b, d, x, y) | Смещает элементы и задает способ их трансформации, позволяя объединить несколько функций 2D-трансформаций в одной. В качестве трансформации допустимы поворот, масштабирование, наклон и изменение положения.  Значение a изменяет масштаб по горизонтали. Значение от 0 до 1 уменьшает элемент, больше 1 — увеличивает.  Значение c деформирует (сдвигает) стороны элемента по оси Y, положительное значение — вверх, отрицательное — вниз.  Значение b деформирует (сдвигает) стороны элемента по оси X, положительное значение — влево, отрицательное — вправо.  Значение d изменяет масштаб по вертикали. Значение меньше 1 уменьшает элемент, больше 1 — увеличивает.  Значение x смещает элемент по оси X, положительное — вправо, отрицательное — влево.  Значение y смещает элемент по оси Y, положительное значение — вниз, отрицательное — вверх. |
| translate(x,y) | Сдвигает элемент на новое место, перемещая относительно обычного положения вправо и вниз, используя координаты x и y, не затрагивая при этом соседние элементы. Если нужно сдвинуть элемент влево или вверх, то нужно использовать отрицательные значения. |
| **1** | **2** |
| translateX(n) | Сдвигает элемент относительно его обычного положения по оси X. |
| translateY(n) | Сдвигает элемент относительно его обычного положения по оси Y. |
| scale(x,y) | Масштабирует элементы, делая их больше или меньше. Значения от 0 до 1 уменьшают элемент. Первое значение масштабирует элемент по ширине, второе — по высоте. Отрицательные значения отображают элемент зеркально. |
| scaleX(n) | Функция масштабирует элемент по ширине, делая его шире или уже. Если значение больше единицы, элемент становится шире, если значение находится между единицей и нулем, элемент становится уже. Отрицательные значения отображают элемент зеркально по горизонтали. |
| scaleY(n) | Функция масштабирует элемент по высоте, делая его выше или ниже. Если значение больше единицы, элемент становится ниже, если значение находится между единицей и нулем — ниже. Отрицательные значения отображают элемент зеркально по вертикали. |
| rotate(угол) | Поворачивает элементы на заданное количество градусов, отрицательные значения от -1deg до-360deg поворачивают элемент против часовой стрелки, положительные — по часовой стрелке. Значение rotate(720deg) поворачивает элемент на два полных оборота. |
| skew(x-угол,y-угол) | Используется для деформирования (искажения) сторон элемента относительно координатных осей. Если указано одно значение, второе будет определено браузером автоматически. |
| skewX(угол) | Деформирует стороны элемента относительно оси X. |
| skewY(угол) | Деформирует стороны элемента относительно оси Y. |
| Initial | Устанавливает значение свойства в значение по умолчанию. |
| Inherit | Наследует значение свойства от родительского элемента. |

Допустимые значения:

- matrix() – любое число;

- translate(), translateX(), translateY() – единицы длины (положительные и отрицательные), %;

- scale(), scaleX(), scaleY()– любое число;

- rotate()  – угол (deg, grad, rad или turn);

- skew(), skewX(), skewY()– угол (deg, grad, rad);

Пример:

<!doctype html>

<head>

<meta charset="windows-1251" />

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />

<title>Анимация CSS3</title>

<link type="text/css" rel="stylesheet" href="style/tanime2.css" media="all" />

</head>

<body><h1>CSS3 2D Transform examples (on <kbd>:hover</kbd>)</h1>

<br>

<div class="wrap"><div class="one"><kbd>translateX(20px)</kbd></div></div>

</body>

</html>

файл .css

body {font-family: 'Playfair Display', serif; margin: 0;text-align: center}

h1 {font-weight: normal;color: #6A5953}

kbd {font-size: 0.9em;display:inline-block;line-height:1.1;}

.wrap {

display: inline-block;

margin: 0 40px 2em 0;

background: rgba(228, 225, 228, .5);}

div{

width: 170px;

height: 100px;

line-height: 100px;

margin: 0 auto;

-o-transition: all 0.5s ease-in-out;

-moz-transition: all 0.5s ease-in-out;

-webkit-transition: all 0.5s ease-in-out;

transition: all 2s ease-in-out;

}

.one {

background: rgba(135, 86, 120, .4);

}

.one:hover {

-o-transform: translateX(20px);

-ms-transform: translateX(20px);

-moz-transform: translateX(20px);

-webkit-transform: translateX(20px);

transform: translateX(20px);

}

Пример:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>transition</title>

<style>

#bar {

top:-5.5em; right:5em; /\* Положение \*/

padding: .5em; /\* Поля \*/

margin: 0; /\* Отступы \*/

position: absolute; /\* Абсолютное позиционирование \*/

width: 2em; /\* Ширина \*/

background: #333; /\* Цвет фона \*/

color: #fff; /\* Цвет текста \*/

text-align: center; /\* Выравнивание по центру \*/

/\* Переход \*/

-webkit-transition: top 1s ease-out 0.5s;

-moz-transition: top 1s ease-out 0.5s;

-o-transition: top 1s ease-out 0.5s;

transition: top 1s ease-out 0.5s;

}

#bar:hover { top: 0; }

</style>

</head>

<body>

<ul id="bar">

<li>1</li><li>2</li>

<li>3</li><li>4</li>

<li>&darr;</li>

</ul>

</body>

</html>

В данном примере при наведении курсора на стрелку, раскрывается блок с числами.

Точка трансформации – **transform-origin** – свойство позволяет сместить центр трансформации, относительно которого происходит изменение положения/размера/формы элемента. Значение по умолчанию – center, или 50% 50%. Задаётся только для трансформированных элементов. Не наследуется. Может принимать следующие значения:

- ось Х(left,center,right, длина, %), ось Y(top,center,bottom, длина, %) – пара значений, заданная с помощью ключевых слов, единиц длины или процентов определяет, относительно какой части элемента будет происходить трансформация. Значения больше 100% увеличивают область трансформации элемента;

- initial – устанавливает значение свойства в значение по умолчанию;

- inherit – наследует значение свойства от родительского элемента.

div {

-webkit-transform: rotate(45deg);

-ms-transform: rotate(45deg);

transform: rotate(45deg);

-webkit-transform-origin: 20% 40%;

-ms-transform-origin: 20% 40%;

transform-origin: 20% 40%;

}

Пример:

div{

width: 170px;

height: 100px;

line-height: 100px;

margin: 0 auto;

-o-transition: all 0.5s ease-in-out;

-moz-transition: all 0.5s ease-in-out;

-webkit-transition: all 0.5s ease-in-out;

transition: all 2s ease-in-out;

}

div:hover {

transform: rotate(45deg);

transform-origin: 50% 50%;}

}

Множественные трансформации заключаются в объединении нескольких трансформаций одного элемента, перечислив их через пробел в порядке проявления.

div {transform: scale(1.5) rotate(-10deg);}

Пример:

div:hover {

transform: scale(1.5) rotate(-10deg);}

transform-origin: 0% 0%;}

**Transform-style** – определяет, как дочерние элементы должны отображаться в 3D-пространстве. Это свойство должно использоваться совместно с [transform](http://htmlbook.ru/css/transform).

Синтаксис:

transform-style: flat | preserve-3d

где flat – дочерние элементы лежат в той же плоскости, что и их родитель;

preserve-3d – дочерние элементы будут отображаться в 3D-пространстве.

**5. Порядок выполнения работы**

1. Создать HTML страницу с использованием CSS-анимации, добавив 3 круга которые будут передвигаться друг за другом от правого края до левого края экрана.
2. Создать HTML страницу с использованием CSS-анимации, добавив 4 круга которые будут передвигаться друг за другом по кругу по часовой стрелке.
3. Создать HTML страницу с кнопками для навигации, как показано на рисунке 12.1. Добавить эффект анимации для ссылок в соответствии с вариантом.

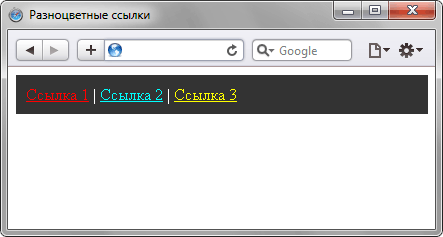


Рисунок 12.1 – Кнопки навигации

1. Выполнить задание, соответствующее вашему варианту.

Вариант 1 Создать часы с секундной и минутной стрелкой и, используя анимацию CSS, осуществите движение стрелок.

При наведении на ссылку она должна увеличиваться и сдвигаться вверх, а при нажатии на ссылку она должна уменьшиться по ширине.

Вариант 2 Создать часы с секундной стрелкой и, используя анимацию CSS, осуществите движение стрелки.

При наведении на ссылку она должна увеличиваться и сдвигаться вправо, а при нажатии на ссылку она должна уменьшиться по высоте.

Вариант 3 Создать часы с секундной и минутной стрелкой и, используя анимацию CSS, осуществите движение стрелок. При выведении курсора из области часов стрелки должны останавливаться.

При наведении на ссылку она должна увеличиваться и сдвигаться вниз.

Вариант 4 Создать часы с секундной и минутной стрелкой и, используя анимацию CSS, осуществите движение стрелок. При нажатии на область часов стрелки должны останавливаться

При наведении на ссылку она должна отображаться зеркально, при этом необходимо соблюдать очередность: 1-ая и 3-я ссылки зеркально по горизонтали, 2-ая зеркально по вертикали.

Вариант 5 Создать часы с секундной стрелкой и, используя анимацию CSS, осуществите движение стрелки в обратном порядке

При наведении на ссылку она должна увеличиваться и делать один полный поворот по часовой стрелке.

Вариант 6 Создать часы с секундной стрелкой и, используя анимацию CSS, осуществите движение стрелки. При выведении курсора из области часов стрелки должны останавливаться

При наведении на ссылку она должна увеличиваться, при этом ее необходимо деформировать (исказить стороны) на 300.

Вариант 7 Создать часы с секундной стрелкой и, используя анимацию CSS, осуществите движение стрелки в обратном порядке. При выведении курсора из области часов стрелка должны останавливаться.

При наведении на ссылку она должна она должна выполнить вращение.

Вариант 8 Создать часы с секундной стрелкой и, используя анимацию CSS, осуществите движение стрелки. При нажатии на область часов стрелка должны останавливаться

При наведении на ссылку она должна увеличиваться и делать один полный поворот по часовой стрелке относительно верхнего правого угла.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Перечислите способы создания анимации средствами CSS. Укажите их отличия.

2 Что такое «ключевые кадры»?

3. Укажите формат описания CSS animations.

4. Укажите формат описания CSS transitions.

5. Перечислите под-свойства свойства animation.

6. Перечислите под-свойства свойства transitions.

7. Раскройте суть понятия «множественные трансформации».

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 13**

## **Тема работы: «Создание XML-документа. Изучение различий языков разметки XML и HTML»**

**1. Цель работы**

Формирование умений создания XML-документов.

**2. Задание**

Создать XML-документ в соответствии с вариантом задания.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Расширяемый язык разметки XML (eXtensible Markup Language) предназначен для описания данных с помощью разметки, синтаксис которой определен соответствующей спецификацией.

Синтаксис разметки XML-документа, представляя собой со­вокупность тегов и их атрибутов, внешне напоминает HTML. Однако есть существенное отличие: XML является языком ме­таразметки, что означает отсутствие фиксированного набора тегов и атрибутов для описания данных, и позволяет создателям документов определять названия тегов и атрибутов в соответст­вии с предметной областью и содержанием хранимых данных.

Разметка в XML-документе описывает структуру содержащихся в нем данных, не указывая при этом, как они должны отображаться. XML-документ может создаваться и редактироваться с помощью любого текстового редактора (например, Блокнота) или специализированного XML-редактора (например, XML Notepad).

Рассмотрим структуру XML-документа, содержащего информацию об имеющихся в наличии книгах и статьях:

<?xml version="1.0" encoding="Windows-1251"?>

<!DOCTYPE documents SYSTEM "documents.dtd">

<documents>

<books>

<book id="Book1">

<author>Эрих Мария Ремарк</author>

<title>Три товарища</title>

<translate> Иванова И.В. </translate>

<year\_publication>2000</year\_publication>

</book>

<book id="Book2">

<author>Рэй Брэдбери</author>

<title>Вино из одуванчиков</title>

<translate> Петрова А.К. </translate>

<year\_publication>2009</year\_publication>

</book>

</books>

<articles>

<article id="Article1">

<author>Виктор Кожевников</author>

<title>Классификация электронных средств обучения</title>

<year\_publication>1998</year\_publication >

</article>

</articles>

</documents>

Первые две строчки называются прологом XML-документа. Непосредственно данные с разметкой являются телом документа, которое должно быть заключено в так называемый корневой элемент (в примере <documents>).

В объявлении типа документа (строка <?xml version="1.0" encoding="Windows-1251"?>) определяются версия XML и кодировка, используемая в документе.

В разделе DOCTYPE указывается имя корневого элемента и DTD-тип документа, описанный в отдельном файле (как в примере) или непосредственно в XML-документе в квадратных скобках **[ ]**.

Содержимое XML-документа представляет собой набор элементов, секций CDATA, директив анализатора, комментариев, специальных символов, текстовых данных.

Единица данных вместе с относящейся к ней разметкой называется элементом. Любой непустой элемент должен состоять из начального, конечного тэгов и данных, заключенных между ними. Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные, и поэтому учитываются все символы форматирования (пробелы, переводы строк, табуляции не игнорируются, как в HTML). Если элемент не имеет содержимого, то есть данных, которые он должен определять, он называется пустым. Для определения пустого элемента используется один тег с косой чертой перед закрывающей угловой скобкой (например, **<empty/>**).

Если при определении элементов необходимо задать какие-либо параметры, уточняющие его характеристики, то можно использовать атрибуты элемента. В примере для указания идентификатора книги или статьи используется атрибут **id**.

Комментарии обозначаются последовательностью символов **<!-- и -->**.

Чтобы задать область документа, которую при разборе XML-анализатор будет рассматривать как простой текст, игнорируя любые инструкции и специальные символы, но, в отличие от комментариев, иметь возможность использовать их в приложении, необходимо использовать тэги **<![CDATA[ и ]]>**. Внутри этого блока можно помещать, например, инструкции JavaScript.

Для того чтобы включить в документ символ, используемый для определения каких-либо конструкций языка (например, символ угловой скобки) и не вызвать при этом ошибок в процессе разбора, нужно использовать его специальный символьный либо числовой идентификатор. Например, &amp; &lt; &gt; &quot; или &#60; (десятичная форма записи символа<), &#x3c (шестнадцатеричная форма записи) и т. д.

Инструкции, предназначенные для анализаторов языка, описываются при помощи тэгов **<? и ?>**.

В общем случае XML-документы должны удовлетворять следующим требованиям:

- для непустых элементов обязателен закрывающий тег;

- нужно учитывать регистр символов;

- все значения атрибутов должны быть заключены в кавычки;

- элементы не должны перекрываться;

- должен быть только один корневой элемент;

- элемент не может иметь несколько атрибутов с одним именем.

**5. Порядок выполнения работы**

1. Создайте XML-документ согласно варианту. Документ должен содержать не менее трех записей.

Вариант 1. Информация о книгах в библиотеке:

- идентификационный номер; стеллаж (номер стеллажа, жанр, тематика); автор или авторы (наличие необязательно); название; под редакцией (наличие необязательно); общее количество экземпляров; количество выданных экземпляров.

Вариант 2. Информация о сотрудниках фирмы:

- фамилия; имя; отчество; адрес (индекс, тип населенного пункта (город, поселок, деревня), область, район, название населенного пункта, улица, дом, корпус (наличие необязательно), квартира); телефон (домашний, мобильный (может быть несколько)); должность; подразделение.

Вариант 3. «Кулинарная книга»:

- тип блюда; название блюда; мера весов; ингредиент 1; количество ингредиента 1;…; рецепт; количество калорий.

Вариант 4. Информация об услугах фитнес-центра:

- разновидность услуги; стоимость каждой услуги; инструктор(фамилия, имя, отчество, образование (уровень квалификации); дни недели и время занятий.

Вариант 5. Информация о подписке на периодические издания:

- тип издания (газета или журнал); тематика; название издания; периодичность выхода издания; цена; подписной индекс.

Вариант 6. Информация о расписании междугородних автобусов:

- направление; остановки; дата отправления; время отправления (автобусы могут отправляться не­сколько раз в день); стоимость билетов; количество свободных мест.

Вариант 7. Информация о передачах на ТВ:

- передача; день(число, месяц, год, день недели); канал; жанр; время начала; продолжительность.

Вариант 8. Информация о графике работы врачей медицинского центра:

- специализация врача; врач(фамилия, имя, отчество); дни приема; время приема по дням; номер кабинета.

Вариант 9. Информация о предложениях туристического агенства:

- страна; тип тура (экскурсия, индивидуальный тур, тур выходного дня); описание тура; горящий тур (непарный элемент); стоимость; количество.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Для чего используется язык XML?

2. Чем XML отличается от HTML?

3. Что такое элемент XML?

4. Для чего используется раздел DOCTYPE?

5. Как обозначаются комментарии?

6. Перечислите правила построения XML-элементов.

**8. Рекомендуемая литература**

1. **Кищенко, О. Н.** Языки информационного обмена / О. Н. Кищенко. – М.: НОУ "Интуит", 2016. – 243с.
2. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
3. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
4. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
5. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
6. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 14**

## **Тема работы: «Описание структуры XML-документа средствами DTD и на языке XSD»**

**1. Цель работы**

Формирование умений описания структуры XML-документа средствами DTD и на языке XSD.

**2. Задание**

Выполнить DTD-описание и разработать XSD-схему XML-документа.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

В XML-документах DTD определяет набор действительных элементов, идентифицирует элементы, которые могут находиться в других элементах, и определяет действительные атрибуты для каждого из них. В XML использовать DTD не обязательно – документы, созданные без этих правил, будут правильно обрабатываться программой-анализатором, если они удовлетворяют основным требованиям синтаксиса XML. Однако, контроль за типами элементов и корректностью отношений между ними в этом случае будет полностью возлагаться на автора документа. До тех пор, пока грамматика нашего нового языка не описана, его сможем использовать только мы, и для этого мы будем вынуждены применять специально разработанное программное обеспечение, а не универсальные программы-анализаторы.

В DTD для XML используются следующие типы правил:

- правила для элементов и их атрибутов;

- описания категорий (макроопределений);

- описание форматов бинарных данных.

Все они описывают основные конструкции языка – элементы, атрибуты, символьные константы, внешние файлы бинарных данных.

Для того чтобы использовать DTD в документе, можно или описать его во внешнем файле и при описании DTD просто указать ссылку на этот файл, или же непосредственно внутри самого документа выделить область, в которой определить нужные правила. В первом случае в документе указывается имя файла, содержащего DTD- описания:

<?xml version=“1.0” standalone=“no” ?>

<! DOCTYPE documents SYSTEM «def.dtd»>

Внутри документа DTD-декларации включаются следующим образом:

<! DOCTYPE documents [

<!ELEMENT documents (books+, articles+)>

…

]>

Значение **standalone** определяет, может ли этот документ быть обработан без чтения каких-либо других файлов. Например, если XML-документ не ссылается на другие файлы, вы должны указать standalone=“yes”. Если же XML-документ ссылается на другие файлы, которые описывают, что документ может содержать, нужно указать standalone=“no”.

Атрибут SYSTEM указывает на то, что def.dtd находится на локальном сервере.

В том случае, если используются одновременно внутренние и внешние описания, то программой-анализатором будут сначала рассматриваться внутренние, то есть их приоритет выше. При проверке документа XML-процессор в первую очередь ищет DTD внутри документа. Если правила внутри документа не определены и не задан атрибут standalone =“yes” , то программа загрузит указанный внешний файл и правила, находящиеся в нем, будут считаны оттуда. Если же атрибут standalone имеет значение yes, то использование внешних DTD-описаний будет запрещено.

Элемент в DTD определяется с помощью дескриптора !ELEMENT, в котором указывается название элемента и структура его содержимого.

Например, для элемента **<author>** можно определить следующее правило:

<!ELEMENT author (#PCDATA)>

В определении элемента нужно указать сначала название элемента (author), а затем модель его содержимого – другие элементы или типы данных, которые могут встречаться внутри него. В данном случае содержимое элемента author будет определяться при помощи специального маркера #PCDATA (что означает parseable character data – любая информация, с которой может работать программа-анализатор). Существуют еще две инструкции, определяющие тип содержимого: **EMPTY** и **ANY**. Первая указывает на то, что элемент должен быть пустым (например, <red/>), вторая – на то, что содержимое элемента специально не описывается.

Последовательность дочерних для текущего элемента объектов задается в виде списка названий элементов, разделенных запятыми. При этом для того, чтобы указать количество повторений включений этих элементов, могут использоваться символы «+», «?». Например:

<!ELEMENT book (author+, title, translate?, year\_publication)>

При этом определении элемент **<book>** должен включать элементы author, title, translate и year\_publication, причем элементы title и year\_publication являются обязательными и могут встречаться лишь однажды, элемент author может встречаться несколько раз, а элемент translate является опциональным, то есть может отсутствовать. В том случае, если существуют несколько возможных вариантов содержимого определяемого элемента, их следует разделять при помощи символа **«|»**. Например:

<!ELEMENT flower (#PCDATA | title)\*>

Символ **«\*»** указывает на то, что определяемая последовательность внутренних элементов может быть повторена несколько раз или же совсем не использоваться.

Если в определении элемента указывается «смешанное» содержимое (текстовые данные или набор элементов), то необходимо сначала указать PCDATA, а затем разделенный символом «|» список элементов.

Списки атрибутов элемента определяются с помощью ключевого слова !ATTLIST. Внутри него задаются названия атрибутов, типы их значений и дополнительные параметры. Например, для элемента <name>

<name first= “Иван” second= “Петрович” surname= “Сидоров”>

атрибуты могут быть определены следующим образом:

<!ATTLIST name first CDATA #REQUIRED second CDATA #IMPLIED surname CDATA #REQUIRED>

Всего существуют шесть возможных типов значений атрибута:

- CDATA – содержимым документа могут быть любые символьные данные;

- NMTOKEN (NMTOKENS) – содержимым элемента может быть только одно отдельное слово (этот параметр является ограниченным вариантом CDATA);

- ID – определяет уникальный идентификатор элемента в документе;

- IDREF (IDREFS) – указывает, что значением атрибута должно выступать название (или несколько таких названий, разделенных пробелами во втором случае) уникального идентификатора определенного в этом документе элемента;

- ENTITY (ENTITIES) – значение атрибута должно быть на­званием (или списком названий, если используется ENTITIES) компонента (макроопределения), определенного в документе;

- Список допустимых значений – определяется список значений, которые может иметь данный атрибут (в скобках без кавычек).

В определении атрибута также можно использовать следующие параметры:

- #REQUIRED – определяет обязательный атрибут, который должен быть задан во всех элементах данного типа;

- #IMPLIED – атрибут не является обязательным;

- #FIXED «значение» – указывает, что атрибут должен иметь только указанное значение, однако само определение атрибута не является обязательным, но в процессе разбора его значение в любом случае будет передано программе-анализатору;

- Значение – задает значение атрибута по умолчанию.

Рассмотрим XML-документ с DTD-определением:

<?xml version=“1.0” encoding=“Windows-1251”?>

<!DOCTYPE documents SYSTEM [

<!ELEMENT documents (books+, articles+)>

<!ELEMENT books (book\*)>

<!ELEMENT articles (article\*)>

<!ELEMENT book (author+, title, translate?, year\_publi-cation)>

<!ELEMENT article (author+, title, year\_publication)>

<!ATTLIST book id ID #REQUIRED>

<!ATTLIST article id ID #REQUIRED>

<!ELEMENT author (#PCDATA)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT translate (#PCDATA)>

<!ELEMENT year\_publication (#PCDATA)>

]>

<documents>

<books>

<book id=“Book1” >

<author>Эрих Мария Ремарк</author>

<title>Три товарища</title>

<translate> Иванова И.В. </translate>

<year\_publication>2000</year\_publication>

</book>

<book id=“Book2” >

<author>Рэй Брэдбери</author>

<title>Вино из одуванчиков</title>

<translate> Петрова А.К. </translate>

<year\_publication>2009</ year\_publication>

</book>

</books>

<articles>

<article id=“Article1” >

<author>Виктор Кожевников</author>

<title>Классификация электронных средств обучения</title>

<year\_publication>1998</year\_publication>

</article>

</articles>

</documents>

Довольно часто при создании XML-элемента разработчику требуется определить, данные какого типа могут использоваться в качестве его содержимого.

Например, некоторый элемент <last-modified>10.10.98</last-modified> должен содержать строку, представляющую собой дату, а не число или произвольную последовательность символов.

С помощью типизации данных можно создавать элементы, значения которых могут использоваться, например, в качестве параметров SQL-запросов. Программа-клиент в этом случае должна знать, к какому типу данных относится текущее значение элемента, и в случае соответствия формирует SQL-запрос.

Если в качестве программы на стороне клиента используется верифицирующий XML-процессор, то информацию о типе можно передавать при помощи специально созданного для этого атрибута элемента, имеющего соответствующее DTD-определение. В процессе разбора программа-анализатор передаст значение этого атрибута клиентскому приложению, которое сможет использовать эту информацию должным образом. Например, чтобы указать, что содержимое элемента должно быть длинным целым, можно использовать следующее DTD-определение:

<!ELEMENT counter (#PCDATA)>

<!ATTLIST counter data\_long CDATA #FIXED «LONG»>

Задав атрибуту значение по умолчанию LONG и определив его как FIXED, мы, тем самым, позволили программе-клиенту получить необходимую информацию о типе содержимого данного элемента, и теперь она может самостоятельно определить соответствие типа этого содержимого указанному в DTD-определении.

Пример:

<?xml version=“1.0” encoding=“Windows-1251”?>

<!DOCTYPE houses SYSTEM [

<!ELEMENT price (#PCDATA)>

<!ATTLIST price data\_currency CDATA #FIXED «CURRENCY»>

<!ELEMENT rooms\_num (#PCDATA)>

<!ATTLIST rooms\_num data\_byte CDATA #FIXED «BYTE»>

<!ELEMENT floor (#PCDATA)>

<!ATTLIST floor data\_byte CDATA #FIXED «INTEGER»>

<!ELEMENT living\_space (#PCDATA)>

<!ATTLIST living\_space data\_float CDATA #FIXED «FLOAT»>

<!ELEMENT counter (#PCDATA)>

<!ATTLIST counter data\_long CDATA #FIXED «LONG»>

<!ELEMENT is\_tel (#PCDATA)>

<!ATTLIST is\_tel data\_bool CDATA #FIXED «BOOL»>

<!ELEMENT houses (house\*)>

<!ELEMENT house (rooms\_num, floor,living\_space, is\_tel, counter, price)>

<!ATTLIST house id ID #REQUIED>

]>

<houses>

…

<house id=«0»>

<rooms\_num>5</rooms\_num>

<floor>2</floor>

<living\_space>32.5</living\_space>

<is\_tel>true</is\_tel>

<counter>18346</counter>

<price>34 р. 28 к.</price>

</house>

…

</houses>

Как видно из примера, механизм создания элементов документа при этом нисколько не изменился. Вся необходимая для проверки типов данных информация заложена в определении элементов внутри блока DTD.

Язык XML Schema предоставляет набор встроенных типов данных, которые могут использоваться разработчиками для определения содержимого документа (например: float, string, boolean и т. д.).

Кроме предоставления встроенных типов, в XML Schema, как и во многих языках программирования, разработчикам разрешено определять свои собственные типы, обычно называемые определенными пользователем типами (UDT). При описании UDT можно также определить для них и пространство имен, чтобы не путать их с другими UDT, случайно использующими такое же имя.

Имена, определенные в схеме, принадлежат так называемому **целевому пространству имен**. Само по себе пространство имен является фиксированным, произвольным именем, которое должно соответствовать синтаксису URL (например, http://example.org/publishing). В то же время, объявление про­странства имен не ссылается на файл с описанием схемы, а лишь указывает его имя.

Определения и объявления в схеме могут ссылаться также на имена, которые принадлежат другим пространствам имен, называемым исходными. В каждой схеме может быть определено одно целевое пространство имен и возможно использование множества исходных пространств имен.

Рассмотрим особенности объявления и использования пространств имен на основе упрощенного шаблона XML-схемы:

<xsd:schema

targetNamespace="http://example.org/publishing"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:tns="http://example.org/publishing">

<!--определение типов-->

<xsd:simpleType name="AuthorType">

<!--определение области значений типа-->

...

</xsd:simpleType>

<!--глобальное объявление элементов и их атрибутов-->

<xsd:element name="author" type="tns:AuthorType"/>

<xsd:element name="year" type="xsd: gYear"/>

...

</xsd:schema>

Элемент **xsd:schema** определяет содержимое пространства имен, а атрибут **targetNamespace** – его имя. Таким образом, целевым пространством имен является http://example.org/publishing, оно содержит имена AuthorType, year и author. Имена schema, element, simpleType и gYear принадлежат стандартному исходному пространству имен http://www.w3.org/2001/XMLSchema.

Имена пространства имен сокращаются при помощи конструкции с зарезервированным словом xmlns. Так, в примере целевому пространству имен назначается псевдоним **tns**, а исходному – **xsd**. Затем в документе указание конкретного имени предваряется псевдонимом пространства имен, к которому оно принадлежит (например, tns:AuthorType).

Определение элемента заключается в определении его имени и модели (типа) контента.

Шаблон определения элемента выглядит следующим образом:

<xsd:element name="…" type="…" [minOccurs="…"] [maxOccurs="…"] [default="…"| fixed="…"]>

Обязательными являются только параметры **name** и **type**, определяющие соответственно имя и тип элемента. Параметры minOccurs и maxOccurs определяют минимальное или максимальное количество вхождений элемента. Например, если элемент является обязательным, значение minOccurs больше или равно 1. Параметр maxOccurs может кроме числового значения принимать также значение unbounded, что указывает на отсутствие ограничения максимального числа появлений. Значение по умолчанию для minOccurs и для maxOccurs равно 1.

Атрибут элемента определяется по следующему шаблону:

<xsd:attribute name="…" type="…" [use="…"] [default="…"| fixed="…"]>

Атрибуты, в отличие от элементов, могут появиться только однажды или ни разу. Поэтому и синтаксис для определения появления атрибутов отличается от синтаксиса для определения числа появлений элементов. В частности, атрибуты могут быть объявлены с параметром use. В зависимости от значения этого параметра атрибут обязателен (use="required"), необязателен (use="optional") или запрещен (use="prohibited").

Значения по умолчанию и атрибутов, и элементов могут быть объявлены с использованием параметра default, хотя этот параметр в том или ином случае работает по-разному. Атрибут со значением, определенным по умолчанию, может появляться или не появляться в документе. Если атрибут не появляется в документе, то обработчик схемы обеспечивает атрибут со значением, равным значению default. Таким образом, значения по умолчанию для атрибутов имеют смысл, только если сами атрибуты являются необязательными, поэтому будет ошибкой определить значение по умолчанию вместе с параметром **use**, отлич­ным от use="optional".

Значение по умолчанию для элементов обрабатывается немного по-другому. Если элемент появляется в документе, но не содержит какого-либо значения, то в качестве его значения подставляется значение по умолчанию. Однако, если элемент не появляется в документе, то обработчик схемы не обеспечивает его значения вообще. В общем, различия между значениями по умолчанию элемента и атрибута в следующем: заданное по умолчанию значение атрибута применяется тогда, когда атрибут отсутствует, а заданное по умолчанию значение элемента применяются тогда, когда элемент присутствует в документе, но не имеет значения (пуст).

Атрибут fixed используется в объявлениях и атрибутов, и элементов. Он используется, чтобы указать, что атрибут или элемент принимают фиксированные значения (независимо от их наличия в документе). Обратите внимание, что понятия «фиксированное значение» и «значение по умолчанию» являются взаимоисключающими, поэтому объявление не может одновременно содержать атрибуты fixed и default.

Тип элемента может быть простым или комплексным (сложным). Элемент простого типа не может содержать другие элементы или атрибуты. Комплексный тип может использоваться для встраивания элементов в другие элементы или ассоциирования атрибутов с элементом.

Предопределенный простой тип ограничивает значения по их базовому типу и создается с помощью элемента **simpleType**. Для ограничения диапазона значений используется вложенный элемент restriction. Он позволяет определить, как именно нужно ограничивать базовый тип, сужая один или более из его аспек-тов ( facets).

Например, чтобы определить ограничения для элемента, содержащего год издания книги или журнала, может использоваться следующий код:

<xsd:simpleType name="YearType">

<xsd:restriction base="xsd:gYear">

<xsd:minInclusive value="1900"/>

<xsd:maxInclusive value="2012"/>

</xsd:restriction>

</xsd:simpleType>

Таким образом, допустимые значения года издания находятся в интервале от 1900 до 2012.

Возможные аспекты базового типа XSD представлены в таблице 14.1.

Таблица 14.1 – Аспекты базового типа XSD

|  |  |
| --- | --- |
| **Аспект (Facet)** | **Описание** |
| **1** | **2** |
| Enumeration | Определяет фиксированное значение, с которым должен совпадать тип |
| fractionDigits | Определяет максимальное количество десятичных знаков справа от десятичной точки |
| Length | Определяет количество символов в строковом типе, количество байтов в двоичном типе или количество элементов в списочном типе |
| maxExclusive | Определяет исключающую верхнюю границу области значений типа |
| maxInclusive | Определяет включающую верхнюю границу области значений типа |
| maxLength | Определяет максимальное количество символов в строковом типе, максимальное количество байтов в двоичном типе или максимальное количество элементов в списочном типе |
| minExclusive | Определяет исключающую нижнюю границу облас-ти значений типа |
| minInclusive | Определяет включающую нижнюю границу области значений типа |
| minLength | Определяет минимальное количество символов в строковом типе, минимальное количество байтов в двоичном типе или минимальное количество элементов в списочном типе |
| Pattern | Определяет шаблон, основанный на регулярном выражении, с которым должен совпадать тип |
| totalDigits | Определяет максимальное количество десятичных знаков для типов, унаследованных от *number* |
| **1** | **2** |
| whiteSpace | Определяет правила нормирования пробелов |

Большинство аспектов неприменимы ко всем типам (некоторые имеют смысл только в определенных типах). Элемент pattern применяется, если ограничением выступает регулярное выражение.

Например, чтобы создать ограничение для элемента, содержащего информацию об авторе, нужно определить следующий простой тип:

<xsd:simpleType name="AuthorType" base="string">

<pattern value="^[а-яА-Я. ]+ "/>

</xsd:simpleType>

Базовым является тип string (строка), ограничение по базовому типу создается с использованием регулярных выражений. В примере информация об авторе может содержать один или более символов, включающих строчные или прописные буквы русского алфавита, точку, а также пробел. Символ "^" указывает на начало строки.

Развернутое описание синтаксиса регулярных выражений приведено в таблице 14.2.

Таблица 14.2 – Представление символов в регулярных выражениях

|  |  |
| --- | --- |
| **Символ** | **Значение** |
| **1** | **2** |
| **\** | Делает обычные символы специальными и наоборот. Например, выражение "s" ищет просто символ 's'. Если поставить слэш перед s, то "\s" уже обозначает пробельный символ. И наоборот, если символ специальный, например \*, то слэш сделает его просто обыч-ным символом «звездочка». Например, "a\*" ищет 0, или больше подряд идущих символов 'a'. Чтобы найти 'a\*', нужно поставить слэш перед специальным символом: "a\\*" |
| **^** | Обозначает начало входных данных. Если установ-лен флаг многострочного поиска ("m"), то также сработает при начале новой строки. Например, "^A" не найдет 'A' в "an A", но найдет первое 'A' в "An A" |
| $ | Обозначает конец входных данных. Если установлен флаг многострочного поиска, то также сработает в конце строки. Например, "t$" не найдет 't' в "eater", но найдет в "eat" |
| **\*** | Обозначает повторение 0 или более раз. Например, "bo\*" найдет 'boooo' в "A ghost booooed" и 'b' в "A bird warbled", но ничего не найдет в "A goat grunted" |
| **+** | Обозначает повторение 1 или более раз. Эквивалент-но {1,}. Например, "a+" найдет 'a' в "candy" и все 'a' в "caaaaaaandy" |
| ? | Обозначает, что элемент может как присутствовать, так и отсутствовать. Например, "e?le? " найдет 'el' в "angel" и 'le' в "angle" |
| **1** | **2** |
| **.**  (десятичная точка) | Обозначает любой символ, кроме перевода строки: \n \r \u2028 or \u2029. Например, ".n" найдет 'an' и 'on' в "nay, an apple is on the tree", но не 'nay' |
| (*x*) | Находит *x* и запоминает. Это называется «запоми­наю­щие скобки». Например, "(foo) " найдет и запомнит 'foo' в "foo bar." Кроме того, скобки объединяют то, что в них находится, в единый элемент паттерна. Например, (abc)\* – повторение abc0 и более раз |
| (?:*x*) | Находит *x*, но не запоминает найденное. Это называется «незапоминающие скобки». Как и все скобки, объединяют находящееся в них в единый паттерн |
| *x*(?=*y*) | Находит *x*, только если за *x* следует *y*. Например, "Jack(?=Sprat)" найдет 'Jack', только если за ним следует 'Sprat'. "Jack(?=Sprat|Frost)" найдет 'Jack', только если за ним следует 'Sprat' или 'Frost'. Однако, ни 'Sprat' ни 'Frost' не войдут в результат поиска |
| *x*(?!*y*) | Находит *x*, только если за *x* не следует y. Например, "\d+(?!\.)" найдет число, только если за ним не следует десятичная точка |
| *x*|*y* | Находит *x* или *y*. Например, "green|red" найдет 'green' в "green apple" и 'red' в "red apple" |
| {*n*} | Находит ровно *n* повторений предшествующего элемента (*n* – положительное целое число). Например, "a{2}" не найдет 'a' в "candy," но найдет оба 'a' в "caandy" и первые два 'a' в "caaandy" |
| {*n*,} | Находит *n* и более повторений элемента (*n* – положи-тельное целое число). Например, "a{2,} " не найдет 'a' в "candy", но найдет все 'a' в "caandy" и в "caaaaaaandy" |
| {*n*,*m*} | Находит от *n* до *m* повторений элемента (*n* и *m* – положительные целые числа) |
| [*xyz*] | Находит любой из перечисленных символов. Можно указать промежуток, используя тире. Например, [abcd] – то же самое, что [a-d]. Найдет 'b' в "*brisket*", а также 'a' и 'c' в "ache" |
| [^*xyz*] | Находит любой символ, кроме указанных в наборе. Также можно указать промежуток. Например, [^abc] – то же самое, что [^a-c]. Найдет 'r' в "brisket" и 'h' в "chop" |
| [\*b*] | Находит символ backspace (не путать с \b) |
| \*b* | Находит границу слов (латинских), например, про-бел. (Не путать с [\b]). Например, "\bn\w" найдет 'no' в "noonday"; "\wy\b" найдет 'ly' в "possibly yesterday*"* |
| \*B* | Обозначает не границу слов. Например, "\w\Bn" найдет 'on' в "noonday", а "y\B\w" найдет 'ye' в "possibly yesterday" |
| \*cX* | Находит контрольный символ в строке (*X* – буква от A до Z). Например, "\cM" обозначает символ Ctrl + M |
| \*D* | Находит нецифровой символ (все алфавиты). [^0-9] – эквивалент для обычных цифр. Например, "\D" или "[^0-9] " найдет 'B' в "B2 is the suite number" |
| \*n* | Находит перевод строки |
| **1** | **2** |
| \*s* | Находит любой пробельный символ, включая пробел, табуляцию, переводы строки. Например, "\s\w\*" найдет ' bar' в "foo bar" |
| \*S* | Находит любой символ, кроме пробельного. Например, "\S\w\*" найдет 'foo' в "foo bar" |
| \*t* | Находит символ табуляции |
| \*v* | Находит символ вертикальной табуляции |
| \*w* | Находит любой словесный (латинский алфавит) символ, включая буквы, цифры и знак подчеркивания. Эквивалентно [A-Za-z0-9\_]. Например, "\w" найдет 'a' в "apple," '5' в "$5.28," и '3' в "3D" |
| \*W* | Находит любой нелатинский словесный символ. Эквивалентно [^A-Za-z0-9\_]. Например, "\W" и "[^$A-Za-z0-9\_]" одинаково найдут '%' в "50%" |
| \n | Обратная ссылка на *n*-ю запомненную скобками подстроку (*n* – целое число)  Например, "apple(,)\sorange\1" найдет 'apple, orange,' в "apple, orange, cherry, peach" |
| \0 | Находит символ NUL |

Комплексный (составной) тип определяется для элементов, содержащих вложенные элементы, и элементов с атрибутами.

