

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Н. А. Жилияк, Е. В. Барковский

КОМПЬЮТЕРНЫЕ ЯЗЫКИ РАЗМЕТКИ ЛАБОРАТОРНЫЙ ПРАКТИКУМ

*Рекомендовано
учебно-методическим объединением по образованию
в области информатики и радиоэлектроники
в качестве учебно-методического пособия
для студентов учреждений высшего образования
по специальностям 6-05-0611-01 «Информационные системы
и технологии», 6-05-0612-01 «Программная инженерия»*

Минск 2023

УДК 004.439(076.5)(075.8)
ББК 32.973-01я73
Ж72

Рецензенты:

кафедра информационных технологий автоматизированных систем
Белорусского государственного университета
информатики и радиоэлектроники
(кандидат физико-математических наук, доцент, заведующий
кафедрой, , доцент *А. А. Навроцкий*);
кандидат физико-математических наук, доцент, доцент кафедры
управления информационными ресурсами Академии управления
при Президенте Республики Беларусь *В.К. Шешолко*

Все права на данное издание защищены. Воспроизведение всей книги или ее части не может быть осуществлено без разрешения учреждения образования «Белорусский государственный технологический университет».

Жиляк, Н. А.

Ж72 Компьютерные языки разметки. Лабораторный практикум :
учеб.-метод. пособие для студентов специальностей 6-05-
0611-01 «Информационные системы и технологии», 6-05-
0612-01 «Программная инженерия» / Н. А. Жиляк,
Е. В. Барковский. – Минск : БГТУ, 2023. – 114 с.
ISBN 978-985-897-109-0

Лабораторный практикум предназначен для приобретения практических навыков разработки структуры веб-сайтов в соответствии с учебной программой дисциплины «Компьютерные языки разметки». Каждая лабораторная работа сопровождается методическими указаниями по языкам разметки HTML5 и XML, таблицам стилей CSS3, препроцессору SCSS, масштабируемой векторной графике (SVG) и языке программирования JavaScript.

Кроме того, в пособие включены контрольные вопросы для текущей и итоговой аттестации и список литературы для самостоятельного изучения дисциплины.

УДК 004.439(076.5)(075.8)
ББК 32.973-01я73

ISBN 978-985-897-109-0 © УО «Белорусский государственный
технологический университет», 2023
© Жиляк Н. А., Барковский Е. В., 2023

ПРЕДИСЛОВИЕ

Лабораторный практикум по дисциплине «Компьютерные языки разметки» предназначен для закрепления теоретических знаний и практических навыков студентами специальностей 6-05-0611-01 «Информационные системы и технологии», 6-05-0612-01 «Программная инженерия». Целью дисциплины является формирование профессиональных компетенций по разработке, внедрению, адаптации и использованию информационных технологий, связанных с использованием языков веб-программирования.

Предлагаемый лабораторный практикум включает 15 лабораторных работ. Тематика работ соответствует практическому применению современных методов и средств для разработки структуры веб-сайта.

Лабораторные работы № 1–7 посвящены использованию языка HTML5 и таблиц стилей CSS3. В лабораторных работах № 8–10 приводятся основы создания навигационной панели, реализация адаптивной верстки веб-сайта с помощью свойств grid- и flex-верстки. В лабораторной работе № 11 рассматриваются функциональные возможности препроцессора SCSS для создания стилей CSS3.

В лабораторных работах № 12 и 13 уделяется отдельное внимание основам создания XML-документов с использованием расширения XSLT, а в работе № 14 приводятся практические навыки применения масштабируемой векторной графики (SVG).

Для управления элементами веб-сайта используется язык программирования JavaScript, которому посвящена лабораторная работа № 15.

К числу обязательных условий выполнения лабораторных работ относятся:

- 1) предварительное ознакомление студента с теоретическими сведениями, приведенными в конспекте лекций, а также рекомендуемой в издании литературой;
- 2) выполнение работы в соответствии с пошаговыми инструкциями и рекомендациями преподавателя;
- 3) защита лабораторных работ, которая включает устные вопросы на контрольные вопросы.

Лабораторная работа № 1

ОСНОВНЫЕ ТЕГИ HTML5

Цель работы: изучить структуру HTML-документа и использование основных тегов и атрибутов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

ОСНОВЫ HTML5

HTML (Hyper Text Markup Language) – гипертекстовый язык разметки. Для разметки HTML-документа используют теги (флаги разметки).

Тег – это определенная последовательность символов, заключенных между знаками < (больше) и > (меньше).

Для того, чтобы создать HTML-документ, необходимо:

- открыть любой текстовый редактор (например, Visual Studio Code);
- набрать произвольный текст и разметить его HTML-тегами;
- сохранить файл с расширением .html.

Теперь если открыть созданный файл с помощью веб-браузера, он будет отображен как веб-страница.

Структура веб-страницы ограничена тегами **<html>** и **</html>** и разбивается на две части: заголовок и тело. В заголовке указывается служебная информация обо всей веб-странице, а в теле описывается ее содержимое вместе с правилами оформления. Заголовок веб-страницы ограничивается тегами **<head>** и **</head>**, а тело документа обозначается тегами **<body>** и **</body>**.

В заголовок документа могут входить: тег, отображающий наименование веб-страницы, тег стилевого оформления, тег выполняемого сценария и так называемые метаданные.

Между тегами **<title>** и **</title>** указывается название страницы, которое отображается в строке заголовка окна при просмотре странички в браузере. Для внедрения метаданных в веб-страницу применяется тег **<meta>**. Тегов **<meta>** может быть несколько.

Перед тегом **<html>** ставится идентификатор применяемого стандарта HTML для совместимости отображения веб-страницы в браузере, например, для версии HTML5 **<!DOCTYPE html>**. Структура любой веб-страницы представлена на рис. 1.1.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body> Тело документа </body>
</html>
```

Рис. 1.1. Структура веб-страницы

В теле документа заключается гипертекст, который определяет собственно веб-страницу. Гипертекстом является произвольная часть документа, которую разрабатывает автор и которая позволяет устанавливать смысловые связи между элементами текста на экране компьютера таким образом, чтобы можно было легко осуществлять переходы от одного элемента к другому. Внутри элемента **<body>** можно использовать все элементы, предназначенные для дизайна веб-страницы. Внутри начального элемента **<body>** могут располагаться атрибуты, обеспечивающие установки параметров стилизации для всей веб-страницы в целом. Синтаксис использования атрибутов представлен на рис. 1.2.

```
<body атрибут_1="значение1"
      атрибут_2="значение2"...>
```

Рис. 1.2. Синтаксис использования атрибутов HTML

При создании веб-сайта необходимо создавать несколько веб-страниц. Для того чтобы работать с большим объемом кода HTML-документа, используются комментарии, которые создаются следующим образом: **<!-- текст комментария -->**. Текст комментария не отображается в браузере.

У тега **<body>** могут быть следующие атрибуты:

- 1) ***marginheight*** – определяет ширину (в пикселах) верхнего и нижнего полей документа. Работает только в браузерах Netscape;
- 2) ***topmargin*** – задает ширину (в пикселах) верхнего и нижнего полей документа. Работает только в браузерах Internet Explorer;
- 3) ***marginwidth*** – определяет ширину (в пикселах) левого и правого полей документа. Работает только в браузерах Netscape;
- 4) ***leftmargin*** – задает ширину (в пикселах) левого и правого полей документа. Работает только в браузерах Internet Explorer;
- 5) ***background*** – определяет изображение для "заливки" фона. Значение задается в виде полного URL или имени файла с картинкой в формате gif или jpg;
- 6) ***bgcolor*** – определяет цвет фона документа.
- 7) ***text*** – задает цвет текста в документе.
- 8) ***link*** – определяет цвет гиперссылок в документе;
- 9) ***alink*** – задает цвет подсветки гиперссылок в момент нажатия;
- 10) ***vlink*** – определяет цвет гиперссылок на документы, которые вы уже просмотрели.

Шрифт задается тегом ****, который имеет следующие атрибуты:

- ***face*** – имена шрифтов, разделенные запятыми;
- ***size*** – размер от 1 до 7 (по умолчанию 3);
- ***color*** – цвет шрифта.

В HTML существуют специальные теги для заголовков: от **<h1>** (самого крупного) до **<h6>** (самого мелкого).

Каждая HTML-страница имеет свой уникальный адрес в Интернете, который называется универсальным указателем ресурса (URL). Для перехода на веб-страницу служит гиперссылка. За организацию гиперссылок в языке HTML отвечает тег **<a>...**, который чаще всего использует следующий шаблон, представленный на рис. 1.3.

<code> текст для щелчка </code>
--

Рис. 1.3. Создание гиперссылок

Также можно создавать внутренние ссылки, позволяющие переходить между элементами документа. Сначала в нужных местах устанавливается метка с помощью атрибута ***name*** (рис. 1.4).

```
<a name="metka">...</a>  
или  
<a name="http://адрес/файл.html#метка">...</a>
```

Рисунок 1.4 Создание меток внутренних гиперссылок

Затем определяется гиперссылка на метку (рис. 1.5).

```
<a href="#metka">текст для щелчка </a>
```

Рис. 1.5. Создание внутренних гиперссылок

Для хранения изображений используются десятки различных форматов, например, gif, jpeg, bmp, psx, vmf. Однако для работы с изображениями в документах HTML обычно выбирают форматы gif, jpeg, так как они распознаются браузерами. Для отображения других форматов необходимо устанавливать плагины или запускать Java-апплеты.

Для вставки изображения используется тег **** с атрибутом src, который указывает URL графического файла: ****.

При организации ссылки, в качестве которой используется изображение, применяется шаблон из комбинации двух тегов, (рис. 1.6).

```
<a href="Адрес ссылки"></a>
```

Рис. 1.6. Создание гиперссылки из изображения

Теги списков

Список отличается от обычного текста нумерацией его пунктов. Если список дополняется новыми пунктами или укорачивается, нумерация корректируется автоматически. Различают маркированный, нумерованный списки, списки с определениями.

В теге маркированного списка для обозначения маркера можно применить атрибут **type**, принимающий значения *disc* (круг), *circle* или *round* (окружность), *square* (квадрат). В качестве маркера можно использовать графические изображения. После указания **** определяется тег ****.

Структура маркированного или нумерованного списка представлена на рис. 1.7.

```
<ul>
  <li> элемент 1</li>
  <li> элемент 2</li>
  <li> элемент 3</li>
</ul>
```

Рис. 1.7. Структура маркированного списка

Структура нумерованного списка показана на рис. 1.8.

```
<ol>
  <li> элемент 1</li>
  <li> элемент 2</li>
  <li> элемент 3</li>
</ol>
```

Рис. 1.8. Структура нумерованного списка

Каждый пункт маркируется элементом упорядоченной последовательности: арабскими или римскими цифрами, буквами латинского алфавита. Способ нумерации задается при помощи атрибута ***type*** тега ****, который может принимать следующие значения:

- 1) ***type*** = "1" – 1,2,3,4;
- 2) ***type*** = "i" – i, ii, iii, iv;
- 3) ***type*** = "I" – I, II, III, IV;
- 4) ***type*** = "a" – a, b, c, d;
- 5) ***type*** = "A" – A, B, C, D.

Можно формировать также вложенные списки, структура которых приведена на рис. 1.9.

```
<ol>
<li>Пункт 1</li>
<li>Пункт 2
  <ul>
    <li>Подпункт 2.1.</li>
    <li>Подпункт 2.2.</li>
  </ul>
</li>
<li>Пункт 3</li>
</ol>
```

Рис. 1.9 Структура вложенного списка

Теги таблиц








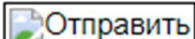
Теги `<table>` и `</table>` служат контейнером для элементов, определяющих содержимое таблицы. Таблица состоит из строк и ячеек, которые задаются с помощью тегов `<tr>` и `<td>`. Для отображения границ таблицы используется атрибут ***border***. С помощью тега `<th>` можно создать заголовки в головке таблицы. Текст элемента `<th>` центрируется и выделяется жирным шрифтом. С помощью атрибута ***colspan*** можно указать, на сколько столбцов должна быть растянута указанная ячейка, а с помощью атрибута ***rowspan*** – на сколько строк должна быть растянута указанная ячейка. Теги для создания ячеек и строк таблицы имеют соответствующие им закрывающие теги.






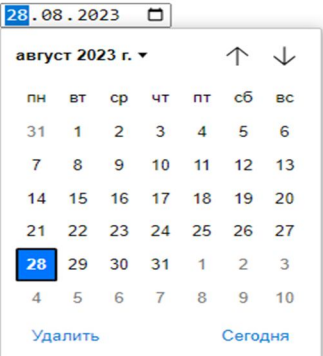
Теги создания веб-форм

Форма ограничивается тегами `<form>` и `</form>`. Между этими тегами обычно располагаются теги, создающие элементы управления формы. При необходимости, элементы управления могут размещаться в ячейках таблицы, которая полностью располагается в форме.

Большинство элементов управления отображаются при помощи тега `<input />`. Конкретный вид элемента управления зависит от атрибута ***type*** (`<input type="xxx" />`). Значения атрибута представлены в таблице.

Значения атрибута ***type***

Значение атрибута	Функциональное назначение	Внешний вид
<i>text</i>	Обычное текстовое поле	
<i>password</i>	«Маскированное» текстовое поле	
<i>checkbox</i>	Флажок	
<i>radio</i>	Переключатель	
<i>reset</i>	Кнопка для очистки полей формы	
<i>time</i>	Поле для ввода времени	
<i>button</i>	Обычная кнопка (ее действие можно будет задать позднее)	
<i>image</i>	Изображение-кнопка, определяющая координаты нажатия	

Значение атрибута	Функциональное назначение	Внешний вид
<i>file</i>	Кнопка выбора файла для присоединения к форме	
<i>search</i>	Поле для поискового запроса, имеющее кнопку очистки и подсказки из истории поиска в браузере.	
<i>email, url, tel</i>	Поля, которые имеют вид обычного текстового поля, однако они используются при обработке браузерами мобильных устройств.	
<i>range</i>	Ползунок, позволяющий указать числовое значение	
<i>number</i>	Текстовое поле с кнопками увеличения/уменьшения числового значения	
<i>date</i>	Поле для ввода даты	

Атрибут ***value*** используется для определения начального текста в текстовых полях или для надписи на кнопках. Атрибут ***checked*** позволяет отметить флажок или переключатель по умолчанию. Атрибут ***size*** описывает длину поля ввода. Атрибут ***disabled*** позволяет «деактивировать» элемент управления.

Также существуют атрибуты, поддерживаемые еще не всеми браузерами, но достаточно удобные: атрибут ***placeholder*** позволяет задать отображаемый текст, если текстовое поле не заполнено; атрибут ***autofocus*** дает возможность установить фокус ввода на элемент после загрузки страницы; атрибут ***required*** позволяет пометить поле как обязательное для заполнения. Чтобы из группы переключателей выбирался только один, необходимо всем элементам из группы задать одинаковый атрибут ***name***.

Для создания выпадающих списков используется тег `<select>...</select>`. Внутрь него вкладываются теги `<option>...</option>` с возможными вариантами. Тег `<option>` также имеет атрибуты *checked* и *disabled*.

Для создания многострочного текстового поля используется парный тег `<textarea>...</textarea>`, который поддерживает атрибуты *rows* и *cols*, задающие количество строк и столбцов в символах.

Тег `<legend>` применяется для создания заголовка группы элементов формы, которая определяется с помощью тега `<fieldset>`. Группа элементов обозначается в браузере с помощью рамки, а текст, который располагается внутри контейнера `<legend>`, встраивается в эту рамку.

Предназначение тегов `<div>` и ``

Тег `<div>` является блочным элементом и предназначен для выделения фрагмента документа с целью изменения вида содержимого. Как правило, вид блока управляется с помощью стилей.

Тег `` предназначен для определения строчных элементов документа. В отличие от блочных элементов, таких как `<table>`, `<p>` или `<div>`, с помощью тега `` можно выделить часть информации внутри других тегов и установить для нее свой стиль. Например, внутри абзаца (тега `<p>`) можно изменить цвет и размер первой буквы, если добавить начальный и конечный тег `` и определить для него стиль текста.

Семантические теги

В стандарт HTML5 были введены семантические теги, с помощью которых можно сделать страницы веб-сайтов более понятными для поисковых систем и браузеров. Следует отметить, что есть несколько семантических тегов, которые рекомендуется использовать для разметки страниц вместо `<div>` и ``:

- тег `<header>` – заголовочный блок веб-сайта, который обычно содержит навигацию, повторяется на всех страницах веб-сайта;

- тег `<footer>` – заключительная часть смыслового раздела или всего сайта, которая содержит информацию об авторах, список литературы, копирайт;

- тег **<nav>** – навигационное меню;
- тег **<main>** – основное, не повторяющееся на других страницах, содержание страницы;
- тег **<section>** – позволяет группировать логически связанное содержимое в документе, может применяться для блока новостей, контактной информации, глав текста, вкладок в диалоговом окне;
- тег **<article>** – это может быть пост на форуме, статья в журнале или газете, заметка в блоге, сообщение пользователя или другая независимая контент-единица;
- тег **<mark>** – позволяет выделить (подсветить) важную часть в тексте.

В HTML5 можно создавать подписи для иллюстраций с помощью тегов **<figure>** и **<figcaption>** (рис. 1.10).

```
<figure>
  <img src='foto.jpg' width='300' height='230' />
  <figcaption>Моя          замечательная          фотография
</figcaption>
</figure>
```

Рис. 10.1. Создание подписи к рисункам тегом **<figcaption>**

ЗАДАНИЯ

Задание 1. Создать документ `index.html` с произвольным текстом одной тематики со следующим содержимым:

1. В созданном документе должен быть заголовок в центре страницы (тег **<h1>**) черного цвета, произвольной гарнитуры и начертания. Текст заголовка должен совпадать с названием веб-страницы. После заголовка создать заголовок тегом **<h2>** и три абзаца текста. Каждый абзац составляет более 4–5 строк. Текст в абзацах должен быть черного цвета, прямого начертания, произвольной гарнитуры. Для каждого абзаца придумать заголовок (тег **<h3>**) Внутри каждого абзаца выделить несколько слов элементами **** произвольной гарнитуры шрифта, размером, цветом.

2. В этом же документе после текста создать список согласно рис. 1.11, который по содержанию должен соответствовать выбранной тематике.

Перед списком вставить заголовок, отражающий его содержание (тег **<h2>**).

1. Пункт первый
• Подпункт 1.1
• Подпункт 1.2
2. Пункт 2
▪ Подпункт 2.1
▪ Подпункт 2.2
▪ Подпункт 2.3
3. Пункт 3
○ Подпункт 3.1
○ Подпункт 3.2

Рис. 1.11. Список к заданию

3. Таблица, содержащая произвольный текст выбранной тематики с числовой информацией. Перед таблицей вставить заголовок, который создать с помощью тега `<caption>...</caption>` внутри тега `<table>`. Форма таблицы может быть произвольная, но обязательно должны быть объединенные ячейки строк и столбцов. Пример таблиц представлен на рис. 1.12.

Цена товара

Производитель	Товар	Цена товара	
		с НДС	без НДС
Erich Krause	Карандаш	30	40
	Ластик	20	30
	Ручка гелевая	10	20
Итого		60	90

Рис. 1.12. Пример таблицы к заданию 1

Задание 2. Создать HTML-документ `second.html` с текстовой информацией по тематике, связанной с первой веб-страницей в следующем порядке:

1. Создать два элемента `<div>` с произвольным текстом. Перед каждым элементом добавить заголовок (тег `<h3>`).

2. Вставить рисунок, перед которым добавить заголовок (тег `<h3>`).

3. Создать элемент ``, содержащий авторский знак © и ФИО автора

Задание 3. Элемент текста на первой веб-странице и в таблице сделать гиперссылкой на вторую веб-страницу, а рисунок на второй веб-странице – гиперссылкой на первую.

Задание 4. Создать новый документ с веб-формой, согласно рис. 1.13.



Данные студента

Имя студента

Отчество студента

Фамилия студента

Адрес почты

Контактный номер

Какую социальную сеть вы используете? ☐ VK ☐ Facebook ☐ Instagram

Рисунок 1.13 Форма для задания 4

Задание 5. Создать четвертый документ, используя семантические теги. В документе должны быть верхние (**<header>**) и нижние (**<footer>**) колонтитулы; кроме того, информация должна быть разбита на три секции (**<section>**) внутри тега **<main>**, а также должны быть использованы теги **<aside>**, **<figure>**, **<nav>**.

Тег **<footer>** должен содержать данные автора (ФИО, курс, группа, телефон и email).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как создать простейшую веб-страницу?
2. Что представляет собой структура HTML-документа?
3. Дайте определение понятиям «тег» и «элемент» HTML-документа?
4. Какие теги относятся к служебным, а какие – к структурным?
5. В каком разделе страницы указывается заголовок веб-страницы?
6. Для чего используются метаданные? Каким тегом они указываются в структуре HTML-документа?

7. Для чего используется **!DOCTYPE**?
8. Каким образом можно изменить установки для всей страницы в целом?
9. Перечислите виды списков, которые предусмотрены в HTML?
10. Как изменить вид маркера в маркированном списке?
11. Поясните, как изменить вид маркера в нумерованном списке.
12. Каким образом изменить нумерацию в нумерованном списке?
13. Из каких частей состоит список с определениями?
14. Каким образом можно построить вложенный список?
15. Создайте вложенный список из нумерованного, состоящего из двух пунктов, в каждом из которых два маркированных подпункта. Символ маркера квадрат.
16. С помощью каких тегов создается простая таблица?
17. Для чего предназначен тег **<th>**?
18. Какие элементы определяют строку таблицы?
19. Что такое гипертекст?
20. Как организовать гиперссылку?
21. Каким образом организовать переход в начало или конец документа?
22. Создайте таблицу с гиперссылкой на другую веб-страницу.
23. Каким образом организовать гиперссылку, используя графическое изображение?
24. Какие теги называются семантическими? Назовите основные семантические теги.
25. Для чего необходимы теги **<section>** и **<article>**?
26. Какую функцию выполняют теги **<footer>** и **<header>**?
27. Для чего предназначены теги **<nav>** и **<main>**?
28. Какие элементы необходимы для построения формы?
29. Назовите атрибут тега **<input>**, который определяет вид элемента формы.
30. Какие значения принимает атрибут **type**?

Лабораторная работа № 2

ОСНОВЫ CSS

Цель работы: изучить основные способы подключения CSS, типы селекторов, каскадность и наследование стилей.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Основы синтаксиса CSS

CSS (Cascading Style Sheets) – каскадные таблицы стилей, которые предназначены для придания HTML-документам внешнего вида.

Синтаксис CSS, представленный на рис. 2.1, состоит из селектора и блока объявлений. Блок объявлений включает свойство и значение. Следует отметить, что после значения свойства необходимо ставить точку с запятой. Пропуская точку с запятой, таблица стилей будет нарушена и веб-страница отобразится некорректно.

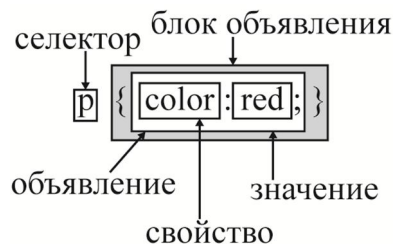


Рис. 2.1. Синтаксис CSS

В качестве базовых селекторов используются теги, классы и идентификаторы, но существуют другие типы. Примеры создания различных селекторов представлены в таблице.

Типы селекторов

Тип селектора	Пример создания	Использование
Селектор тега	<pre>h1 { font-family: Arial, sans-serif; color: #CCCCFF; }</pre>	<pre><h1>Заголовок</h1></pre>

Тип селектора	Пример создания	Использование
Классы	.special { <i>color</i> : #FF0000; }	<code><p class="special">...</p></code>
Идентификатор	#banner { <i>background</i> : #CC0000; <i>height</i> : 300px; }	<code><div id="banner">...</div></code>
Групповой селектор	h1, .copyr, #banner { <i>color</i> : red;}	<code><h1>...</h1></code> <code><p class="copyr">...</p></code> <code><div id="banner">...</div></code>
Универсальный селектор	* { <i>font-weight</i> : bold;}	ко всем элементам веб-страницы
Селектор потомков	li a { <i>font-family</i> : Arial; .intro h2 { <i>color</i> : yellow;}	<code><div class="intro"></code> <code><h2>...</h2></code> <code></div></code> <code></code>
Дочерние селекторы	body > h2 { <i>color</i> : green;}	<code><body></code> <code><h2>...</h2></code> <code><div></code> <code><h2>...</h2></code> <code></div></code> <code></body></code>
Родственные селекторы	h2+p { <i>color</i> : green;} h2 ~ p { <i>color</i> : green;}	«+» форматирует элемент p , который сразу следует за h2 . «~» форматирует все элементы p , родственные к элементу h2

Способы подключения CSS

Внешнее подключение является наиболее удобным способом использования стилей и сокращает время обновления веб-страниц. Внешние таблицы стилей хранятся в отдельном файле, который может быть использован для любых веб-страниц, и для подключения используется тег `<link>` внутри тега `<head>` (рис. 2.2).

