Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

М. И. ОЗЕРОВА

ОСНОВЫ ИНФОРМАЦИОННОГО ДИЗАЙНА

Практикум

Владимир 2020

УДК 004.92
ББК 32.972
О-46

Рецензенты:
Доктор технических наук, профессор
зав. кафедрой вычислительной техники
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
В. Н. Ланцов

Кандидат технических наук
генеральный директор ООО «Системный подход»
А. В. Шориков

Издаётся по решению редакционно-издательского совета ВлГУ

Озерова, М. И.

О-46 Основы информационного дизайна : практикум / М. И. Озерова ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2020. – 140 с. – ISBN 978-5-9984-1167-0.

Включает ознакомительный цикл лабораторных работ по web-программированию, в котором описаны основные термины и понятия, характеризующие современный web-дизайн, а также технологии, применяемые для web-разработки, такие как HTML, CSS, JavaScript. Позволяет самостоятельно закрепить и проверить полученные теоретические знания и приобрести практические навыки в разработке современных web-сайтов.

Предназначен для студентов 2-го курса высших и средних учебных заведений очной формы обучения по направлениям подготовки 09.04.02 – Информационные системы и технологии, 09.04.04 – Программная инженерия, 09.02.03 – Программирование в компьютерных системах, 09.02.07 – Информационные системы и программирование. Может быть полезен широкому кругу читателей, самостоятельно осваивающих новые технологии создания сайтов.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 61. Табл. 17. Библиогр.: 8 назв.

УДК 004.92
ББК 32.972

ISBN 978-5-9984-1167-0

ВВЕДЕНИЕ

Прогресс ведёт к тому, что процесс цифровизации общества всё больше распространяется на все сферы жизни человека. Значительная часть информации представлена в Интернете такими структурами, как сайты, а веб-страницы – это единицы представления информации, которые объединяются в сайты.

Специальный структурированный вид информации задаёт язык гипертекстовой разметки HTML, который имеет средства для создания ссылок, связывающих веб-страницы между собой, и возможности оформления. HTML достаточно, чтобы создать нормально функционирующий сайт.

Однако современное общество, кроме простого представления информации, требует её яркого оформления. Сайт с грамотным дизайном притягивает внимание и выигрывает борьбу за пользователей у сайтов с аналогичной информацией. Для создания дизайна используются каскадные таблицы стилей CSS, которые применяют теги HTML, чтобы задать каждому элементу стили, в совокупности создающие дизайн.

Кроме того, современный сайт должен обрабатывать информацию пользователя и мгновенно выдавать ему решение простых задач. Логика сайта, его умение решать задачи без участия человека прямо в браузере, создаются посредством языка JavaScript. Он встраивается непосредственно в веб-страницу, может получать, дополнять и изменять её содержимое, а также теги и стили.

Это не полный перечень потребностей современного сайта. Но с умением применять HTML, CSS и JavaScript уже можно достичь больших успехов в создании интернет-ресурсов, для того чтобы представить и донести необходимую информацию до пользователя через Интернет. Этому и посвящен практикум.

Лабораторная работа № 1

ОСНОВЫ ЯЗЫКА ГИПЕРТЕКСТОВОЙ РАЗМЕТКИ HTML

Цель работы

1. Изучить основы языка разметки HTML.
2. Научиться создавать статические веб-страницы.
3. Научиться переходить с одной веб-страницы на другую.

Ход работы

Язык HTML (от англ. Hypertext Markup Language – «язык разметки гипертекста») используется при разработке сайтов и представляет из себя набор кодовых символов – *тегов*.

Теги – специальные слова, заключенные в угловые скобки (<>), которые создают элементы веб-страницы или их свойства. Элементы веб-страницы бывают *блочными* и *строчными.*

Парные теги имеют открывающий (<…>) и закрывающий (</ …>) теги. *Одинарные* элементы создаются одним тегом.

Внутри угловых скобок пишется название тега и его **атрибуты,** которые определяют свойства элемента. Атрибут – это специальное слово, после которого пишется знак равенства (=) и значение свойства. Атрибуты тегов расширяют возможности тега и позволяют гибко управлять его настройками. Согласно основной спецификации HTML все атрибуты тегов следует указывать в двойных («значение») или одинарных кавычках (‘значение’).

HTML-документ для дальнейшего использования в браузере должен быть сохранен со следующим наименованием: `file_name.html`

Структура документов. Каждый HTML-документ начинается со строки <!DOCTYPE…>, в которой указывается, какой спецификации соответствует данный документ (указывается тип документа). Это необходимо для правильной интерпретации веб-страницы браузером.

Далее после строки <!DOCTYPE…> идет основной тег, объявляющий начало разметки <html>, а также «головы» <head> и «тела» <body>. Все эти теги парные и требуют закрывающих тегов, соответственно </html>, </head>, </body> (рис. 1.1).

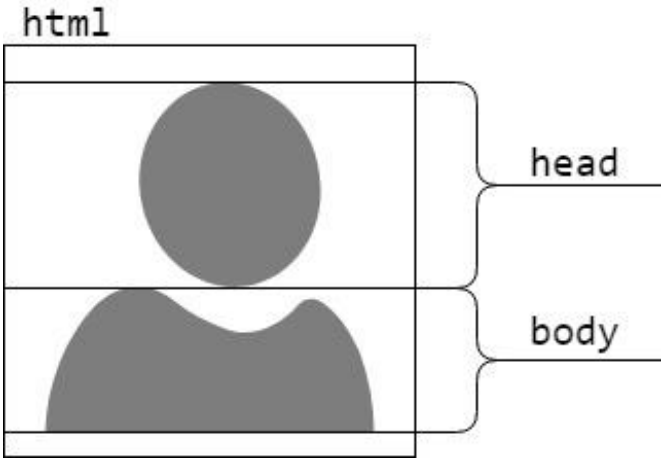


Рис. 1.1. Иллюстрация структуры HTML-документа

Таким образом, общая структура HTML-документа такова:

```
<!doctype html>
<html>
<head></head>
<body></body>
</html>
```

Содержимое страницы. Все HTML-элементы делятся на две группы:

- блочные (block);
- строчные (inline).

Блочные элементы – большие строительные блоки веб-страницы. Блочный элемент занимает всю доступную ширину (отображается в виде прямоугольника), а высота блочного элемента вычисляется браузером автоматически исходя из объема его содержимого. Обычно блочные элементы используются, чтобы разделить содержимое веб-страницы на логические блоки.

Строчные элементы – настройка и разметка содержимого страницы. Ширина строчного элемента равна объему содержимого. Несколько строчных элементов могут идти подряд друг за другом на одной строке.

Внутрь блочных элементов можно вставлять как другие блочные, так и строчные элементы. Внутрь строчных элементов допустимо помещать другие строчные элементы. Вставлять блочные элементы внутрь строчных запрещено.

В теге <head>…</head> прописывается информация, необходимая для отображения страницы в браузере. Это может быть указание кодировки и название всей веб-страницы.

Кодировка указывается в строчном теге <meta> через атрибут charset. Название пишется внутри тега <title>…</title>.

В теге <body>…</body> прописывается наполнение страницы с помощью специальных тегов (табл. 1.1).

Таблица 1.1. *Теги для наполнения страницы текстом*

Тег	Вид	Описание
 	–	Тег перехода на следующую строку
<hr>	–	Добавляет горизонтальную разделительную черту
<img>	–	Тег вставки изображения
… … …	Блочный	Создаёт контейнер, содержащий в себе заголовок уровня *(h1 – самый крупный размер, h6 – самый маленький)*
…	Блочный	Создаёт абзац
<a>…	Строчный	Тег создания ссылок
…	Строчный	Определяет текст насыщенным (полужирным)
…	Строчный	Определяет курсивный текст
_…	Строчный	Определяет подстрочный текст (индекс)
[…]	Строчный	Определяет надстрочный текст (степень)
<u>…</u>	Строчный	Определяет подчёркнутый текст

Выравнивание текста на странице производится с помощью атрибута align – определяет способ горизонтального выравнивания содержимого. Возможные значения: left – по левому краю; center – по центру; right – по правому краю; justify – по ширине.

Пример содержимого:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>My first Web-page</title>
</head>

<body>
    <h1 align="center">Header</h1>
    <p>Content Left</p>
    <p align="right">Content Right</p>
</body>
</html>
```

Результат выравнивания текста по ширине представлен на рис. 1.2.

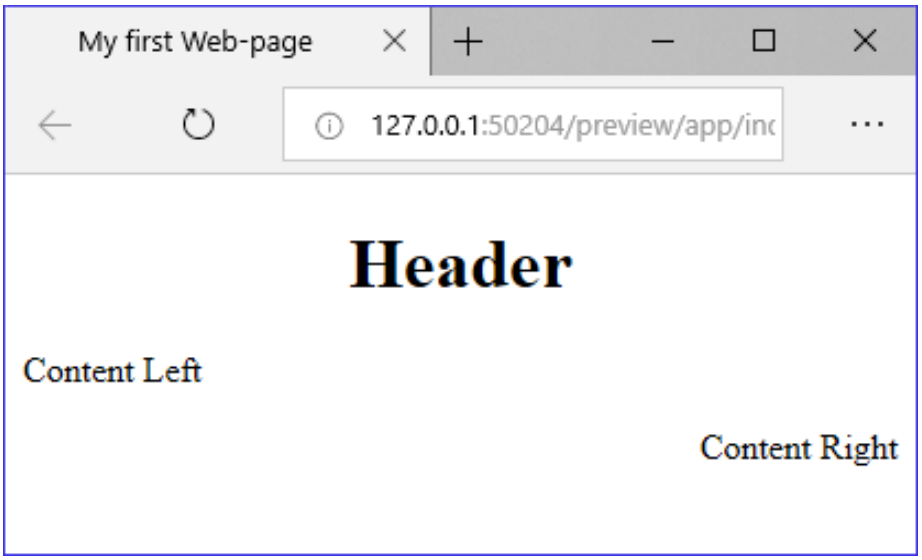


Рис. 1.2. Результат создания веб-страницы с содержимым

Элемент списка. Существует два основных тега для описания списков – `<ol>` и `<ul>`.

Тег `<ol>…</ol>` определяет нумерованный список. Элементы списка каждого из этих тегов создаются при помощи тега `<li>…</li>`.

Вид и тип нумерации задается атрибутом `type`. После знака равенства могут быть следующие значения:

«`1`» – арабские цифры;

«`i`» – маленькие (строчные) римские буквы;

«`I`» – большие (прописные) римские буквы;

«`A`» – заглавные буквы;

«`a`» – строчные буквы.

Тег `<ul>` определяет маркированный список.

Вид маркера также задается атрибутом `type`. Допустимые значения:

«`disc`» – кружок;

«`circle`» – окружность;

«`square`» – квадрат.

Пример нумерованного и маркированного списков (рис. 1.3):

```
<ol type="a">
        <li>First element</li>
        <li>Second element</li>
        <li>Third element</li>
    </ol>
    <ul type="square">
        <li>First element</li>
        <li>Second element</li>
        <li>Third element</li>
    </ul>
```

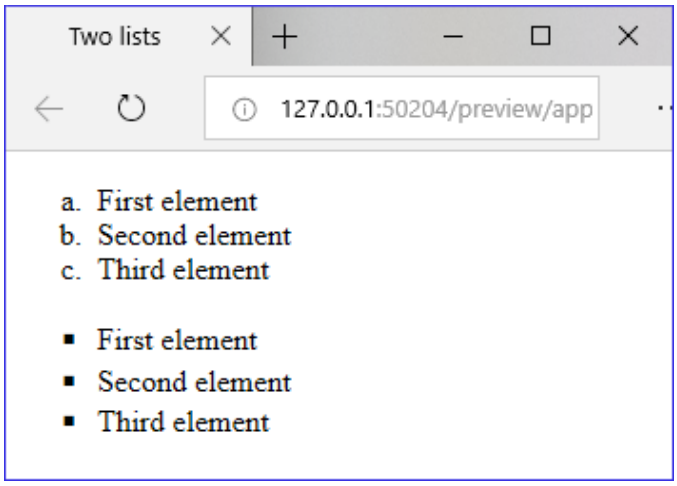


Рис. 1.3. Иллюстрация нумерованного и маркированного списков

Элемент таблицы. Для создания таблицы существует тег `<table>…</table>`. Для создания ряда используется тег `<tr>…</tr>`. Внутри тега с рядом создаются ячейки с данными при помощи тега `<td>…</td>`. Также можно использовать тег `<th>…</th>` для создания ячеек-заголовков (с насыщенным шрифтом и выравниванием по центру).

Атрибуты таблиц в HTML (применимы внутри любого тега таблицы):

- `border` – определяет толщину рамки в таблице;
- `cellpadding` – определяет расстояние между рамкой каждой ячейки таблицы и содержащимися в ней данными, значение – число;
- `cellspacing` – определяет расстояние между границами соседних ячеек таблицы, значение – число;
- `width` – определяет ширину таблицы, значение – число пикселей, возможно указание в процентах, например width=«25%»;
- `height` – определяет высоту таблицы, значение – число пикселей, например height=«25%»;
- `align` – определяет способ горизонтального выравнивания таблицы или содержимого ячеек. Возможные значения: `left` – по левому краю; `center` – по центру; `right` – по правому краю;
- `valign` – определяет способ вертикального выравнивания таблицы или содержимого ячеек таблицы. Возможные значения: `top` – по верхнему краю; `middle` – по середине; `bottom` – по нижнему краю;
- `bgcolor` – определяет цвет фона таблицы, значение – название цвета или его шестнадцатеричный код, например bgcolor=«red», bgcolor=«#000000» (табл. 1.2);
- `background` – позволяет заполнить фон таблицы рисунком. В качестве значения необходимо задать адрес файла с рисунком. Пример представлен на рис. 1.4.

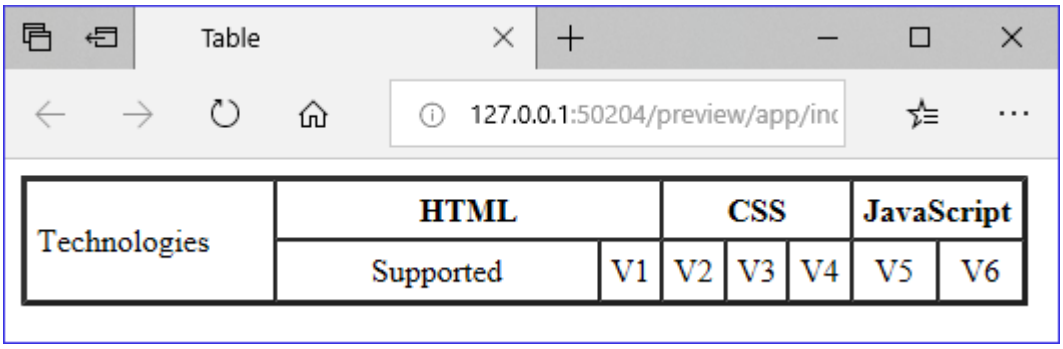
Таблица 1.2. *Цвета в HTML*

Имя цвета	Описание	Шестнадцатеричный код
Black	Черный	#000000
Blue	Синий	#0000FF
Fuchsia	Светло-фиолетовый	#FF00FF
Gray	Темно-серый	#808080
Green	Зеленый	#008000
Lime	Светло-зеленый	#00FF00
Maroon	Темно-красный	#800000
Navy	Темно-синий	#000080
Olive	Оливковый	#808000
Purple	Темно-фиолетовый	#800080
Red	Красный	#FF0000
Silver	Светло-серый	#C0C0C0
Teal	Сине-зеленый	#008080
White	Белый	#FFFFFF
Yellow	Желтый	#FFFF00

Атрибуты таблиц в HTML (применимы внутри ячейки):

- `colspan` – определяет количество столбцов, которые занимает данная ячейка. При этом если ячейка занимает больше одного столбца, то другие занимаемые ячейки создавать не надо;
- `rowspan` – определяет количество рядов, которые занимает данная ячейка.

```
<table border="2" cellpadding="5" cellspacing="0">
   <tr>
      <td rowspan="2" width="25%">Technologies</th>
      <th colspan="2">HTML</th>
      <th colspan="3">CSS</th>
      <th colspan="2">JavaScript</th>
   </tr>
   <tr align="center">
      <td width=150>Supported</td>
      <td>V1</td><td>V2</td><td>V3</td>
      <td>V4</td><td>V5</td><td>V6</td>
   </tr>
</table>
```



Technologies	HTML		CSS			JavaScript	
	Supported	V1	V2	V3	V4	V5	V6

Рис. 1.4. Иллюстрация таблицы

Создание гиперссылок. Гиперссылка – это часть текста, выделенная особым образом, при клике на которую открывается другая веб-страница. Гиперссылка создаётся с помощью тега `<a>…</a>`. Внутри тега пишется текст, который будет отображаться как ссылка, – *адрес*.

Адрес – это путь к HTML-документу или графическому файлу. Адреса бывают абсолютными и относительными. Каждая веб-страница обладает собственным адресом. Этот адрес называется URL (Uniform Resource Location) и является *абсолютным*.

Относительные адреса указываются от корня сайта или текущего документа. Например, код <img src=”image.jpg”> означает загрузить графический файл с именем image.jpg, который располагается в той же папке, что и сама веб-страница. Слэш перед адресом говорит о том, что адресация начинается от корня сайта. Например, ссылка /site/index2.html ведет на документ index2.html, который находится в

папке site. А та, в свою очередь, размещена в корне сайта. Две точки перед именем указывают браузеру перейти на уровень выше в списке каталогов сайта, например ..//frame/1.html. Если перед именем папки нет никаких дополнительных символов вроде точек или слэша, то папка размещена внутри текущего каталога, а уже в ней располагается файл, например images/picture.gif.

Атрибуты:

- `href` – содержит адрес, на который переходит гиперссылка.
- `title` – содержит название, которое будет появляться при наведении на ссылку.

Пример создания ссылок:

```
<a href="table.html" title="Show table">Table</a>
<a href="first_page.html" title="Show first page">My first page</a>
<a href="two_lists.html" title="Show headers">Two lists</a>
```

Все ссылки по умолчанию – синего цвета, а посещенные – фиолетового. Для смены цвета ссылок существует несколько атрибутов:

- `link` – цвет ссылки;
- `alink` – цвет активной ссылки (в момент нажатия на нее);
- `vlink` – цвет посещенной ссылки.

Все цвета задаются при помощи чисел в шестнадцатеричной системе счисления либо одним из базовых цветов. Пример:

```
<body link="red" vlink="green" alink="blue">
```

Следует отметить, что при применении этих атрибутов к тегу `body` все ссылки в документе будут иметь заданный цвет (рис. 1.5).

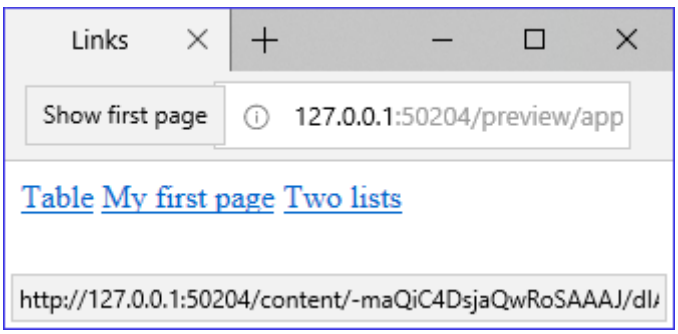


Рис. 1.5. Иллюстрация создания ссылок

Задания

Задание 1. Создайте веб-страницу, демонстрирующую все шесть уровней заголовков, показанных на рис. 1.6.

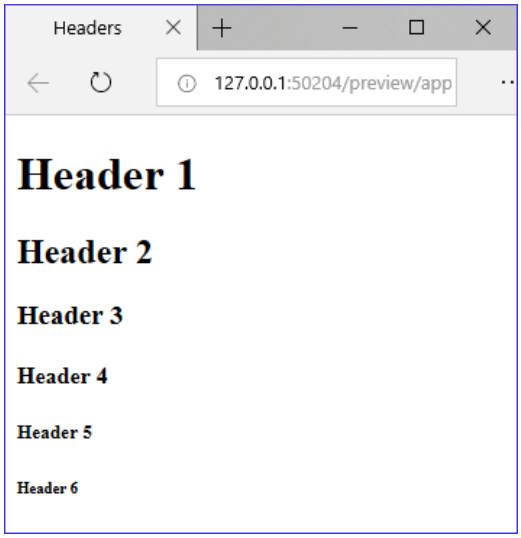


Рис. 1.6. Иллюстрация к заданию 1

Задание 2. Создайте веб-страницу с заголовком 1-го уровня по центру, заголовком 3-го уровня курсивом справа и абзацем текста, выровненным по ширине (рис. 1.7).

В качестве текста используйте этот тестовый текст: «*Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat*».

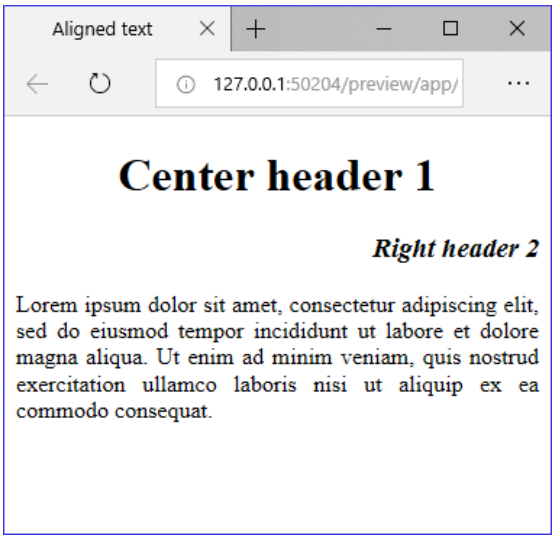


Рис. 1.7. Иллюстрация к заданию 2

Задание 3. Создайте веб-страницу с заголовком 2-го уровня по центру и нумерованным римскими цифрами списком. Во втором элементе списка должен быть маркированный окружностями подсписок, а текст должен быть насыщенным (рис. 1.8).

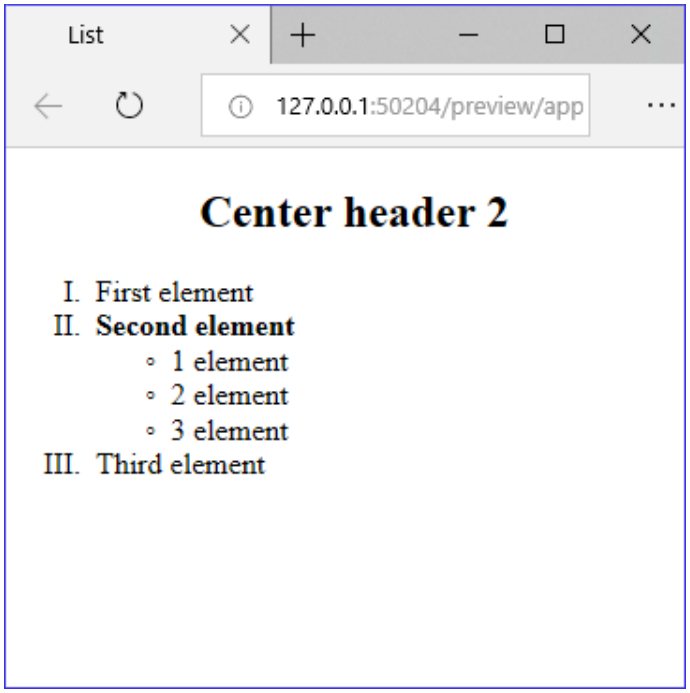


Рис. 1.8. Иллюстрация к заданию 3

Задание 4. Создайте таблицу, общий размер которой будет 300 × 200. В таблице должно быть три равных столбца, каждый из которых должен быть раскрашен своим цветом (рис. 1.9).

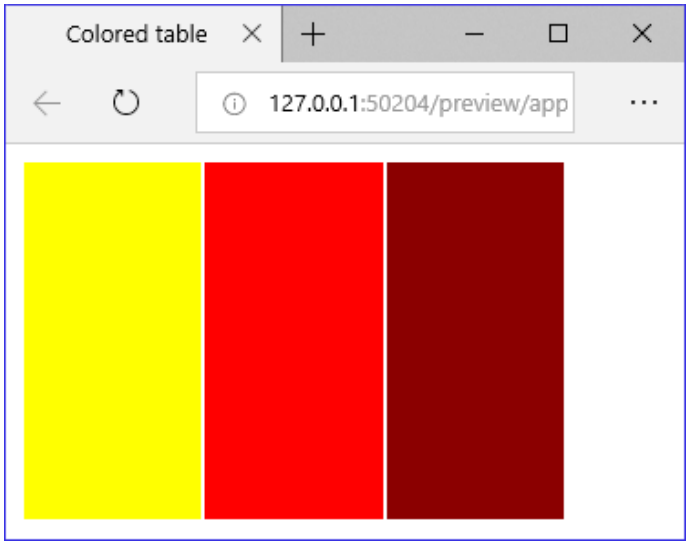


Рис. 1.9. Иллюстрация к заданию 4

Задание 5. Создайте веб-страницу с таблицей. Первый столбец должен иметь заголовок #, ширину 50, в нём пишется номер задания. Во втором столбце должны быть ссылки на выполненные задания; заголовок «Exercise». Третий столбец – объединенный, в нём должны содержаться имя выполнившего задания, заголовок «Name» (рис. 1.10).

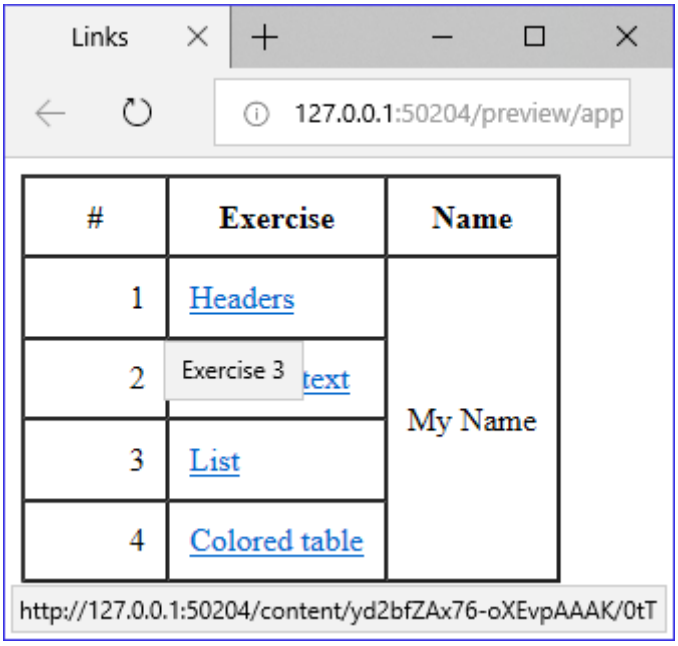


Рис. 1.10. Иллюстрация к заданию 5

Контрольные вопросы

1. Как выглядит структура HTML-документа?
2. Зачем нужен элемент `<!doctype …>`?
3. Какой уровень заголовка самый маленький по размеру?
4. Можно ли поменять тип нумерации или вид маркера в списке?
5. Как выглядит структура таблицы в HTML?
6. Возможно ли раскрасить ячейки таблицы разным цветом?
7. Возможно ли объединить несколько ячеек в одну?
8. Что такое гиперссылка?
9. В каком атрибуте пишется адрес гиперссылки?
10. Какие бывают адреса для гиперссылки?

Лабораторная работа № 2

КАСКАДНЫЕ ТАБЛИЦЫ СТИЛЕЙ CSS

Цель работы

1. Изучить структуру и основы каскадных таблиц стилей CSS.
2. Научиться использовать каскадные таблицы стилей для разметки и создания веб-сайтов.
3. Научиться применять CSS к HTML-элементам.

Ход работы

CSS (от англ. Cascading Style Sheets – «каскадные таблицы стилей») необходимы для описания внешнего вида документа, написанного языком разметки. Обычно CSS-стили применяются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML.

Один CSS-файл может быть использован множеством веб-страниц. Это значит, что если поменять размер шрифта и цвет текста в файле CSS, то они поменяются во всех документах HTML, связанных с ним.

Стиль – это набор атрибутов, определяющих визуальное представление некоторого объекта или тега.

Каскадные таблицы стилей – это набор правил, указаний, атрибутов и их значений, необходимых для составления внешнего представления веб-страницы.

Основной синтаксис языка CSS:

```
Selector1 {
    Attribute1: value;
    Attribute2: value;
    …
}

Selector2 {
    Attribute1: value;
    Attribute2: value;
    …
}
```

Селектор – это имена объектов, которым нужно задать определённый набор свойств и правил, или имена отдельных стилей (наборов особых свойств и правил). Чаще всего это имена тегов или классов оформления.

Атрибут – это свойство или правило, применяемое к объектам веб-страниц.

Для применения разных стилей к одному тегу можно использовать так называемые классы оформления – это те же самые селекторы, только вместо имён тегов используются произвольные названия – идентификаторы.

Пример CSS (рис. 2.1):

```
.header1 {
    color: blue;
}
.header2 {
    color: red;
}
```

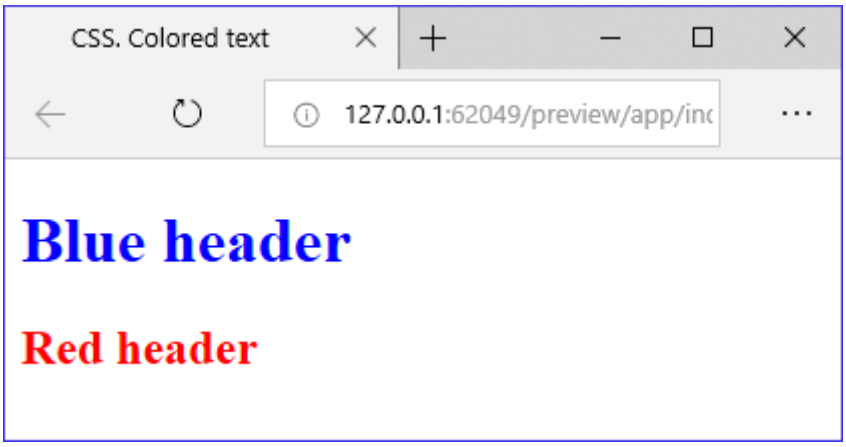


Рис. 2.1. Пример применения CSS

Применение в документе HTML:

```
<h1 class="header1">Blue header</h1>
<h2 class="header2">Red header</h2>
```

Подключение CSS к HTML. CSS подключается к HTML:

- как внутренний стиль;
- встроенные таблицы стилей;
- внешние таблицы стилей.

Внутренний стиль. Стиль задают внутри элемента HTML с помощью атрибута `style`. Как значение этого атрибута используются CSS-атрибуты (без указания селекторов). Если необходимо указать несколько свойств, это делается через запятую. Минус этого способа очевиден: параметр `style` приходится задавать каждому заголовку, поэтому теряется преимущество CSS.

Пример (рис. 2.2):

```
        <h1 style="color: red; text-transform: upper-
case">Red uppercased header</h1>
```

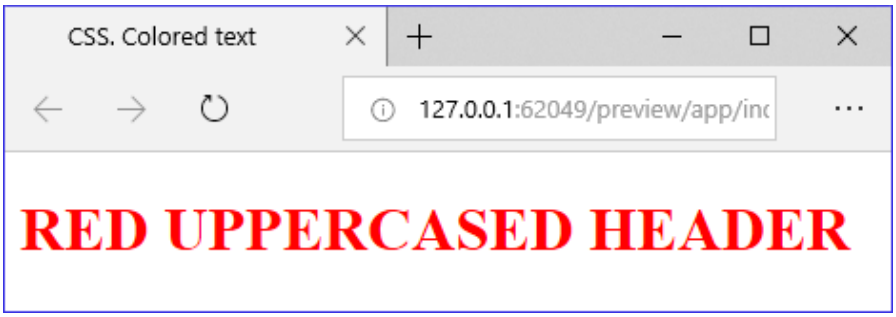


Рис. 2.2. Пример внутреннего CSS

Встроенные таблицы стилей. CSS пишут по всем правилам структуры и помещают его внутрь тега `<style>…</style>` внутри «головы» HTML-документа (`<head>…</head>`). Обязательно следует прописать тип таблицы стилей. Для этого ставится атрибут `type="text/css"` для тега `style`.

Теперь стили, прописанные внутри тега `<style>…</style>`, будут применяться ко всем элементам этого HTML-документа.

Пример (рис. 2.3):

```
        <!doctype html>
        <html>
        <head>
         <meta charset="utf-8">
         <title>CSS. Colored text</title>
         <style type="text/css">
             .redHeader {
                 color: red;
                 text-transform: uppercase;
             }
```

```
 </style>
</head>

<body>
 <h1 class="redHeader">Red uppercased h1</h1>
 <h2 class="redHeader">Red uppercased h2</h2>
 <h3 class="redHeader">Red uppercased h3</h3>
</body>
</html>
```

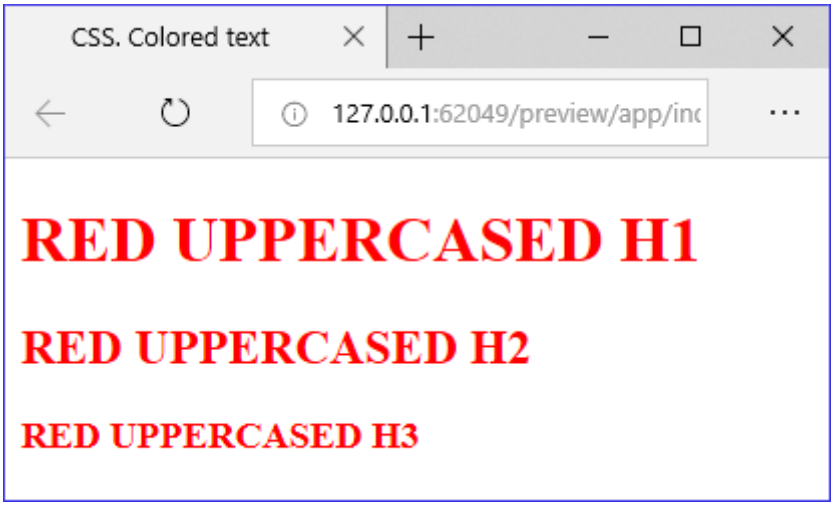


Рис. 2.3. Пример встроенного CSS

Внешние таблицы стилей. CSS по всем правилам и структуре пишется в отдельном файле, который сохраняется со следующим названием: `style_name.css`

Для подключения этого файла CSS необходимо вставить в `head` документа следующий код:

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

Этот способ наиболее предпочтителен, так как позволяет создать единое оформление для многих веб-страниц. Таблицы стилей вынесены в отдельный файл, поэтому создание структуры веб-страницы (в HTML) полностью отделено от создания дизайна (CSS). Это позволяет легко изменить дизайн без изменения структуры.

Пример:
Файл style.css

```
.redHeader {
    color: red;
    text-transform: uppercase;
}
```

Файл red_headers.html

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS. Colored text</title>
 <link     rel="stylesheet"     type="text/css"
href="style.css">
</head>

<body>
 <h1 class="redHeader">Red uppercased h1</h1>
 <h2 class="redHeader">Red uppercased h2</h2>
 <h3 class="redHeader">Red uppercased h3</h3>
</body>
</html>
```

Использование CSS в веб-страницах. Атрибуты фона элемента представлены в табл. 2.1.

Таблица 2.1. *Атрибуты настройки фона CSS*

Атрибут	Описание	Значение
background-color	Цвет фона	Название цвета Шестнадцатеричный код Цвета Цветовые модели
background-image	Изображение фона	Адрес изображения

Окончание табл. 2.1

Атрибут	Описание	Значение
`background-repeat`	Повторение изображения	`no-repeat` *(нет)* `repeat-x` *(по горизонтали)* `repeat-y` *(по вертикали)* `repeat` *(во все стороны)*
`background-attachment`	Прокрутка фона	`fixed` *(нет)* `scroll` *(прокручивается)*

`background-position`	Позиция фонового изображения:
	по вертикали и по горизонтали, например:
	`top left`
	`right bottom`
	`0% 50%`
		`left`
		`right`
		`top`
		`bottom`
		`center`
		`%`
		`px`

`background-size`	Размер фонового изображения:
	сначала ширина, затем высота *(если указать только ширину – высота автоматическая)*
		`cover` *(по ширине экрана)*
		`contain` *(по размерам блока)*
		`px`
		`%`
		и другие единицы измерения

Пример фона (рис. 2.4):

```
body {
 background-color:#A6E4E7 ;
 background-image : url(fruits.jpg);
 background-repeat : repeat-y;
 background-position : center 50px;
 background-size: 250px;
}
```

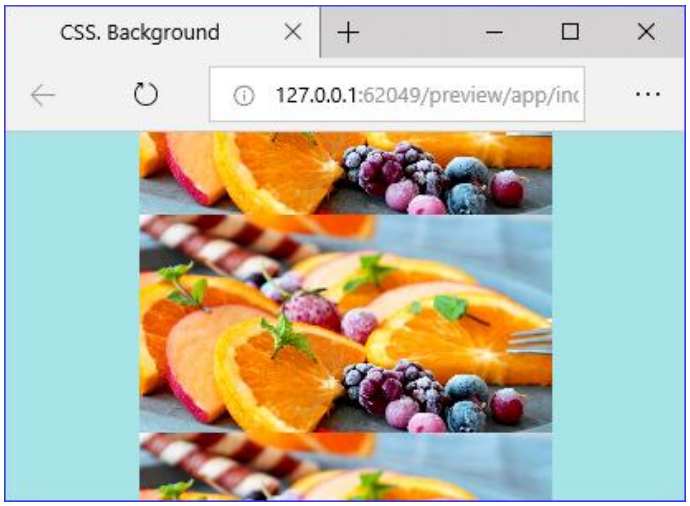


Рис. 2.4. Пример фона CSS по центру

Универсальное свойство `background` позволяет установить одновременно несколько характеристик фона. Значения могут идти в любом порядке – браузер сам определит, какое из них соответствует нужному свойству.

Пример (рис. 2.5):

```
body {
   background: #A6E4E7 url(fruits.jpg) no-repeat;
   background-size: cover;
}
```

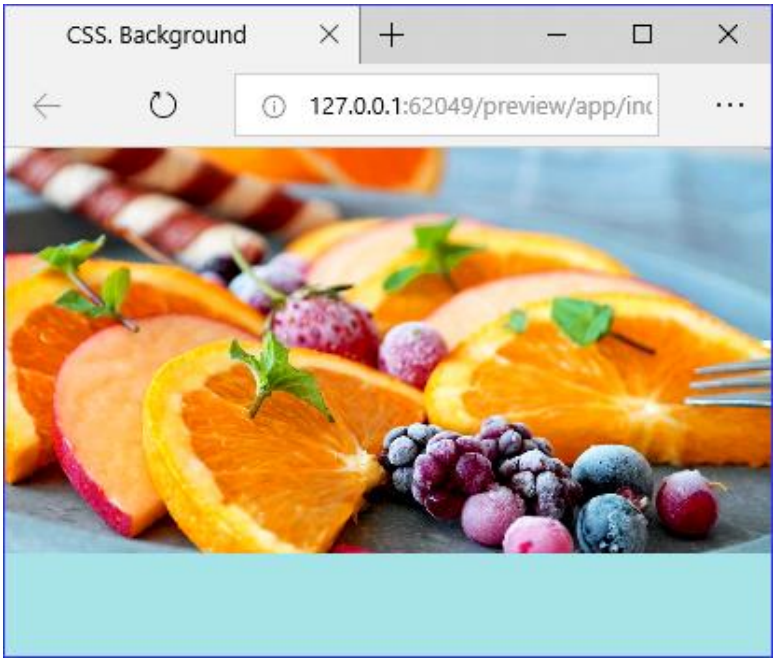


Рис. 2.5. Пример фона CSS по всей ширине

Атрибуты:

- `text-align` – атрибут выравнивания текста внутри элемента.

Значения: `right`, `left`, `center`, `justify`.

- `color` – атрибут цвета текста элемента.

Значения: название цвета, шестнадцатеричный код цвета; также существуют способы вывода цветовых моделей, таких как `rgb` и `hsl`.

- `margin` – отступ от каждого внешнего края элемента.

Значения: px, %, другие единицы измерения.

С помощью `margin` можно выровнять элемент по центру. Для этого свойство необходимо прописать так: `margin: 0 auto;`

- `padding` – отступ от внутреннего края элемента.

Значения: px, %, другие единицы измерения.

- `width` и `height` – ширина и высота блока.

Значения: px, %, другие единицы измерения.

- `border` – атрибут создания границы вокруг элемента. Универсальное свойство `border` позволяет одновременно установить толщину, стиль и цвет границы вокруг элемента. Для установки границы только на определенных сторонах элемента воспользуйтесь свойствами `border-top`, `border-bottom`, `border-left`, `border-right`.

`border` характеризует все свойства, перечисленные в табл. 2.2, и позволяет задавать их в одну строчку.

Таблица 2.2. *Атрибуты настройки границы CSS*

Атрибут	Описание	Значение
`border-color`	Цвет границы	Название цвета Шестнадцатеричный код цвета Цветовые модели

`border-width`	Толщина границы	`px`
		`%`
		и другие единицы измерения

`border-style`	Стиль границы	`dotted` *(точечная)* `dashed` *(пунктирная)* `solid` *(сплошная)* `double` *(двойная)* `groove` *(объемная 1)* `ridge` *(объемная 2)* `inset` *(объемная 3)* `outset` *(объемная 4)*

Пример создания границы (рис. 2.6):
Файл red_headers.html

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8">
 <title>CSS. Border</title>
 <link   rel="stylesheet"   type="text/css"
href="style.css">
</head>

<body>
<ul>
 <li class="dotted">dotted</li>
 <li class="dashed">dashed</li>
 <li class="solid">solid</li>
 <li class="double">double</li>
 <li class="groove">groove</li>
 <li class="ridge">ridge</li>
 <li class="inset">inset</li>
 <li class="outset">outset</li>
 </ul>
</body>
</html>
```

Файл style.css

```
ul li {
   list-style-type: none;
 display: inline;
 border: 5px #C35104;
 padding:3px;
 margin: 3px;
}

.dotted {border-style: dotted}
.dashed {border-style: dashed}
.solid {border-style: solid}
.double {border-style: double}
.groove {border-style: groove}
.ridge {border-style: ridge}
.inset {border-style: inset}
.outset {border-style: outset}
```

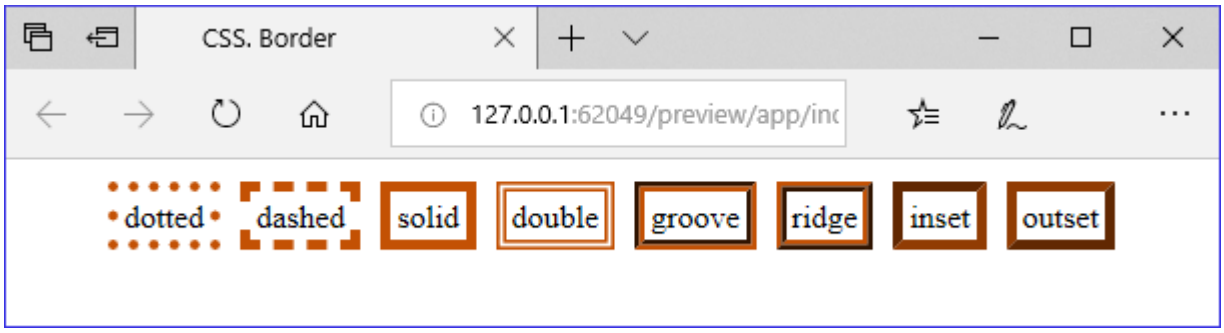


Рис. 2.6. Различные границы

Работа с текстом. Рассмотрим атрибуты работы с текстом (табл. 2.3).

Таблица 2.3. *Атрибуты настройки текста CSS*

Атрибут	Значение	Описание	Пример
`line-height`	normal множитель значение %	Интерлиньяж (меж-строчный интервал)	*line-height: normal* *line-height: 1.5* *line-height: 12px* *line-height: 120%*
`text-decoration`	none underline overline line-through blink	Убрать все оформление Подчеркивание Линия над текстом Перечеркивание Мигание текста	*text-decoration: none*
`text-transform`	none capitalize uppercase lowercase	Убрать все эффекты Начинать С Прописных ВСЕ ПРОПИСНЫЕ все строчные	*text-transform: capitalize*
`text-align`	left right center justify	Выравнивание текста	*text-align: justify*
`text-indent`	значение %	Отступ первой строки	*text-indent: 15px;* *text-indent: 10%*

Работа со шрифтами. Рассмотрим атрибуты для работы со шрифтами (табл. 2.4).

Таблица 2.4. *Атрибуты настройки шрифтов CSS*

Атрибут	Значение	Описание	Пример
`font-family`	имя шрифта	Задает список шрифтов	*P {font-family: Arial, serif}*
`font-style`	normal italic oblique	Нормальный шрифт Курсив Наклонный шрифт	*P {font-style: italic}*
`font-variant`	normal small-caps	Капитель (особые прописные буквы)	*P {font-variant: small-caps}*
`font-weight`	normal lighter bold bolder 100 – 900	Нормальная жирность Светлое начертание Полужирный Жирный 100 – тонкий шрифт, 900 – самый жирный	*P {font-weight: bold}*
`font-size`	normal pt px %	Нормальный размер Пункты Пиксели Проценты	*font-size: normal* *font-size: 12pt* *font-size: 12px* *font-size: 120%*

Работа со списками. Рассмотрим атрибуты работы со списками (табл. 2.5).

Таблица 2.5. *Атрибуты настройки списков CSS*

Атрибут	Значение	Описание	Пример
`list-style-type`	disc circle square decimal lower-roman upper-roman lower-alpha upper-alpha none	Вид маркера. Первые три используются для создания маркированного списка, а остальные – для нумерованного	*LI {list-style-type: circle}* *LI {list-style-type: upper-alpha}*
`list-style-image`	none URL	Устанавливает символом маркера любую картинку	*LI {list-style-image: url(check.gif)}*
`list-style-position`	outside inside	Выбор положения маркера относительно блока строк текста	*LI {list-style-position: inside}*
`list-style`	URL outside inside disc circle square	Универсальное свойство, включает одновременно все вышеперечисленные свойства	*LI {list t-style: list-style-type || list-style-position || list-style-image | inherit}*

Группирование селекторов. При создании стиля для сайта, когда одновременно используется множество селекторов, возможно появление повторяющихся стилевых правил. Для того чтобы не повторять дважды одни и те же элементы, их можно сгруппировать для удобства представления и сокращения кода. Селекторы группируются в виде списка тегов, разделенных между собой запятыми.

Пример:

```
h1, h3 {
        text-align: center;
        color: steelblue;
    }
```

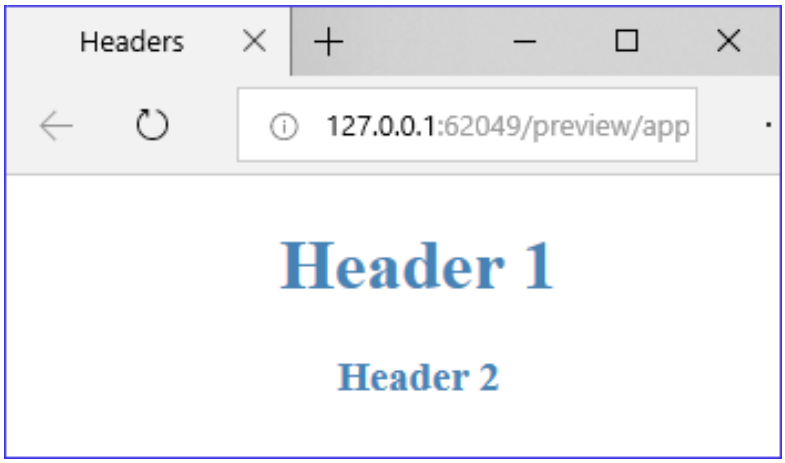


Рис. 2.7. Повторное использование стилей

Задания

Задание 1. Создайте веб-страницу, на которой будут шесть уровней заголовков (рис. 2.8). Используйте заголовки разного цвета. Для задания цвета примените внутренний стиль CSS.

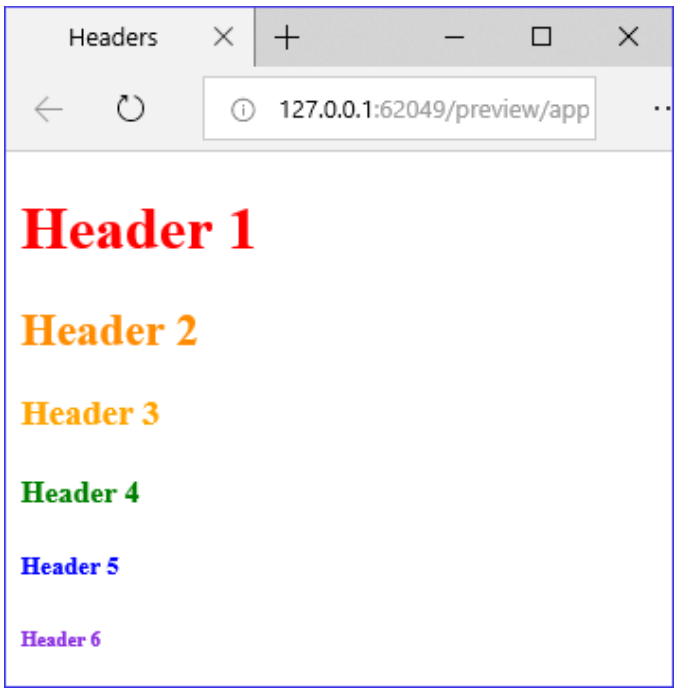


Рис. 2.8. Иллюстрация к заданию 1

Задание 2. Создайте веб-страницу с заголовком 1-го уровня по центру, заголовком 3-го уровня курсивом справа и абзацем текста, выровненным по ширине. Для оформления используйте встроенный CSS в теге `<head>…</head>` (рис. 2.9).

Фон `body` цвета `lightgrey`, фон заголовка `h1` `lightblue`, текст заголовков `h1`, `h2` и граница h1 цвета `steelblue`. В теге `p` свойство `text-indent 50px`.

В качестве текста используйте этот тестовый текст: «*Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat*».

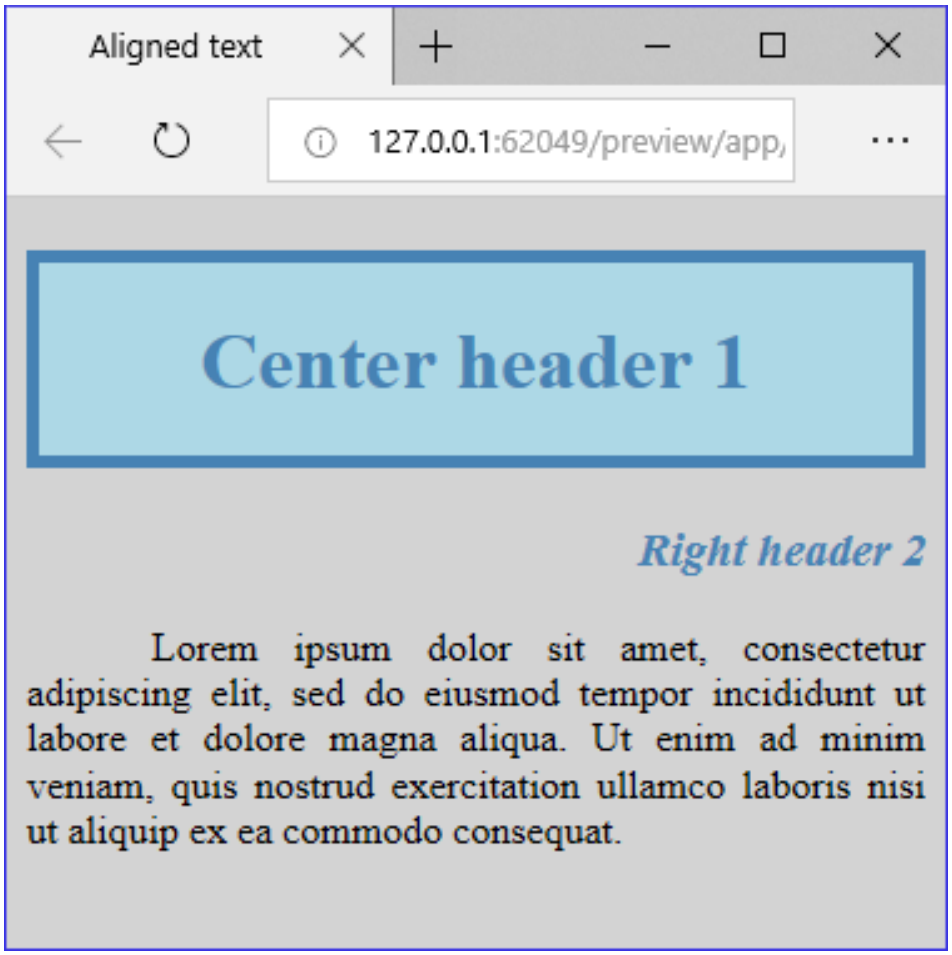


Рис. 2.9. Иллюстрация к заданию 2

Задание 3. Создайте три заголовка 1-го уровня. Для задания условий оформления используйте внешний файл CSS и классы.

У всех заголовков должен быть белый фон. Необходимо подчеркнуть первый заголовок сверху, второй снизу, третий зачеркнуть. `Text-transform` у первого – `uppercase`, у второго – `capitalize`, у третьего – `lowercase`. Все они должны быть разных цветов и разного выравнивания.

На фоне разместите картинку по ширине (как на рис. 2.10).

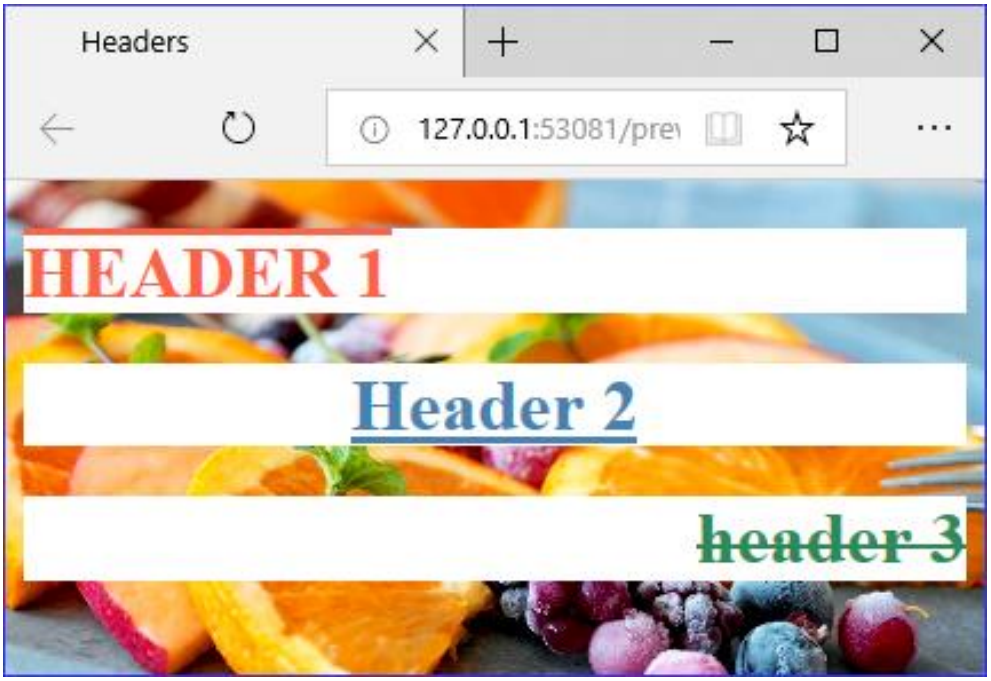


Рис. 2.10. Иллюстрация к заданию 3

Задание 4. Создайте нумерованный список в прямоугольном блоке шириной 250px. Для оформления используйте CSS во внешнем файле. Граница блока должна быть белой и точечной. Задайте фон блока. Для нумерации списка используйте маленькие буквы по алфавиту (рис. 2.11).

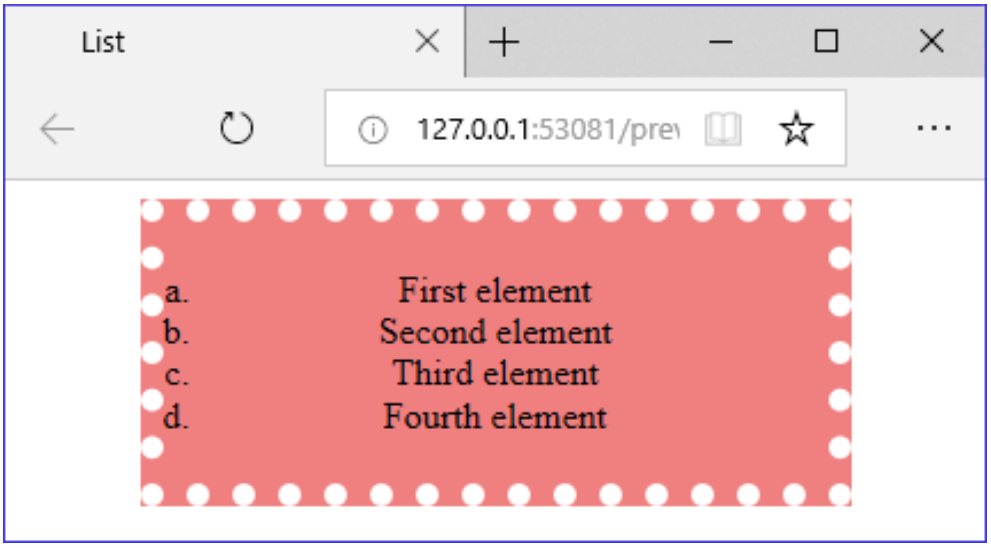


Рис. 2.11. Иллюстрация к заданию 4

Контрольные вопросы

1. Зачем нужны каскадные таблицы стилей CSS?
2. Какова структура CSS-стиля?
3. Как можно подключить CSS-файл к веб-странице?
4. Какие атрибуты служат для настройки фона элемента?
5. Какими способами можно задать цвет?
6. Какое значение `background-size` позволяет растянуть изображение фона по ширине?
7. Какие вы знаете стили границы?
8. Какие настройки шрифтов вы знаете?
9. Что можно изменить в оформлении списка с помощью CSS?
10. Как можно группировать селекторы?

Лабораторная работа № 3

ИНТЕРАКТИВНЫЕ ЭЛЕМЕНТЫ И АНИМАЦИЯ С HTML И CSS

Цель работы

1. Научиться создавать простые интерактивные элементы с помощью HTML и CSS.
2. Рассмотреть простую анимацию в CSS3.

Ход работы

Самые простые интерактивные элементы на веб-странице – это ссылки и их комбинации с другими элементами. К таким элементам относятся:

- ссылки на другие веб-страницы;
- «якори» на странице (ссылки внутри страницы);
- ссылки на адрес электронной почты;
- изображения-ссылки;
- карты-изображения.

Ссылки на другие веб-страницы. Благодаря гиперссылкам пользователи сайта могут посмотреть отдельные фрагменты HTML-документа или открыть интересующую их страницу.

Для создания гиперссылки используется тег `<a>…</a>`. Общий вид тега представлен ниже:

```
<a href=“address” title=“pop-up text”>Hyper-link name</a>
```

Атрибут `href` (сокращение от англ. hypertext reference) указывает на место в документе или на адрес страницы, на что осуществляется ссылка.

При нажатии на текст гиперссылки внутри тега попадают либо на нужное место документа («якорь»), либо на страницу, адрес которой указан в атрибуте `href`.

«Якори» на странице (ссылки внутри страницы). С помощью таких ссылок можно создать своеобразное оглавление веб-страницы. При клике на такую ссылку веб-страница будет прокручиваться до места, где был оставлен «якорь».

Для того чтобы сделать ссылку внутри страницы, сначала необходимо создать «якорь». «**Якорь**» – это специальная метка в теге, на которую может быть ссылка. При создании «якоря» как атрибута тега следует прописать аналогичный код: `id="anchor_name1"` или в теге прописать `<a>` атрибут `name="anchor_name2"`

Ссылки на эти «якоря» выглядят одинаково: `href="#anchor_name1"`. Сначала пишется знак решётки, затем название «якоря». Перед знаком решётки можно указать адрес веб-страницы, где содержится соответствующий «якорь».

«Якорь» может указывать на начало страницы, тогда в ссылке нужно прописать: `href="#top"`

Пример (рис. 3.1):

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Anchors</title>
    <link   rel="stylesheet"   type="text/css"
href="style.css">
</head>

<body>
    <a href="#anchor1">To text</a><br>
    <a href="#anchor2">To continue</a>
    <br><br><br>
    <p id="anchor1">Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor in-
cididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat</p>
    <a name="anchor2">Continue</a><br>
    <a href="#top">To top</a>
</body>
</html>
```

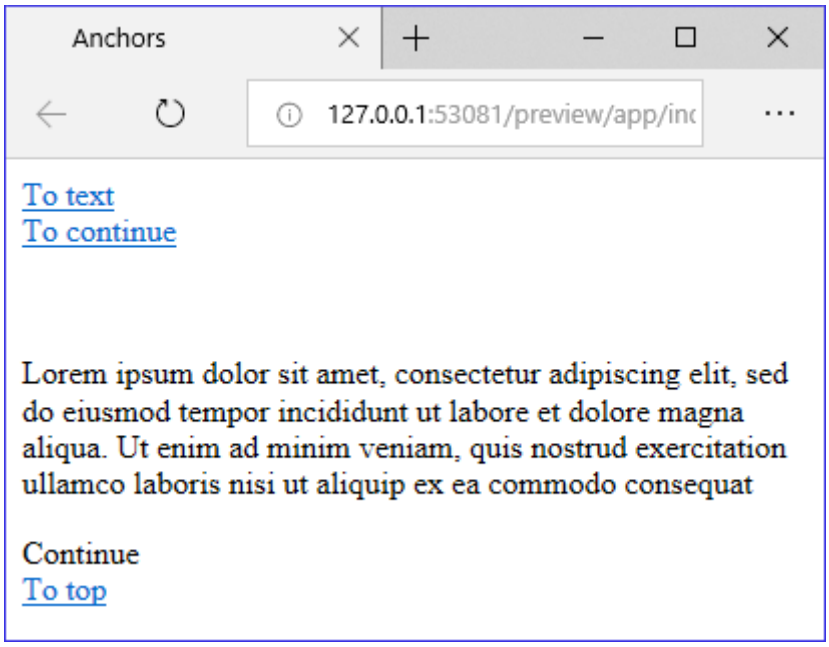


Рис. 3.1. Страница с «якорями» и ссылками на них

Ссылки на адрес электронной почты. Ссылка на адрес электронной почты создается почти так же, как и ссылка на веб-страницу. Только атрибут `href` имеет следующий вид: `href="mailto:e-mail_address"`. При нажатии на такую ссылку запускается почтовая программа. Пример:

```
<a href="mailto:test@site.com">Send email</a>
```

Изображения-ссылки. Изображения помогают создавать общую функциональную и графическую структуру. Поддерживаются три формата изображений, которые можно вставить в веб-страницу:

- gif;
- jpg/jpeg;
- png.

Для вставки изображений на страницу используется тег `<img>`. Синтаксис тега в соответствии со стандартами HTML выглядит так:

```
<img src="image_address" alt="alternative text">
```

Данная конструкция не требует закрывающего тега.

Атрибуты тега `<img>`:

- `align` – определяет выравнивание картинки. Значения: `left` и `right`. При использовании изображений в абзаце данный атрибут задает обтекание изображения текстом;

• hspace и vspace – отступы в пикселях по горизонтали или вертикали от картинки до других объектов документа:

• height и width – высота и ширина изображения в пикселях или процентах. Пример: width=“20%”;

• alt – если браузер не может найти изображение, то отображает текст, указанный в атрибуте вместо картинки;

• title – задает всплывающий текст, который отображается при наведении на изображение;

• border – задает ширину рамки вокруг изображения (в пикселях). Атрибут border со значением 0 необходим, чтобы убрать рамку, которая появляется вокруг изображения-ссылки.

Когда изображение помещено внутрь тега <a>…, оно становится интерактивным и выполняет роль ссылки. Это часто применяется при создании графического меню. Для изображений-ссылок необходимо отдельно прописывать дизайн CSS, так как дизайн текстовых ссылок отображаться не будет.

Изображение также можно сделать ссылкой – для этого достаточно поместить тег изображения внутри тега ссылки. Пример перехода на сайт при нажатии на изображение из файла picture.jpg:

```
<a href=“link_address”>
   <img src=“picture.jpg” border=”0”>
</a>
```

Карты-изображения. Карты-изображения позволяют привязывать ссылки к разным областям одного изображения.

Для создания карты-изображения в теге <img> применяется атрибут usemap. Он является ссылкой на описание карты под тегом <map>. Значение атрибута name данного тега должно соответствовать имени в usemap. Для задания области-ссылки используется тег <area>.

Атрибуты тега <area>:

• shape – определяет форму активной области.

Значения: circle *(окружность)*, rect *(прямоугольник)*, poly *(полигон)*.

• alt – добавляет альтернативный текст для каждой области; служит лишь комментарием для ссылки, поскольку на экран не выводится.

• coords – задает координаты активной области. Они отсчитываются в пикселях от левого верхнего угла изображения, которому соответствует значение 0,0. Первое число является координатой по горизонтали, второе – по вертикали. Список координат зависит от формы области.

Для окружности задаются три числа – координаты центра круга и радиус:

```
   <area    shape="circle"    coords="230,340,    100"
href="circle.html">
```

Для прямоугольника задаются координаты левого верхнего и правого нижнего угла:

```
   <area    shape="rect"    coords="24,18,    210,56"
href="rect.html">
```

Для полигона задаются координаты его вершин:

• href – определяет адрес ссылки для области. Правила записи такие же, как и для тега <a>….

Пример кода карты-изображения (рис. 3.2):

```
          <!doctype html>
          <html>
          <head>
           <meta charset="utf-8">
           <title>Map</title>
          </head>

          <body>
            <p><img     src="map.gif"     width="411"
height="46" usemap="#map" alt="Navigation"></p>
            <p><map  name="map">  <area  shape="poly"
alt="Bookmark 2"
              coords="210,27,  203,9,  202,6,  197,2,
192,1,  120,1,  115,2,  110,6,  112,9,  119,27,  119,32,
211,32, 210,27"
              href="Link2.html">
              <area shape="poly" alt="Bookmark 3"
```

```
                coords="302,27,  295,9,  293,6,  289,2,
283,1,  212,1,  206,2,  202,6,  203,9,  210,27,  211,32,
284,32, 303,32, 302,27" href="Link3.html">
                <area shape="poly" alt="Bookmark 4"
                coords="302,27, 303,32, 394,32, 393,27,
386,9,  382,3,  375,1,  303,1,  298,2,  293,6,  295,9,
302,27"
                href="Link4.html">
              </map></p>
             </body>
            </html>
```

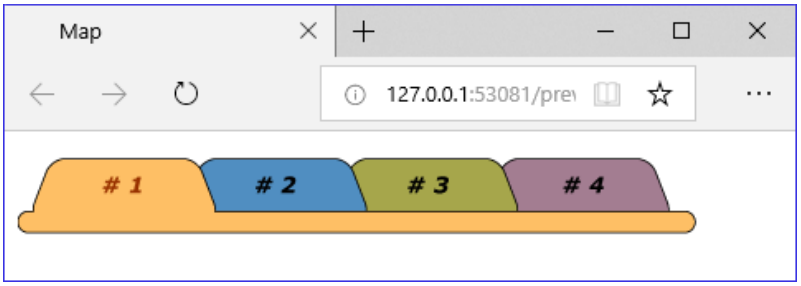


Рис. 3.2. Карта-изображение

Оформление ссылок в HTML. По умолчанию ссылки – синего цвета, а после посещения – фиолетового. Для смены цвета ссылок существует несколько атрибутов, которые прописываются в теге `<body>`:

- `link` – цвет ссылки;
- `alink` – цвет активной ссылки (в момент нажатия на нее);
- `vlink` – цвет посещенной ссылки.

Все цвета задаются при помощи чисел в шестнадцатеричной системе счисления либо одним из базовых цветов. Пример:

```
<body link="red" vlink="green" alink="blue">
```

Следует отметить, что при применении этих атрибутов к тегу «body» все ссылки в документе будут иметь заданный цвет.

Оформление ссылок в CSS. Ссылки можно отображать как динамические объекты с несколькими состояниями. **Псевдоклассы** – определяют стили состояния элемента. Псевдокласс `:hover` можно применять не только к ссылкам, но и к другим элементам (табл. 3.1).

Таблица 3.1. *Псевдоклассы для ссылок*

Атрибут	Описание
`a:link`	Стиль для обычной ссылки
`a:visited`	Стиль для посещенной ссылки
`a:active`	Стиль для ссылки в момент нажатия на неё
`a:hover`	Стиль для ссылки при наведении на нее мышью

Пример:

```
a:link {color: orange}
a:visited {color: green}
a:active {color: red}
a:hover {color: blue}
```

Простая анимация в CSS. Анимация – это плавное изменение свойства элемента при взаимодействии с ним. В стиле CSS прописывается атрибут, который указывает, что свойство будет анимироваться при помощи специальных CSS-правил. При изменении этого свойства браузер сам обработает анимацию.

Атрибуты, задающие анимацию, представлены в табл. 3.2.

Таблица 3.2. *Атрибуты анимации CSS*

Атрибут	Описание	Значение

`transition-property`	Список свойств, которые будут анимироваться	`all` *(анимировать все свойства)*
		`margin`
		`height`
		`color`
		и другие…

`transition-duration`	Продолжительность анимации	`s` *(количество секунд)* `ms` *(количество миллисекунд)*

`transition-timing-function`	Вид распределения анимации во времени *(например, начнётся ли анимация медленно, чтобы потом ускориться, или наоборот)*	`cubic-bezier(0, 0, 1, 1)`
		`ease`
		`ease-in`
		`ease-out`
		`ease-in-out`
		`steps(9, start)`

`transition-delay`	Задержка до анимации *(например, если значение 1s, то начнётся через 1 с)*	`s` *(количество секунд)* `ms` *(количество миллисекунд)*

Универсальное свойство `transition` может перечислять все свойства в следующем порядке: `property duration timing-function delay`, а также задавать анимацию нескольких свойств сразу.