Чтобы определить новый комплексный тип в целевом пространстве имен схемы, используется элемент **xsd:complexType:**

<xsd:complexType name="…">

<!--описание модели содержимого комплексного типа -->

</xsd:complexType>

Описание модели содержимого комплексного типа обычно включает объявления элементов, ссылки на элементы (<xsd:ele-ment ref="имя элемента, объявленного ранее">) и объявления атрибутов.

При описании модели содержимого также используются конструкции **xsd:sequence**, **xsd:choice** и **xsd:all, называемые составителями. Контейнер** xsd:sequence определяет порядок указания дочерних элементов, контейнер xsd:choice указывает на необходимость выбора только одного дочернего элемента из указанных, а контейнер xsd:all предписывает, что дочерние элементы могут размещаться в произвольном порядке. Описания атрибутов помещаются не в составителе, а после него в конце определения составного типа.

Пример возможной XSL-схемы для документа из лабораторной работы №14:

<xsd:schema targetNamespace="http://example.org/publishing"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:tns="http://example.org/publishing">

<xsd:element name="documents" type="tns: DocsType">

<xsd:element name="author" type="tns:AuthorType"/>

<xsd:element name="title" type="xsd:string"/>

<xsd:element name="year\_publication" type="tns:YearType "/>

<xsd:complexType name="DocsType">

<xsd:sequence>

<xsd:element name="books">

<xsd:complexType>

<xsd:element name="book" type="tns: BookType">

</xsd:complexType>

<xsd:element name="articles">

<xsd:complexType>

<xsd:element name="article" type="tns: ArticleType">

</xsd:complexType>

</xsd: sequence>

</xsd:complexType>

<xsd:complexType name="BookType">

<xsd:sequence>

<xsd:element ref="author">

<xsd:element ref="title">

<xsd:element name="translate" type="tns:AuthorType">

<xsd:element ref=" year\_publication">

</xsd: sequence>

<xsd:attribute name="id" type="xsd:string" use="required">

</xsd:complexType>

<xsd:complexType name="ArticleType">

<xsd:sequence>

<xsd:element ref="author">

<xsd:element ref="title">

<xsd:element ref=" year\_publication">

</xsd: sequence>

<xsd:attribute name="id" type="xsd:string" use="required">

</xsd:complexType>

<xsd:simpleType name="AuthorType" base="string">

<pattern value="^[а-яА-Я. ]+ "/>

</xsd:simpleType>

<xsd:simpleType name="YearType">

<xsd:restriction base="xsd:gYear">

<xsd:minInclusive value="1900"/>

<xsd:maxInclusive value="2012"/>

</xsd:restriction>

</xsd:simpleType>

</xsd:schema >

В начале документа определяется корневой элемент и элементы, на которые далее будет выполняться ссылка. Затем по порядку определяются комплексные и простые типы элементов.

Если элементы документа не принадлежат никакому пространству имен и записаны без префикса, то для связи документа XML со схемой XSD в корневом элементе документа записывается атрибут noNamespaceSchemaLocation:

<documents xmlns:xsi="http://www.w3.org/2001/XMLSchema"

xsi:noNamespaceSchemaLocation="sch.xsd">

Если же элементы документа относятся к некоторому пространству имен, то применяется атрибут schemaLocation, в котором через пробел перечисляются пространство имен и расположение файла со схемой, описывающей это пространство имен:

<documents xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi: schemaLocation=

"http://some.firm.com /someNames A.xsd

http://some.firm.com /anotherNames B.xsd”

xmlns:pr1= http://some.firm.com/someNames

xmlns:pr2= “http://some.firm.com /anotherNames">

Значением атрибута xsi:schemaLocation является список пар имен пространства имен и размещения URI, разделенных пробелами, который показывает список, где искать определенный файл схемы.

Таким образом, в документе можно использовать имена, определенные в схемах A.xsd и B.xsd, снабжая их префиксами pr1 и pr2 соответственно.

**5. Порядок выполнения работы**

1. Выполнить задание в соответствии с вариантом.

Вариант 1. Выполнить DTD-описание XML-документа, созданного в ла-бораторной работе №13, которое должно включать: описание элементов; описание атрибутов элементов; определение типа элементов.

Вариант 2. Разработайть XSD-схему XML-документа, созданного в лабораторной работе №13, которая должна включать: описание элементов; описание атрибутов элементов; определение типа элементов; определение последовательности использования элементов.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Для чего используется DTD?
2. Как определить элемент в DTD?
3. Как определить атрибут в DTD?
4. Какие параметры могут использоваться в определении атрибута?
5. Как определить тип элемента?
6. Как связать XML-документ c DTD-определением?
7. В чем отличие XSD от DTD?
8. Как определить элемент в XSD?
9. Как определить атрибут элемента в XSD?
10. Как подключить XSL-схему к HTML-документу?
11. В каких случаях используется сложный (комплексный) тип данных?

**8. Рекомендуемая литература**

1. **Кищенко, О. Н.** Языки информационного обмена / О. Н. Кищенко. – М.: НОУ "Интуит", 2016. – 243с.
2. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
3. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
4. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
5. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
6. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 15**

## **Тема работы: «Форматирование вывода и преобразование XML-документа средствами SLT»**

**1. Цель работы**

Формирование умений использования XSL-таблиц для форматирования вывода XML-документа.

**2. Задание**

Выполнить вывод содержимого XML-документа на HTML-страницу.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Расширяемый язык таблиц стилей (XSL) основан на XML и предназначен для трансформации документа XML в другой документ XML или отображения в виде HTML.

Язык XSL основан на каскадных таблицах стилей CSS (Сas-cading Style Sheets) и на языке DSSSL (Document Style Semantics and Specification Language – язык спецификации и семантики стиля документа). По мере развития язык XSL становится все ближе к CSS и дальше от DSSSL.

Документ XSL (файл с расширением .xsl) фактически представляет собой документ XML, который может содержать теги HTML.

XSL-таблица стилей является мощным и гибким инструментом для отображения XML-документов. Используя XSL-таблицы стилей, можно не только задать формат для каждого элемента XML, как при использовании CSS, но и обеспечить средства контроля над выводимыми данными: выбрать для отобра-жения конкретные данные XML, представить эти данные в любом порядке, модифицировать или добавлять информацию.

Вместе с тем, технология XSLT является более сложной для понимания, чем CSS. Кроме того, это новая технология, следовательно, имеет меньшую степень поддержки, то есть меньшую степень унификации среди современных браузеров.

Документ XSL может быть создан в любом текстовом редакторе.

Корневым элементом таблицы стилей является элемент <xsl: stylesheet>:

<xsl: stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

Первым атрибутом этого элемента является version, определяющий версию XSL, вторым – атрибут xmlns:xsl, определяющий пространство имен.

Элемент xsl:stylesheet должен содержать один или несколько шаблонов элементов, создаваемых с использованием элемента <xsl:template>:

<xsl:template match="/">

<!--дочерние элементы... -->

</xsl:template>

Браузер использует шаблон для отображения определенной ветви элементов в иерархии XML-документа, с которым связана таблица стилей. Атрибут match шаблона указывает на определенную ветвь (он аналогичен селектору в правиле CSS). Значение атрибута match носит название образца (pattern). Образец в данном примере ("/") представляет корневой элемент всего XML-документа. Таким образом, этот шаблон содержит инструкции для отображения всего XML-документа.

Для того чтобы связать таблицу стилей XSL с XML-доку­ментом, используется инструкция xml-stylesheet, которая имеет следующую обобщенную форму записи:

<?xml-stylesheet type="text/xsl" href="путь к файлу XSL"?>

В качестве значения атрибута href может использоваться абсолютный (URL полностью) или относительный путь.

Пример указания абсолютного пути:

<?xml-stytesheet type="text/xsl"

href="http:/www.my\_domain.com/lnventory.xsl"?>

Пример указания относительного пути (при условии, что lnventory.xsl находится в той же папке, что и текущий XML-документ):

<?xml-stylesheet type="text/xsl" href="lnventory.xsl"?>

Обычно инструкция xml-stylesheet добавляется в пролог XML-документа вслед за объявлением XML (например, <?xml version="1.0" encoding="Windows-1251"?>).

Если XML-документ связан с таблицей стилей XSL, то при его открытии в браузере отобразится содержимое, оформленное в соответствии с инструкциями по преобразованию, содержащи-мися в таблице стилей. В отличие от CSS, если XML-документ связан с несколькими таблицами стилей XSL, браузер использует первую таблицу и игнорирует все остальные. Если вы свяжете с XML-документом и CSS-таблицу, и XSL-таблицу стилей, браузер использует только XSL-таблицу стилей.

Если XML-документ не связан ни с CSS-таблицей, ни с XSL-таблицей стилей, браузер отобразит документ с помощью встроенной XSL-таблицы, которая используется по умолчанию. Эта таблица стилей отображает исходный XML-текст в виде дерева с возможностью свертывания/развертывания уровней.

Рассмотрим основные дескрипторы XSL для преобразования XML- документа в HTML. Шаблон <xsl:template> содержит два вида элементов:

1) элементы, представляющие собой HTML-разметку. Например:

<H2>Заголовок</H2>

2) XSL-элементы. Например:

<xsl:value-of select="/books/book/author"/>

Браузер отличает XML-элемент от элемента, представляющего HTML, поскольку первый имеет в качестве префикса описание пространства имен xsl:. XSL-элементы в шаблоне не копируются на выход HTML. Они лишь содержат инструкции по выбору и модификации данных XML либо используются для выполнения других задач.

XSL-элемент value-of добавляет текстовое содержимое определенного XML-элемента и его дочерних элементов в выходной модуль HTML. Конкретный элемент задается атрибутом select XSL-элемента value-of.

Адресация к XML-данным строится на выражениях языка [xPATH](http://ru.wikipedia.org/wiki/XPath). Самые простые выражения xPATH похожи на пути в файловых системах. Путь в файловой системе состоит из цепочек директорий, разделенных слэшем. xPATH-пути выглядят аналогичным образом:

- "/" – путь к корневому узлу;

- " /books" – путь к элементу books;

- "/books/book" – путь к элементу book, содержащему информацию о книге;

- "/books/book/author" – путь к элементу author, содержащему фамилию и имя автора книги.

Так же как и в файловых системах, в [xPATH](http://ru.wikipedia.org/wiki/XPath) есть понятия «относительного пути» и «абсолютного пути». Абсолютный путь отсчитывается от корневого узла и начинается с одинарного слэша "/" как в примерах выше. Относительный путь отсчитывается от текущего положения, определяемого значением атри-бута match элемента <xsl:template>.

Для того, чтобы добавить в выходной модуль HTML значение атрибута, используется символ @. Например, для вывода значения атрибута id тега <book id="Book1"> может исполь-зоваться следующая запись:

<xsl:value-of select="/books/book/author/@id"/>

Чтобы вывести содержимое элемента, включающее HTML-теги, используется конструкция  <xsl:copy-of select="название эле-мента/\*"/>. Значение атрибута select предполагает выбор всех дочерних узлов (элементов, атрибутов, текста, комментариев) конкретного элемента.

В XSL предусмотрены два условных оператора: xsl:if и xsl:choose.

Рассмотрим использование оператора xsl:if. Например, чтобы проверить, издана ли книга позднее 1990 года из XML-документа, может использоваться конструкция:

<xsl:if test="year\_publication > 1990"> Действия при выполнении условия</xsl:if>

Если нужно проверить обратное, то есть, что книга издана ранее 1990 гоа, нужно использовать не символ <, а его amp-последовательность (&lt;):

<xsl:if test="year\_publication &lt; 1990"> Действия при выполнении условия</xsl:if>

Так как XML-документ чаще всего содержит не одну запись, а набор записей одинаковой структуры, требуется использование специального оператора для их перебора. Таким оператором является xsl:for-each.

Рассмотрим пример вывода XML-документа в виде таблицы на HTML-страницу:

<?xml version="1.0" encoding="WINDOWS-1251" ?>

<xsl:stylesheet

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html>

<head>

</head>

<body>

<table border="1">

<tr bgcolor="#CCCCCC">

<td align="center"><strong>Автор</strong></td>

<td align="center"><strong>Название</strong></td>

<td align="center"><strong>Дата</strong></td>

</tr>

<tr>

<td align="center" colspan="3">

<strong>Книги</strong></td>

</tr>

<xsl:for-each select="documents/books/book">

<tr bgcolor="#F5F5F5">

<td><xsl:value-of select="author"/></td>

<td><xsl:value-of select="title"/></td>

<td><xsl:value-of select="translate"/></td>

<td><xsl:value-of select="year\_publication"/></td>

</tr>

</xsl:for-each>

<tr>

<td align="center" colspan="3">

<strong>Журналы</strong></td>

</tr>

<xsl:for-each select="documents/articles/article">

<tr bgcolor="#F5F5F5">

<td><xsl:value-of select="author"/></td>

<td><xsl:value-of select="title"/></td>

<td><xsl:value-of select="year\_publication"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

Здесь оператор xsl:for-each организует перебор всех элементов book и article. Далее все, что находится внутри этих элементов, адресуется относительно текущего элемента, заданного атрибутом select инструкции xsl:for-each.

Конструкция xsl:sort служит для сортировки XML-элементов. Например, чтобы сортировать книги по дате рождения автора, нужно модифицировать предыдущий пример следующим образом:

<?xml version="1.0" encoding="WINDOWS-1251" ?>

<xsl:stylesheet

xmlns:xsl=" http://www.w3.org/1999/XSL/Transform ">

<xsl:template match="/">

<html>

<head>

</head>

<body>

<table border="1">

<tr bgcolor="#CCCCCC">

<td align="center"><strong>Автор</strong></td>

<td align="center"><strong>Название</strong></td>

<td align="center"><strong>Дата</strong></td>

</tr>

<tr>

<td align="center" colspan="3">

<strong>Книги</strong></td>

</tr>

<xsl:for-each select="documents/books/book">

<xsl:sort select=" year\_publication"/>

<tr bgcolor="#F5F5F5">

<td><xsl:value-of select="author"/></td>

<td><xsl:value-of select="title"/></td>

<td><xsl:value-of select="translate"/></td>

<td><xsl:value-of select="year\_publication"/></td>

</tr>

</xsl:for-each>

<tr>

<td align="center" colspan="3">

<strong>Журналы</strong></td>

</tr>

<xsl:for-each select="documents/articles/article">

<tr bgcolor="#F5F5F5">

<td><xsl:value-of select="author"/></td>

<td><xsl:value-of select="title"/></td>

<td><xsl:value-of select="year\_publication"/></td>

</tr>

</xsl:for-each>

</table>

<html>

<head>

</head>

<body>

</xsl:template>

</xsl:stylesheet>

Опция select элемента xsl:sort определяет, по какому ключу идет сортировка (в примере – по году издания книги).

Элемент xsl:sort имеет также дополнительные параметры:

- порядок сортировки (убывание/возрастание, атрибут ca-se-order= "lower-first|upper-first");

- тип данных (числовой/текстовый, атрибут data-type="num-ber|text").

**5. Порядок выполнения работы**

1. Вывести на HTML-страницу содержимое XML-документа, созданного в лабораторной работе №13, используя XSL.
2. Выполнить сортировку записей по одному из возможных.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Для чего используется язык XSL?
2. Как связать таблицу стилей XSL с XML-документом?
3. Как создаются шаблоны элементов в XSL?
4. Для чего используется XSL-элемент value-of?
5. Как осуществляется адресация к XML-данным?
6. Какие условные операторы предусмотрены в XSL? В чем особенности их использования?
7. Для чего используется оператор for-each?
8. Как выполняется сортировка XML-данных?

**8. Рекомендуемая литература**

1. **Кищенко, О. Н.** Языки информационного обмена / О. Н. Кищенко. – М.: НОУ "Интуит", 2016. – 243с.
2. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
3. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
4. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
5. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
6. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 16**

## **Тема работы: «Обработка событий с использованием DHTML»**

**1. Цель работы**

Формирование умений создания динамических HTML-документов, навыков обработки событий с использованием DHTML.

**2. Задание**

Создать HTML-документ c динамически изменяемым содержимым.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Динамический HTML (Dynamic HTML или DHTML) не является каким-то особым языком разметки страниц. Это всего лишь термин, применяемый для обозначения HTML-страниц с динамически изменяемым содержимым.

Реализация DHTML состоит из трех компонентов: непосредственно HTML, каскадных таблицах стилей (Cascade Style Sheets – CSS) и языке сценариев (JavaScript или VBScript). Эти три компонента DHTML связаны между собой объектной моделью документа (Document Object Model – DOM), являющейся, по сути, интерфейсом прикладного программирования (API).

DOM обеспечивает динамический доступ к содержимому документа, его структуре и стилям. В DOM каждый элемент Web-страницы является объектом, который можно изменять. DOM не определяет новых тэгов и атрибутов, а просто обеспечивает возможность программного управления всеми тэгами, атрибутами и каскадными таблицами стилей (CSS).

Любое действие пользователя (щелчок кнопкой мыши, перемещение мыши в окне браузера или нажатие клавиши клавиатуры) объектной моделью документа трактуется как событие, которое может быть перехвачено и обработано процедурой сценария.

DHTML включает в себя средства динамического изменения размера, цвета, формы символов, точного позиционирования и перемещения отдельных элементов страницы. Основную идею динамического HTML можно сформулировать очень просто: полный контроль языка сценариев над всеми элементами документа, параметрами их оформления и размещения (как подразумеваемыми в HTML, так и задаваемыми с помощью CSS) и над самим текстом страницы.

На WEB-страницах можно обеспечить реакцию на определенные действия посетителя или изменения состояния документа или окна, которые вызывают определенные события.

Существуют следующие категории событий:

- стандартные события (могут быть связаны с любым элементом HTML);

- события для элементов, получающих фокус ввода;

- события, специфические для отдельных элементов.

Как правило, обработчик событий имеет имя, совпадающее с именем события, с приставкой "on", таблица 16.1.

Таблица 16.1 – Типы событий и обработчиков

|  |  |  |
| --- | --- | --- |
| **Событие** | **Когда происходит** | **Обработчик**  **события** |
| Blur | Потеря фокуса ввода элементом формы | onBlur |
| Click | Пользователь щелкает мышкой по элементу формы или связи | onClick |
| Change | Пользователь изменяет значение элемента text, textarea, или выделяет элемент | onChange |
| Focus | Установка фокуса ввода на элементе формы | onFocus |
| Load | Пользователь загружает страницу в браузер | onLoad |
| Mouseover | Пользователь двигает мышью над связью или якорем | onMouseOver |
| Select | Пользователь выбирает поле ввода элемента формы | onSelect |
| Submit | Пользователь посылает форму | onSubmit |
| Unload | Пользователь выходит со страницы | onUnload |

Стандартные события приведены в таблице 16.2.

Таблица 16.2 – Стандартные события JavaScript

|  |  |
| --- | --- |
| **Обработчик**  **события** | **Действие** |
| Onclick | Щелчок левой кнопкой мыши по области элемента |
| Ondblcliсk | Двойной щелчок левой кнопкой мыши по области элемента |
| Onmousedown | Левая кнопка мыши нажата, когда указатель находится в области элемента |
| Onmousemove | Указатель мыши перемещается по области элемента |
| Onmouseup | Левая кнопка мыши отпущена, когда указатель находится в области элемента |
| Onmouseover | Указатель мыши входит в пределы области элемента |
| Onmouseout | Указатель мыши выходит за пределы элемента |
| Onkeydown | Нажата и удерживается клавиша на клавиатуре |
| Onkeypress | Нажата и отпущена клавиша на клавиатуре |
| Onkeyup | Отпущена клавиша на клавиатуре |
| Onscroll | Элемент прокручивается |

События для элементов, получающих фокус ввода, приведены в таблице 16.3.

Таблица 16.3 – События для получения фокуса ввода

|  |  |
| --- | --- |
| **Обработчик**  **события** | **Действие** |
| Onenter | Произошел переход к элементу |
| Onfocus | Устанавливается фокус ввода для элемента |
| Onexit | Произошел выход из области элемента |
| Onblur | Снимается фокус с элемента |

События, используемые для отдельных элементов, приведены в таблице 16.4.

Таблица 16.4 – События, используемые для отдельных элементов

|  |  |
| --- | --- |
| **Обработчик**  **события** | **Действие** |
| Onabort | Пользователь прерывает прием изображения |
| Onchange | объект изменяется при вводе данных пользователем (для элементов формы <INPUT type=file| text| password| radio| checkbox>, <SELECT>, <TEXTAREA>) |
| Onload | Элемент (<BODY> или <IMG>) полностью загружен |
| Onreset | Нажата кнопка Reset, т. е. пользователь очистил форму |
| Onselect | Отпущена кнопка мыши при выделении текста пользователем |
| Onsubmit | Нажата кнопка Submit, т. е. форма отправлена на обработку |
| Onunload | Документ начал выгружаться |

Для того чтобы связать объект с обработчиком события, нужно в теге, определяющем данный объект, указать атрибут, соответствующий событию, и, в качестве значения, записать программный код (например, на JavaScript) или имя функции.

Основной синтаксис, используемый для указания обработчиков событий:

<TAGeventHandler="JavaScriptCode">

где TAG – это некоторый тег HTML;

eventHandler – и мя обработчика событий.

Пример:

<H1 onmouseover="this.style.color='red'"> Эта строка покраснеет</H1>

Ключевое слово **this** указывает на то, что изменяется стиль элемента, в теге которого определен обработчик события, в данном случае элемент **H1**.

Пример:

<html>

<head>

</head>

<BODY>

Введите Ваше имя:

<INPUT type="text" name="user" value="" onblur="salut(this.value)">

<SCRIPT>

function salut (name)

{

alert ("Здравствуйте, ", name, " !");

}

</SCRIPT>

</BODY>

</html>

Другой пример иллюстрирует обработку события **onClick** при нажатии кнопки «Выход» – происходит вызов функции подтверждения **confirm**:

<FORM>

<INPUT type="button" name="btn1" value="Выход" onClick="confirm('Вы желаете покинуть страницу?')">

</FORM>

<body>

<H2>Изменение вида данного элемента</H2>

<p id="B">Для изменения цвета и шрифта данного текста

нажимайте и отпускайте кнопку мыши<br></p>

<script>

<!--

document.getElementById('B').onmousedown=new

Function("this.style.fontStyle='italic'; this.style.color='red';");

document.getElementById('B').onmouseup=new

Function("this.style.fontStyle=''; this.style.color='blue';");

//-->

</script>

</body>

Если обработка события предполагает изменение ряда свойств элемента, можно определить два различных класса и применять их при возникновении событий:

<HTML>

<HEAD> </HEAD>

<STYLE>

.red{color:red;font-style:italic}

.green{color:green;font-weight:bold}

</STYLE>

<BODY>

<P onMouseOver="this.className='green'"

onMouseOut="this.className='red'">

Этот текст будет изменяться при наведении указателя мыши

</P>

</BODY>

</HTML>

При наведении указателя мыши будет применяться класс **green**, при выходе указателя за пределы элемента – класс **red**.

**5. Порядок выполнения работы**

1. Создайте документ Lab16-1.html, содержащий таблицу следующего вида:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Элемент1 | Элемент2 | Элемент3 | Элемент4 | Элемент5 | Элемент6 |

1. Определите следующие классы:
   1. cl1 – текст синего цвета, шрифт Courier New, 12 px;
   2. cl2 – текст красного цвета, полужирный;
   3. cl3 – текст зеленого цвета, шрифт Times New Roman, курсивное начертание;
   4. cl4 – текст черного цвета, в красной рамке (как показано на рисунке 16.1);
   5. cl5 – шрифт синего цвета, полужирный, курсивное начертание.



Рисунок 16.1 – Отображение стиля класса «cl4»

1. Ко всем элементам таблицы примените класс cl1.
2. При наведении указателя мыши на Элемент1 примените класс cl2. Когда указатель мыши выходит за пределы данного элемента таблицы, должен применяться класс cl5.
3. При двойном щелчке на Элемент2 примените класс cl3.
4. При одинарном щелчке на Элемент3 примените класс cl4.
5. При наведении курсора мыши на Элемент4 измените цвет текста на желтый.
6. При двойном щелчке на Элемент5 измените цвет фона ячейки.
7. При одинарном щелчке на Элемент6 измените начертание на курсивное.
8. Установите размеры ячеек таблицы таким образом, чтобы при изменении стиля элемента не перемещались границы ячеек.
9. Создайте документ Lab16-2.html. Выполните задание в соответствии с вариантом.

Вариант №1. Создайте три кнопки, по нажатии на которые будут открываться разные web-страницы (например, с разными картинками). На странице поместить надпись «Я, (ФИО), учусь работать с JavaScript». При наведении курсора на надпись она должна временно менять свой цвет и размер.

Вариант №2. Создайте кнопку, при нажатии на которую надпись на кнопке будет приобретать жирное начертание, а при отпускании кнопки будет возвращено обычное начертание. На странице поместить надпись «Я, (ФИО), учусь работать с JavaScript». При наведении курсора на надпись она должна временно становиться курсивным начертанием.

Вариант №3. Создайте 2 элемента на странице (кнопки, текста или ссылки), при наведении на которые размер шрифта соответствующих элементов увеличивался на 5. На странице поместить надпись «Я, (ФИО), учусь работать с JavaScript». При выходе из области элемента надпись должна менять цвет.

Вариант №4. Создать скрипт, который после загрузки Web-страницы выведет служебное окно с сообщением «Добро пожаловать!». На странице поместить надпись «Я, (ФИО), учусь работать с JavaScript». При наведении курсора на надпись она должна временно становиться жирным начертанием.

Вариант №5. Создать три кнопки, при нажатии на которые будут выводится соответствующие служебные сообщения: «Вы нажали кнопку №\_». На странице поместить надпись «Я, (ФИО), учусь работать с JavaScript». При наведении курсора на надпись она должна временно становиться подчеркнутой.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Поясните суть механизма DHML.

2. Перечислите категории событий.

3. Какие стандартные события вы знаете?

4. Перечислите известные вам события для элементов, получающих фокус ввода.

5. Перечислите известные вам события, используемые для отдельных элементов.

6. Как изменить стиль элемента при возникновении определенного события?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 17**

## **Тема работы: «Создание HTML-страниц с использованием сценариев JavaScript. Использование управляющих конструкций в JavaScript»**

**1. Цель работы**

Формирование умений создания простейших сценариев на языке программирования JavaScript и добавления их на HTML-страницу.

**2. Задание**

Создать HTML-документ, содержащий сценарий на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

**Сценарий** – это код, включенный в состав Web-страницы. Сценарий выполняется либо на клиентском компьютере, когда пользователь работает с элементом управления, либо на Web-сервере перед отправкой страницы клиенту.

Клиентский сценарий выполняется на компьютере-клиенте. Программы просмотра снабжены встроенным интерпретатором, который может считывать и выполнять сценарии.

Основная цель добавления клиентского сценария к Web-странице – создание событийных процедур для элементов управления, которая будет запускать определенную функцию, когда пользователь нажмет соответствующую кнопку.

Серверный сценарий выполняется в рамках активной страницы на Web-сервере до того, как тот вернет пользователю готовую HTML-страницу. Когда пользователь запрашивает активную серверную страницу, сервер выполняет сценарии и создает HTML-код, который и передается пользователю. В результате пользователь не видит серверного сценария на полученной Web-странице.

Язык сценариев (или скриптовый – это язык программирования, который разработан для записи последовательностей операций («сценариев»), выполняемых пользователем на своем компьютере.

Основные правила синтаксиса:

1) чувствительность к регистру. Все ключевые слова пишутся в нижнем регистре. Все переменные и названия функций пишутся точно так же, как и были определены (например, переменные Str и str являются разными переменными);

2) пробелы, табуляция и перевод строки. Эти символы игнорируются в JavaScript, так что можно использовать их для форматирования кода с тем, чтобы его было удобно читать;

3) символ точка с запятой (;). Все операторы должны быть разделены этим символом. Если оператор завершается переводом строки, то точку с запятой можно опустить. При этом нужно следить за тем, чтобы при разрыве строки одного оператора, новая строка не начиналась бы с самостоятельного оператора;

4) комментарии. JavaScript игнорирует любой текст расположенный между символами /\* и \*/. Также игнорируется текст начинающийся символами // и заканчивающийся концом строки;

5) идентификаторы. Идентификаторами являются имена переменных, функций, а также меток. Идентификаторы образуются из любого количества букв ASCII, подчеркивания (\_) и символа доллара ($). Первым символом не может быть цифра, а в версии JavaScript 1.0 не допускается использования и символа $;

6) ключевые слова. Ключевые слова не могут использоваться в качестве идентификаторов. Ключевыми словами являются: break, case, continue, default, delete, do, else, export, false, for, function, if, import, in, new, null, return, switch, this, true, typeof, with.

Для того, чтобы браузер распознал сценарий, встроенный в HTML-код, его (сценарий) следует заключать в контейнер:

<script>...</script>

Пример:

<html>

<head>

<title>Изучаем JavaScript</title>

</head>

<body>

Текст документа

<script>

...

Код сценария

...

</script>

Текст документа

</body>

</html>

Пример:

<HTML>

<BODY>

<SCRIPT LANGUAGE="JavaScript">

<!- Пример простого сценария

document.write("HellofromJavaSript")

// -->

</SCRIPT>

</BODY>

</HTML>

С помощью метода **document.writeln** можно выводить текст с использованием тегов форматирования <I>, <B>, <U> и др. Это позволяет динамически формировать гипертекст.

Используя эти возможности, легко вывести на экран графическое изображение из указанного файла:

<SCRIPT>

document.writeln("<IMGSRC=\"logo.gif\">");

</SCRIPT>

Документ может содержать несколько сценариев – каждый из них должен быть заключен в контейнер **<script>...</script>.** Исключение составляют сценарии, выполняемые при обработке какого-либо события: они (без контейнера <script>...</script>) помещаются непосредственно в соответствующий тэг. Например, при щелчке на кнопке вызывается метод **alert**:

<input type="Button" value="Проверим работу метода alert" onclick="alert('ЭТО НАШ ПЕРВЫЙ JAVASCRIPT-сценарий!!!!');">

Каждая инструкция в JS должна быть отделена точкой с запятой “;”, хотя это не всегда обязательно. Инструкцией называют строку кода, которая эквивалентна некоторому действию, например:

a = 1;

alert(a);

Комментарии – это строки кода, которые не обрабатываются транслятором. Комментарии в JS бывают двух типов – однострочные и многострочные. Однострочные начинаются с символов **“//”** и перестают быть комментарием при переходе на следующую строку. Многострочные начинаются с символов **“/\*”** и заканчиваются символами **“\*/”**. Закрывающие комментарий символы могут находиться на строке, которая ниже текущей. Рассмотрим примеры:

<script>

// This is a single-line comment

a = 54;

/\*

This is multi-line comment.

We can insert some text here.

\*/

b = a + 1;

</script>

Переменная – это символьное представление для некоторого значения, которое сохраняется в оперативной памяти.

Переменные в JS не являются типизированными. Это означает, что при создании переменной не указывается, какого типа информация будет в ней находиться. Для того, чтобы использовать переменную, ее необходимо объявить и инициализировать.

Объявление переменной реализуется при помощи ключевого слова **var**.

Инициализация переменной – это присвоение переменной некоторого начального значения. Инициализация также может иметь место при объявлении переменных.

var a, b;

var x = 12;

t = 10;

В JS, как и в других языках программирования, названия переменных должны соответствовать некоторым правилам. Название переменной в JS может содержать символы английского алфавита, цифры, подчеркивания и символ **$**, однако не может начинаться с цифры. Также, названия переменных регистро-зависимы.

Выражение – это некоторая конструкция, которая состоит из значений, переменных (констант) и операторов. Рассмотрим основные операторы JS. Выделяют:

- арифметические;

- логические;

- сравнения.

К арифметическим операторам относятся следующие бинарные:

**+** (сложение), **-** (вычитание), **\*** (умножение), **/** (деление), **%** (взятие остатка от деления).

Приоритет этих операций совпадает с оригинальным приоритетом в математике. К унарным операторам относятся следующие:

**++** инкремент (увеличить значение переменной на 1), **--** декремент (уменьшить значение переменной на 1), **-** изменение знака на противоположный.

Унарный оператор "**+**" также используется для преобразования строки к числу:

alert(+"123"+1) // 124

В отличие от некоторых других языков, оператор "**+**" делает конкатенацию, если хотя бы один операнд – строка, причем, не обязательно первый.

Пример:

а = “строка1”+”строка2”; // “строка1строка2”

[Логические](http://javascript.ru/basic/operators#logicheskie): **&&** (логическое И), **||** (логическое ИЛИ), **!** (логическое НЕ).

Логические операторы, как правило, используются с булевыми значениями и возвращают булево значение.

Однако, в JS операторы **&&** и **||** возвращают не булево значение, а один из операндов, определивший значение выражения. Конечно, результат в таком случае может быть и не булевым.

«Логическое И» возвращает последний операнд, если все операнды верны. Если хоть один из операндов неверен, то возвратит первый неверный операнд.

Оператор «логического ИЛИ» возвращает первое верное значение. А если верных значений вообще нет, то последнее неверное.

Логическое НЕ, также удобно для преобразования в Boolean.

Сравнения (и отношения): **==** (равно), **!=** (не равно), **<** (строго меньше), **>** (строго больше),**<=** (меньше или равно), **>=** (больше или равно), **===** (равно и тип данных операндов совпадает), **!==** (не равно или не совпадает тип данных операндов).

Операции отношения имеют равный приоритет. Операции сравнения имеют равный друг относительно друга приоритет, но ниже чем приоритет операций отношения.

Пример:

<html>

<head>

<title>Сравнение чисел</title>

<script language="JavaScript">

var first, //первая введенная пользователем строка

second; //вторая введенная пользователем строка

// считываем первое число как строку

first=window.prompt(”Введите первое целое число:”, “0”);

// считываем второе число как строку

second =window.prompt(”Введите второе целое число:”, “0”);

document.writeln(“<h1>Результаты сравнения</h1>”);

document.writeln(“<table border=’1’ width=’100%’>”);

if (first==second)

document.writeln(“<tr><td> “+ first +”==”+ second +” </td></tr>”);

if (first!=second)

document.writeln(“<tr><td> “+ first +”!=”+ second +” </td></tr>”);

if (first > second)

document.writeln(“<tr><td> “+ first +”>”+ second +” </td></tr>”);

if (first<second)

document.writeln(“<tr><td> “+ first +”<”+ second +” </td></tr>”);

if (first<=second)

document.writeln(“<tr><td> “+ first +”<=”+ second +” </td></tr>”);

if (first>=second)

document.writeln(“<tr><td> “+ first +”>=”+ second +” </td></tr>”);

// Окончание вывода результатов

document.writeln(“</table>”);

</script>

</head>

<body>

<p> обновите страницу для перезапуска сценария </p>

</body>

</html>

В примере использовали управляющую структуру **if**, которая используется в программах, в том числе, и для принятия решений на основе того, является ли условие этой структуры истинным или ложным. Если условие выполняется (значение истинно), то выполняется оператор, включенный в эту структуру управления. Если условие не выполняется (значение ложно), то этот оператор не выполняется

Язык JavaScript поддерживает стандартный набор управляющих конструкций:

- последовательная структура;

- три типа структур выбора: **if** (структура с единичным выбором); **if/else** (с двойным выбором) , **switch** (с множественным выбором);

- четыре типа структур повторения: **while**, **do/while**, **for**, **for/in**.

Любая программа может быть составлена на основе выбора нужных для реализации алгоритма программы типов управляющих структур из восьми имеющихся, соединяемых друг с другом только двумя способами: сложение управляющих структур друг за другом и вложение управляющих структур друг в друга.

Пример последовательной структуры:

total=total+grade;

counter=counter+1;

Пример использования if:

if (first==second)

document.writeln(“<tr><td> “+ first +”==”+ second +” </td></tr>”);

если выражение в условии принимает значение **true** – выполняется указанное действие, если **false** – не выполняется.

Пример использования if/else:

if (first==second)

document.writeln(“<tr><td> “+ first +”==”+ second +” </td></tr>”);

else

document.writeln(“<tr><td> “+ first +”!=”+ second +” </td></tr>”);

JavaScript поддерживает единственную тернарную (троичную) операцию «?:», которая используется в качестве сокращенной формы записи конструкции **if...else**:

document.writeln

(first==second ? “(”+ first +””==””+ second+”)”:”(+ first +”!=”+ second +)”);

Для обработки сложных условий используются вложенные структуры **if...else:** когда одни структуры **if...else** помещаются внутрь других структур **if...else**.

if (age < 3) {

message = 'Здравствуй, малыш!';}

else if (age < 18) {

message = 'Привет!';}

else if (age < 100) {

message = 'Здравствуйте!';}

else {

message = 'Какой необычный возраст!';

}

**Switch** представляет собой более наглядный способ сравнить выражение сразу с несколькими вариантами.

Пример использования switch:

switch(x) {

case 'value1': // if (x === 'value1')

...

[break]

case 'value2': // if (x === 'value2')

...

[break]

default:

...

[break]

}

Переменная **x** проверяется на строгое равенство первому значению **value1**, затем второму **value2** и так далее.

Если соответствие установлено – **switch** начинает выполняться от соответствующей директивы **case** и далее, до ближайшего **break** (или до конца **switch**).

Если ни один **case** не совпал – выполняется (если есть) вариант **default**.

Несколько значений **case** можно группировать:

Пример:

var a = 2+2;

switch (a) {

case 4:

alert('Верно!');

break;

case 3: // (\*)

case 5: // (\*\*)

alert('Неверно!');

alert('Немного ошиблись, бывает.');

break;

default:

alert('Странный результат, очень странный');

}

Оператор **while** составляет тело структуры повторений: если условие истинно, то выполняются соответствующие действия; когда условие станет ложным – выполнение структуры повторения будет завершено.

Пример использования while:

i = 0;

while (i < 10) {

document.write(i + «<br>»);

i ++;

} //Данный код выведет столбик из 10 цифр.

Условие в скобках интерпретируется как логическое значение, поэтому вместо **while (i!=0)** обычно пишут **while (i)**:

var i = 3;

while (i) { // при i, равном 0, значение в скобках будет false и цикл остановится

alert( i );

i--;

}

Цикл **do...while** выполняется по меньшей мере один раз до вычисления значения условного выражения:

Пример использования do/while:

var i = 0;

do {

document.write(i + " ");

i++;

} while (i < 10);

// Output: 0 1 2 3 4 5 6 7 8 9

Пример использования for:

for (var i = 0; i <= 9; i++) {

document.write (i);

document.write (" ");

}

// Output: 0 12 3 4 5 6 7 8 9

Пример использования for/in:

var a = [0, 1, 2, 3, 4, 5, 6], props = '';

for (var i in a) {

props += i;

}

alert(props); // В большинстве случаев 0123456

Для управления HTML-страницами и их элементами JavaScript использует объектную модель документов DOM (Document Object Model). Упрощенная схема объектной модели представлена на рисунке 17.1.

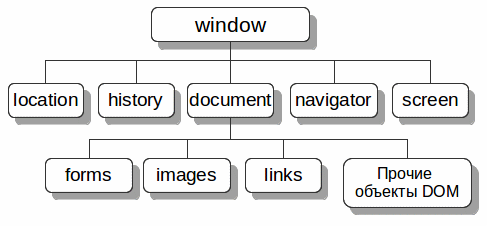


Рисунок 17.1 – Объектная модель документа

Объекты появляются после загрузки документа браузером или как результат работы программы. Каждый объект характеризуется набором методов, свойств и возможных событий.

Свойства объектов соответствуют атрибутам определенных тегов; их использование позволяет получить или изменить характеристики окна браузера, загруженных документов и элементов HTML-страницы. Например, к свойствам изображения относятся src, width, height и т. д.

Методы объекта представляют собой действия, которые могут быть выполнены по отношению к данному объекту. Например, одним из методов объекта document является write, который помещает в документ определенный текст.

Создадим документ pr\_DOM.html:

<HTML>

<HEAD> <TITLE> DOM </TITLE> </HEAD>

<BODY>

<FORM name=Sum>

Первое число <INPUT type="text" name="number1"><BR> Второе число <INPUT type="text" name="number2"><BR>

<INPUT type="button" name="but"

value="Вычислить сумму"><BR><BR> Сумма <INPUT type="text" name="result">

</FORM>

</BODY>

</HTML>

При открытии данного документа создается объектная модель, представленная на рисунке 17.2.

**window**

**document**

**forms[0] name=Sum**

**elements[0] name=number1**

**elements[1] name=number2**

**elements[2] name=but**

**elements[3] name=result**

Рисунок 17.2 – Иерархия объектов документа pr\_DOM.html

Изменим описание кнопки на странице следующим образом:

<input type="button" name="but" value=" Вычислить сумму "

onClick="document.forms[0].elements[3].value= parseInt(document.forms[0].elements[0].value) + parseInt(document.forms[0].elements[1].value);">

В результате изменений при нажатии кнопки значения, введенные пользователем в два первых текстовых поля, будут переводиться в числовой тип данных (функцией **parseInt**) и суммироваться; результат будет помещен в третье текстовое поле.