```
<link href="styles.css" rel="stylesheet" media="all">
```

Рис. 2.2. Внешнее подключение таблицы стилей

Внутренний стиль определяется в самом документе и задается тегом `<style>`, который должен находиться в элементе `<head>`.

По своей гибкости и возможностям этот способ использования стиля уступает предыдущему, но также позволяет размещать все стили в одном месте.

Для строкового способа определения стиля используется атрибут *style*, а его значения указываются с помощью языка каскадных таблиц стилей.

Свойства CSS

Цвет текста присваивается с помощью свойства *color*. Его можно задавать разными способами: по шестнадцатеричному значению, по названию, в формате *rgb*, *rgba*, *hsl*, *hsla*.

Свойство *text-decoration* добавляет оформление текста в виде его подчеркивания, перечеркивания, линии над текстом и мигания. Одновременно можно применить более одного стиля, перечисляя значения через пробел. Свойство *text-decoration* имеет следующие значения:

- *blink* – устанавливает мигающий текст. Такой текст периодически, примерно раз в секунду исчезает, потом вновь появляется на прежнем месте;
- *line-through* – создает перечеркнутый текст;
- *overline* – линия проходит над текстом;
- *underline* – устанавливает подчеркнутый текст;
- *none* – отменяет все эффекты, в том числе и подчеркивание у гиперссылок, которое задано по умолчанию.

Свойство *font-family* устанавливает семейство шрифта, которое будет использоваться для оформления текста содержимого. Список шрифтов может включать одно или несколько названий, разделенных запятой. Если в имени шрифта содержатся пробелы, например Trebuchet MS, оно должно заключаться в одинарные или двойные кавычки.

Свойство *font-size* определяет размер шрифта элемента. Разрешается использовать любые допустимые единицы CSS: *em* (высота шрифта элемента), *ex* (высота символа *x*), *pt* (пункты), *px* (пиксели), % (проценты). За 100% берется размер шрифта родительского элемента.

Каскадность таблицы стилей

Каскадность – это набор правил, который определяет разрешение конфликтов применения стилей.

Когда объявления конфликтуют, то необходимо учитывать следующие показатели:

- 1) источник стилей;
- 2) специфичность селекторов;
- 3) исходный порядок.

Специфичность селекторов определяется типом селекторов. Наиболее специфичным будет селектор с идентификаторами. Далее идет селектор с наибольшим количеством классов. Следующим по специфичности будет селектор с наибольшим количеством тегов.

Если источник и уровень специфичности одинаковы, то объявление стилей, которое указано позже или находится в таблице стилей, на которую ссылаются на веб-странице позже, имеет больший приоритет.

Например, было определено во внешнем стилевом файле (.css), что текст в теге **<p>** должен быть написан при помощи шрифта высотой 10 *pt*. Однако, если при внутреннем подключении стиля дополнительно укажем, что тот же текст в теге **<p>** должен быть написан шрифтом в 12 *pt*, то текст будет выведен шрифтом 12 *pt*, т. е. внутренний стиль в заголовке странички является приоритетным стилем во внешнем файле.

Наследование стилей

Некоторые значения наследуются дочерними элементами. Однако не все свойства наследуются, а по умолчанию – только определенные. Это прежде всего относящиеся к шрифтам: ***color***, ***font***, ***font-family***, ***font-size***, ***font-weight***, ***font-variant***, ***font-style***, ***line-height***, ***letter-spacing***, ***text-align***, ***text-indent***, ***text-transform***, ***white-space*** и ***word-spacing***. Наследуются и некоторые другие, такие как свойства списков: ***list-style***, ***list-style-type***, ***list-style-position*** и ***list-style-image***. Свойства границ таблицы ***border-collapse*** и ***border-spacing*** также наследуются.

Значения, заданные в процентах, не наследуются, а вычисляемые значения наследуются. Например, пусть задана следующая таблица стиля, представленная на рис. 2.3, и фрагмент документа (рис. 2.4).

<pre>body {font-size: 10pt; } h1 { font-size: 120%;}</pre>
--

Рис. 2.4 Пример таблицы стилей

```
<body>
<h1>Некоторый<em>крупный</em>заголовок</h1>
</body>
```

Рис. 2.5 Фрагмент HTML-документа

Свойство *font-size* элемента **<h1>** будет иметь вычисленное значение 12 pt, т. е. 120% от 10 pt, являющегося значением свойства родительского элемента. Поскольку вычисляемое значение свойства является наследуемым, то элемент **** также будет иметь вычисленное значение 12 pt

ЗАДАНИЯ

Задание 1. Убрав устаревшие теги и атрибуты, подключить стили разными способами для копии HTML-документа из задания 1 лабораторной работы № 1, следующим образом:

1. Во внутреннем подключении задать текст абзацев синим цветом, гарнитурой Arial и размером 16 pt.
2. Для внешнего подключения создать групповой селектор для заголовков всех уровней, в которых указать цвет пурпурный и гарнитуру Monotype Corsiva.

Задание 2 Используя классы во внешнем подключении элементы списка отформатировать следующим образом:

1. Текст подпунктов 1 пункта задать красным цветом, гарнитурой Times New Roman и размером 18 pt.
2. Текст подпунктов 2 пункта задать желтым цветом, гарнитурой Times New Roman и размером 20 pt.
3. Текст подпунктов 3 пункта задать зеленым цветом, гарнитурой Times New Roman и размером 22 pt.

Задание 3 Подключить внутренние стили для HTML-документа из задания 2 лабораторной работы № 1, предварительно сделав его копию и заменив устаревшие теги и атрибуты стилями следующим образом:

1. С помощью универсального селектора задать фоновый цвет страницы следующим образом: #BDB76B.
2. Внутри первого элемента **<div>** создать заголовок второго уровня и с помощью дочерних селекторов изменить цвет шрифта на rgb(0, 250, 154).

3. Используя селектор потомков, изменить цвет шрифта заголовков третьего уровня на желтый.

4. Содержимое элемента `` с помощью идентификатора сделать перечеркнутым.

Задание 4. Создать новый HTML-документ под названием `related.html`, в котором должен быть заголовок `<h1>` и два абзаца после него, заголовок `<h2>` и два абзаца после него.

Используя родственные селекторы, для первого абзаца после заголовка `<h1>` задать фон желтым цветом, а для всех абзацев после заголовка `<h2>` – цвет шрифта синий.

Задание 5. С помощью селекторов потомков изменить цвет содержимого ячеек таблицы в копии HTML-документа из задания 1 лабораторной работы № 1.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое CSS? Как расшифровывается CSS?
2. Что представляет из себя синтаксис CSS?
3. Как подключаются внутренние таблицы стилей?
4. Каким образом создаются внешние таблицы стилей?
5. Что такое строковое подключение стилей?
6. Как создаются классы?
7. Каким образом создаются идентификаторы?
8. Как подключается универсальный стиль?
9. Что такое дочерний элемент?
10. Раскройте суть понятия «родительский элемент».
11. Что такое родственные элементы?
12. В чем заключается каскадность стилей?
13. Каков смысл наследования стилей?
14. Что означает групповой селектор и как он создается?
15. Какие свойства не наследуемые?
16. Перечислите способы, которыми можно задать цвет в CSS.
17. Что такое специфичность селектора?
18. Создайте на веб-странице текст «CSS» и первую букву выделите красным цветом; вторую букву – зеленым, третью – синим цветом.

Лабораторная работа № 3

ОФОРМЛЕНИЕ ТЕКСТА И СПИСКОВ ПЕРЕЧИСЛЕНИЯ НА CSS3

Цель работы: изучить основные свойства CSS для форматирования текста и списков, основы верстки текста на веб-странице, а также познакомиться с псевдоэлементами и псевдоклассами.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Псевдоклассы и псевдоэлементы

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также в зависимости от положения в DOM. Псевдоэлементы позволяют задать стиль определенной части, не относящийся к элементам документа, а также генерировать содержимое, которого нет в исходном коде текста.

К псевдоэлементам относятся следующие:

- ***first-letter*** – позволяет оформить первую букву указанного элемента;
- ***first-line*** – позволяет оформить первую строчку заданного элемента;
- ***before, after*** – позволяет вставлять произвольное содержимое соответственно до и после указанных элементов.

В спецификации CSS3 псевдоэлементы обозначаются двойным символом двоеточия (::), но для лучшей совместимости с более старыми версиями желательно использовать один символ двоеточия (:).

Подключение шрифта

Чтобы подключить новые шрифты, необходимо сначала скачать шрифты в папку на компьютере, где хранятся файлы для вашего веб-сайта. Для этого можно создать выделенную папку в корневом каталоге веб-сайта, которая носит имя *fonts*.

Для подключения скачанных шрифтов используется правило **@font-face**, которое присваивает шрифту имя и сообщает браузеру, где найти файл шрифта для его загрузки. Например, в папке *fonts* находится шрифт формата Web Open Font с именем Lobster (рис. 3.1).

```
@font-face
{
  font-family: 'Lobster';
  src: url('fonts/lobster.woff') format('woff');
      url('fonts/lobster.svg') format('svg');
}
```

Рис. 3.1. Пример использования правила @font-face

С помощью свойства **font-family** присваивается шрифту имя. Затем это имя используется, когда соответствующий шрифт нужно применить к стилю. Например, нужно использовать шрифт Lobster для всех заголовков первого уровня на странице. Для этого можно будет указать стиль следующего вида: **p {font-family: 'Lobster';}**.

Если необходимо использовать несколько шрифтов, то потребуется несколько правил **@font-face**. Формат шрифта woff в большинстве случаев предпочтительнее, файлы формата svg намного больше по размеру и поддерживаются только браузером Safari.

Свойства форматирования текста

Значение *italic* свойства **font-style** изменяет начертание текста на курсивное, а для того чтобы сделать текст полужирным, используется значение *bold* свойства **font-weight**.

Чтобы изменить регистр текста, необходимо применить свойство **text-transform** со значением *uppercase*. Сделать буквы строчными можно с помощью значения *lowercase*, а превратить первые буквы в прописные — с помощью слова *capitalize*. Для создания стиля с малыми прописными буквами используется свойство **font-variant** со значением *small-caps*.

С помощью свойства **text-align** можно расположить абзац в центре (*center*) веб-страницы, вдоль левого (*left*) или правого (*right*) края или выровнять по ширине (формату) (*justify*).

Для создания эффекта тени используется свойство **text-shadow**, которое требует задания трех параметров: горизонтального смещения (насколько левее или правее текста должна отобра-

жаться тень), вертикального смещения (насколько выше или ниже текста должна появиться тень), степени размытости тени и цвета отбрасываемой тени.

Для установки размера междустрочного интервала предназначено свойство ***line-height***, задаваемое в процентах. Чтобы увеличить междустрочный интервал, т. е. распределить строки дальше друг от друга, используется значение, большее 120%.

Чтобы полностью избавиться от верхнего и нижнего полей, применяется следующий код: **p {*margin-top*: 0; *margin-bottom*: 0;}**.

Для того чтобы добавить абзацный отступ (отступ первой строки), служит свойство ***text-indent***.

Свойства форматирования списков

Используя свойство ***list-style-type***, можно выбрать маркеры маркированного списка трех типов: *disc* (сплошной кружок), *circle* (полый кружок), *square* (сплошной квадрат). Для нумерованных списков предусмотрено шесть вариантов нумерации: *decimal*, *decimal-leading-zero*, *upper-alpha*, *lower-alpha*, *upper-roman*, *lower-roman*. С помощью свойства ***list-style-position*** устанавливается местоположение маркера вне (значение *outside*) или внутри (*inside*) текстовых блоков элементов списка.

Браузеры создают для списков отступ, смещая все содержимое вправо. Однако можно переопределить стиль, присвоив свойствам ***margin-left*** и ***padding-left*** значение 0.

Если изменить только маркер списка, то можно применять CSS-код, представленный на рис. 3.2.

```
ol li {
  font-family: Verdana;
  font-size: 20pt;
  list-style-type: none;
  counter-increment: item; /* Указываем идентификатор */
}
ol li:before {
  content: counter(item, upper-roman) "."; /* Выводим текст перед содержимым тега li */
  color: red; }
```

Рис. 3.2. Изменение маркера нумерованного списка

Чтобы использовать в качестве маркера графическое изображение, существует свойство *list-style-image*, которое определяет путь к графическому символу на сервере. Следует отметить, что необходимо учитывать размер данного изображения. Синтаксис используется следующий: *list-style-image: url(images/bullet.gif)*.

ЗАДАНИЯ

Задание 1. Создать новый HTML-документ под названием text.html, в котором должны быть следующие элементы: заголовок **<h1>** и после него два заголовка **<h2>**, три абзаца, из которых два абзаца должны быть расположены после первого заголовка **<h2>**.

Изменить гарнитуру текста с помощью правила **@font-face**, выбрав из папки *fonts* шрифты и переместив их в созданную предварительно папку *fonts* в корневом каталоге веб-страницы, следующим образом:

1. Для заголовка **<h1>** применить шрифт Carveat;
2. Для заголовка **<h2>** применить шрифт Echo2;
3. Для текста применить шрифт Comfortaa.

Задание 2. Используя свойства для форматирования текста, изменить его следующим образом:

1. Для заголовка **<h1>** убрать отступ до следующего заголовка, применить выравнивание по центру, сделать начертание полужирным, прописные буквы размером 45 px, добавить тень.
2. Для заголовков **<h2>** установить отступ до текста снизу 1.2 em, отступ сверху 0.2 em, отступ первой строки, равным 25 px, выравнивание по левому краю.
3. Для абзацев сделать отступ первой строки равным 25 px, выравнивание по ширине, убрать отступы между абзацами, интерлиньяж задать 200%, размер шрифта 20 px.

Задание 3. Используя псевдоэлементы изменить первую букву каждого абзаца, сделав полужирным начертанием, размером 45 px, белого цвета, фоном `hsla(275, 100%, 10%, 1)`.

Задание 4. В этом же документе создать нумерованный список из 5 пунктов. Нумерацию списка сделать из заглавных римских цифр синего цвета, текст пункта черного цвета. Гарнитура и размер текста должны соответствовать тексту предыдущих абза-

цев. Перед списком добавить заголовок второго уровня, после которого должен отсутствовать отступ.

Задание 5. После списка создать новый абзац из трех строк и изменить цвет только первой строки на красный. Отступ первой строки абзаца сделать равным 25 px, текст абзаца – размером шрифта 20 px и гарнитурой шрифта Comfortaa.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое псевдоэлементы? Назовите псевдоэлементы, которые вы знаете.
2. Раскройте суть понятия «псевдоклассы».
3. Для чего используется псевдоэлемент *before*? Приведите пример.
4. С какой целью применяется псевдоэлемент *after*? Приведите пример.
5. Каким образом изменить первую букву текста?
6. Как создать отступ первой строки?
7. Расскажите, как убрать отступы между фрагментами текста?
8. Каким образом изменить регистр?
9. Поясните, как сделать малые прописные буквы.
10. Как добавить тень к тексту?
11. Расскажите, как изменить только цвет маркера нумерованного списка.
12. Какие действия надо выполнить, чтобы изменить выравнивание текста на странице?
13. Как добавить новый шрифт?
14. В каких форматах добавляется новый шрифт и от чего это зависит?
15. Как изменить маркер на изображение для маркированного списка?
16. В каких единицах можно задавать параметры текста?
17. Как изменить междустрочный интервал?
18. Каким образом текст сделать полужирным или курсивом?
19. Как изменить гарнитуру шрифта из набора существующих?
20. Каким образом изменить размер текста?
21. Как изменить интервал между словами и символами?

Лабораторная работа № 4

БЛОЧНАЯ МОДЕЛЬ CSS

Цель работы: познакомиться с блочной моделью, изучить свойства блочных элементов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Блочные элементы CSS

В CSS существуют два различных типа элементов: блочные и строчные. Строчные элементы не создают отступы до и после, они отображаются на одной строке с содержимым рядом стоящих элементов. Примерами строчных элементов являются ****, ****, ****.

В блочных элементах создается разрыв строки перед элементом и после него. Например, абзац **<p>** создает отступ, отделенный от элементов, расположенных выше и ниже его. Другими примерами являются теги заголовков, контейнеры **<div>**, таблицы, теги списков.

Свойства блочных элементов

Браузер обрабатывает все элементы как небольшие блоки. Этот блок состоит из нескольких слоев, которыми можно независимо управлять с помощью CSS. Основной частью каждого блока элемента является область содержимого. Вокруг области содержимого есть три слоя пространства, которые управляются следующими свойствами (рис. 4.1):

- ***padding*** – отступ, пространство между содержимым и границей.
- ***border*** – граница, линия вдоль края блока.
- ***margin*** – поле, которое определяет расстояние от границы одного элемента до другого.

Для блочных элементов значение ширины области содержимого элемента по умолчанию равно 100%. У строчных элементов

ширина зависит от количества содержимого. Высота блока зависит от содержимого, но для блочных элементов можно указать конкретную высоту.

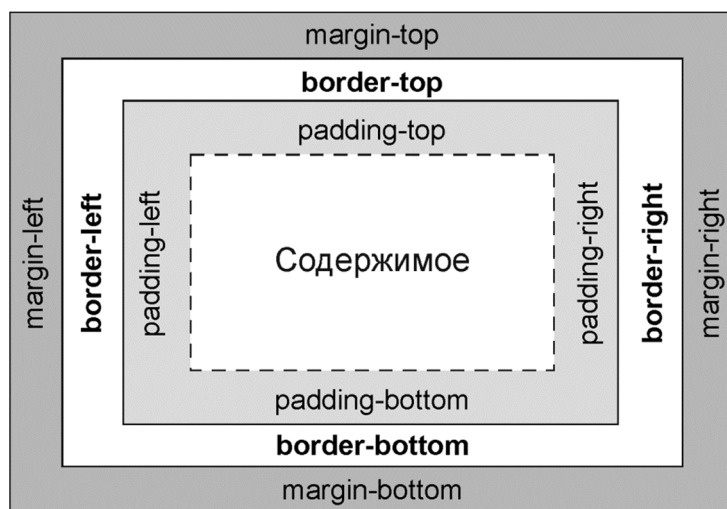


Рис. 4.1. Свойства блочных элементов

Для форматирования элемента можно использовать любые из этих свойств в любом сочетании или все сразу. Для данных свойств применяются любые единицы измерения, принятые в языке CSS для определения размеров полей и отступов (рис. 4.2).

```
margin-right: 20px;  
padding-top: 3em;  
margin-left: 10%
```

Рис. 4.2. Примеры использования свойств блочных элементов

Можно использовать сокращенные варианты свойств ***margin*** и ***padding*** для быстрой установки всех четырех параметров одновременно. Они должны указываться в следующей последовательности: сверху, справа, снизу и слева (рис. 4.3).

```
margin: 0 10px 10px 20px;  
padding: 10px 5px 5px 10px;
```

Рис. 4.3. Сокращенная запись свойств ***margin*** и ***padding***

Также блочные элементы могут быть выровнены по центру установкой с левой и правой стороны значения *auto* для свойства ***margin***.

Свойство **display**

В некоторых случаях требуется, чтобы строчные элементы вели себя так же, как блочные, или наоборот. В языке CSS позволяет это сделать свойство **display**. С его помощью можно заставить блочный элемент функционировать как строчный: **display: inline**. Чтобы строчные элементы, например изображение или гиперссылка, вели себя как блочные, присваивается **display: block**.

Также можно заставить элемент действовать и как блочный, и как строчный. Значение **inline-block** не создает разрывов ни до, ни после элемента и одновременно заставляет элемент подчиняться верхним и нижним полям и отступам, а также настройкам высоты: **display: inline-block**.

Свойства границ

Можно управлять тремя различными свойствами любой из границ: **color** (цвет), **width** (ширина) и **style** (стиль). Для ширины границы используются любые единицы измерения каскадных таблиц стилей (кроме процентов) или ключевые слова *thin* (тонкая линия), *medium* (средняя) и *thick* (толстая). Самые распространенные единицы измерения для данного свойства – пикселы.

Свойство **style** управляет типом линии границы. В каскадных таблицах стилей для границ имеются следующие стили: *solid*, *dotted*, *dashed*, *double*, *groove*, *ridge*, *inset*, *outset*, *none* и *hidden*. Ключевые слова *none* и *hidden* работают одинаково: они полностью удаляют границы, но значение *none* удобно применять для удаления границы с одной стороны элемента. Для установки границ можно использовать сокращенную запись или расширенную следующим образом (рис. 4.4).