Пример анимации при наведении на элемент (рис. 3.3):

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Animated square</title>
  <style type="text/css">
    .animated {
      transition-property: background-color height;
      transition-duration: 5s;
      transition-timing-function: ease-in-out;
      transition-delay: .5s;

      width: 150px;
      height: 150px;
      background-color: blue;
    }
    .animated:hover {
        height: 50px;
        background-color: tomato;
    }
  </style>
</head>

<body>
  <p class="animated"></p>
 </body>
</html>
```

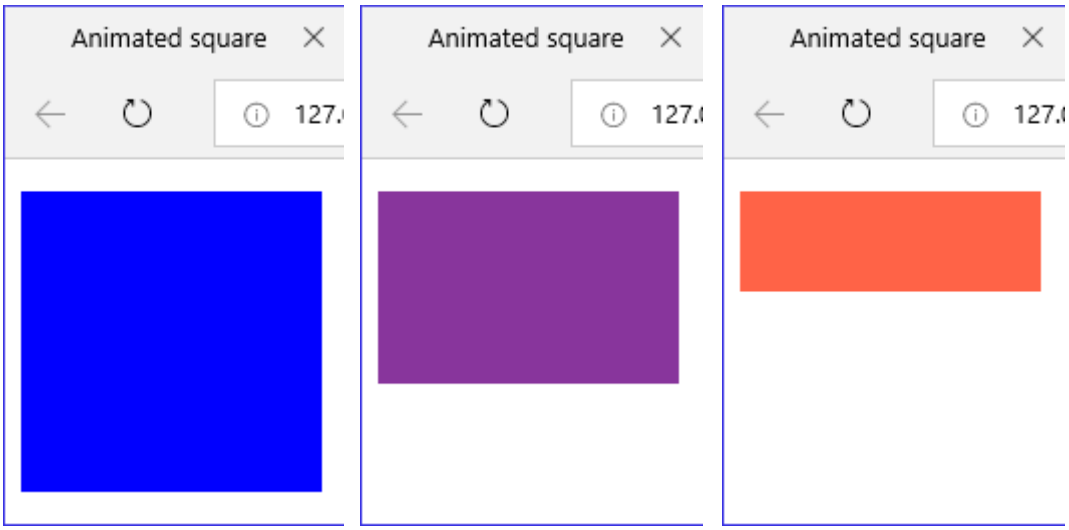


Рис. 3.3. Анимация цвета и высоты квадрата

Задания

Задание 1. Создайте веб-страницу с текстом, выровненным по ширине, в конце которого будет ссылка «Наверх», возвращающая к вершине сайта (рис. 3.4).

В качестве текста используйте этот тестовый текст: «*Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum».*

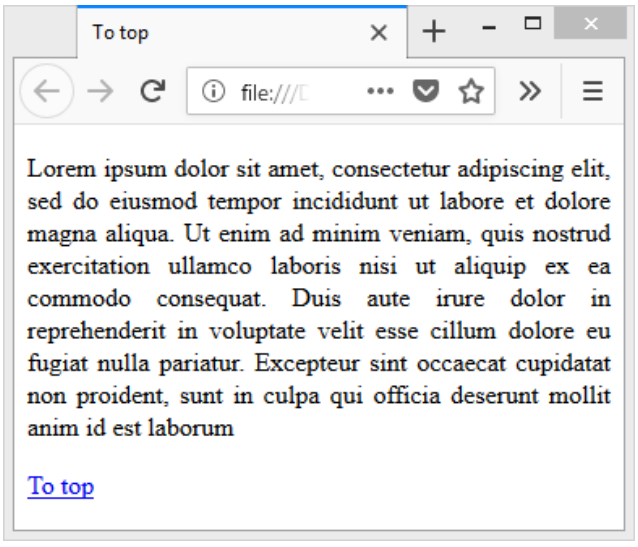


Рис. 3.4. Иллюстрация к заданию 1

Задание 2. Создайте веб-страницу с пятью абзацами текста с заголовками. Каждый абзац и его заголовок должны быть своего цвета. В начале страницы создайте «оглавление» со ссылками на каждый заголовок. Цвет абзацев задайте с помощью CSS. Пример представлен на рис. 3.5.

В качестве текста абзаца используйте этот тестовый текст: «*Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat*».

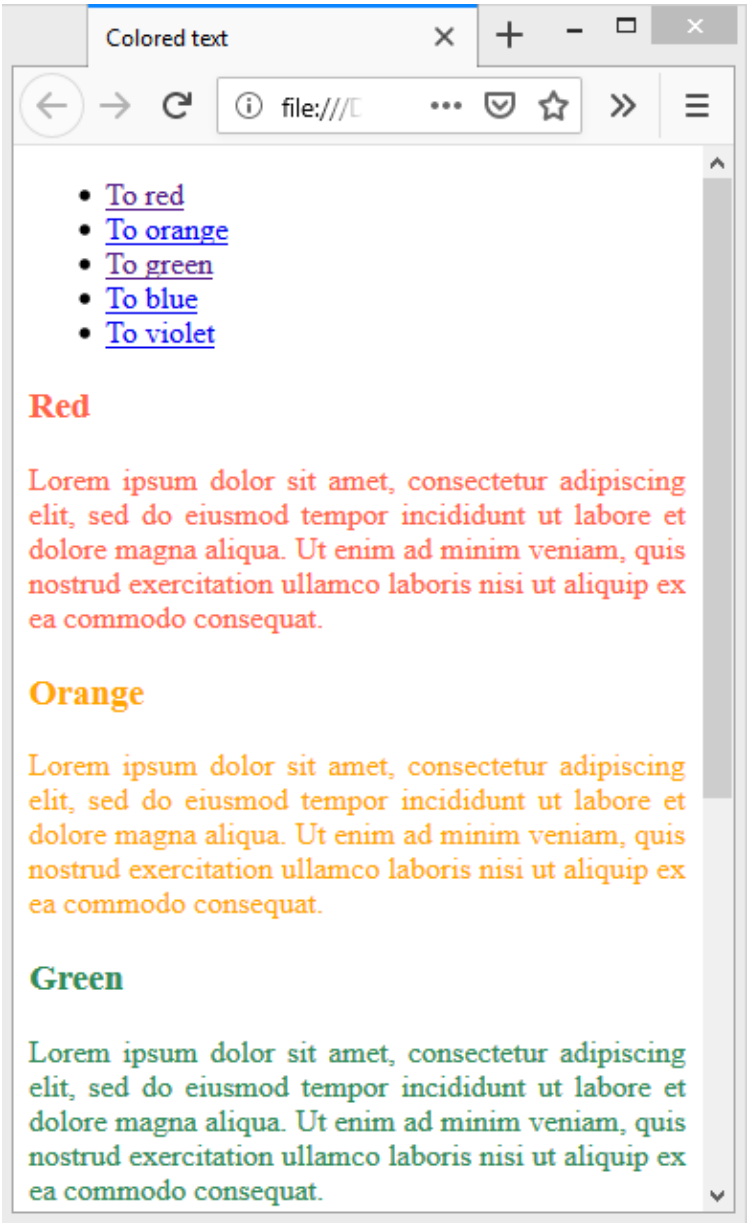


Рис. 3.5. Иллюстрация к заданию 2

Задание 3. Для выполнения этого задания используйте веб-страницу из задания 2.

С помощью HTML задайте цветовое оформление ссылок на странице. Ссылка должна быть зелёной, посещённая ссылка – фиолетовой, активная – оранжевой. Пример представлен на рис. 3.6.

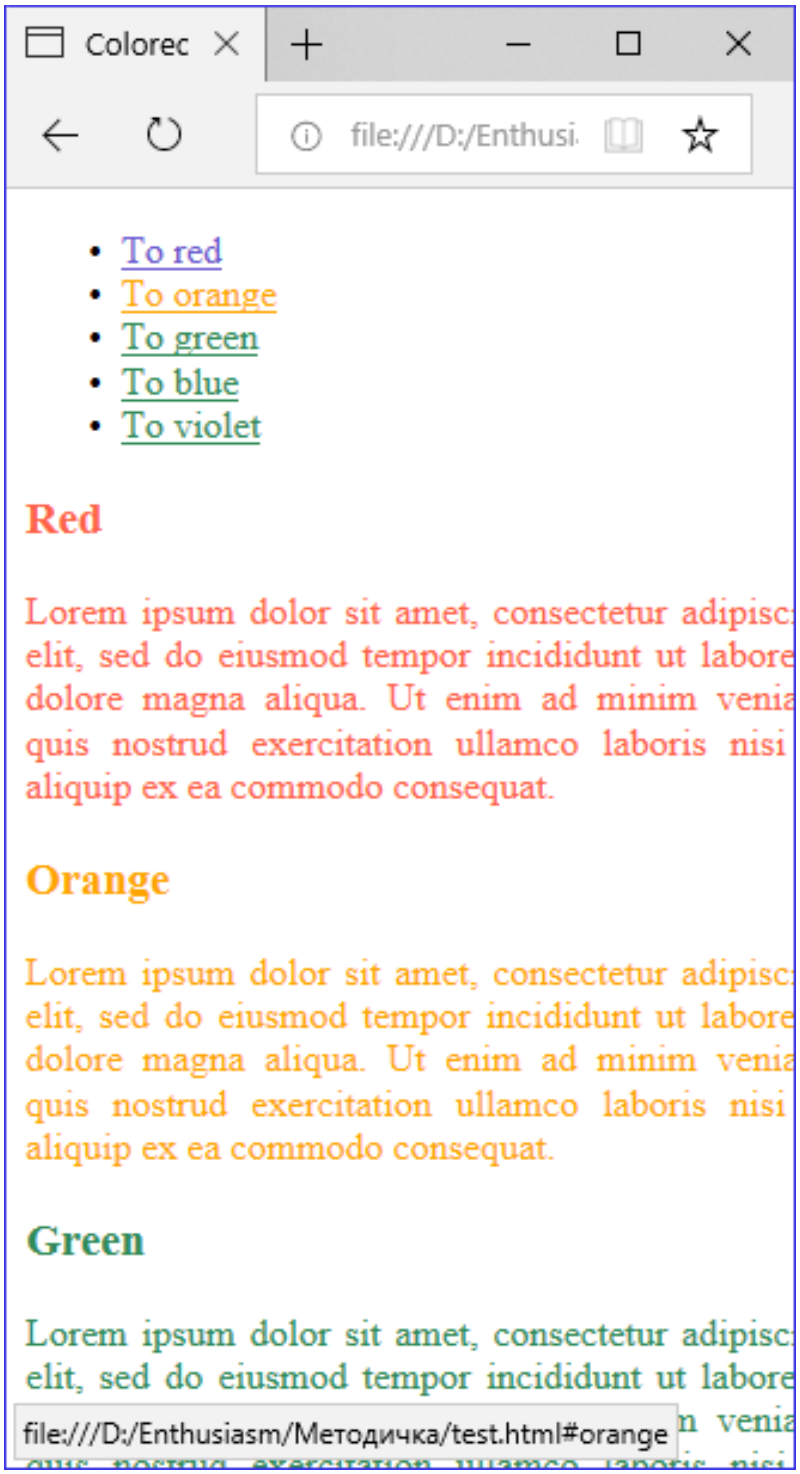


Рис. 3.6. Иллюстрация к заданию 3

Задание 4. Создайте три заголовка первого уровня. Внутри тега заголовка пропишите ссылки. Для задания оформления используйте внешний файл CSS и классы. Каждый заголовок должен иметь свой класс.

Задайте каждой ссылке свои свойства с помощью классов. При наведении курсора на ссылку она должна отображаться на экране в следующем виде: подчеркнуть первую ссылку сверху, вторую снизу, третью зачеркнуть.

Цвета обычной ссылки, активной, посещенной и ссылки при наведении должны различаться. У каждого класса должна быть своя цветовая гамма.

Пример показан на рис. 3.7.

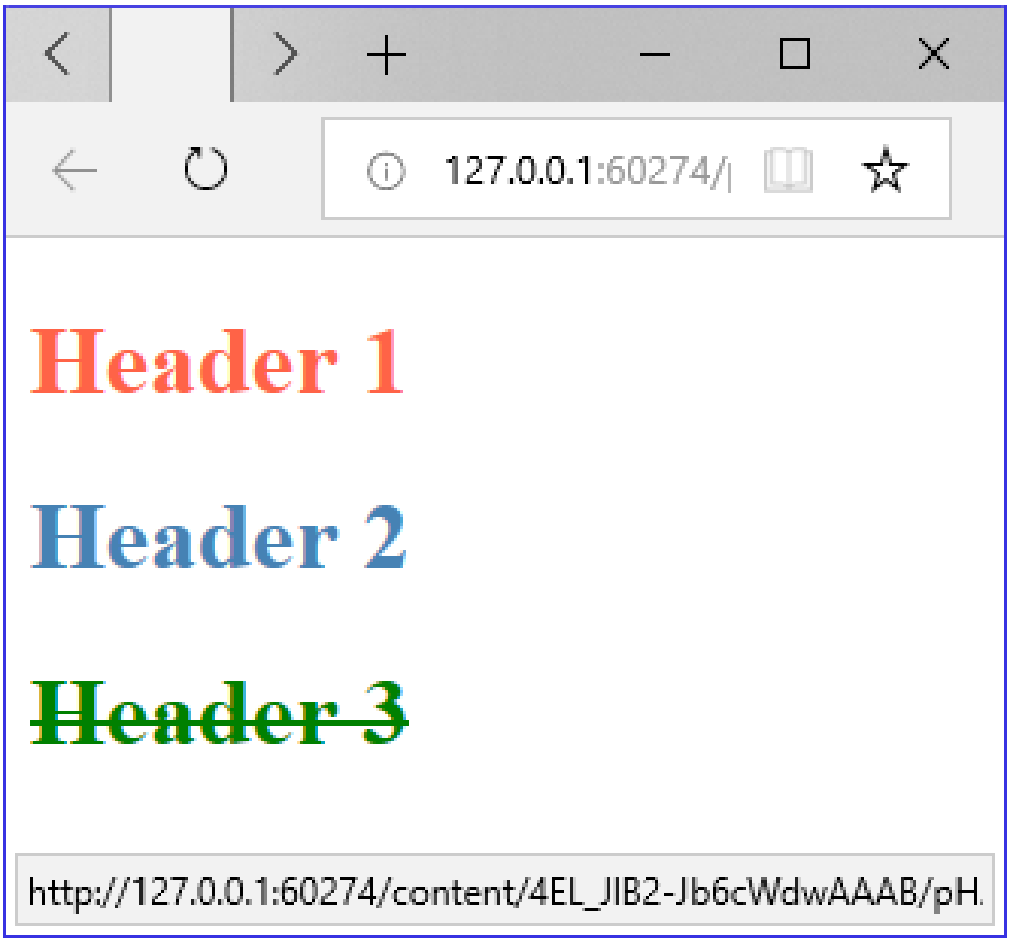


Рис. 3.7. Иллюстрация к заданию 4

Задание 5. Для выполнения задания используйте веб-страницу, созданную в задании 4.

Задайте ссылкам анимацию. Для этого в файле CSS пропишите класс каждого заголовка и за ним через пробел – название тега ссылки. В таком блоке укажите параметры анимации: задержка – 1 с, длительность анимации – 3 с, распределение анимации по времени – `ease`. Используйте все свойства (`all`).

Контрольные вопросы

1. Какие интерактивные элементы можно создать средствами HTML?
2. Как создать «якорь» на странице?
3. Как сделать изображение-ссылку?
4. Какие фигуры могут быть использованы для задания `area`?
5. Откуда считаются координаты в карте-изображении?
6. Какие псевдоклассы применимы к ссылкам?
7. Что такое анимация?
8. В каких единицах времени можно задать задержку?
9. Как анимировать все свойства сразу?
10. Как задать скорость протекания анимации?

Лабораторная работа № 4

БЛОЧНАЯ ВЁРСТКА HTML5

Цель работы

1. Ознакомиться с семантическими элементами HTML5.
2. Освоить блочную верстку сайта с помощью семантических элементов и их CSS-позиционирования.

Ход работы

Семантические элементы HTML5 доступно описывают свой смысл, или назначение, как для браузеров, так и для веб-разработчиков. Они имеют смысловые названия и подразумевают наполнение определёнными элементами.

До появления стандарта HTML5 вся разметка страниц выполнялась преимущественно с помощью элементов `<div>`, которым присваивали классы `class` или идентификаторы `id` для наглядности разметки (например, `<div id="header">`). С их помощью в HTML-документе размещали верхние и нижние колонтитулы, боковые панели, навигацию и многое другое.

Стандарт HTML5 предоставил новые элементы для структурирования, группировки контента и разметки текстового содержимого. Новые семантические элементы позволили улучшить структуру веб-страницы, добавив смысловое значение заключенному в них содержимому (было `<div id="header">`, стало `<header>`).

Элемент `<header>…</header>`. Представляет собой вводный контент, обычно группу вводных или навигационных средств. Он может содержать другие элементы-заголовки, а также логотип, форму поиска, имя автора и другие элементы.

Элемент `<nav>…</nav>`. Подразумевает хранение панели навигации веб-сайта. В этом элементе выводится группа главных ссылок сайта, которые перенаправляют пользователя на страницы структуры сайта (например, «О сайте», «Главная», «Категории»). Панелей навигации с элементом `<nav>…</nav>` может быть несколько (например, отдельная панель пользователя – «Регистрация», «Вход», «Личный кабинет»).

Есть отдельные принятые правила создания панели навигации (меню). Меню создается в семантическом элементе <nav>…</nav>. Так как меню – это перечень ссылок, ссылки пишутся в виде списка .

```
<nav>
      <ul>
         <li><a href="about.html">About</a>
         <li><a href="index.html">Home</a>
         <li><a href="cat.html">Categories</a>
      </ul>
</nav>
```

По умолчанию элементы списка расположены вертикально и имеют маркеры, а ссылки – единый дизайн по всему сайту. С помощью CSS можно определять стиль и вид текста:

```
nav ul {
 list-style-type: none; /*remove list msarkers*/
 margin: 0; /*remove useless auto margin*/
 padding-left: 0; /*remove useless auto padding*/
 text-align: center; /*place menu at the center*/
}
nav ul li {
 display: inline-block; /*show in one line*/
 margin: 5px; /*margin between elements*/
 padding: 10px; /*padding inside element*/
}
nav a {
text-decoration: none; /*remove link underline*/
}
```

После этого можно провести гибкую настройку для создания дизайна. Для этого подойдут анимация и псевдоклассы из предыдущей работы.

В приведенном ниже примере кода CSS использовано свойство, которое определяет размер шрифта «`font-size: 1.2em;`», где `em` – новая единица измерения; `1em` – размер стандартного шрифта (по

умолчанию 12px); 1.2em обозначает, что размер шрифта меню равен размеру обычного текста, умноженному на 1,2.

В коде также использованы знаки /* … */ – это обозначение комментариев, т. е. тех строк, которые помогают разработчику ориентироваться в коде, но не отображаются на экране.

Пример верхней части сайта с панелью навигации:

Файл header.html

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8">
 <title>Header</title>
 <link     rel="stylesheet"     type="text/css"
href="style.css">
</head>

<body>
  <header>
   <h1>Site name</h1>
   <h3>Site description</h3>
   <nav>
       <ul>
          <li><a href="about.html">About</a>
          <li><a href="index.html">Home</a>
          <li><a href="cat.html">Categories</a>
       </ul>
   </nav>
 </header>
  </body>
</html>
```

Файл style.css

```
header {font-family: 'Trebuchet MS', 'Arial'}
header h1 {
 text-align: center;
}
```

```
        header h3 {
        text-align: right;
        }
        nav ul {
        list-style-type: none; /*remove list msarkers*/
        margin: 0; /*remove useless auto margin*/
        padding-left: 30%; /*remove useless auto pad-
ding*/
        text-align: center; /*place menu at the cen-
ter*/
        background-color: #EFEFEF;
        background-image: url("fruits.jpg");
        background-size: 30%;
        background-position: center left;
        background-repeat: no-repeat;
        height:50px;
        }
        nav ul li {
        display: inline-block; /*show in one line*/
        margin: 5px; /*margin between elements*/
        padding: 10px; /*padding inside element*/
        transition: all 0.5s ease-in-out; /*ainmation*/
        border-bottom: 0px solid red;
        }
        nav a {
        text-decoration: none; /*remove link under-
line*/
        font-size: 1.2em;
        color: tomato;
        transition: all 0.5s ease-in-out; /*ainmation*/
        }
        nav a:hover {/*link under pointer*/
        color: red;
        }
        nav ul li:hover { /*element under pointer*/
        border-bottom: 5px solid red;
        }
```

На рис. 4.1 – 4.3 представлено то, что получилось и что могут видеть все: готовая верхняя часть сайта с панелью навигации.

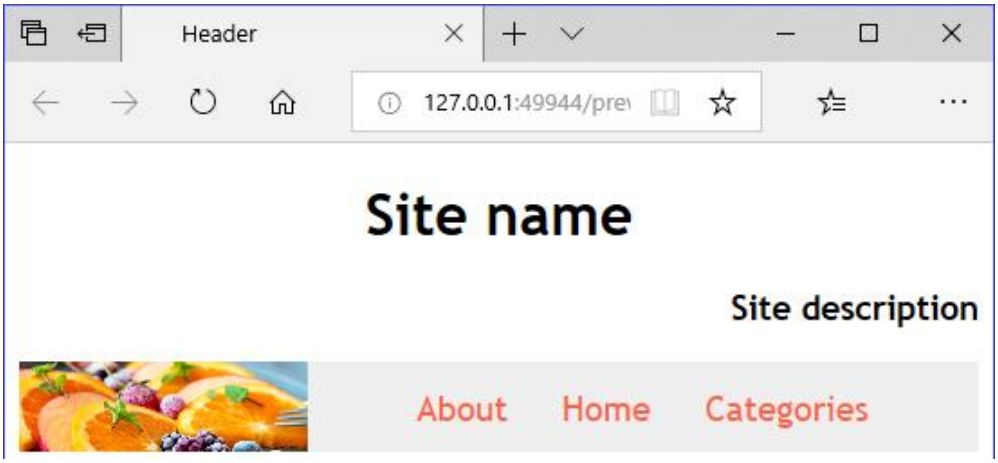


Рис. 4.1. Панель навигации до наведения курсора

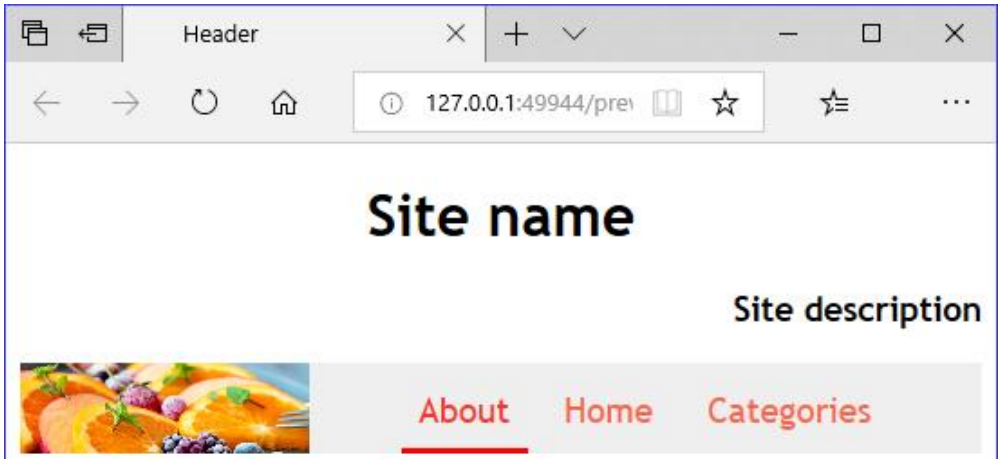


Рис. 4.2. Панель навигации при наведении курсора

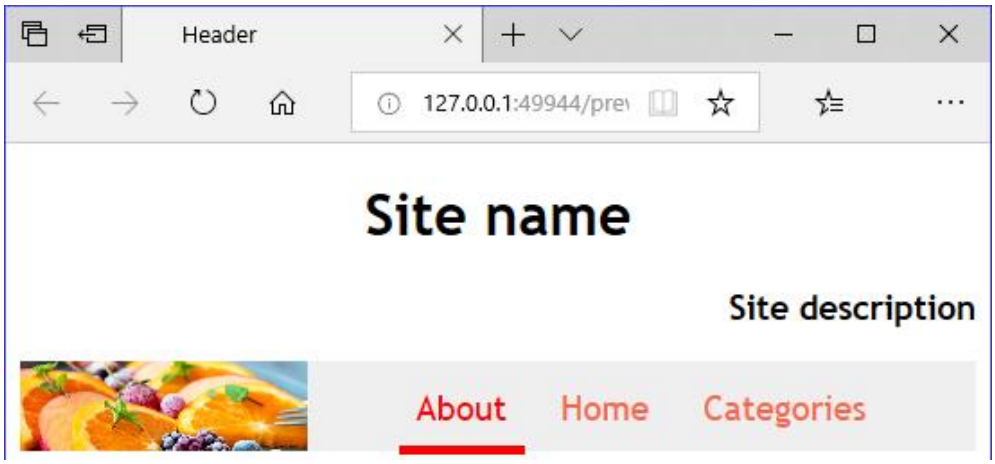


Рис. 4.3. Панель навигации с наведением курсора

Элемент <article>…</article>. Представляет собой блок многократного использования с содержимым вроде записей, статей или комментариев. Необходим для придания однотипности оформления схожему по смыслу содержимому. Обычно содержит элемент заголовка <header>…</header>.

Элемент <section>…</section>. Представляет собой логическую секцию страницы, например оглавление, раздел. Может содержать внутри элементы <article>…</article> или, наоборот, делить элемент <article>…</article> на разделы.

Элемент <aside>…</aside>. Предполагает содержимое, не относящееся к информации страницы напрямую: если убрать это содержимое, то смысл страницы не изменится. Это может быть рекламный блок, цитата, побочные новости.

Элемент <footer>…</footer>. Представляет собой нижний колонтитул элемента или всей страницы. Содержит информацию об авторе, контактную информацию, сведения об авторских правах, дружественные баннеры и т. п. Страница допускает использование нескольких элементов <footer>…</footer>.

Элемент <address>…</address>. Используется для определения контактной информации автора/владельца документа или статьи. Для обозначения автора документа тег размещают внутри элемента <body>, для отображения автора статьи – внутри тега <article>. В браузере обычно отображается курсивом.

Элемент <main>…</main>. Представляет основное содержимое документа (содержимое элемента <body>). Контент, находящийся внутри элемента, должен быть уникальным и не повторяться. Элемент <main> не может быть потомком таких элементов, как <article>, <aside>, <footer>, <header> или <nav>.

Перечисленные элементы представляют собой основные блоки веб-страницы. В блоках размещено содержимое определённого назначения. В упрощенном виде веб-страница – это набор блоков (рис. 4.4).