Объект **window** можно не указывать (как в данном примере), если имеется в виду текущее окно браузера.

Для обращения к объектам страницы можно также использовать имя, указанное в атрибуте **name**.

Например, обработку нажатия кнопки можно определить следующим образом:

<input type="button" name="but" value=" Вычислить сумму "

onClick="document.Sum.result.value=parseInt(document.Sum.number1.value) + parseInt(document.Sum.number2.value);">

Для расчета суммы можно определить отдельную функцию **summa( )** и вызвать ее при помощи атрибута **onClick**:

<html>

<head>

<title> DOM </title>

<script>

// определяем функцию summa()

function summa(){

// объявляем переменные n1 и n2

var n1=parseInt(document.Sum.number1.value);

var n2=parseInt(document.Sum.number2.value);

/\*рассчитываем сумму и записываем ее в поле result\*/

document.Sum.result.value=n1+n2;

}

</script>

</head>

<body>

<form name=Sum>

...

<input type="button" name="but" value="Вычислить сумму" onClick="summa()">

...

</form>

</body>

</html>

**5. Порядок выполнения работы**

1. Создайте HTML-документ Lab17\_1.html. Установите название документа, цвет фона и текста.
2. Создайте сценарий JavaScript, который при загрузке данного документа будет вызывать окно, запрашивающее имя пользователя (используйте метод prompt).
3. Введенное имя сохраните в переменной name.
4. Выведите на страницу приветствие вида «Здравствуйте, name!». Если имя не введено, приветствие должно выглядеть следующим образом: «Здравствуйте! Было бы приятно с Вами познакомиться».
5. При щелчке левой кнопкой мыши на приветствии должно появляться окно confirm, в котором содержится вопрос «Вы хотите изменить имя пользователя?». Если ответ утвердительный, вновь вызовите окно запроса имени и соответствующим образом измените имя пользователя на странице.
6. Создайте документ Lab17\_2.html, в соответствии с вариантом, приведенным в таблице 17.2. Определите соответствующие поля формы для получения входных данных. Пользователю должна быть предоставлена возможность выбора формы предъявления результата: в стандартном окне alert или на той же странице.

Таблица 17.2 – Варианты заданий

|  |  |
| --- | --- |
| **Вариант** | **Задание** |
| 1 | Вычислите сумму n натуральных чисел |
| 2 | Определите n-й член ряда Фибоначчи |
| 3 | Вычислите факториал числа |
| 4 | Рассчитайте сумму двух чисел |
| 5 | Рассчитайте разность двух чисел |
| 6 | Рассчитайте произведение двух чисел |
| 7 | Определите частное двух чисел |
| 8 | Найдите корни квадратного уравнения |
| 9 | Определите, является ли високосным год, введенный пользователем |
| 10 | Определите сумму n членов ряда Фибоначчи |
| 11 | Определите длину окружности по ее радиусу |
| 12 | Определите результат вычисления арифметического выражения, введенного в текстовое поле |
| 13 | Найдите наименьший общий делитель двух чисел |
| 14 | Найдите наибольший общий делитель двух чисел |
| 15 | Найдите площадь прямоугольника по его ширине и высоте |
| 16 | Найти площадь треугольника |
| 17 | Рассчитать площадь круга вписанного в квадрат со стороной а |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Укажите способы использования кода JavaScript в составе HTML-страницы.
2. На HTML-странице определена следующая форма:

<FORM name="f1">

<INPUT type="text" name="txt"> <br>

<INPUT type="button" name="but"> <br>

</FORM>

Каким образом можно поместить в поле ввода txt некоторый текст, изменить размер данного поля, изменить надпись на кнопке?

1. Как в JavaScript определяются комментарии?
2. Опишите управляющие конструкции, поддерживаемые в JavaScript.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 18**

## **Тема работы: «Работа с массивами в JavaScript. Применение методов работы с массивами»**

**1. Цель работы**

Формирование умений работы с массивами, применения методов работы с массивами в JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии работы с массивами на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Массив – это группа ячеек памяти, которые объединены общим именем и, как правило, имеют одинаковый тип данных (хотя это и необязательно).

Массивы в языке JavaScript являются нетипизированными: элементы массива могут иметь любой тип, причем разные элементы одного и того же массива могут иметь разные типы. Элементы массива могут даже быть объектами или другими массивами, что позволяет создавать сложные структуры данных, такие как массивы объектов и массивы массивов.

Отсчет индексов массивов в языке JavaScript начинается с нуля – первый элемент массива имеет индекс 0.

Первый способ создания массива - с помощью литерала, который представляет собой простой список разделенных запятыми элементов массива в квадратных скобках. Значения в литерале массива не обязательно должны быть константами - это могут быть любые выражения, в том числе и литералы объектов:

var empty = []; // Пустой массив

var numbers = [2, 3, 5, 7, 11]; // Массив с пятью числовыми

элементами

var misc = [ 1.1, true, "a", ]; // 3 элемента разных типов +

завершающая запятая

var base = 1024;

var table = [base, base+1, base+2, base+3]; // Массив с переменными

var arrObj = [[1,{x:1, y:2}], [2, {x:3, y:4}]]; // 2 массива внутри,

содержащие объекты

Другой способ создания массива состоит в вызове конструктора Array(). Вызвать конструктор можно тремя разными способами:

1. вызвать конструктор без аргументов:

var arr = new Array();

В этом случае будет создан пустой массив, эквивалентный литералу [].

1. вызвать конструктор с единственным числовым аргументом, определяющим длину массива:

var arr = new Array(10);

В этом случае будет создан пустой массив указанной длины. Такая форма вызова конструктора Array() может использоваться для предварительного распределения памяти под массив, если заранее известно количество его элементов.

1. явно указать в вызове конструктора значения первых двух или более элементов массива или один нечисловой элемент:

var arr = new Array(5, 4, 3, 2, 1, "тест");

В этом случае аргументы конструктора становятся значениями элементов нового массива. Использование литералов массивов практически всегда проще, чем подобное применение конструктора Array().

Элементами массива могут быть переменные любого типа. Интересная особенность JavaScript – массив может одновременно содержать элементы различных типов, в том числе массивы:

Пример: массив с элементами разных типов

//

var myArrayS = new Array();

myArrayS[0] = 'Ярославль';

myArrayS[1] = 10000;

myArrayS[2] = 'Ямал';

myArrayS[3] = 5000;

for(i = 0; i < myArrayS.length; i += 2)

alert('Город: ' + myArrayS[i] + '\n' +

'Население: ' + myArrayS[i + 1] + '\n')

Пример: массив с элементами-массивами

//

var myArrayA = new Array();

myArrayA[0] = new Array('Ярославль', 10000);

myArrayA[1] = new Array('Ставрополь', 5000);

for(i = 0; i < myArrayA.length; i++)

alert('Город: ' + myArrayA[i][0] + '\n' +

'Население: ' + myArrayA[i][1] + '\n')

Чтобы узнать длину массива, необходимо использовать свойство **length**:

var myArray = new Array();

...

// узнать количество элементов

alert(myArray.length)

Значение **Array.length** на единицу больше номера последнего элемента массива, т.к. нумерация в массиве начинается с нуля, а свойство **length** показывает общее количество элементов.

Имеется возможность самостоятельно менять **length**:

var a = [1, 2, 3];

a.length = 5;

alert(a.length + '; ' + [0 in a, 1 in a, 2 in a, 3 in a, 4 in a]); // 5;

// true,true,true,false,false

Видим, что length равен 5, хотя элементов в массиве всего 3.

В общем случае невозможно узнать, сколько элементов в массиве, не перебирая их.

При добавлении элементов массивы автоматически определяют значение свойства **length**. При удалении элементов **length** не изменяется.

Доступ к элементам массива осуществляется с помощью оператора **[]**. Слева от скобок должна присутствовать ссылка на массив. Внутри скобок должно находиться произвольное выражение, возвращающее неотрицательное целое значение. Этот синтаксис пригоден как для чтения, так и для записи значения элемента массива. Следовательно, допустимы все приведенные далее JavaScript-инструкции:

// Создать массив с одним элементом

var arr = ["world"];

// Прочитать элемент 0

var value = arr[0];

// Записать значение в элемент 1

arr[1] = 3.14;

// Записать значение в элемент 2

i = 2; arr[i] = 3;

// Записать значение в элемент 3

arr[i + 1] = 'привет';

// Прочитать элементы 0 и 2, записать значение в элемент 3

arr[arr[i]] = arr[0];

Добавление элементов производится простой инициализацией соответствующего элемента:

var myArray = new Array();

myArray[0] = 'Питер';

Удалять элементы массива можно с помощью оператора **delete**, как обычные свойства объектов:

var arr = [1,2,'three'];

delete arr[2];

2 in arr; // false, индекс 2 в массиве не определен

arr.length; // 3: оператор delete не изменяет свойство length массива

Для перебора элементов обычно используется цикл:

var arr = ["Яблоко", "Апельсин", "Груша"];

for (var i = 0; i < arr.length; i++) {

alert( arr[i] );

}

Наиболее правильной конструкцией для перебора элементов массива является:

for (var i = 0, length = a.length; i < length; i++) {

if (i in a) {

// делаем что-то с элементом массива.

}

}

Пример использования массива:

<html>

<head>

<script>

function start ()

{

var colors = new Array ("cyan", "magenta", "yellow", "black");

var integers1 = [2,4,6,8];

var integers2 = [2, , ,8];

outputArray ("Содержимое массива colors", colors);

outputArray ("Содержимое массива integers1", integers1);

outputArray ("Содержимое массива integers2", integers2);

}

function outputArray (header,theArray)

{

document.writeln ("<H2>"+header+"</H2>");

document.writeln ("<TABLE BORDER=’1’ WIDTH=’100%’>");

document.writeln ("<TR><TD WIDTH=’100’><B>Индекс</B>"+"<TD><B>Значение</B></TR>");

for (var i=0; i<theArray.length; i++){

document.writeln ("<TR><TD>"+i+"<TD>"+theArray[i]+"</TR>");}

document.writeln ("</TABLE>");

}

</script>

</head>

<body onload= "start()"></body>

</html>

Многомерный массив – это массив, значениями которого являются другие массивы.

JavaScript не поддерживает «настоящие» многомерные массивы. Однако, в JavaScript массивы могут содержать значения любых типов, в том числе и другие массивы.

var arr = [

  [1,2,3],

  [4,5,6],

  [7,8,9],

];

Для доступа к элементам, которые в качестве значения содержат массив, используются всё те же квадратные скобки. В первых скобках указывается индекс элемента для доступа к внутреннему массиву, так как в качестве значения элемента содержится ещё один массив, то во вторых скобках указывается индекс элемента внутреннего массива:

Пример:

var arr = [  [1,2,3],  [4,5,6],  [7,8,9],];

alert(arr[1][2]);   // 6

Для перебора многомерного массива придётся использовать вложенный цикл. Первый цикл будет использоваться для доступа к элементам внешнего массива, а второй – для доступа к элементам вложенных массивов:

var arr = [

  [1,2,3],

  [4,5,6],

  [7,8,9],

];

for(let j = 0; j < arr.length; j++) {

  for(let n = 0; n < arr.length; n++) {

    if (n === arr.length - 1)

      document.write(arr[j][n]);

    else

      document.write(arr[j][n] + ", ");

  }

  document.write("<br>");

}

Метод **push** добавляет переданные элементы в конец массива:

var a = [];

a.push(5, 6, 7);

alert(a.length + ';' + a); // 3;5,6,7

Метод **pop** возвращает последний элемент массива и удаляет его:

var a = [];

a.push(5, 6, 7);

alert(a.pop()); // 7

alert(a.length + ';' + a); // 2;5,6

Метод **unshift** добавляет переданные элементы в начало массива, при этом существующие элементы в массиве смещаются в позиции с более высокими индексами. Элементы будут располагаться в том же порядке, в каком они были переданы:

var a = [1];

a.unshift(9, 8, 7);

alert(a.length + ';' + a); // 4;9,8,7,1

Метод **shift** возвращает первый элемент массива и удаляет его:

var a = [1];

a.unshift(9, 8, 7);

alert(a.shift()); // 9

alert(a.length + ';' + a); // 3;8,7,1

Метод **splice** может удалять элементы из массива, вставлять новые элементы или выполнять обе операции одновременно. Он изменяет значение свойства length и сдвигает элементы массива с более низкими или высокими индексами по мере необходимости, чтобы после вставки или удаления образовывалась непрерывная последовательность.

Первый аргумент метода **splice()** определяет позицию в массиве, начиная с которой будет выполняться вставка и/или удаление. Если первый аргумент отрицательный, то индекс, с которого начнётся удаление, будет равен «**length + первый аргумент**». Второй аргумент определяет количество элементов, которые должны быть удалены (вырезаны) из массива. Если второй аргумент опущен, удаляются все элементы массива от указанного до конца массива. За этими аргументами может следовать любое количество дополнительных аргументов, определяющих элементы, которые будут вставлены в массив, начиная с позиции, указанной в первом аргументе.

Метод **splice()** возвращает массив удаленных элементов или (если ни один из элементов не был удален) пустой массив:

var arr = [1,2,3,4,5,6,7,8];

arr.splice(4); // Вернет [5,6,7,8], arr = [1,2,3,4]

arr.splice(1,2); // Вернет [2,3], arr = [1,4]

arr.splice(1,1); // Вернет [4]; arr = [1]

Метод join() преобразует все элементы массива в строки, объединяет их и возвращает получившуюся строку. В необязательном аргументе методу можно передать строку, которая будет использоваться для отделения элементов в строке результата. Если строка-разделитель не указана, используется запятая.

Например, следующий фрагмент дает в результате строку "1,2,3":

var arr = [1,2,3];

arr.join(); // '1,2,3'

arr.join("-"); // '1-2-3'

Метод **reverse** меняет порядок следования элементов в массиве на обратный и возвращает переупорядоченный массив. Перестановка выполняется непосредственно в исходном массиве, т.е. этот метод не создает новый массив с переупорядоченными элементами, а переупорядочивает их в уже существующем массиве:

var arr = [1,2,3];

arr.reverse().join(); // "3,2,1"

Метод **sort** сортирует элементы в исходном массиве и возвращает отсортированный массив. Если метод sort() вызывается без аргументов, сортировка выполняется в алфавитном порядке (для сравнения элементы временно преобразуются в строки, если это необходимо). Неопределенные элементы переносятся в конец массива.

Для сортировки в каком-либо ином порядке, отличном от алфавитного, методу **sort()** можно передать функцию сравнения в качестве аргумента:

Array.sort(функция\_сортировки);

Эта функция устанавливает, какой из двух ее аргументов должен следовать раньше в отсортированном списке. Если первый аргумент должен предшествовать второму, функция сравнения должна возвращать отрицательное число. Если первый аргумент должен следовать за вторым в отсортированном массиве, то функция должна возвращать число больше нуля. А если два значения эквивалентны (т.е. порядок их следования не важен), функция сравнения должна возвращать 0:

var arr = [33, 4, 1111, 222];

arr.sort(); // Алфавитный порядок: 1111, 222, 33, 4

arr.sort(function(a,b) {

// Числовой порядок: 4, 33, 222, 1111

return a-b; // Возвращает значение < 0, 0 или > 0

// в зависимости от порядка сортировки a и b

});

Сортируем в обратном направлении, от большего к меньшему:

arr.sort(function(a,b) {return b-a});

Метод **concat** создает и возвращает новый массив, содержащий элементы исходного массива, для которого был вызван метод concat(), и значения всех аргументов, переданных методу concat(). Если какой-либо из этих аргументов сам является массивом, его элементы добавляются в возвращаемый массив. Следует, однако, отметить, что рекурсивного превращения массива из массивов в одномерный массив не происходит. Метод concat() не изменяет исходный массив. Ниже приводится несколько примеров:

var arr = [1,2,3];

arr.concat(4, 5); // Вернет [1,2,3,4,5]

arr.concat([4,5]); // Вернет [1,2,3,4,5]

arr.concat([4,5],[6,7]) // Вернет [1,2,3,4,5,6,7]

arr.concat(4, [5,[6,7]]) // Вернет [1,2,3,4,5,[6,7]]

Метод **slice** возвращает фрагмент, или подмассив, указанного массива. Два аргумента метода определяют начало и конец возвращаемого фрагмента. Возвращаемый массив содержит элемент, номер которого указан в первом аргументе, плюс все последующие элементы, вплоть до (но не включая) элемента, номер которого указан во втором аргументе.

Если указан только один аргумент, возвращаемый массив содержит все элементы от начальной позиции до конца массива. Если какой-либо из аргументов имеет отрицательное значение, он определяет номер элемента относительно конца массива. Так, аргументу -1 соответствует последний элемент массива, а аргументу -3 - третий элемент массива с конца. Вот несколько примеров:

var arr = [1,2,3,4,5];

arr.slice(0,3); // Вернет [1,2,3]

arr.slice(3); // Вернет [4,5]

arr.slice(1,-1); // Вернет [2,3,4]

arr.slice(-3,-2); // Вернет [3]

**5. Порядок выполнения работы**

1. Создайте массив musicStyles c 3-мя произвольными элементами.
2. Добавьте в конец массива новое значение
3. Замените предпоследнее значение на «Блюз». Учтите, что код не должен быть привязан к конкретному количеству элементов.
4. Удалите первое значение и выведите его в alert.
5. Добавьте в начало 2 новых значения.
6. Отсортировать элементы массива по убыванию, без учета регистра.
7. Создавать массив с размерностью (№ по журналу+4). Циклом While вывести элементы с 10 по 1. Для вывода использовать document.write().
8. Написать примеры кода, используя методы .join(Array) и .split(string)
9. Выполнить транспанирование матрицы.
10. Найти min и max элементы массива.
11. Выполнить индивидуальное задание в соответствии с вариантом.

Вариант 1. Найти сумму элементов массива кратных данному k.

Вариант 2. Дан массив состоящий из строк и чисел. Выяснить, какое число встречается раньше –положительное или отрицательное.

Вариант 3. Дана последовательность натуральных чисел. Создать массив из четных чисел этой последовательности. Если таких чисел нет, то вывести сообщение об этом.

Вариант 4. Определить, сколько процентов от всего количества элементов массива составляют четные элементы.

Вариант 5. Дан одномерный массив. Посчитать, сколько элементов массива до первого нулевого элемента.

Вариант 6. В массиве поменять местами наибольший и наименьший элементы.

Вариант 7. Подсчитать в массиве количество отрицательных, положительных и нулевых элементов.

Вариант 8. Вывести на экран элементы массива, удовлетворяющие следующему условию M[i]>i.

Вариант 9. Определить, сколько процентов от всего количества элементов массива составляют нулевые элементы.

Вариант 10. В массиве A записаны оценки экзаменующихся, полученные на первом экзамене. Посчитать, сколько человек не допущено ко второму экзамену.

Вариант 11. Определить, сколько процентов от всего количества элементов массива составляют положительные элементы.

Вариант 12. Определить, сколько элементов в массиве больше k, меньше k и равных k.

Вариант 13. Вычислить среднее арифметическое элементов массива.

Вариант 14. Определить сколько процентов от всего количества элементов массива составляют нечетные элементы.

Вариант 15. Вычислить сумму элементов массива, порядковые номера которых совпадают со значениями этого элемента.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Опишите способы создания массивов.
2. Перечислите способы изменения длины массива.
3. Каким образом можно создать двумерный массив в JavaScript?
4. Опишите методы работы с массивами, поддерживаемые в JavaScript.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 19**

## **Тема работы: «Использование функций при процедурном подходе в программировании на языке JavaScript. Управление видимостью переменных при помощи замыканий»**

**1. Цель работы**

Формирование умений написания функции на языке JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии работы с функциями на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Функция в JavaScript – специальный тип объектов, позволяющий формализовать средствами языка определённую логику поведения и обработки данных.

В Javascript функции являются полноценными объектами встроенного класса Function. Поэтому их можно присваивать переменным, передавать и, конечно, у них есть свойства.

Существует три способа объявления функции: Function Declaration, Function Expression и Named Function Expression.

Function Declaration (FD, декларативный стиль) – это "классическое" объявление функции. В JavaScript функции объявляются с помощью литерала функции. Синтаксис объявления FD:

//Объявление функции

function имяФункции(пер1, пер2){

Код функции

}

//Вызов функции

имяФункции(пер1,пер2);

function sayHi() {

  alert("Hello");

}

Встречая ключевое слово **function** интерпретатор создаёт функцию и затем присваивает ссылку на неё переменной с именем **sayHi** (переменная с данным именем создаётся интерпретатором автоматически).

**Function Expression** (FE, функциональный стиль) – это объявление функции, которое является частью какого-либо выражения (например, присваивания). Такие функции, как правило, анонимны. Синтаксис объявления FE:

//Объявление функции

var имяфункции=function(пер1, пер2){Код функции}

//Вызов функции

имяфункции(пер1,пер2);

var sayHi = function () {

  alert("Hello");

}

**Named Function Expression** (сокращённо NFE) – это объявление функции, которое является частью какого-либо выражения (например, присваивания). Синтаксис объявления NFE:

var sayHi = function foo() {

  alert("Hello");

}

Объявления FE и NFE обрабатываются интерпретатором точно так же, как и объявление FD: интерпретатор создаёт функцию и сохраняет ссылку на неё в переменной **sayHi**.

Существует ещё один способ создания функции. Он позволяет создавать функцию полностью «на лету» из строки:

var sum = new Function('a,b', ' return a+b; ');

var result = sum(1, 2);

alert( result ); // 3

то есть, функция создаётся вызовом:

new Function(params, code):

где params – параметры функции через запятую в виде строки;

сode – код функции в виде строки.

Программный код, расположенный в теле функции, выполняется не в момент объявления функции, а в момент её вызова. Для вызова функции используется оператор **()** ([вызов функции](http://www.puzzleweb.ru/javascript/6_operators4.php#a3)):

function sayHi() {

  alert("Hello");

}

sayHi();    // "Hello"

Разница между представленными тремя объявлениями заключается в том, что функции, объявленные как FD, создаются интерпретатором до начала выполнения кода (на этапе анализа), поэтому их можно вызывать в своей области видимости до объявления:

// Вызов функции до её объявления в коде верхнего уровня

foo();

function foo() {

  alert("Вызов функции foo() в глобальной области видимости.");

  // Вызов функции до её объявления в области видимости функции

  bar();

  function bar() {

    alert("Вызов функции bar() в области видимости функции.");

  }

}

Функции, объявленные как FE или NFE, создаются в процессе выполнения кода, поэтому их можно вызывать только после того как они объявлены:

// sayHi();  // Ошибка. Функция sayHi ещё не существует

var sayHi = function () {

  alert("Hello!");

}

sayHi();

В JavaScript есть три области видимости: глобальная, область видимости функции и блочная.

Область видимости переменной – это участок исходного кода программы, в котором переменные и функции видны и их можно использовать.

Переменная, которая была объявлена с помощью ключевого слова var или let за пределами функции, называется глобальной переменной (global variable). Глобальные переменные уничтожаются, когда вы закрываете страницу. Такая переменная имеет глобальную область видимости, это означает, что она доступна в любом месте исходного кода:

var str = "Hello!";   // Глобальная переменная

function foo() {

  alert(str);

}

foo();                // Доступна внутри функции

alert(str);           // Доступна вне функции

Переменная, объявленная внутри блока с помощью ключевого слова let, называется переменной блочного уровня (block level variable) или кратко блочной переменной. Такая переменная находится в блочной области видимости, это означает, что она доступна только внутри блока:

{

  let a = 5;

}

alert(a);   // Ошибка. Переменная a не видна за пределами блока

Переменная, объявленная с помощью ключевого слова var или let внутри функции, называется локальной переменной (local variable). Такая переменная находится в области видимости функции, это означает, что она доступна в любом месте внутри тела функции, в которой была объявлена. Локальная переменная создаётся каждый раз при вызове функции и уничтожается при выходе из неё (при завершении работы функции):

function foo() {

  var str = "Hello!";   // Локальная переменная

  alert(str);

}

foo();                  // Доступна внутри функции

alert(str);             // Ошибка. Не доступна вне функции

Локальная переменная имеет преимущество перед глобальной переменной с тем же именем, это означает, что внутри функции будет использоваться локальная переменная, а не глобальная:

var x = "глобальная";   // Глобальная переменная

function checkscope() {

  var x = "локальная";  // Локальная переменная с тем же именем, что и у глобальной

  document.write(x);    // Используется локальная переменная, а не глобальная

}

checkscope();           // => "локальная"

Если вы объявляете переменную, не используя "var", переменная всегда становится глобальной.

Рассмотрим следующий пример:

var num = 5;

function foo() {

  var num2 = 10;

  function bar() {

    var num3 = 15;

  }

}

В этом коде три области видимости: глобальная, область видимости функции **foo()** и область видимости функции **bar()**. В глобальной области видимости определены переменная **num** и функция **foo()**. В области видимости функции **foo()** определены переменная **num2** и функция **bar()**, в ней также доступна переменная **num** из глобальной области видимости. Область видимости функции **bar()** содержит одну переменную **num3**, которая доступна только внутри функции **bar()**. В области видимости функции **bar()** также доступны переменные из двух других областей, потому что они являются родительскими по отношению к ней. Цепочка областей видимости для этого примера представлена на рисунке 19.1:

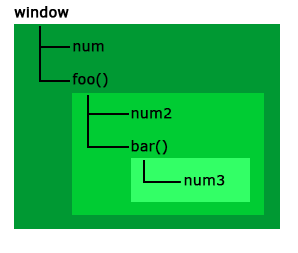


Рисунок 19.1 – Цепочка областей видимости

На рисунке разные области видимости показаны прямоугольниками разного цвета. Цепочка областей видимости упорядочена. Интерпретатор производит поиск идентификаторов в цепочке областей видимости по направлению наружу, но не внутрь.

Все функции имеют два неявных параметра: **arguments** и **this**.

Параметр **arguments** является объектом, который содержит все аргументы переданные функции.

Объект **arguments** похож на массив, но не имеет никаких его методов и свойств, кроме свойства **length**. Поскольку он похож на массив, обратиться к переданным функции аргументам можно так же, как и к элементам массива – с помощью квадратных скобок, внутри которых указывается индекс аргумента. Аргументы нумеруются, начиная с нуля:

function foo () {

  alert("Переданные аргументы: " + arguments[0] + ", " + arguments[1] +

        "\nВсего передано аргументов: " + arguments.length);

}

foo(5, 10);

С помощью инструкции **return** функция может возвратить некоторое значение (результат работы функции) программе, которая её вызвала. Возвращаемое значение передаётся в точку вызова функции.

Инструкция **return** имеет следующий синтаксис:

return выражение;

В программу возвращается не само выражение, а результат его вычисления. Для дальнейшего использования возвращаемого значения, результат выполнения функции можно присвоить, к примеру, переменной:

Пример:

function calc(a) {

  return a \* a;

}

var x = calc(5);

alert(x);   // 25

Инструкция **return** может быть расположена в любом месте функции. Как только будет достигнута инструкция **return**, функция возвращает значение и немедленно завершает своё выполнение. Код, расположенный после инструкции **return**, будет проигнорирован:

function foo() {

  return 1;

  alert('Не выполнится');

}

var x = foo();

alert(x);   // 1

Внутри функции можно использовать несколько инструкций **return**:

function check(a, b) {

  if(a > b) return a;

  else return b;

}

alert(check(3, 5));   // 5

Рекурсия – это когда функция в своём теле вызывает саму себя. Функцию, которая вызывает сама себя, называют рекурсивной функцией. Вызов рекурсивной функции, называется рекурсивным вызовом.

В качестве примера, вычислим факториал с использованием рекурсии:

function f(n) {

  if (n === 1) return 1;

  return n \* f(n - 1);

}

alert(f(4));

Визуально последовательное выполнение данной функции можно представить как показано на рисунке 19.2:

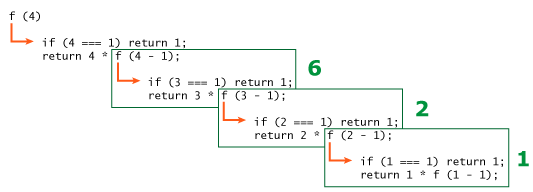


Рисунок 19.2 – Последовательность выполнения рекурсивной функции

Выполнение программы многократно спускается вниз, пока не упрётся в условие выхода из рекурсии. Достигнув конца, она идёт обратно, возвращая результаты сделанных вызовов.

Замыкание — это выражение (обычно функция), которое может иметь свободные переменные, вместе со средой, которая привязывает эти переменные (т.е. “замыкает” это выражение).

Рассмотрим, что происходит внутри функции.

function greeting(name) {

  // LexicalEnvironment = {name: 'Николай', text: undefined}

  var text = 'Здравствуйте, ' + name;

  // LexicalEnvironment = {name: 'Николай', text: 'Здравствуйте, Николай'}

  alert(text);

}

greeting('Николай');

Когда функция вызывается, у нее создается объект **LexicalEnvironment**, в который записываются все локальные переменные и функции, а также ссылка на внешнюю область видимости (об этом позже). В нашем случае у нас есть локальная переменная **name**, у которой сразу есть значение (то, которое мы передаем) и это "Николай". В самом начале функции уже есть переменная **text**, интерпретатор знает про нее, но так как мы еще не дошли по присваивания этой переменной какого-то значения, то она равна **undefined**. Теперь мы присваиваем переменной значение, и наш объект **LexicalEnvironment** меняется. Его свойство **text** становится равным тому, что мы записали ("Здравствуйте, Николай" в нашем случае). После того, как функция отработала, объект **LexicalEnvironment** уничтожается. При последующих вызовах функции он будет создан снова и т.д.

Все переменные внутри функции – это свойства специального внутреннего объекта **LexicalEnvironment**, который создаётся при её запуске.

Таким образом, при запуске функция создает объект **LexicalEnvironment**, записывает туда аргументы, функции и переменные. Процесс инициализации выполняется в том же порядке, что и для глобального объекта, который, вообще говоря, является частным случаем лексического окружения.

В отличие от **window**, объект **LexicalEnvironment** является внутренним, он скрыт от прямого доступа.

Интерпретатор, при доступе к переменной, сначала пытается найти переменную в текущем **LexicalEnvironment**, а затем, если её нет – ищет во внешнем объекте переменных.

Такой порядок поиска возможен благодаря тому, что ссылка на внешний объект переменных хранится в специальном внутреннем свойстве функции, которое называется **[[Scope]]**. Это свойство закрыто от прямого доступа, но знание о нём очень важно для понимания того, как работает JavaScript.

При создании функция получает скрытое свойство **[[Scope]]**, которое ссылается на лексическое окружение, в котором она была создана.

Это свойство никогда не меняется. Оно всюду следует за функцией, привязывая её, таким образом, к месту своего рождения.

При запуске функции её объект переменных **LexicalEnvironment** получает ссылку на «внешнее лексическое окружение» со значением из **[[Scope]]**.

Если переменная не найдена в функции – она будет искаться снаружи.

Объект **LexicalEnvironment** уничтожается каждый раз после выполнения функции и создается снова при повторном вызове.

Замыкания применяются для того, чтобы можно было сохранить все, что записано в LexicalEnvironment, и использовать при следующих вызовах.

Рассмотрим более «продвинутый» вариант, при котором внутри одной функции создаётся другая и возвращается в качестве результата.

В разработке интерфейсов это совершенно стандартный приём, функция затем может назначаться как обработчик действий посетителя.

Здесь мы будем создавать функцию-счётчик, которая считает свои вызовы и возвращает их текущее число.

В примере ниже makeCounter создает такую функцию:

function makeCounter() {

var currentCount = 1;

return function() { // (\*\*)

return currentCount++;

};

}

var counter = makeCounter(); // (\*)

// каждый вызов увеличивает счётчик и возвращает результат

alert( counter() ); // 1

alert( counter() ); // 2

alert( counter() ); // 3

// создать другой счётчик, он будет независим от первого

var counter2 = makeCounter();

alert( counter2() ); // 1

Как видно, мы получили два независимых счётчика **counter** и **counter2**, каждый из которых незаметным снаружи образом сохраняет текущее количество вызовов во внешней переменной **currentCount**, которая у каждого счётчика своя.

Переменную во внешней области видимости можно не только читать, но и изменять.

**5. Порядок выполнения работы**

1. Вычислить сумму чисел, представляющих собой последовательность, начинающуюся с 0 и 1, в которой каждое последующее число является суммой двух предыдущих чисел (числа Фибоначчи). Поместите эту функцию в сценарий, который должен получать от пользователя значение индекса числа в последовательности с помощью HTML-формы.
2. Напишите функцию, которая выполняет возведение целого числа в целую степень. Поместите эту функцию в сценарий, который должен получать от пользователя значение числа и значение необходимой степени с помощью HTML-формы.
3. Напишите функцию distance, которая вычисляет расстояние между двумя точками (х1,у1) и (х2, у2). Все исходные и возвращаемые значения должны быть числами с плавающей точкой. Поместите эту функцию в сценарий, который должен получать от пользователя координаты точек с помощью HTML-формы.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой функция в JavaScript?
2. Перечислите способы объявления функций.
3. Что представляет собой область видимости переменной?
4. Опишите виды переменных в JavaScript.
5. Для чего используется инструкция return?
6. В чем заключается механизм замыканий?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 20**

## **Тема работы: «Создание объектов и методов. Управление контекстом вызова»**

**1. Цель работы**

Формирование умений создания объектов и методов на языке JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии работы с объектами на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

В JavaScript почти все является объектами. Все примитивные типы за исключением **null** и **undefined** обрабатываются как объекты. Они могут быть назначены свойствам, и у них есть все характеристики, присущие объектам.

В JavaScript объект – это самостоятельная единица, имеющая свойства и определенный тип. Свойство объекта можно понимать как переменную, закрепленную за объектом.

Основные операции, производимые с объектами, – это добавление новых свойств, изменение уже существующих свойств, удаление свойств и обращение к свойствам.

Пустой объект может быть создан одним из двух синтаксисов: с помощью литерала объекта или оператора **new** с конструктором:

o = new Object();

o = {}; // пустые фигурные скобки

Например, объект Car:

var Car = new Object();

При использовании инициализатора объекта сразу задается как объект, так и значения. Для этого используется следующий синтаксис:

ИмяОбъекта = { Свойство1: Значение1, Свойство2: Значение2, ..., СвойствоN: ЗначениеN }

Следующие два фрагмента кода создают одинаковый объект:

var menuSetup = {

width: 300,

height: 200,

title: "Menu"

};

// то же самое, что:

var menuSetup = {};

menuSetup.width = 300;

menuSetup.height = 200;

menuSetup.title = 'Menu';

Следующий пример создает объект **myHonda** с тремя свойствами, свойство **engine**, в свою очередь, также является объектом со своими собственными свойствами.

var myHonda = {

color: "red",

wheels: 4,

engine: {

cylinders: 4,

size: 2.2

}

};

alert(myHonda. color) // " red "

alert(myHonda. engine. cylinders) // 4

Использование функции конструктора – другой способ создать объект в два шага:

1) определить объект, набрав функцию-конструктор. Название такой функции, как правило, начинается с заглавной буквы;

2) создать экземпляр объекта с помощью ключевого слова **new**.

Например, необходимо создать тип объект для **cars**. Объект этого типа должен называться **car**, и иметь свойства **make**, **model**, и **year**.

function Car(make, model, year) {

this.make = make;

this.model = model;

this.year = year;

}

Теперь можно создать объект, называемый **mycar**, следующим образом:

var mycar = new Car("Eagle", "Talon TSi", 1993);

В результате вызова new Car("Eagle", "Talon TSi", 1993) будет создан такой объект:

mycar = {

make:” Eagle”,

model: "Talon TSi",

year: 1993

}

Конструктором становится любая функция, вызванная через **new**.

Технически, любая функция может быть использована как конструктор. То есть, любую функцию можно вызвать при помощи **new**. Как-то особым образом указывать, что она – конструктор – не надо. Но, чтобы выделить функции, задуманные как конструкторы, их называют с большой буквы.

Теперь многократными вызовами **new Car** с разными параметрами можно создать столько объектов, сколько нужно. Поэтому такую функцию и называют конструктором – она предназначена для «конструирования» объектов.

Добавить свойство можно и после создания объекта:

mycar.color = "black";

Для доступа к свойству используется один из операторов доступа: **.** ([точка](http://www.puzzleweb.ru/javascript/6_operators5.php#a1)) или **[]** ([квадратные скобки](http://www.puzzleweb.ru/javascript/6_operators5.php#a2)):

var o = {};

o.x = 5;       // Добавили новое свойство

o["y"] = 10;   // Добавили новое свойство

Определение свойства происходит простым указанием его значения:

- точечная нотация:

ИмяОбъекта.ИмяСвойства

Пример:

Car.make = "Ford";

Car.model = "Mustang";

Car.year = 1969;

-квадратные скобки:

объект['свойство']

Пример:

var person = {};

person['name'] = 'Вася'; // то же что и

person.name = 'Вася';

Квадратные скобки также позволяют обратиться к свойству, имя которого хранится в переменной:

var person = {};

person.age = 25;

var key = 'age';

alert( person[key] ); // выведет 25

Если имя свойства хранится в переменной (var key = "age"), то единственный способ к нему обратиться – это квадратные скобки person[key].

Доступ через точку используется, если мы на этапе написания программы уже знаем название свойства. А если оно будет определено по ходу выполнения, например, введено посетителем и записано в переменную, то единственный выбор – квадратные скобки.

Обращение к свойству и изменение значения осуществляется точно так же (с помощью операторов доступа):

var o = {x:5};

alert(o.x);      // Обращение к свойству

alert(o["x"]);   // Обращение к свойству

o.x = 10;        // Изменяем значение

Удаление свойства осуществляется с помощью оператора [delete](http://www.puzzleweb.ru/javascript/7_delete.php):

var o = {x:5};

alert("x" in o);   // true

delete o.x;

alert("x" in o);   // false

Пример:

delete Car.year;

Остались только свойства make и model.

Метод объекта в javascript – это просто функция, которая добавлена в ассоциативный массив. Т.е. свойство, значением которого является функция, называется методом.

Чтобы создать метод, следует, прежде всего, написать функцию, которая будет его реализовывать, а затем указать его в конструкторе объекта.

function displayCar() {

var str = "<table border=\"1\">";

str += "<tr><td>Производитель:</td><td>"+this.make+"</td></tr>";

str += "<tr><td>Модель:</td><td>"+this.model+"</td></tr>";

str += "<tr><td>Год выпуска:</td><td>"+this.year+"</td></tr>";

str += "<tr><td>Владелец:</td><td>"+this.owner.name+", "+this.owner.age+"л.</td></tr>";

str += "</table>";

document.write (str);

}

К свойствам объекта функция обращается при помощи указателя this – такой синтаксис подразумевает, что данная функция будет методом объекта.

function Car(make, model, year) {

this.make = make;

this.model = model;

this.year = year;

this.displayCar = displayCar; }

Вызов метода осуществляется точно также, как и вызов обычной функции – с помощью оператора () ([оператор вызова](http://www.puzzleweb.ru/javascript/6_operators4.php#a3)):

var o = {

  sayHi: function() {

    alert("Hello!");

  }

};

o.sayHi();   // "Hello!"

Теперь, создав объект типа **Car**, можно просто воспользоваться его собственным методом для того, чтобы вывести все параметры в виде таблицы:

var MyCar = new Car("Ford", "Escort", 1997, new owner(44, Petrov));

MyCar.displayCar();

Добавление метода в объект осуществляется простым присвоением функции function(n) { ... }свойству:

var MyCar = new car("Ford", "Escort", 1997, new owner(44, Petrov));

MyCar.displayCar();

MyCar.probeg=function (x){

document.write("Пробег:"+x);

}

MyCar.probeg (234); // Пробег:234

Удаление методов^

delete MyCar. displayCar;

Таким образом, в любой объект в любое время можно добавить новый метод или удалить существующий.

Кроме того, что методы можно вызывать из объекта, методы имеют доступ к самому объекту и могут менять находящиеся в нем данные с помощью ключевого слова **this**. Значение **this** называется контекстом вызова и будет определено в момент вызова функции.

Простой вызов функции (без контекста):

function f() { console.log(this === window); // true

}

f();

В данном случае **this** внутри функции **f** равен глобальному объекту (например, в браузере это window).

Если функция вызывается через **new** как конструктор объекта, то **this** ставится на создаваемый объект, рисунок 20.1:

function Animal(name) {

    this.name = name;

}



Рисунок 20.1 – Вызов функции в качестве конструктора

Если функция запущена как свойство объекта, то в **this** будет ссылка на этот объект. **this** обозначает объект, вызвавший функцию, при этом совершенно неважно, откуда эта функция взялась. Важно лишь, какой перед функцией стоит объект.