```
border: 2px double #FFCC33;  
/* или */  
border-width: 2px;  
border-style: double;  
border-color: #FFCC33;
```

Рис. 4.4. Свойства границ блочных элементов

Следует отметить, что каждая сторона имеет свой набор из трех свойств, которые удобно использовать для отмены одного.

Для правой границы предусмотрены следующие свойства: ***border-right-width***, ***border-right-style*** и ***border-right-color***. Левая, верхняя и нижняя границы имеют похожие свойства: ***border-left-width***, ***border-left-style*** и т. п.

Однако можно задать собственные значения сразу для каждой стороны границы, используя сокращенную запись. Например, правило ***border-width: 10px 5px 15px 13px;*** применит четыре различных значения ширины для каждой из сторон (верхней, правой, нижней и левой).

В языке CSS существует также свойство ***border-radius***, позволяющее добавлять скругления к одному или нескольким углам элемента. Для каждого угла можно указать отдельные значения, задав четыре параметра. Объявление свойства имеет следующий вид: ***border-radius: 0 30px 10px 5px;***. Сначала задается числовое значение для левого верхнего угла блока, а затем по часовой стрелке для всех остальных углов.

Чтобы добавить эллиптические углы, нужно создать следующее объявление: ***border-radius: 40px/20px;***.

Добавление тени

Для добавления теней к блоку, обрамляющему элемент, используется свойство ***box-shadow***. По сравнению со свойством ***text-shadow*** тень можно добавлять внутри блока с помощью ключевого слова ***inset***. Основной синтаксис свойства ***box-shadow*** следующий (рис. 4.5).

```
box-shadow: inset 4px 4px 8px 12px rgba(0,0,0,.75);
```

Рис. 4.5. Синтаксис свойства ***box-shadow***

Первое значение задает горизонтальное смещение, которое приводит к перемещению тени влево или вправо от элемента. Положительное число вызывает перемещение тени вправо, а отрицательное число – влево.

Второе значение задает вертикальное смещение – позицию тени либо над элементом, либо под ним. При положительном значении тень располагается ниже нижнего края блока, а при отрицательном значении тень помещается над верхним краем блока.

Третье значение задает радиус размытия тени. Оно определяет степень размытости и ширины тени. Чем выше значение, тем более размытой и тусклой становится тень.

Четвертое значение (между радиусом размытия тени и ее цветом) добавляет расширение тени на указанное значение.

Последнее значение задает цвет отбрасываемой тени. Можно воспользоваться любым обозначением цвета, принятым в языке CSS, но RGBA-значения позволяют управлять прозрачностью (*alpha*) тени, делая ее более реалистичной.

Размеры блочных элементов

Браузеры вычисляют ширину блочного элемента, складывая значения свойств ***border***, ***padding*** и ***width***. Свойство ***box-sizing*** изменяет порядок вычисления браузером экранной ширины (и высоты) элемента в зависимости от следующих значений:

1) ***content-box*** – добавляет ширину границ и значения отступов к значениям, установленным для свойств ширины и высоты;

2) ***padding-box*** – сообщает браузеру, что значения свойства ширины или высоты включают в себя отступы ***padding*** как часть своего значения, но не ***margin*** и ***border***;

3) ***border-box*** – включает в значения свойства ***width*** и ***height*** значения отступов и границ, но не ***margin***. Это значение можно использовать для универсального селектора.

Когда содержимое форматируемого элемента имеет размеры больше определенных свойствами ***width*** и ***height***, то используется свойство ***overflow*** со следующими значениями:

– ***visible*** – имеет тот же эффект, что отсутствие установки свойства ***overflow***;

– ***scroll*** – позволяет добавить полосы прокрутки, создавая окно внутри веб-страницы, которое выглядит подобно устаревшим HTML-фреймам;

– ***auto*** – выполняет ту же функцию, что и ***scroll***, но полосы прокрутки в данном случае появляются только при необходимости;

– ***hidden*** – скрывает любое содержимое, выходящее за пределы блочного элемента.

Свойство ***overflow*** является сокращенной записью для свойств ***overflow-x*** и ***overflow-y***.

ЗАДАНИЯ

Задание 1 Создать HTML документ с заголовком «Блочная модель», в котором будет создано семь `<div>`, согласно рис. 4.6.

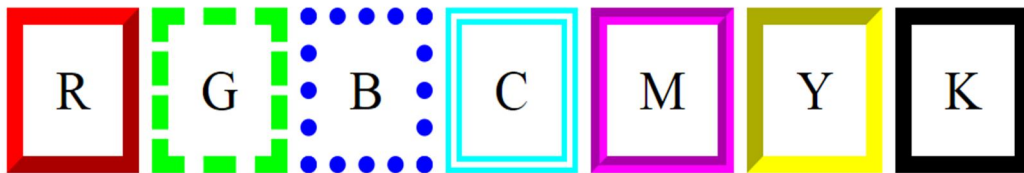


Рис. 4.6. Результат выполнения задания 1

1. Все блоки должны располагаться в одну линию.
2. Цвета изменяются в следующем порядке: красный, зеленый, синий, голубой, пурпурный, желтый, черный. Задать их с помощью `rgb`.

3. Установить все отступы по 10 px, отступ сверху 10 px.

Задание 2. Создать в этом же документе три абзаца, в каждый из абзацев добавить по одному строчному элементу ``.

Строчные элементы преобразовать таким образом, чтобы они могли иметь и свойства блочных элементов. Для них задать отступы и поля по 5 px, фон цветом `#FFA500`, сплошную границу цветом `#FF4500`. Также для каждого абзаца присвоить следующие свойства:

1. Для первого абзаца задать ширину 400 px и его выравнивание с текстом по центру, сплошную границу цвета `#FF7F50`.

2. Для второго абзаца установить сплошную границу красного цвета с толщиной 5 px и с эллиптическими углами округления 20 px/40 px; добавить внутреннюю тень `rgba(0, 0, 0, .5)` цвета со смещением по горизонтали вправо и по вертикали вниз на 2 px, с размытием 8 px и расширением 8 px. Установить значение *auto* свойства *overflow*.

3. Третий абзац должен иметь сплошную границу цвета `#FF6347`.

Задание 3. Скопировать `block.html` из папки *labs*. Добавить необходимые элементы, а также свойства абзацам согласно их описанию.

Задание 4. Создать элементы, представленные на рис. 4.7. Ширина совпадает с высотой. Для первого элемента необходимо

задать значение *border-box*, а для второго – *content-box* (самостоятельно определить свойство этих значений).

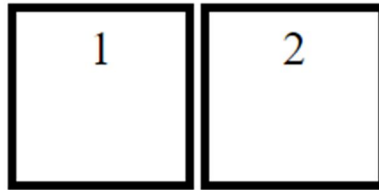


Рис. 4.7. Результат выполнения задания 4

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте понятие «блочный элемент». Приведите примеры.
2. Раскройте суть понятия «строчный элемент». Приведите примеры.
3. Как задать свойства блочных элементов строчным?
4. Что включает в себя сокращенная запись свойства *margin*?
5. Какие значения имеет свойство *box-shadow*?
6. Как разместить элемент по центру?
7. Какие свойства имеют блочные элементы?
8. Перечислите свойства, которые имеет граница блочных элементов.
9. Каким образом определяется ширина блочного элемента?
10. Для чего используется свойство *box-sizing*? Какие значения принимает это свойство?
11. Для чего используется свойство *overflow*?
12. Что обозначают значения в следующем объявлении: *border-color: yellow red green blue*;
13. Какое свойство можно использовать, чтобы задать ширину только правой границы?
14. Назовите, какие значения принимает свойство *border-style*.
15. Создайте блочный элемент и, используя сокращенную запись, установите 5 px пунктирную границу красного цвета с отступом от текста сверху 20 px.

Лабораторная работа № 5

ФОРМАТИРОВАНИЕ ТАБЛИЦ И ВЕБ-ФОРМ НА CSS3

Цель работы: получить навыки создания таблиц, изучить основные свойства CSS для форматирования таблиц и веб-форм.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Псевдоклассы для форматирования таблиц

Для форматирования таблиц можно использовать следующие псевдоклассы:

- ***first-child*** – позволяет оформить первый дочерний элемент;
- ***last-child*** – дает возможность оформить последний дочерний элемент;
- ***nth-child(odd)*** – позволяет оформить чередующиеся четные дочерние элементы;
- ***nth-child(even)*** – дает возможность оформить чередующиеся нечетные дочерние элементы;
- ***nth-child(3n)*** – позволяет оформить каждый третий элемент;
- ***nth-child(5n+3)*** – оформляет каждый пятый элемент, начиная с третьего.

Свойства форматирования таблиц

Текстовая информация таблицы располагается в ячейках, которые имеют границы. Для создания границ используется свойство ***border***, а свойство ***border-spacing*** предназначено для управления размером промежутка между ячейками, который образуется при создании таблицы.

Если убрать промежутки между ячейками, то их границы будут удваиваться. Чтобы избавиться от промежутков между ячейками, необходимо использовать свойство ***border-collapse***. Значение ***separate*** эквивалентно тому, как обычно и отображаются таблицы:

с промежутками между ячейками и двойными границами. Значение *collapse* позволяет избавиться от удвоения границ.

Для изменения отступа от границ до содержимого ячеек служит свойство ***padding***, пример использования которого представлен на рис. 5.1.

```
td {  
padding-top: 10px;  
padding-right: 5px;  
padding-bottom: 3px;  
padding-left: 5px;  
}  
td { padding: 10px 5px 3px 5px; }
```

Рис. 5.1. Пример использования padding

Чтобы настроить расположение содержимого внутри ячейки, применяются свойства ***text-align*** для выравнивания по горизонтали и ***vertical-align*** для выравнивания по вертикали. Свойство ***vertical-align*** принимает значения *top* (выравнивание по верхнему краю), *baseline*, *middle* (выравнивание по центру) или *bottom* (выравнивание по нижнему краю). При указании значения *baseline* выравнивание происходит так же, как и при установке значения *top*, за исключением того, что браузер выравнивает первую строку текста в каждой ячейке относительно строки родительского элемента таблицы.

Чтобы добавить к ячейкам, но не к таблице скругленные углы, применяется свойство ***border-radius***. Для того чтобы скрыть пустые ячейки, используется свойство ***empty-cells*** со значением *hide*, примененному к таблице. Следует учитывать, что, если свойству ***border-collapse*** присвоить значение *collapse*, браузеры проигнорируют свойства ***empty-cells*** и ***border-radius***.

Свойства форматирования веб-форм

По умолчанию тег **<label>** строчный, игнорирующий свойства, доступные блочным элементам, включая свойства ***width***, ***height*** и ***text-align***. Если их превратить в блочные элементы с помощью значения *inline-block* свойства ***display***, то они будут располагаться рядом с элементами веб-формы и иметь свое строчное происхождение, но и не будут игнорировать свойства блочных элементов.

Для того чтобы выровнять текст метки по правому краю, чтобы каждая метка отображалась рядом с соответствующим элементом веб-формы и увеличить поле справа от метки можно использовать стиль, показанный на рис. 5.2.

<pre>.label { display: inline-block; width: 20em; vertical-align: top; text-align: right; margin-right: 15px; }</pre>	<div>Имя <input style="width: 100%;" type="text"/></div> <div>Телефон <input style="width: 100%;" type="text"/></div> <div>Адрес <input style="width: 100%;" type="text"/></div>
---	--

Рис. 5.2. Свойства и результат расположения элементов формы

Для изменения внешнего вида текстового поля при щелчке на нем кнопкой мыши или при переходе на него нажатием клавиши **Tab** используется псевдокласс *:focus*. Это называется фокусировкой на элементе веб-формы.

Чтобы выбрать конкретные элементы веб-формы, например, только текстовые поля в форме веб-страницы, служит выражение *input[type="text"]*, которое представляет собой селектор атрибутов.

ЗАДАНИЯ

Задание 1. Создать HTML-документ, в котором будет расположена таблица, содержащая расписание занятий подгруппы с понедельника по субботу на две недели. Для примера рассмотреть вид расписания факультета.

Между ячейками полностью убрать отступ, к первой строке и первому столбцу применить фон произвольного цвета. Цвет первой строки должен быть отличным от цвета первого столбца. К таблице добавить заголовок, содержащий ФИО студента, курс, группу, подгруппу.

Задание 2. Используя теги `<thead>`, `<tfoot>`, `<tbody>` создать таблицу в новом документе, представленную на рис. 5.3.

1. Убрать пустые ячейки с помощью свойства CSS.
2. Установить следующие заливки: для головки таблицы – `#00BFF`, для второго столбца – *darkblue*, для третьего столбца – *purple*, для четвертого столбца – зеленый.

3. Для заголовка таблицы задать следующие свойства: размер текста 1.2 em, выравнивание по центру.

Russia:Top smartphone vendors
Q1 2021

	Vendor	Unit Share	Annual Growth
#1 ⬆	Xiaomi	32%	+125%
#2 ⬆	Samsung	27%	+41%
#3 ⬆	Honor	10%	0%
#4 ↔	Apple	9%	+24%
#5 ↔	Realme	9%	+682%
	Total	82%	682%

Рис. 5.3. Результат выполнения задания 1

Задание 3. Создать HTML-страницу, на которой разместить форму, показанную на рис. 5.4. Для изменения отдельных элементов формы использовать селекторы атрибутов.

Регистрация постоянного клиента

Фамилия:

Имя:

Отчество:

Ваш возраст:

Пол: ☐ мужской ☐ женский

Email:

Страна проживания:

Откуда вы узнали о нас?

Дата заполнения

Рис. 5.4 Результат выполнения задания 3

1. Цвет фона установить indigo.
2. Для текстовых полей и поля *email* задать следующие параметры цвета: цвет фона – `rgba(255, 255, 255, .5)`, цвет текста –

rgba(255, 255, 255, 1), тень – **box-shadow: inset 0 0 10px rgba(255, 255, 255, .75).**

3. Установить размер текста 1.2em.

4. Кнопкам присвоить следующие цветовые параметры: фон – navy, тень – **box-shadow: 0 0 4px white;**

5. При вводе текста в текстовые поля фоновый цвет должен измениться на белый, а цвет ввода на черный. Поле *email* оставить без изменений.

Задание 4. Используя только псевдоклассы **:first-child**, **:last-child**, **:nth-child(odd)** и **:nth-child(even)**, создать шахматную доску в новом документе, приведенную на рис. 5.5. Цвета можно использовать произвольные.

	a	b	c	d	e	f	g	h	
8									8
7									7
6									6
5									5
4									4
3									3
2									2
1									1
	a	b	c	d	e	f	g	h	

Рис. 5.5 Пример результата задания 4

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет из себя селектор атрибутов?
2. Какие типы селекторов атрибутов вы знаете?
3. Расскажите, как убрать отступы между ячейками таблицы.
4. Как убрать пустые ячейки таблицы?
5. Что означают значения в сокращенной записи свойства *padding*?

6. Для чего используется псевдокласс *focus*?
7. С какой целью применяется псевдокласс *nth-child(odd)*?
8. Для чего предназначен псевдокласс *nth-child(even)*?
9. Каким образом сделать тень к элементам формы?
10. Как изменить промежуток между ячейками?
11. Для чего используется свойство *border-radius*?
12. Какое свойство используется для выравнивания текста в таблице по вертикали в ячейках?
13. Назовите свойство, которое предназначено для выравнивания текста в таблице по горизонтали в ячейках.
14. Какие значения может принимать свойство *vertical-align*?
15. Для чего используется псевдокласс *first-child*?
16. С какой целью применяется псевдокласс *last-child*?
17. Для чего предназначены теги **<thead>**, **<tfoot>** и **<tbody>**?
18. Какому свойству относится значение *separate* и для чего оно используется?
19. Какому свойству относится значение *collapse* и для чего оно используется?
20. Для чего предназначены теги **<colgroup>** и **<col>**?
21. Что означает значение *baseline*?
22. Создайте таблицу из 6 строк и 5 столбцов и выделите серым цветом каждую третью строку.
23. Создайте таблицу из 4 строк и 9 столбцов и выделите серым цветом каждый второй столбец начиная с третьего.

Лабораторная работа № 6

ГРАФИКА НА ВЕБ-СТРАНИЦЕ

Цель работы: научиться использовать фоновые изображения, создавать градиентный фон, получить навыки позиционирования графических изображений на веб-странице.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Фоновые изображения

Для того чтобы установить фоном цвет, используется свойство ***background-color***, а чтобы сделать изображение фоном, применяется свойство ***background-image***. Свойство принимает единственное значение – ключевое слово *url*, за которым следует путь к графическому файлу, заключенный в круглые скобки. Можно использовать абсолютный URL-адрес, например, *url:(http://www.cosmofarmer.com/image/bg.gif)*, или относительный путь от документа или корневого каталога сайта, представленный на рис. 6.1.

```
url(../images/1.jpg) /* относительный путь от документа */  
url(/images/1.jpg) /* относительный путь от корневого каталога */
```

Рис. 6.1. Относительная адресация

Современные браузеры работают только с четырьмя графическими форматами: gif, jpeg, png и svg.

Если свойство ***background-image*** используется без указания дополнительных свойств, то фоновое изображение многократно повторяется, заполняя всю веб-страницу. Чтобы определить, каким образом будет повторяться фоновое изображение, используется свойство ***background-repeat*** со следующими значениями:

– *repeat* – параметр по умолчанию, обеспечивает повторное отображение фонового изображения слева направо и сверху вниз до полного заполнения всего пространства веб-страницы;

- *no-repeat* – отображает фоновое изображение без повторения;
- *repeat-x* – вызывает повторение фонового изображения горизонтально вдоль оси *X*;
- *repeat-y* – повторяет фоновое изображение вертикально вдоль оси *Y*;
- *round* – повторяет фоновое изображение таким образом, чтобы в области поместилось целое число без обрезки;
- *space* – повторяет фоновое изображение по горизонтали и вертикали без искажений и обрезание изображений.

Для управления расположением графики используется свойство позиционирования фонового изображения ***background-position*** (рис. 6.2). Свойство должно принимать два значения: одно из них для горизонтального позиционирования (*left*, *center*, *right*), а другое – вертикального (*top*, *center*, *bottom*).

```
background-position: center center;
background-position: right top;
```

Рис. 6.2. Пример использования свойства **background-position**

Позиционировать фоновые изображения можно, используя точные значения в пикселах или единицах *em*. Начальную позицию изображения можно изменить, воспользовавшись свойством ***background-origin***, которое принимает такие значения как:

- *border-box* – изображение помещается в верхний левый угол области границы;
- *padding-box* – изображение вставляется в верхний левый угол области отступа, это место по умолчанию;
- *content-box* – изображение помещается в левый верхний угол области содержимого.

Следует отметить, что свойство ***background-origin*** может использоваться со свойством ***background-clip***, которое ограничивает область появления фонового изображения и имеет следующие значения:

- *border-box* – позволяет рисунку отображаться позади содержимого и любых границ;
- *padding-box* – ограничивает любое фоновое изображение областью отступов и содержимого элемента;
- *content-box* – позволяет ограничить фоновое изображение областью содержимого элемента;

Для управления размером фонового изображения служит свойство ***background-size***.

Следует отметить, что для использования изображения как типа маркера предназначено свойство ***background-image*** (рис. 6.3).

```
.spisok li {  
  list-style: none;  
  background-image: url(images/2.png);  
  background-repeat: no-repeat;  
  background-position: 0 4px;  
  padding-left: 25px;  
  margin-bottom: 10px;  
}
```

Рис. 6.3. Использование изображения в виде маркера

С помощью свойства ***background-attachment*** можно закреплять положение изображения на веб-странице. Значение по умолчанию *scroll* определяет такое поведение браузера, при котором фоновое изображение прокручивается вместе с текстом и другим контентом, а значение *fixed* предотвращает перемещение, фиксируя его положение на заднем плане.

Градиентные фоны

Градиент – это плавный переход от одного цвета к другому. Существует линейный и радиальный градиенты. Для указания градиента используются значения *linear-gradient()* или *radial-gradient()* свойства ***background-image***. Линейный градиент значения *linear-gradient()* распространяется по прямой от одного конца элемента к другому, демонстрируя плавный переход от одного цвета к другому. Необходимо указать его направление в градусах или с помощью значений *top*, *bottom*, *right*, *left* либо их комбинаций.

Значения в градусах повторяют часовую стрелку на циферблате, поэтому значение 90deg соответствует правому краю элемента (*right*), 180deg – нижнему краю элемента (*bottom*), а 270deg – левому краю элемента (*left*) и 0deg – значению *top*.

Также можно создавать составной градиент в процентах из нескольких цветов: ***background-image: linear-gradient(to right, red 20%, orange 30%, orange 70%, red 80%)***. Это означает, что первые 20% протяженности элемента (слева направо) будут иметь фон из

сплошного красного цвета. Затем от 20%-ной до 30%-ной точки градиент плавно перейдет от красного к оранжевому цвету.

Пример создания линейного градиента приведен на рис. 6.4.

```
background-image: linear-gradient(to right, black,
white);
или
background-image: linear-gradient(90deg, black,
white);
```

Рис. 6.4. Черно-белый градиент от левого угла к правому

Также можно использовать значение *transparent* для любого цвета в градиенте, чтобы видеть область, расположенную позади него, например фоновый цвет элемента или другой линейный градиент (рис. 6.5).