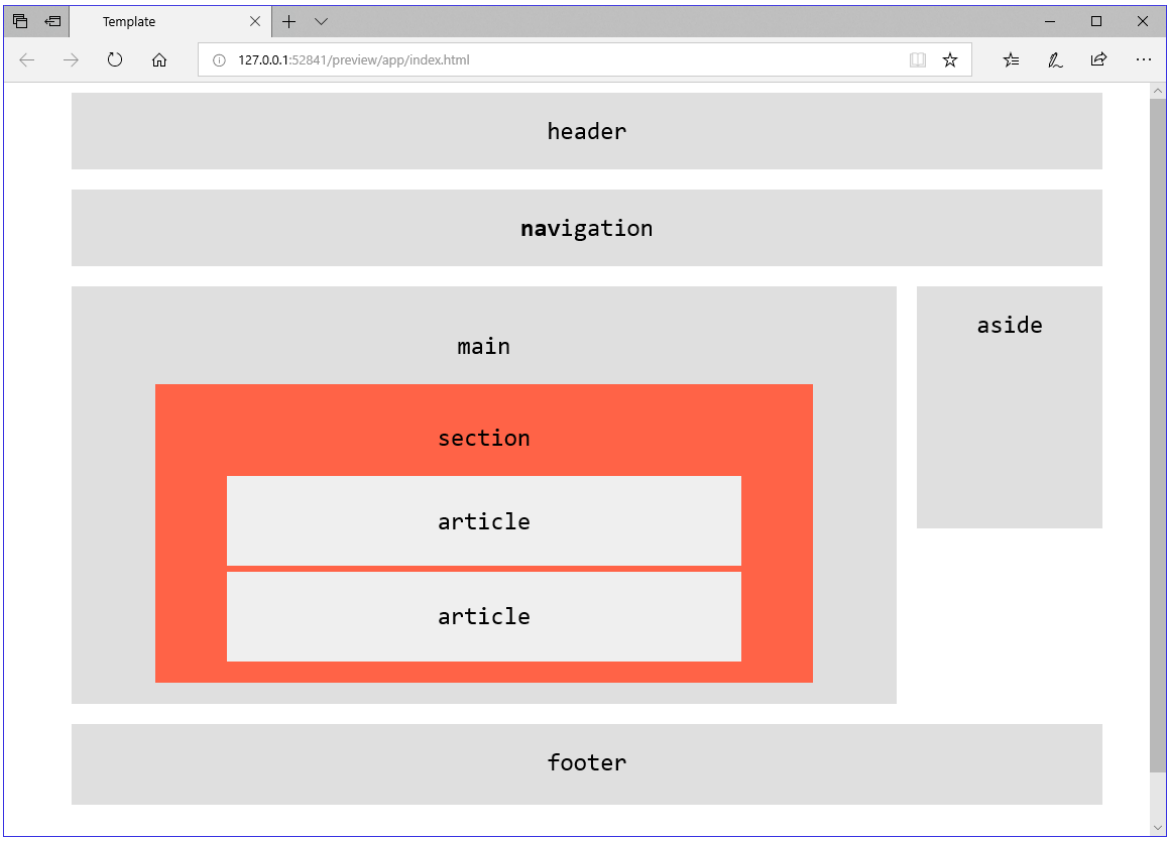


Рис. 4.4. Шаблон блочного сайта

Атрибуты тегов HTML5. В HTML5 существуют атрибуты, которые можно добавлять к любым тегам. Такие теги называются глобальными. Перечислим основные из них:

- `accesskey` – позволяет активировать ссылку при помощи сочетания клавиш;
- `contenteditable` – сообщает, что элемент доступен для редактирования пользователем;
- `class` – связывает стилевые свойства с тегом;
- `draggable` – позволяет перетаскивать элемент;
- `hidden` – скрывает содержимое элемента;
- `lang` – указывает на смену языка внутри тега;
- `style` – задает стили элемента при помощи css;
- `title` – создает всплывающую текстовую подсказку.

Позиционирование блоков в CSS. Позиционированием называется метод размещения блочных элементов HTML. За размещение отвечает свойство `position`, которое может принимать пять значений:

- `absolute` – абсолютное;
- `relative` – относительное;

- `fixed` – фиксированное;
- `static` – статичное;
- `inherit` – наследование.

По умолчанию в HTML ко всем элементам применяется статичное позиционирование: это означает, что все элементы размещаются последовательно в документе.

Для размещения элемента существует несколько свойств:

- `top` – задает отступ сверху;
- `right` – задает отступ справа;
- `left` – задает отступ слева;
- `bottom` – определяет положение относительно нижней границы родительского элемента.

Плавающие блоки (float). Плавающие блоки – ещё один способ размещения блочных элементов. Это позволяет задать расположение нескольких блоков в линии относительно друг друга. На рис. 4.4 к блоку `aside` применено свойство `float:left`. Может принимать следующие значения:

- `left` – блок смещается к левому краю родительского элемента, при этом внешнее содержимое начинает «обтекать» элемент справа;
- `right` – блок смещается к правому краю родительского элемента, при этом внешнее содержимое начинает «обтекать» элемент слева;
- `none` – значение блока по умолчанию, при котором смещения не происходит.

Для использования плавающих блоков есть ещё одно дополнительное свойство, которое называется `clear`. Оно указывает, что блочный элемент не может располагаться на одной строке с элементом, находящимся слева или справа от него. Возможны следующие значения свойства `clear`:

- `left` – блок размещается ниже всех левосторонних плавающих блоков;
- `right` – блок размещается ниже всех правосторонних плавающих блоков;
- `both` – блок размещается ниже всех плавающих блоков;
- `none` – значение по умолчанию, никаких смещений не происходит.

Для того чтобы расположить несколько блоков в линию, необходимо задать им свойство `display: inline-block`.

Код файла CSS, соответствующий позиционированию элементов на рис. 4.4:

```
body{
     width:90%;
     margin:0 auto;
}
header {
     width:100%;
}
nav {
     width:100%;
}
main {
     width: 76%;
     display: inline-block;
}
aside {
     width:18%;
     display: inline-block;
     margin-right: -20px;
     float: right;
}
footer {
     width:100%;
     float: left;
}
section {
     width: 80%;
     margin: 0 auto;
}
article {
     width: 80%;
     margin: 10px auto;
}
```

Задания

Задание 1. Создайте веб-страницу с шаблоном блочной вёрстки (см. рис. 4.4). У блока `aside` свойство `float: right;` ширина блока `aside` – 18 %, ширина блока `main` – 80 %; ширина `body` – 90 %. Горизонтальные блоки по ширине – 100 %. У блоков `main` и `aside` свойство `display: inline-block`.

Задание 2. В блоке `main` создайте два блока `section`. Во втором блоке `section` создайте пять блоков `article` с заголовками и текстом внутри. Каждый заголовок превратите в «якорь» (см. лабораторную работу № 3). В первом блоке `section` создайте оглавление для блоков `article`. Цвета ссылок поменяйте (рис. 4.5).

В качестве текста абзаца используйте отрывок: «*Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat*».

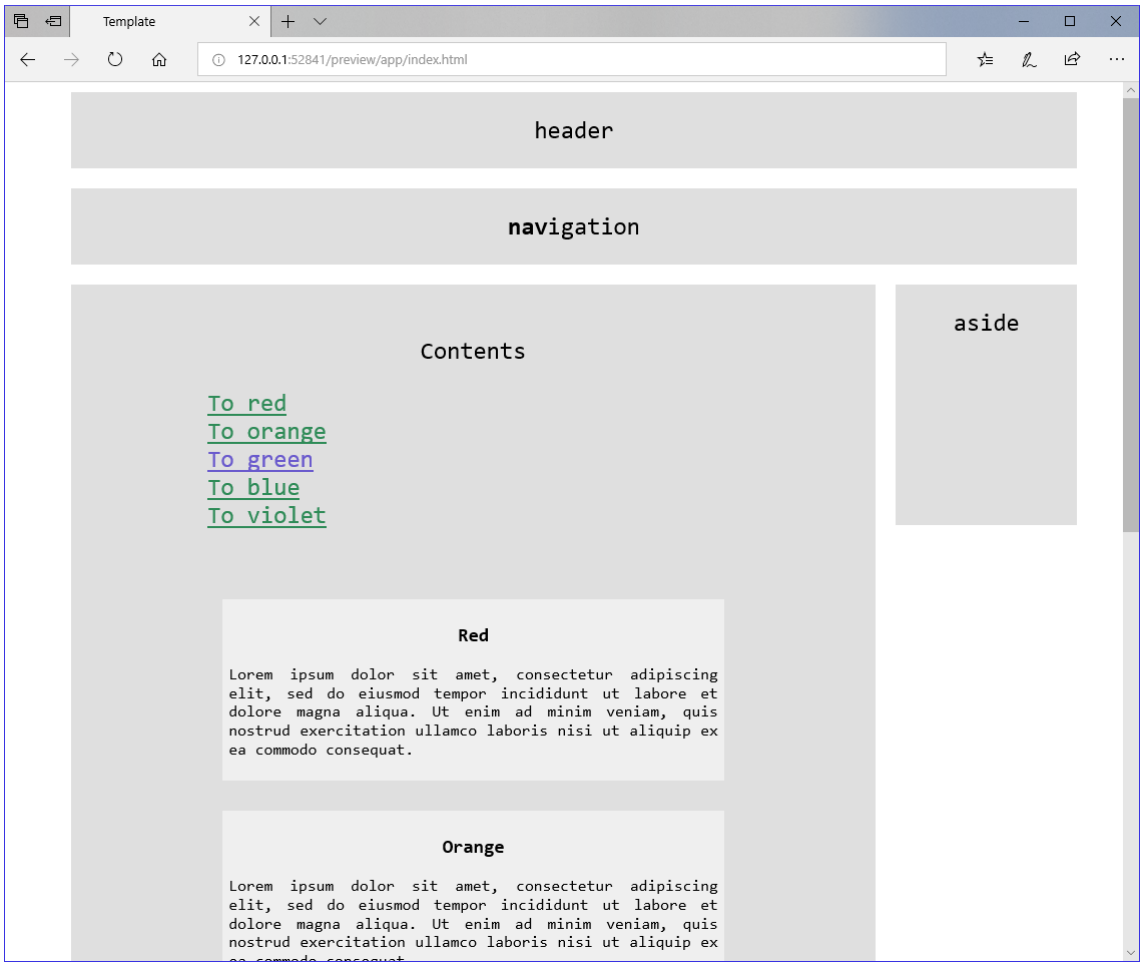


Рис. 4.5. Иллюстрация к заданию 2

Задание 3. Используя веб-страницу из задания 2, в блок `nav` поместите панель навигации. Она должна повторять панель на рис. 4.6, но цвета должны отличаться. Границы должны появляться при наведении со всех четырех сторон, а не только снизу. Изображение должно быть размещено справа, а не слева.

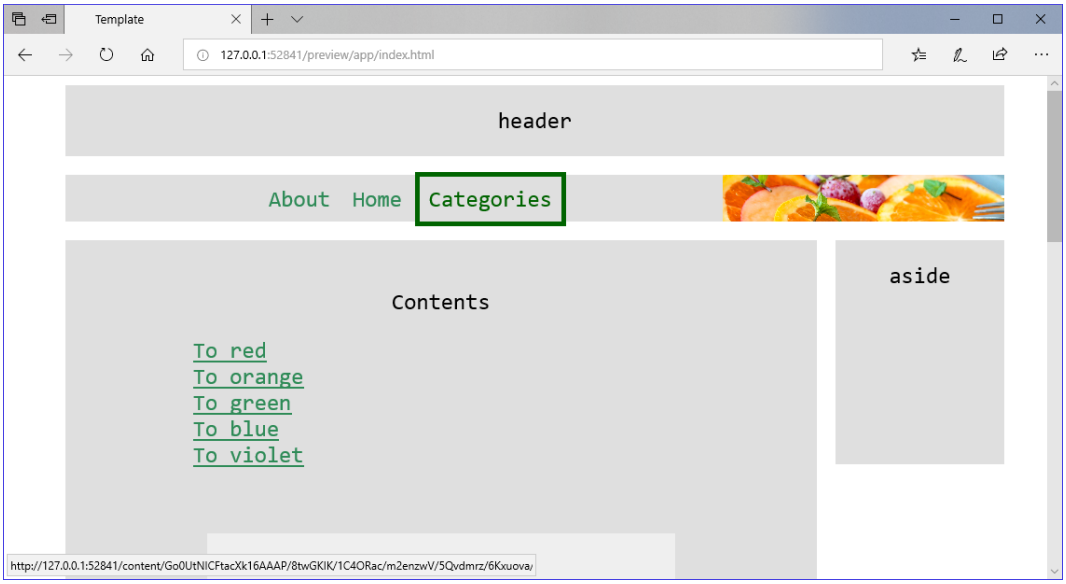


Рис. 4.6. Иллюстрация к заданию 3

Задание 4. Сделайте фиксированную сверху панель навигации. Пример показан на рис. 4.7. Для этого в свойствах CSS элемента `nav` пропишите: `position: fixed; top:0; left: 0;` а в свойствах CSS тега `body` – `margin-top: 50px;`

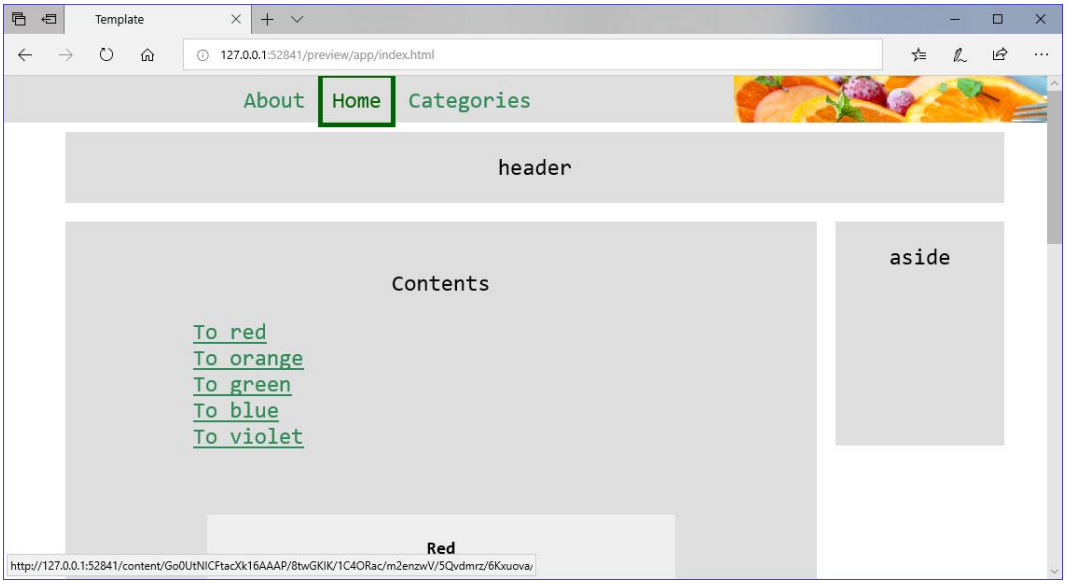


Рис. 4.7. Иллюстрация к заданию 4

Контрольные вопросы

1. Что такое семантические элементы в HTML5?
2. Чем можно заменить семантические элементы?
3. Для чего нужен элемент `nav`?
4. Сколько элементов `main` может быть на веб-странице?
5. Может ли на странице быть только один элемент `article`?
6. Что целесообразно помещать в элемент `aside`?
7. В какой элемент целесообразно поместить копирайт сайта?
8. Как разместить несколько элементов в одну строку?
9. Какие значения может принимать параметр `float`?
10. Как зафиксировать элемент внизу страницы?

Лабораторная работа № 5

CSS3: НОВЫЕ ВОЗМОЖНОСТИ

Цель работы

1. Научиться применять различные модели задания цвета.
2. Ознакомиться с новыми вариантами оформления.
3. Изучить различные единицы измерения в CSS для создания адаптивных веб-страниц.

Ход работы

CSS-цвета. Модуль CSS `color` описывает значения, которые позволяют определять цвета и непрозрачность HTML-элементов.

Свойство `color`. Определяет цвет текста для элемента. В качестве значения указывается цвет в поддерживаемом CSS-формате. Значение `inherit` определяет наследование цвета родительского элемента.

Название цвета. Существует список ключевых слов, которые указывают название цвета (см. таблицу).

Основные цвета

Название	HEX	RGB
black	#000000	0,0,0
silver	#C0C0C0	192,192,192
gray	#808080	128,128,128
white	#FFFFFF	255,255,255
maroon	#800000	128,0,0
red	#FF0000	255,0,0
purple	#800080	128,0,128

Название	HEX	RGB
fuchsia	#FF00FF	255,0,255
green	#008000	0,128,0
lime	#00FF00	0,255,0
olive	#808000	128,128,0
yellow	#FFFF00	255,255,0
navy	#000080	0,0,128
blue	#0000FF	0,0,255
teal	#008080	0,128,128
aqua	#00FFFF	0,255,255

Пример:

```
    color: teal;
```

Кроме основных есть ещё множество других цветов.

Шестнадцатеричное представление цвета (HEX). Представление в виде трёх пар шестнадцатеричных цифр `#123ABC`, где каждая пара отвечает за свой цвет:

- две первые цифры – красный;
- две в середине – зелёный;
- две последние цифры – синий.

Возможно также краткое представление цвета в виде `#ABC`, что будет интерпретировано как `#AABBCC`.

Пример:

```
    color: #00FF00;
```

Цвета модели RGB. Формат значения RGB в функциональной нотации – `rgb(`, за которым следует разделенный запятыми список из трех числовых значений (либо трех целочисленных значений, либо

трех процентных значений), за которыми следует символ). Целочисленное значение 255 соответствует 100 % и F или FF в шестнадцатеричной записи:

```
rgb (255,255,255) = rgb (100%, 100%, 100%) = #FFF
```

Символы пробела допускаются вокруг числовых значений.

Пример:

```
color: rgb(255,0,0);
color: rgb(100%, 0%, 0%);
```

Цвета модели HSL. Цвета HSL интуитивно понятны (в отличие от RGB). Они кодируются как тройка (оттенок, насыщенность, яркость). Оттенок представлен как угол цветного круга (т. е. радуга, представленная в круге). По определению красный = 0 = 360, а остальные цвета распределены по кругу, поэтому зеленый = 120, синий = 240 и т. д. Насыщенность и яркость представлены в процентах: 100 % – это полное насыщение, а 0 % – это оттенок серого. Яркость 0 % – черная, 100 % – белая, 50 % – нормальная.

```
color: hsl(0, 100%, 50%);
color: hsl(120, 100%, 50%);
```

Непрозрачность. Существует два способа задать непрозрачность элемента: задать непрозрачность фона или непрозрачность элемента в целом.

В первом случае работают со свойством `background`. В качестве значения пишется `rgba`, затем в скобках через запятую перечисляются три числа: значения красного, синего и зеленого компонентов цвета.

После перечисления значений компонентов цвета идёт четвёртое число – значение альфа-канала, т. е. непрозрачности. Это дробное число от 0 до 1.

Пример (полупрозрачный оранжевый):

```
background: rgba(255, 150, 100, 0.8);
```

С помощью цвета в формате `rgba` можно сделать полупрозрачным и текст, и любой элемент с цветом в целом.

Аналогично для модели `hsl` есть форма `hsla`, где альфа-канал имеет значение 4.

Пример:

```
color: hsla(240, 100%, 50%, 0.5);
```

Можно использовать ключевое слово `transparent`. Оно является сокращением для прозрачного черного цвета `rgba (0,0,0,0)` и его вычисленным значением.

Пример:

```
color: transparent;
```

Свойство `opacity` пишется в CSS-свойствах элемента. Его значение – дробное число от 0 до 1.

Пример:

```
opacity: 0.75;
```

Скругленные края у блоков. Для того чтобы создать скругленные края рамки у блоков, в CSS3 применяется обобщенное свойство `border-radius`. В его значении пишутся от одного до четырех значений в пикселях (или других единицах измерения), каждое обозначает величину скругления на одном из четырех углов блока. Углы считают по часовой стрелке, начиная с левого верхнего угла.

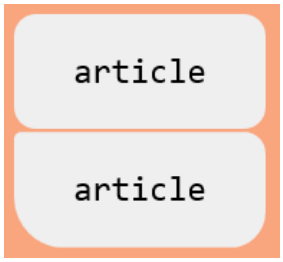


Рис. 5.1. Пример закругления углов

Можно прописать одно значение, и оно будет применено ко всем углам сразу (рис. 5.1).

Пример:

```
<article style="border-radius: 15px;">
  <p>article</p>
</article>
<article style="border-radius: 5px 15px 25px 35px;">
  <p>article</p>
</article>
```

Тень. Тени у блочного элемента создаются свойством `box-shadow`, который обладает шестью значениями.

Основные параметры (указаны в порядке написания):

- ключевое слово `inset` устанавливает тень внутри элемента;
- сдвиг тени по горизонтали;
- сдвиг тени по вертикали;
- радиус размытия тени;
- растяжение тени;
- цвет тени.

Аналогично задаётся тень текста, но с иным названием свойства – `text-shadow`. У тени текста нет параметра `inset` и растяжения тени, т. е. нужно указывать только три числа и цвет.

Пример (рис. 5.2):

```
  <article style="box-shadow: 5px 5px 0px 15px white">
       <p>article</p>
   </article>
   <article style="box-shadow: inset 1px 5px 15px 5px
#000000">
       <p style="text-shadow: 3px 3px 5px rgba(255,
150, 100, 0.8);">article</p>
   </article>
```

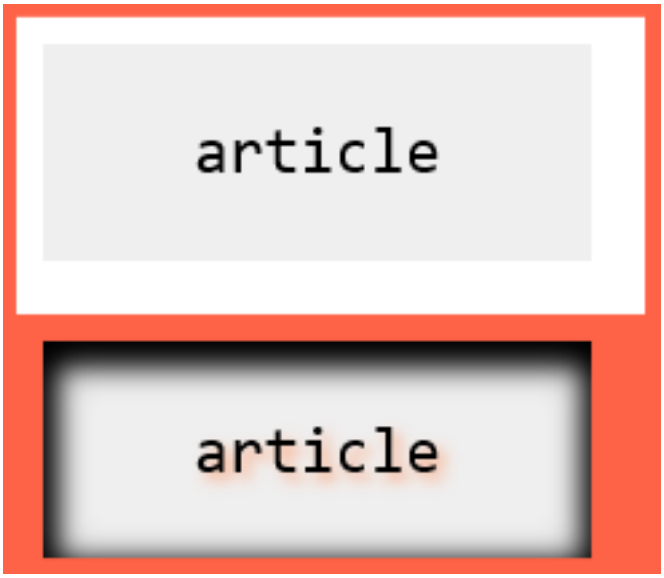


Рис. 5.2. Пример теней

Единицы измерения

Пиксель (px). Базовая, абсолютная и окончательная единица измерения. Количество пикселей задаётся в настройках разрешения экрана, один `px` – одна единица разрешения. Все значения браузер в итоге пересчитает в пиксели.

Пиксели могут быть дробными, например размер можно задать в `16.5px`. При окончательном отображении дробные пиксели округляются и становятся целыми.

Существуют также «производные» от пикселя единицы измерения: `mm`, `cm`, `pt` и `pc`, но они устарели, так как не соотносятся с реальными единицами, а в браузере всё равно пересчитываются в пиксели:

- 1 mm (мм) = 3.8px;
- 1 cm (см) = 38px;
- 1 pt (типографский пункт) = 4/3 px;
- 1 pc (типографская пика) = 16px.

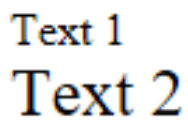


Рис. 5.3. Относительный размер шрифта

Относительно шрифта (em). 1 em – текущий размер шрифта. Можно брать любые пропорции от текущего шрифта: `2em`, `0.5em` и т. п.

Пример (рис. 5.3):

```
<div style="font-size: 16px;">Text 1
        <div style="font-size: 1.5em">Text 2</div>
    </div>
```

Проценты. Относительная единица, которая высчитывается от какого-либо свойства родителя. Например, если в процентах задаётся размер шрифта, то процент будет отсчитываться от размера шрифта родителя; если же отступ, то от ширины или высоты элемента. Проценты высчитываются индивидуально и использовать их нужно с осторожностью.

Пример (аналогично рис. 5.3):

```
<div style="font-size: 16px;">Text 1
        <div style="font-size: 150%">Text 2</div>
    </div>
```

Проценты от размера окна. Эти новые единицы были созданы специально для вёрстки адаптивных веб-страниц – страниц, которые выглядят должным образом на разных устройствах и при разном разрешении экрана:

- `vw` – 1 % ширины окна;
- `vh` – 1 % высоты окна;
- `vmin` – наименьшее из `vw`, `vh`;
- `vmax` – наибольшее из `vw`, `vh`.

Эти значения были созданы, в первую очередь, для поддержки мобильных устройств. Их основное преимущество в том, что любые размеры, которые в них заданы, автоматически масштабируются при изменении размеров окна.

Задания

Задание 1. Создайте веб-страницу с шестью заголовками разных уровней от 1 до 6. Цвета каждого задайте следующим образом:

1) с помощью названия цвета;
2) шестнадцатеричным кодом;
3) моделью `RGB`;
4) моделью `HSL`;
5) моделью `RGBA`;
6) моделью `HSLA`.

Пример показан на рис. 5.4.

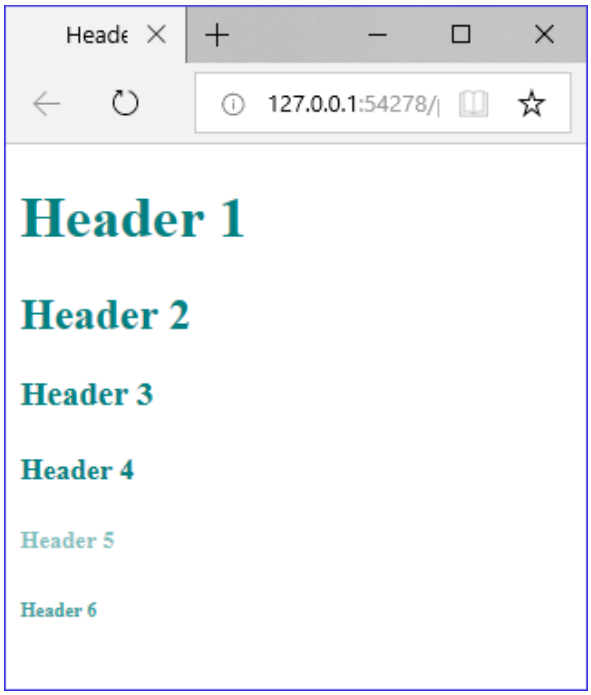


Рис. 5.4. Иллюстрация к заданию 1

Задание 2. Создайте блок с полупрозрачным фоном (`rgba` или `hsla`) с закруглением (различного радиуса для каждого угла), с внутренней (`inset`) тенью, прозрачным текстом и тенью от него (рис. 5.5).