[Пример](http://javascript.ru/tutorial/object/thiskeyword#primer):

Создадим два любых объекта:

vasya = {

    name: "Василий"

}

dima = {

    name: "Дмитрий"

}

Определим никак не связанную с ними функцию say:

say = function() {

    alert("Привет, я "+this.name)

}

Присвоим функцию свойству sayHi для обоих объектов:

vasya.sayHi = say

dima.sayHi = say

vasya.sayHi()    // =>  "Привет, я Василий"

dima.sayHi()    // => "Привет, я Дмитрий"

say();

При последнем вызове **this** ставится равным глобальному объекту.

Методы **apply** и **call** позволяют задать контекст для выполняемой функции. Разница между **apply** и **call**— только в способе передачи параметров в функцию. Первый параметр обеих функций определяет контекст выполнения функции (то, чему будет равен **this**).

Например, у нас есть функция showFullName, которая работает с this:

function showFullName() {

alert( this.firstName + " " + this.lastName );

}

var user = {

firstName: "Василий",

lastName: "Петров"

};

showFullName.call(user)

Метод **apply()** вызывает функцию с указанным значением **this** и аргументами, предоставленными в виде массива (либо [массивоподобного объекта](https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Predefined_Core_Objects#Working_with_Array-like_objects)).

В частности, эти две строчки сработают одинаково:

showFullName.call(user, 'firstName', 'surname');

showFullName.apply(user, ['firstName', 'surname']);

Преимущество **apply** перед **call** отчётливо видно, когда массив аргументов формируется динамически.

При помощи **apply** можно найти максимум в произвольном массиве, вот так:

var arr = [];

arr.push(1);

arr.push(5);

arr.push(2);

// получить максимум из элементов arr

alert( Math.max.apply(null, arr) ); // 5

**5. Порядок выполнения работы**

1. Создать объект с 5 свойствами (на выбор). Добавить метод объекта, который выводит приветственное сообщение. Вывести все свойства созданного объекта двумя способами.
2. Создайте калькулятор, который будет включать следующие методы: read() запрашивает prompt два значения и сохраняет их как свойства объекта, sum() возвращает сумму этих двух значений, mul() возвращает произведение этих двух значений.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой объект в JavaScript?
2. Перечислите основные операции, производимые с объектами.
3. Опишите способы создания пользовательских объектов в JavaScript.
4. Каким образом осуществляется доступ к свойствам объектов?
5. Что такое метод объекта?
6. Каким образом определяется контекст вызова функции?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 21**

## **Тема работы: «Работа со стандартными объектами JavaScript: объект Date, объект Math»**

**1. Цель работы**

Формирование умений работы со стандартными объектами Date и Math, применения методов объектов при решении задач на языке JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии работы с объектами Date и Math на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Для работы с датами в языке JavaScript не предусмотрено специального типа данных, однако, как и в случае с массивами, имеется специальный объект **Date.** Для создания объекта даты можно воспользоваться любым из следующих способов:

Например, можно создать объект без аргументов:

var myDate = new Date();

Если Date вызывается без аргументов, то создаётся объект с текущим временем.

alert(new Date()); /\* покажет текущее время \*/

Можно также в качестве аргумента передать **milliseconds**:

var myDate = new Date(milliseconds);

alert(new Date(3000)); /\* Thu Jan 01 1970 00:00:03 GMT+0000 \*/

Если передаётся один числовой аргумент, то он интерпретируется как количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу.

Можно передать в качестве аргумента строку с датой:

var myDate = new Date("Дата\_в\_виде\_строки");

Если указывается дата в виде строки, то она должна быть вида «Feb 28, 2005». Допустимо также задать дату, используя целочисленные значения для года, месяца, дня и т.п.:

alert(new Date('Dec 13, 1932')); /\* Tue Dec 13 1932 00:00:00 GMT+0000 \*/

Если объекту Date передаётся от 2 до 7 целых чисел, то они интерпретируются как непосредственное задание времени. Аргументы последовательно задают год, месяц, день месяца, часы, минуты, секунды и миллисекунды.

new Date(Год, Месяц, День [, Час, Минута, Секунда, Миллисекунда])

Примечания:

- отсчёт месяцев начинается с 0(январь) и до 11(декабрь), а день – от 0 до 31. Соответственно, значения времени так же ограничены: для часов это будет целое от 0 до 23, для секунд и минут – от 0 до 59, а для миллисекунд – от 0 до 999;

- год – это любое 4-значное число (если указать 2-значное, то к нему добавится 1900);

- если указаны не все аргументы, то пропущенные аргументы времени устанавливаются равными 0, а день месяца устанавливается равным 1;

- данный формат создаёт объект локального (местного) времени, то есть в том часовом поясе, который установлен операционной системой пользователя;

- если переданные параметры выходят за допустимый диапазон значений, то JavaScript автоматически преобразует их в допустимые диапазоны. Например, если передать 65 секунд, то JavaScript представит их как 1 минуту и 5 секунд.

Таким образом, чтобы в качестве значения объекта типа Date указать 9 мая 2005 года, следует написать:

var anyday = new Date (2005, 5, 9);

Объекты **Date** могут быть созданы только путем вызова функции **Date** в качестве конструктора: обычный вызов функции (то есть без использования оператора **new**) вернет строку вместо объекта Date; в отличие от других объектных типов javascript, объекты **Date** не имеют литерального синтаксиса.

Методы объекта **Date** позволяют выделять нужную часть даты (год, месяц, число, время), выводить ее в том или ином формате и т.д.

Метод **now()** возвращает количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу. Метод не имеет аргументов.

alert(Date.now()); /\* покажет количество миллисекунд \*/

Метод **UTC()** возвращает количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу до того момента времени, который передан аргументами. Этот метод принимает от 2 до 7 аргументов, которые задают определённое время: год, месяц, день месяца, часы, минуты, секунды, миллисекунды.

alert(Date.UTC(1996, 11, 20, 13, 0, 1800)); // 851088600000

Метод **parse()** преобразует переданную дату в количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу, и возвращает это значение. Метод принимает в качестве аргумента дату в строковом формате.

alert(Date.parse('Dec 13, 1932')); // -1169262000000

Методы наследников **Date** разделяются следующим образом:

- методы, начинающиеся с **«set»**, предназначены для установки даты и времени в объектах **Date**;

- методы, начинающиеся с **«get»**, предназначены для извлечения даты, времени или их частей из объектов **Date**;

- методы, начинающиеся с **«to»**, возвращают дату и время (или их части) в виде строковых значений.

В таблице 21.1 приведены способы получения отдельных значений даты.

Таблица 21.1 – Методы получения отдельных значений даты

| **Метод** | **Описание** |
| --- | --- |
| **1** | **2** |
| **getFullYear()** | Год из указанной даты по местному времени. |
| **getMonth()** | Месяц из указанной даты по местному времени. Число от 0 до 11. |
| **getDate()** | День месяца из указанной даты по местному времени. Число от 1 до 31. |
| **getDay()** | День недели из указанной даты по местному времени. Число от 0 до 6. (0 - воскресенье). |
| **getHours()** | Часы из указанной даты по местному времени. Число от 0 до 23. |
| **getMinutes()** | Минуты из указанной даты по местному времени. Число от 0 до 59. |
| **getSeconds()** | Секунды из указанной даты по местному времени. Число от 0 до 59. |
| **getMilliseconds()** | Миллисекунды из указанной даты по местному времени. Число от 0 до 999. |
| **1** | **2** |
| **getUTCFullYear()** | Год из указанной даты по всемирному времени. |
| **getUTCMonth()** | Месяц из указанной даты по всемирному времени. Число от 0 до 11. |
| **getUTCDate()** | День месяца из указанной даты по всемирному времени. Число от 1 до 31. |
| **getUTCDay()** | День недели из указанной даты по всемирному времени. Число от 0 до 6. (0 - воскресенье). |
| **getUTCHours()** | Часы из указанной даты по всемирному времени. Число от 0 до 23. |
| **getUTCMinutes()** | Минуты из указанной даты по всемирному времени. Число от 0 до 59. |
| **getUTCSeconds()** | Секунды из указанной даты по всемирному времени. Число от 0 до 59. |
| **getUTCMilliseconds()** | Миллисекунды из указанной даты по всемирному времени. Число от 0 до 999. |
| **getTime()** | Количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу до указанной даты. |
| **getTimezoneOffset()** | Смещение местного времени от всемирного (UTC+0) в минутах. Например, для часового пояса UTC+3 смещение равно -180 минут. |

В таблице 21.2 приведены способы изменения отдельных значений даты.

Таблица 21.2 – Методы изменения отдельных значений даты

| **Метод** | **Описание** |
| --- | --- |
| **1** | **2** |
| **setFullYear()** | Год из указанной даты по местному времени. Дополнительно можно указать месяц и день месяца. |
| **setMonth()** | Месяц из указанной даты по местному времени. Дополнительно можно указать день месяца. |
| **setDate()** | День месяца из указанной даты по местному времени. |
| **setHours()** | Часы из указанной даты по местному времени. Дополнительно можно указать минуты, секунды и миллисекунды. |
| **setMinutes()** | Минуты из указанной даты по местному времени. Дополнительно можно указать секунды и миллисекунды. |
| **setSeconds()** | Секунды из указанной даты по местному времени. Дополнительно можно указать миллисекунды. |
| **setMilliseconds()** | Миллисекунды из указанной даты по местному времени. |

| **1** | | **2** |
| --- | --- | --- |
| **setUTCFullYear()** | | Год из указанной даты по всемирному времени. Дополнительно можно указать месяц и день месяца. |
| **setUTCMonth()** | | Месяц из указанной даты по всемирному времени. Дополнительно можно указать день месяца. |
| **setUTCDate()** | День месяца из указанной даты по всемирному времени. | |
| **setUTCHours()** | Часы из указанной даты по всемирному времени. Дополнительно можно указать минуты, секунды и миллисекунды. | |
| **setUTCMinutes()** | Минуты из указанной даты по всемирному времени. Дополнительно можно указать секунды и миллисекунды. | |
| **setUTCSeconds()** | Секунды из указанной даты по всемирному времени. Дополнительно можно указать миллисекунды. | |
| **setUTCMilliseconds()** | Миллисекунды из указанной даты по всемирному времени. | |
| **setTime()** | Количество миллисекунд, прошедших с полуночи 1 января 1970 года по Гринвичу до указанной даты. | |

В таблице 21.3 приведены способы преобразования даты в строку.

Таблица 21.3 – Методы преобразования даты в строку

| **Метод** | **Описание** |
| --- | --- |
| **1** | **2** |
| **toString()** | Возвращает строковое представление объекта. Вызывается всегда, когда объект должен быть представлен в виде строкового значения.  **alert(new Date(123456789)); /\* Fri Jan 02 1970 13:17:36 GMT+0300 \*/** |
| **toUTCString()** | Возвращает строковое представление объекта по всемирному времени (UTC+0).  **var date = new Date('Jan 02 1970 10:00:00 GMT+0300');**  **alert(date.toUTCString()); /\* Fri, 02 Jan 1970 07:00:00 GMT \*/** |
| **toISOString()** | Возвращает строковое представление объекта по всемирному времени (UTC+0) в формате ISO 8601 (**YYYY-MM-DDTHH:mm:ss.sssZ**).  **var date = new Date('Jan 02 1970 10:00:00 GMT+0300');**  **alert(date.toISOString()); /\* 1970-01-02T07:00:00.000Z \*/** |
| **toDateString()** | Возвращает строковое представление объекта, содержащее только дату.  **var date = new Date('Jan 02 1970 10:00:00 GMT+0300');**  **alert(date.toDateString()); /\* Fri Jan 02 1970 \*/** |

| **1** | **2** |
| --- | --- |
| **toTimeString()** | Возвращает строковое представление объекта, содержащее только время.  **var date = new Date('Jan 02 1970 10:00:00 GMT+0300');**  **alert(date.toTimeString()); /\* 10:00:00 GMT+0300 \*/** |

В следующем примере осуществляется вывод на страницу текущего времени:

<HTML>

<HEAD> <TITLE> Объект Date </TITLE> </HEAD>

<BODY>

<script language="JavaScript">

var now= new Date();

document.write("Time: " + now.getHours() + ":" +

now.getMinutes() + "<br>");

document.write("Date:" +now.getDate() + "/" +

(now.getMonth() + 1) + "/" + (now.getYear()));

</BODY>

</HTML>

В примере, приведенном выше, создается экземпляр объекта Date с именем **now**, позволяющий получить доступ к текущим дате и времени.

Обратите внимание на выражение (now.getMonth()+1). Поскольку месяцы представлены числами от 0 (январь) до 11 (декабрь), для привычного отображения номера месяца к результату, возвращаемому методом getMonth(), нужно прибавить 1

Автоисправление — очень удобное свойство объектов Date. Оно заключается в том, что можно устанавливать заведомо некорректные компоненты (например 32 января), а объект сам себя поправит:

var d = new Date(2013, 0, 32); // 32 января 2013 ?!?

alert(d); // ... это 1 февраля 2013!

**Math** – встроенный объект языка JavaScript, имеющий в качестве методов и свойств основные математические константы и функции.

В отличие от других глобальных объектов, Math не является конструктором. Все свойства и методы Math статичны. А поскольку все свойства объекта Math являются предопределенными константами, то создавать другие объекты типа Math недопустимо, а обращаться к ним надо всегда одним и тем же способом.

Обращение к константе pi имеет следующий вид:

var CircleLength = diameter \* Math.PI;

В таблице 21.4 приведены свойства объекта Math.

Таблица 21.4 – Свойства объекта Math

| **Свойство** | **Описание** |
| --- | --- |
| Math.PI | Число π (Пи). Примерно равно 3.14 |
| Math.E | Число e (число Эйлера). Примерно равно 2.72 |
| Math.LN2 | Натуральный логарифм из 2. Примерно равен 0.69 |
| Math.LN10 | Натуральный логарифм из 10. Примерно равен 2.30 |
| Math.LOG2E | Логарифм e по основанию 2. Примерно равен 1.44 |
| Math.LOG10E | Логарифм e по основанию 10. Примерно равен 0.43 |
| Math.SQRT1\_2 | Квадратный корень из 1/2. Примерно равен 0.71 |
| Math.SQRT2 | Квадратный корень из 2. Примерно равен 1.41 |

В таблице 21.5 приведены методы объекта Math.

Таблица 21.5 – Методы объекта Math

| **Метод** | **Описание** |
| --- | --- |
| **1** | **2** |
| Math.random() | Случайное число в диапазоне **[0 … 1)** (от **0** включительно до **1** не включительно). |
| Math.abs(x) | Модуль числа **x** или **NaN**, если передано не число. |
| Math.ceil(x) | Округляет число **x** в большую сторону. |
| Math.floor(x) | Округляет число **x** в меньшую сторону. |
| Math.round(x) | Округляет число **x** в ближайшую сторону. |
| Math.exp(x) | Число, равное **e** в степени **x**. **e** - число Эйлера. |
| Math.pow(x, y) | Число, равное **x** в степени **y**. |
| Math.log(x) | Натуральный логарифм числа **x**. |
| Math.sqrt(x) | Квадратный корень числа. |
| Math.max(x1, …) | Наибольшее число из переданных аргументов. |
| Math.min(x1, …) | Наименьшее число из переданных аргументов. |
| Math.cos(x) | Косинус угла **x** (в радианах). |
| Math.acos(x) | Арккосинус числа **x**. Результат в радианах. |
| Math.sin(x) | Синус угла **x** (в радианах). |
| **1** | **2** |
| Math.asin(x) | Арксинус числа **x**. Результат в радианах. |
| Math.tan(x) | Тангенс угла **x** (в радианах). |
| Math.atan(x) | Арктангенс числа **x**. Результат в радианах. |
| Math.atan2(y, x) | Угол для точки с координатами **(x, y)**. Результат в радианах. |

Все тригонометрические функции значения углов принимают и возвращают в радианах. Градусы и радианы связаны следующей формулой:

угол\_в\_радианах = угол\_в\_градусах \* π / 180

Пример:

var radius = 5;

var circleArea = Math.PI \* radius \* radius;

var volume = (4/3)\*(Math.PI\*Math.pow(radius,3));

Как и в случае со свойствами, для использования методов объекта Math следует ссылаться непосредственно на сам объект, а не на его копии.

Для примера напишем функцию, которая будет возвращать диаметр исходя их площади круга:

function CircleDiam(Square) {

Radius = Math.sqrt(Square / Math.PI) ;

}

Чтобы воспользоваться ею на практике, можно прибегнуть к помощи методов prompt и alert:

var sq = prompt("Введите площадь",1);

var di = CircleDiam(sq);

alert("Диаметр окружности: "+di);

**5. Порядок выполнения работы**

1. Напишите сценарий, который бы в зависимости от времени суток выводил бы на экран окно сообщений с приветствием.
2. Напишите сценарий, который по введенной дате определяет день недели.
3. Напишите сценарий, который по дате рождения человека определяет знак его зодиака в соответствии с таблицей 21.6.

Таблица 21.6 – Значения знаков зодиаков в зависимости от даты

|  |  |
| --- | --- |
| **Название знака** | **Период** |
| Овен | 21 марта – 20 апреля |
| Телец | 21 апреля – 20 мая |
| Близнецы | 21 мая – 21 июня |
| Рак | 22 июня – 22 июля |
| Лев | 23 июля – 23 августа |
| Дева | 24 августа – 23 сентября |
| Весы | 24 сентября – 22 октября |
| Скорпион | 23 октября – 22 ноября |
| Стрелец | 23 ноября – 21 декабря |
| Козерог | 22 декабря – 20 января |
| Водолей | 21 января – 19 февраля |
| Рыбы | 20 февраля – 20 марта |

1. Напишите сценарий, который находит минимальный элемент в массиве. Массив необходимо заполнить, используя диалоговое окно «prompt».

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Каким образом осуществляется доступ к методам объекта?
2. Перечислите методы объекта Date.
3. Перечислите известные вам методы и свойства объекта Math.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 22**

## **Тема работы: «Работа с элементами объекта браузера средствами JavaScript. Обработка событий браузера»**

**1. Цель работы**

Формирование умений работы с объектной моделью браузера, навыков обработки событий браузера на языке JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии работы с объектной моделью браузера на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Объектная модель браузера (анг. Browser Object Model (BOM)) – это часть JavaScript, которая позволяет скрипту взаимодействовать с программой просмотра веб-страниц. BOM представляет объекты, через свойства и методы которых можно управлять внешним видом и поведением обозревателя.

Окно браузера представлено на рисунке 22.1:

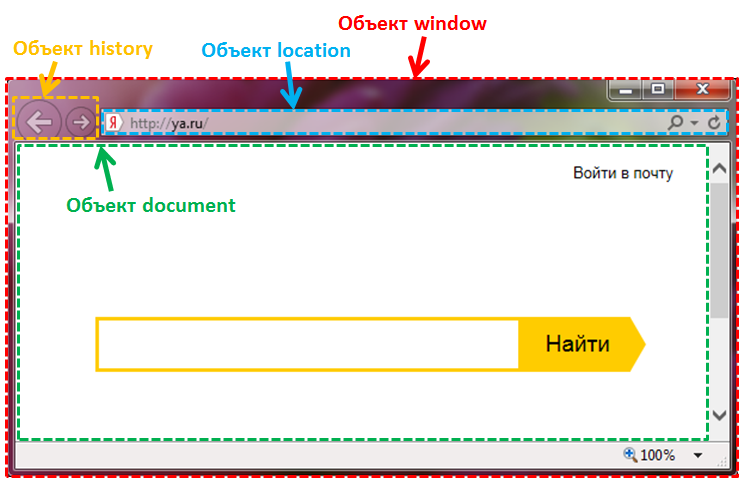


Рисунок 22.1 – Окно браузера

BOM-интерфейс позволяет работать с окном браузера как с объектом, у которого имеются свои свойства и методы. Данным объектом является объект window (он же является [глобальным объектом в JavaScript](http://webgent.ru/javascript/standard-objects/global-object)). Объект window представляет собой одно из окон или вкладку браузера с его панелями инструментов, меню, строкой состояния, HTML страницей и другими объектами.

Некоторые браузеры открывают страницы не в отдельных окнах, а во вкладках одного окна. В этом случае каждая вкладка также имеет свой собственный объект window. Однако, при обращении к свойствам, которые не могут относиться к вкладкам (например, изменяющие размер окна браузера), автоматически используется общий объект window окна браузера. Например, если внутри отдельной вкладки происходит вызов метода window.resizeTo(), тогда window ссылается на общий объект окна браузера.

Основные объекты BOM и связи между ними приведены на рисунке 22.2.

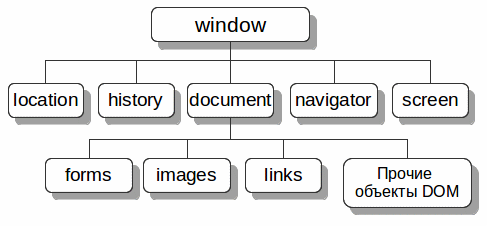
**

Рисунок 22.2 – Объекты браузера

Window – самый главный объект в браузере, который отвечает за одно из окон (вкладок) браузера. Он является корнем иерархии всех объектов доступных веб-разработчику в сценариях JavaScript.

Объект window имеет много разнообразных свойств, но используются из них лишь некоторые. Весь список свойств можно получить с помощью простого кода:

<html>

<head>

<title>BOM-интерфейс</title>

</head>

<body>

</body>

</html>

<script>

var all = '';

for (var prop in window) {

all += window.prop + '\n';

}

alert(all);

</script>

Объект window кроме глобальных объектов (document, screen, location, navigator и др.) имеет собственные свойства и методы.

К объекту Window нельзя обратиться напрямую, однако он имеет свойство window, ссылающееся на сам объект, которое можно использовать вместо ключевого слова this для ссылки на глобальный объект.

Чтобы получить объект window, достаточно написать в html документе:

<script type="text/javascript">

window //объект window

</script>

Пример создания нового окна:

var win = window.open("http://mrk-bsuir.by", "My Window);

Будет создано окно, в которое будет подгружен сайт: "http://mrk-bsuir.by", а имя нового окна будет: "My Window".

У метода **open()** существует ещё один необязательный параметр, с настройками нового окна:

var win = window.open("http://mrk-bsuir.by", "My Window", "width=400,height=500,menubar=yes,toolbar=no,location=yes,scrollbars=yes");

Этот параметр имеет очень жёсткие условия в плане синтаксиса: никаких пробелов быть вообще не должно:

- width - отвечает за ширину нового окна;

- height - отвечает за высоту нового окна;

- menubar - если стоит "yes", то в новом окне будет меню, если стоит "no", то, соответственно, меню не будет;

- toolbar - отвечает за наличие панели инструментов;

- location- отвечает за наличие адресной строки в новом окне;

- scrollbars - отвечает за наличие полос прокрутки.

Методы **focus()** и **blur()** позволяют навести фокус на окно (focus()) и, наоборот, потерять его (blur()).

Пример:

var win = window.open("http://mrk-bsuir.by", "My Window);

win.focus();

//Некая задержка

win.blur();

Метод **close()** позволяет закрыть окно:

win.close();

Данный метод не всегда сработает для глобального окна. А если и сработает, то выдаст предупреждение пользователю о закрытии окна, в котором он может либо подтвердить, либо отклонить закрытие.

Методы **back()** и **forward()** заменяют кнопки "Назад" и "Вперёд" в браузере. Иногда это используют, чтобы сайт был более удобен для пользователя (например, сделать кнопки на сайте для возврата на предыдущую страницу).

Пример применения этих методов:

window.back();

так тоже будет работать:

back();

В JavaScript имеется всего два метода для работы с таймерами. Первый метод – это **setTimeout()**. Эта функция принимает два параметра: функцию (либо код), которую нужно выполнить, и второй параметр – задержку (в миллисекундах), через которую надо выполнить функцию (либо код).

Пример:

setTimeout(func, 3000);

function func () {

alert("Прошло 3 секунды");

}

Данный скрипт через 3 секунды после начала выполнения выведет информационное сообщение.

Т.к. метод setTimeout() применён к глобальному окну, поэтому мы не писали так: "window.setTimeout()".

Если требуется остановить таймер, то используется метод **clearTimeout()**, который принимает в качестве параметра идентификатор, полученный при использовании метода setTimeout():

var id = setTimeout("3 + 4", 5000);

clearTimeout(id);

То есть сразу после запуска таймера, мы его останавливаем, соответственно, выражение: "3 + 4" посчитано не будет.

Ключевая особенность метода setTimeout() в том, что он выполняется всего один раз. В нашем примере, он вызвал функцию func() и всё. Больше эта функция вызываться через таймер не будет. Для того, чтобы настроить регулярное срабатывание таймера через равные промежутки времени используется метод **setInterval()**:

var counter = 0;

var id = setInterval(func, 5000, "Tommy");

function func(name) {

counter++;

alert(name + ", уже прошло " + (counter \* 5) + " секунд!");

if (counter == 4) clearInterval(id);

}

Глобальная переменная counter показывает, сколько раз была вызвана функция. От неё считается и прошедшее время после начала выполнения скрипта. Также заметьте, что функция func() требует параметр. Этот параметр передаётся в методе setInterval() сразу после параметра с задержкой. Если будет два параметра в функции, то в методе setInterval() также надо будет дописать ещё два передающихся параметра. Такие же правила и для метода setTimeout(). После 4-х вызовов функции мы останавливаем непрерывное выполнение таймера.

В таблице 22.1 приведены методы объекта Window.

Таблица 22.1 – Методы объекта Window

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Метод** | **Описание** | | | | |
| **1** | **2** | | | | |
| **методы ввода-вывода информации** | | | | | |
| alert() | Выводит модальное диалоговое окно с сообщением и кнопкой OK. | | | | |
| confirm() | Отображает модальное диалоговое окно, содержащее сообщение и кнопки ОК и Отмена.Возвращает соответственно true или false. | | | | |
| prompt() | Отображает диалоговое окно с сообщением и полем ввода для пользователя. Возвращает строку, содержащую введенные данные. | | | | |
|  | function alert(){  alert("hello");} | var a=confirm("This is a cat?");  alert(a); | | var a=prompt("подсказка", "default");  alert(a); | |
| **открытие — закрытие — печать окон** | | | | | |
| open() | Создает и открывает новое окно. Его параметры: адрес страницы, которую надо открыть; внутренне имя окна, которое хотите ему присвоить; набор свойств открываемого окна (location, width, height, menubar, toolbar, scrollbars). | | | | |
| **1** | **2** | | | | |
| close() | Закрывает окно, которое было открыто с помощью метода window.open(). При закрытии выскакивает предупреждение браузера из соображений безопасности. Без параметров. | | | | |
| print() | Печатает содержимое текущего окна. Без параметров. | | | | |
| focus() | Устанавливает фокус на текущее окно. Без параметров. | | | | |
| blur () | Убирает фокус у окна, уводит окна браузера на задний план, под все открытые окна. Без параметров**.** | | | | |
|  | open("http://phpmove.ru"); //можно с одним параметром  open("http://phpmove.ru","site","width=150"); //можно с 3 параметрами | | | | close(); |
| **методы для управления открытым окном — положение и размеры окна** | | | | | |
| moveBy() | Перемещает текущее окно на заданную величину относительно левого верхнего угла. | | | | |
| moveTo() | Перемещает окно в соответствии с указанными координатами. | | | | |
|  | moveTo(10,20);  mobeBy(10,20); //смещение относительно текущего левого верхнего угла | | | | |
| resizeBy() | Изменяет текущее окно на указанное число пикселей по горизонтали и вертикали. | | | | |
| resizeTo() | Изменяет размер окна до указанной ширины и высоты. | | | | |
|  | resizeTo(500,500); //задает размер окна  resizeBy(10,20); //увеличивает размер окна | | | | |
| scrollBy() | Прокрутка документа в окне на указанное количество пикселей относительно текущего положения | | | | |
| scrollTo() | Прокрутка документа до указанных координат: 1-й параметр — прокрутка слева направо, 2 — сверху-вниз | | | | |
|  | scrollTo(10,20); //просто прокрутка  scrollBy(11,12); //относительно текущего положения | | | | |
| atob() | Декодирует строку данных, которая была закодирована с использованием кодировки base-64. | | | | |
| btoa() | Создает строку ASCII в кодировке base-64 из строки бинарных данных. | | | | |
| clearInterval() | Отменяет выполнение действий, заданных с помощью метода setInterval(). | | | | |
|  | // через 5 сек остановить повторы  setTimeout(function() {  clearInterval(timerId);  alert( 'стоп' );  }, 5000); | | | | |
| clearTimeout() | Отменяет выполнение действий, заданных с помощью метода setTimeout(). | | | | |
| **1** | **2** | | | | |
|  | var timerId = setTimeout(...);  clearTimeout(timerId); | | | | |
| setInterval() | Вызывает функцию или вычисляет выражение в определенные промежутки времени (в миллисекундах). | | | | |
|  | // начать повторы с интервалом 2 сек  var timerId = setInterval(function() {  alert( "тик" );  }, 2000); | | | | |
| setTimeout() | Вызывает функцию или вычисляет выражение после указанного числа миллисекунд. | | | | |
|  | setTimeout("alert('Привет')", 1000);  // 1000 миллисекунд равны 1 секунде  // выведет сообщение через 1 с. | | function func(phrase, who) {  alert( phrase + ', ' + who );  }  setTimeout(func, 1000, "Привет", "Вася"); // Привет, Вася | | |
| stop() | Останавливает загрузку окна. | | | | |

В таблице 22.2 приведены своства объекта Window.

Таблица 22.2 – Свойства объекта Window

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| **1** | **2** |
| Closed | Возвращает логическое значение, указывающее закрыто окно или открыто. |
| document | Возвращает объект Document текущего окна. |
| Frames | Возвращает массив со всеми элементами <iframe> текущего окна. |
| History | Возвращает ссылку на объект History. |
| innerHeight | Возвращает высоту области просмотра окна. |
| innerWidth | Возвращает ширину области просмотра окна. |
| Length | Возвращает количество <iframe> элементов в окне. |
| location | Возвращает ссылку на объект Location. |
| Name | Задает или получает значение, указывающее имя окна. |
| navigator | Возвращает ссылку на объект Navigator. |
| Opener | Задает или получает ссылку на окно, которое было создано в текущем окне. |
| outerHeight | Возвращает внешнюю высоту окна, включая панели инструментов и полосы прокрутки. |
| outerWidth | Возвращает внешнюю ширину окна, включая панели инструментов и полосы прокрутки. |
| **1** | **2** |
| pageXOffset | Возвращает количество пикселей, на которое текущий документ был прокручен (по горизонтали) от верхнего левого угла окна. |
| pageYOffset | Возвращает количество пикселей, на которое текущий документ был прокручен (по вертикали) от верхнего левого угла окна. |
| Parent | Возвращает родительское окно текущего окна. |
| Screen | Возвращает ссылку на объект Screen, связанный с окном. |
| screenLeft | Получает x-координату верхнего левого угла окна относительно верхнего левого угла |
| screenTop | Получает y-координату верхнего угла окна, по отношению к верхней части экрана. |
| scroll | Эквивалент свойства pageXOffset. |
| scrollY | Эквивалент свойства pageYOffset. |
| Self | Извлекает ссылку на текущее окно или фрейм. |
| status | Получает/устанавливает текст в строке состояния в нижней части браузера. |

Доступ к различным объектам Window осуществляется с помощью следующих основных объектов:

- navigator;

- history;

- location;

- screen;

- document.

Так как данные объекты являются дочерними по отношению к объекту window, то обращение к ним происходит как к свойствам объекта window.

Например, для того чтобы обратиться к объекту screen, необходимо использовать следующую конструкцию:

window.screen

Но, если мы работаем с текущим окном, то "window." можно опустить. Например, вместо window.screen можно использовать просто:

screen

Объект **navigator** – это чисто информационный объект, он дает информацию о самом браузере (название, версия, установленные компоненты). Получить объект navigator можно как одноименное свойство window:

<script type="text/javascript">

window.navigator

</script>

Используя его свойства можно узнать какой браузер (версия) и операционную систему имеет пользователь.

В таблице 22.3 представлены свойства объекта navigator.

Таблица 22.3 – Свойства объекта navigator

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| appName | Возвращает имя браузера. Свойство доступно только для чтения.  document.write("Вы используете браузер " + navigator.appName); |
| appVersion | Возвращает название браузера, его версию и платформу (ОС) на которой он установлен. Свойство доступно только для чтения.  document.write("Версия Вашего браузера " + navigator.appVersion); |
| cookieEnabled | Показывает доступно или не доступно использование cookie файлов в браузере. |
| geolocation | Получает ссылку на объект Geolocation, который сообщает географическое расположение устройства, на котором запущено приложение. |
| onLine | Возвращает значение булева типа, значение true для работы браузера в онлайн режиме, значение false для автономного режима работы. |
| platform | Возвращает тип платформы (ОС) под которую был скомпилирован данный браузер. |
| systemLanguage | Показывает, какой язык по умолчанию используется операционной системой. |
| userAgent | Возвращает дополнительную информацию о браузере, так же определяет, какой движок использует ваш браузер. |

Пример:

var browser = navigator.appName;

if (browser == "NetScape")

document.write("Тут можно выполнять скрипты для обладателей браузеров NetScape");

Объект **History** предоставляет журнал навигации за всё время работы с конкретным окном. Так как он является свойством объекта Window, то у каждой вкладки в браузере, также как и у каждого [элемента <iframe>](https://puzzleweb.ru/html/tag_iframe.php) будет свой объект Window и свой объект History.

Из соображений безопасности браузеры не позволяют определять [URL-адреса](https://puzzleweb.ru/html/3_directory_path.php#a3) страниц, которые посещал пользователь, но можно перемещаться по их списку вперёд и назад, не зная URL-адреса.

У объекта **History** есть только одно свойство – length. Оно возвращает целое число, указывающее количество страниц в истории сеанса работы с текущим окном браузера, включая текущую загруженную страницу. Это свойство можно использовать следующим образом:

alert("Количество страниц, просмотренных в текущем окне, равно " + history.length);

Данный код отправляет пользователю сообщение с информацией о количестве посещенных страниц в текущем окне браузера. Например, для загруженной в новой вкладке страницы это свойство вернет значение 1.

Метод **back()** загружает предыдущую страницу из истории сеанса (выполняет тоже действие, что и кнопка назад в браузере). Эквивалентен history.go(-1).

<!DOCTYPE html>

<html>

<body>

<button onclick="goBack()">Вернуться назад</button>

<script>

function goBack() {

history.back();

}

</script>

</body>

</html>

Метод **forward()** загружает следующую страницу в истории сеанса (выполняет тоже действие, что и кнопка вперед в браузере). Эквивалентен history.go(1).

history.forward();

Метод **go()** загружает страницу из истории посещения текущего окна или вкладки браузера. Этот метод принимает один аргумент: строку или целое число. Целое положительное число указывает на какое количество страниц вперед нужно перейти, отрицательное число указывает на сколько страниц назад нужно вернуться. С помощью строкового аргумента можно указать точный URL-адрес из истории посещения, на который нужно перейти.

// переход назад на одну страницу

history.go(-1);

// переход вперед на две страницы

history.go(.2);

Кроме этого, с появлением HTML5 History API веб-разработчику стали доступны методы для добавления и изменения записей в истории, а также событие, с помощью которого можно обрабатывать нажатие кнопок forward (вперёд) и back (назад).

Объект **Location** содержит информацию о URL-адресе текущего документа, с которым он связан.

Чтобы изменить или получить текущий URL, не обязательно использовать методы или свойства объекта, это можно также сделать обратившись напрямую к свойству location:

var someURL = window.location;

window.location = 'httр://www.puzzleweb.ru';

В таблице 22.4 представлены методы объекта Location.

Таблица 22.4 – Методы объекта Location

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| assign() | Загружает новый документ в текущее окно (вкладку) браузера. |
| reload() | Перезагружает документ с текущем URL-адресом. Может принимать один аргумент булева типа: true - перезагрузка страницы с сервера, false - перезагрузка страницы из кэша браузера. |
| replace() | Заменяет текущий документ на документ, расположенный по адресу, переданному методу в качестве аргумента. Отличие от метода assign() заключается в том, что после использования replace() текущая страница удаляется из истории посещения текущего окна, это значит что пользователь не сможет вернуться на нее нажав кнопку назад в браузере. |

В таблице 22.5 представлены свойства объекта Location.

Таблица 22.5 – Свойства объекта Location

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| hash | Задает или получает часть URL-адреса, следующую за знаком #. |
| host | Задает или возвращает имя хоста и порт URL-адреса. |
| hostname | Задает или возвращает имя хоста из URL-адреса. |
| href | Задает или возвращает полный URL-адрес документа. |
| origin | Возвращает строку, содержащую URL-адрес, включающий протокол, домен и порт. |
| pathname | Строка, содержащая имя файла или путь к текущему файлу от корневой папки. |
| port | Содержит строку с портом. |
| protocol | Задает или получает протокол. |
| search | Задает или возвращает часть URL-адреса, следующую за знаком вопроса (?), включая сам знак вопроса. |

Объект **Screen** (который также является свойством объекта Window) – один из немногих JavaScript объектов, которые практически не используются в коде. Он просто предоставляет сведения о графических параметрах клиентской системы вне окна браузера, такие как ширина и высота в пикселях. Доступность тех или иных свойств объекта Screen зависит от браузера.

В таблице 22.6 представлены свойства объекта Screen.

Таблица 22.6 – Свойства объекта Screen

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| **1** | **2** |
| availHeight | Возвращает высоту рабочей области экрана системы в пикселях, за исключением элементов пользовательского интерфейса операционной системы (например панели задач в Windows). |
| **1** | **2** |
| availWidth | Возвращает ширину рабочей области экрана системы в пикселях, за исключением элементов пользовательского интерфейса операционной системы. |
| colorDepth | Возвращает целое число: 1, 4, 8, 15, 16, 24 или 32, указывающее количество бит на пиксель для цветов. |
| height | Возвращает общую высоту экрана в пикселях. |
| pixelDepth | Возвращает целое число: 1, 4, 8, 15, 16, 24 или 32, указывающее количество бит на пиксель для цветов. |
| width | Возвращает общую ширину экрана в пикселях. |

На мобильных устройствах размеры экрана определяются немного иначе. Устройства с системой iOS всегда возвращают размеры экрана в книжной ориентации (1024 x 768). Что касается устройств на Android, они настраивают значения screen.width и screen.height правильно.

Пример:

var foo = 'String';

console.log( navigator.userAgent ); //содержит информацию о браузере.

console.log( navigator.platform ); //содержит информацию о ОС

console.log( screen.width + ' x ' +screen.height ); //разрешение вашего экрана

console.log( location.toString() ); //ваш URL в виде строки

console.log( history ); //история браузера

//window.confirm('BOM');

var bar = window.prompt();

if (bar === 'Привет') {

alert('Привет и тебе от BOM!')

} else {

alert('Пока!');

}

**5. Порядок выполнения работы**

1. Создать страницу с именем Laba22.html.
2. Организовать посещение сайта <http://mrk-bsuir.by> всеми посетителями странички Laba22.html через 3 секунды, после полной загрузки страницы. Перед переходом на страничку <http://mrk-bsuir.by> необходимо вывести модальное окно с сообщением о переходе на новую страницу.
3. После перехода на страницу <http://mrk-bsuir.by> по истечении 10 секунд страницу необходимо закрыть.
4. Создайте 2 элемента на странице (кнопки, текста или ссылки), при нажатии на которые будут создаваться 2 разных окна со следующими свойствами:

1-ое: ширина 200, высота 200, не отображение панели инструментов, не отображение панели меню, отображение строки состояния, не отображение адресной строки, не отображение кнопок браузера, отображение полос прокрутки;

2-ое: ширина 400, высота 400, отображение панели инструментов, отображение панели меню, отображение строки состояния, не отображение адресной строки, отображение кнопок браузера, не отображение полос прокрутки.

1. Выполнить задание в соответствии с вариантом.

Вариант 1. С помощью методов для управления открытым окном переместите текущее окно на указанное число пикселей относительно левого верхнего угла. Реализуйте ввод значений через input.

Вариант 2. С помощью методов для управления открытым окном измените текущее окно на указанное число пикселей по горизонтали и вертикали. Реализуйте ввод значений через input.

Вариант 3. С помощью методов для управления открытым окном переместите окно в соответствии с указанными координатами. Реализуйте ввод значений через input.

Вариант 4. С помощью методов для управления открытым окном измените размер окна до указанной ширины и высоты. Реализуйте ввод значений через input.

Вариант 5. С помощью методов для управления открытым окном создайте строку ASCII в кодировке base-64 из строки бинарных данных. Реализуйте ввод значений через input.

Вариант 6. С помощью методов для управления открытым окном декодируйте строку данных, которая была закодирована с использованием кодировки base-64. Реализуйте ввод значений через input.