```
background-image: linear-gradient(cyan, transparent),
linear-gradient(225deg, magenta, transparent);
```

Рис. 6.5. Использование значения *transparent*

Чтобы получить повторяющийся линейный градиент, следует написать ***background-image: repeating-linear-gradient(45deg, #900 20px, #FC0 30px, #900 40px)***.

Радиальный градиент распространяется наружу по круговой или эллиптической схеме. Необходимо только задать начальный цвет (цвет в середине градиента) и конечный (цвет в конце), например ***background-image: radial-gradient(red, blue)***. Чтобы создать круговые градиенты, необходимо добавить перед указанием цветов *circle*. Значения позиционирования центра записываются перед значениями, определяющими форму и цвета, например, ***background-image: radial-gradient(circle at 20% 40%, red, blue)***.

С помощью свойства ***background-blend-mode*** можно применять режимы смешивания, которые используются в Adobe Photoshop.

К свойствам, которые разработаны для конкретного браузера или еще не рекомендованы к использованию консорциумом W3C, требуется добавить префиксы. Так для браузеров Microsoft требуется префикс ***-ms***, для Chrome и Safari – префикс ***-webkit***, для Opera префикс – ***-o*** и для Firefox префикс – ***-moz***, например, ***-ms-radial-gradient***.

Обтекание изображений текстом

HTML-контент отображается начиная с верхнего края окна браузера по направлению к нижнему друг под другом. Свойство ***float*** перемещает любой элемент в нужную позицию, выравнивая его по левому или правому краю веб-страницы. В процессе перемещения содержимое, находящееся ниже позиционируемого плавающего элемента, смещается вверх и обтекает плавающий элемент. В качестве значения используются *left*, *right* или *none*.

HTML-код обтекаемого элемента должен быть указан перед HTML-кодом элемента, который его обтекает.

Чтобы запретить влияние обтекания на другие элементы, используется свойство ***clear*** со следующими значениями:

1) *left* – формируемый элемент смещается вниз относительно плавающего с установленным обтеканием слева, но обтекание справа остается;

2) *right* – формируемый элемент смещается вниз относительно плавающего с установленным обтеканием справа, но обтекание слева остается;

3) *both* – вызывает перемещение формируемого элемента вниз относительно плавающего с установленным обтеканием и слева и справа;

4) *none* – полностью отменяет запрет обтекания – сообщает браузеру, что формируемый элемент должен вести себя так, как предусмотрено, т. е. он может обтекать плавающие элементы как с левой, так и с правой стороны.

ЗАДАНИЯ

Задание 1. Для HTML-документа `index.html` из задания 1 лабораторной работы № 1, предварительно сделав его копию, осуществить следующие преобразования:

1. Маркеры маркированного списка сделать в виде изображения, используя свойства ***background-image***.

2. Добавить в виде фона логотип факультета в правом верхнем углу, чтобы он прокручивался вместе с содержимым.

3. Не должно быть устаревших атрибутов и тегов, заменить их свойствами CSS.

Задание 2. Добавить в документ для задания 1 настоящей работы изображения, согласно прототипу (рис. 6.6).

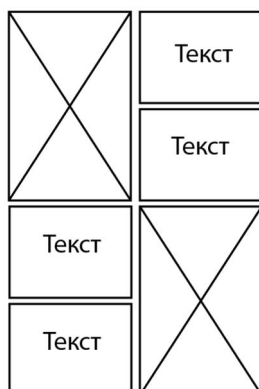


Рис. 6.6. Прототип к заданию 2

Форматирование абзацев должно включать в себя отсутствие отступов между ними и отступ первой строки.

Задание 3. Создать в этом же документе гиперссылку на новую веб-страницу, на которой необходимо расположить фотогалерею из 12 фотографий по 4 фото в одном ряду на произвольную тему. Фотографии должны быть размещены с использованием тега `<figure>` и каждая иметь название.

Задание 4. Создать веб-страницу, на которой разместить элементы `<div>` с произвольными цветами (рис. 6.7).



Рис. 6.7. Результат выполнения задания 4

Задание 5. Создать карту-изображение, используя лекцию 3. Карта-изображение должна содержать гиперссылки на 4 изображения.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет из себя абсолютная адресация?
2. В чем суть относительной адресации?

3. Что такое относительный путь от документа? от корневого каталога?
4. Каким образом создать линейный градиент?
5. Поясните, как создать радиальный градиент.
6. Каким образом создать обтекание изображения текстом?
7. Создайте веб-страницу, на которой обтекание изображения происходит текстом слева.
8. Для чего необходимо свойство ***background-repeat***?
9. Какие форматы изображений можно загружать на веб-страницу?
10. Для чего используется свойство ***background-image***? Какие значения оно может принимать?
11. Каким образом изменить размер изображения в CSS?
12. Для чего предназначены префиксы?
13. Каким образом зафиксировать фоновое изображение?
14. Для чего используется свойство ***background-blend-mode***? Назовите значения свойства.
15. С какой целью применяется ***transparent***?
16. Что такое карта-изображение?
17. Как сделать вид маркера графическим?
18. Для чего необходимо свойство ***background-origin***?

Лабораторная работа № 7

ПРЕОБРАЗОВАНИЯ. ПЕРЕХОДЫ.

АНИМАЦИЯ

Цель работы: изучить свойства преобразования, применение анимации к элементам, научиться использовать переходы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Преобразования

К свойствам преобразования объекта относятся вращение, масштабирование, смещение и наклон. Преобразование осуществляется с использованием свойства ***transform***, которое имеет следующие значения:

- *scale(0.5)* – увеличивает или уменьшает в размерах элемент (например, для значения 0.5 в 2 раза);
- *rotate(45deg)* – поворачивает элемент на определенный угол заданный в deg;
- *translate(40px, 60px)* – смещает элемент из его текущей позиции на некоторое расстояние вправо на 40 px и вниз на 60 px;
- *skew(15deg)* – наклоняет элемент, смещая верхний край элемента в одну сторону, а нижний – в противоположную.

По умолчанию точкой трансформации является центр элемента, но ее можно изменить с помощью свойства ***transform-origin***. Например, ***transform-origin: 100% 0;*** устанавливает точку поворота в правом верхнем углу.

Вращения и перемещения могут производиться во всех трех измерениях – *X*, *Y* и *Z*. Значение *rotateZ* эквивалентно *rotate*, так как выполняет вращение вокруг оси *Z*. Значения *rotateX* и *rotateY* вращают элемент вокруг горизонтальной оси *X* (наклоняя элемент вперед или назад) и вертикальной оси *Y* (поворачивая элемент вправо и влево) соответственно.

Также можно задать значение перспективы, используя трансформацию со значением *perspective()* или же свойство

perspective. В качестве значения указывается расстояние от плоскости монитора до точки сходимости линий. Чем меньше значение, тем более выраженной выглядит перспектива. По умолчанию перспектива отображается так, как если бы наблюдатель находился прямо по центру элемента. Свойство ***perspective-origin*** смещает вправо или влево, вверх или вниз отображение перспективы.

Свойство ***transform-style*** позволяет располагать элементы в трехмерном пространстве, задавая значение ***preserve-3d***.

Переходы

Переход представляет собой анимацию смены одного набора свойств CSS другим за определенный промежуток времени. Переходы задаются с помощью свойства ***transition***. На рис. 7.1 приведен переход изменения цвета блока с синего на красный с использованием сокращенной записи свойства ***transition***.

```
.div:hover {  
  background-color: rgb(255,0,0);  
  transition: background-color 1s linear 0.5s;  
}  
.div {  
  background-color: rgb(0,0,255);  
}
```

Рис. 7.1. Сокращенная запись свойства ***transition***

В сокращенной записи первым указывается значение свойства ***transition-property***, которое определяет для каких свойств должен присутствовать переход. Значение ***all*** означает, что переход задается для всех свойств, которые изменяются. Вторым следует свойство ***transition-duration***, которое указывает продолжительность перехода к конечному значению, например 0.5s.

Для замедления или ускорения перехода после значения продолжительности используется свойство ***transition-timing-function*** со значениями ***linear***, ***ease-in***, ***ease-out*** и ***cubic-bezier()***:

- 1) ***linear*** – переход изменяется с постоянной скоростью;
- 2) ***ease-in*** – изменение вначале протекает медленно, но ускоряется до самого конца перехода;
- 3) ***ease-out*** – изменение начинается быстро, но к концу перехода замедляется;

4) *cubic-bezier(x1, y1, x2, y2)* – график изменения перехода по кривой Безье

Для задержки перехода последним устанавливается значение свойства ***transition-delay***, например 0.5s.

Анимация

Создание анимации проходит в два этапа:

- определение анимации, которое включает настройку ключевых кадров со списком анимируемых CSS-свойств;
- применение анимации к элементу.

Кадры определяются с помощью правила **@keyframes**. На рис. 7.2 представлен вариант создания анимации, состоящей из двух кадров.

```
@keyframes nameAnimation {  
  from {  
    /* здесь перечисляются свойства CSS */  
  }  
  to {  
    /* здесь перечисляются свойства CSS */  
  }  
}
```

Рис. 7.2. Создание кадров анимации

В блоке объявления селектора элемента, к которому применяется анимация, может быть использована расширенная запись (рис. 7.3).

```
.anime {  
  animation-name: nameAnimation;  
  animation-duration: 2s;  
  animation-timing-function: ease-in-out;  
  animation-delay: 5s;  
  animation-iteration-count: 2;  
  animation-direction: alternate;  
  animation-fill-mode: forwards; }
```

Рис. 7.3. Расширенная запись свойства ***animation***

Свойство ***animation-name*** задает имя анимации, определяемое правилом **@keyframes**, свойство ***animation-duration*** устанавливает длительность анимации, свойство ***animation-timing-***

function задает временную функцию, описывающую ускорение и/или замедление воспроизведения анимации, свойство **animation-iteration-count** определяет количество повторений анимации, свойство **animation-direction** задает обратное воспроизведение анимации при повторном воспроизведении.

Чтобы анимация была бесконечной, необходимо использовать значение *infinite* для свойства **animation-iteration-count**. Следует отметить, что для работы анимации необходимо задавать продолжительность и имя анимации.

Для того чтобы использовать анимацию перемещения элемента, представленную на рис. 7.4, следует изначально для него задать свойство **position** со значением *absolute* или *relative*. Также на рис. 7.4 показана анимация из нескольких кадров с помощью разбиения ее на несколько кадров.

```
@keyframes around {
  0% { left: 0; top: 0; }
  25% { left: 200px; top: 0; }
  50% { left: 200px; top: 200px; }
  75% { left: 0; top: 200px; }
  100% { left: 0; top: 0; }
}
p {
  position: relative;
  animation: around 4s linear infinite;
}
```

Рис. 7.4. Создание анимации из нескольких кадров

Следует отметить, что для эффектов анимации можно применять свойство **filter** со следующими значениями:

1) *grayscale*(значение) – преобразует цвета в черно-белые, и значение задается как в процентах (0 – 100%), так и в десятичных дробях (0–1);

2) *saturate*(значение) – изменяет насыщенность цвета;

3) *sepia*(значение) – создает эффект сепии, т. е. тонирование в коричневый цвет;

4) *hue-rotate*(угол) – изменяет цвета изображения в зависимости от заданного угла, который определяет, на сколько изменится данный цвет в цветовом круге от красного до фиолетового;

5) *opacity*(значение) – определяет прозрачность элемента;

6) *invert*(значение) – инвертирует цвета, т. е. изменяет цвета на противоположные;

7) *brightness*(значение) – изменяет яркость цвета;

8) *contrast*(значение) – изменяет контрастность цвета;

9) *blur*(радиус) – создает эффект размытости и значение указывается в пикселах (px).

Для расположения элементов друг над другом можно использовать для каждого из элементов свойство ***z-index***, значение которого определяет место расположения элемента.

Чтобы задать дополнительную внешнюю границу, можно применить свойство ***outline***. В отличие от ***border*** свойство ***outline*** не участвует в блочной модели CSS и не изменяет размер элемента. Поэтому его используют, когда хотят добавить рамку без изменения других CSS-параметров. Свойств ***outline-top***, ***outline-left*** не существует. Все современные браузеры также поддерживают свойство ***outline-offset***, задающее отступ ***outline*** от внешней границы элемента.

ЗАДАНИЯ

Создать новый HTML-документ `animation.html` с внешним подключением стилей, в котором должны быть элементы согласно заданиям представленным в настоящей лабораторной работе, заключенные в `<div>`.

Задание 1. Создать четыре изображения, к которым необходимо добавить следующие преобразования при наведении на них:

1. Первое изображение масштабируется в сторону уменьшения в 2 раза.

2. Второе изображение повернуть на 45 градусов.

3. Третье изображение сместить вправо на 50px.

4. При наведении на четвертое изображение оно наклоняется на -20° .

Задание 2. Создать пять кнопок, при наведении на которые они должны изменяться, согласно рис. 7.5. Для кнопок подобрать произвольное название.

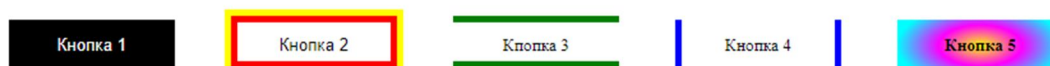


Рис. 7.5. Результат выполнения задания 4

Задание 3. Расположить произвольные изображения, согласно рис. 7.6. Значения перспективы выбрать произвольные.

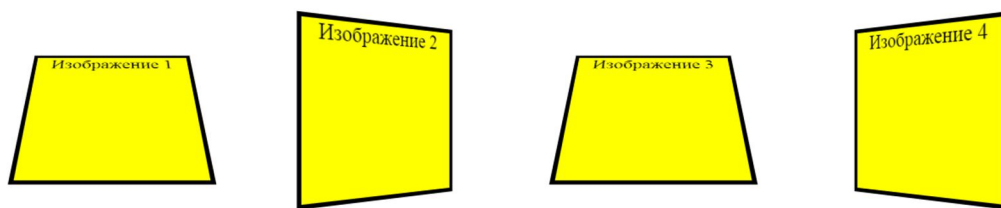


Рис. 7.6. Позиционирование изображений для задания 2

Задания 4. Создать анимацию из четырех изображений, расположение которых соответствует кадрам, представленным на рис. 7.7. В последнем кадре необходимо для второго изображения установить *blur(10px)*, третьего изображения – *grayscale(1)*, четвертого изображения – *invert(1)*.

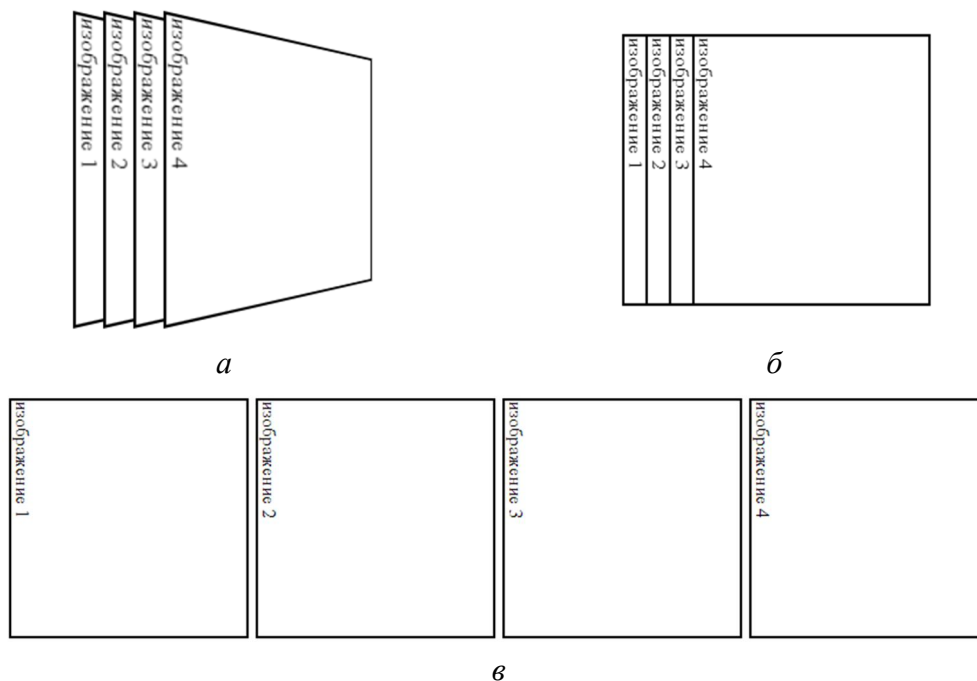


Рис. 7.7. Результат положения изображений из задания 4 в кадрах:
а – первый кадр; *б* – 50%-ный кадр; *в* – последний кадр

Задание 5. Создать анимацию непрерывной смены пяти изображений с продолжительностью 5 с с линейным ускорением. Изображения должны иметь общую границу произвольного цвета.

Задание 6. Создать из изображения пять копий и для них применить следующие эффекты:

1. Непрерывное перемещение первой копии по горизонтали на 200 px.

2. Непрерывное перемещение второй копии по прямой с изменением геометрической формы с квадрата на круг.

3. Непрерывное перемещение третьей копии по прямой с появлением слева из-за экрана и со скрыванием справа за экраном.

4. Непрерывное перемещение четвертой копии по вертикали вниз на 50px;

5. Непрерывное перемещение пятой копии по траектории квадрата, используя *left, top* для указания положения.

Задание 7. Создать плавный переход цвета текста при наведении на него в блоке размером 300 px на 300 px, выбрать произвольные цвета, время перехода установить в 5 с, переход сделать линейным. Блок должен быть максимально заполнен текстом.

Задание 8. Для каждого задания добавить заголовок, к которому применить непрерывное изменение прозрачности от 0 до 80% с обратным воспроизведением.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. С помощью какого свойства производится трансформация?
2. Как осуществить наклон?
3. Каким образом выполнить смещение?
4. Поясните, как осуществляется вращение.
5. Каким образом масштабировать элементы?
6. Что будет происходить, если использовать *rotateY*?
7. Какой будет результат от применения *rotateX*?
8. Как создать перспективу изображения?
9. Для чего используется свойство *transition*?
10. Что такое переходы?
11. Каким образом увеличить продолжительность перехода?
12. Для чего используется *transition-timing-function*?
13. Что входит в сокращенную запись свойства *transition*?
14. Назовите этапы создания анимации
15. Какие свойства включает сокращенная запись *animation*?
16. Каким образом сделать анимацию непрерывной?

17. Как изменить прозрачность элемента?
18. Каким образом создать несколько кадров анимации? только 2 кадра?
19. Для чего необходимо свойство *z-index*?
20. Чем отличаются переходы от анимации?
21. Как остановить анимацию?
22. Для чего можно применять свойство *filter*?
23. Создайте плавный переход изменения цвета круга с синего на красный при наведении.

Лабораторная работа № 8

ПАНЕЛЬ НАВИГАЦИИ

Цель работы: изучить свойства форматирования гиперссылок, получить навыки создания навигационной панели веб-сайта.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Форматирование гиперссылок

Для форматирования гиперссылок используются псевдоклассы, учитывающие их состояние:

- ***:link*** – непосещенная ссылка;
- ***:visited*** – посещенная ссылка;
- ***:hover*** – ссылка, на которой находится указатель;
- ***:active*** – ссылка в момент нажатия;
- ***:focus*** – состояние ссылки при нажатии клавиши **Tab**.

Чтобы убрать стандартное подчеркивание, применяется свойство ***text-decoration*** со значением *none*: **`a {text-decoration: none;}`**.

Вертикальная панель навигации

Для создания вертикальной панели навигации можно использовать маркированный или нумерованный список и заключается в следующих этапах:

1. Создается список со ссылками. Пример списка для панели навигации представлен на рис. 8.1.

```
<ul class="nav">
<li>
<a href="firstpage.html">Первая страница</a>
</li>
<li>
<a href="secondpage.html">Вторая страница</a>
</li>
</ul>
```

Рис. 8.1. Список панели навигации

2. Удаляются маркеры списка с помощью следующего блока объявления: **ul.nav** {*list-style-type: none;*}.

3. Элемент **<a>** преобразуются в блочный элемент, так как он является строчным элементом, посредством **ul.nav a** {*display: block;*}. Это позволяет для гиперссылок определить границы, отступы, поля.

4. Ограничивается ширина и высота элементов, поскольку ширина гиперссылки равна ширине окна браузера.

Горизонтальная панель навигации

Создание горизонтальной панели навигации можно осуществлять по следующим этапам:

1. Создается, например, маркированный список с классом **.nav**.

2. Преобразуются пункты списка в строчные (**.nav li** {*display: inline;*}) или в плавающие элементы (**.nav li** {*float: left;*}).

3. Для гиперссылок устанавливается *display: inline-block* и применяются свойства для придания внешнего оформления.

При использовании *display: inline;* справа образуется промежуток, равный 0.4 em, чтобы убрать его, необходимо установить **.nav li** {*margin-right: -4px;*}.

Панель навигации с выпадающим меню

Для создания выпадающего меню необходимо предварительно создать контейнер, используя **<nav>**, в котором должен быть список, к нужным пунктам которого добавляется вложенный список **** или ****. Пример HTML-разметки документа показан на рис. 8.2.

```
<nav class="nav">
  <ul><li><a href="1.html">Первая страница</a>
    <ul class="second">
      <li><a>1.1</a></li>
      <li><a>1.2</a></li>
    </ul>
  </li>
  <li><a href="2.html">Вторая страница</a></li>
</ul>
</nav>
```

Рис. 8.2. Пример HTML-разметки для выпадающего меню

Для позиционирования выпадающего меню относительно основного меню объявляются следующие стили:

- **nav>ul>li** {*position: relative*; } для элемента списка, в который вложено выпадающее меню;

- **nav ul** {*position: absolute*; } для выпадающего меню.

Чтобы скрыть выпадающее меню используется свойство *display* со значением *none*. При наведении курсора мыши на меню появляется выпадающий список благодаря значению *block: nav li:hover ul {display: block;}*. Также можно использовать свойство *visibility: hidden*; для сокрытия меню, а для отображения – *visibility: visible*;

Позиционирование элементов на странице

Свойство *position* определяет тип позиционирования, используемого для элемента из пяти разных значений:

- 1) *static* (статический) – означает, что расположение элементов соответствует их расположению в HTML-коде;

- 2) *relative* (относительный) – позиционируется относительно своего исходного положения;

- 3) *fixed* (фиксированный) – позиция элемента блокируется в определенной позиции на экране;

- 4) *absolute* (абсолютный) – позволяет поместить элемент в любой позиции страницы с точностью до одного пиксела относительно ближайшего позиционированного предка или окна просмотра;

- 5) *sticky* (закрепленный) – позиционируется на основе позиции прокрутки пользователя.

Использование ролловеров

Ролловеры – меню, вид которых меняется при наведении курсора мыши. Для смены одного графического вида на другой применяется метод CSS-спрайт, который использует одно изображение, включающее в себя представление различных состояний меню. Этапы реализации метода заключаются в следующем:

1. В программе редактирования изображений создается один рисунок с различными вариантами (изображение-ролловер).

2. Измеряется расстояние от верхнего края получившегося изображения до верхнего края каждого следующего изображения.

3. Создается стиль для ссылки в обычном ненажатом состоянии: **a {background: url(facultet.png) no-repeat 0 0;}**.

4. Создается стиль с псевдоклассом **:hover**: **a:hover {background-position: 0 -39px;}**. Второе значение соответствует расстоянию, измеренному в п. 2.

Чтобы для отдельного элемента панели навигации появлялось свое отображение, можно использовать классы. Пример разметки показан на рис. 8.3