Рис. 5.5. Иллюстрация к заданию 2

Задание 3. Для выполнения задания используйте веб-страницу из лабораторной работы № 4 (задание 3 или 4). Задайте размер текста в `body` равным `3vh`. Размер текста в панели навигации задайте равным `1.3em`. Протестируйте получившуюся веб-страницу с различной величиной окна. Пример показан на рис. 5.6.

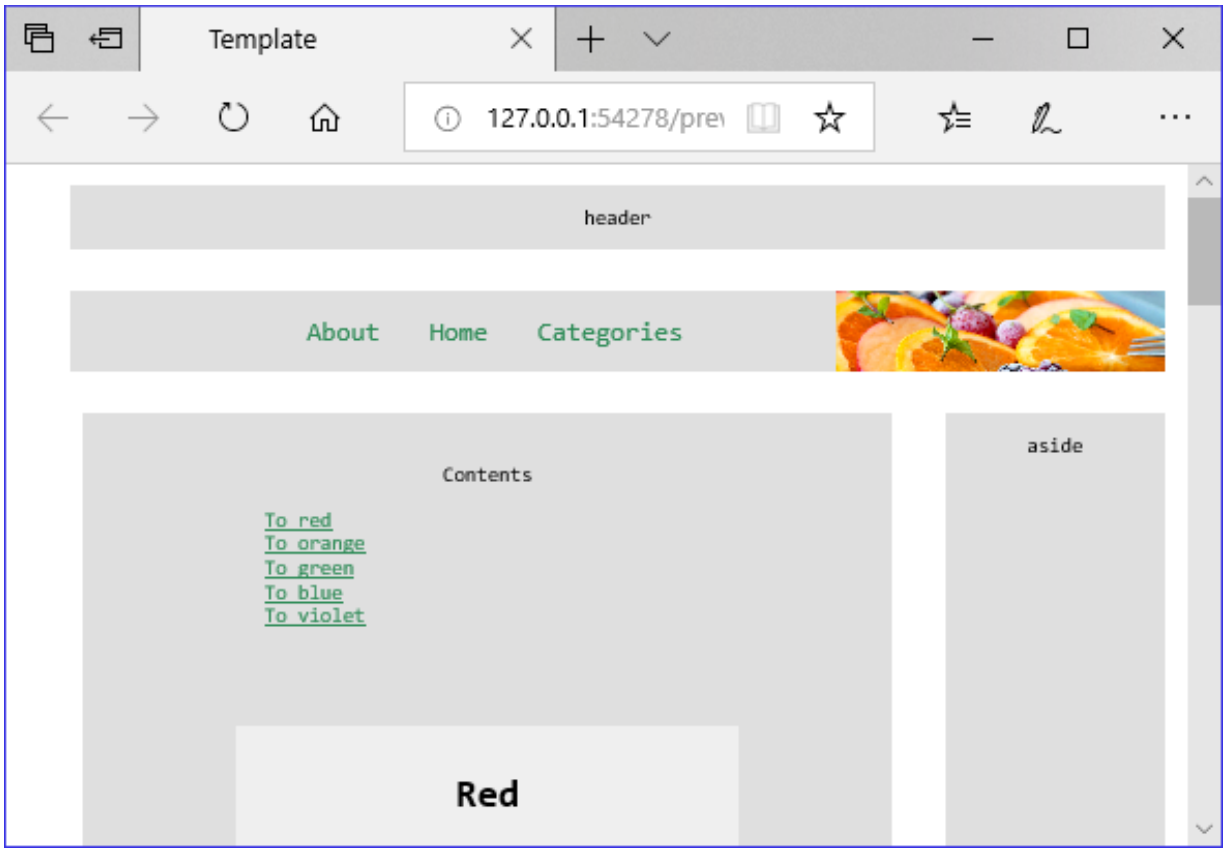


Рис. 5.6. Иллюстрация к заданию 3

Задание 4. В веб-странице из предыдущего задания измените размер текста в `body` на `3vmin`. Протестируйте веб-страницу. Повторите тестирование со значением `3vmax`.

Контрольные вопросы

1. Как можно задать цвет в CSS3?
2. Какой цвет подразумевают следующие коды: `#00FF00`, `#008080`, `#fff`?
3. Какие способы сделать элемент полупрозрачным существуют?
4. Как установить прозрачный цвет текста?
5. Как задать тень блоку?
6. Что отсутствует у тени текста по сравнению с тенью блока?
7. Как у блока одинаково закруглить все четыре угла?
8. Какая самая однозначная единица измерения для элементов?
9. От чего зависит размер текста при использовании единицы `em`?
10. Процент от чего считает единица измерения `vmax`?

Лабораторная работа № 6

ФОРМЫ HTML

Цель работы

Научиться создавать HTML-формы.

Ход работы

Форма HTML предоставляет пользователю область для ввода данных. В дальнейшем веб-страница может эти данные обработать. Таким образом, форма – посредник между пользователем и логикой сайта.

Элементами формы могут быть кнопки, поля для ввода, список элементов с возможностью выбора и т. д. Существует множество типов элементов, которые рассчитаны на ввод особого содержимого (например, паролей, дат, веб-адресов).

Элемент `<form></form>` – основной контейнер, внутри которого содержатся элементы формы. У этого элемента существует множество атрибутов (табл. 6.1).

Таблица 6.1. *Атрибуты элемента `form`*

Атрибут	Значение/описание
`accept-charset`	Список кодировок символов через пробел
`action`	Обязательный атрибут, который указывает url обработчика формы на сервере, которому передаются данные. Представляет из себя файл (например, action.php) с программой обработки. Для сценариев JavaScript можно указать значение #
`autocomplete`	Автоподстановка данных при вводе: on – включена, off – отключена
`method`	Задает способ передачи данных формы на сервер

Окончание табл. 6.1

Атрибут	Значение/описание
name	Задает имя формы, которое будет использоваться для доступа к элементам формы
novalidate	Отключает проверку по кнопке для отправки формы. Атрибут используется без указания значения
target	Указывает окно, в которое будет направлена информация

Подписи к полям. За подпись к полю отвечает тег <label>...</label>, внутри которого пишется название. У тега есть атрибут for. В качестве его значения нужно указать название идентификатора поля, к которому название относится.

Элемент <fieldset></fieldset>. Создает логическую группировку элементов внутри формы. Может иметь название в теге <legend> </legend>, который располагается сразу после открывающего тега <fieldset>. Такую группировку элементов можно отключить, сделать недоступной для ввода данных. Для этого в теге fieldset необходимо прописать атрибут disabled без значения.

У группы fieldset может быть название. Оно прописывается как значение атрибута name. Пример формы с группировкой представлен на рис. 6.1.

Пример:

```
<form>
  <fieldset>
      <legend>Contact us</legend>
      <p><label for="name">Name</label>
          <input type="text" id="name"></p>
      <p><label for="email">E-mail</label>
          <input type="email" id="email"></p>
  </fieldset>
  <p><input type="submit" value="Submit"></p>
</form>
```

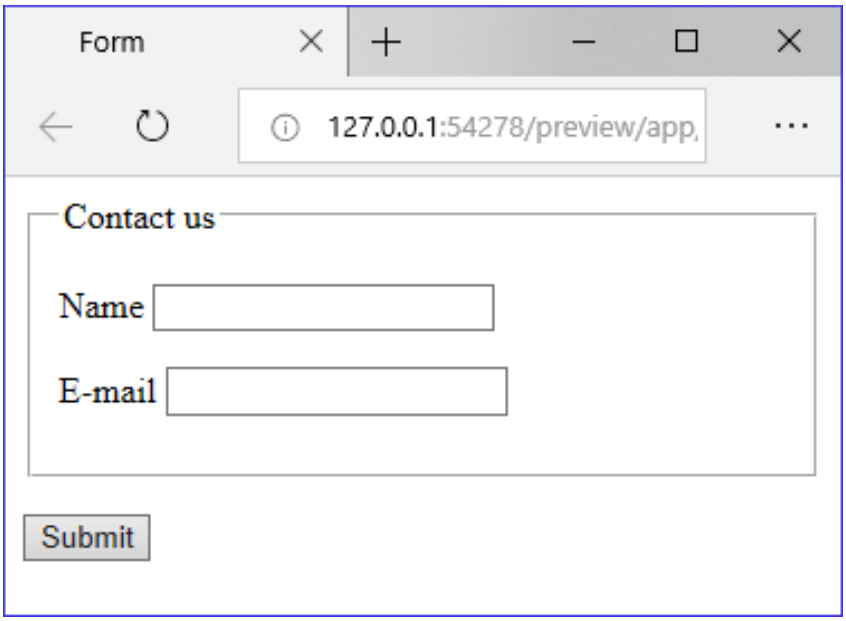


Рис. 6.1. Пример формы с группировкой

Поля формы. Для создания полей формы используются различные элементы.

`<input>`. Непарный тег, основной функционал которого определяется его атрибутами. Основной его атрибут – `type` – задаёт тип вводимого содержимого и его представление соответственно содержимому. Доступные типы перечислены в табл. 6.2.

Таблица 6.2. *Типы полей для тега* `input`

Атрибут	Значение/описание
`button`	Кнопка
`checkbox`	Пометка в виде флажка
`color`	Выбор цвета
`date`	Дата в формате дд.мм.гггг.
`datetime-local`	Дата и время по шаблону дд.мм.гггг чч:мм.
`email`	Электронная почта (включает в себя допустимые в почте символы, значок @ и точку в определённых местах)
`file`	Файлы с компьютера пользователя

Окончание табл. 6.2

Атрибут	Значение/описание
`image`	Создает кнопку из изображения. Работает с атрибутом src=”ссылка на изображение” и alt=”замещающий при ошибке текст”
`month`	Год и номер месяца по шаблону гггг-мм
`number`	Целочисленное значение. Атрибуты `min`, `max` и `step` задают верхний, нижний пределы и шаг между значениями соответственно
`password`	Ввод пароля. Введённые символы заменяются при отображении
`radio`	Переключатель (выбор одного значения из нескольких)
`range`	Выбор целого значения из диапазона (ползунок, min/max) позволят установить числовое значение диапазона выбора
`reset`	Кнопка, очищающая введённые данные
`search`	Поле поиска
`submit`	Стандартная кнопка отправки введённых данных на сервер
`text`	Текстовое поле
`time`	Время в 24-часовом формате по шаблону чч:мм
`url`	Веб-адрес
`week`	Неделя года в формате нн-гггг. В зависимости от года число недель может быть 52 или 53

Кроме типа элемента применяются другие вспомогательные атрибуты, которые перечислены в табл. 6.3.

Таблица 6.3. *Атрибуты* `<input>` *(кроме* `type`*)*

Атрибут	Значение/описание
`autocomplete`	Автоподстановка данных при вводе: `on` – включена, `off` – отключена
`autofocus`	Автоматический выбор поля для ввода
`checked`	Проверка на наличие флажка для полей типа `type="checkbox"` и `type="radio"`
`disabled`	Отключает взаимодействие с полем
`formaction`	Задает URL файла, который будет обрабатывать введенные в поля данные при отправке формы
`formmethod`	Атрибут определяет метод, который браузер будет использовать для отправки данных формы на сервер. Задается только для полей типа `type="submit"` и `type="image"`
`formtarget`	Определяет, где выводить ответ, полученный после отправки формы
`height`	Высота в пикселях
`max`	Максимальное вводимое значение. Работает со следующими типами полей: `number`, `range`, `date`, `datetime`, `datetime-local`, `month`, `time` и `week`
`maxlength`	Атрибут задает максимальное количество символов, вводимых в поле. Значение по умолчанию – 524288 символов
`min`	Минимальное вводимое значение
`multiple`	Позволяет пользователю ввести несколько значений атрибутов, разделяя их запятой. Применяется в отношении файлов и адресов электронной почты. Указывается без значения атрибута

Окончание табл. 6.3

Атрибут	Значение/описание
`name`	Определяет имя для доступа к элементу
`pattern`	Позволяет определять с помощью регулярного выражения синтаксис данных, ввод которых должен быть разрешен в определенном поле
`placeholder`	Подсказка в поле
`readonly`	Не позволяет пользователю изменять значения элементов формы; выделение и копирование текста при этом доступно. Указывается без значения атрибута
`required`	Делает поле обязательным для заполнения
`size`	Ширина поля в символах. Значение по умолчанию – 20. Работает со следующими типами полей: `text, search, tel, url, email и password`
`src`	Задает URL изображения, используемого в качестве кнопки отправки данных формы. Указывается только для поля `<input type="image">`
`step`	Шаг для увеличения/уменьшения числового значения
`value`	Название кнопки или текст, уже введённый в поле
`width`	Ширина элемента в пикселях

<textarea>…</textarea>. Аналог `<input type=”text”>`, приспособленный для отображения длинного текста. Имеет те же атрибуты, что доступны для `<input type=”text”>`.

Особые атрибуты: `cols` – размер по горизонтали в символах, `rows` – размер по вертикали в символах; `width` – ширина в пикселях, `height` – высота в пикселях. `Wrap` определяет, должен ли текст сохранять переносы строк при отправке формы. Значение `hard` сохраняет перенос, а значение `soft` не сохраняет. Если используется значение `hard`, то должно указываться значение атрибута `cols`.

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

Флажки допускают множественный выбор. Переключатель может быть выбран только один из группы.

Для того чтобы отметить флажок или переключатель кликом по тексту, необходимо поместить `<input …>` внутрь тега `<label> </label>`.

Для выбора значения из группы и контроля числа выбранных пунктов (например, выбор не более одного элемента переключателем) последние обязательно следует связывать между собой. Пункты являются связанными, если для них указан одинаковый идентификатор `name`.

Пример (рис. 6.2):

```
<form>
  <fieldset style="display:inline-block">
    <legend>Single</legend>
    <p><label><input type="radio" name="counter">
First</label></p>
    <p><label><input type="radio" name="counter">
Second</label></p>
    <p><label><input type="radio" name="counter">
Third</label></p></fieldset>
    <fieldset style="display:inline-block">
    <legend>Multiple</legend>
    <p><label><input type="checkbox" name="check">
First</label></p>
    <p><label><input type="checkbox" name="check">
Second</label></p>
    <p><label><input type="checkbox" name="check">
Third</label></p></fieldset>
</form>
```

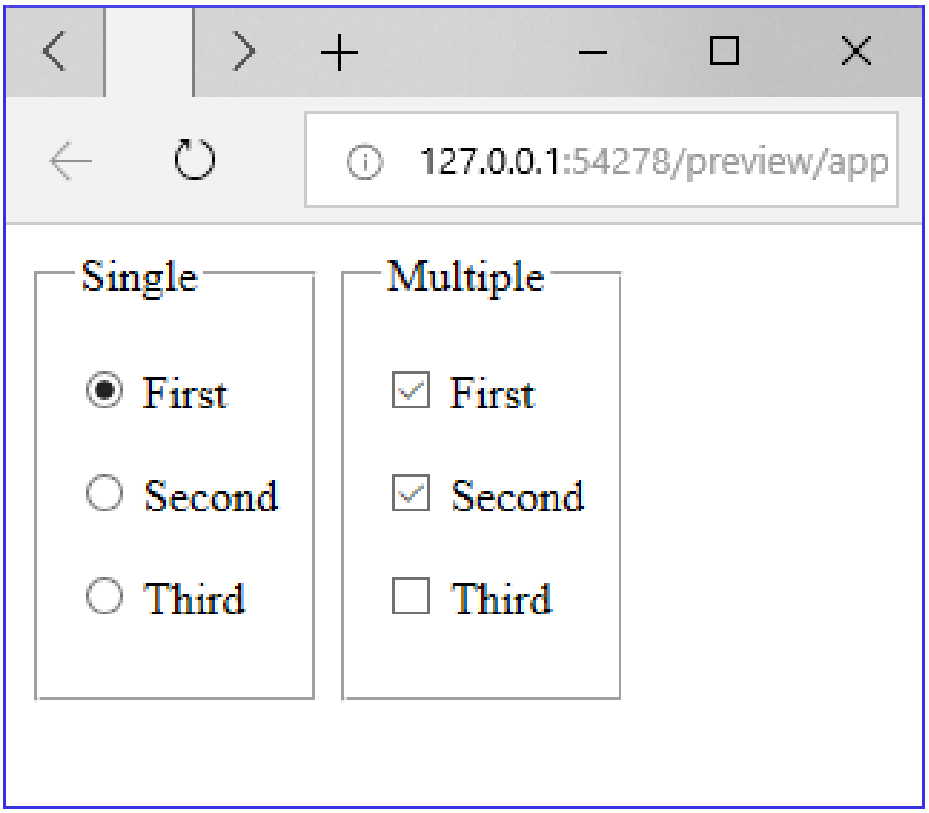


Рис. 6.2. Переключатели (слева) и флажки (справа)

Выпадающий список <select></select>. Выбор одного или нескольких значений из предложенного множества. Пункт добавляется тегом `<option></option>` внутри `select`. Возможна группировка пунктов с помощью `<optgroup></optgroup>`.

Для тега `select` доступны следующие атрибуты: `autofocus`, `disabled`, `multiple` (выбор нескольких пунктов), `name`, `required`, `size` (сколько пунктов выводится на экран одновременно).

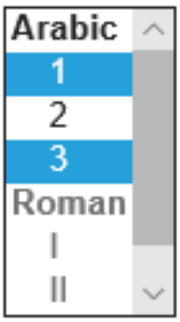


Рис. 6.3. Выпадающий список

Атрибуты тега `option`: `disabled`, `label` (значение – название пункта), `selected` (пункт становится выбранным по умолчанию).

Атрибуты тега `optgroup`: `disabled`, `label` (значение – название группы пунктов).

Пример (рис. 6.3):

```
<form>
      <select size="5" multiple required>
```

```
            <optgroup label="Arabic">
                <option>1</option>
                <option>2</option>
                <option>3</option>
            </optgroup>
            <optgroup label="Roman" disabled>
                <option>I</option>
                <option>II</option>
                <option>III</option>
            </optgroup>
        </select>
    </form>
```

Кнопки <button>...</button>. Этот элемент создает кликабельные кнопки. В отличие от `<input type="submit">` (или `<input type="image">`, `<input type="reset">`, `<input type="button">`), внутрь элемента `<button>` можно поместить наполнение – текст или изображение.

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

Кнопке можно задать отдельное оформление. Ее основная задача – выполнять какое-либо действие по клику.

Для кнопки доступны следующие атрибуты: `autofocus`, `disabled`, `formaction`, `formenctype`, `formmethod`, `formnovalidate`, `formtarget`, `name`, `type`, `value`.

Атрибут `value` задаёт текст, который написан на кнопке.

Задания

Задание 1. Создайте форму авторизации в группе `fieldset` с полями “`Name`” и “`Password`”, флажком “`I agree`” и кнопкой “`Sign in`” (`type=“submit”`). Текстовые поля и флажок обязательны

(`required`). Текстовые поля должны иметь подсказку (`placeholder`). Всем элементам необходимо задать атрибут `name`.

Названия полей `Name` и `Password` (тег `label`) должны располагаться в одном блоке `div`, поля `input` – в другом. Указанные блоки должны иметь `display: inline-block;` (чтобы быть двумя столбцами) (рис. 6.4).

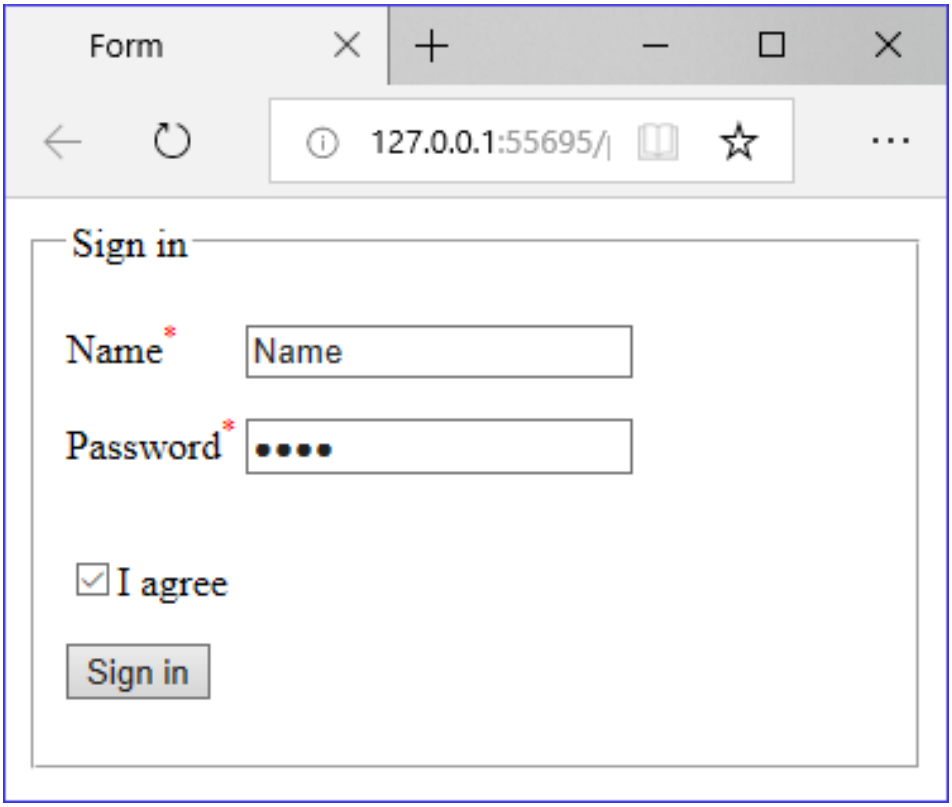


Рис. 6.4. Иллюстрация к заданию 1

Задание 2. Создайте группу `fieldset` с полем выбора цвета (`type="color"`), а затем блок с переключателями и блок с флажками (рис. 6.5). Указанные блоки должны быть блочно-строчными (для этого, как в предыдущем задании, поместите их в отдельные блоки `div` и задайте им `display: inline-block;`). Для того чтобы блоки были на одной высоте, задайте им `vertical-align: top;`

Всем элементам обязательно задайте атрибут `name`.

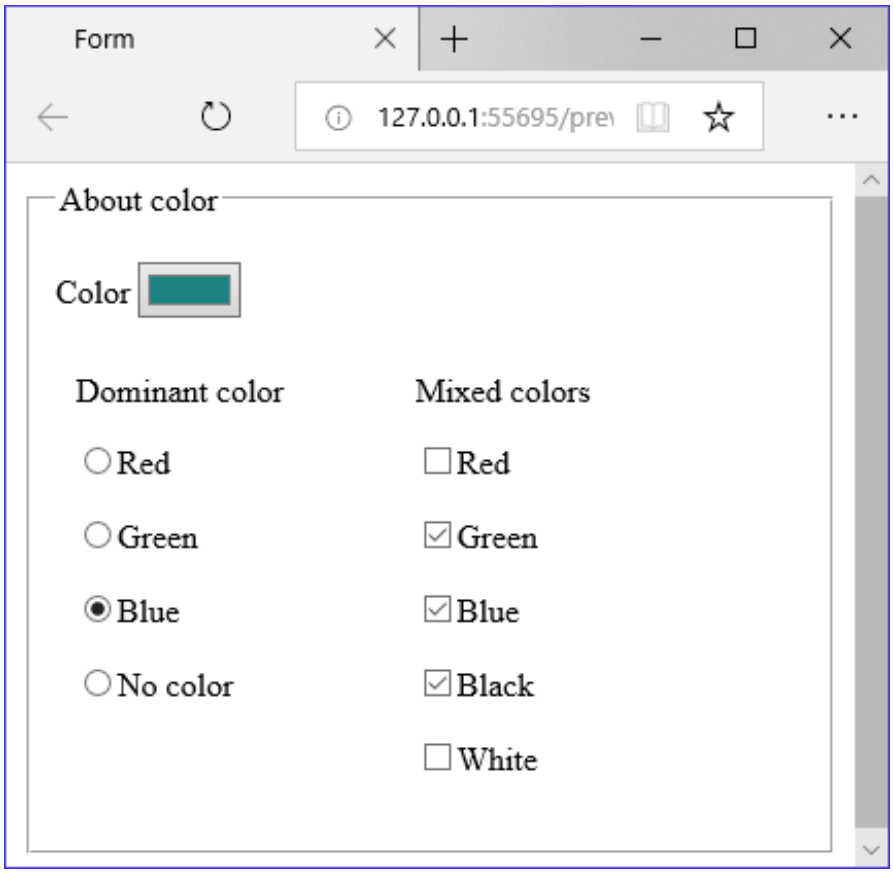


Рис. 6.5. Иллюстрация к заданию 2

Задание 3. Создайте форму профиля человека. В этой форме должны быть следующие поля:

1) имя;
2) фамилия;
3) дата рождения;
4) флажок «мне больше 18»;
5) номер телефона;
6) электронная почта;
7) ссылка на аккаунт в социальной сети (`url`);
8) фото (файл с компьютера);
9) информация о себе (`textarea`);
10) кнопки `Submit` и `Reset`.