Вариант 7. С помощью свойства объекта Window получает x-координату верхнего левого угла окна относительно верхнего левого угла экрана.

Вариант 8. С помощью свойства объекта Window установите текст в строке состояния в нижней части браузера. Реализуйте ввод значений через input.

Вариант 9. С помощью свойства объекта Navigator выведите название браузера, его версию и платформу (ОС) на которой он установлен.

Вариант 10. С помощью свойства объекта Navigator выведите, какой язык по умолчанию используется операционной системой.

Вариант 11. С помощью свойства объекта Navigator выведите дополнительную информацию о браузере, а так же, какой движок использует ваш браузер.

Вариант 12. С помощью свойства объекта Screen выведите высоту рабочей области экрана системы в пикселях.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой объектная модель браузера (BOM)?
2. Перечислите основные объекты BOM.
3. Опишите свойства объекта Window.
4. Укажите параметры окна, которое открывается при выполнении следующего кода:

msgWindow=window.open("1.html", "displayWindow", "width=300,height=300,location=no, status=yes,toolbar=yes,menubar=no, scrollbars=no,resizable=no")

Как можно закрыть данное окно?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 23**

## **Тема работы: «Использование объектов HTML и объектной модели документа»**

**1. Цель работы**

Формирование умений работы с объектной моделью документа на языке JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии работы с объектной моделью документа на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Объект **document** – самый сложный объект браузера, который в свою очередь дает начало объектной модели документа (DOM). Данный объект отвечает за html документ, загруженный в окно браузера. Его получают аналогично:

<script type="text/javascript">

window.document

</script>

С помощью свойств и методов данного объекта можно получить доступ к содержимому HTML-документа, а также изменить его содержимое, структуру и оформление.

Document как и другие объекты, является частью объекта window и, следовательно, он может быть доступен как window.document.

Каждая веб-страница, загружаемая в браузере, имеет свой собственный объект Document. Веб-разработчик работает не с HTML кодом html-страницы, а с объектами (DOM), которые браузер создаёт на основе этого кода. Взаимодействие веб-разработчика с html-страницей показано на рисунке 23.1.

**Веб-страница**

**Сценарий на JavaScript**

DOM

HTML

Рисунок 23.1 – Схема взаимодействия веб-разработчика с html-страницей

Исходный код веб-страницы (HTML документа) состоит из HTML тегов и текста. Теги – это базовая синтаксическая конструкция кода HTML. Большинство тегов являются парными, т.е. один из них открывающий, а другой закрывающий. Одна такая пара тегов образует элемент HTML. В документе HTML элементы вкладываются друг в друга. В итоге веб-страница (HTML документ) представляет собой множество вложенных друг в друга элементов HTML.

Согласно DOM-модели, документ является иерархией: всё содержимое страницы (элементы и текст) представляется как иерархия узлов. Каждый HTML-тег образует отдельный элемент-узел, каждый фрагмент текста – текстовый элемент, и т.п. Таким образом, DOM – это представление документа в виде дерева тегов. Это дерево образуется за счет вложенной структуры тегов плюс текстовые фрагменты страницы, каждый из которых образует отдельный узел. Это обеспечивает унифицированный способ навигации по документу, имеющего одну из наиболее употребительных структур в программировании.

Браузер, получая код HTML из интернета, сначала строит дерево узлов (объектов) в своей памяти. А уже потом на основе этого дерева "рисует" картинку этой страницы в своём окне или вкладке, как показано на рисунке 23.2.

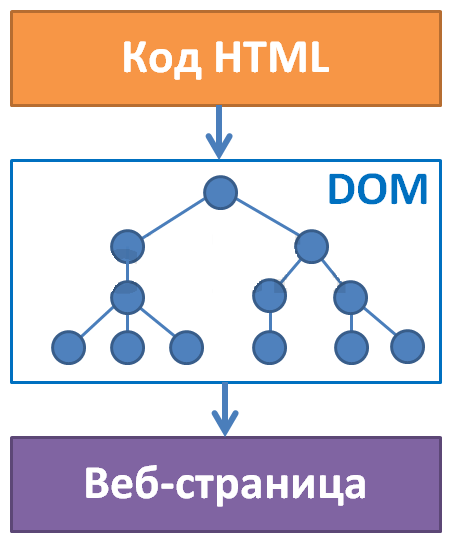


Рисунок 23.2 – Схема создания веб-страницы

Рассмотрим следующий код:

<html>

<head>

<title>Заголовок</title>

</head>

<body>

Прекрасный документ

</body>

</html>

Дерево DOM для него будет иметь следующий вид, как показано на рисунке 23.3:

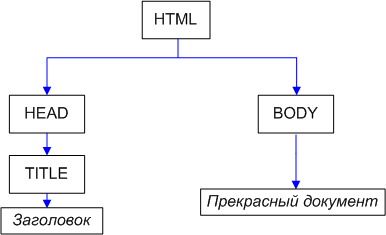


Рисунок 23.3 – DOM дерево

Дерево DOM – это множество узлов (объектов, элементов) связанных друг с другом. При этом каждый элемент HTML образует узел в этом дереве.

Теги образуют узлы-элементы (element node). Текст представлен текстовыми узлами (text node). И то и другое – равноправные узлы дерева DOM.

Пример д[ругих типов узлов](https://learn.javascript.ru/dom-nodes#другие-типы-узлов) показан на рисунке 23.4:

<!DOCTYPE HTML>

<html>

<body>

Правда о лосях

<ol>

<li>Лось — животное хитрое</li>

<!-- комментарий -->

<li>...и коварное!</li>

</ol>

</body>

</html>

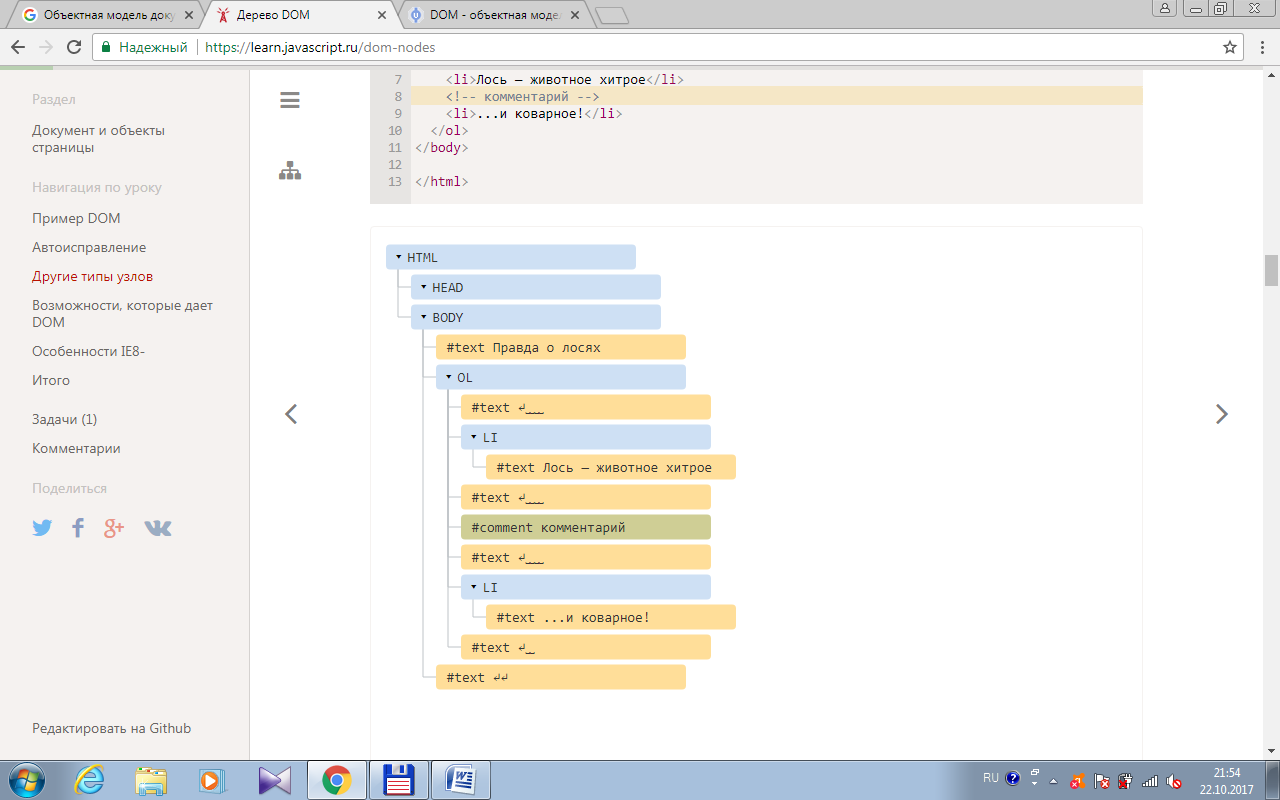
**

Рисунок 23.4 – DOM дерево с различными типами узлов

К элементу страницы нельзя обратиться раньше, чем он отобразится в браузере.

DOM-структура документа создаётся по мере вывода страницы на экран. Если элемент ещё не отображён в браузере, значит его нету в дереве DOM. Соответственно, к такому элементу нельзя обратиться.

При выполнении следующего скрипта ничего не произойдёт:

<html>

<head>

<title>DOM интерфейс</title>

<script>

document.body.style.background = '#599';

</script>

</head>

<body>

</body>

</html>

Браузер, построив на основе кода HTML дерево узлов, всегда может осуществить обратную операцию: на основе этого дерева создать код HTML.

Основная работа с HTML страницей в JavaScript состоит в том, чтобы работать с этими узлами. Как только происходит изменение свойства этих узлов в дереве, добавляются новые или удаляются существующие, то автоматически происходят изменения на экране пользователя и автоматически изменяется код HTML. Т.е. происходят обратные действия, на основе дерева строится код HTML.

Рассмотрим пример:

<div class="container">

**<h1 id="h1">Hello, world!</h1>**

<p class="yellow" id="yellow">Lorem ipsum dolor sit amet.</p>

<p class="grey" id="grey">Lorem ipsum dolor sit amet.</p>

</div>

Обратимся к заголовку и выберем его для дальнейшей работы. Заголовок H1 имеет одноименный идентификатор. JavaScript предлагает нам метод **getElementById()**, который позволяет взять из объекта один элемент с указанным идентификатором. К слову, если мы используем определенный **id**, тогда он должен быть уникален в пределах одного документа, т.е. нельзя размещать в одном документе два элемента с одинаковым атрибутом **id**.

var h1 = document.getElementById('h1');

console.log(h1);

В результате мы сохранили в переменной h1 объект, которым является конкретный элемент страницы. Теперь можно делать с данным объектом что угодно: изменить шрифт, цвет, фон и т.п. Для того, чтобы изменить фон необходимо:

var h1 = document.getElementById('h1');

h1.style.background = '#ccc';

Собственно, если элемент нам не нужен ни для каких других операций, тогда мы даже можем не сохранять его в переменную, а сразу изменить стиль элемента:

document.getElementById('h1').style.background = '#ccc';

Точно таким же образом можно обратиться к любому элементу документа, которому присвоен идентификатор.

Метод **getElementsByClassName()** позволяет получить коллекцию элементов с указанным классом.

Создадим пару абзацев с одинаковым классом и посмотрим, как эти выбранные элементы отобразятся в консоли, рисунок 23.5:

var yellow = document.getElementsByClassName('yellow');

console.log(yellow);

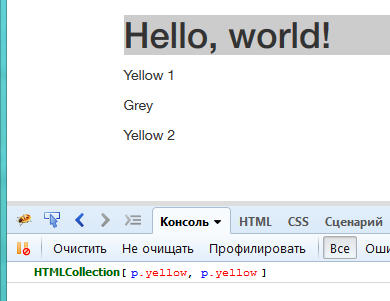


Рисунок 23.5 – Результат работы программы

Для того, чтобы окрасить в соответствующие цвета параграфы на странице, рисунок 23.6:

var yellow = document.getElementsByClassName('yellow');

yellow[0].style.background = 'yellow';

yellow[1].style.background = 'yellow';

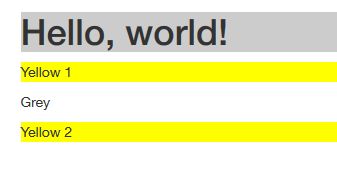


Рисунок 23.6 – Результат работы программы

Если элементов на странице много, тогда того же результата можно достичь, используя цикл:

for(var k = 0; k < yellow.length; k++){

yellow[k].style.background = 'yellow';

}

Метод **getElementsByTagName** позволяет получить коллекцию элементов по их имени, т.е. по имени тега (div, p, input и т.п.).

Вывод всех параграфов:

<div class="container" id="container1">

<h1 id="h1">Hello, world!</h1>

<p class="yellow">Yellow 1</p>

<p class="grey">Grey</p>

<p class="yellow">Yellow 2</p>

</div>

<div class="container"  id="container2">

<p class="green">Grey</p>

</div>

var p = document.getElementsByTagName('p');

console.log(p);

В результате все 4 параграфа попали в коллецию, как показано на рисунке 23.7:

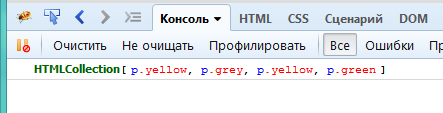


Рисунок 23.7 – Результат работы программы

Если нужны не все параграфы на странице, а только параграфы определенного блока, метод getElementsByTagName может искать элементы не только во всем документе, но и в локальном его участке:

var p = document.getElementById('container1').getElementsByTagName('p');

console.log(p);

В итоге получили только нужные элементы страницы, как показано на рисунке 23.8.

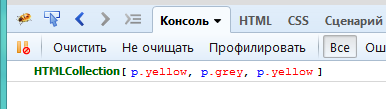


Рисунок 23.8 – Результат работы программы

Также метод getElementsByTagName в качестве параметра может применить звездочку **(\*)**, это будет означать, что необходимо получить все элементы, а не какой-то конкретный.

Пример использования (\*), рисунок 23.9:

var p = document.getElementById('container1').getElementsByTagName('\*');

console.log(p);

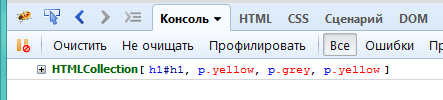


Рисунок 23.9 – Результат работы программы

Метод **querySelectorAll** возвращает все элементы по переданному селектору CSS.

В качестве примера давайте выберем все параграфы с классом yellow внутри первого блока, рисунок 23.10:

var p = document.getElementById('container1').querySelectorAll('p.yellow');

console.log(p);

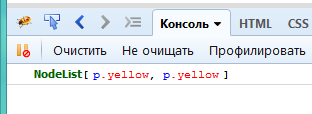


Рисунок 23.10 – Результат работы программы

Метод **querySelector** в отличие от предыдущего в том, что querySelector возвращает лишь один, первый попавшийся элемент, рисунок 23.11:

var p = document.getElementById('container1').querySelector('p.yellow');

console.log(p);

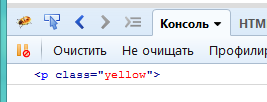


Рисунок 23.11 – Результат работы программы

Метод **closest()** ищет ближайший родительский элемент по заданному селектору.

Пример использования метода closest(), рисунок 23.12:

<div class="container" id="container1">

<div class="row" id="row">

<div class="col-md-12" id="col">

<h1 id="h1">Hello, world!</h1>

<p class="yellow">Yellow 1</p>

<p class="grey">Grey</p>

<p class="yellow">Yellow 2</p>

</div>

</div>

</div>

var elem = document.getElementById('container1').querySelector('h1');

console.log(elem.closest('div'));

console.log(elem.closest('#row'));

console.log(elem.closest('.container'));

console.log(elem.closest('h1'));

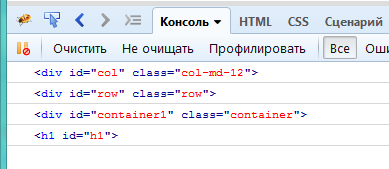


Рисунок 23.12 – Результат работы программы

Обратите внимание на последний вывод в консоль. Кроме родительского элемента, сам элемент также участвует в поиске, так что будьте аккуратны, поскольку вместо родительского элемента можно выбрать и сам элемент.

С помощью рассмотренных методов можно работать с уже готовыми элементами DOM дерева, т.е. с готовыми элементами документа.

Однако JavaScript позволяет создавать и новые элементы в документе. Сделать это можно с помощью метода **createElement**.

Метод **appendChild** позволяет добавить созданный элемент в конец.

Попробуем создать в документе какой-нибудь элемент и добавим созданный динамически элемент в body:

Пример:

var newDiv = document.createElement('div');

document.body.appendChild(newDiv);

Как видим, новый div добавлен самым последним элементом в body и разместился перед закрывающим тегом body, после подключения скриптов, рисунок 23.13.

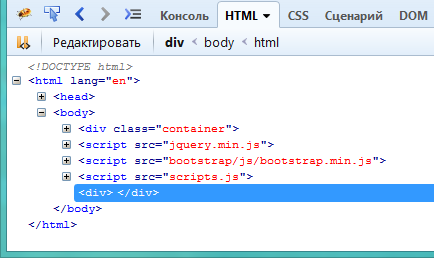


Рисунок 23.13 – Результат работы программы

Если же необходимо вставить новый элемент до подключения скриптов:

Метод **insertBefore** позволяет добавить элемент, переданный первым параметром, перед элементом, который указан вторым параметром. Результат использования метода показан на рисунке 23.14.

var newDiv = document.createElement('div');

var script = document.getElementsByTagName('script')[0];

var parent = script.parentNode;

parent.insertBefore(newDiv, script);



Рисунок 23.14 – Результат работы программы

На самом же деле родительский элемент скриптов — это body. Соответственно, можно не получать родительский блок и сразу обратиться к body:

var newDiv = document.createElement('div');

var script = document.getElementsByTagName('script')[0];

// var parent = script.parentNode;

document.body.insertBefore(newDiv, script);

Результат будет аналогичен предыдущему.

Добавить контент во вновь созданный элемент поможет свойство **innerHTML**. На рисунке 23.15 приведен результат использования свойства:

var newDiv = document.createElement('div');

newDiv.innerHTML = '<h2>Заголовок</h2><p>Lorem ipsum dolor sit amet.</p>';

var script = document.getElementsByTagName('script')[0];

document.body.insertBefore(newDiv, script);

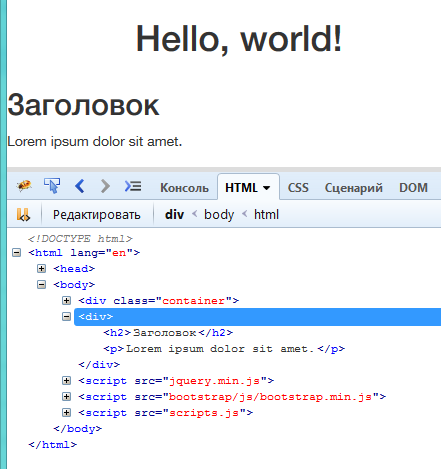


Рисунок 23.15 – Результат работы программы

Для того, чтобы добавить элементу класс, можно воспользоваться свойством **className**, рисунок 23.16:

var newDiv = document.createElement('div');

newDiv.innerHTML = '<h2>Заголовок</h2><p>Lorem ipsum dolor sit amet.</p>';

newDiv.className = 'container';

var script = document.getElementsByTagName('script')[0];

document.body.insertBefore(newDiv, script);

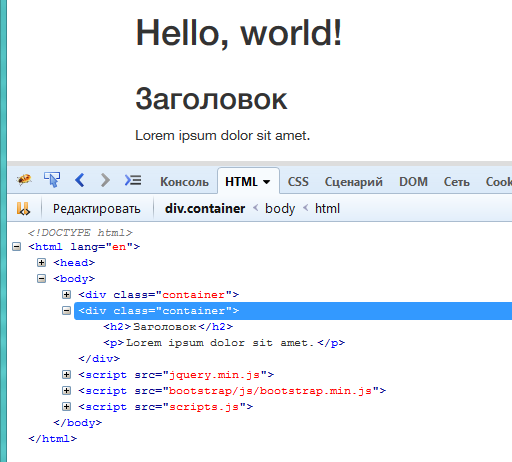


Рисунок 23.16 – Результат работы программы

Удалить элемент из DOM дерева можно с помощью метода **removeChild**.

Например, имеется следующая верстка:

<div class="container">

<h1>Hello, world!</h1>

<p>Lorem ipsum dolor sit amet.</p>

</div>

Необходимо удалить заголовок. Это можно сделать примерно так, рисунок 23.17:

var elem = document.getElementsByTagName('h1')[0];

elem.parentNode.removeChild(elem);

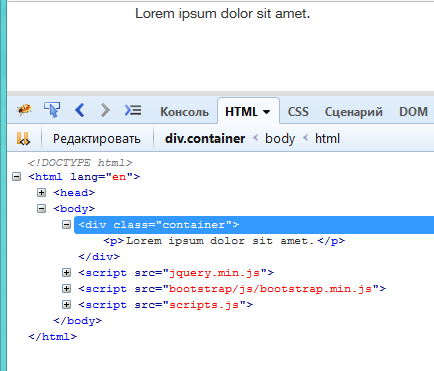


Рисунок 23.17 – Результат работы программы

Пример, получить содержимое заголовка H1, рисунок 23.18:

<div class="container" id="container1">

<div class="row" id="row">

<div class="col-md-12" id="col">

<h1 id="h1">Hello, world!</h1>

<p class="yellow">Yellow 1</p>

<p class="grey">Grey</p>

<p class="yellow">Yellow 2</p>

</div>

</div>

</div>

var elem = document.getElementById('h1');

var content = elem.innerHTML;

console.log(content);

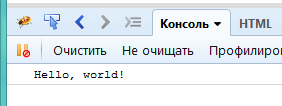


Рисунок 23.18 – Результат работы программы

Или получим содержимое всего блока, рисунок 23.19:

var elem = document.getElementById('container1');

var content = elem.innerHTML;

console.log(content);

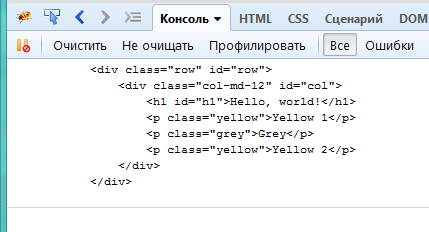


Рисунок 23.19 – Результат работы программы

Таким образом, при необходимости можно изменить полученную строку. Ну и, как отмечалось выше, свойство innerHTML позволяет записывать новое содержимое для выбранного элемента.

Пример, рисунок 23.20:

var elem = document.getElementById('container1');

var content = elem.innerHTML ='<h1>Новый заголовок</h1>\n<p>Новая строка</p>';

console.log(content);

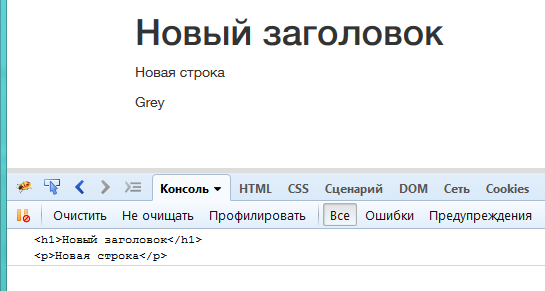


Рисунок 23.20 – Результат работы программы

Как видно на рисунке, содержимое блока полностью переписано, с помощью свойства innerHTML добавлен новый заголовок и параграф.

В отличие от innerHTML, которое получает все содержимое элемента, включая теги и контент, свойство **textContent** позволяет получить лишь контент, без тегов.

Пример (вернемся к нашей верстке), рисунок 23.21:

<div class="container" id="container1">

<div class="row" id="row">

<div class="col-md-12" id="col">

<h1 id="h1">Hello, world!</h1>

<p class="yellow">Yellow 1</p>

<p class="grey">Grey</p>

<p class="yellow">Yellow 2</p>

</div>

</div>

</div>

var elem = document.getElementById('container1');

var content = elem.textContent;

console.log(content);

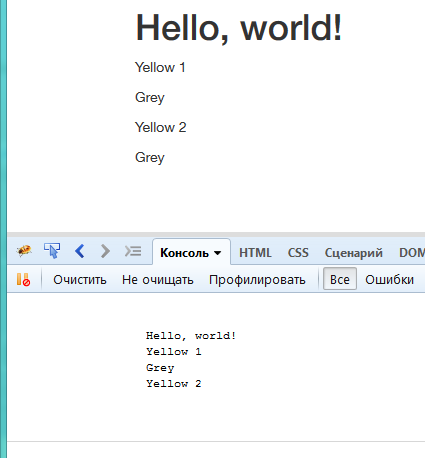


Рисунок 23.21 – Результат работы программы

Как и ожидалось, все теги были вырезаны и мы получили чистый контент блока.

Кроме того, можно перезаписать текст элемента, присвоив свойству textContent элемента новое значение, рисунок 23.22:

var elem = document.getElementById('h1');

var content = elem.textContent ='Новый заголовок';

console.log(content);

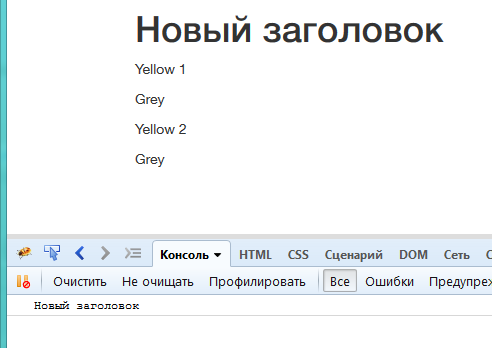


Рисунок 23.22 – Результат работы программы

Пример использования свойства bgColor (мигающий фон):

var counter =0;

setInterval(blink, 1000);

function blink() {

  if (counter % 2 == 0) document.bgColor = "red";

  else document.bgColor = "white";

  counter++;

}

**5. Порядок выполнения работы**

1. Создать страницу с именем Laba23.html. Страница должна иметь название, а также содержать заголовки 1 и 2 уровня.
2. Вывести название страницы, а также содержимое заголовков 1 и 2 из документа.
3. Создать таблицу, которая будет хранить информацию о средней температуре воздуха (по месяцам). Для наглядности, применять следующий фон ячеек (фон устанавливается скриптом, путем добавления атрибутов): t<0 – синий, t>0 – красный, t=0 – белый. Посчитать сколько месяцев в году было теплая погода (t>14), сколько месяцев были заморозки (t<0). Данную информацию вывести в виде списка под таблицей (список создается динамически).
4. Для дерева из тегов <ul>/<li> напишите код, который для каждого элемента <li> добавит информацию (непосредственно после текста) о количестве вложенных в него элементов <li> — всех, с учётом вложенных.
5. Напишите скрипт для создания элемента согласно варианту. Содержимое каждого пункта запрашивать у пользователя с помощью диалогового окна prompt. Процесс прерывается, когда пользователь нажимает отмена. Все элементы должны создаваться динамически. Задания по вариантам представлены в таблице 23.1.

Таблица 23.1 – Варианты заданий

|  |  |
| --- | --- |
| **№ варианта** | **Добавляемый элемент** |
| 0 | <li> в список <ul> |
| 1 | Абзацы в <div> |
| 2 | <li> в список <ol> |
| 3 | Ссылки в <div> |
| 4 | Блоки внутрь <div> |
| 5 | Ячейки <td> в с троки таблицы |
| 6 | Строки в таблицу |
| 7 | <input> в <div> |
| 8 | <span> в заголовок <h3> |
| 9 | Ссылки в абзац |

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой объектная модель документа (DOM)?
2. Что такое DOM-дерево?.
3. Перечислите известные Вам узлы DOM-дерева.
4. Опишите методы объекта document.
5. Опишите свойства объекта document.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 24**

## **Тема работы: «Использование разных моделей событий. Остановка всплытия и перехват событий»**

**1. Цель работы**

Формирование умений обработки разных моделей событий на языке JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии обработки событий на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Событие в JavaScript – это определённое действие, которое вызвано либо пользователем, либо браузером. К ним относятся движение мыши, нажатия на кнопки мыши и клавиатуры, наведение фокуса на элемент, изменение значения в каком-нибудь текстовом поле, изменение размеров окна браузера и так далее.

В JavaScript выделяют следующие типы событий:

- события мыши (перемещение курсора, нажатие мыши и т.д.);

- события клавиатуры (нажатие или отпускание клавиши клавиатуры);

- события жизненного цикла элементов (например, событие загрузки веб-станицы);

- события элементов форм (нажатие кнопки на форме, выбор элемента в выпадающем списке и т.д.);

- события, возникающие при изменении элементов DOM;

- события, возникающие при касании на сенсорных экранах;

- события, возникающие при возникновении ошибок.

События являются основным инструментом создания динамических страниц. Все события объединены в отдельный интерфейс событий.

Для реакции на определённые события (действия пользователя) назначаются обработчики событий.

Обработчик события – это функция, которая выполняется в тот момент, когда происходит событие, и определяет, что будет происходить при возникновении определённого события. Обработчики событий имеют следующий общий вид:

onНазваниеСобытия

То есть вначале идёт приставка "on", а дальше название события, например, такие обработчики событий: onFocus, onClick, onSubmit и так далее.

Если пользователем инициируются сразу несколько событий, то они попадают в очередь и выполняются друг за другом. Например, простое с точки зрения пользователя нажатие на кнопку клавиатуры вызывает сразу три события в такой последовательности:

- keydown - при нажатии на кнопку;

- keypress - при нажатии на кнопку;

- keyup - при отпускании кнопки.

Обработчики событий могут быть привязаны к объекту Element, Document, Window и т.д. Затем, в тот момент, когда происходит какое-либо событие, создаётся объект Event (событие), который передаётся в качестве аргумента обработчику события.

Интерфейс события объектной модели документа (DOM) доступен только через объект Event, который передаётся в качестве аргумента в обработчик события.

Следующий пример показывает, как объект Event передаётся обработчику события и может быть использован внутри него:

window.addEventListener("keydown", foo, false);

function foo(event) {

  // параметр event неявно инициализируется объектом Event

  alert(event);

}

В таблице 24.1 приведены основные события в JavaScript: название события, элементы, которые могут генерировать это событие и описание самого события.

Таблица 24.1 – Основные события в JavaScript

| **Событие** | **Объект** | **Причина возникновения** |
| --- | --- | --- |
| **1** | **2** | **3** |
| Abort | Image | Прерывание загрузки изображения |
| Blur | Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window | Потеря фокуса элемента |
| Change | FileUpload, Select, Text, Textarea | Смена значения |
| Click | Area, Button, Checkbox, Document, Link, Radio, Reset, Submit | Клик мыши на элементе |
| DblClick | Area, Document, Link | Двойной клик на элементе |

| **1** | **2** | **3** |
| --- | --- | --- |
| DragDrop | Window | Перемещение в окно браузера |
| Focus | Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window | Установка фокуса на элементе |
| KeyDown | Document, Image, Link, Textarea | Нажатие клавиши на клавиатуре |
| KeyPress | Document, Image, Link, Textarea | Удержание клавиши на клавиатуре |
| KeyUp | Document, Image, Link, Textarea | Отпускание клавиши на клавиатуре |
| Load | Document, Image, Layer, Window | Загрузка элемента |
| MouseDown | Button, Document, Link | Нажата кнопка мыши |
| MouseMove | Window | Мышь в движении |
| MouseOut | Area, Layer, Link | Мышь выходит за границы элемента |
| MouseOver | Area, Layer, Link | Мышь находится над элементом |
| MouseUp | Button, Document, Link | Отпущена кнопка мыши |
| Move | Frame | Перемещение элемента |
| Reset | Form | Сброс формы |
| Resize | Frame, Window | Изменение размеров |
| Select | Text, Textarea | Выделение текста |
| Submit | Form | Передача данных |
| Unload | Window | Выгрузка текущей страницы |

Назначить обработчик события можно несколькими способами.

1. Самый простой способ задания обработчика – это использование [атрибутов событий](http://webgent.ru/html/event-attribute) прямо в HTML-коде.

Значением атрибутов событий указывается код JavaScript или название готовой функции. Если указывается просто код JavaScript, то браузер автоматически создаёт анонимную функцию, телом которой является исходный код.

<html>

<head>

<title>События</title>

</head>

<body>

<p onmouseover="this.style.background = 'red';" onmouseout="this.style.background = 'transparent';">Наведите курсор</p>

</body>

</html>

<html>

<head>

<title>События</title>

</head>

<body>

<p onmouseover="alert('Ok');">Наведите курсор</p>

</body>

</html>

<script>

var par = document.body.firstElementChild;

alert(par.onmouseover); /\* 'function ...' (в свойстве записана функция) \*/

</script>

При использовании HTML-атрибутов для задания обработчика события разработчик ограничен стандартом HTML. Использование DOM-интерфейса расширяет возможности по созданию динамических страниц, позволяя устанавливать обработчики на любые события.

2. Ещё один способ установки обработчика события – это прямое использование DOM свойств, то есть сразу записывать обработчик в свойство элемента.

Имя свойства должно быть в нижнем регистре. В отличие от HTML-атрибутов DOM-свойства регистрозависимы.

Значением свойства обязательно должна быть функция.

<html>

<head>

<title>События</title>

</head>

<body>

<p>Наведите курсор</p>

</body>

</html>

<script>

var par = document.body.firstElementChild;

par.onmouseover = function () {

this.style.background = 'green';

};

par.onmouseout = function () {

this.style.background = 'transparent';

};

</script>

Использование HTML-атрибутов или DOM-свойств позволяет устанавливать только по одному обработчику на каждое событие.

3. Для установки и удаления любого количества обработчиков на одно событие используются методы **addEventListener()** и **removeEventListener()**.

Метод addEventListener() добавляет обработчик события.

element.addEventListener(имя\_события, обработчик[, стадия])

где имя\_события – название события в кавычках;

обработчик – функция-обработчик события (можно указать анонимную функцию или ссылку на готовую);

стадия – необязательный аргумент. Он содержит информацию, на какой [стадии](http://webgent.ru/javascript/event-stage) сработает обработчик: (true – на стадии перехвата, false (используется по умолчанию) – на стадии всплытия).

<html>

<head>

<title>События</title>

</head>

<body>

<input type="button" value="Кнопка">

</body>

</html>

<script>

var button = document.body.firstElementChild;

button.addEventListener('click', function() {alert('Обработчик 1');});

button.addEventListener('click', function() {alert('Обработчик 2');});

</script>

При нажатии на кнопку сработают оба обработчика.

Метод **removeEventListener()** используется для удаления обработчика события, назначенного с помощью метода addEventListener(). Обязательно должны быть указаны те значения, которые использовались при назначении обработчика.

element.removeEventListener(имя\_события, обработчик[, стадия])

где имя\_события – название события в кавычках;

обработчик – функция-обработчик события. Должна указываться именно ссылка, чтобы браузер понимал, какой обработчик нужно удалить. Соответственно, если обработчик был назначен напрямую (анонимной функцией), то его удалить не удастся;

стадия – необязательный аргумент. Указывается, если использовался при назначении обработчика.

<html>

<head>

<title>События</title>

</head>

<body>

<input type="button" value="Кнопка">

</body>

</html>

<script>

function handler() {alert('Обработчик 1');}

var button = document.body.firstElementChild;

/\* добавление обработчиков \*/

button.addEventListener('click', handler);

button.addEventListener('click', function() {alert('Обработчик 2');});

/\* удаление обработчиков \*/

button.removeEventListener('click', handler);

button.removeEventListener('click', function() {alert('Обработчик 2');});

</script>

Второй обработчик остался, так как он назначен анонимной функцией.

Все интернет-страницы состоят из отдельных элементов (тегов). При выводе станицы на экран браузер создаёт [DOM-дерево](http://webgent.ru/javascript/dom-tree), отражающее структуру документа (связи между родительскими и дочерними элементами).

В DOM-модели каждый дочерний элемент является частью родительского элемента. Следовательно, если на элементе возникает какое-либо событие (например, клик мышью), то оно же возникает и на его родителе, и так далее вверх по DOM-дереву вплоть до корневого элемента (объект document). Если нескольким из таких элементов будут назначены обработчики события, то все они выполнятся.

В примере ниже клик мышью по элементу <mark> вызовет обработчик onclick и на родительском элементе <div>.

<html>

<head>

<title>События</title>

</head>

<body>

<div onclick="alert('Обработчик DIV')">

<mark onclick="alert('Обработчик MARK')">MARK</mark> внутри DIV

</div>

</body>

</html>

Чтобы знать, в какой последовательности будут выполняться обработчики, необходимо иметь представление о том, как браузер обрабатывает событие.

Само исходное событие для всех обработчиков является общим. Его обработка делится на три стадии:

- стадия перехвата (capturing stage);

- стадия цели (target stage);

- стадия всплытия (bubbling stage).

Элемент, на котором возникает событие, называется целевым.

При возникновении события браузер поочерёдно проходит от верхнего элемента DOM-дерева (document) вниз через все промежуточные элементы к целевому. Эта стадия обработки называется стадией перехвата. Обработчики запускаются до того, как событие дойдёт до целевого элемента. Событие как бы перехватывается. Отсюда и следует такое название стадии.

Обработчики события выполняются только в том случае, если для них задано выполнение на стадии перехвата. Для этого необходимо использовать значение true для третьего атрибута метода [addEventListener()](http://webgent.ru/javascript/event-handler#addEventListener). Это единственный способ использовать обработчик на стадии перехвата.

Стадия перехвата используется очень редко, но иногда может пригодиться.

После того, как событие опустилось до целевого элемента, стадия перехвата завершается и выполняются обработчики целевого элемента. Это вторая стадия – стадия цели. Очерёдность выполнения обработчиков на целевом элементе зависит только от очерёдности их назначения. Использование третьего аргумента метода addEventListener() никак не повлияет на порядок их запуска.

Далее начинается последняя стадия. Событие проходит поочерёдно от целевого элемента через цепочку родителей до корневого элемента документа. Событие продвигается вверх по DOM-дереву. Оно будто всплывает. Отсюда и название – стадия всплытия.

На данной стадии выполняются все остальные обработчики. Обработчики, назначенные с помощью атрибутов событий и DOM-свойств всегда выполняются на стадии всплытия. В методе addEventListener() для использования стадии всплытия можно просто опустить третий аргумент.

Прохождение всех стадий продемонстрировано на примере ниже (запускается кликом по жёлтому элементу):

<html>

<head>

<title>События</title>

<style>

div {

width: 60px;

height: 100px;

position: relative;

background: red;

}

aside {

width: 80px;

height: 60px;

background: yellow;

position: absolute;

top: 20px;

left: 70px;

}

</style>

</head>

<body>

<div>DIV

<aside>ASIDE</aside>

</div>

</body>

</html>

<script>

var div = document.getElementsByTagName('div')[0];

var aside = document.getElementsByTagName('aside')[0];

div.addEventListener('click', function() {alert('DIV - перехват')}, true);

div.addEventListener('click', function() {alert('DIV - всплытие')});

aside.addEventListener('click', function() {alert('ASIDE - всплытие')});

aside.addEventListener('click', function() {alert('ASIDE - перехват')}, true);

</script>

// div-перехват, aside-всплытие, aside-перехват, div-всплытие

Обработчики элемента <aside> запускаются в порядке назначения.

В этом примере наглядно видно, что событие всегда всплывает по дереву DOM, несмотря на то, что визуально событие не произошло над элементом <div>.

Помимо выполнения назначенных обработчиков, события могут вызывать действия браузера по умолчанию. Эти действия как бы подразумеваются сами собой в зависимости от элемента страницы, таблица 24.2.

Таблица 24.2 – Основные события по умолчанию

| **Событие** | **Действие браузера** |
| --- | --- |
| **1** | **2** |
| **Click** | Клик по ссылке вызывает переход на новую страницу. |
| **Click** | Клик по неактивному полю для ввода текста делает его активным. |
| **contextmenu** | Клик правой кнопкой мыши открывает контекстное меню. |
| **Dblclick** | Двойной клик на тексте выделяет его. |
| **mousedown** | Нажатие левой кнопки мыши и удержание её над текстом начинает его выделение. |
| **mousewheel** | Движение колёсика мыши вызывает прокрутку страницы. |
| **Keydown** | Нажатия на кнопки клавиатуры внутри текстового поля приводят к набору текста. |
| **Keydown** | Нажатие на кнопку **Enter** в активном поле вызывает отправку формы на сервер. |

Обычно, если на конкретное событие устанавливается обработчик, то действия браузера по умолчанию не нужны. Тогда их просто можно отменить. Отменить действия браузера по умолчанию можно простым вызовом return false в конце обработчика. Но это можно использовать только внутри обработчиков, установленных через HTML-атрибут или DOM-свойство.

<html>

<head>

<title>События</title>

</head>

<body>

<label>Попробуйте поставить галочку <input type="checkbox" onclick="return false;"></label>

</body>

</html>

При помощи мыши поставить галочку не получится.