```
/*CSS*/
.nav li a {
background: url(facultet.png) no-repeat 0 0;
}
.nav .it a:hover {background-position: 0 -90px;}
<!-- HTML-->
<nav class="nav">
<ul>
<li><a href="2.html">БГТУ</a></li>
<li class="it"><a href="3.html">ИТ</a></li>
</ul>
</nav>
```

Рис. 8.3. Пример использования ролловеров для конкретного элемента

ЗАДАНИЯ

Задание 1. Создать HTML-документ, который должен быть разбит на части таким образом, чтобы справа было вертикальное меню тега **<aside>**, слева – информация о свойствах CSS. Вертикальное боковое навигационное меню должно содержать гиперссылки на задания из предыдущих лабораторных работ. В меню должно быть четыре разных гиперссылки, каждая из которых соответствует одному из следующих заданий:

1. Непосещенная гиперссылка красного цвета без подчеркивания.
2. После посещения гиперссылка меняет цвет на #8B0000.
3. При наведении на гиперссылку она становится зеленой.
4. При нажатии на гиперссылку она становится цвета rgb(184, 134, 11).

Задание 2 Для копии документа лаб. раб. № 7 создать горизонтальную панель навигации представленную на рис. 8.4.

Задание 1	Задание 2	Задание 3	Задание 4	Задание 5	Задание 6	Задание 7	Задание 8
-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------

Рис. 8.4. Результат выполнения задания 2

Гиперссылки должны ссылаться на заголовок каждого из заданий, для каждого заголовка удалить непрерывное изменение прозрачности.

Задание 3. Создать новый HTML-документ, в котором создать панель навигации с выпадающими элементами (рис. 8.5).



Рис. 8.5. Результат выполнения задания 3

Подпункты должны быть со гиперссылками согласно их названиям на задания из лабораторных работ № 4 и 5.

Задание 4. Создать горизонтальную панель навигации из четырех элементов, представленную на рис. 8.6. Использовать из папки *labs* изображение-ролlover *facultet.png*.

При наведении курсора на названия факультетов должен появляться его логотип, а для факультета ИТ сделать выпадающее меню с кафедрами, относящимися к факультету. Также должен быть переход на веб-сайты факультета и кафедр. Подобрать произвольные цвета текста и фона. Пример выполнения задания показан на рис. 8.6.



Рис. 8.6 Пример выполнения задания 4

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. С помощью каких псевдоклассов можно изменить состояние гиперссылки?
2. Для чего используется псевдокласс *:hover*?
3. Каково назначение псевдокласса *:active*?
4. Для чего используется псевдокласс *:link*?
5. Каково назначение псевдокласса *:visited*?
6. Как создать вертикальную панель навигации?
7. Перечислите основные этапы создания горизонтальной панели навигации.
8. Как создать панель навигации с выпадающими элементами?
9. Для чего используется метод CSS-спрайт?
10. Что представляет из себя изображение-ролlover?
11. Каким образом скрыть выпадающие элементы?
12. Для чего используется свойство *position*?
13. Какое значение позиционирования должно быть у выпадающего меню и основного меню?
14. Создайте вертикальное меню из двух элементов, для одного из которых должно справа появляться выпадающее вертикальное меню из трех элементов.
15. Какие типы позиционирования вы знаете?
16. Что такое относительное позиционирование?
17. Раскройте суть абсолютного позиционирования.
18. Что такое статическое позиционирование?
19. Раскройте суть фиксированного позиционирования
20. Создайте горизонтальное меню из основных элементов, для одного из которых должен быть выпадающий список при наведении на него, состоящий из трех элементов.
21. Что такое «липкое» позиционирование?

Лабораторная работа № 9

СИСТЕМА МОДУЛЬНОЙ ВЕРСТКИ

Цель работы: изучить веб-верстку с использованием модульной сетки, научиться составлять модульные сетки и использовать свойства grid-верстки.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Понятие модульной сетки

Модульная сетка определяет двумерную разметку колонок и строк, в которые можно поместить элементы. Страница по ширине делится на определенное количество столбцов модульной сетки, которые группируются для создания колонок контейнера. Некоторые элементы заполняют только одну ячейку сетки, а другие способны расширяться и размещаться в соседних колонках и строках.

Система модульной верстки основана на создании следующих элементов:

- контейнера **<div>**, который содержит одну или несколько строк;
- строки, которая является **<div>** элементом, помещенным в контейнер;
- колонки, которая определяется элементами **<div>** в строке.

Система модульной верстки Skeleton

Система Skeleton – адаптивная система модульной верстки веб-страниц. Система позволяет сверстать сетчатый макет, структурировав контент страницы в несколько колонок на планшетных устройствах, ноутбуках и компьютерах. На смартфоне содержимое страницы преобразуется в одноколоночный контент.

Чтобы начать работу, необходимо посетить сайт *getskeleton.com* и нажать кнопку **Download**. Загруженный архив содержит несколько папок и файлов. В каталоге *css* расположены необходимые файлы *normalize.css*, которые сбрасывают базовые стили CSS,

чтобы браузеры отображали HTML-элементы одинаково, и файл *skeleton.css*, содержащий набор стилей для компоновки макета с использованием модульной сетки.

Система Skeleton основана на модульной сетке, состоящей из 12 столбцов, поэтому каждый из добавленных элементов **<div>** должен быть не менее одного или не более 12 (полная ширина контейнера) столбцов в ширину. Например, чтобы создать три равные по ширине колонки, можно добавить три элемента **<div>**, согласно рис. 9.1

```
<div class="container">
  <div class="row">
    <div class="four columns">
    </div>
    <div class="four columns">
    </div>
    <div class="four columns">
    </div>
  </div>
</div>
```

Рис. 9.1. Разбиение страницы на три колонки

Таким образом, система модульной верстки реализуется следующим образом:

1. Подключаются CSS-файлы *normalize.css* и *skeleton.css*.
2. Добавляются контейнеры **<div>**.
3. Добавляются контейнеры **<div>** для строк.
4. Добавляются контейнеры **<div>** для колонок.
5. Добавляется контент в элементы **<div>** колонок.
6. Создаются собственные стили, например *custom.css*.

Использование Grid-верстки

Grid-верстка – это разбиение макета на области. Сетка (grid) представляет собой совокупность пересекающихся горизонтальных и вертикальных линий, разделяющих пространство grid-контейнера на области сетки, в которые может быть помещено содержимое элементов сетки. Контейнер ведет себя как блочный элемент, заполняя 100% доступной ширины. На рис. 9.2 представлены компоненты сетки, к которым относятся ячейка, область сет-

ки, полоса (колонка или строка) сетки, вертикальные и горизонтальные линии сетки. Полосой сетки является расстояние между двумя соседними линиями сетки. Ячейка сетки – пространство в сетке, где пересекаются горизонтальная и вертикальная полосы сетки. Одна или несколько ячеек составляет область сетки.

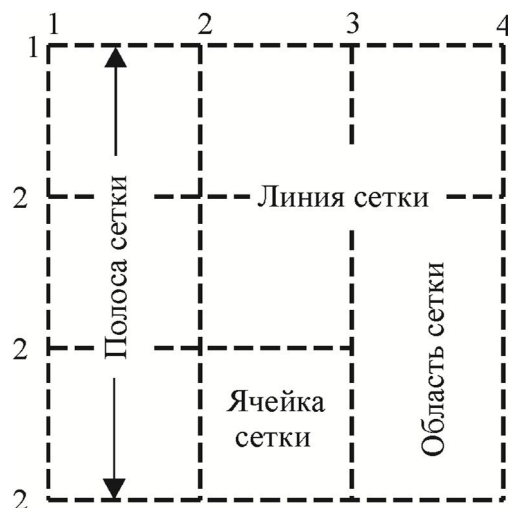


Рис. 9.2. Компоненты grid-верстки

Предварительно необходимо создать grid-контейнеры с помощью ***display: grid*** или ***display: inline-grid*** (рис. 9.3).

```
<div class="grid"> .grid {
  <div>1</div>      display: grid;
  <div>2</div>      grid-template-columns: 40px
  <div>3</div>      60px 40px;
  <div>4</div>      grid-template-rows: 10px
  <div>5</div>      10px;
  <div>6</div>      grid-gap: 5px;
</div>             }
```

Рис. 9.3. Создание grid-контейнера

Например, на рис. 9.3 элемент с классом **grid** становится контейнером сетки, его потомки – элементами сетки и он занимает 100% доступной ширины. Свойства ***grid-template-columns*** и ***grid-template-rows*** устанавливают размер каждой колонки и каждой строки сетки. Для выравнивания текстовой информации по центру горизонтально в ячейках сетки необходимо применить ***text-align: center***;; а для вертикального выравнивания использовать ***padding***.

Для создания сетки можно использовать свойство ***grid-template-areas***, значениями которого являются имена ячеек сетки, задаваемые с помощью свойства ***grid-area***.

```
<div class="grid">
  <div
    style="grid-area: x;">1</div>
  <div
    style="grid-area: y;">2</div>
  <div
    style="grid-area: w;">3</div>
  <div
    style="grid-area: x2;">4</div>
  <div
    style="grid-area: y2;">5</div>
</div>

.grid {
  display:grid;
  grid-template-areas:
    "x y w"
    "x2 y2 w2";
  grid-template-columns:
    20px 20px 20px;
  grid-template-rows:
    20px 20px 20px;
  grid-gap: 5px;
}
```

Рис. 9.4 Создание сетки с помощью ***grid-template-areas***

Чтобы установить расстояния между колонками или строками, используются свойства ***grid-column-gap*** и ***grid-row-gap***. Следует отметить, что расстояния между колонками или строками не могут быть разными.

Для определения области сетки для одной ячейки можно установить свойства ***grid-column-start*** (***grid-column-end***) и ***grid-row-start*** (***grid-row-end***), значениями которых является номер линии сетки. Можно использовать сокращенную запись ***grid-row: n/m;*** (***grid-column: n/m;***), где *n* является номером линии, с которой начинается область сетки, а *m* определяет линию, где она заканчивается. Результат применения свойств ***grid-row: 2/3;*** и ***grid-column: 2/4;*** показан на рис. 9.5.



Рис. 9.5 Пример использования ***grid-row*** и ***grid-column***

ЗАДАНИЯ

Задание 1. Создать два HTML-документа, согласно макетам, представленным на рис. 9.6, с использованием элементов grid-верстки и добавить в соответствующие ячейки и заголовок информацию на тему «HTML5 и CSS3». Элементы занимают видимую область экрана браузера.

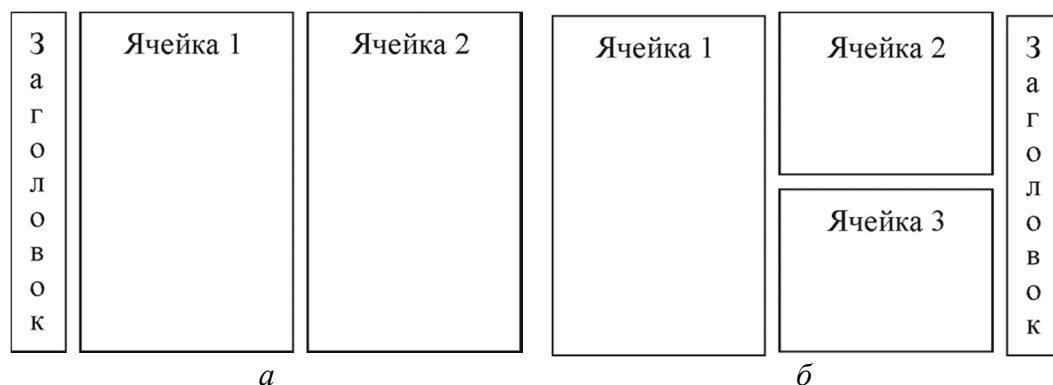


Рис 9.6. Макеты для создания веб-страниц:
а – первая веб-страница, б – вторая веб-страница

Задание 2. Создать веб-страницу, согласно рис. 9.7. В ячейку 9 необходимо добавить gif-изображение, а ячейка 10 должна содержать фото с обтеканием текстом. Из остальных ячеек сделать фотогалерею с названиями для каждой фотографии. Элементы занимают видимую область экрана браузера.

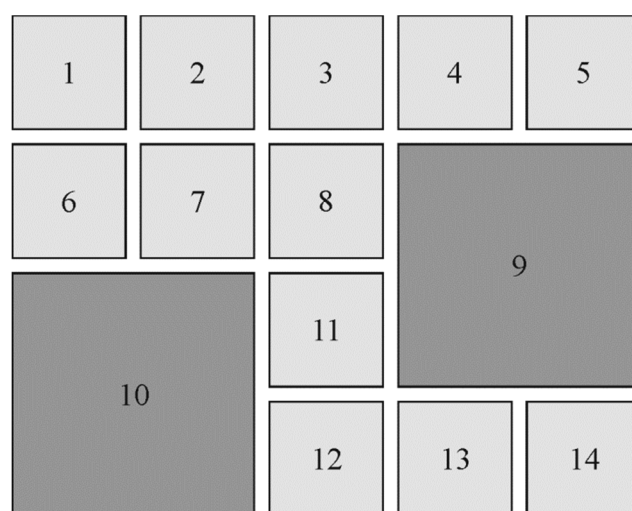


Рис. 9.7. Макет сетки для задания 2

Задание 3. Создать два HTML-документа, соответствующих макетам (рис. 9.8), заполнить их произвольной информацией на тему «Grid-верстка». В меню *nav* должны быть гиперссылки на ресурсы, содержащие информацию по HTML5, CSS3 и grid-верстке. Для разметки использовать свойство *grid-template-areas*. В «подвале» веб-страницы должна быть размещена информация о студенте, выполнявшем задание. Элементы занимают видимую область экрана браузера.

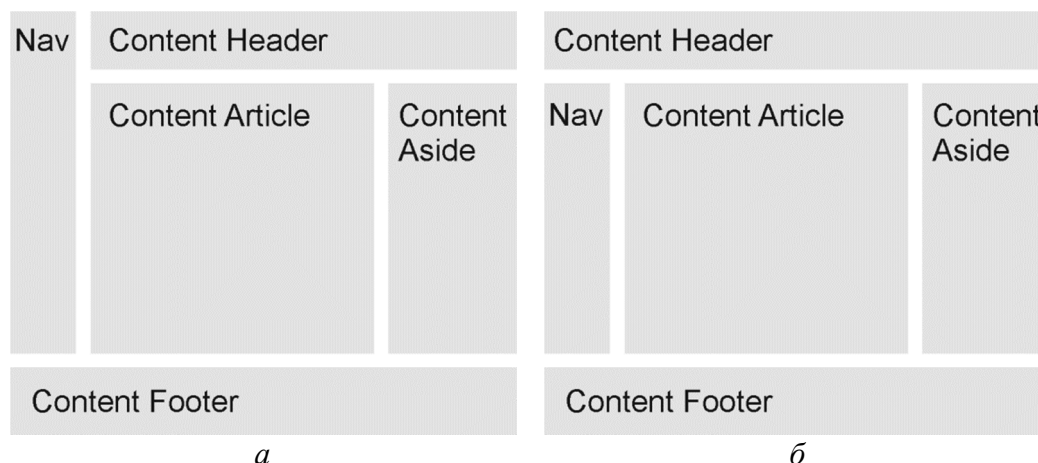


Рис 9.8. Макеты для задания 3:
а – первый документ; б – второй документ

Задание 4. Используя файлы системы модульной верстки Skeleton создать HTML-документ следующим образом:

1. Создать в первой строке два столбца, в первом столбце разместить логотип факультета, во втором – горизонтальное навигационное меню из гиперссылок на страницы предыдущих заданий.
2. Создать во второй строке четыре столбца с заголовками.
3. Создать в третьей строке четыре столбца, соответствующие заголовкам п. 2, с информацией о свойствах grid-верстки.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет из себя модульная сетка?
2. Поясните суть системы модульной верстки Skeleton.
3. Каким образом можно использовать систему Skeleton?

4. Что означают стили в файле *skeleton.css*?
5. Назовите этапы реализации системы модульной верстки Skeleton.
6. Что такое grid-верстка?
7. Для чего необходимо свойство *grid-template-areas*?
8. С какой целью используется свойство *grid-template-columns*?
9. Для чего применяется свойство *grid-area*?
10. С какой целью используется свойство *grid-row*?
11. Что означает *grid-template-columns: 20px 20px 20px*;
12. Поясните запись *grid-row-end: 4*;
13. Для чего используется свойство *grid-gap*?
14. Создайте документ, содержащий три колонки и три строки, вторую строку сделайте красного цвета, в первую добавьте абзац синего цвета.
15. С какой целью применяется свойства *align-self* и *justify-self*?

Лабораторная работа № 10

АДАПТИВНЫЙ ВЕБ-ДИЗАЙН. FLEXBOX-ВЕРСТКА

Цель работы: изучить принципы адаптивного веб-дизайна и использование медиазапросов, получить навыки применения flexbox-верстки.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Понятие «адаптивный веб-дизайн». Медиазапросы

Адаптивный веб-дизайн позволяет изменять структуру веб-страницы на основе ширины окна браузера на различных устройствах (планшеты, смартфоны). Для его осуществления используются гибкие сетки, гибкие изображения и медиазапросы CSS, предназначенные для создания различных стилей для экранов с разным решением. Существует несколько методов по созданию адаптивного веб-дизайна:

- Метод **Mobile First**, который заключается в разработке мобильной версии до создания настольной версии;
- Правило **@media**, с помощью которого стили адаптируются под разные области просмотра;
- Использование резиновых макетов, которые допускают масштабирование контейнеров в зависимости от ширины области просмотра.

Для отображения содержимого страницы на экране смартфона можно использовать следующий метатег `<meta name="viewport" content="width=device-width">`. Браузеры не уменьшают масштаб, а настраивают ширину экрана на текущее разрешение по горизонтали экрана смартфона. В мобильной версии часто используется значок в виде трех горизонтальных линий, который называется «гамбургером». Для вставки этого символа в стандарте Юникод предусмотрено введение для HTML **≡** и для CSS **\2261**.

Медиазапросы назначают страницам стили на основе размера окна браузера. С помощью них можно создавать пользовательские

стили для браузеров смартфонов, планшетов и компьютеров и производить настройку отображения сайта на экране каждого типа устройств. Медиазапрос с помощью правила **@media** выглядит следующим образом (рис. 10.1).

```
@media (min-width: 480px) {  
  div > h1 { font-size: 2.25rem;}  
}
```

Рис. 10.1. Пример синтаксиса медиазапроса

В данном случае браузер проверяет, выполняется ли условие **min-width: 480px**; Если размер экрана больше или равен 560 px, то применяются заданные стили. Также можно использовать атрибут **media** в HTML-коде **<link href="css/small.css" rel="stylesheet" media="(max-width: 480px)">**. В этом случае стиль **small.css** будет применяться, если разрешение не более 480 px.

Правило **@import** также можно использовать для создания медиазапросов. Оно должно помещаться в начало таблицы стилей и подключаться в соответствии с рис. 10.2.

```
@import url(css/base.css); /* без медиазапроса */  
@import url(css/medium.css) (min-width:560px) and  
(max-width:760);  
@import url(css/small.css) (max-width: 480px);
```

Рис. 10.2 Пример использования правила **@import**

Основным компонентом адаптивного дизайна считаются гибкие сетки, которые являются частью резинового дизайна. Применение гибких сеток заключается в установке процентного значения ширины вместо абсолютного в пикселах.

Различные ширины экрана, которые указываются с помощью медиазапросов, называются точками останова. Стандартными принято считать следующие разрешения: для мобильных устройств – 480 px, для планшетных компьютеров – 768 px, для нетбуков – 1024 px, для компьютеров – 1280 px и больше.

Flexbox-верстка

Flexbox-верстка – разметка макетов веб-страниц, которая позволяет автоматически настраивать ширину элементов, находящихся

ся внутри flex-контейнера. Для применения данного способа разметки следует учитывать разделение применяемых свойств на свойства flex-контейнера и свойства flex-элемента.

С помощью *display: flex*; осуществляется преобразование HTML-элемента в flex-контейнер, а элементы, находящиеся внутри него, в flex-элементы. По умолчанию flex-элементы помещаются друг за другом в одной строке. Свойство *flex-flow* позволяет выбрать направление отображения элементов в контейнере, а также указать их перенос на следующую строку. Первое значение определяет направление, а второе – перенос на следующую строку.

Свойство *flex-flow* является сокращенной записью и содержит значения свойств *flex-direction* и *flex-wrap*. Flex-элементы могут размещаться в строке (значение *row*) или в колонке (значение *column*). Перенос элементов осуществляется с помощью значения *wrap*.

Свойство *justify-content* определяет способ выключки flex-элементов в строке. Для выравнивания по левому краю используется значение *flex-start*, а по правому краю – *flex-end*. Для равномерного распределения элементов, за счет создания пространства между ними необходимо применить значение *space-between*, а значение *space-around* добавляет поля по левому и правому краям крайних элементов.

Свойство *align-items* определяет, как flex-элементы различной высоты будут выровнены по высоте строки в flex-контейнере. Для выравнивания по верхнему краю используется значение *flex-start*, а по нижнему краю – *flex-end*. Значение *stretch* позволяет растянуть каждый элемент по высоте контейнера, делая высоты элементов одинаковыми. Свойство *align-content* определяет, как будут размещены flex-элементы, занимающие несколько строк.

Для flex-элементов основным свойством является *flex*, которое обеспечивает их гибкость и управляет шириной, что позволяет создавать «гибкие» колонки или изменять их ширину в соответствии с размером контейнера, даже если размер неизвестен или меняется динамически. Первое значение свойства *flex* – число свойства *flex-grow*, которое указывает на относительную ширину flex-элемента, определяющую, во сколько раз размеры элементов отличаются между собой. Второе значение – число свойства *flex-shrink*, которое определяет, насколько flex-элемент может быть сжат, если суммарная ширина элементов больше ширины контейнера. Последнее значение – свойство *flex-basis*, которое определяет базовую

вую ширину flex-элемента. Пример кода с возможными свойствами показан в таблице.

Пример flex-верстки

Структура страницы	Значения flex-контейнера	Значения flex-элемента
<code><div class="container"> <div class="div1"></div> <div></div> </div></code>	<code>.container { display: flex; flex-flow: row wrap; align-content: space-between; align-items: flex-start; justify-content: center;} </code>	<code>.div1 { order: 1; align-self: flex-end; flex: 1 1 300px; } </code>

ЗАДАНИЯ

Задание 1. Создать макет приведенный на рис. 10.3, с использованием свойств flexbox-верстки, сделав предварительно копию. В ячейку 1 добавить три видео файла, в ячейке 2 должна быть информация на тему «Flexbox-верстка», в ячейке 3 должно быть три аудиофайла.

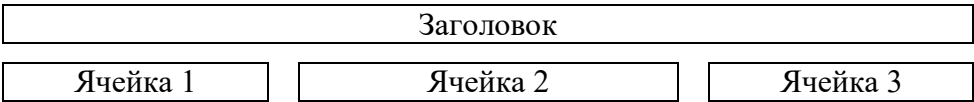


Рис. 10.3. Макет для задания 1

Задание 2. В этом же документе создать горизонтальное меню из четырех гиперссылок после заголовка, которое в мобильной версии преобразуется в меню «гамбургер». Вид мобильной версии представлен на рис. 10.4.

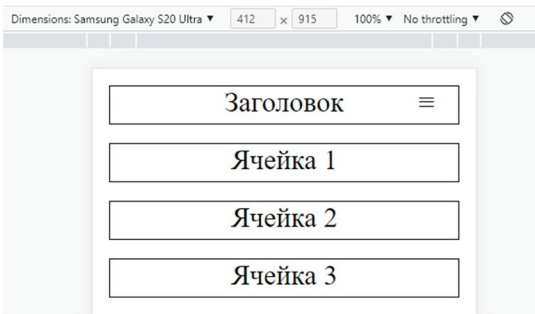


Рис. 10.4. Вид мобильной версии для задания 2

Самостоятельно подобрать ширину экрана и способ подключения для медиазапроса.

Задание 3. В документе к заданию 1 добавить фотогалерею из 8 фотографий с надписью и продемонстрировать подключение разными способами медиазапросов, которые будут изменять цвет шрифта надписи в зависимости от размера экрана: 480 px – красный, внутреннее подключение, 768 px – синий, использовать правило **@import**, 1280 px – зеленый, через тег **<link>**. Также следует для размера экранов меньше 1280 px установить размер области, занимаемой одной фотографией, равной ширине экрана, и расположение фотографий сделать вертикальным.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Раскройте суть фиксированной верстки веб-страницы?
2. Что такое резиновый макет веб-страницы?
3. Дайте определение понятия «адаптивный дизайн»
4. Что такое медиазапросы? Как подключить медиазапросы?
5. Для чего используется правило **@import**?
6. С какой целью применяется правило **@media**?
7. Для чего предназначено свойство **flex**?
8. Что означает **@media (min-width: 560px)**?
9. Расшифруйте запись **@media (max-width: 960px)**.
10. Что означает **flex: 1 2 200px**;
11. Для чего предназначено значение **wrap**?
12. С какой целью используется свойство **flex-flow**?
13. Для чего применяется свойство **flex-direction**?
14. С какой целью используются свойства **align-items** и свойство **align-content**?
15. Какие значения имеет свойство **align-self**?
16. Назовите свойства, которые имеет flex-контейнер.
17. Какими свойствами обладают flex-элементы?
18. Для чего предназначено свойство **justify-content**?
19. Как вставить видео? Каким образом вставить аудио?
20. Чем отличается свойство **align-items** от свойства **align-self**?
21. Как создать адаптивное меню?
22. Создайте документ, в котором цвет абзаца изменяется на красный при изменении ширины до 768 px.

Лабораторная работа № 11

ПРЕПРОЦЕССОР SCSS

Цель работы: изучить принципы метаязыка Sass, научиться использовать функциональные возможности препроцессора SCSS.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Метаязык Sass

Sass (Syntactically Awesome Stylesheets) – препроцессор CSS, который представляет собой метаязык на основе CSS, предназначенный для расширения возможностей и сокращения записи CSS. Препроцессор – программа, имеющая собственный синтаксис, который затем компилируется в стандартный код другой программы. Кроме Sass, существуют такие препроцессоры, как Stylus, PostCSS, SCSS, Less.

Для того чтобы использовать Sass необходимо установить программное обеспечение среды разработки языка программирования Ruby, а также основные файлы Sass. Загрузить установочный пакет Ruby можно с веб-сайта rubyinstaller.org/downloads. В редакторе Visual Studio Code для использования Sass необходимо установить расширение Live Sass Compiler и изменить параметры (рис. 11.1), предварительно выбрав **Ctrl + Shift + P > Open Workspace Setting(JSON)**