Поля 1 – 4, 5 – 7, 8 – 9 должны быть сгруппированы в `fieldset`. Поля 5 – 7 должны иметь подсказку. Поля 1 – 4 обязательные.

Элементам `label` задайте одинаковую ширину с помощью CSS и `display: inline-block;`. Текстовым полям задайте ширину в символах (атрибут `size`). Пример показан на рис. 6.6.

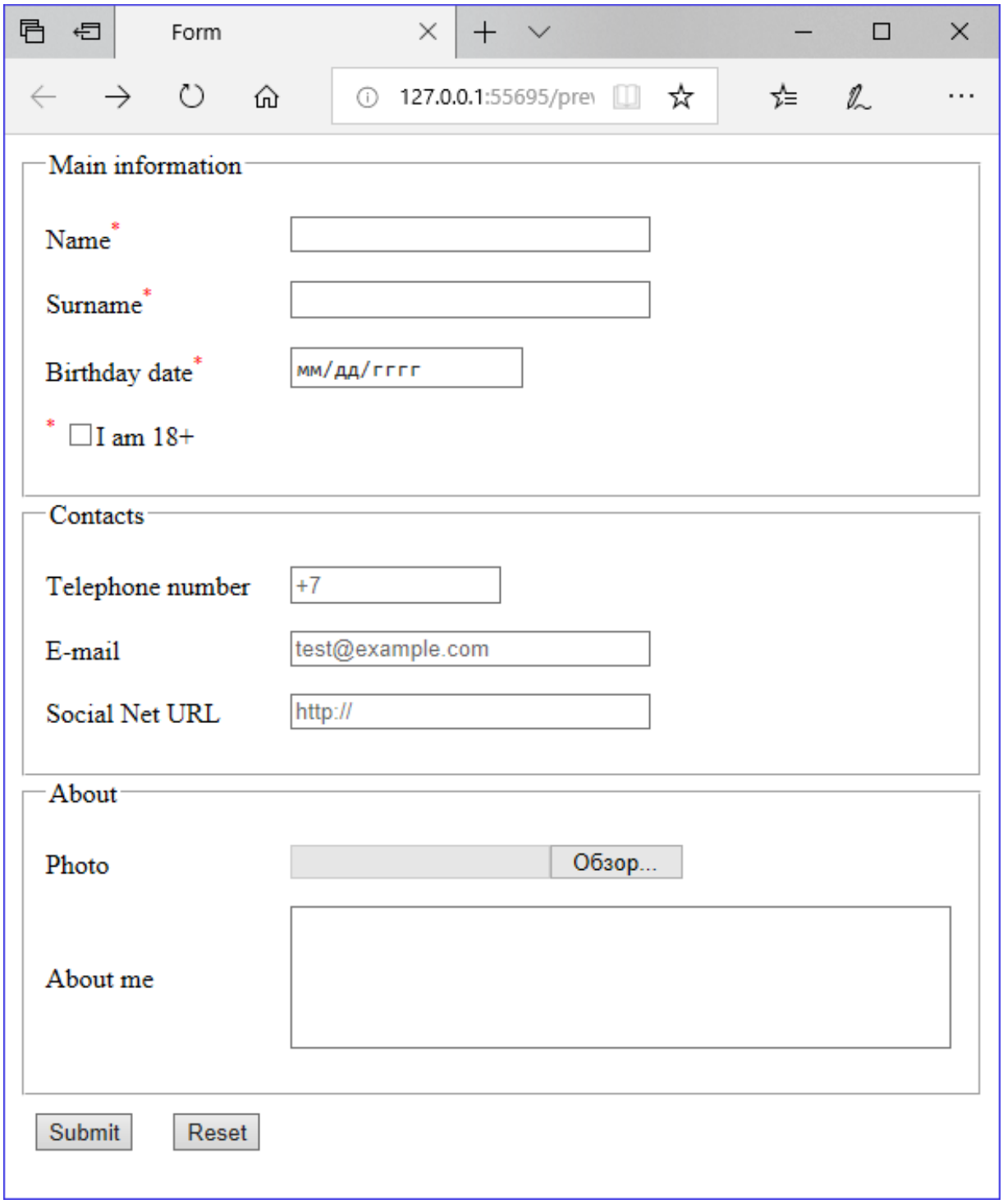


Рис. 6.6. Иллюстрация к заданию 3

Задание 4. Создайте два выпадающих списка: со свойством `size` (где значение больше 1) и без него. В обоих списках должно быть не менее семи пунктов. Часть пунктов отключите (сделайте с атрибутом `disabled`). Пример показан на рис. 6.7.

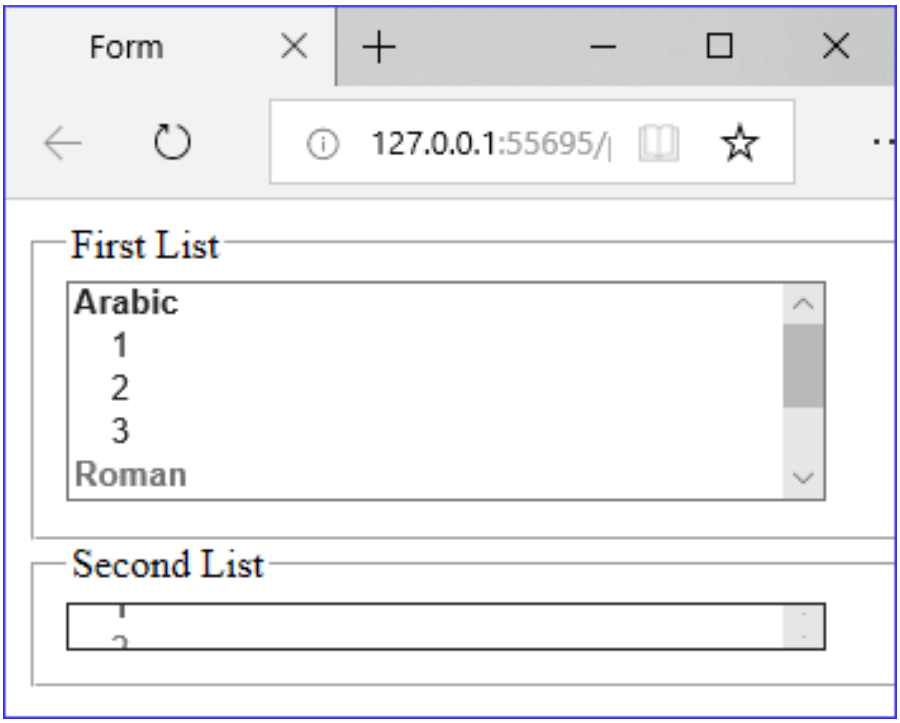


Рис. 6.7. Иллюстрация к заданию 4

Контрольные вопросы

1. Зачем нужны HTML-формы?
2. В какой блок необходимо помещать все элементы формы?
3. Как задаётся название группе элементов `fieldset`?
4. Какой основной тег для создания полей формы?
5. Как создать поле для ввода адреса веб-страницы?
6. Какая основная функция у кнопки?
7. Как сделать так, чтобы в группе можно было отметить только один переключатель?
8. Какие теги используются в создании выпадающего списка?
9. Какой универсальный атрибут определяет отключение взаимодействия с элементом?
10. Какой атрибут привязывает `label` к элементу?

Лабораторная работа № 7

ОСНОВЫ JAVASCRIPT

Цель работы

Усвоить основы языка JavaScript.

Ход работы

Язык JavaScript – это состоящий из скриптов язык программирования, интерпретируемый браузером во время загрузки веб-страницы. Программы, написанные на этом языке, могут встраиваться в веб-страницу и взаимодействовать с HTML-элементами и CSS.

Подключение программы на JavaScript к HTML. Программа может содержаться в документе HTML (*встроенный скрипт*), а может быть написана отдельно в файле с расширением .js (*внешний скрипт*).

Для подключения программы на JavaScript к HTML используется тег `<script>…</script>`. Написать js-код можно внутри этого тега или же в отдельном файле и указать ссылку на этот файл в атрибуте `src` тега `script`.

У тега `script` есть атрибут `type`, который позволяет быстрее опознать язык скрипта. Для `js` этот атрибут выглядит так: `type="text/javascript"`.

Тег `script` может быть помещён внутрь тега `<head></head>` веб-страницы – тогда он будет интерпретирован до ее загрузки. Это необходимо, когда на веб-странице вызывается функция из скрипта.

В другом случае тег может быть помещён внутрь тега `<body></body>`, чаще перед закрытием тега. Так скриптом удобно обрабатывать уже существующие на веб-странице данные. Скрипт интерпретируется после ее загрузки.

Внутри тега `<script></script>`. Тег может быть помещён как внутрь тега `<head></head>`, так и внутрь тега `<body></body>`.

Пример:

```
<script type="text/javascript">
      document.write('Script loaded!');
</script>
```

Ссылка на js-файл. Если код js-программы длинный или используется на нескольких веб-страницах, то можно вынести его в отдельный файл и указать ссылку на него в веб-странице.

Ссылка на файл пишется в атрибуте `src` тега `script`.

Пример:

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

Обработчик событий в скрипте. Событие – это сигнал от браузера о том, что что-то произошло. Событие `onclick` возникает при щелчке левой кнопкой мыши по элементу, к которому добавлен атрибут `onclick`. В качестве значения этого атрибута пишется название js-функции. Таким образом, например, можно задать выполнение кода при нажатии на кнопку.

Пример:

```
<script type="text/javascript">
     function test() {
         alert('Script loaded!');
     }
 </script>
 <button onclick="test();">Click me!</button>
```

Всплывающие окна. Всплывающие окна имеют простую логику и полезны для тестирования скрипта. Язык JavaScript зачастую капризен. Зафиксировано множество ситуаций, когда неизвестно, возникла ошибка в логике скрипта или в синтаксисе. На помощь приходят всплывающие окна. Они позволяют быстро проверить, работает ли конкретный фрагмент скрипта. Если всплывающее окно не сработало, то ошибка в синтаксисе, если сработало – то в логике скрипта вокруг всплывающего окна.

Сообщение alert(). Для вызова этого окна внутри скрипта пишут `alert('Message');`, где message – текст сообщения. Появится всплывающее окно с кнопкой подтверждения и текстом сообщения. В качестве текста, если убрать апострофы, можно задать и переменную (рис. 7.1).

Пример:

```
<script type="text/javascript">
          alert('Script loaded!');
  </script>
```

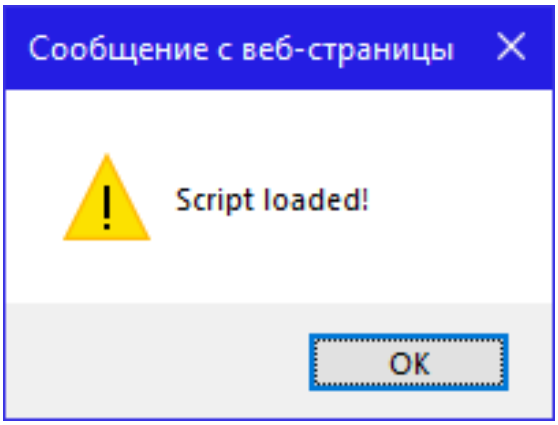


Рис. 7.1. Всплывающее окно `alert`

Подтверждение confirm(). Всплывающее окно имеет две кнопки: «ОК» и «Отмена». При нажатии на кнопку «ОК» в скрипт возвращается значение `true`. При нажатии на кнопку «Отмена» (или при закрытии окна) возвращается `false`. Эти значения можно использовать в дальнейшем в скрипте.

Синтаксис для вызова окна аналогичный. Пишется слово `confirm('Message');`, где message – текст сообщения окна.

Пример проверки работы `confirm` показан на рис. 7.2 (окно вызывается три раза; результат вызова выводится на экран: первый раз была нажата кнопка «ОК», во второй – «Отмена»):

```
<script type="text/javascript">
      var a = confirm('Choose a button!');
      var b = confirm('Choose a button!');
      document.write(a, ' ', b);
      confirm('Choose a button!');
  </script>
```

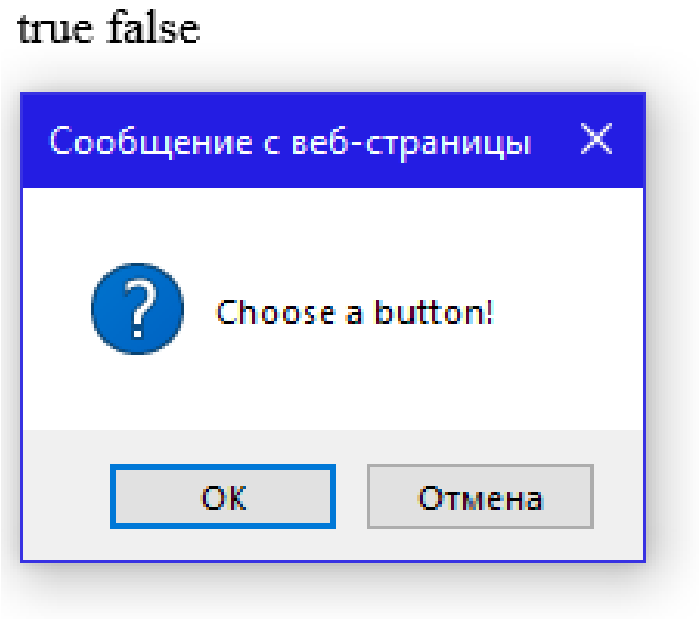


Рис. 7.2. Проверка работы `confirm`

Запрос текста prompt(). Во всплывающем окне есть кнопки «ОК», «Отмена» и поле для ввода текста пользователем. Окно возвращает пользовательский текст (если он введён и нажата кнопка «ОК» или null).

Синтаксис следующий: `prompt(message, help);` где `message` – сообщение, выводимое в окне; `help` – текст, уже по умолчанию введённый в текстовое поле.

Проверка работы `prompt` отражена на рис. 7.3 (окно вызывается три раза; результат вызова выводится на экран: первый раз было введено «`Text`» и нажата кнопка «ОК», во второй – кнопка «Отмена»):

```
<script type="text/javascript">
     var a = prompt('Write a text', 'Write here');
     var b = prompt('Write a text', 'Write here');
     document.write(a, ' ', b);
     prompt('Write a text', 'Write here');
</script>
```

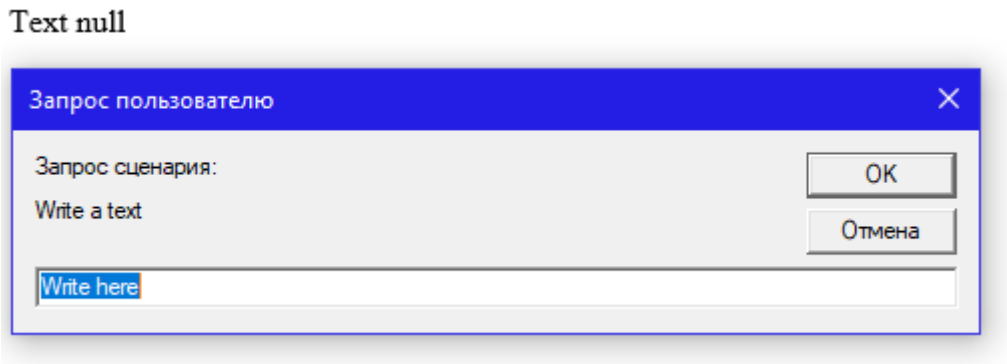


Рис. 7.3. Проверка работы `prompt`

Функции. Функции позволяют выполнять один набор инструкций сколько угодно раз при вызове. Вызов всплывающих окон – вызов функции, выводящей всплывающее окно. Можно создавать собственные функции и вызывать их, как и всплывающее окно.

Функция позволяет выполнять одно и то же действие без дублирования кода.

Синтаксис. Сначала пишется ключевое слово `function`. Затем пишется название функции.

Название содержит латинские буквы `A – z`*, цифры* `0 – 9`*, символы* `$` *(только первым символом) и* `_`*. Название не может содержать пробелов. Цифра не может быть первым символом в названии. Название не может состоять из ключевого слова, такого как* `function`*,* `var`*,* `switch` *и т. д.*

После названия идут скобки, в которых можно перечислить принимаемые *параметры* – переменные, которые поступают в функцию при её вызове и могут быть обработаны внутри. Для того чтобы объявить параметры, необходимо написать их названия через запятую. Правила для названий такие же, как для названий функций или переменных.

После скобок ставятся фигурные скобки, внутри них пишется тело функции – набор инструкций, которые она выполняет.

Каждое действие, выполняемое в скрипте, должно быть завершено точкой с запятой. Так, например, после вызова функции или объявления ставится точка с запятой – это обязательное правило для функционирования скрипта. После фигурных скобок ставить точку с запятой не нужно.

Возврат значения. После вызова функция может вернуть какое-либо значение, полученное в ходе её работы. Это значение можно будет использовать далее в программе.

Для возврата значения в теле функции пишется ключевое слово `return` и после него указывается возвращаемое значение в виде конечного значения или переменной.

Пример функции (запрашивает у пользователя текст и выводит его):

```
<script type="text/javascript">
      function test(test1, test2) {
          var a = prompt(test1, test2);
          return a;
      }
      alert(test('Text One', 'Text Two'));
</script>
```

Типы и переменные. Данные в языке JavaScript могут быть представлены в типах: числовой, строковый, логический, `null`, `undefined`. JavaScript – не типизированный язык. При создании переменной не нужно указывать её тип. Он определяется программой при задании значения переменной.

Переменная – контейнер для хранения данных любого типа; тип может изменяться с изменением значения переменной. В одну и ту же переменную можно поместить сначала число, затем строку, потом логический тип – и эти действия не вызовут ошибок.

Для того чтобы присвоить переменной строку, необходимо написать текст строки в двойных или одинарных кавычках; для того чтобы присвоить число – число, дробное число – целую и дробную часть, разделённые точкой. Логический тип имеет значения `true` и `false`. Объявленная переменная без значения в ней имеет тип `undefined`.

Переменная объявляется ключевым словом `var`. После ключевого слова пишется название переменной. Для того чтобы задать значение переменной, после её названия пишется символ `=` и присваиваемое значение.

Название содержит латинские буквы `A – z`, цифры `0 – 9`, символы `$` (только первым символом) и `_`. Название не может содержать

пробелов. Цифра не может быть первым символом в названии. Название не может состоять из ключевого слова, такого как function, var, switch и т. д.

Пример (выведет «50 true Text»):

```
<script type="text/javascript">
      var a1 = 50;
      var b_2 = 1.2;
      var cThree = "Text";
      b_2 = true;
      document.write(a1, ' ', b_2, ' ', cThree);
</script>
```

Переменные, объявленные внутри функции, могут быть использованы только внутри этой функции (локальные переменные). Переменные, объявленные вне тела функции (просто внутри тега `script`), могут быть использованы любой функцией в веб-документе (глобальные переменные).

Операторы. Операторы – это те специальные символы, которые позволяют выполнять какие-либо действия над значениями или переменными. Такое действие – это выражение. С помощью наборов выражений можно создать программу, которая будет преобразовывать исходные данные нужным способом.

Операнды – это данные, обрабатываемые сценарием JavaScript. В качестве операндов могут выступать как простые типы данных, так и сложные, а также другие выражения.

Операторы – это символы языка, выполняющие различные операции с данными. Операторы могут записываться с помощью символов пунктуации или ключевых слов.

В зависимости от количества операндов различают следующие типы операторов:

- **унарный** – в операции участвует один операнд;
- **бинарный** – в операции участвуют два операнда;
- **тернарный** – комбинирует три операнда.

Арифметические операторы – операторы, приспособленные для выполнения арифметических операций (табл. 7.1).

Таблица 7.1. *Арифметические операторы*

Оператор	Операция	Описание
+	Сложение (бинарный)	Складывает два операнда
-	Вычитание (бинарный)	Вычитает второй операнд из первого
-	Минус (унарный)	Делает положительное число отрицательным, а отрицательное – положительным
*	Умножение (бинарный)	Умножает один операнд на другой
/	Деление (бинарный)	Делит первый операнд на второй
%	Остаток от деления (бинарный)	Возвращает остаток от деления первого операнда на второй

Операторы присваивания – операторы, которые записывают значение (справа от оператора) в переменную (слева от оператора):

- Простой оператор присваивания – символ равенства (=). С ним в переменную записывается в точности то значение, что указано справа;
- Комбинированные операторы присваивания – бинарный арифметический оператор и символ равенства (+=, -=, *=, /=, %=). Такие операторы записывают в переменную результат выражения, состоящего из существующего значения переменной, указанного арифметического оператора, нового присваиваемого значения. Так, выражение `x += 5` даёт тот же результат, что и `x = x + 5`. В итоге получаем более короткую запись.

```
var a = b = x = 2; //a, b, c равны 2
x += 5; //то же, что и x = x + 5;
x -= 5; //то же, что и x = x - 5;
x *= 5; //то же, что и x = x * 5;
x /= 5; //то же, что и x = x / 5;
x %= 5; //то же, что и x = x % 5;
```

Инкремент и декремент. Самая частая операция присваивания – увеличение или уменьшение на единицу. Она применяется в различных счётчиках. Для выполнения этих операций существует короткая запись.

Инкремент и декремент – унарные операторы, которые имеют две формы: префиксную и постфиксную. Префиксная форма – изменение значения переменной, затем её использование. Постфиксная форма – использование переменной, затем изменение её значения (табл. 7.2).

Таблица 7.2. *Инкремент и декремент*

Оператор	Форма оператора	Операция
++x	Префиксная	Увеличит x на 1, затем использует его
x++	Постфиксная	Использует x, затем увеличит его на 1
--x	Префиксная	Уменьшит x на 1, затем использует его
x--	Постфиксная	Использует x, затем уменьшит его на 1

Пример:

```
var x = y = 5;
```

Разница между префиксной и постфиксной формами ощутима только при использовании инкремента или декремента в выражениях. При выполнении этих операций отдельно (в строке только инкремент или декремент и точка с запятой) разницы нет.

Пример работы инкремента и декремента приведен в табл. 7.3.

Таблица 7.3. *Пример работы инкремента и декремента*

Выражение	Значения	x после выражения	y после выражения
`++x * 10`	`y = 6 * 10`	6	60
`x++ * 10`	`y = 5 * 10`	6	50
`--x * 10`	`y = 4 * 10`	4	40
`x-- * 10`	`y = 5 * 10`	4	50

Операторы сравнения – бинарные операторы, сравнивающие операнды между собой. Если условие выполнено, то возвращается `true`, если нет – `false`.

В качестве операндов используются числа и строки. Те операнды, которые не являются числами или строками, преобразуются к строкам или числам. Список операторов сравнения в JavaScript приведен в табл. 7.4.

Таблица 7.4. *Операторы сравнения*

Оператор	Описание	Условие
==	Равенство	`true`, если операнды равны между собой, даже если у них разные типы
:=	Неравенство	`true`, если операнды НЕ равны между собой, даже после приведения к одному типу
===	Идентичность	`true`, если операнды равны между собой и у них одинаковые типы
:==	Не идентичность	`true`, если операнды НЕ равны между собой, не приводя к одному типу
>	Больше	`true`, если первый операнд строго больше второго
>=	Больше или равно	`true`, если первый операнд больше второго или равен ему
<	Меньше	`true`, если первый операнд строго меньше второго
<=	Меньше или равно	`true`, если первый операнд строго меньше второго или равен ему

Пример:

```
2 == "2"; // true
2 != -2.0; // true
2 === "2"; // false
false === false; // true
1 !== true; // true
1 != true; // false, так как true преобразуется в 1
1 > -3; // true
3 >= "4"; // false
```

Логические операторы – работают со значениями `true` и `false`. С помощью операторов сравнения и логических операторов создаются условия – специальные ограничения для конструкций ветвления или условных циклов:

- && – «логическое И» – бинарный оператор; возвращает `true`, если оба операнда являются `true`;

• || – «логическое ИЛИ» – бинарный оператор; возвращает `true`, если хотя бы один оператор является `true`;

• ! – «логическое НЕ» – унарный оператор; возвращает `true`, если значение равно `false`. Возвращает `false`, если значение равно `true`.