Если обработчик устанавливается с помощью метода [addEventListener()](http://webgent.ru/javascript/event-handler#addEventListener), тогда вызов return false ни к чему не приведёт. Универсальным способом для отмены действий браузера по умолчанию является встроенный метод **event.preventDefault()**.

Вызов preventDefault() во время любой стадии обработки события отменяет действия браузера. Действия назначенных обработчиков данный метод не отменяет.

<html>

<head>

<title>События</title>

</head>

<body>

<input type="text" placeholder="Введите текст">

</body>

</html>

<script>

document.body.firstElementChild.onkeypress = function(event) {

event.preventDefault();

alert('Отменено');

}

</script>

В получившемся поле нельзя ввести текст, используя клавиатуру.

Действия браузера можно отменить не для всех событий. Вызов preventDefault() на неотменяемом событии результата не даст.

Для остановки выполнения события на текущем элементе используется метод **event.stopPropagation()**. Данный метод отменяет выполнение обработчиков тех элементов, до которых событие не дошло. Однако, все обработчики текущего элемента будут выполнены даже после вызова stopPropagation().

Метод stopPropagation() не отменяет действия браузера по умолчанию.

<html>

<head>

<title>События</title>

</head>

<body>

<form name="test">

<input type="text" placeholder="Введите текст" onkeypress="alert('input')">

</form>

</body>

</html>

<script>

document.forms.test.addEventListener('keypress', function(event) {

alert('form 1');

event.stopPropagation();

}, true);

document.forms.test.addEventListener('keypress', function(event) {

alert('form 2');

}, true);

</script>

// form1, form2

Первый обработчик элемента <form> на стадии перехвата отменяет дальнейшее распространение события keypress. При этом второй обработчик элемента <form> всё равно выполняется. А также выполняется действие браузера по умолчанию.

Чтобы отменить выполнение всех невыполненных обработчиков события, даже назначенных текущему элементу, используется метод **event.stopImmediatePropagation()**.

Если одному элементу на одно событие назначено несколько обработчиков, то они выполняются в порядке назначения. Если один из них вызывает метод stopImmediatePropagation(), тогда оставшиеся обработчики не выполняются. Дальнейшее распространение события не происходит.

Метод stopImmediatePropagation() не отменяет действия браузера по умолчанию.

<html>

<head>

<title>События</title>

</head>

<body>

<form name="test">

<input type="text" placeholder="Введите текст" onkeypress="alert('input')">

</form>

</body>

</html>

<script>

document.forms.test.addEventListener('keypress', function(event) {

alert('form 1');

event.stopImmediatePropagation();

}, true);

document.forms.test.addEventListener('keypress', function(event) {

alert('form 2');

}, true);

</script>

Первый обработчик элемента <form> на стадии перехвата отменяет выполнение последующих обработчиков события keypress, назначенных текущему элементу, а также предотвращает дальнейшее распространение события. Второй обработчик элемента <form> не выполняется. Действие браузера по умолчанию всё равно выполняется.

Пример:

<html>

<head>

<style>

  a {

    color: blue;

    text-decoration: underline;

    cursor: pointer;

  }

</style>

<script language = 'javascript'>

var default\_color = "white";

function setTempColor(color) {

  document.bgColor = color;

}

function setDefaultColor(color) {

  default\_color = color;

  document.bgColor = default\_color;

}

function defaultColor() {

  document.bgColor = default\_color;

}

</script>

</head>

<body>

  <a onMouseOver = "setTempColor('white');" onMouseOut = "defaultColor()" onClick = "setDefaultColor('white');">Белый</a>

  <a onMouseOver = "setTempColor('yellow');" onMouseOut = "defaultColor()" onClick = "setDefaultColor('yellow');">Жёлтый</a>

  <a onMouseOver = "setTempColor('green');" onMouseOut = "defaultColor()" onClick = "setDefaultColor('green');">Зелёный</a>

</body>

</html>

На странице имеются три ссылки. Каждая из ссылок отвечает за разный цвет фона (допустим, белый, жёлтый и зелёный). Вначале фон белый. При наведении мыши на определённую ссылку цвет фона меняется. При отведении мыши цвет фона возвращается на цвет по умолчанию. При щелчке мыши по ссылке цвет фона сохраняется, как по умолчанию.

При возникновении какого-либо события на элементе страницы браузер автоматически создаёт объект, в котором содержится полезная информация о наступившем событии (например, свойство type указывает тип события).

Объект любого события является наследником общего объекта-конструктора Event. Это означает, что все объекты обладают общими свойствами и методами, которые наследуются от Event, таблица 24.3. Таких свойств немного и используются из них лишь некоторые.

Таблица 24.3 – Свойства наследников объекта Event

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| **1** | **2** |
| Bubbles | Возвращает логическое значение, которое указывает, является ли событие всплывающим. |
| Cancelable | Возвращает логическое значение, указывающее, является ли событие отменяемым. |
| currentTarget | Возвращает целевой объект события, обрабатываемого в настоящее время. |
| defaultPrevented | Получает значение, указывающее, следует ли отменить действи по умолчанию. True – действия по умолчанию должны быть отменены, false – действия по умолчанию разрешаются. |
| eventPhase | Указывает, какая фаза события, в настоящее время проверяется. |
| Target | Ссылается на элемент, который является целевым объектом данного события. |
| timeStamp | Получает время в миллисекундах, когда произошло событие. |
| Type | Имя события (без учета регистра). |
| isTrusted | Получает значение, указывающее, было ли событие инициировано в браузере (события браузера и пользовательские события) или в сценарии. |

<html>

<head>

<title>События</title>

</head>

<body>

<input type="button" value="Кнопка">

</body>

</html>

<script>

var button = document.body.firstElementChild;

button.onclick = function(event) {

alert(event.type); /\* 'click' \*/

}

</script>

Чтобы в любой момент выполнения события обратиться к целевому элементу, используется свойство **event.target**. Оно всегда содержит ссылку на тот элемент, на котором было вызвано событие.

Значение target отличается от **currentTarget**, если обработчик выполняется на стадии перехвата или всплытия.

Свойство target широко используется. Когда имеется несколько однотипных элементов (например, элементы меню), устанавливается только один обработчик на родительский элемент. А с помощью свойства target определяется элемент, для которого должно выполниться событие.

<html>

<head>

<title>События</title>

</head>

<body>

<ul>

<li>lorem</li>

<li>ipsum</li>

<li>dolor</li>

<li>sit</li>

</ul>

</body>

</html>

<script>

var uList = document.body.firstElementChild;

uList.onclick = function (event) {

this.removeChild(event.target);

}

</script>

В данном примере клик на любом элементе списка удаляет его. При этом скрипт использует всего один обработчик. Особенно это полезно, когда список элементов заведомо не известен, а создаётся динамически.

Свойство **event.eventPhase** указывает, какая [фаза события](http://webgent.ru/javascript/event-stage) проходит в настоящее время. Оно имеет числовое значение, которое соответствует стадии обработки:

- 0 – никакое событие не обрабатывается;

- 1 – стадия перехвата;

- 2 – стадия цели;

- 3 – стадия всплытия.

Точное время, когда произошло событие, хранится в свойстве **event.timeStamp**.

Время представлено как количество миллисекунд, прошедших с начала эпохи Unix (1 января 1970 года по Гринвичу).

В свойстве **event.type** содержится название события (click, keypress и т.п.).

В свойстве **event.isTrusted** содержится информация о том, кем вызвано событие. Свойство имеет булевый тип:

- true – событие вызвано действием пользователя или браузером;

- false – событие вызвано скриптом, например, с помощью метода [dispatchEvent](http://webgent.ru/javascript/event-dispatch#dispatchEvent).

Информация о том, будет ли событие всплывать, хранится в свойстве **event.bubbles**. Свойство имеет булевый тип:

- true – событие будет всплывать;

- false – событие не будет всплывать.

Свойство **event.cancelable** содержит информацию о том, можно ли отменить стандартные действия браузера, выполняемые по умолчанию. Свойство имеет булевый тип:

- true – действия браузера можно отменить;

- false – действия браузера отменить нельзя.

Свойство **event.defaultPrevented** содержит информацию о том, было ли отменено стандартное действие браузера, выполняемое по умолчанию. Свойство имеет булевый тип:

- true – действие браузера было отменено;

- false – действие браузера не было отменено.

События на странице могут запускаться не только браузером или посетителем сайта, но и самим скриптом. Для этого предварительно необходимо создать объект события, который будет передаваться обработчику. Делается это с помощью объекта-конструктора Event.

New Event(имя\_события[, свойства])

где имя\_события – указывается название события в кавычках. Можно использовать не только стандартные события, но и придумывать свои собственные (тогда обработчик назначается с помощью метода addEventListener());

свойства – необязательный аргумент. Должен быть объектом с двумя свойствами:

- bubbles – всплывать ли событию (true/false);

- cancelable – возможность отмены действий браузера по умолчанию (true/false).

Данные свойства будут унаследованы объектом события.

По умолчанию: {bubbles: false, cancelable: false}.

Чтобы сгенерировать событие на элементе, используется метод **dispatchEvent()**.

element.dispatchEvent(объект\_события)

где объект\_события – ссылка на специально созданный объект события. Данный объект будет передан обработчикам события.

Вызов dispatchEvent() приводит к выполнению обработчиков события в соответствующем порядке. Правила обработки события браузером (погружение и всплытие) сохраняются. Но для всплытия события необходимо указать свойство bubbles: true при создании объекта события.

<html>

<head>

<title>События</title>

</head>

<body onclick="alert('Событие click всплывает.');">

<p>Параграф</p>

</body>

</html>

<script>

var par = document.body.firstElementChild;

var event = new Event('click', {bubbles: true});

par.dispatchEvent(event); /\* вызов 'click' на параграфе \*/

</script>

В примере вызывается событие click на параграфе и оно всплывает.

Метод dispatchEvent() возвращает значение false, если хотя бы один из выполняемых обработчиков вызывает event.preventDefault() (при создании объекта события должно быть указано свойство cancelable: true). В остальных случаях возвращается true.

<html>

<head>

<title>События</title>

</head>

<body>

<input type="button" onclick="event.preventDefault();" value="Кнопка">

</body>

</html>

<script>

var button = document.body.firstElementChild;

var event\_1 = new Event('click', {cancelable: true});

alert(button.dispatchEvent(event\_1)); /\* false, действие отменено \*/

var event\_2 = new Event('click', {cancelable: false});

alert(button.dispatchEvent(event\_2)); /\* true, действие не отменено \*/

</script>

Если вызывается стандартное событие (click и т.п.), тогда данная возможность позволяет узнать, было ли отменено действие браузера по умолчанию.

В случае с нестандартными событиями вызов event.preventDefault() позволяет получить дополнительную информацию от обработчика события.

Пример:

элементу <input> назначен обработчик нестандартного события check\_value для проверки введённого значения:

<html>

<head>

<title>События</title>

</head>

<body>

<input type="text" oninput="mess(); this.value='';" placeholder="Введите цифру 5">

</body>

</html>

<script>

var input = document.body.firstElementChild;

input.addEventListener('check\_value', function(event) {

if (this.value !== '5') {

event.preventDefault();

}

});

function mess() {

var check\_event = new Event('check\_value', {cancelable: true});

if (input.dispatchEvent(check\_event) == true) {

alert('Поздравляем. Вы ввели цифру 5.');

} else {

alert('Введите цифру 5.');

}

}

</script>

При вводе любого символа в текстовое поле запускается функция mess(). В ней генерируется событие check\_value для проверки значения. В зависимости от результата выполнения обработчика события check\_value выводится нужное сообщение.

Генерация событий в скрипте используется редко. Это вполне логично, ведь обработчики событий должны реагировать на действия, совершённые пользователем или браузером. А для выполнения команд, не связанных с этими действиями, можно использовать простые функции. Однако, есть случаи, когда генерация событий может пригодиться.

Можно выделить два основных случая для генерации событий в скрипте:

1) генерация стандартных событий с целью проверки работоспособности кода. Например, когда необходимо проверить реакцию скрипта на многократные действия пользователя. Или когда запуск программы осуществляется сложной комбинацией событий, а тестирование надо провести много раз;

2) генерация собственных (нестандартных) событий для улучшения прозрачности кода. Например, изменение на странице сайта (раскрытие подменю, добавление товара в корзину и т.п.), выполненное скриптом, может восприниматься и как определённое событие. Запуск события (а не функции) в таком случае более понятен, особенно если такое событие в свою очередь приводит к выполнению определённых действий (затемнение фона при раскрытии подменю, анимация добавления в корзину и т.п.).

**5. Порядок выполнения работы**

1. Выполнить изменение рисунков при наведении мыши. Для этого необходимо подготовить 2 рисунка, например, 1.png, 2.png. Наберите следующий код: <img src=‖1.png‖ name=‖ris1‖ onMouseOver=‖ris1.src=’2.png’‖ onMouseOut=‖ris1.src=’1.png’‖>.
2. Подготовьте рисунок. Разделите его на 6 равных фрагментов. Разделенные фрагменты должны появиться в произвольном месте страницы. Реализовать возможность перетаскивания фрагментов на странице, а именно – при щелчке мышкой по фрагменту, он должен «приклеиться» к курсору, при повторном щелчке «отклеиться». Добавить кнопку проверки правильности сбора рисунка.
3. Выполнить задание в соответствии с вариантом

Вариант 1. Напишите сценарий, с помощью которого циклически изменяется изображения (т.е. поочередно сменяют друг друга, минимально 5 изображений).

Вариант 2. Напишите сценарий, в котором при наведении на изображение, оно бы увеличивалось постепенно, а при покидании – постепенно возвращалось к своему первоначальному размеру.

Вариант 3. Создайте горизонтальное меню из 3-х элементов (изображений), при нажатии на каждый элемент появляется соответствующий слой с текстом (нажали на первый элемент – появится первый слой и т.д.).

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой объект Event?
2. Перечислите типы событий.
3. Какие элементы могут генерировать события?
4. Опишите способы назначения обработчиков событий.
5. На какие стадии делится обработка события? В чем их суть?
6. Перечислите методы объекта Event.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 25**

## **Тема работы: «Проверка корректности информации, введенной пользователем. Обработка данных формы»**

**1. Цель работы**

Формирование умений обработки данных форм на языке JavaScript, навыков проверки корректности информации, введенной пользователем.

**2. Задание**

Создать HTML-документ, содержащий сценарии обработки данных форм на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

При работе с формами в JavaScript используется объект **Form**, который также является очередным свойством объекта Document. **forms** содержит коллекцию всех форм на веб-странице.

Доступ к форме осуществляется по её имени или по индексу.

document.forms.my – форма с именем 'my'

document.forms[0] – п ервая форма в документе

Общий вид получения доступа к полю формы по ее имени следующий:

document.имя\_формы.имя\_поля.value;

То есть, сначала обращаемся к объекту Document, затем к его свойству Form (через имя формы), потом к имени поля данной формы, и, наконец, к значению поля.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search"></form>

</body>

</html>

<script>

var form = document.forms.search; // или

var form = document.forms['search']; // или

var form = document.forms[0];

</script>

Доступ к элементу формы осуществляется аналогично свойству document.forms. Каждая форма имеет свойство **elements**, в котором содержится список всех элементов формы, как показано на рисунке 25.1.

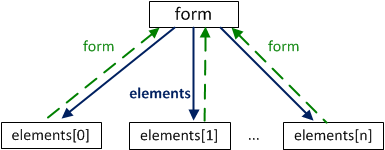


Рисунок 25.1 – Свойство elements объекта form

Пример:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search">

<input name="query">

</form>

</body>

</html>

<script>

var form = document.forms.search;

var input = form.elements.query; // или

var input = form.elements['query']; // или

var input = form.elements[0];

</script>

Если в форме используется несколько элементов с одинаковым именем, то обращение к элементу по имени вернёт список всех элементов с этим именем.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search">

<input type="radio" name="query" value="1">

<input type="radio" name="query" value="2">

</form>

</body>

</html>

<script>

var form = document.forms.search;

var input = form.elements.query;

alert(input[0].value); /\* 1 \*/

alert(input[1].value); /\* 2 \*/

</script>

Свойство elements также есть у элементов **<fieldset>**. Вот пример:

<body>

<form>

<fieldset name="set">

<legend>fieldset</legend>

<input name="text" type="text">

</fieldset>

</form>

<script>

var form = document.forms[0];

alert( form.elements.text ); // INPUT

alert( form.elements.set.elements.text ); // INPUT

</script>

</body>

По элементу можно получить его форму, используя свойство **element.form**:

<body>

<form>

<input type="text" name="surname">

</form>

<script>

var form = document.forms[0];

var elem = form.elements.surname;

alert (form.elements.surname); // [object HTMLInputElement]

alert(elem.form == form); // true

</script>

</body>

Свойство **value** содержит текущее введённое пользователем значение. До начала ввода пользователем нового значения данное свойство определяется атрибутом **value**, если он указан:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="user\_form" onsubmit="showValue(); return false" autocomplete="off">

<input type="text" placeholder="some text" name="user\_input">

<input type="submit" value="Показать VALUE">

</form>

</body>

</html>

<script>

var form = document.forms.user\_form;

var input = form.elements.user\_input;

function showValue() {

alert(input.value); /\* всегда выводит текущее значение \*/

}

</script>

Если ничего не вводить в текстовое поле, то при отправке формы выведется пустая строка, так как атрибут value не задан.

У элементов с типом **checkbox** и **radio** свойство value не может изменяться пользователем. При их использовании необходимо знать, выбраны ли они пользователем. Для этого используется логическое свойство **checked**. Оно отражает текущее состояние элемента (отмечен или не отмечен) и позволяет его изменить:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search" autocomplete="off">

<input type="radio" name="query" value="1" checked>

<input type="radio" name="query" value="2" >

</form>

</body>

</html>

<script>

var form = document.forms.search;

var input = form.elements.query;

if (input[0].checked) {

var a=input[0].value

alert(a);}

else {

alert(input[1].value); }

Поле для отправки файла (тип file) имеет свойство **files**. Это объект, содержащий список всех вставленных файлов.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="upload" onsubmit="showFile(); return false">

<input type="file" name="user\_file">

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

</form>

</body>

</html>

<script>

function showFile() {

var form = document.forms.upload;

var files = form.elements.user\_file.files;

alert(files[0].name); /\* покажет имя файла \*/

alert(files[0].lastModifiedDate); /\* покажет время последнего изменения файла \*/

}

</script>

С помощью методов **focus()** и **blur()** поле ввода можно сделать активным или неактивным. Например, можно автоматически сделать активным то поле, в котором допущена ошибка, и т.п.:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search">

<label>Введите что-нибудь: <input name="text\_in" oninput="show()"></label>

<br>

<label>Вы ввели: <input name="text\_out"></label>

</form>

</body>

</html>

<script>

var form = document.forms.search;

var text\_in = form.elements.text\_in;

var text\_out = form.elements.text\_out;

text\_in.focus(); /\* автофокус на поле \*/

function show() {

text\_in.blur(); /\* снимает фокус с поля \*/

text\_out.value = text\_in.value;

text\_in.value = '';

}

</script>

Метод **select()** используется для выделения введённого пользователем текста. При этом поле становится активным:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="search">

<label>Введите что-нибудь: <input name="text\_in" oninput="show()"></label>

<br>

<label>Вы ввели: <input name="text\_out"></label>

</form>

</body>

</html>

<script>

var form = document.forms.search;

var text\_in = form.elements.text\_in;

var text\_out = form.elements.text\_out;

text\_in.focus();

function show() {

text\_out.value = text\_in.value;

text\_out.select(); /\* выделяет содержимое \*/

}

</script>

Элемент **<textarea>** аналогичен полю <input>. Для него так же доступны методы focus(), blur(), select(). Их значения и действия полностью совпадают с теми же методами элемента <input>.

Элемент <textarea> не имеет атрибута value, но свойство value всё равно имеет. Его значением является содержимое тега:

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="test" autocomplete="off">

<textarea name="user\_textarea">Текст</textarea>

</form>

</body>

</html>

<script>

var form = document.forms.test;

var textarea = form.elements.user\_textarea;

alert(textarea.value); /\* Текст \*/

</script>

Элемент **<select>** не имеет атрибута value. Однако, свойство value всё равно присутствует. Его значение берётся из той опции (из её свойства value), которая выбрана в данный момент. Если выбрано несколько опций, то значение берётся из первой.

У элементов **option** также есть особые свойства, которые могут оказаться полезными:

- selected – выбрана ли опция;

- index – номер опции в списке селекта;

- text – текстовое содержимое опции (то, что видит посетитель).

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="test" autocomplete="off">

<select name="user\_select" multiple>

<option value="1" selected>Число 1</option>

<option value="2" selected>Число 2</option>

</select>

</form>

</body>

</html>

<script>

var form = document.forms.test;

var select = form.elements.user\_select;

alert(select.value); /\* 1 \*/

</script>

Селект в JavaScript можно установить двумя путями: поставив значение **select.value**, либо установив свойство **select.selectedIndex** в номер нужной опции.

Свойство selectedIndex содержит индекс той опции, которая выбрана в данный момент (если используется атрибут **multiple**, то свойство содержит индекс первой из выбранных опций). Изменяя данное свойство, можно изменить текущее значение в поле<select>. Нумерация начинается с нуля. Значение”-1”устанавливается для очистки поля.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="test" autocomplete="off">

<select name="user\_select">

<option value="1">Число 1</option>

<option value="2" selected>Число 2</option>

<option value="3">Число 3</option>

</select>

</form>

</body>

</html>

<script>

var form = document.forms.test;

var select = form.elements.user\_select;

alert(select.selectedIndex); /\* 1 \*/

select.selectedIndex = -1;

alert(select.value); /\* пустая строка \*/

</script>

Свойство **options** содержит список всех тегов <option> внутри данного элемента <select>.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="user\_form" onsubmit="showNum(); return false" autocomplete="off">

<select name="user\_select">

<option value="1">Число 1</option>

<option value="2">Число 2</option>

<option value="3">Число 3</option>

<option value="4">Число 4</option>

<option value="5">Число 5</option>

<option value="6">Число 6</option>

</select>

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

</form>

</body>

</html>

<script>

var form = document.forms.user\_form;

var select = form.elements.user\_select;

function showNum() {

alert('Доступно опций: ' + select.options.length);

}

</script>

Список элементов-опций доступен через select.options. Свойство **selectedOptions** содержит список тех тегов <option>, которые в данный момент выбраны.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="user\_form" onsubmit="showNum(); return false" autocomplete="off">

<select name="user\_select" multiple>

<option value="1">Число 1</option>

<option value="2">Число 2</option>

<option value="3">Число 3</option>

<option value="4">Число 4</option>

<option value="5">Число 5</option>

<option value="6">Число 6</option>

</select>

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

</form>

</body>

</html>

<script>

var form = document.forms.user\_form;

var select = form.elements.user\_select;

function showNum() {

alert('Выбрано опций: ' + select.selectedOptions.length);

}

</script>

Если select допускает множественный выбор (атрибут multiple), то значения можно получить/установить, сделав цикл по select.options. При этом выбранные опции будут иметь свойство option.selected = true:

<form name="form">

<select name="genre" multiple>

<option value="blues" selected>Мягкий блюз</option>

<option value="rock" selected>Жёсткий рок</option>

<option value="classic">Классика</option>

</select>

</form>

<script>

var form = document.forms[0];

var select = form.elements.genre;

for (var i = 0; i < select.options.length; i++) {

var option = select.options[i];

if(option.selected) {

alert( option.value );

}

}

</script>

Элементы <option>имеют булевое свойство selected. Оно характеризует состояние опции: выбрана или не выбрана.

<html>

<head>

<title>DOM интерфейс</title>

</head>

<body>

<form name="test" autocomplete="off">

<select name="user\_select">

<option value="1">Число 1</option>

<option value="2" selected>Число 2</option>

</select>

</form>

</body>

</html>

<script>

var form = document.forms.test;

var select = form.elements.user\_select;

alert(select.options[0].selected); /\* false \*/

alert(select.options[1].selected); /\* true \*/

</script>

Элемент **label** – один из самых важных в формах. Клик на label засчитывается как фокусировка или клик на элементе формы, к которому он относится. Это позволяет посетителям кликать на большой красивой метке, а не на маленьком квадратике input type=checkbox (radio).

Имеется два способа показать, какой элемент относится к label:

1) присвоить метке атрибут for, равный id соответствующего input:

<table>

<tr>

<td>

<label for="agree">Согласен с правилами</label>

</td>

<td>

<input id="agree" type="checkbox">

</td>

</tr>

<tr>

<td>

<label for="not-a-robot">Я не робот</label>

</td>

<td>

<input id="not-a-robot" type="checkbox">

</td>

</tr>

</table>

2) завернуть элемент в label. В этом случае можно обойтись без дополнительных атрибутов:

<label>Кликни меня <input type="checkbox"></label>

В стандарте **the option element** есть короткий синтаксис для создания элемента с тегом option:

option = new Option(text, value, defaultSelected, selected);

где text – содержимое;

value – значение;

defaultSelected и selected необходимо поставить в true, чтобы сделать элемент выбранным.

Описанный способ можно использовать вместо document.createElement('option'), например:

var option = new Option("Текст", "value");

// создаст <option value="value">Текст</option>

Такой же элемент, но выбранный:

var option = new Option("Текст", "value", true, true);

Форма – это самый популярный способ общения пользователя с сайтом, и очень важно, чтобы данные, ведённые пользователем были корректными. И чтобы форма была отправлена с верными данными необходимо её перед этим проверить.

Создадим форму, которую будем проверять:

<form name = 'form1' action = 'handler.php' method = 'post'>

  Ваше имя:

  <input type = 'text' name = 'firstname' />

  <br />

  Ваш пароль:

  <input type = 'password' name = 'pass' />

  <br />

  Ваш пол:

  <input type = 'radio' name = 'sex' value = 'male' />

  <input type = 'radio' name = 'sex' value = 'female' />

  <br />

  Выберите число:

  <select name = 'number'>

    <option value = '1'>1</option>

    <option value = '2'>2</option>

    <option value = '3'>3</option>

  </select>

  <br />

  Ваше сообщение:

  <textarea name = 'message'></textarea>

  <br />

  Согласитесь с нашими правилами:

  <input type = 'checkbox' name = 'rules' />

  <br />

  <input type = 'hidden' value = 'hidefield' />

  Загрузите файл:

  <input type = 'file' name = 'fileupload' />

  <br />

  <input type = 'submit' value = 'Отправить' name = 'sub' />

  <br />

  <input type = 'button' value = 'Просто кнопка' name = 'but' />

</form>

Выведем все поля, которые подчиняются общему виду:

var form = document.form1;

document.write(form.firstname.value + "<br />");

document.write(form.pass.value + "<br />");

document.write(form.number.value + "<br />");

document.write(form.message.value + "<br />");

document.write(form.rules.value + "<br />");

document.write(form.hidefield.value + "<br />");

document.write(form.fileupload.value + "<br />");

document.write(form.sub.value + "<br />");

document.write(form.but.value + "<br />");

Это все поля, которые имеют свойство value, и к которым можно получить доступ таким образом.

Для того, чтобы получить доступ к значению элемента radio, необходимо:

var sex = (document.form1.sex[0].checked)?

document.form1.sex[0].value : document.form1.sex[1].value;

В массиве sex находятся два элемента radio. Индексы у них 0 и 1. В данном скрипте мы проверяем, если у нас первый элемент включён (checked), то присваиваем значение первого элемента, иначе присваиваем значение второго элемента.

Можно было бы написать и так:

var sex;

if (document.form1.sex[0].checked) sex = document.form1.sex[0].value;

else sex = document.form1.sex[1].value;

Далее в тег <form> необходимо добавить атрибут "onSubmit" со значением "return check();". Данный атрибут будет делать следующее: перед отправкой формы вызывать функцию, которая должна будет вернуть либо true, либо false. Если она вернёт true, то форма отправится на сервер, а если false, то форма не будет отправлена.

Теперь можно создать функцию check(), которая должна, во-первых, проверять полностью форму, сообщать пользователю об ошибках, а также возвращать true (если форма полностью правильная), либо false (если форма содержит ошибки).

function check(form) {

  var firstname = form.firstname.value;

  var pass = form.pass.value;

  var message = form.message.value;

  var rules = form.rules.value;

  var file = form.fileupload.value;

  var bad = "";

  if (firstname.length < 3)

     bad += "Имя слишком короткое" + "\n";

  if (firstname.length > 32)

    bad += "Имя слишком длинное" + "\n";

  if (pass.length < 3)

    bad += "Пароль слишком короткий" + "\n";

  if (pass.length > 32)

    bad += "Пароль слишком длинный" + "\n";

  if (message.length < 3)

    bad += "Сообщение слишком короткое" + "\n";

  if (rules != "on")

    bad += "Вы не согласились с правилами" + "\n";

  if (file.length == 0)

    bad += "Вы не выбрали файл для загрузки" + "\n";

  if (bad != "") {

    bad = "Неверно заполнена форма:" + "\n" + bad;

    alert(bad);

    return false;

  }

  return true;

}

В данном скрипте мы сначала получаем все данные, нуждающиеся в проверке, и записываем их в переменные. Затем создаём переменную bad, в которую записываем все некорректные данные. Затем идёт целый набор IF, которые проверяют поля в форме и записывают все ошибки в переменную bad. Затем, если переменная bad не пустая (то есть были ошибки), то выводим окно (alert()) с ошибками. И возвращает false, чтобы форма не была отправлена. Если переменная bad пустая, то дальше просто возвращаем true, чтобы форма была отправлена уже на обработку в "handler.php".

Динамическая проверка формы на JavaScript может выглядеть следующим образом:

<form name="myform" action="#" method="post">

  <p>Логин:

<input type="text" name="login" onkeyup="check(this.value)" />

 <span id="e\_login" style="display: none; color: #c00;">Логин введён неправильно</span>

</p>

</form>

Ключевой момент – это обработчик события onkeyup. Событие keyup отвечает за отпускание клавиши. Обратите внимание, что очень часто ставят onkeypress или onkeydown – это неправильно, так как будет происходить отставание на 1 шаг.

JavaScript-код тоже очень простой:

<script type="text/javascript">

  function check(login) {

    if (login.length < 3) document.getElementById("e\_login").style.display = "inline";

    else document.getElementById("e\_login").style.display = "none";

  }

</script>

В функции check() мы проверяем длину логина, и если она меньше 3-х символов, то мы выводим ошибку (делая соответсвующий элемент видимым), иначе прячем это сообщение.

Аналогично, можно проверять абсолютно любые формы.

Однако, перед отправкой надо целиком проверить ещё раз всю форму с помощью onsubmit у тега form.

Вместе с тем, проверка формы на JavaScript не отменяет проверку формы в PHP.

**5. Порядок выполнения работы**

1. Реализовать произвольную форму с обязательными полями. Например, форму регистрации. Через команду alert вывести значения элементов формы. Использовать элементы text, textarea, checkbox, radio, select, button.
2. Создать тест на любую тематику с числовыми значениями. При нажатии на «готово» функция должна проверить, все ли поля заполнены. Если заполнены все поля, то должен быть выдан результат.
3. Выполнить валидацию форм в соответствии с вариантом.

Вариант 1. Форма для обмена бытовой техники по гарантии.

Вариант 2. Форма сотрудника библиотеки.

Вариант 3. Форма сотрудника ГАИ.

Вариант 4. Форма сотрудника ботанического сада.

Вариант 5. Форма сотрудника ж/д станции.

Вариант 6. Форма сотрудника автовокзала.

Вариант 7. Форма сотрудника отдела кадров.

Вариант 8. Форма менеджера автосалона.

Вариант 9. Форма по начислению студентам стипендии.

Вариант 10. Форма сотрудника туристической фирмы.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Для чего в JavaScript используется объект Form?
2. Опишите способы организации доступа к форме.
3. Каким образом осуществляется доступ к элементам формы?
4. В чем заключается валидация формы?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 26**

## **Тема работы: «Управление CSS-свойствами HTML-элементов при помощи языка JavaScript»**

**1. Цель работы**

Формирование умений управления CSS-свойствами HTML-элементов на языке JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии управления CSS-свойствами HTML-элементов на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Для работы со стилевыми свойствами элементов в JavaScript применяются, главным образом, два подхода:

- изменение свойства style;

- изменение значения атрибута class.

Свойство **style** представляет сложный объект для управления стилем и напрямую сопоставляется с атрибутом style html-элемента. Этот объект содержит набор свойств CSS: element.style.свойствоCSS.

Пример, установим цвет шрифта:

var root = document.documentElement;

// устанавливаем стиль

root.style.color = "blue";

// получаем значение стиля

document.write(root.style.color); // blue

В данном случае название свойства color совпадает со свойством css. Аналогично можно установить цвет с помощью css:

html{

color:blue;

}

Однако ряд свойств css в названиях имеют дефис, например, font-family. В JavaScript для этих свойств дефис не употребляется. Только первая буква, которая идет после дефиса, переводится в верхний регистр:

var root = document.documentElement;

root.style.fontFamily = "Verdana";

С помощью свойства className можно установить атрибут class элемента html:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8" />

    <style>

    .blueStyle{

        color:blue;

        font-family:Verdana;

    }

    .article{

        font-size:20px;

    }

    </style>

</head>

<body>

<div class="article">

<h3>Заголовок статьи</h3>

<p>Первый абзац</p>

<p>Второй абзац</p>

</div>

<script>

var articleDiv = document.querySelector("div.article");

// установка нового класса

articleDiv.className = "blueStyle";

// получаем название класса

document.write(articleDiv.className);

</script>

</body>

</html>

Благодаря использованию классов не придется настраивать каждое отдельное свойство css с помощью свойства style.

Но при этом надо учитывать, что прежнее значение атрибута class удаляется. Поэтому, если необходимо добавить класс, надо объединить его название со старым классом:

articleDiv.className = articleDiv.className + " blueStyle";

И если надо вовсе удалить все классы, то можно присвоить свойству пустую строку:

articleDiv.className = "";

Для управления множеством классов гораздо удобнее использовать свойство classList. Это свойство представляет объект, реализующий следующие методы:

- add(className) – добавляет класс className;

- remove(className) – удаляет класс className;

- toggle(className) – переключает у элемента класс на className. Если класса нет, то он добавляется, если есть, то удаляется.

Пример:

var articleDiv = document.querySelector("div.article");

// удаляем класс

articleDiv.classList.remove("article");

// добавляем класс

articleDiv.classList.add("blueStyle");

// переключаем класс

articleDiv.classList.toggle("article");

**5. Порядок выполнения работы**

1. Создайте две кнопки и текстовое поле ввода.
2. По нажатию на кнопку должно появляться диалоговое окно, в которое необходимо ввести тест. Данный текст должен быть помещен в текстовое поле.
3. По нажатию на вторую кнопку текст, хранящийся в input должен стать курсивного начертания красного цвета. Также, по нажатию на данную кнопку текстовое поле ввода блокируется (не доступно для изменения).
4. Поместить картинку на страницу. При наведении на область картинки высота и ширина должны увеличиваться на 20 px.
5. Реализовать кнопку сброса до начального размера картинки.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой объект Style?
2. Опишите способы изменения стилевых свойств элементов.
3. В каких случаях целесообразно использовать свойство classList?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 27**

## **Тема работы: «Применение объектно-ориентированного подхода в программировании на языке JavaScript»**

**1. Цель работы**

Формирование умений применения объектно-ориентированного подхода в программировании на языке JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии с применением объектно-ориентированного подхода на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Объект в JavaScript – это просто коллекция пар ключ-значение. Однако, в JavaScript нет концепции класса. К примеру, объект с свойствами {name: Linda, age: 21} не является экземпляром какого-либо класса или класса Object. И Object, и Linda являются экземплярами самих себя. Они определяются непосредственно собственным поведением. Тут нет слоя мета-данных (т.е. классов), которые говорили бы этим объектам как нужно себя вести.

Объекты – это просто пары уникальных ключей с соответствующими значениями – такие пары называются свойства. К примеру, вы хотите описать несколько аспектов своего старого друга (назовём его Мишей, он же Mikhail), таких как возраст, имя и пол. Пример представлен на рисунке 27.1

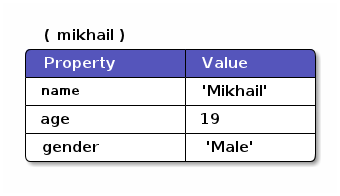


Рисунок 27.1 – Объект JS

Объект в JavaScript создаётся с помощью функции **Object.create**. Эта функция из родителя и опционального набора свойств создаёт новую сущность. Пока что мы не будем беспокоиться о параметрах.

Пустой объект – это объект без родителя, без свойств. Посмотрим на синтакс создания такого объекта в JavaScript:

var mikhail = Object.create(null)

Создание свойств. Например, у нас уже есть объект, но у него пока нет свойств.

Свойства в JavaScript являются динамическими. Это означает, что мы их можем создавать или удалять в любое время. Свойства уникальны в том смысле, что ключ свойства внутри объекта соответствует ровно одному значению.

Создадим новые свойства через функцию **Object.defineProperty**, которая в качестве аргументов использует объект, имя свойства для создания и дескриптор, описывающий семантику свойства.

Object.defineProperty(mikhail, 'name', { value: 'Mikhail'

, writable: true

, configurable: true

, enumerable: true })

Object.defineProperty(mikhail, 'age', { value: 19

, writable: true

, configurable: true

, enumerable: true })

Object.defineProperty(mikhail, 'gender', { value: 'Male'

, writable: true

, configurable: true

, enumerable: true })

Функция Object.defineProperty создаёт новое свойство, если свойство с данным ключём ранее не существовало (в противном случае произойдёт обновление семантики и значения существующего свойства).

Также можно использовать Object.defineProperties когда необходимо добавить больше одного свойства в объект:

Object.defineProperties(mikhail, { name: { value: 'Mikhail'

, writable: true

, configurable: true

, enumerable: true }

, age: { value: 19

, writable: true

, configurable: true

, enumerable: true }

, gender: { value: 'Male'

, writable: true

, configurable: true

, enumerable: true }})

Очевидно, что оба вызова аналогичны, они вполне конфигурируемы, но не предназначены для конечного пользователя кода. Лучше создать уровень абстракции над ними.

Дескрипторы. Маленькие объекты, которые содержат в себе семантику, называются дескрипторами (мы их использовали при вызове Object.defineProperty). Дескрипторы бывают одного из двух типов – дескрипторы данных и дескрипторы доступа.

Оба типа дескрипторов содержат флаги, которые определяют как свойство будет рассматриваться языком. Если флаг не установлен, то его значение по умолчанию false (к сожалению, это не всегда хорошее значение по умолчанию, что влечёт возрастание объёма описания дескрипторов).

Рассмотрим некоторые флаги:

- writable – значение свойства может быть изменено, используется только для дескрипторов данных;

- configurable – тип свойства может быть изменён или свойство может быть удалено;

- enumerable – свойство используется в общем перечислении.

Дескрипторы данных таковы, что определяют конкретное значение, которое соответствует дополнительному value-параметру, описывающему конкретные данные, привязанные к свойству:

- value – значение свойства.

Дескрипторы доступа определяют доступ к конкретному значению через getter-ы и setter-ы функций. Если не установлены, то по умолчанию равны undefined:

- get()– функция вызывается без аргументов, когда происходит запрос к значению свойства;

set(new\_value) – функция вызывается с аргументом – новым значением для свойства, когда пользователь пытается модифицировать значение свойства.

Доступ к свойствам. Очень просто получить значение, хранящиеся в заданном свойстве – синтаксис очень похож на создание свойства с той лишь разницей, что в нём нет присваивания.

Например, если мы хотим узнать возраст Миши, то мы напишем:

mikhail['age']

// => 19

Но если мы попробуем получить значение свойства, которого не существует в нашем объекте, то мы получим undefined:

mikhail['address']

// => undefined

Удаление свойств. Для удаления свойства из объекта в JavaSсript предусмотрен оператор **delete**. К примеру, если вы хотите удалить свойство gender из нашего объекта mikhail:

delete mikhail['gender']

// => true

mikhail['gender']

// => undefined

Оператор delete вернёт true, если свойство удалено, и false в противном случае.

**Getter**-ы и **setter**-ы обычно используются в классических объектно-ориентированных языках для обеспечения инкапсуляции. Но, с любой точки зрения, они позволяет обеспечить proxy для запросов на чтение и запись свойств. Например, у нас были отдельные слоты для имени и фамилии, но мы хотим иметь удобный способ читать и устанавливать их.