```
{
  "liveSassCompile.settings.formats": [{
    "format": "expanded",
    "extensionName": ".css",
    "savePath": "/css",
  }],
}
```

Рис. 11.1. Настройки параметров Live Sass Compiler

На рис. 11.1 в `extensionName` указано только расширение, что означает что имя файла соответствует имени файла с расширением `.scss`.

В `savePath` указывается путь сохранения файла с расширением `.css`. После создания файла с расширением `.scss` следует нажать кнопку *Watch Sass* в нижней панели.

При использовании Sass создается файл с расширением `.scss`, который пользователь редактирует, и файл, который сгенерирует препроцессор Sass, т. е. файлы с расширением `.css`. Таким образом, наиболее распространенный способ организовать их расположение в корневом каталоге веб-сайта заключается в создании папки `css`, которая используется для хранения CSS-файлов, скомпилированных с помощью Sass, и другой папки `scss` – для файлов с расширением `.scss`.

Синтаксис SCSS

В синтаксисе SCSS используются переменные, вложенные правила, миксины, встроенный импорт, функции и операторы, директивы.

Создание переменной осуществляется по следующему синтаксису: **Имя переменной: значение;**. Имя переменной начинается с символа доллара и может содержать латинские символы, цифры, дефис и подчеркивание. Регистр букв имеет значение. После двоеточия указывается значение переменной, а в конце ставится точка с запятой. Пример создания переменных и их использования представлен на рис. 11.2

<pre>\$white: #ffffff; \$black: #000000; body { color: \$white; background-color: \$black; }</pre>	<pre>body { color: #ffffff; background-color: #000000; }</pre>
<i>a</i>	<i>б</i>

Рис. 11.2. Пример использования переменных в SCSS:
a – создание переменных; *б* – результат компиляции

Переменные могут быть глобальными, если созданы вне блоков объявлений, и локальными, если будут созданы внутри блока объявления. Переменную можно подставить в комментарий, строку, селектор, используя следующую форму записи: **#{\$имя переменной}**.

В большинстве случаев при написании CSS селекторы часто дублируются. Для избежания дублирования используются вложенные правила или атрибут амперсанда **&** (рис. 11.3).

<pre> \$red: #f50b0b; \$blue: #630dec; div { background: \$red; p { color: \$blue; } } </pre>	<pre> div { background: #f50b0b; } div p { color: #630dec; } </pre>
<i>а</i>	<i>б</i>
<pre> \$red: #f50b0b; \$blue: #630dec; div { background: \$red; &:hover { background: \$blue; } } </pre>	<pre> div { background: #f50b0b; } div:hover { background: #630dec; } </pre>
<i>в</i>	<i>г</i>

Рис. 11.3. Пример использования вложенных правил SCSS:
а, б – вложенность в SCSS; *в, г* – результат в CSS

Для создания определений стилей, которые можно повторно использовать, предназначены шаблоны и миксины (примеси). Миксины, в отличие от шаблонов, могут содержать параметры, что позволяет настраивать значения стиля или их генерировать. Вместо селектора можно указать **%название**. Такой вид называется шаблонным селектором или селектором-заполнителем. Чтобы вставить шаблон, нужно воспользоваться директивой **@extend**. Шаблонный селектор можно указать в составе обычных селекторов. Миксины позволяют вставить одно или несколько правил стиля вместо директивы **@include** с названием миксина сколько угодно раз. Для создания миксина используется директива **@mixin**. После директивы приводится название миксина и внутри круглых скобок могут указываться параметры, которые являются локальными переменными. Если миксин не содержит параметров, то можно добавить пустые круглые скобки или вообще их не до-

бавлять. На рис. 11.4 приведен пример использования шаблонов и миксинов.

<pre> %red-color { color: red; } p { @extend %red-color; } h1 { @extend %red-color; } </pre> <p style="text-align: center;"><i>а</i></p> <pre> @mixin flex { display: flex; justify-content: space-between; } .header { @include flex; } .container { @include flex; } </pre> <p style="text-align: center;"><i>б</i></p>	<pre> h1, p { color: red; } </pre> <p style="text-align: center;"><i>б</i></p> <pre> .header { display: flex; justify-content: space- between; } .container { display: flex; justify-content: space- between; } </pre> <p style="text-align: center;"><i>в</i></p>
---	--

Рис. 11.4. Пример использования шаблонов и миксинов:
а, б – шаблоны и миксины в Sass; *б, в* – результат в CSS

В Sass можно создавать фрагменты кода, которые можно вызывать неоднократно из любого места программы. Эти фрагменты кода называются функциями. Пример функции и ее вызов представлен на рис. 11.5.

```

@function max-width() {
    @return 1000px;
}
$value: max-width();
div { width: 500px / max-width() * 100%; }

```

Рис. 11.5. Пример использования функции

Определение пользовательской функции начинается с директивы **@function**, после которой указывается имя функции с круг-

лыми скобками. В круглых скобках могут быть приведены параметры функции, которые являются локальными переменными. Функция всегда что-то возвращает, и возвращаемое значение указывается после оператора **@return**.

В функции можно применить логические, арифметические, условные операторы и циклы.

Оператор условия **@if** позволяет в зависимости от значения логического выражения исполнить блок программы. Для цикла используется оператор **@for**, чтобы выполнить блок программы определенное число раз. Пример использования операторов цикла и условия приведен на рис. 11.6.

<pre> @mixин p- color(\$height,\$width) { @if ((\$height<\$width) and (\$width >= 30px)) { color:red; } @else { color:blue; } } .class { @include p-color(10px,30px) } </pre>	<pre> .class { color: red; } </pre>
<i>а</i>	<i>б</i>
<pre> @for \$i from 1 through 3 { .class-#{ \$i } { width: 10px*\$i; } } </pre>	<pre> .class-1 {width: 10px;} .class-2 {width: 20px;} .class-3 {width: 30px;} </pre>
<i>в</i>	<i>г</i>
<pre> \$index: 5; @while \$index > 0 { .class-#{ \$index } { width: 10px * \$index; } \$index: \$index - 1; } </pre>	<pre> .class-5 {width: 50px;} .class-4 {width: 40px;} .class-3 {width: 30px;} .class-2 {width: 20px;} .class-1 {width: 10px;} </pre>
<i>д</i>	<i>е</i>

Рис. 11.6. Пример использования операторов цикла и условия:
а, в, д – циклы и условия в Sass; *б, г, е* – результат в CSS

Производить арифметические операции позволяют операторы сложения (+), вычитания (–), унарного минуса, умножения (*), де-

ления (/), остатка от деления (%). Чтобы выполнить деление, необходимо либо заключить выражение в круглые скобки, либо сохранить результат в переменной, либо вернуть результат из функции. Приоритет выполнения операции умножения выше, затем следуют операции деления и сложения. Изменить порядок вычисления выражения можно с помощью круглых скобок. Операции, заключенные в них, имеют наивысший приоритет.

К операторам сравнения относятся равно (==), не равно (!=), меньше (<), больше (>), меньше или равно (<=), больше или равно (=>). Два значения равны, если имеют одинаковый тип данных и значение. Числа равны, если равны их значения и единицы измерения.

Директива @import

Sass-правила можно разделить на несколько файлов, а затем объединить в один файл. В Sass эти файлы называются фрагментами. Чтобы сообщить препроцессору, что необходимо преобразовать эти фрагменты в отдельные CSS-файлы, их имена должны начинаться с символа нижнего подчеркивания. Чтобы скомпилировать определенный фрагмент, используется директива **@import**.

Наиболее распространенный способ заключается в том, чтобы подготовить один Sass-файл, который будет преобразован в законченный CSS-файл и который не содержит ничего, кроме нескольких директив импорта. Содержимое Sass-файла может выглядеть следующим образом (рис. 11.7).

```
@import '_value.scss';  
@import '_flex.scss';  
@import '_grid.scss';
```

Рис. 11.7. Пример использования @import

При импорте фрагментов в другой Sass-файл нижнего подчеркивания () и расширение .scss можно опустить.

ЗАДАНИЯ

Задание 1. Создать веб-страницу, согласно прототипу, представленному на рис. 11.8. Для файлов с расширениями .css и .scss

должны быть соответствующие им отдельные папки. Переменные должны храниться в отдельном файле.

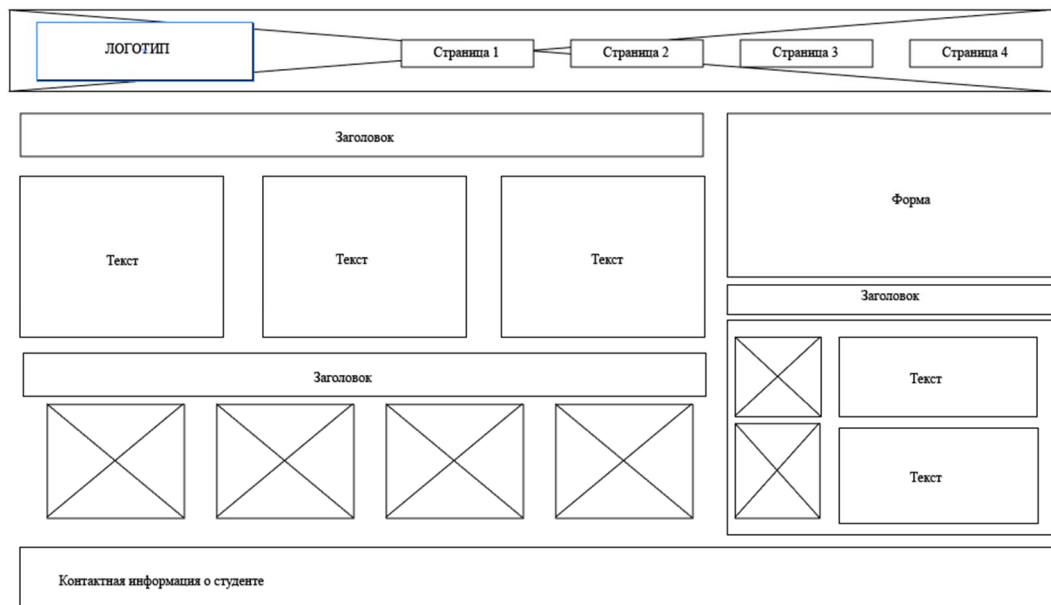


Рис. 11.8. Прототип веб-страницы к заданию 1

Задание 2. Для веб-страницы применить вложенные правила при создании навигационного меню. При оформлении формы использовать шаблонные селекторы и миксины. Также их можно применить и для других элементов.

Задание 3. Создать функции, которые устанавливают зависимость между размером шрифта заголовка и текста после этого заголовка, а также функции с наличием условных и логических операторов.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое Sass? Как расшифровывается Sass?
2. Как установить Sass?
3. Что такое препроцессор SCSS?
4. Как создаются переменные в SCSS?
5. На какие виды подразделяются переменные? В чем между ними разница?
6. Для чего используется **@import** в препроцессоре SCSS?
7. Что такое шаблонные селекторы?

8. Раскройте суть понятия «миксин». Чем они отличаются от шаблонных селекторов?
9. Для чего предназначена директива **@mixin**?
10. Каким образом создать функции в Sass?
11. Назовите операторы в Sass, которые вы знаете.
12. Создайте файл SCSS с условием, которое заключается в том, что для текста размером 20 px цвет будет красный.
13. Какие операторы относятся к арифметическим?
14. Назовите операторы, которые относятся к операторам сравнения.
15. Перечислите директивы цикла и условия
16. Создайте файл SCSS с переменными и вложенными правилами и примените стили к элементам на странице.

Лабораторная работа № 12

СОЗДАНИЕ И ВАЛИДАЦИЯ XML-ДОКУМЕНТОВ

Цель работы: изучить правила создания XML-документов, получить навыки создания валидных документов и их проверки.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Основы XML

XML (eXtensible Markup Language) – расширяемый язык разметки, созданный для хранения, транспортировки и обмена данными. Он включает в себя элементы SGML и предназначен для определения HTML-подобных языков. В нем отсутствуют элементы SGML, не применимые к языкам типа HTML, а другие элементы упрощены, чтобы облегчить их понимание и использование. В языке XML нет predefined тегов, автор определяет свои языковые теги и свою структуру документа. Формально XML представляет собой набор правил для создания собственных языков разметки, а также чтения и написания документов на языке разметки. Пример XML-документа представлен на рис. 12.1.

```
<?xml version="1.0" encoding="UTF-8"?>
  <note>
    <to>Анна</to>
    <from>Дмитрий</from>
    <heading>Напоминание</heading>
    <body>Не забудь обо мне в эти выходные!</body>
  </note>
```

Рис. 12.1. Пример структуры XML-документа

Согласно примеру, XML-документ состоит из элементов, которые представляют собой контент между начальным и конечным тегом. Первые строки называются прологом и объявляют процес-

сору XML, что данный документ размечен в XML. Пролог включает объявление (XML-декларация), которое начинается с разделителя, состоящего из пяти символов **<?xml**, за которым следует некоторое количество определений свойств, каждое из которых состоит из имени свойства и его значения, заключенного в кавычки. Объявление завершается закрывающимся разделителем из двух символов **?>**. Варианты объявлений приведены на рис. 12.2

```
<?xml version="1.0"?>
<?xml          version='1.0'          encoding='UTF-8'
standalone='yes'?>
<?xml version = '1.0' encoding='UTF-8' standalone
="no"?>
```

Рис. 12.2. Варианты объявлений XML-документа

На рис. 12.1 элемент документа **<note>** является корневым. Следующие 4 строки описывают дочерние элементы корневого элемента: **<to>**, **<from>**, **<heading>**, **<body>**. Последняя строка определяет конец корневого элемента **</note>**. В общем виде структуру XML-документа после пролога можно представить в следующем виде (рис. 12.3).

```
<корневой>
  <потомок>
    <подпотомок> . . . . . </подпотомок>
  </потомок>
</корневой>
```

Рис. 12.3. Структура XML-документа

При открытии в браузерах, например Mozilla или Opera, будет отображаться XML-документ согласно рис. 12.4.

С этим XML-файлом не связана ни одна таблица стилей. Ниже показано дерево элементов.

```
<note>
  <to>Анна</to>
  <from>Дмитрий</from>
  <heading>Напоминание</heading>
  <body>Не забудь обо мне в эти выходные!</body>
</note>
```

Рис. 12.4 Пример XML-документа в браузере

Таким образом, необходимо подключение таблицы стилей для создания конечного продукта после объявления XML-документа. Пример XML-документа с подключением CSS приведен на рис. 12.5.

<pre><?xml version="1.0" encoding="UTF-8"?> <?xml-stylesheet type="text/css" href="forXML.css"?> <books> <head> <author>Автор</author> <ntitle>Название книги </ntitle> </head> <book> <author>С.В.Одиночкина </author> <title>Основы технологии XML</title> </book> <book> <author>Эрик Рэй</author> <title>Изучаем XML</title> </book> </books></pre>	<pre>books { display: block; border-collapse: collapse; margin: 10px; } book, head { display: table-row; } ntitle, nauthor { display: table-cell; border: 1px solid black; padding: 5px; text-align: center; background: #ffff00; } title, author { display: table-cell; border: 1px solid black; padding: 5px; }</pre>
<i>a</i>	<i>б</i>

Автор	Название книги
С.В. Одиночкина	Основы технологий XML
Эрик Рэй	Изучаем XML

в

Рис. 12.5. Пример оформления XML-документа:

a – XML-документ; *б* – CSS-файл;

в – результат в браузере с подключением CSS

Также следует различать синтаксически верный документ и валидные документы. К синтаксически верным (корректным) от-

носятся XML-документы, которые соответствуют следующим правилам синтаксиса XML:

1. Все XML элементы должны иметь закрывающий тег.
2. Теги XML являются регистрозависимыми.
3. Перед закрывающейся угловой скобкой в пустых элементах XML требуется ставить косую черту.
4. Значения должны быть заключены в одинарные или двойные кавычки.
5. Все элементы обязаны соблюдать корректную вложенность.
6. XML-документ должен содержать один корневой элемент, который будет родительским для всех других элементов.
7. Учитываются все символы форматирования (пробелы, переводы строк, табуляции не игнорируются как в HTML).

Имена элементов могут начинаться только с букв и символов подчеркивания и могут содержать только буквы, цифры, дефисы, точки и символы подчеркивания и не могут начинаться с сочетания «xml». В качестве имен можно использовать любые слова, зарезервированных слов нет.

Валидные XML-документы

Валидный XML-документ должен быть синтаксически верным и соответствовать одному из типов определения документов. Правила, устанавливающие допустимые элементы и атрибуты для XML-документа, называются определениями документа или схемами документа.

С XML можно использовать различные типы определений документа:

- оригинальное определение типа документа (Document Type Definition, DTD).
- XML Schema – тип определений, основанный на XML-схеме.

DTD (Document Type Definition, определение типа документа) – это язык описания структуры XML-документа, который используется для проверки грамматики XML-документа и его соответствия определенному типу. Цель DTD состоит в том, чтобы определить структуру XML документа. Это делается путем определения списка допустимых элементов. В прологе декларация объявления элементов может быть внутренняя или внешняя (табл. 12.1).

Виды деклараций DTD

Внутренняя декларация DTD	Внешняя декларация DTD
<pre><?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE note [<!ELEMENT note (to, from, Sbj, msg)> <!ELEMENT to (#PCDATA)> <!ELEMENT from (#PCDATA)> <!ELEMENT Sbj (#PCDATA)> <!ELEMENT msg (#PCDATA)>]> <note> <to>Sunny</to> <from>Oliver</from> <Sbj>Hello</Sbj> <msg>This is a good day!</msg> </note></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE note SYSTEM "note.dtd"> <note> <to>Sunny</to> <from>Oliver</from> <Sbj>Hello</Sbj> <msg>This is a good day!</msg> </note></pre> <p>файл note.dtd содержит:</p> <pre><!ELEMENT note (to,from,heading,body)> <!ELEMENT to (#PCDATA)> <!ELEMENT from (#PCDATA)> <!ELEMENT heading (#PCDATA)> <!ELEMENT body (#PCDATA)></pre>

DTD в приведенном примере интерпретируется следующим образом:

- 1) **!DOCTYPE note** – устанавливает, что корневым элементом документа является note;
- 2) **!ELEMENT note** – определяет, что элемент note содержит четыре элемента: to, from, heading, body;
- 3) **!ELEMENT to** – задает, что элемент to должен быть типа "#PCDATA";
- 4) **!ELEMENT from** – определяет, что элемент from должен быть типа "#PCDATA";
- 5) **!ELEMENT heading** – задает, что элемент heading должен быть типа "#PCDATA";
- 6) **!ELEMENT body** – определяет, что элемент body должен быть типа "#PCDATA".

#PCDATA означает разбираемые текстовые данные. Кроме инструкции ELEMENT, существуют следующие:

- ATTLIST для перечисления и объявления атрибутов, которые могут принадлежать элементу.
- ENTITY для определения сущностей в DTD с целью их использования как в связанном с DTD XML-документе, так и собственно в DTD (рис. 12.6).