Для начала, создадим имя и фамилию нашего друга, описав соответствующие свойства:

Object.defineProperty(mikhail, 'first\_name', { value: 'Mikhail'

, writable: true })

Object.defineProperty(mikhail, 'last\_name', { value: 'Weiß'

, writable: true })

Затем мы опишем общий способ получения и установки сразу двух свойств за один раз — назовём их объединение name:

// () → String

// Returns the full name of object.

function get\_full\_name() {

return this.first\_name + ' ' + this.last\_name

}

// (new\_name:String) → undefined

// Sets the name components of the object, from a full name.

function set\_full\_name(new\_name) { var names

names = new\_name.trim().split(/\s+/)

this.first\_name = names[⁣'0'] || ''

this.last\_name = names['1'] || ''

}

Object.defineProperty(mikhail, 'name', { get: get\_full\_name

, set: set\_full\_name

, configurable: true

, enumerable: true })

Теперь, каждый раз, когда мы попытаемся узнать значение свойства name нашего друга на самом деле, вызовется функция get\_full\_name:

mikhail.name

// => 'Mikhail Weiß'

mikhail.first\_name

// => 'Mikhail'

mikhail.last\_name

// => 'Weiß'

mikhail.last\_name = 'White'

mikhail.name

// => 'Mikhail White'

Также можно установить name объекта, обратившись к соответствующему свойству, но на самом деле вызов set\_full\_name выполнит всю грязную работу:

mikhail.name = 'Michael White'

mikhail.name

// => 'Michael White'

mikhail.first\_name

// => 'Michael'

mikhail.last\_name

// => 'White'

Есть сценарии, в которых действительно удобно так делать, но стоит помнить, что такой механизм работает [очень медленно](http://jsperf.com/getter-setter/8).

Кроме того, следует учитывать что getter-ы и setter-ы обычно используются в других языках для инкапсуляции, а в ECMAScript 5 вы всё ещё не можете так делать – все свойства объекта являются публичными.

Перечисление свойств. Ввиду того, что свойства являются динамическими, JavaScript обеспечивает функционал по проверке набора свойств объекта. Существует два способа перечисления всех свойств объекта, зависящих от того, какой вид свойств вас интересует.

Первый способ заключается в вызове функции **Object.getOwnPropertyNames**, которая вернёт вам **Array**, содержащий имена всех свойств, установленных для данного объекта – мы будет называть эти свойства собственными. Например, посмотрим, что мы знаем о Мише:

Object.getOwnPropertyNames(mikhail)

// => [ 'name', 'age', 'gender', 'first\_name', 'last\_name' ]

Второй способ заключается в использовании **Object.keys**, который вернёт список собственных свойств, которые помечены флагом enumerable :

Object.keys(mikhail)

// => [ 'name', 'age', 'gender' ]

Методы. До сих пор объект Mikhail имел только слоты для хранения данных (ну, за исключением getter/setter для свойства name). Описание действий, которые можно делать с объектом делается в JavaScript очень просто.

Опишем действие над данным объектом, просто установив функцию, как значение нашего свойства. К примеру, мы хотим, чтобы Миша мог приветствовать других людей:

// (person:String) → String

// Greets a random person

mikhail.greet = function(person) {

return this.name + ': Why, hello there, ' + person + '.'

}

После выставления значения свойства, мы можем использовать аналогичный способ для выставления конкретных данных, связанных с объектом. Таким образом, доступ к свойствам будет возвращать ссылку на функцию, хранящуюся в нём, которую мы можем вызвать:

mikhail.greet('you')

// => 'Michael White: Why, hello there, you.'

mikhail.greet('Kristin')

// => 'Michael White: Why, hello there, Kristin.

Существует четыре различных способа разрешения **this** в функции, зависящие от того, как функция вызывается: непосредственно, как метод, явно применяется, как конструктор. Мы посмотрим первые три, а к конструкторам вернёмся позже.

Для следующих примеров примем:

// Returns the sum of the object's value with the given Number

function add(other, yet\_another) {

return this.value + other + (yet\_another || 0)

}

var one = { value: 1, add: add }

var two = { value: 2, add: add }

Если функция вызывается, как метод объекта, то this внутри функции ссылается на сам объект. Т.е. когда мы явно указываем какой объект выполняет действие, то объект и будет значением this в нашей функции.

Это произойдёт, когда мы вызовем mikhail.greet(). Эта запись говорит JavaScript-у, что мы хотим применить действие greet к объекту mikhail.

one.add(two.value) // this === one

// => 3

two.add(3) // this === two

// => 5

one['add'](two.value) // brackets are cool too

// => 3

Когда функция вызывается непосредственно, то this разрешается в глобальный объект движка (window в браузере)

add(two.value) // this === global

// => NaN

// The global object still has no `value' property, let's fix that.

value = 2

add(two.value) // this === global

// => 4

Функция может быть явно применена к любому объекту, несмотря на то, есть ли в объекте соответствующее свойство или нет. Эта функциональность достигается с помощью методов call или apply.

Различие между двумя методами заключается в параметрах передаваемых в функцию и времени исполнения — apply работает примерно в 55 раз медленнее, чем непосредственный вызов, а вот call обычно не особо хуже.

В любом случае, call ожидает объект, как первый параметр функции, за которым следуют обычные аргументы исходной функции:

add.call(two, 2, 2) // this === two

// => 6

add.call(window, 4) // this === global

// => 6

add.call(one, one.value) // this === one

// => 2

С другой стороны, apply позволяет описывать вторым параметром массив параметров исходной функции:

add.apply(two, [2, 2]) // equivalent to two.add(2, 2)

// => 6

add.apply(window, [ 4 ]) // equivalent to add(4)

// => 6

add.apply(one, [one.value]) // equivalent to one.add(one.value)

// => 2

Разрешение this в null или undefined зависит от семантики используемого движка. Обычно результат бывает таким же, как и применение функции к глобальному объекту. Но если движок работает в [strict mode](https://developer.mozilla.org/en/JavaScript/Strict_mode), то this будет разрешено как и ожидается – ровно в ту вещь, к которой применяется:

window.value = 2

add.call(undefined, 1) // this === window

// => 3

void function() {

"use strict"

add.call(undefined, 1) // this === undefined

// => NaN

// Since primitives can't hold properties.

}()

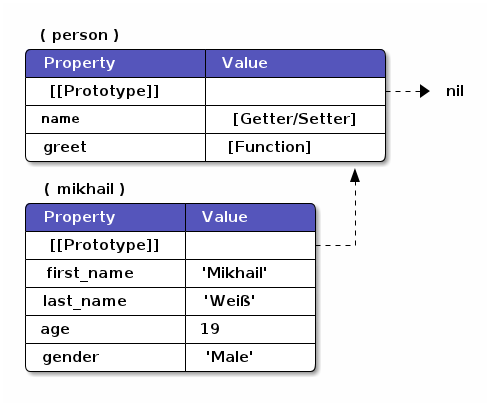
Наследование в JavaScript осуществляется через клонирование поведения объекта и расширение его специализированным поведением. Объект, поведение которого клонируют, называется прототипом.

Прототип – это обычный объект, который делится своим поведением с другими объектами – в этом случае он выступает в качестве родителя.

Концепт клонирования поведения не означает, что вы будете иметь две различные копии одной и той же функции или данных. На самом деле JavaScript реализует наследование через делегирование, т.е. все свойства хранятся в родителе, а доступ к ним расширен через ребёнка.

Как упоминалось ранее, родитель (или [[Prototype]]) объекта определяется вызовом Object.create с первым аргументом, ссылающимся на объект-родитель.

Вернёмся к примеру с Мишей. Выделим его имя и способность приветствовать людей в отдельный объект, который поделится с Мишей своим поведением. На рискнке 27.2 представлена описанная модель.

  
Рисунок 27.2 – Модель объекта

Реализуем её на JavaScript:

var person = Object.create(null)

// Here we are reusing the previous getter/setter functions

Object.defineProperty(person, 'name', { get: get\_full\_name

, set: set\_full\_name

, configurable: true

, enumerable: true })

// And adding the `greet' function

person.greet = function (person) {

return this.name + ': Why, hello there, ' + person + '.'

}

// Then we can share those behaviours with Mikhail

// By creating a new object that has it's [[Prototype]] property

// pointing to `person'.

var mikhail = Object.create(person)

mikhail.first\_name = 'Mikhail'

mikhail.last\_name = 'Weiß'

mikhail.age = 19

mikhail.gender = 'Male'

// And we can test whether things are actually working.

// First, `name' should be looked on `person'

mikhail.name

// => 'Mikhail Weiß'

// Setting `name' should trigger the setter

mikhail.name = 'Michael White'

// Such that `first\_name' and `last\_name' now reflect the

// previously name setting.

mikhail.first\_name

// => 'Michael'

mikhail.last\_name

// => 'White'

// `greet' is also inherited from `person'.

mikhail.greet('you')

// => 'Michael White: Why, hello there, you.'

// And just to be sure, we can check which properties actually

// belong to `mikhail'

Object.keys(mikhail)

Как вы видели в прошлом примере, ни одно из свойств, определённых в Person мы не определяли явно в Mikhail, но всё же смогли получить к ним доступ. Это произошло благодаря тому, что JavaScript реализует делегирование доступа к свойствам, т.е. свойство ищется через всех родителей объекта.

Эта цепь родителей определяется скрытым слотом в каждом объекте, который называется [⁣[Prototype]⁣]. Вы не можете изменить его непосредственно, существует только один способ задать ему значение – при создании нового объекта.

Когда свойство запрашивается из объекта, движок сначала пытается получить свойство из целевого объекта. Если свойство не найдено, то рассматривается непосредственный родитель объекта, затем родитель родителя и т.д.

Это означает, что мы можем изменить поведение прототипа в середине программы, то автоматически изменится поведение всех объектов, которые были от него унаследованы. Например, пусть мы хотим изменить приветствие по умолчанию:

// (person:String) → String

// Greets the given person

person.greet = function(person) {

return this.name + ': Harro, ' + person + '.'

}

mikhail.greet('you')

// => 'Michael White: Harro, you.'

Самым очевидным решением является назначение **\_\_proto\_\_** в конструкторе.

Например, если я хочу, чтобы у всех объектов, которые создаются new Rabbit, был прототип animal, я могу сделать так:

var animal = {

eats: true

};

function Rabbit(name) {

this.name = name;

this.\_\_proto\_\_ = animal;

}

var rabbit = new Rabbit("Кроль");

alert( rabbit.eats ); // true, из прототипа

Чтобы новым объектам автоматически ставить прототип, конструктору ставится свойство prototype.

При создании объекта через new, в его прототип \_\_proto\_\_ записывается ссылка из prototype функции-конструктора.

Например, код ниже полностью аналогичен предыдущему и является кроссбраузерным.

var animal = {

eats: true

};

function Rabbit(name) {

this.name = name;

}

Rabbit.prototype = animal;

var rabbit = new Rabbit("Кроль"); // rabbit.\_\_proto\_\_ == animal

alert( rabbit.eats ); // true

Установка Rabbit.prototype = animal буквально говорит интерпретатору следующее: "При создании объекта через new Rabbit запиши ему \_\_proto\_\_ = animal".

Свойство с именем prototype можно указать на любом объекте, но особый смысл оно имеет, лишь если назначено функции-конструктору.

Само по себе, без вызова оператора new, оно вообще ничего не делает, его единственное назначение – указывать \_\_proto\_\_ для новых объектов.

Технически, в это свойство можно записать что угодно.

Однако, при работе new, свойство prototype будет использовано лишь в том случае, если это объект. Примитивное значение, такое как число или строка, будет проигнорировано.

У каждой функции по умолчанию уже есть свойство prototype.

Оно содержит объект такого вида:

function Rabbit() {}

Rabbit.prototype = {

constructor: Rabbit

};

В коде выше создание Rabbit.prototype происходило вручную, но ровно такой же – генерируется автоматически.

Проверим:

function Rabbit() {}

// в Rabbit.prototype есть одно свойство: constructor

alert( Object.getOwnPropertyNames(Rabbit.prototype) ); // constructor

// оно равно Rabbit

alert( Rabbit.prototype.constructor == Rabbit ); // true

Можно его использовать для создания объекта с тем же конструктором, что и данный:

function Rabbit(name) {

this.name = name;

alert( name );

}

var rabbit = new Rabbit("Кроль");

var rabbit2 = new rabbit.constructor("Крольчиха");

Эта возможность бывает полезна, когда, получив объект, мы не знаем в точности, какой у него был конструктор (например, сделан вне нашего кода), а нужно создать такой же.

JavaScript никак не использует свойство constructor. То есть, оно создаётся автоматически, а что с ним происходит дальше – это уже наша забота. В стандарте прописано только его создание.

В частности, при перезаписи Rabbit.prototype = { jumps: true } свойства constructor больше не будет.

Сам интерпретатор JavaScript его в служебных целях не требует, поэтому в работе объектов ничего не «сломается». Но если мы хотим, чтобы возможность получить конструктор, всё же, была, то можно при перезаписи гарантировать наличие constructor вручную:

Rabbit.prototype = {

jumps: true,

constructor: Rabbit

};

Либо можно поступить аккуратно и добавить свойства к встроенному prototype без его замены:

// сохранится встроенный constructor

Rabbit.prototype.jumps = true

**5. Порядок выполнения работы**

1. Реализуйте на ООП программу-напоминалку. Алгрим работы следующи: вы записываете в специальное поле какие-то события, о которых вам нужно напомнить и в нужное время вкладка браузера с этой программой должна напомнить об этом событии.

2. Ресторан предлагает 2 вида бутербродов: маленький (5 byn, 20 калорий) и большой (10 byn, 40 калорий). Бутерброд может быть с одним из нескольких видов начинок (обязательно):

* сыром (+ 1 byn, + 20 калорий)
* салатом (+ 2 byn, + 5 калорий)
* картофелем (+ 1.5 byn, + 10 калорий)

Дополнительно, бутерброд можно посыпать приправой (+ 1.5 byn, 0 калорий) и полить майонезом (+ 2 byn, + 5 калорий). Напиши программу, рассчитывающую стоимость и калорийность бутерброда. Используй ООП подход (подсказка: нужен класс Бутерброд, константы, методы для выбора опций и расчета нужных величин).

Код должен быть защищен от ошибок. Представьте, что вашим классом будет пользоваться другой программист. Если он передает неправильный тип бутерброда, например, или неправильный вид добавки, должно выбрасываться исключение (ошибка не должна молча игнорироваться).

Написанный класс должен содержать указанные методы, которые принимают и возвращают данные указанного типа и выбрасывают исключения указанного типа.

3. Выполнить задание в соответствии с вариантом.

Вариант 1. Реализуйте класс Worker (Работник), который будет иметь следующие свойства: name (имя), surname (фамилия), rate (ставка за день работы), days (количество отработанных дней). Также класс должен иметь метод getSalary(), который будет выводить зарплату работника. Зарплата – это произведение (умножение) ставки rate на количество отработанных дней days.

Вариант 2. Реализуйте класс MyString, который будет иметь следующие методы: метод reverse(), который параметром принимает строку, а возвращает ее в перевернутом виде, метод ucFirst(), который параметром принимает строку, а возвращает эту же строку, сделав ее первую букву заглавной и метод ucWords, который принимает строку и делает заглавной первую букву каждого слова этой строки.

Вариант 3. Реализуйте класс Student (Студент), который будет наследовать от класса User. Этот класс должен иметь следующие свойства: name (имя, наследуется от User), surname (фамилия, наследуется от User), year (год поступления в вуз). Класс должен иметь метод getFullName() (наследуется от User), с помощью которого можно вывести одновременно имя и фамилию студента. Также класс должен иметь метод getCourse(), который будет выводить текущий курс студента (от 1 до 5). Курс вычисляется так: нужно от текущего года отнять год поступления в вуз. Текущий год получите самостоятельно.

Вариант 4. Реализуйте класс Rectangle. У него должны быть следующие свойства: ширина width, высота height. Также у него должны быть следующие методы: получить ширину getWidth, установить ширину setWidth, получить высоту getHeight, установить высоту setHeight.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой объект в JavaScript?
2. Перечислите способы создания объектов в JavaScript.
3. Для чего используется функция Object.defineProperty?
4. Что представляют собой getter-ы и setter-ы?
5. Что представляет собой прототип?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 28**

## **Тема работы: «Применение функций временной задержки. Анимации в JavaScript»**

**1. Цель работы**

Формирование умений применения функций временной задержки, навыков создания анимации средствами JavaScript.

**2. Задание**

Создать HTML-документ, содержащий сценарии с использованием функций временной задержки на языке JavaScript.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

JavaScript-анимация применяется там, где не подходит CSS.

Например, по сложной траектории, с временной функцией, выходящей за рамки кривых Безье, на canvas. Иногда её используют для анимации в старых IE.

С точки зрения HTML/CSS, анимация – это постепенное изменение стиля DOM-элемента. Например, увеличение координаты style.left от 0px до 100px сдвигает элемент.

Если увеличивать left от 0 до 100 при помощи setInterval, делая по 50 изменений в секунду, то это будет выглядеть как плавное перемещение. Тот же принцип, что и в кино: для непрерывной анимации достаточно 24 или больше вызовов setInterval в секунду.

Псевдокод для анимации выглядит следующимобразом:

var fps = 50; // 50 кадров в секунду

var timer = setInterval(function() {

if (время вышло) clearInterval(timer);

else немного увеличить left

}, 1000 / fps)

Более полный пример кода анимации:

var start = Date.now(); // сохранить время начала

var timer = setInterval(function() {

// вычислить сколько времени прошло с начала анимации

var timePassed = Date.now() - start;

if (timePassed >= 2000) {

clearInterval(timer); // конец через 2 секунды

return;

}

// рисует состояние анимации, соответствующее времени timePassed

draw(timePassed);

}, 20);

// в то время как timePassed идёт от 0 до 2000

// left принимает значения от 0 до 400px

function draw(timePassed) {

train.style.left = timePassed / 5 + 'px';

}

Если у нас не один такой setInterval, а несколько в разных местах кода, то браузеру нужно в те же 20 мс работать со страницей уже несколько раз. А ведь кроме setInterval есть ещё другие действия, к примеру, прокрутка страницы, которую тоже надо нарисовать.

Если все действия по перерисовке производить независимо, то будет выполняться много двойной работы.

Гораздо выгоднее с точки зрения производительности – сгруппировать все перерисовки в одну и запускать их централизованно, все вместе.

Для этого в JavaScript-фреймворках, которые поддерживают анимацию, есть единый таймер:

setInterval(function() {

/\* отрисовать все анимации \*/

}, 20);

Все анимации, которые запускает такой фреймворк, добавляются в общий список, и раз в 20 мс единый таймер проверяет его, запускает текущие, удаляет завершившиеся.

Современные браузеры, кроме IE9-, поддерживают стандарт Animation timing, который представляет собой дальнейший шаг в этом направлении. Он позволяет синхронизировать наши анимации со встроенными механизмами обновления страницы. То есть, сгруппированы будут не только наши, но и CSS-анимации и другие браузерные перерисовки.

При этом графический ускоритель будет использован максимально эффективно, и исключена повторная обработка одних и тех же участков страницы. А значит – меньше будет загрузка CPU, да и сама анимация станет более плавной.

Для этого используется функция requestAnimationFrame.

Синтаксис:

var requestId = requestAnimationFrame(callback)

Такой вызов планирует запуск callback в ближайшее время, когда браузер сочтёт возможным осуществить анимацию.

Если запланировать в callback какое-то рисование, то оно будет сгруппировано с другими requestAnimationFrame и с внутренними перерисовками браузера.

Возвращаемое значение requestId служит для отмены запуска:

// отменить запланированное выше выполнение callback

cancelAnimationFrame(requestId);

Функция callback получает один аргумент – время, прошедшее с начала загрузки страницы, результат вызова performance.now().

Как правило, запуск callback происходит очень скоро. Если у процессора большая загрузка или батарея у ноутбука почти разряжена – то пореже.

Если вы запустите этот код, то увидите промежутки между первыми 20 запусками requestAnimationFrame. Как правило, это 10-20 мс, но бывает и больше и меньше. Это оптимальная частота анимации с точки зрения браузера.

<script>

var prev = performance.now();

var times = 0;

requestAnimationFrame(function measure(time) {

document.body.insertAdjacentHTML("beforeEnd", Math.floor(time - prev) + " ");

prev = time;

if (times++ < 10) requestAnimationFrame(measure);

})

</script>

Функция анимации на основе requestAnimationFrame:

// Рисует функция draw

// Продолжительность анимации duration

function animate(draw, duration) {

var start = performance.now();

requestAnimationFrame(function animate(time) {

// определить, сколько прошло времени с начала анимации

var timePassed = time - start;

// возможно небольшое превышение времени, в этом случае зафиксировать конец

if (timePassed > duration) timePassed = duration;

// нарисовать состояние анимации в момент timePassed

draw(timePassed);

// если время анимации не закончилось – запланировать ещё кадр

if (timePassed < duration) {

requestAnimationFrame(animate);

}

});

}

Использование для поезда:

animate(function(timePassed) {

train.style.left = timePassed / 5 + 'px';

}, 2000);

На основе requestAnimationFrame можно соорудить и гораздо более мощную, но в то же время простую функцию анимации.

У анимации есть три основных параметра:

1) **duration** – общее время, которое должна длиться анимация, в мс. Например, 1000;

2) **timing(timeFraction)** – временная функция, которая, по аналогии с CSS-свойством transition-timing-function, будет по текущему времени вычислять состояние анимации. Она получает на вход непрерывно возрастающее число timeFraction – от 0 до 1, где 0 означает самое начало анимации, а 1 – её конец. Её результатом должно быть значение завершённости анимации, которому в CSS transitions на кривых Безье соответствует координата y. Также по аналогии с transition-timing-function должны соблюдаться условия:

timing(0) = 0

timing(1) = 1

То есть, анимация начинается в точке (0,0) – нулевое время и нулевой прогресс и заканчивается в (1, 1) – прошло полное время, и процесс завершён. Например, функция-прямая означает равномерное развитие процесса:

function linear(timeFraction) {

return timeFraction;

}

График функции представлен на рисунке 28.1

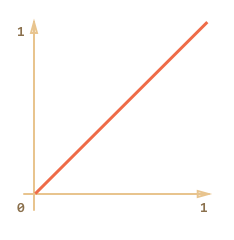


Рисунок 28.1 – График линейной функции

Как видно, график полностью совпадает с transition-timing-function: linear, и эффект абсолютно такой же;

3) **draw(progress)** – функция, которая получает состояние завершённости анимации и рисует его. Значению progress=0 соответствует начальная точка анимации, progress=1 – конечная. Именно эта функция и осуществляет, собственно, анимацию.

Например, может двигать элемент:

function draw(progress) {

train.style.left = progress + 'px';

}

Возможны любые варианты, анимировать можно что угодно и как угодно.

Анимируем ширину элемента width от 0 до 100%, используя нашу функцию. Рассмотрим примеры анимации движения с использованием различных timing.

Вот еще один простой случай – **progress** в степени n. Частные случаи – квадратичная, кубическая функции и т.д.

Для квадратичной функции:

function quad(progress) {

return Math.pow(progress, 2)

}

График квадратичной функции представлен на рисунке 28.2.

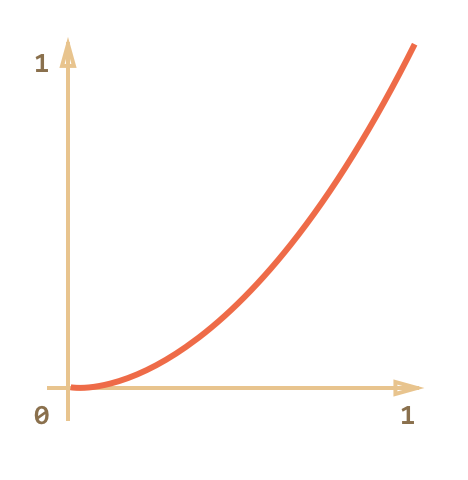


Рисунок 28.2 – График квадратичной функции

Дуга:

function circ(timeFraction) {

return 1 - Math.sin(Math.acos(timeFraction))

}

Представьте, что мы отпускаем мяч, он падает на пол, несколько раз отскакивает и останавливается.

Функция **bounce** делает то же самое, только наоборот: «подпрыгивание» начинается сразу.

Эта функция немного сложнее предыдущих и использует специальные коэффициенты:

function bounce(timeFraction) {

for (var a = 0, b = 1, result; 1; a += b, b /= 2) {

if (timeFraction >= (7 - 4 \* a) / 11) {

return -Math.pow((11 - 6 \* a - 11 \* timeFraction) / 4, 2) + Math.pow(b, 2)

}

}

}

**Упругая** анимация. Эта функция зависит от дополнительного параметра x, который определяет начальный диапазон.

function elastic(x, timeFraction) {

return Math.pow(2, 10 \* (timeFraction - 1)) \* Math.cos(20 \* Math.PI \* x / 3 \* timeFraction)

}

Набор текста. Можно, к примеру, анимировать набор текста в «скачущем» режиме. Анимация выполняется путём вызовов requestAnimationFrame. Для поддержки IE9- желательно подключить полифилл, который будет внутри использовать setTimeout. Это будет всё равно лучше, чем независимые setInterval.

Реализация анимации – очень простая и вместе с тем гибкая:

function animate(options) {

var start = performance.now();

requestAnimationFrame(function animate(time) {

// timeFraction от 0 до 1

var timeFraction = (time - start) / options.duration;

if (timeFraction > 1) timeFraction = 1;

// текущее состояние анимации

var progress = options.timing(timeFraction)

options.draw(progress);

if (timeFraction < 1) {

requestAnimationFrame(animate);

}

});

}

**5. Порядок выполнения работы**

1. Реализуйте слайдер минимум из 5 изображений. Реализуйте плавную смену изображений.

2. Выполните задание в соответствии с вариантом.

Вариант 1.

1. Полет мяча (bounce)
2. Горизонтальное двухуровневое меню, раскрывающееся при наведении указателя мыши. Второй уровень меню скрывается, когда указатель мыши покидает область выбранного пункта или появившегося меню.
3. Разместите на странице два изображения и кнопку «Переставить». При нажатии на данную кнопку изображения должны поменяться местами.

Вариант 2.

1. Полет снежинки (синусоида).
2. Вертикальное двухуровневое меню, раскрывающееся при наведении указателя мыши. Второй уровень меню скрывается, когда указатель мыши покидает область выбранного пункта или появившегося меню.
3. Разместите на странице четыре изображения, два поля ввода и кнопку «Переставить». При нажатии на данную кнопку изображения с номерами, введенными в текстовые поля, должны поменяться местами

Вариант 3.

1. Полет мухи (random).
2. Горизонтальное двухуровневое меню, раскрывающееся при щелчке левой кнопкой мыши. Второй уровень меню скрывается, когда указатель мыши покидает область выбранного пункта или появившегося меню.
3. Добавьте на страницу изображение; при наведении указателя мыши оно должно постепенно увеличиваться в размерах, создавая иллюзию приближения изображения (предельные значения высоты и ширины установите самостоятельно).

Вариант 4.

1. Рисование заданного графика (sin(x)).
2. Вертикальное двухуровневое меню, раскрывающееся при щелчке левой кнопкой мыши. Второй уровень меню скрывается, когда указатель мыши покидает область выбранного пункта или появившегося меню.
3. Добавьте на страницу изображение. При наведении указателя мыши оно должно постепенно уменьшаться в размерах, создавая иллюзию удаления изображения (предельные значения высоты и ширины установите самостоятельно).

Вариант 5.

1. Рисование заданного графика (x2).
2. Горизонтальное двухуровневое меню, раскрывающееся при щелчке левой кнопкой мыши. Второй уровень меню скрывается при повторном щелчке по пункту меню.
3. Организуйте движение изображения слева направо.

Вариант 6.

1. Полет мяча (bounce).
2. Создайте вертикальное двухуровневое меню, раскрывающееся при щелчке левой кнопкой мыши. Второй уровень меню скрывается при повторном щелчке по выбранному пункту.
3. Организуйте движение изображения справа налево.

Вариант 7.

1. Полет снежинки (синусоида).
2. Создайте текстовое двухуровневое меню: при выборе определенного пункта меню пункты верхнего уровня раздвигаются и вставляются подпункты выбранного пункта, то есть пункт меню «раскрывается».
3. Организуйте движение изображения из левого верхнего угла окна к правому нижнему.

Вариант 8

1. Полет мухи (random).
2. Создайте вертикальное двухуровневое меню, раскрывающееся при наведении указателя мыши. Второй уровень меню скрывается, когда указатель мыши покидает область выбранного пункта или появившегося меню.
3. Организуйте движение изображения из правого верхнего угла окна к левому нижнему.

Вариант 9.

1. Рисование заданного графика (sin(x)).
2. Создайте горизонтальное двухуровневое меню, раскрывающееся при щелчке левой кнопкой мыши. Второй уровень меню скрывается при повторном щелчке по пункту меню.
3. Организуйте движение изображения сверху вниз

Вариант 10.

1. Рисование заданного графика (x2).
2. Создайте горизонтальное двухуровневое меню, раскрывающееся при наведении указателя мыши. Второй уровень меню скрывается, когда указатель мыши покидает область выбранного пункта или появившегося меню.
3. Организуйте движение изображения снизу вверх

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Для чего необходим setInterval?
2. Опишите структуру анимации.
3. Охарактеризуйте функцию bounce.

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 29**

## **Тема работы: «Создание приложения с использованием асинхронных событий»**

**1. Цель работы**

Формирование умений разработки приложений web 2.0 c применением асинхронных событий.

**2. Задание**

Создать HTML-документ, содержащий сценарии асинхронного выполнения событий.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

События могут возникать не только по очереди, но и «пачкой» по много сразу. Возможно и такое, что во время обработки одного события возникают другие, например, пока выполнялся код для onclick – посетитель нажал кнопку на клавиатуре (событие keydown).

Здесь мы разберём, как браузер обычно работает с одновременно возникающими событиями и какие есть исключения из общего правила.

В каждом окне выполняется только один главный поток, который занимается выполнением JavaScript, отрисовкой и работой с DOM.

Он выполняет команды последовательно, может делать только одно дело одновременно и блокируется при выводе модальных окон, таких как alert.

Есть и другие, служебные потоки, например, для сетевых коммуникаций.

Поэтому скачивание файлов может продолжаться пока главный поток ждёт реакции на alert. Но управлять служебными потоками мы не можем.

Существует спецификация [Web Workers](http://www.w3.org/TR/workers/), которая позволяет запускать дополнительные JavaScript-процессы(workers).

Они могут обмениваться сообщениями с главным процессом, но у них свои переменные, и работают они также сами по себе.

Такие дополнительные процессы не имеют доступа к DOM, поэтому они полезны, преимущественно, при вычислениях, чтобы загрузить несколько ядер/процессоров одновременно.

Если произошло одновременно несколько событий или во время работы одного случилось другое, а главный поток прямо сейчас занят, то он не может срочно выйти из середины одной функции и прыгнуть в другую. А потом третью. Отладка при этом могла бы превратиться в кошмар, потому что пришлось бы разбираться с совместным состоянием нескольких функций сразу.

Поэтому используется альтернативный подход. Когда происходит событие, оно попадает в очередь. Внутри браузера непрерывно работает «главный внутренний цикл», который следит за состоянием очереди и обрабатывает события, запускает соответствующие обработчики и т.п. Иногда события добавляются в очередь сразу пачкой.

Например, при клике на элементе генерируется несколько событий:

1. Сначала mousedown – нажата кнопка мыши.
2. Затем mouseup – кнопка мыши отпущена и, так как это было над одним элементом, то дополнительно генерируется click (два события сразу).

В действии:

<textarea rows="8" cols="40" id="area">Кликни меня

</textarea>

<script>

area.onmousedown = function(e) { this.value += "mousedown\n"; this.scrollTop = this.scrollHeight; };

area.onmouseup = function(e) { this.value += "mouseup\n"; this.scrollTop = this.scrollHeight; };

area.onclick = function(e) { this.value += "click\n"; this.scrollTop = this.scrollHeight; };

</script>

Таким образом, при нажатии кнопки мыши в очередь попадёт событие mousedown, а при отпускании – сразу два события: mouseup и click. Браузер обработает их строго одно за другим: mousedown → mouseup → click.

При этом каждое событие из очереди обрабатывается полностью отдельно от других.

Обычно возникающие события «становятся в очередь». Но в тех случаях, когда событие инициируется не посетителем, а кодом, то оно, как правило, обрабатывается синхронно, то есть прямо сейчас.

Рассмотрим в качестве примера событие onfocus. Когда посетитель фокусируется на элементе, возникает событие onfocus. Обычно оно происходит, когда посетитель кликает на поле ввода, например:

<p>При фокусе на поле оно изменит значение.</p>

<input type="text" onfocus="this.value = 'Фокус!'" value="Кликни меня">

Но ту же фокусировку можно вызвать и явно, вызовом метода elem.focus():

<input type="text" id="elem" onfocus="this.value = 'Фокус!'">

<script>

// сфокусируется на input и вызовет обработчик onfocus

elem.focus();

</script>

При этом обработчик onclick вызовет метод focus() на текстовом поле text. Код обработчика onfocus, который при этом запустится, сработает синхронно, прямо сейчас, до завершения onclick.

<input type="button" id="button" value="Нажми меня">

<input type="text" id="text" size="60">

<script>

button.onclick = function() {

text.value += ' ->в onclick ';

text.focus(); // вызов инициирует событие onfocus

text.value += ' из onclick-> ';

};

text.onfocus = function() {

text.value += ' !focus! ';

};

</script>

При клике на кнопке в примере выше будет видно, что управление вошло в onclick, затем перешло в onfocus, затем вышло из onclick.

Так ведут себя все браузеры, кроме IE. В нём событие onfocus – всегда асинхронное, так что будет сначала полностью обработан клик, а потом – фокус. В остальных – фокус вызовется посередине клика. Попробуйте кликнуть в IE и в другом браузере, чтобы увидеть разницу.

А что, если мы хотим, чтобы сначала закончилась обработка onclick, а потом уже произошла обработка onfocus и связанные с ней действия?

Можно добиться и этого.

Один вариант – просто переместить строку text.focus() вниз кода обработчика onclick.

Если это неудобно, можно запланировать text.focus() чуть позже через setTimeout(..., 0), вот так:

<input type="button" id="button" value="Нажми меня">

<input type="text" id="text" size="60">

<script>

button.onclick = function() {

text.value += ' ->в onclick ';

setTimeout(function() {

text.focus(); // сработает после onclick

}, 0);

text.value += ' из onclick-> ';

};

text.onfocus = function() {

text.value += ' !focus! ';

};

</script>

Выводы:

- JavaScript выполняется в едином потоке. Современные браузеры позволяют порождать подпроцессы [Web Workers](http://www.w3.org/TR/workers/), они выполняются параллельно и могут отправлять/принимать сообщения, но не имеют доступа к DOM;

- обычно события становятся в очередь и обрабатываются в порядке поступления, асинхронно, независимо друг от друга;

- синхронными являются вложенные события, инициированные из кода;

- чтобы сделать событие гарантированно асинхронным, используется вызов через setTimeout(func, 0).

Отложенный вызов через setTimeout(func, 0) используется не только в событиях, а вообще – всегда, когда мы хотим, чтобы некая функция func сработала после того, как текущий скрипт завершится

**5. Порядок выполнения работы**

1. Создайте текстовое поле ввода.

2. Измените содержимое поля ввода при получении фокуса.

3. Создайте асинхронное событие для обработки формы, в котором сперва сработает событие onclick(), а потом onfocus().

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. В каком порядке обрабатываются события в JavaScript?
2. Дайте характеристуку асинхронных событий.
3. Чем асинхронные события отличаются от синхронны?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 30**

## **Тема работы: «Получение данных в формате JSON и их обработка на стороне клиента»**

**1. Цель работы**

Формирование умений обработки данных в формате JSON.

**2. Задание**

Создать HTML-документ, содержащий сценарии обработки данных в формате JSON.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Для того, чтобы создать строку JSON необходимо руководствоваться основными правилами для создания строки JSON.

Строка JSON содержит либо массив значений, либо объект (ассоциативный массив пар имя/значение).

Массив заключается в квадратные скобки ([ и ]) и содержит разделенный запятой список значений.

Объект заключается в фигурные скобки ({ и }) и содержит разделенный запятой список пар имя/значение.

Пара имя/значение состоит из имени поля, заключенного в двойные кавычки (“ ”), за которым следует двоеточие (:) и значение поля.

Значение в массиве или объекте может быть:

- числом (целым или с плавающей точкой);

- строкой (в двойных кавычках);

- логическим значением (true или false);

- другим массивом (заключенным в квадратные скобки);

- другой объект (заключенный в фигурные скобки);

- значение null.

Чтобы включить двойные кавычки в строку, нужно использовать обратную косую черту: \". Так же, как и во многих языках программирования, можно помещать управляющие символы и шестнадцатеричные коды в строку, предваряя их обратной косой чертой. Смотрите детали на сайте JSON.

Простой пример строки JSON: оформление заказа в формате JSON:

{

  "orderID": 12345,

  "shopperName": "Ваня Иванов",

  "shopperEmail": "ivanov@example.com",

  "contents": [

    {

      "productID": 34,

      "productName": "Супер товар",

      "quantity": 1

    },

    {

      "productID": 56,

      "productName": "Чудо товар",

      "quantity": 3

    }

  ],

  "orderCompleted": true

}

Рассмотрим строку подробно:

- создаем объект с помощью фигурных скобок  ({ и });

- "orderID": 12345 – Свойство с именем "orderId" и целочисленным значением  12345;

- "shopperName": "Ваня Иванов" – свойство с именем "shopperName" и строковым значением "Ваня Иванов";

- "shopperEmail": [johnsmith@example.com](mailto:johnsmith@example.com) – свойство с именем "shopperEmail" и строковым значением [ivanov@example.com](mailto:ivanov@example.com);

- "contents": [ ... ] – свойство с именем "contents", значение которого является массивом;

- "orderCompleted": true – свойство с именем "orderCompleted" и логическим значением true.

В массиве "contents" есть 2 объекта, представляющие отдельные позиции в заказе. Каждый объект содержит 3 свойства: productID, productName, и quantity.

Кстати, так как JSON основан на объявлении объектов JavaScript, то вы можете быстро и просто сделать выше приведенную строку JSON объектом JavaScript:

<script type="text/javascript">

var cart = {

  "orderID": 12345,

  "shopperName": "Ваня Иванов",

  "shopperEmail": "ivanov@example.com",

  "contents": [

    {

      "productID": 34,

      "productName": "Супер товар",

      "quantity": 1

    },

    {

      "productID": 56,

      "productName": "Чудо товар",

      "quantity": 3

    }

  ],

  "orderCompleted": true

};

</script>

Библиотека jQuery также предоставляет специальный метод getJSON. Этот метод отправляет на сервер GET-запрос и в ответ получает данные json.

Этот метод принимает следующие параметры:

- url: обязательный параметр, содержащий адрес ресурса, к которому будет обращаться запрос;

- data: необязательный параметр, содержащий простой объект javascript или строку, которые будут отправлены на сервер вместе с запросом;

- success(data, textStatus, jqXHR): необязательный параметр, представляющий функцию обратного вызова, которая будет выполняться при успешном выполнении запроса.

На выходе метод getJSON возвращает объект jqXHR, связанный с текущем запросом.

Допустим, на сервере у нас есть следующий файл countries.json, который содержит перечисление стран в формате json:

{

    "Великобритания": 1,

    "Китай": 2,

    "Россия": 3,

    "США": 4,

    "Франция": 5

}

Добавим на страницу выпадающий список (элемент select) и загрузим в него данные из этого файла:

<select id="countries">

    <option disabled>Выберите страну</option>

</select>

<script type="text/javascript">

$(function(){

    $.getJSON('countries.json', function(data) {

                $.each(data, function(key, val) {

                    $('#countries').append('<option value="' + val + '">' + key + '</option>');

                });

    });

});

</script>

В качестве значения параметра url используется название файла json. Так как приходящие данные представляют собой набор, то мы можем пройтись по ним в цикле each и применить к ним функцию обработки.

Обрабатывающая функция вызывается для каждого элемента в наборе и принимает два параметра: название данных или ключ (key) и значение (val). То есть в строке "Россия": 3 "Россия" – это ключ, а 3 – значение.

В данном случае у нас структура данных довольно проста. Теперь посмотрим на использование данных с более сложной структурой. Например, у нас определен такой файл users.json:

{

    "users": [{

        "id" : 1,

        "name": "Bill Gates",

        "age": 43

    }, {

        "id" : 2,

        "name": "Sergey Brin",

        "age": 33

    }, {

        "id" : 3,

        "name": "Larry Page",

        "age": 34

    }]

}

Попробуем вывести его в табличку на странице:

<table id="users">

    <tr><td>Id</td><td>Имя</td><td>Возраст</td><tr>

</table>

<script type="text/javascript">

$(function(){

    $.getJSON('users.json', function(data) {

            for(var i=0;i<data.users.length;i++){

                $('#users').append('<tr><td>' + data.users[i].id + '</td><td>' + data.users[i].name +

                '</td><td>' + data.users[i].age + '</td><tr>');

            }

    });

});

</script>

Используя названия свойств объектов json, мы можем получить к ним доступ (например, data.users[i].age) и тем самым вывести их значения на страницу.

**5. Порядок выполнения работы**

1. Преобразовать объект leader из примера ниже в JSON:

var leader = {

name: "Василий Иванович",

age: 35

};

2. Преобразовать объект team из примера ниже в JSON:

var leader = {

name: "Василий Иванович"

};

var soldier = {

name: "Петька"

};

// эти объекты ссылаются друг на друга!

leader.soldier = soldier;

soldier.leader = leader;

var team = [leader, soldier];

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой формат JSON?
2. Как записываются данные в формате JSON?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 31**

## **Тема работы: «Подключение библиотеки jQuery к приложению и получение доступа к элементам документа»**

**1. Цель работы**

Формирование умений использования библиотеки JQuery.

**2. Задание**

Создать HTML-документ, содержащий сценарии изменения свойств html-элементов с помощью JQuery.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

jQuery – библиотека, которая позволяет делать код короче, а также позволяет внутри страницы настроить код, который бы срабатывал как триггер (предопределенный набор действий, который выполняется автоматически при наступлении связанного с ним события, если этот код описываем в области <head> … </head>).

JQuery библиотека содержит следующий функционал:

- операции с HTML/DOM (манипулирование компонентами HTML/DOM);

- операции с CSS-селекторами;

- HTML-обработчики событий;

- эффекты анимации;

- AJAX;

- Utilities.

JQuery упрощает работу с JavaScript, а также вызовы AJAX и DOM-манипуляции. Есть много фреймворков JavaScript, но JQuery, является самым популярным и используемым за счет своей расширяемости.

Библиотеку jQuery можно скачать с сайта http://jquery.com, а можно вставить в документ, используя известные интернет-адреса:

- по адресу http://code.jquery.com/jquery-latest.js – доступна всегда последняя версия;

- с Google: <https://developers.google.com/speed> /libraries/devguide?hl=ru# jquery можно загрузить любую из не слишком старых версий. Синтаксис такой: src="http://ajax.googleapis.com/ajax/libs/jquery /1.8.3/jquery.min.js", где 1.8.3 — версия, причём можно указать её приблизительно: 1.8 означает последнюю версию вида 1.8.\*, а 1 — последнюю версию вида 1.\*. Файл jquery.min.js обозначает сжатый код, а jquery.js — несжатый, для удобства отладки;

- либо с Microsoft CDN: src="http://ajax.aspnetcdn.com/ ajax/jQuery/jquery-1.9.1.min.js"

Пример подключения приведен на рисунке 31.1 (но использовать один из вариантов подключения библиотеки).



Рисунок 31.1 – Примеры подключения библиотеки jQuery

jQuery синтаксис. Базовая команда для библиотеки вяглядит как: $(селектор).action():

где $ – предписание использовать jQuery;

(селектор) – это "запрос или элементы поиска" в HTML элементах страницы;

action() – это действия, которые должны быть выполнены над найденными элементами (это те элементы, которые удовлетворяют условиям селектора).

Например:

- $(this).hide() – скрывает текущий элемент (где this – это указатель на текущий элемент, позволяет делать код универсальным за счет того, что не надо писать здесь имя или id элемента, над которым будет производится действие hide() );

- $("p").hide() – скрывает все <p> элементы на странице;

- $(".test").hide() – скрывает все элементы на странице, которые ассоциированы с классом "test";

- $("#test").hide() – скрывает все элементы на странице, у которых id="test".

Событие Ready у объекта страницы Document. В большинстве примеров jQuery-методы находятся внутри события документа Ready():

$(document).ready(function(){

// jQuery-методы размещаем здесь...

});

Это необходимо для предотвращения любых срабатываний JQuery-кода, прежде чем документ не закончит полную загрузку. Это хорошая практика, чтобы дождаться, пока документ будет полностью загружен и готов к работе с ним. Это также позволяет сформировать свой JavaScript код в головной части, прежде чем тело документа.

jQuery селекторы позволяют делать выборку (поиск) и манипулировать с элементами HTML. Селекторы по сути это набор условных обозначений и правил для выборки и манипулирования (в конце лабораторной в приложении дан большой список примеров селекторов).

С jQuery селекторами можно найти элементы страницы, основанные на идентификаторе id, классах (class), типах (type), атрибутах (attribute), значениях атрибутов (value) и др. Также они базируются и на CSS Selectors, в дополнение можно создать свой селектор. Все типы селекторов в jQuery начинаются с указания $ и парных скобок: $( ).

Например, в следующем коде при нажатии на кнопку выполняется поиск на странице всех элементов, обозначенных тегом <p> , и все эти элементы скрываются на странице (срабатывает метод hide( ) ):

<!DOCTYPE html>

<html>

<head>

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

</script>

<script>

$(document).ready(function(){

$(“button”).click(function(){

$(“p”).hide();

});

});

</script>

</head>

<body>

<h2>This is a heading</h2>

<p>This is a paragraph.</p>

<p>This is another paragraph.</p>

<button>Click me</button>

</body>

</html>

Пояснения к скрипту:

<script>

//в строке ниже в качестве селектора использован весь объект – документ,

//при этом будет срабатывать jquery-запрос, как только наступит событие полной готовности

//страницы к работе с пользователем и запустится метод ready(),

// при срабатывании которого будет создана следующая функция

$(document).ready(function () {

//созданная функция в свою очередь будет jquery-запросом, который ищет все элементы типа

//button – кнопка, и с их методами click связывает (ассоциирует) действие в виде функции,

$("button").click(function () {

// которая выполнит jquery-запрос, который для всех найденных элементов внутри тега <p> выполнит метод hide(), т.е. скроет их со страницы

$("p").hide();

//далее закрываем внутреннюю функцию

});

//далее закрываем внешнюю функцию

});

</script>

В таком исполнении скрипт, помещенный в заголовок страницы внутрь метода объект document.ready( ) работает как триггер, т.е. автоматически срабатывает при наступлении определенного события на странице. В примере – это событие click( ) кнопки. А так как в методе hide( ) нет никакого описания другого кода, то он выполняет те действия, для которых он изначально создан, а именно скрывает объект.

Селектор jQuery #id использует id атрибут в HTML-тегах, чтобы найти определенный элемент. Id должен быть уникальным внутри всей страницы, если вы хотите найти с его помощью конкретный уникальный элемент. Чтобы найти элемент с помощью id, то перед названием искомого идентификатора ставится знак #, например: $("#test")

В примере на рисунке 31.2, при нажатии на кнопку ищется элемент с идентификатором test и скрывается со страницы.

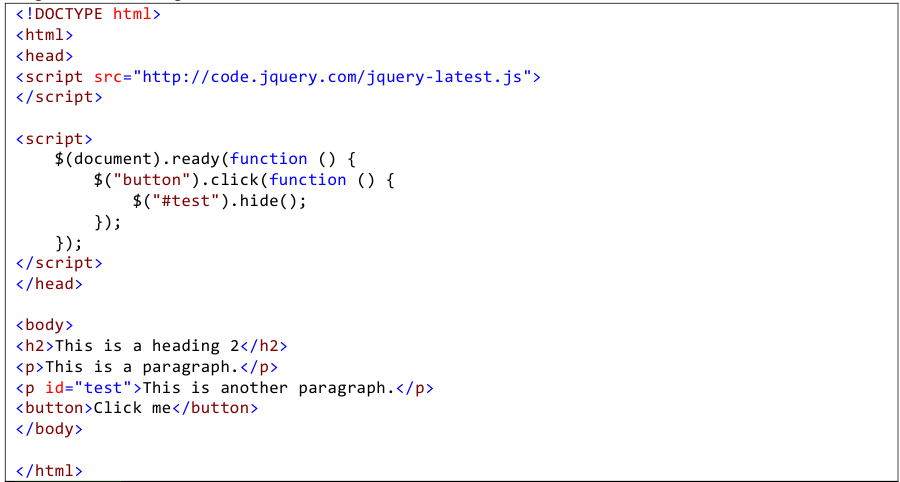


Рисунок 31.2 – Пример поиска элемента по идентификатору

На рисунке 31.3 приведен пример, в котором работают две кнопки: одна – скрывает элемент с id=”test”, вторая – отображает этот элемент (используется метод show( ) ). Обратите внимание, чтобы распараллелить код по двум кнопкам, для каждой из них тоже были определены id, по которым определяется какую функцию запускать.

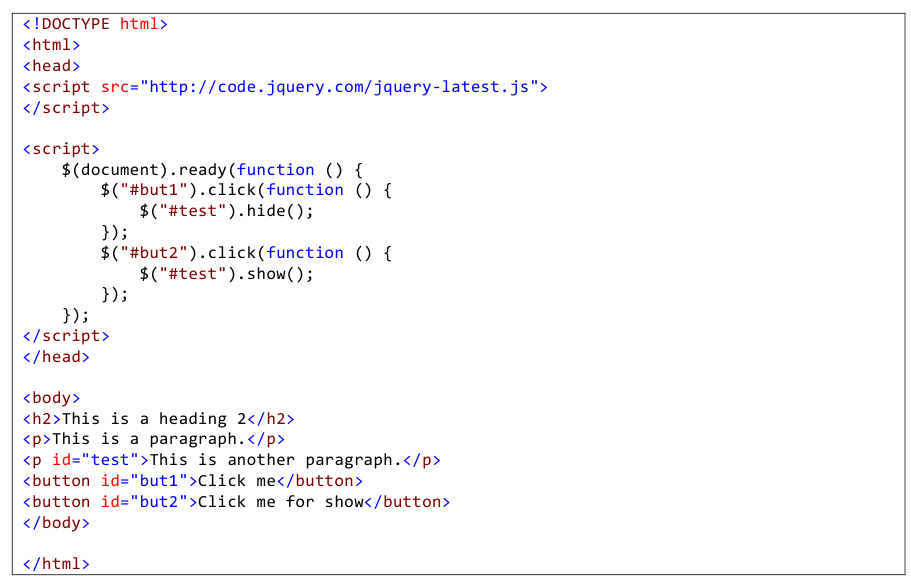


Рисунок 31.3 – Пример работы с элементом с помощью идентификаторов

Селектор jQuery class находит элементы определенного класса. Для поиска элементов определенного класса указывается перед названием точка, например: $(".test")

В таблице 31.1 представлены общие с технологией DOM события.

|  |  |  |  |
| --- | --- | --- | --- |
| **События мыши** | **События клавиатуры** | **События формы** | **События документа/окна** |
| Click | keypress | submit | load |
| dblclick | keydown | change | resize |
| mouseenter | keyup | focus | scroll |
| mouseleave |  | blur | unload |

Таблица 31.1 – Общие с технологией DOM события

В jQuery, большинство DOM-событий имеют эквивалентный jQuery-метод. Чтобы назначить событие нажатия мышкой на все элементы <p> на странице, вы можете написать:

$("p").click();

Следующий шаг – это определение того, что будет происходить, когда наступит указанное событие. Вы должны определить функцию для события:

$("p").click(function(){

// описание действий в функции

});

Часто используемые методы JQuery:

- $(document).ready() – метод позволяет выполнить функцию, когда документ полностью загружен;

- click( ) – функция выполняется, когда пользователь нажимает на HTML элемент;

- dblclick( ) – срабатывает, когда пользователь двойным щелчком нажимает на HTML-элемент;

- mouseenter() – выполняется, когда указатель мыши наводится на HTML-элемент;

- blur() – выполняется, когда поле формы теряет фокус;

- hide() и show() – позволяют скрывать и отражать HTML-элементы. Также возможно настроить время затухания и появления. Синтаксис:

$(selector).hide(speed);

$(selector).show(speed);

Необязательный параметр скорости определяет скорость скрытия / показа, и может принимать следующие значения: "slow", "fast" или в миллисекундах.

**5. Порядок выполнения работы**

1. Выполнить подключение библиотеки jQuery. Повторить пример из пункта №4, проверить работу метода hide ( ).

2. Выполнить задание в соответствии с вариантом.

Вариант 1. В исходный файл Experiments.html добавить в начало две кнопки: одну-для скрытия элементов, другую – для отображения скрытых элементов. Настроить методы click( ) кнопок, так чтобы они то скрывали, то отображали нечетные элементы с классом MsoNormal.

Вариант 2. В исходный файл Experiments.html добавить в начало две кнопки: одну-для скрытия элементов, другую – для отображения скрытых элементов. Настроить методы click( ) кнопок, так чтобы они то скрывали, то отображали четные элементы типа <tr>.

Вариант 3. В исходный файл Experiments.html добавить в начало две кнопки: одну-для скрытия элементов, другую – для отображения скрытых элементов. Настроить методы click( ) кнопок, так чтобы они то скрывали, то отображали элементы с атрибутом href.

Вариант 4. В исходный файл Experiments.html добавить в начало две кнопки: одну-для скрытия элементов, другую – для отображения скрытых элементов. Настроить методы click( ) кнопок, так чтобы они то скрывали, то отображали элементы, у которых атрибут align равен значению center.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Опишите структуру документа jQuery

2. Как подключается документ jQuery

3. Для чего предназначен символ $?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 32**

## **Тема работы: «Создание анимаций при помощи библиотеки jQuery»**

**1. Цель работы**

Формирование умений создания анимаций при помощи библиотеки jQuery.

**2. Задание**

Используя методы animate() создать HTML-документ, содержащий эффекты и анимации

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Эффекты, которых нет в библиотеке jQuery, можно создавать с помощью метода **.animate()**. Интерпретатор браузера динамически, без перезагрузки страницы, изменяет выбранные свойства на указанные значения. Анимация происходит для всех элементов обернутого набора. Чтобы добавить эффекты для конкретного элемента, нужно воспользоваться фильтрами jQuery для отбора.

Метод позволяет анимировать любое css-свойство, имеющее числовое значение, например, font-size, opacity, border-width, margin, padding, height, width, background-position и т.д. При этом имена свойств должны быть указаны слитно – fontSize, paddingLeft, или должен использоваться css-эквивалент свойства – "font-size". Числовые значения свойств не заключаются в кавычки.

Для любого свойства предварительно должно быть установлено начальное значение, а в css-объявлении должна использоваться полная запись каждого свойства, т.е., вместо свойства border должно быть заданы значения для border-style, border-width и т.д.

Функция обратного вызова вызывается один раз после завершения анимации. Функции не передается никаких аргументов, но анимации выполняется для элемента, переданного свойству this в качестве контекста.

Значениями свойств могут также выступать hide, show или toggle, в результате чего к элементу применится вычисляемое значение — отображение, скрытие или переключение исходных состояний свойств.

Метод .animate()позволяет изменять css-свойства выбранных элементов с возможностью одновременной анимации нескольких свойств, задавая продолжительность анимации в миллисекундах.

$("div").animate({left: "200px", top: "200px"}, 500);

Данная анимация одновременно применяется к свойству left, для которого задано значение 200px, и к свойству top со значением 200px, продолжительность анимации задана 500ms.

Для элемента можно задавать относительное перемещение при каждом вызове анимации с помощью операторов +=, -=, \*=, /=, например:

$("div").animate({left: "+=200", top: "-=200"}, 500);.

Метод .animate() имеет две формы записи. В первой методу передаются четыре аргумента:

1) .animate({свойство1: «значение1», свойство2: «значение2»}, продолжительность, функция перехода, function() {…});

2) {свойство1: «значение1»} – объект свойств, которые собираемся анимировать в формате свойство: «значение», перечисленные в фигурных скобках через запятую.

Продолжительность – необязательный параметр продолжительности анимации, задающий время в миллисекундах или с помощью ключевых слов slow, fast, normal.

Функция перехода – необязательное имя функции перехода.

function() {…} – необязательная функция обратного вызова.

Вторая форма принимает два аргумента: объект свойств и объект дополнительных функциональных возможностей.

Animate({свойство1: «значение1″, свойство2: значение2»}, {duration: «значение», easing: «значение», specialEasing: {

Свойство1: «easing1», свойство2: «easing2»}, complete: function() {…}, queue: true, step: callback});

{свойство1: «значение1»} — объект свойств, которые собираемся анимировать в формате свойство: «значение», перечисленные в фигурных скобках через запятую.

Duration – число или ключевое слово, значение по умолчанию 400. Устанавливает продолжительность анимации.

Easing – имя функции перехода. Значение по умолчанию swing колебательный переход, второе доступное значение – linear – линейный переход. Расширенные возможности реализуются с помощью плагинов, смотрите jQuery UI.

Queue – помещает анимацию в очередь эффектов, по умолчанию true, значение false – немедленно воспроизводит анимацию.

SpecialEasing объект, содержащий одно и более свойств и их значений, описывающих функции перехода. –Step – функция, которая будет вызываться по окончании каждого этапа анимации и будет применяться для каждого элемента обернутого набора. Функции передается порядковый номер этапа и внутренний объект, описывающий эффект.

Progress – функция, которая будет вызываться после каждого шага анимации, только один раз за анимацию независимо от количества анимационных свойств.

Complete – функция, которая должна быть вызвана по окончании воспроизведения анимации.

Start – функция, которая будет вызвана в начале анимации.

Done – функция, вызванная по завершении анимации.

Fail – функция, вызываемая в случае невозможности завершения анимации.

Always – функция, которая будет вызвана в случае остановки или неполного завершения анимации.

Метод .fadeIn().

Управляет прозрачностью, показывая скрытый элемент, при этом свойство opacity выбранного элемента изменяется от 0 до 1. Для этого на странице появляется необходимое пространство для элемента, при этом остальные элементы могут сдвинуться с места:

.fadeIn(длительность, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта с помощью ключевых слов "fast", "normal", "slow" или числовых значений. По умолчанию используется значение "normal", равное 400 миллисекундам.

Функция по завершении анимации – необязательный параметр, задает функцию, которая будет вызвана после проявления элемента.

.fadeIn(объект свойств).

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение»:

.fadeIn(длительность, функция перехода, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода. Значение по умолчанию swing – колебательный переход, второе доступное значение – linear – линейный переход.

Функция по завершении анимации – необязательный параметр, задает функцию, которая будет вызвана после проявления элемента.

Метод .fadeOut().

Заставляет элемент исчезнуть, сделав его прозрачным, сохраняя на странице место, занимаемое элементом. Свойство opacity выбранного элемента изменяется от 1 до 0.

.fadeOut(длительность, функция по завершении анимации).

Длительность – необязательный параметр, задает скорость проявления эффекта с помощью ключевых слов "fast", "normal", "slow" или числовых значений. Значение по умолчанию 400 миллисекунд.

Функция по завершении анимации – необязательный параметр, задает функцию, которая будет вызвана после проявления элемента.

.fadeOut(объект свойств).

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение»:

.fadeOut(длительность, функция перехода, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, задает функцию, которая будет вызвана после проявления элемента.

Метод .fadeTo().

Позволяет изменить степень прозрачности до заданного значения, например:

$("element").fadeTo("normal",.50);.

.fadeTo(длительность, прозрачность, функция по завершении анимации)

Длительность – задает скорость проявления эффекта.

Прозрачность – число от 0 до 1, задающее прозрачность элемента.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента:

.fadeTo(длительность, прозрачность, функция перехода, функция по завершении анимации)

Длительность – задает скорость проявления эффекта.

Прозрачность – число от 0 до 1, задающее прозрачность элемента.

Функция перехода — необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента.

Метод .fadeToggle(). Если элемент скрыт, то он появляется на экране, если виден, то исчезает.

.fadeToggle(длительность, функция перехода, функция по завершении анимации)

Длительность – задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента:

.fadeToggle(объект свойств)

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение».

Метод .hide() – скрывает видимый элемент. При установлении скорости элемент исчезает, как бы сужаясь. Чтобы замедлить действие, нужно передать значение продолжительности:

.hide()

Метод указывается без параметров:

.hide(длительность, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента:

.hide(объект свойств)

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение»:

.hide(длительность, функция перехода, функция по завершении анимации)

Длительность – задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента.

Метод .show() – показывает ранее скрытый элемент. Если не задано значение скорости, то элемент появляется моментально, если скорость задана, то элемент появляется от верхнего левого к нижнему левому углу:

.show()

Метод указывается без параметров:

.show(длительность, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента:

.show(объект свойств)

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение»:

.show(длительность, функция перехода, функция по завершении анимации)

Длительность – задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента.

Метод .toggle() – одиночный метод, переключает выбранный элемент из одного состояния в другое, в зависимости от его текущего состояния, скрывая или отображая его:

.toggle(длительность, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента:

.toggle(объект свойств)

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение»:

.toggle(длительность, функция перехода, функция по завершении анимации)

Длительность – задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента.

.toggle(display). Display – если параметр display установлен, то элемент будет показан.

Метод .slideDown() – заставляет скрытый элемент появиться на веб-странице. Элемент проявляется постепенно – сначала его верхняя часть, и по мере проявления остальной части то, что находилось под элементом, сдвигается вниз. Поэтому для того, чтобы контент не перемещался по странице, можно использовать абсолютное позиционирование элемента {position:absolute;}. Если необходимо разместить данный элемент относительно другого, то задайте относительное позиционирование {position:relative;} для элемента, который окружает абсолютно позиционированный элемент:

.slideDown(длительность, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента:

.slideDown(объект свойств)

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение»:

.slideDown(длительность, функция перехода, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента.

Метод .slideUp() – изменяет свойство height элемента, пока она не станет равной 0, после скрывает элемент display: none;. При этом удаление начинается снизу, и если для элемента не задано позиционирование, то контент, находившийся ниже, перемещается вверх:

.slideUp(длительность, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента:

.slideUp(объект свойств)

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение»:

.slideUp(длительность, функция перехода, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента.

Метод .slideToggle() – скрывает видимый элемент и показывает скрытый, т.е. может использоваться в качестве переключателя, позволяющего как отображать, так и скрывать элемент. При этом элемент проявляется постепенно, сверху вниз:

.slideToggle(длительность, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента:

.slideToggle(объект свойств)

Объект свойств – представляет собой пары дополнительных опций, указанных в формате свойство: «значение»:

.slideToggle(длительность, функция перехода, функция по завершении анимации)

Длительность – необязательный параметр, задает скорость проявления эффекта.

Функция перехода – необязательный параметр, задает функцию перехода.

Функция по завершении анимации – необязательный параметр, описывает функцию, которая будет вызвана после проявления элемента.

Анимация элемента реализуется только с помощью методов, создающих анимационные эффекты. Если анимация реализована цепочкой методов, каждый эффект выполняется последовательно, друг за другом. Например, $("div").slideDown().fadeOut(); сначала начнется скольжение блока вниз, а выцветание поставится в очередь "fx" и будет вызвано только при завершении скольжения.

Чтобы добавить пользовательские эффекты, создаются функции, которые также добавляются в очередь "fx". Очередь представляет массив функций, существующих на уровне элемента и хранящихся в jQuery.data. Каждый элемент может иметь одну или несколько очередей функции, но обычно используется только одна очередь по умолчанию "fx".

Функции включаются в очередь с помощью метода .queue(), без него функции выполняться не будут. Каждая функция по своему завершению должна вызвать метод .dequeue(), чтобы передать управление следующей функции в очереди.

$("div").slideUp();

$("div").queue(function() {

document.write("Hellow");

$(this).dequeue();

});

Метод .queue() – метод показывает текущую очередь анимации или манипулирует очередью функций, создающих анимационные эффекты, для элемента/элементов набора:

.queue(имяОчереди)

ИмяОчереди – строка, содержащая имя очереди. По умолчанию текущая очередь имеет зарезервированное имя "fx". Возвращает массив функций, находящихся в очереди:

.queue(имяОчереди, функция)

ИмяОчереди – строка, содержащая имя очереди.

Функция – функция обратного вызова, которая добавляется в конец очереди с именем Очереди для всех элементов обернутого набора:

.queue(имяОчереди, имяНовойОчереди)

ИмяОчереди – строка, содержащая имя очереди.

ИмяНовойОчереди – заменяет текущую очередь всех элементов набора на новуюОчередь, состоящую из массива функций.

Метод .dequeue() – ставит следующую анимацию в очередь. Это означает, что если выполняется анимация, когда этот метод вызывается, то новый анимация будет начата сразу после того, как текущая закончит свое выполнение. Выполняет первую функцию очереди для всех элементов набора, после выполнения функция удаляется из очереди. Обработчики события будут выполнять и удалять из очереди поочередно последующие функции. Если метод вызывается внутри функции, добавляемой в очередь, это вызовет поочередное исполнение всех функций, входящих в очередь:

.dequeue(имяОчереди)

ИмяОчереди – строка, содержащая имя очереди.

Метод .clearQueue() – удаляет все функции анимации из очереди, не только текущую, но и все последующие, без их выполнения. Может использоваться для удаления любых очередей:

.clearQueue(имяОчереди)

ИмяОчереди – строка, содержащая имя очереди.

Метод .delay() – позволяет установить отсрочку для запуска эффектов, ожидая определенное количество миллисекунд, прежде чем запустить следующий эффект в очереди:

.delay(число или строка, имяОчереди)

Число или строка – устанавливает продолжительность задержки в миллисекундах или с помощью ключевых слов – fast (эквивалентно 200ms) и slow (эквивалентно 600ms).

ИмяОчереди – необязательный параметр, указывает на очередь, для которой устанавливается задержка, по умолчанию — это очередь "fx".

Метод .stop() – останавливает текущую анимацию в очереди сразу после его запуска для элементов набора. Если при вызове метода передаются некоторые аргументы, то вы также можете очистить очередь и определить, должны ли элементы при остановке анимации остаться на месте, или они должны вернуться к первоначальному состоянию. По умолчанию оба параметра имеют значение false:

.stop(логическое значение1, логическое значение2)

Логическое значение1 – значение true останавливает все анимационные эффекты, находящиеся в очереди.

Логическое значение2 – значение true останавливает все эффекты, кроме текущего.

Метод .finish() – останавливает текущую анимацию, удаляет все очереди анимации и завершает все анимации для соответствующих элементов:

.finish(имяОчереди)

ИмяОчереди – задает имя очереди анимации, которую нужно остановить. По умолчанию – очередь "fx".

**5. Порядок выполнения работы**

1. Создайте несколько параграфов с элементами strong и em. Задайте обработчики событий наведения мышью на эти элементы: при наведении, цвет элементов должен отличаться от обычного, а размер плавно увеличиваться.

2 Создать метод у объекта jQuery — parabola(), который принимает аргументы a,b и c и анимирует элементы по параболе.

3. При наведении на элемент с идентификатором #box, поворачивать элемент с классом .some на 90 градусов, при событии выведения мыши из области элемента возвращать .some в исходную позицию.

4. Назначить обработчики событий для кнопок, такие, чтобы после нажатия на кнопку элемент должен двигаться на 100px в направлении, указанном кнопкой.

5. При нажатии на любую точку страницы, перемещать с анимацией этот элемент к нужному месту.

6. Создать эффект переключателей с использованием библиотеки jQuery

7. Для изображений создать обработчики событий, чтобы работа скрипта была похожа на работу слайдера.

8. Создать эффект лайтбокса средствами jQuery.

9. Создать эффект карусели.

10. Создать эффект галереи.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Опишите способ создания собственных эффектов с помощью метода .animate().

2. Какие анимационные эффекты jQuery вы знаете?

3. Дайте характеристику метода.slideDown()

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 33**

## **Тема работы: «Использование виджетов jQuery UI»**

**1. Цель работы**

Формирование умений применения и модификации виджетов библиотеки jQuery UI.

**2. Задание**

Создать HTML-документ, содержащий сценарии с использованием виджетов jQuery UI.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

jQuery UI предоставляет набор готовых виджетов предназначенных для создания пользовательского интерфейса веб-приложений.

Поведение виджетов может настраиваться с помощью передаваемых им опций.

Внешний вид виджетов может быть настроен:

- с помощью выбора одной из стандартных тем доступных при скачивании библиотеки;

- с помощью themeroller'a;

- вручную путем редактирования файла jquery-ui-1.8.12.custom.css.

Общий вид создания виджетов jQuery UI выглядит примерно следующим образом:

- группировка элементов на странице особым образом (индивидуально для каждого виджета);

- применение к сгруппированным элементам специального метода, который превращает их в виджет.

Виджет accordion представляет собой группу объединенных выпадающих меню из которых только одно может быть отрыто одновременно.

Данный виджет используется для группировки информации на странице с целью экономии места.

Синтаксис:

/\* 1. Группируем элементы: \*/

<div id="accordion">

// 1.1. Определяем заголовок элемента

<h2><a href="#head1">Заголовок 1</a></h2>

/\* 1.2. Определяем содержимое элемента (значение атрибута id содержимого должно совпадать со значением href заголовка) \*/

<div id="head1">Содержимое 1</div>

</div>

// 2. Применяем метод к сгруппированным элементам:

$("#accordion").accordion({опция1:значение1,опцияN:значениеN});

Обратите внимание: с помощью опций, передающихся методу, Вы можете настраивать дополнительные аспекты поведения виджетов. Виджетам может быть передано сразу несколько опций:

$(document).ready(function() {

$("#accordion").accordion();

});

Виджет autocomplete позволяет ускорить ввод данных в поле, отображая по мере введения определенные предположения. Выбрав одно из предположений, пользователь может автоматически завершить ввод.

Предположения выводится в случае, если данные введенные пользователем совпадают со значением одного из элементов из списка данных.

Вы можете подключать к виджету как локальные (т.е. определенные в скрипте на этой же странице) так и удаленные списки (т.е. находящиеся на удаленном сервере).

Локальные списки удобны для хранения небольших наборов данных (например список улиц города), а удаленные списки подходят для хранения больших наборов данных (например база данных всех городов мира).

Синтаксис:

/\* 1. Определяем элемент input, в который будет производиться ввод информации \*/

<input id='auto' />

/\* 2. Превращаем input в виджет autocomplete \*/

$('#auto').autocomplete({опция1:значение1,опцияN:значениеN})

Подключить к виджету список данных можно с помощью опции source.

С помощью опции minLength Вы можете указать минимальное количество символов, которое должно содержать поле ввода прежде, чем поиск совпадений начнет выполнятся:

$(document).ready(function() {

$("#auto1").autocomplete({source:["Дмитрий","Мария","Владимир","Алексей","Екатерина","Олег","Ольга"]});

});

Виджет datepicker привязывает к текстовому полю интерактивный календарь, который отображается, когда поле становится активным.

Если пользователь щелкнет на какую-либо дату в календаре, она будет автоматически введена в текстовое поле, как будто он ввел ее вручную.

Синтаксис:

// 1. Создадим текстовое поле

<input type='text' id='datepicker' />

// 2. Привяжем к нему виджет datepicker

$('#datepicker').datepicker({опция1:значение1, опцияN:значениеN});

Пример:

$(document).ready(function() {

$("#datepicker").datepicker();

$("#datepicker1").datepicker({

monthNames:["Январь","Февраль","Март","Апрель","Май","Июнь","Июль","Август","Сентябрь","Октябрь","Ноябрь","Декабрь"],

dayNamesMin:["Вс","Пн","Вт","Ср","Чт","Пт","Сб"],

firstDay:1,

dateFormat:"dd.mm.yy"

});

});

С помощью метода button Вы можете стилизовать (оформлять):

- обычные кнопки (определяемые тэгами button и input type='button');

- кнопки отправления форм;

- радио кнопки;

- флажки;

- ссылки;

Синтаксис:

/\* Если Вы хотите оформить только один элемент, необходимо выбрать его с помощью

селектора и вызвать метод button: \*/

$("селектор").button({опция1:значение1, опцияN:значениеN});

/\* Если Вы оформляете группу элементов, необходимо вначале сгруппировать элементы следующим образом: \*/

<div id="groupradio">

<input type="radio" name="radio" id="value1" />

<label for="value1">Радио кнопка 1</label>

<input type="radio" name="radio" id="value2" />

<label for="value2">Радио кнопка 2</label>

<input type="radio" name="radio" id="value3" />

<label for="value3">Радио кнопка 3</label>

</div>

// И затем вызвать метод buttonset:

$("#groupradio").buttonset({опция1:значение1, опцияN:значениеN});

Пример:

$(document).ready(function() {

$("#but1,#but2,#but3,#but4").button();

$("#group1,#group2").buttonset();

});

**5. Порядок выполнения работы**

1. Исправьте ошибки в группировке элементов в коде приведенном ниже, для того, чтобы виджет accordion был отображен корректно.

$(document).ready(function(){

$("#accordion1").accordion();

});

2. Создайте на основе приведенных в задании данных виджет accordion.

3. Создайте виджет autocomplete, который будет производить поиск по списку приведенных в задании стран. Сделайте так, чтобы виджет начинал поиск только если длина ввода пользователя превышает 3 символа.

4. Оформите как кнопки элементы с id=el1, id=el2, id=el3

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Что представляет собой jQuery UI?

2. Опишите основной принцип работы с jQuery UI.

3. Для чего предназначен виджет accordion?

4. Для чего предназначен виджет autocomplete?

5. Для чего предназначен виджет datepicker?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.

# **Лабораторная работа № 34**

## **Тема работы: «Проверка данных форм при помощи библиотеки jQuery»**

**1. Цель работы**

Формирование умений обрабатывать данные форм при помощи библиотеки jQuery.

**2. Задание**

Создать HTML-документ, содержащий сценарии обработки формы при помощи библиотеки jQuery.

**3. Оснащение работы**

ПК, текстовый редактор, браузер.

**4. Основные теоретические сведения**

Валидация форм и проверка их полей на соответствие определенному формату и наличие определенных данных – это фактически стандартная задача при работе с формами на сайте. Решить ее можно различными способами и средствами.

Можно воспользоваться новыми возможностями HTML5, которые позволяют задавать тип полям и даже использовать регулярные выражения для проверки введенных данных. Можно использовать один из многочисленных плагинов для валидации форм. Можно написать собственный скрипт для валидации.

Рассмотрим возможности плагина jQuery Validation и поработаем со следующей формой, представленной на рисунке 34.1.

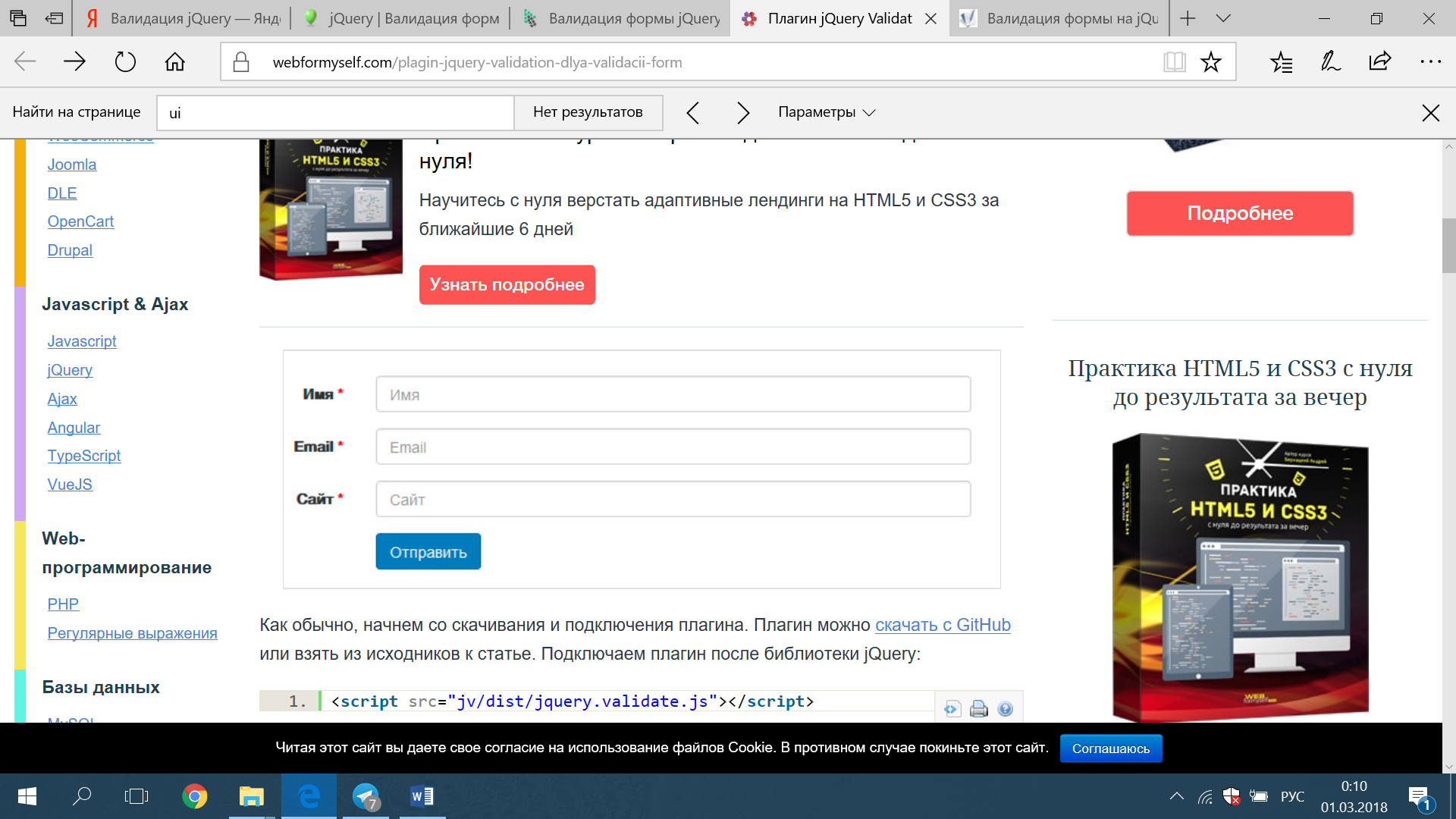


Рисунок 34.1 – Форма HTML

Как обычно, начнем со скачивания и подключения плагина. Подключаем плагин после библиотеки jQuery:

<script src="jv/dist/jquery.validate.js"></script>

Осталось лишь инициализировать плагин, чтобы он начал работать. Для этого достаточно вызвать метод validate() для формы, которую необходимо проверить:

$(function(){

$('#form').validate();

});

Для плагина достаточно атрибута required, чтобы он не пропускал пустое поле. При этом плагин также отлично понимает типы полей. Например, если в поле email мы введем строку, не соответствующую формату email, плагин уведомит нас, причем в режиме онлайн.

Осталось лишь немного настроить jQuery Validation, чтобы получилось законченное решение. Неплохо было бы видеть переведенные сообщения об ошибках или даже текст ошибок.

Для решения первого вопроса передадим в метод validate() объект messages с текстом ошибки. Добавим необходимые сообщения:

$(function(){

$('#form').validate({

rules: {

name: {

required: true,

minlength: 2

}

},

messages: {

name: {

required: "Поле 'Имя' обязательно к заполнению",

minlength: "Введите не менее 2-х символов в поле 'Имя'"

},

email: {

required: "Поле 'Email' обязательно к заполнению",

email: "Необходим формат адреса email"

},

url: "Поле 'Сайт' обязательно к заполнению"

}

});

});

Обратите внимание, для каждого варианта валидации можно добавить свое сообщение. Также мы использовали объект rules, позволяющий добавить правила валидации для того или иного поля.

**5. Порядок выполнения работы**

1. Выполнить валидацию формы с помощью JQuery в соответствии с вариантом.

Вариант 1. Форма для обмена бытовой техники по гарантии.

Вариант 2. Форма сотрудника библиотеки.

Вариант 3. Форма сотрудника ГАИ.

Вариант 4. Форма сотрудника ботанического сада.

Вариант 5. Форма сотрудника ж/д станции.

Вариант 6. Форма сотрудника автовокзала.

Вариант 7. Форма сотрудника отдела кадров.

Вариант 8. Форма менеджера автосалона.

Вариант 9. Форма по начислению студентам стипендии.

Вариант 10. Форма сотрудника туристической фирмы.

**6. Форма отчета о работе**

*Лабораторная работа № \_\_\_*

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

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

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

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

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

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

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

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

**7. Контрольные вопросы**

1. Опишите процесс подключения и работы плагина validate.

2. Каким образом в плагигин validate можно добавить свои сообщения об ошибках?

**8. Рекомендуемая литература**

1. **Макфарланд, Д.** Новая большая книга CSS / Дэвид Макфарланд. – СПб.: Питер, 2016. – 720с.
2. **Никсон, Р.** Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 /Р. Никсон. – 4-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 768 с.
3. **Прохоренок, Н.А**. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А.Прохоренок. – СПб.: БХВ-Петербург, 2010. – 912с.
4. **Робсон, Э.** Изучаем HTML, XHTML и CSS / **Э. Робсон**. – 2-е изд. – СПб.: ООО «ПИТЕР М», 2017. – 720 с.
5. **Фрейн, Б.** HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Бен Фрейн. – СПб.: Питер Пресс, 2017. — 272с.