```

<?xml version="1.0"?>
<!DOCTYPE note [
<!ENTITY name "Hello, world!">
]>
<element>&name;</element>

```

Рис. 12.6. Пример использования сущности

Сущность является заместителем содержания, ее можно однажды объявить и многократно использовать почти в любом месте документа. Встроенные сущности приведены в табл. 12.2

Таблица 12.2

Многократно используемые встроенные сущности

Сущность	Ссылка на сущность	Значение
lt	<	< (меньше чем)
gt	>	> (больше чем)
amp	&	& (амперсанд)
apos	'	' (апостроф или одиночная кавычка)
quot	"	" (двойная кавычка)

К основным недостаткам DTD-схем относят синтаксические отличия от языка XML, а также отсутствие возможностей работы с типами данных. Для решения этих недостатков предлагается использовать XML Schema, пример которой представлен в таблице 12.3.

Таблица 12.3

Пример XML Schema

XML-документ	XML Schema
<pre> <?xml version="1.0" encoding="UTF-8"?> <note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="note.xsd"> <to>Sunny</to> <from>Oliver</from> <Sbj>Hello</Sbj> <msg>This is a good day!</msg> </note> </pre>	<pre> <?xml version="1.0" encoding="UTF-8" ?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> <xs:element name="note"> <xs:complexType> <xs:sequence> <xs:element name="to" type="xs:string"/> <xs:element name="from" type="xs:string"/> <xs:element name="heading" type="xs:string"/> <xs:element name="body" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element> </xs:schema> </pre>

XML Schema представляет из себя язык XML Schema Definition Language и предназначен для создания файлов с расширением .xsd. Приведенная в табл. 12.3 схема интерпретируется следующим образом:

- 1) `<xs:element name="note">` – определяет элемент "note";
- 2) `<xs:complexType>` – у элемента "note" комплексный тип;
- 3) `<xs:sequence>` – последовательность элементов родительского элемента;
- 4) `<xs:element name="to" type="xs:string">` – у элемента "to" строковый тип (текст);
- 5) `<xs:element name="from" type="xs:string">` – у элемента "from" строковый тип;
- 6) `<xs:element name="heading" type="xs:string">` – у элемента "heading" строковый тип;
- 7) `<xs:element name="body" type="xs:string">` – у элемента "body" строковый тип.

Как видно из примера, каждая XML Schema состоит с корневого элемента «schema» и обязательного пространства имен «http://www.w3.org/2001/XMLSchema», а далее идет описание схемы.

Следует отметить, что XML Schema описывают структуру XML-документа. XML-документ, прошедший проверку по XML Schema, является синтаксически верным и валидным, а также имеет ряд преимуществ перед DTD:

- XML Schema пишется на XML;
- XML Schema легко расширяется;
- XML Schema поддерживает типы данных и пространства имен.

ЗАДАНИЯ

Задание 1. Создать корректный XML-документ, который будет содержать информацию минимум о 5 товарах интернет-магазина. При выполнении задания для внешнего оформления использовать CSS.

Задание 2. Преобразовать XML-документ из задания 1 в XML DTD, предварительно сделав его копию. Должно быть внутреннее объявление DTD, в котором должны быть использованы сущности, которые затем применяются в XML-документе.

Задание 3. Преобразовать XML-документ из задания 1 в XML Schema, предварительно сделав его копию.

Задания 4. Провести проверку валидности XML-документов, подтвердив скриншотами результатов.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое XML?
2. Чем является первая строка в коде XML-документа?
3. Назовите правила синтаксиса XML-документа.
4. Какие должны быть имена элементов?
5. Как вы понимаете правило соблюдения корректной вложенности?
6. Какие элементы являются корневыми в XML-документе?
7. Для чего необходима валидация XML-документа?
8. В чем заключается разница между простыми и комплексными типами элементов в XML Schema?
9. Перечислите преимущества XML Schema перед DTD.
10. Какие типы элементов XML Schema вы знаете?
11. Что означает **xs:element**?
12. Поясните, что означает **xs:complexType**.
13. В чем заключается предназначение **xs:sequence**?
14. Для чего используется **xs:attribute**?
15. Каково предназначение ENTITY?
16. Что такое сущности? Каким образом вы использовали сущности?
17. Какие встроенные сущности вы знаете?
18. Для чего используется ATTLIST?
19. Раскройте суть валидного XML-документа. Как типы вы знаете?
20. Что означает следующая запись **<!ENTITY name “Hello, world!”>**?
21. Поясните следующую запись **<!ELEMENT to (#PCDATA)>**.
22. Какие параметры и значения имеет инструкция ATTLIST?
23. Назовите параметры, которые имеет инструкция ELEMENT.
24. Что находится на <http://www.w3.org/2001/XMLSchema>?
25. Каким образом можно объявить DTD? Как расширяется аббревиатура?

Лабораторная работа № 13

РАСШИРЕНИЕ XSLT

Цель работы: изучить расширение XSLT, получить навыки создания таблиц стилей с элементами сортировки и элементом с условиями.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Понятие XSL

XSL (eXtensible Stylesheet Language) – язык таблиц стилей для XML. XSL служит языком трансформирования документов XML и состоит из XML-словаря семантики форматирования. Имея класс произвольно структурированных XML-документов и файлов данных, дизайнеры используют таблицы стилей XSL для указания на то, как это структурированное содержимое должно быть представлено; как содержимое-источник должно быть стилизовано, расположено и разбито на странице в веб-браузере.

Процессор таблицы стилей XSL принимает документ или данные на языке XML и таблицу XSL и производит представление содержимого XML-источника так, как это задумано дизайнером данной таблицы стилей. Существуют два аспекта этого процесса представления: первый – конструирование результирующего дерева из дерева XML-источника, второй – интерпретация результирующего дерева для производства форматированного вывода, пригодного для показа на экране дисплея. Первый аспект называется трансформацией дерева, а второй – форматирование. Процесс форматирования выполняется форматировщиком. Этот форматировщик может быть утилитой вывода браузера.

Форматирование становится доступным при включении семантики форматирования в результирующее дерево. Семантика форматирования выражена в терминах каталога классов объектов форматирования. Узлы результирующего дерева – это объекты форматирования. Классы объектов форматирования обозначают типографические абстракции, например, такие как страница, пара-

граф, таблица. Точный контроль за представлением этих абстракций осуществляется с помощью набора свойств форматирования, XSL-классы объектов форматирования и свойства форматирования предоставляют словарь для выражения целей представления.

Имеются четыре класса свойств XSL, которые можно идентифицировать как:

- свойства, скопированные из CSS (с неизменной семантикой CSS2);
- свойства CSS с расширенными значениями;
- свойства CSS, разбитые на части и/или расширенные;
- «чистые» свойства XSL.

Процессоры XSL обязаны использовать механизмы пространства имен XML Names для распознавания элементов и атрибутов пространства имен <http://www.w3.org/1999/XSL/Format>.

Элементы из пространства имен XSL распознаются только в таблицах стилей, но не в документе-источнике. Разработчики обязаны не расширять пространство имен XSL за счет дополнительных элементов и атрибутов. Вместо этого любое расширение обязано находиться в отдельном пространстве имен.

Спецификация XSL применяет префикс **fo:** для ссылки на элементы пространства имен XSL. В то же время таблицы стилей XSL могут использовать любые префиксы при наличии объявления пространства имен, связывающего префикс с URL пространства имен XSL. Спецификация XSL размещена по адресу <https://www.w3.org/TR/2001/REC-xsl-20011015/>, в ней представлены свойства и примеры префиксов, применяемых в XSL.

Следует отметить, что XSL использует XSLT и XPath для конструирования дерева отбора паттернов, предоставляя таким образом возможности управления тем, как представлены части содержимого-источника и какие свойства ассоциированы с этими частями содержимого, где используются смешанные пространства имен. XSLT характеризует набор примитивов для описания преобразования документа, а XPath определяет синтаксис описания различных мест в документах XML.

Расширение XSLT

XSLT (eXtensible Stylesheet Language Transformations) – это декларативное описание преобразования (трансформации) любого

XML-документа. Спецификация XSLT входит в состав XSL и является рекомендацией W3C.

Существует три основных способа преобразования XML-документов с помощью XSLT в другие форматы, например, в HTML:

1. XML-документ и связанная с ним таблица стилей отправляются клиенту (веб-браузеру), который преобразует документ, как указано в таблице стилей и затем предоставляет результат преобразования пользователю.

2. Сервер применяет таблицу стилей XSLT к XML-документу и преобразует его в другой формат (обычно в HTML). После этого результат отправляется клиенту (веб-браузеру).

3. Какая-то программа преобразует оригинальный XML-документ в другой формат (обычно в HTML), затем результат помещается на сервер. Таким образом, сервер и клиент имеют дело с преобразованным документом.

При помощи XSLT можно добавлять/удалять элементы и атрибуты в конечный файл. Также можно реорганизовывать и сортировать элементы, выполнять тесты, определять, какие элементы скрыть или отобразить, и т. п. Таблица стилей XSL содержит один или больше наборов правил преобразования, которые называются шаблонами преобразования. Шаблон преобразования содержит правила, которые применяются, когда найден узел (элемент, атрибут, текст, комментарий), соответствующий условию поиска. Пример использования XSLT представлен в таблице.

Расширение XSLT

XML-документ	XSLT
<pre><?xml version="1.0" encoding="UTF-8"?> <?xml-stylesheet type="text/xsl" href="catalog.xsl"?> <catalog> <cd> <title>Empire Burlesque</title> <artist>Bob Dylan</artist> <country>USA</country> <company>Columbia</company> <price>10.90</price> <year>1985</year> </cd></pre>	<pre><?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/T ransform"> <xsl:template match="/"> <html> <head><title>My first template rule</title> </head> <body> <h2>My CD Collection</h2> <table border="1"> <tr bgcolor="#9acd32"> <th>Title</th></pre>

XML-документ	XSLT
<pre> <cd> <title>Hide your heart</title> <artist>Bonnie Tyler</artist> <country>UK</country> <company>CBS ords</company> <price>9.90</price> <year>1988</year> </cd></catalog> </pre>	<pre> <th>Artist</th></tr> <xsl:for-each select="catalog/cd"> <xsl:sort select="artist"/> <tr> <td><xsl:value-of select="title"/></td> <td><xsl:value-of select="artist"/></td> </tr> </xsl:for-each> </table></body></html> </xsl:template> </xsl:stylesheet> </pre>

Приведенный пример трактуется следующим образом:

- **<xsl:stylesheet>** – определяет, что данный документ является таблицей стилей XSLT с атрибутами номера версии и пространства имен XSLT;
- **<xsl:output>** – выбирает HTML как выходной формат;
- **<xsl:template>** – указывает, как должны преобразовываться части документа XML. Значение «/» атрибут **match** использует, чтобы определить шаблон для всего XML-документа целиком;
- **<xsl:value-of>** – используется для извлечения значения отобранного XML-элемента и добавления его в выходной поток преобразовываемого документа;
- **<xsl:for-each>** – может использоваться для выбора каждого XML элемента заданного узлового набора;
- **<xsl:sort>** – служит для сортировки выходных данных и располагается внутри элемента **<xsl:for-each>**.

Также могут быть использованы следующие элементы:

- 1) **<xsl:apply-templates>** – применяет некий шаблон к текущему элементу или к дочернему узлу текущего элемента. Если в элемент **<xsl:apply-templates>** добавить атрибут **select**, то он будет относиться только к дочернему элементу, который соответствует значению этого атрибута и может использоваться для определения порядка, в котором будут обрабатываться дочерние узлы;
- 2) **<xsl:choose>** – используется вместе с элементами **<xsl:when>** и **<xsl:otherwise>**, чтобы осуществить проверку на выполнение условия. Пример использования элемента выбора с

условием показан на рисунке. Согласно примеру, если условие выполняется, то ячейка будет выделена цветом *#ff00ff*.

```
<xsl:for-each select="catalog/cd">
  <tr>
    <td><xsl:value-of select="title"/></td>
    <xsl:choose>
      <xsl:when test="price > 10">
        <td bgcolor="#ff00ff">
          <xsl:value-of select="artist"/></td>
        </xsl:when>
        <xsl:otherwise>
          <td><xsl:value-of select="artist"/></td>
        </xsl:otherwise>
      </xsl:choose>
    </tr>
  </xsl:for-each>
```

Пример использования элементов условия

К атрибутам элемента **<xsl:sort>** относятся:

- ***select*** – обязательный атрибут, значением которого является выражение, называемое также ключевым. Это выражение вычисляется для каждого узла обрабатываемого множества, преобразуется в строку и затем используется как значение ключа при сортировке. По умолчанию значением этого атрибута является ".", что означает, что в качестве значения ключа для каждого узла используется его строковое значение;

- ***order*** – необязательный атрибут, который определяет порядок, в котором узлы должны сортироваться по своим ключам. Этот атрибут может принимать только два значения – "ascending", указывающее на восходящий порядок сортировки, и "descending", свидетельствующее о нисходящем порядке. Значением по умолчанию является "ascending", т. е. восходящий порядок;

- ***lang*** – необязательный атрибут, который определяет язык ключей сортировки. В разных языках символы алфавита могут иметь различный порядок, что, соответственно, должно учитываться при сортировке. Атрибут ***lang*** в XSLT может иметь те же самые значения, что и атрибут ***xml:lang*** (например, "en", "en-us", "ru" и т. д.). Если значение этого атрибута не установлено, процессор может либо определять язык исходя из параметров системы,

либо сортировать строки исходя из порядка кодов символов Unicode;

– ***data-type*** – необязательный атрибут, определяющий тип данных, которые несут строковые значения ключей.

Все атрибуты элемента **xsl:sort** должны обладать фиксированными значениями.

ЗАДАНИЯ

Задание 1 Оформить задание 1 из лабораторной работы № 12 через подключение XSLT с сортировкой по возрастанию.

Задание 2 Создать новый XML-документ, в котором должна быть информация об аттестации студентов, при этом преобразовав с помощью XSLT в таблицу с условием, при котором ячейки с оценками ниже 4 должны быть выделены красным фоном, а оценки выше 8 – зеленым фоном.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое XSL? Чем является XSLT?
2. Каково основное назначение технологии XSLT?
3. Для чего предназначен **<xsl:template>**?
4. Что означает значение **match="/"**?
5. Как подключить XSLT к XML?
6. Для чего предназначено **<xsl:stylesheet>**?
7. **Что означает <xsl:apply-templates>**?
8. Для чего предназначено и какие атрибуты имеет **<xsl:sort>**?
9. С помощью какого элемента можно осуществить сортировку с условиями?
10. Для чего используется элемент **<xsl:otherwise>**?
11. В чем заключается предназначение **<xsl:when>**?
12. Что относится к XSL?
13. Как строятся шаблоны преобразований в XSLT?
14. Каков алгоритм преобразования XML-документа с помощью языка XSLT?
15. Для чего предназначено **<xsl:value-of>**?
16. С какой целью используются **<xsl:for-each>**? Какие элементы XSL могут быть внутри него?

Лабораторная работа № 14

МАСШТАБИРУЕМАЯ ВЕКТОРНАЯ ГРАФИКА

Цель работы: изучить способы вставки SVG-изображения на веб-страницу, принципы создания svg-фигур и svg-контуров; познакомиться с правилами применения трансформации, градиентной заливки и анимации к svg-фигурам.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Использование SVG

Масштабируемая векторная графика (Scalable Vector Graphics, SVG) представляет собой вид графики, который создается с помощью математического описания геометрических примитивов (линий, кругов, эллипсов, прямоугольников, кривых), которые образуют изображение. Изображения SVG описываются текстовыми файлами с применением языка XML и предназначены для описания двухмерной векторной или смешанной векторно-растровой графики.

К преимуществам SVG-изображений относятся:

- отсутствие потери качества при масштабировании;
- возможность создания и редактирования в любом текстовом редакторе;
- совместимость со стандартами консорциума W3C: DOM и XSL;
- размеры их файлов небольшие по сравнению с любым другим типом файлов изображений;
- возможность добавлять несколько гиперссылок.
- поддержка скриптов и анимации в SVG, которая позволяет создавать динамичную и интерактивную графику.

Преимущественно SVG используют в дизайне иконок, логотипов и элементов пользовательского интерфейса для веб-сайтов, а также можно создавать графики и диаграммы, простую инфографику, масштабируемые дорожные карты, игры вроде sudoku.

Существуют следующие способы использования svg в веб-браузерах:

1) вставка SVG-файла в HTML-документ с помощью тегов ``, `<embed>`, `<object>` и `<iframe>` (рис. 14.1);

```

<embed src="example.svg" type="image/svg+xml">
<object                                data="example.svg"
type="image/svg+xml"></object>
<iframe src="example.svg" width="200" height="300"
style="border: none"></iframe>
```

Рис. 14.1. Способы вставки файла с расширением svg

2) вставка кода в HTML-документ в элементе `<svg>...</svg>` (рис. 14.2);

```
<svg      xmlns="http://www.w3.org/2000/svg"      ver-
sion="1.1">
  <!-- SVG-код -->
</svg>
```

Рис. 14.2. Вставка векторной графики с помощью тега `<svg>`

3) подключение в PHP-документ с помощью функции `include`: `<? include("example.svg"); ?>`;

4) использование SVG-файла в качестве фонового изображения в CSS: ***background: url(example.svg)***.

Контейнер SVG имеет бесконечные размеры. ***Viewport*** и ***viewBox*** – это две прямоугольные области просмотра, которые ограничены конечными значениями высоты и ширины, указанными в атрибутах ***width*** и ***height***. Область видимости ***viewport*** принимает значения атрибутов высоты и ширины, а ***viewBox*** дает возможность отобразить без искажений или трансформировать конкретный фрагмент SVG. Например, `<svg width="400" height="400" version="1.1" viewBox="0 0 800 800" xmlns="//www.w3.org/2000/svg">` определяет пользовательскую область просмотра ***viewport***, равной 400×400 px.

Первые два значения атрибута ***viewBox*** *min-x* и *min-y* определяют начало системы координат пользовательской области просмотра, последующие два значения – ее ширину и высоту и одновременно масштабирование изображения. В примере пользовательская область занимает прямоугольный фрагмент размером

800×800 px, то есть область видимости *viewport* и дополнительно по 400 px справа и снизу. Таким образом, масштаб видимости SVG-изображения уменьшается.

Так как *viewport* является предком для *viewBox*, то начало системы координат *viewBox*, по умолчанию также, как и системы координат *viewport*, находится в левом верхнем углу (0, 0) и положительное направление оси *X* – будет справа, а оси *Y* – внизу.

Основные элементы SVG

К основным элементам, которые могут быть созданы, относятся прямая линия, ломаная линия, многоугольник, прямоугольник, круг, эллипс, сложная траектория. Соответствующие им теги представлены в табл. 14.1.

Таблица 14.1

Основные геометрические элементы SVG

Элементы SVG	Атрибуты
<line> (прямая линия)	x1 – координата начальной точки линии по оси <i>X</i> ; y1 – координата начальной точки линии по оси <i>Y</i> ; x2 – координата конечной точки линии по оси <i>X</i> ; y2 – координата конечной точки линии по оси <i>Y</i>
<polyline> (ломаная линия)	points – координаты ломанной линии парами <i>x, y</i> через пробел
<polygon> (многоугольник)	
<rect> (прямоугольник)	x – координата левой верхней точки прямоугольника по оси <i>X</i> ; y – координата левой верхней точки прямоугольника по оси <i>Y</i> ; width – ширина прямоугольника; height – высота прямоугольника; rx – радиус закругления углов прямоугольника по оси <i>X</i> ; ry – радиус закругления углов прямоугольника по оси <i>Y</i> ;
<circle> (круг)	cx – координата центра круга по оси <i>X</i> ; cy – координата центра круга по оси <i>Y</i> ; r – радиус круга;
<ellipse> (эллипс)	cx – координата центра эллипса по оси <i>X</i> ; cy – координата центра эллипса по оси <i>Y</i> ; rx – радиус эллипса по оси <i>X</i> ; ry – радиус эллипса по оси <i>Y</i>

Создание сложной траектории осуществляется тегом **<path>**, который позволяет создавать произвольные фигуры. Форма фигуры задается атрибутом **d**, значение которого – это набор специальных команд. Эти команды могут быть и в верхнем, и в нижнем регистре. Верхний регистр указывает на то, что применяется абсолютное позиционирование, а нижний – относительное. Список команд и их значений представлены в табл. 14.2, а пример приведен на рис. 14.3.

Таблица 14.2

Значения элемента <path>

Значения атрибута тега <path>	Варианты значений атрибута
M, m – начальная точка	mx, my – координаты точки
L, l – отрезок прямой	lx, ly – координаты от текущей точки линии к указанной
H, h – горизонтальная линия	hx – координата, до которой создается линия по оси X
V, v – вертикальная линия	vy – координата, до которой создается линия по оси Y
A, a – дуга эллипса	rx,ry – радиусы дуги эллипса; x-axis-rotation – угол поворота дуги относительно оси X; large-arc-flag – если (=1), то строится большая часть дуги, если (=0) – меньшая; sweep-flag – если (=1), то дуга строится по часовой стрелке, если (=0) – против часовой стрелки; x,y – координаты конечной точки дуги
C, c – кубическая кривая Безье	x1, y1 – координаты первой контрольной точки; x2, y2 – координаты второй контрольной точки; x, y – координаты конечной точки кривой
S, s – гладкая кубическая кривая Безье	x2, y2 – координаты второй контрольной точки; x, y – координаты конечной точки кривой. Первая контрольная точка является зеркальным отражением второй контрольной точки
Q, q – квадратичная кривая Безье	x1, y1 – координаты контрольной точки; x, y – координаты конечной точки кривой.
T, t – гладкая квадратичная кривая Безье	x, y – координаты конечной точки кривой. Контрольная точка этой команды является зеркальным отражением контрольной точки предыдущей команды.
Z, z – замыкание траектории	Не имеет значений

Сложные SVG фигуры можно нарисовать в векторных редакторах Adobe Illustrator, CorelDRAW, Inkscape (рекомендуемый свободный редактор SVG-графики) и сохранить в формате *svg*. Далее полученный документ открывается в Блокноте, FrontPage или любом другом редакторе, в окне которого будет представлен автоматически корректно созданный код. Данный код можно скопировать и вставить в HTML-документ.

```
<svg      xmlns="http://www.w3.org/2000/svg"      ver-
sion="1.1" width="600" height="100">
  <path d="M10,15 h50 v60 L110,55 A25,35 -30 0,1
150,30 M160,50 C160,110 260,110 260,50 S360,-10 360,50
M370,50 Q420,100 470,50 T570,50 z" stroke="#b4241b"
stroke-width="3" fill="none" />
</svg>
```

Рис. 14.3. Пример создания сложной траектории

К общим атрибутам, которые могут быть во всех элементах, относятся:

- ***stroke*** – цвет линии;
- ***stroke-width*** – толщина линии;
- ***stroke-linecap*** – стиль концов линии. Возможные значения атрибута: *round* – по форме круга; *square* – по форме квадрата;
- ***stroke-dasharray*** – чередование штрихов и пробелов в пунктирной линии;
- ***fill*** – цвет заливки (*none* – без заливки);
- ***fill-opacity*** – прозрачность заливки (от 0 до 1);
- ***fill-rule*** – правило заливки. Возможные значения атрибута: *nonzero* – сплошная заливка; *evenodd* – внутренняя часть фигуры не заливается;
- ***style*** – стиль элемента;
- ***class*** – класс элемента.

Преобразования задаются в атрибуте ***transform***. Можно указать несколько преобразований через пробел. Существуют следующие виды трансформации:

- 1) *rotate*(*rotate-angle* [*cx cy*]) – поворот;
- 2) *scale*(*sx* [*sy*]) – масштабирование;
- 3) *translate*(*tx* [*ty*]) – перенос;
- 4) *skewX*(*skew-angle*) – наклон по оси *X*;
- 5) *skewY*(*skew-angle*) – наклон по оси *Y*.

Для хранения повторно используемого содержимого применяется тег **<defs>**. Содержимое в этом теге является скрытым и будет использовано только при обращении к нему по `id`. В теге можно хранить, например, градиентную заливку (**<linearGradient>**, **<radialGradient>**) и применить ее к отдельным фигурам. Также можно хранить любые элементы SVG (рис. 14.4).

```
<defs>
  <linearGradient id = "MyGradient">
    <stop offset = "30%" stop-color = "red"/>
    <stop offset = "70%" stop-color = "yellow"/>
  </linearGradient>
</defs>
<rect x = "0" y = "0" width = "150" height="150"
fill = "url(#MyGradient)"/>
```

Рис. 14.4. Использование тега **<defs>**

С целью объединения нескольких фигур в группу для последующих действий над ней как над одним целым используется парный тег **<g>**. Группе так же может быть присвоен уникальный `id` для повторного применения. Несколько групп могут быть объединены в одну.

```
<g id = "gr1-gr2" transform="translate(50,70)">
  <g id = "gr1">
    <rect x="30" y="50" width="120" height="50"
style="fill-opacity: 0.7; fill: red;" />
    <rect x="30" y="140" width="120" height="50"
style="fill:yellow; stroke-width:3; stroke:
blue;"/></g>
  <g id="gr2">
    <rect x="30" y="230" width="120" height="50"
style="fill:none; stroke-width:3; stroke: blue;"/>
    <rect x="30" y="320" width="120" height="50"
style="fill:none; stroke-width:8; stroke: red; stroke-
opacity:0.4;"/>
  </g>
</g>
```

Рис. 14.5. Пример использования тега **<g>**

Для создания копий SVG-фигур и их размещения на странице, а также добавления различных преобразований используется тег

<use>, указывается id контура и прописываются его координаты, например **<use xlink:href="#myCircle" x="10" fill="blue"/>**.

Анимация SVG

SMIL (Synchronized Multimedia Integration Language) – язык разметки на основе XML, с помощью которого осуществляется анимация. Каждой отдельной геометрической SVG-фигуре можно присвоить анимации SMIL. Для этого используется непарный тег **<animate>**, который анимирует отдельные свойства. Свойства прописываются с указанием анимированного свойства в атрибуте **attributeName**. В примере на рис. 14.6 анимируется свойство **cx**, расположение по оси *X* изменяется от 100 до 300 px за 5 секунд.

```
<circle cy="70" r="50" fill="red">
  <animate attributeName="cx" from="100"
to="300" dur="5s"/>
</circle>
```

Рис. 14.6. Пример использования анимации SVG-фигуры

Можно задавать сразу несколько анимаций, и они будут выполняться одновременно. В теге **<animate>** можно сослаться на анимируемый объект через его id с помощью атрибута **xlink:href**.

Для создания анимации трансформаций предназначен тег **<animateTransform>**, вид трансформации указывается в атрибуте **type**: **<animateTransform xlink:href="#mygroup" attributeName="transform" attributeType="XML" type="rotate" from="0,60 50" to="45,60,50" dur="5s" additive="sum" fill="freeze"/>**.

Для обработки событий запуска анимации можно воспользоваться атрибутами **begin** и **end**, например **begin="mouseover"** для начала анимации при наведении на элемент, **end="mouseout"** для завершения анимации при отводе курсора мыши.

Работа с текстом

Текст в элементе SVG определяется с помощью тега **<text>**. К специфическим атрибутам, используемыми для работы с текстом, относятся:

- **x** и **y** – координаты расположения текста на экране: **<text x="0" y="20">Text</text>**;

- *dx* и *dy* – размещение текстовых областей относительно текущей позиции;
- *text-anchor* – выравнивание текстовой строки относительно точки (x, y). Может принимать значения *start*, *middle*, *end*;
- *rotate* – поворот текста на заданный угол;
- *textLength* – устанавливает ширину строки;
- *lengthAdjust* – сжатие и растягивание текста, используется вместе с атрибутом *textLength*. Может принимать значения *spacing* и *spacingAndGlyphs*.

Тег **<tspan>** в SVG аналогичен тегу ****. Используется при необходимости применить стиль к определенной содержимой. Для ссылки на существующий текст можно воспользоваться тегом **<tref>**.

С помощью тега **<textPath>** осуществляется отображение текста вдоль направляющей линии. Пример использования **<textPath>** представлен на рис. 14.7.

```
<defs>
<path id="idname" fill="..." stroke="..." d="..." />
</defs>
<use xlink:href="#idname" />
<text x="..." y="..." font-family="Arial">
<textPath xlink:href="#idname">
Write your text here.
</textPath>
</text>
```

Рис. 14.7. Пример использования тега **<textPath>**

Текст в SVG может быть стилизован с помощью свойств CSS, которые могут быть установлены как атрибуты.

ЗАДАНИЯ

Задание 1. Создать новый документ `svgrafik.html`, в котором разместить текст вдоль произвольной кривой. Данный текст расположить по центру, выделить произвольным цветом. Размер шрифта должен составлять 36 px.

Задание 2. Создать в документе задания 1 элементы, показанные на рис. 14.8.

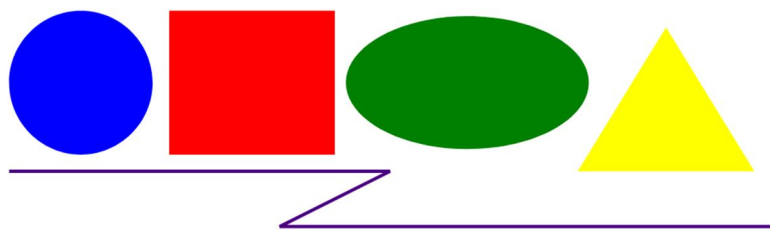


Рис. 14.8. Элементы для задания 2

Задание 3. Сделать в этом же документе элемент, представленный на рис. 14.9, используя графический редактор для работы с векторной графикой. С помощью только тега `<path>` создать несколько дополнительных фигур в виде украшений. К дополнительным фигурам можно применить различные анимации.

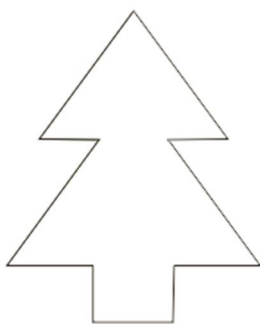


Рис. 14.9. Пример фигуры для задания 3

Задание 4. Открыть `svgicon.html` файл из папки *labs*. Использовать любой `svg`-код иконки из файла и поместить в `<defs>`. Создать пять копий иконок и разместить их в новом ранее созданном HTML-документе. Применить к элементам различные трансформации.

Задание 5. Сделав предварительно копию документа с элементами из задания 2, анимировать для них следующие свойства:

1. Изменить для треугольника желтую заливку на линейную градиентную заливку от зеленого к оранжевому.
2. Для эллипса сделать изменение заливки цветом при щелчке мыши на нем.
3. Для квадрата сделать анимацию появления контура по траектории границы вокруг него.
4. Для круга при наведении мыши сделать изменение цвета контура.

Задание 6. В копии HTML-документа задания 1 лабораторной работы № 2 внизу страницы создать четыре SVG-фигуры в виде кругов радиусом 45 px. Каждый из них должен быть гиперссылкой на задания из лабораторной работы № 14. Для копии документа изменить ранее созданные CSS-стили и устаревшие атрибуты на SCSS.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение понятия SVG. Как расшифровывается аббревиатура?
2. Какие преимущества SVG перед остальными форматами?
3. Как использовать SVG в HTML?
4. Каким образом создать прямую линию и ломанную линию?
5. Расскажите алгоритм создания прямоугольника и многоугольника.
6. Каким образом создать круг и эллипс?
7. Для чего предназначен тег **<path>**? Что означают значения в теге **<path>**?
8. Какие атрибуты относятся к общим?
9. Как создать заливку SVG-фигуры?
10. Поясните, как изменить цвет и размер ширины контура SVG-фигуры.
11. Каким образом трансформировать SVG-фигуру?
12. Для чего используется тег **<use>**?
13. Каким образом использовать графические редакторы для создания SVG?
14. Как создать текст в SVG?
15. Для чего используется тег **<defs>**?
16. Каким образом создать градиентную заливку?
17. Приведите алгоритм создания анимации.
18. Какие атрибуты могут быть использованы при создании анимации?
19. Для чего предназначен атрибут **viewBox**?
20. С какой целью используется тег **<g>**?
21. Создайте логотип компании Apple и браузера Google Chrome, используя только тег **<path>**.

Лабораторная работа № 15

ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА (DOM)

Цель работы: изучить понятие DOM, научиться использовать JavaScript для управления объектной моделью документа.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Понятие DOM

Объектная модель документа (Document Object Model) – это прикладной программный интерфейс для HTML- и XML-документов, который представляет собой иерархическое дерево узлов, позволяющее добавлять, удалять и изменять отдельные части страницы. Доступ к элементам для дальнейшего их изменения осуществляется с помощью языка программирования JavaScript, который является объектно-ориентированным и используется для создания сценариев динамического взаимодействия с веб-страницами.

Код JavaScript в HTML-документ добавляется с помощью тега **<script>**: **<script src="scriptFile.js">**. Браузер читает HTML-документ сверху вниз и, когда он встречается тег **<script>**, рассматривает текст программы как сценарий и выполняет его. Закончив его выполнение, возвращается обратно в HTML-документ и отображает оставшуюся часть документа. Если на веб-странице используется много сценариев, то может возникнуть задержка загрузки веб-страницы. Поэтому считается лучше все ссылки на JavaScript-сценарии указывать после контента страницы перед закрывающим тегом **<body>**.

Вывод результата выполнения сценария или блока кода в консоль веб-страницы можно осуществлять с помощью метода **console.log()**. Можно набрать короткую последовательность команд на консоли, которая является частью комплекта средства разработки, входящего в состав браузера.

Значения в JavaScript могут иметь следующие типы: число, булево значение (*true* или *false*), специальные значения *null* и *undefined*, строка, символ, функция, объект. Значение, отличающееся-

ся от строки, числа, значения *symbol*, *true*, *false*, *null* или *undefined*, является объектом. У объектов есть свойства и методы. Свойства имеют имя и значения, а методы также могут иметь параметры, которые могут вернуть значение преобразования или конкретное значение объекта. Задать объекты можно в переменных, а затем применить к ним свойства или элементы. Объявление переменных осуществляется с помощью ключевых слов *let* или *const*. Устаревшими объявлениями переменных являются ключевое слово *var* и отсутствие ключевого слова.

Доступ к элементам DOM

Все элементы HTML-разметки соответствуют узлам в дереве. Всего существует 12 типов узлов, которые наследуются от одного базового типа. Элемент **<html>** является элементом документа (*document element*). Элемент документа – это корневой элемент в документе, содержащий все остальные элементы. В HTML-документах элементом документа всегда является **<html>**, а в XML-разметке им может быть заданный пользователем элемент.

Узлы в документе связаны с другими узлами в терминах традиционных семейных отношений. Элементы **<head>** и **<body>** являются одноуровневыми, имеющими общий родительский элемент **<html>**. Таким образом, у каждого узла есть свойство *childNodes*, которое содержит объект *NodeList*, используемый для хранения упорядоченного списка узлов, доступных по позиции. Объект *NodeList* является динамически обновляемым объектом, который отражает изменения DOM-структуры. Узлы в списке *childNodes* являются одноуровневыми и переходить от одного узла к другому можно с помощью *previousSibling* и *nextSibling*. Если дочерний узел единственный, то оба эти свойства равны *null*.

Свойства *firstChild* и *lastChild* родительского узла указывают на первый и последний узлы. Значение *someNode.firstChild* равно *someNode.childNodes[0]*, а значение *someNode.lastChild* равно *someNode.childNodes[someNode.childNodes.length-1]*.

Узел документа в JavaScript представлен с помощью типа *Document*, а в коде – как объект *document* типа *HTMLDocument*. К свойствам *document* относятся *documentElement*, *firstChild*, *childNodes[0]*, которые указывают на элемент **<html>**. Также *document*

имеет свойства *head*, *body* и *doctype*, указывающие на соответствующие элементы **<head>**, **<body>** и тег **<!DOCTYPE>**.

В качестве экземпляра типа *HTMLDocument* объект *document* имеет свойства, которые предоставляют информацию о загруженной веб-странице. Первое из них свойство *title*, которое содержит текст тега **<title>**. Следующие три свойства связаны с запросами веб-страниц. Свойство *URL* содержит полный URL-адрес страницы, свойство *domain* – доменное имя страницы, а *referrer* – URL-адрес веб-страницы, с которой выполнен переход на текущую веб-страницу.

Создание нового элемента осуществляется с помощью метода ***createElement()***, который принимает имя тега создаваемого элемента. Например, создать элемент **<div>** можно следующим образом: `let newDiv = document.createElement("div")`. Однако чтобы его окончательно добавить в дерево элементов используются метод ***appendChild()***: `document.body.appendChild(newDiv)`.

Для внесения изменений в текст или структуру внутри элемента применяется свойство *innerHTML*, которое в режиме чтения возвращает HTML-код, представляющий все дочерние узлы элемента, в том числе комментарии и текстовые узлы. В режиме записи свойство *innerHTML* составляет из назначенной ему строки DOM-поддерево и заменяет им все дочерние узлы элемента. Все теги в нем преобразуются в элементы HTML-документа. Если строка не содержит таких элементов, в свойстве сохраняется обычный текст. Для работы с текстовым контентом элемента независимо используется свойство *innerText*.

Для доступа к конкретному элементу или множеству элементов, для выполнения каких-либо действий с ними применяются методы ***getElementById()*** и ***getElementsByName()***. Метод ***getElementById()*** принимает идентификатор элемента, который нужно получить, и возвращает этот элемент или *null*, если его не существует. Если страница содержит несколько элементов с одинаковым идентификатором, то возвращается первый из них. Метод ***getElementsByName()*** принимает значение атрибута *name* элемента и возвращает объект *NodeList*, содержащий эти имена.

В HTML5 был добавлен метод ***getElementsByClass()***, который принимает строку с одним или несколькими именами классов и возвращает объект *NodeList* с элементами, к которым применены

эти классы. Свойство *className* используется для добавления, удаления и замены имен классов.

Для получения с помощью селекторов CSS доступа к элементам консорциумом W3C была разработана спецификация Selectors, в которой представлены методы *querySelector()* и *querySelectorAll()*. Метод *querySelector()* принимает CSS-запрос по указанному селектору и возвращает первый соответствующий ему элемент или значение *null*, а метод *querySelectorAll()* возвращает соответствующие узлы в статическом экземпляре *NodeList*. Пример использования методов доступа к элементам показан на рис. 15.1.



Рис. 15.1. Использование методов доступа к элементам:
а – код HTML-документа; б – результат в консоли браузера

События в DOM

Взаимодействие JavaScript с HTML осуществляется с помощью событий (events), которые сигнализируют, что в документе или окне браузера что-то произошло. Например, когда пользователь нажимает клавишу на клавиатуре, перемещает указатель мыши, щелкает кнопкой мыши или касается сенсорного экрана, веб-браузер генерирует событие.

События соответствуют определенным действиям, которые выполняет пользователь. Функция, выполняемая в ответ на событие, называется обработчиком события или слушателем события. Эти функции регистрируются с помощью метода *addEventListener()*, который принимает имя обрабатываемого события, функцию-обработчик и логическое значение, указывающее,

нужно ли вызывать событие при перехвате или всплытии. Основные события представлены в таблице 15.1.

Основные события

Имя события	Описание события
blur	Элемент теряет фокус
change	Содержимое поля формы изменяется
click	Щелчок мышью на объекте
error	Произошла ошибка при загрузке документа или изображения
focus	Элемент получает фокус
keydown	Клавиша на клавиатуре нажата
keypress	Клавиша на клавиатуре нажата и удерживается
keyup	Клавиша на клавиатуре отпускается
load	Страница или изображение загружены
mousedown	Кнопка мыши нажата
mousemove	Мышь перемещена
mouseout	Мышь отодвигается от элемента
mouseover	Мышь перемещается над элементом
mouseup	Кнопка мыши отпущена
submit	В форме нажата кнопка отправки

Кроме метода *addEventListener()*, добавив к имени события приставку «on» и получив тем самым обработчик события, который можно указать для конкретного элемента в качестве атрибута (*<input onclick="myFunction();">*) или в виде метода объекта (*document.body.onclick="MyFunction()"*). Недостатком этих способов является вызов за один раз только одного события.

С помощью событий и использования свойства можно изменять стили элемента, например, *document.body.style.border*.

Работа с формами

Веб-форма представляется в JavaScript типом *HTMLFormElement*. Все формы на странице содержатся в коллекции *forms*. Каждая форма в ней доступна по числовому индексу (*document.forms[0]*) или по имени (*document.forms["form2"]*).

Элементы формы содержатся в коллекции *elements*. Элементы хранятся в коллекции в том порядке, в котором они расположены в разметке, и индексируются по позиции и имени. Для элементов типа *radio* и *checkbox* используются операторы цикла и

условия для определения выбора значения. Пример представлен на рис. 15.2.

```
<p>
<label><input type="checkbox" class="checkbox" value="checkbox 1">checkbox 1</label>
<label><input type="checkbox" class="checkbox" value="checkbox 2">checkbox 2</label>
<label><input type="checkbox" class="checkbox" value="checkbox 3">checkbox 3</label>
</p>
```

а

```

1  document.querySelector('button').addEventListener('click', function(){
2      let check = document.querySelectorAll('.checkbox');
3      for (let i = 0; i < check.length; i++) {
4          if (check[i].checked) {
5              let list = document.createElement("li")
6              list.innerHTML=check[i].value
7              document.getElementById('footer').append(list);
8          }
9      }
10 }
```

б

Рис. 15.2. Пример вывода значений checkbox:

а – HTML-разметка; *б* – JavaScript-код

Согласно рис. 15.2б, в строках 5–7 создается новый элемент, который будет добавлен в элемент `footer` в переменной `list`. В строке 6 используя свойство `innerHTML`, добавляется значение выбранного `checkbox`.

Чтобы получить текст списка формы, можно использовать свойства `text`. Для доступа к элементам списка предназначена коллекция `options`. Пример показан на рис. 15.3.

```

10  <form>
11      <select>
12          <option value="option 1">option 5</option>
13          <option value="option 2">option 2</option>
14          <option value="option 4">option 3</option>
15      </select>
16      <button>From Select</button>
17  </form>
18  <script>
19      document.querySelector('button').addEventListener('click', function(){
20          let selectbox = document.forms[0].elements[0]
21          let text = selectbox.options[0].text;
22          let value = selectbox.options[2].value;
23          console.log(text)
24          console.log(value)
25      })
26  </script>
```

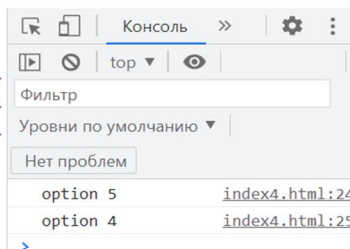


Рис. 15.3. Пример получения значения списка формы

С помощью свойства *selected* можно узнать, какие элементы списка выбраны. Чтобы получить все выбранные элементы, можно перебрать набор элементов в цикле, проверяя их свойство *selected*.

Методом *checkValidity()* можно проверить допустимо ли значение конкретного поля формы. Он доступен для всех элементов и возвращает *true* или *false*.

ЗАДАНИЯ

Задание 1. Создать фотогалерею из трех фотографий. При наведении на первой фотографии необходимо, чтобы появлялось вместо нее описание, состоящее из нескольких строк, при щелчке мыши на второй фотографии должна появляться граница толщиной 10 px сплошного красного фото. При наведении на последнем фото оно должно быть заменено на другое, при отводе курсора возвращаться к исходному.

Задание 2. Создать копию формы задания 4 из лабораторной работы № 1, убрав теги таблицы и добавив к ней раскрывающиеся списки для выбора факультета, группы и курса. Значения заполненной формы должны выводиться в **<footer>** по нажатию кнопки.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение понятия DOM.
2. Перечислите все способы доступа к элементам HTML-документа.
3. Для чего используется свойство *childNodes*?
4. Что такое событие?
5. Для чего предназначен метод *addEventListener()*?
6. Чем отличается использование метода *addEventListener()* от атрибута обработчика события?
7. Что такое *forms*?
8. Раскройте суть *elements*.
9. Что такое свойство *innerHTML* и для чего оно необходимо?
10. Как получить доступ к элементам **<select>**?
11. Для чего используются циклы?
12. Что такое метод *checkValidity()*?
13. Какие события вы знаете?

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Фрейн, Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств / Б. Фрейн; [пер. с англ. Н. Вильчинского]. – 2-е изд. – СПб.: Питер, 2017. – 272 с.
2. Дакетт, Дж. HTML и CSS. Разработка и создание веб-сайтов / Дж. Дакетт; [пер. с англ. М. А. Райтмана]. – М.: Эксмо, 2020. – 474 с.
3. Мангано, С. XSLT. Сборник рецептов / С. Мангано; [пер. с англ. А. А. Слинкина]. – СПб.: ДМК Пресс, 2008. – 861 с.
4. Макфарланд, Д.-С. Большая книга CSS / Д.-С. Макфарланд; [пер. с англ. : И. Дубенка, П. Радченко, В. Радькова]. – СПб. : Питер, 2010. – 512 с.
5. Прохоренок, Н. А. Bootstrap и CSS-препроцессор Sass. Самое необходимое / Н. А. Прохоренок. – СПб.: БХВ-Петербург, 2021. – 496 с.
6. Рэй, Э. Изучаем XML / Э. Рэй; пер. с англ. – СПб.: Символ-Плюс, 2001. – 408 с.
7. Одиночкина, С. В. Основы технологий XML / С. В. Одиночкина. – СПб.: НИУ ИТМО, 2013. – 56 с.
8. Тидуэлл, Д. XSLT, 2-е издание / Д. Тидуэлл; пер. с англ. – СПб.: Символ-Плюс, 2010. – 960 с.
9. Кобайло, А. С. Введение в XML: учеб.-метод. пособие / А. С. Кобайло, Н. А. Жияк. – Минск: БГТУ, 2011. – 321 с.
10. Сидельников, Г. Наглядный CSS / Г. Сидельников. – СПб.: Питер, 2021. – 224 с.
11. Грант, К. CSS для профи / К. Грант. – СПб.: Питер, 2019. – 496 с.
12. Фрисби, М. JavaScript для профессиональных веб-разработчиков / М. Фрисби. – 4-е изд. – СПб.: Питер, 2022. – 1168 с.
13. Хорстман, К. С. Современный JavaScript для нетерпеливых / К. С. Хорстман; [пер. с англ. А. А. Слинкина]. – М.: ДМК Пресс, 2021. – 288 с.
14. Вейл, Э. HTML5. Разработка приложений для мобильных устройств. – СПб.: Питер, 2015. – 480 с.

ОГЛАВЛЕНИЕ

Предисловие	3
Лабораторная работа № 1. Основные теги HTML5.....	4
Лабораторная работа № 2. Основы CSS.....	16
Лабораторная работа № 3. Оформление текста и списков перечисления на CSS3	22
Лабораторная работа № 4. Блочная модель CSS	27
Лабораторная работа № 5. Форматирование таблиц и веб- форм на CSS3.....	34
Лабораторная работа № 6. Графика на веб-странице	40
Лабораторная работа № 7. Преобразования. Переходы. Анимация	47
Лабораторная работа № 8. Панель навигации	55
Лабораторная работа № 9. Система модульной верстки.....	61
Лабораторная работа № 10. Адаптивный веб-дизайн. Fle- box-верстка.....	68
Лабораторная работа № 11. Препроцессор SCSS	73
Лабораторная работа № 12. Создание и валидация XML- документов	81
Лабораторная работа № 13. Расширение XSLT	89
Лабораторная работа № 14. Масштабируемая векторная графика.....	95
Лабораторная работа № 15. Объектная модель документа (DOM).....	105
Список рекомендуемой литературы	112

Учебное издание

Жиляк Надежда Александровна
Барковский Евгений Валерьевич

**КОМПЬЮТЕРНЫЕ ЯЗЫКИ РАЗМЕТКИ
ЛАБОРАТОРНЫЙ ПРАКТИКУМ**

Учебно-методическое пособие

Редактор *Е. С. Ватеичкина*
Компьютерная верстка *Д. С. Жих*
Дизайн обложки *П. П. Падалец*
Корректор *Е.С. Ватеичкина*

Подписано в печать _____. Формат 60×84¹/₁₆.
Бумага офсетная. Гарнитура Таймс. Печать ризографическая.
Усл. печ. л. 7.6. Уч.-изд. л. 9,0.
Тираж 250 экз. Заказ

Издатель и полиграфическое исполнение:
УО «Белорусский государственный технологический университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/227 от 20.03.2014.
Ул. Свердлова, 13а, 220006, г. Минск.