Пример:

```
    (2 < 3) && (3!==”3”); // (2 < 3) = true, 3!==”3” =
true, true && true = true
    (x < 10 && x > 0); // true для x строго меньше 10
и x строго больше 0
    !false; // true
```

Строковые операторы. Существуют особые операторы, которые выполняют операции над строками:

• + – соединение двух строк в одну;

• += – соединение текущего значения переменной и строки.

Операторы сравнения также работают со строками. При сравнении используются номера символов в кодировке `Unicode`, поэтому регистр (написание заглавными или строчными буквами) влияет на результат сравнения.

Пример:

```
  "12" + 8; //"128"
  var x = "Text ";
  x += 1 + 2;
  x == "Text 12"; //true
  "ABC" == "abc"; //false
```

Комментарии – это текст, который не влияет на выполнение кода, не способствует его пониманию. Используется, чтобы объяснить другому разработчику, что выполняет код. Существует два типа комментариев:

• ***однострочный*** – после «//» строка становится комментарием;

• ***многострочный*** – символы и строки между «/*» и «*/» являются комментарием.

Задания

Задание 1. Создайте веб-страницу, выводящую окно с сообщением (`alert`).

Задание 2. Создайте веб-страницу со скриптом. Внутри скрипта объявите переменную `counter`, равную `0`, и функцию `okCheck` без параметров. Функция показывает всплывающее окно `confirm`. Возвращаемое значение окна прибавьте к переменной `counter`. После прибавления будет выведено окно `alert` с переменной `counter`.

На веб-странице должна быть кнопка, которая вызывает функцию.

Задание 3. По клику на кнопку вызовите функцию. Функция выводит окно `prompt`. В окне предлагается ввести число от 1 до 10. Число сравнивается с числом 7.5. Возвращается `true`, если число строго больше 7.5. В окне `alert` выводится сообщение с результатом сравнения.

Задание 4. Создайте переменную и поместите в неё строку «`Result`». На веб-странице пять раз выводится окно `prompt` с сообщением созданной переменной. К этой переменной необходимо каждый раз прибавлять результат окна `prompt`. После всего следует вывести окно `alert` с переменной.

Контрольные вопросы

1. Какие атрибуты тега `script` вам известны?
2. В какой части веб-страницы можно поместить тег `script`?
3. Как вызвать функцию при нажатии на кнопку?
4. Какие виды всплывающих окон существуют?
5. Как получить результат со всплывающего окна `prompt`?
6. Как вернуть значение из функции?
7. Как создать переменную в JavaScript?
8. Какие типы операторов существуют в зависимости от количества операндов?
9. Какой результат выдаст сравнение: 5==5.0, 5===5.0, 5!=“5.0”?
10. Как соединить две строки в одну?

Лабораторная работа № 8

МАССИВЫ И УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ JAVASCRIPT

Цель работы

1. Научиться создавать и использовать массивы.
2. Научиться применять управляющие конструкции JavaScript.

Ход работы

Массивы. Массив (array) – это упорядоченный набор пронумерованных значений. Каждое значение называется элементом массива, а его номер – его индексом.

Особенности массива в JavaScript:

- значения в массиве могут быть разных типов, например в одном массиве можно хранить и имя человека (строка), и его возраст (число), и т. д.;
- к каждому из значений можно обратиться по номеру; нумерация начинается с 0;
- значение под номером можно легко поменять или добавить, если присвоить новому номеру значение.

Массив можно представить в виде таблицы. Первую строку видит только скрипт. Физически она не существует. Вторая строка задаётся при написании скрипта.

0	1	2	3
'User'	'Name'	17	true

Для того чтобы создать массив, нужно объявить переменную с помощью ключевого слова `var`, затем написать название массива. Далее следует присвоить (с помощью символа «=») ему значения, которые пишутся в квадратных скобках, через запятую.

Массив из таблицы создаётся так:

```
var person = ['User', 'Name', 17, true];
```

Можно создать пустой массив:

```
var empty = [];
```

Значения массива получают так: пишут название массива, затем в квадратных скобках – номер нужного значения (нужной ячейки массива).

Получить значение `'Name'` из массива `person` и присвоить его переменной `x`:

```
var x = person[1];
```

Заменить элемент `'Name'` для массива `person`:

```
person[1] = 'New Name';
```

Массив можно вывести, просто написав его название:

```
alert(person);
```

Можно получить длину массива как значение. Длина массива – это количество элементов в нём. Длина равна последнему индексу в массиве плюс один. Свойство `length` возвращает количество элементов: `person.length;`.

Для того чтобы добавить значение в массив, нужно написать имя массива, номер следующего по порядку элемента (ещё не существующего) и присвоить этому элементу значение. Пример для массива `person`:

```
person[4] = 'New Information';
```

или

```
person[person.length] = 'New Information';
```

Управляющие конструкции – это ключевые слова, изменяющие порядок выполнения действий в скрипте. Управляющие конструкции бывают двух типов:

- условные – согласно условию, выполняют одну часть кода, а другую не выполняют;
- циклические – выполняют одну часть кода множество раз.

Циклы можно делать вложенными, т. е. один цикл может быть помещён в другой. Условный оператор `if…else` также можно поместить в тело цикла или другого условного оператора.

Условный оператор if…else. Подразумевает выполнение одной части кода из двух по определённому условию.

Синтаксис. Пишется ключевое слово `if`, после него в скобках – условие. Условие – это выражение, решение которого подразумевает `true` или `false`. Если условие не равно `true` или `false`, JavaScript будет пробовать преобразовать его (*число 0, пустая строка "",* `null` *и* `undefined`*, а также* `NaN` *являются* `false`*, остальные значения –* `true`).

После условия идёт блок кода, который должен выполниться, если условие равно `true`. Его необходимо заключить в фигурные скобки.

Затем можно (но не обязательно) прописать ключевое слово `else`. Часть кода в фигурных скобках после ключевого слова `else` выполнится, если условие равно `false`.

После слова `else` можно написать ещё один блок `if`, если необходимо проверить другое условие.

Условный оператор `if…else` в общем виде выглядит следующим образом:

```
    if (условие1) {
      // выполнится, когда условие верное (true)
    } else if (условие2) {
      // выполнится, когда условие1 неверно, а усло-
вие2 верно
    } else {
      // выполнится, когда ни одно условие не верно
    }
```

Пример:

```
    if (x > 7.5) {
      alert('Greater ');
    } else if (x == 7.5) {
      alert('Equal ');
    } else {
      alert('Lower ');
    }
```

Тернарный оператор – это единственный оператор с тремя операндами в JavaScript. Если в `if…else` нужно обработать только два случая (когда условие верно и когда оно неверно), то можно использовать *тернарный оператор*. Этот оператор проверяет условие и, если

оно верно, возвращает одно значение, а если неверно – другое. Тернарный оператор выглядит так:

```
условие ? значение1 : значение2
```

Пример:

```
alert(x >= 7.5 ? 'Greater or Equal ' : 'Lower ');
```

Цикл while (с предусловием). Цикл `while` – это условный цикл. Перед выполнением тела цикла проверяется условие. Если условие верно (`true`), то выполняется тело цикла. Как только тело выполнилось – условие проверяется снова. Это повторяется до тех пор, пока условие остается верным (`true`). Если условие ложно (`false`), то тело цикла не выполняется. Таким образом, если условие изначально было ложно, то цикл не выполнится ни разу.

Цикл, в котором условие проверяется до выполнения тела цикла, называется *циклом с предусловием*.

Синтаксис. Пишется ключевое слово `while`, затем в скобках – условие, определяющее, до каких пор должен повторяться цикл. Потом в фигурных скобках пишется тело цикла – набор действий программы, которые должны повторяться. В общем виде синтаксис выглядит следующим образом:

```
while (условие) {
     // повторяющееся тело цикла
}
```

Пример (цикл выполняется, пока пользователь нажимает «ОК»):

```
var answer = true;
while (answer == true) {
   answer = confirm('Press OK to continue ');
}
```

Для того чтобы цикл выполнился первый раз, следует задать начальное значение `true` переменной. Чтобы этого не делать, можно использовать другую управляющую конструкцию.

Цикл do…while (с постусловием). Фактически это тот же условный цикл, что и `while`, но условие проверяется после выполнения тела

цикла. В этой конструкции код тела гарантированно выполнится по крайней мере один раз, даже если условие ложно (`false`).

Цикл, в котором условие проверяется после выполнения тела цикла, называется *циклом с постусловием*.

Синтаксис. Пишется ключевое слово `do`, затем в фигурных скобках – тело цикла (набор действий программы, которые должны повторяться). После закрывающей фигурной скобки указывается ключевое слово `while` и сразу в скобках проверяемое условие. После закрывающей скобки должна стоять точка с запятой. В общем виде синтаксис выглядит так:

```
do {
     // повторяющееся тело цикла
} while (условие);
```

Пример (цикл выполняется, пока пользователь нажимает «ОК»):

```
do {
   answer = confirm('Press OK to continue ');
} while (answer == true);
```

Задавать переменной `answer` начальное значение больше не нужно.

Цикл for (счётчик). Тело цикла выполняется определённое количество раз. Для этого в цикл встраивается переменная-счётчик. После каждого выполнения тела цикла её значение изменяется. Как только значение выходит за допустимые границы, цикл заканчивает своё выполнение.

Синтаксис. Пишется ключевое слово `for`, после него в скобках – три составляющие: начало, условие, шаг. Выражение «начало» обычно инициализирует один или несколько счётчиков.

Начало – создание переменной-счётчика, которая должна действовать внутри цикла. Пример: var i=0; var j=1; и т. д.

Условие – проверка переменной-счётчика на выход за границы.

Пример: i < 25; j > -10; и т. д.

Шаг – изменение переменной, т. е. её увеличение или уменьшение. Обычно используется инкремент или декремент. В результате этих операций переменные увеличиваются или уменьшаются на единицу.

Пример: i++; , j -= 2; и т. д.

В общем виде синтаксис выглядит так:

```
for (var переменная_счётчик; условие; шаг) {
     // повторяющееся тело цикла
}
```

Примеры (два различных цикла):

```
for (var i = 0; i < 5; i++) {
     alert(i);
}
for (var j = 16; j > 0; j -= 2) {
     alert(j);
}
```

Пример (вложенный цикл будет осуществляться весь каждый раз, когда будет выполняться тело верхнего цикла):

```
for (var i = 0; i < 5; i++) {
    for (var j = 16; j > 0; j -= 2) {
         alert(j);
    }
    alert(i);
}
```

Можно прервать выполнение цикла в любой момент – для этого используется ключевое слово `break;`. Для того чтобы пропустить выполнение тела цикла один раз, пишется ключевое слово `continue;`.

Пример (цикл пропустит вывод окна для `i==4`, а на `i==7` прервётся):

```
for (var i = 0; i < 15; i++) {
    if (i == 4) {
        continue;
    } else if (i == 7) {
        break;
    }
    alert(i);
}
```

Ветвление switch. Когда нужно выполнять свой блок кода на каждое значение переменной, помогает конструкция `switch`.

Синтаксис. Пишется ключевое слово `switch` и в скобках после него – название переменной, значения которой будут проверяться. Затем в фигурных скобках пишутся блоки `case`: ключевое слово `case`, затем пробел и значение переменной, при котором будет выполняться какое-то действие. После значения ставится двоеточие и пишется набор действий, которые должны быть выполнены. Блок `case` обычно заканчивают ключевым словом `break;`. Если этого не сделать, то значения будут проверяться дальше, а это бесполезно. Если ни одно из значений `case` не было найдено, то выполняется блок кода после ключевого слова `default:`.

В общем виде синтаксис выглядит так:

```
switch(название_переменной) {
  case 'value1':
    ...
    break;
  case 'value2':
    ...
    break;
  default:
    ...
    break;
}
```

Пример:

```
var x = prompt('Enter a number', 'Here');
switch(x) {
  case '1':
      alert('One');
      break;
  case '2':
      alert('Two');
      break;
  default:
      alert('I don’t know this number');
      break;
}
```

Задания

Задание 1. Создайте пустой массив. В цикле `while` присваивайте ему значения, которые пользователь вводит в окно `prompt`, пока пользователь не нажмёт «Отмена» (пока полученное значение не равно `null`). Если полученное значение равно `null`, вызовите `break;`. После завершения цикла выведите массив в окне `alert`.

Задание 2. С помощью конструкции `switch` создайте скрипт, который при клике на кнопку запрашивает у пользователя через окно `prompt` число от 1 до 10 и проверяет, равно ли это число 9. Если число равно 6, 7, 8 или 10, то выводится сообщение «близко» в окне `alert`. Если число равно 9, то выводится сообщение «верно». Если значение другое, выводится «неверно».

Задание 3. Создайте на странице две кнопки. При клике на первую пользователю предлагается ввести число. При клике на вторую запускается функция «Обратный отсчёт».

Функция «Обратный отсчёт» построена на цикле `for`. Этот цикл ведёт счёт от числа, введённого пользователем, до 0. Каждое число выводится в окне `alert`. Когда счётчик равен 0, выводится «Finish». Перед циклом `for` нужно выполнить проверку. Если число неверного формата или не введено, выводится `Wrong number`. Если число отрицательное, необходимо умножить его на -1.

Задание 4. Запросите у пользователя через окно `prompt` число. В цикле `for` спросите у пользователя новые числа столько раз, сколько пользователь указал до этого. Все эти числа запишите в массив. Затем в следующем цикле `for` проверьте каждое число в массиве на чётность. Все нечётные числа запишите в новый массив. Выведите массив с нечётными числами пользователю в окне `alert`.

Для проверки на чётность здесь используется оператор получения остатка от деления %.

Контрольные вопросы

1. Что такое массив в языке JavaScript?
2. Можно ли записать в один массив значения разных типов?
3. С помощью какого слова объявляется переменная в массиве?
4. Как обратиться к элементу массива?
5. Как создаётся условная конструкция `if…else`?
6. Как создать цикл `while`?
7. Чем цикл `do…while` отличается от цикла `while`?
8. Какие три составляющие пишутся в скобках в цикле `for`?
9. Как задаётся шаг для цикла `for`?
10. Зачем нужно писать `break` после каждого блока `case` в конструкции `switch`?
11. Сколько раз выполнится этот цикл?

```
var x = false;
while (x) {
    alert(x);
 }
```

Лабораторная работа № 9

ОБРАБОТКА ЭЛЕМЕНТОВ ВЕБ-СТРАНИЦЫ С ПОМОЩЬЮ JAVASCRIPT

Цель работы

Научиться обрабатывать данные HTML-формы с помощью JavaScript.

Ход работы

Создание формы для теста. Для того чтобы обработать данные с помощью JavaScript, необходимо создать форму, куда эти данные будут занесены. Создадим тест, в котором пользователь будет отвечать на вопросы. Ответы на вопросы могут быть правильными и неправильными. За правильный ответ пользователь будет получать баллы, за неправильный – нет. В итоге будут выводиться баллы, которые набрал пользователь, и максимально возможное количество баллов.

В этом тесте будут обрабатываться данные переключателей (элемент формы `radio`), флажков (`checkbox`) и текстовых полей.

Нужно создать форму с указанными элементами (см. лабораторную работу № 6). После формы стоит кнопка «`Get the score!`», выводящая итоговый результат.

Код формы теста (рис. 9.1):

```
  <form>
   <fieldset style="display:inline-block">
       <legend>First question</legend>
       <p><label><input  type="radio"  name="counter">
Answer 1</label></p>
       <p><label><input  type="radio"  name="counter">
Answer 2</label></p>
       <p><label><input  type="radio"  name="counter">
Answer 3</label></p>
   </fieldset>
   <fieldset style="display:inline-block">
       <legend>Second question</legend>
```

```
        <p><label><input type="checkbox" name="check">
Answer 1</label></p>
        <p><label><input type="checkbox" name="check">
Answer 2</label></p>
        <p><label><input type="checkbox" name="check">
Answer 2</label></p>
    </fieldset>
    <p><label>Text question </label><input type="text"
name="text1"></p>
  </form>
 <button onClick="score();">Get the score!</button>
```

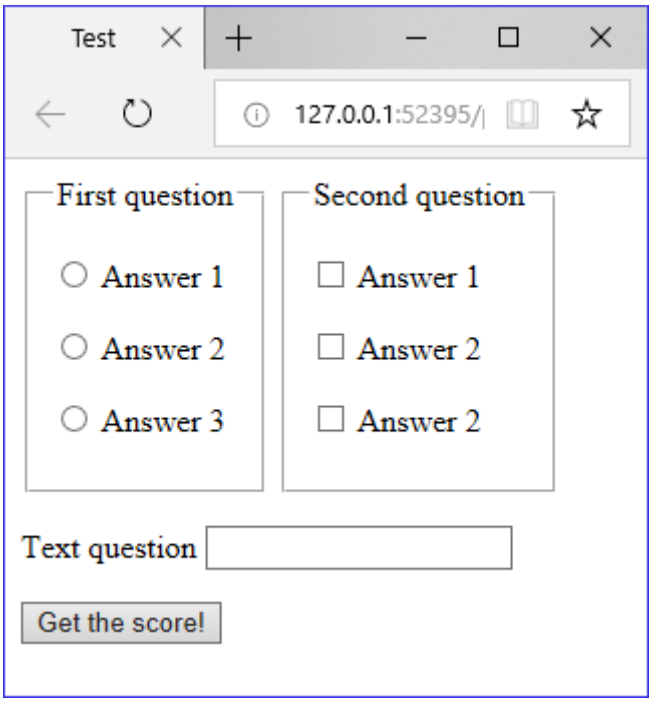


Рис. 9.1. Форма для теста

Идея теста. Форма с тестом содержит атрибут `id="test"`. Скрипт будет проверять форму с таким `id`.

Вопрос пишется в теге `<label></label>` или `<legend></legend>`. Проверять эти теги нет смысла.

Ответы на вопросы, куда пользователь будет вводить данные, содержатся в теге `input`. У этого тега существует атрибут `type`, который определяет, переключатель это, флажок или поле для текста. Исходя из типа, обрабатывать ответы нужно по-разному. Переключатель можно

выбрать один из нескольких, флажков – несколько. Текст можно ввести ЗАГЛАВНЫМИ буквами или строчными или И Заглавными, И Строчными. Всё это необходимо учитывать при обработке результатов.

Вопросы в тесте могут быть разными по уровню сложности. При ответе на лёгкий вопрос можно получить 1 балл, а на более сложный – 2 балла и больше.

Пометка правильного ответа: `data-true="1"`, где `1` – количество баллов, которое пользователь получит за выбор этого варианта.

Правильный ответ на текстовое поле: `data-ans="happy"`, где `happy` – правильный ответ на вопрос.

Атрибуты HTML, которые начинаются на `data-`, – это пользовательские атрибуты. Их можно добавить к тегу, не опасаясь, что это изменит внешний вид элемента.

Внесение нужных изменений в форму:

```
  <form id="test">
    <fieldset style="display:inline-block">
       <legend>First question</legend>
       <p><label><input  type="radio"  name="counter">
Answer 1</label></p>
       <p><label><input  type="radio"  name="counter"
data-true="1"> Answer 2</label></p>
       <p><label><input  type="radio"  name="counter">
Answer 3</label></p>
    </fieldset>
    <fieldset style="display:inline-block">
       <legend>Second question</legend>
       <p><label><input  type="checkbox"  name="check"
data-true="1"> Answer 1</label></p>
       <p><label><input type="checkbox" name="check">
Answer 2</label></p>
       <p><label><input  type="checkbox"  name="check"
data-true="1"> Answer 2</label></p>
    </fieldset>
    <p><label>Text question </label><input type="text"
name="text1" data-true="2" data-ans="happy"></p>
  </form>
```

Добавление скрипта к веб-странице. Необходимо создать файл с расширением `.js`. (пример: `score.js`) В тег `<head></head>` пишется тег `<script src="score.js"></script>`. Теперь файл со скриптом подключен к веб-странице через атрибут `src`.

Создание функции `score()`. В файле `js` следует создать функцию с названием `score`. Параметров она не принимает.

Пример:

```
function score() { }
```

Получение элементов с веб-страницы. Для работы с документами HTML в языке JavaScript существует отдельный объект – Document. Пользуясь его свойствами и методами, сценарий JavaScript может получить информацию о текущем документе, загруженном в окно браузера, а также управлять отображением содержимого этого документа. Слово `document` отправляет к HTML-документу, к которому привязан скрипт.

Метод – функция, которая привязана к элементу определённого типа. При вызове метода выполняется какое-либо действие. ***После вызова метода обязательно идут скобки.*** *Метод пишется после названия переменной через точку.*

Пример (метод `write` пишет текст `Message` на веб-странице `document`):

```
document.write('Message');
```

Если у элемента есть атрибут `id`, то можно получить его вызовом `document.getElementById(id),` где бы он ни находился. Обратимся к форме. Форме был присвоен уникальный идентификатор `id="test"`. Получить доступ к элементу с известным `id` на языке JavaScript можно через метод `getElementById('test')`, где `test` – значение атрибута `id`. Создадим переменную с формой:

```
var test = document.getElementById('test');
```

Эта переменная ссылается на форму на веб-странице. При обращении к форме через переменную `test` будет получена информация с веб-страницы.

Получим все поля input с веб-страницы. Нужный метод getElementsByTagName('input'), где input – название тега, данные которого нужно получить. Метод соберёт все элементы тега input с веб-страницы и поместит их в своеобразный массив (*коллекцию*). Внутри этой коллекции сохранятся все атрибуты тега, его значение, стиль и т. д.

Пример:

```
var elements = test.getElementsByTagName('input');
```

Теперь внутри переменной elements находятся ссылки на все элементы input с веб-страницы.

Обращаться к одному элементу можно так же, как в обычном массиве, например: elements[0] – доступ к первому элементу input на веб-странице.

Метод getAttribute('name'), где name – название нужного атрибута, получает значение атрибута с названием name.

Существует вспомогательный метод hasAttribute('name'), где name – название нужного атрибута. Этот метод возвращает true, если у элемента есть атрибут с названием name. Если атрибута нет, возвращает false. Этот метод используется в конструкции if…else.

Пример:

```
if (elements[i].hasAttribute('name') {
      queName = elements[i].getAttribute('name'));
    }
```

В функцию добавим переменную формы. Из этой переменной возьмём все элементы input. Проверим, сколько их получилось, с помощью слова length и выведем результат в окне.

Пример (рис. 9.2):

```
function score() {
 var test = document.getElementById('test');
 var elements = test.getElementsByTagName('input');
 alert(elements.length);
}
```

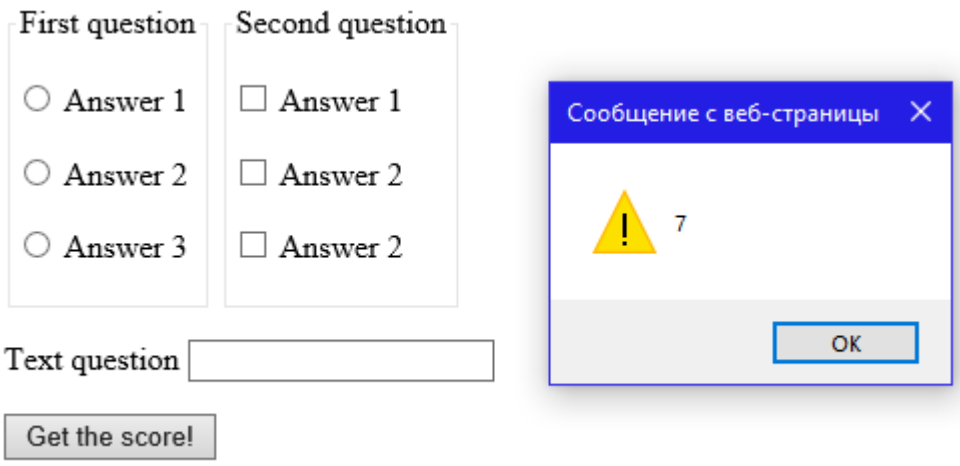


Рис. 9.2. Вывод количества элементов `input`

Функция подсчёта максимального количества баллов. Баллы за каждый правильный ответ записываются в атрибуте `data-true` правильного ответа. Для того чтобы узнать максимальное количество баллов, необходимо посчитать сумму значений всех атрибутов `data-true`.

Функция будет принимать один параметр. Параметры представлены коллекцией всех элементов `input`:

1. Объявляется переменная `tmpScore`, равная 0. В нее будут суммироваться баллы *(`var tmpScore = 0;`)*.
2. Создаётся цикл `for`. Этот цикл проверяет все элементы в коллекции `elements` *(`for (var i = 0; i < elements.length; i++)`)*.
3. В цикле имеется условная конструкция `if…else`. В ней проверяется, есть ли атрибут `data-true` у текущего элемента *(`elements[i].hasAttribute('data-true')`)*.
4. Если атрибут `data-true` присутствует, то берётся его значение *(`elements[i].getAttribute('data-true')`)* и прибавляется к переменной `tmpScore`.
5. После выполнения цикла возвращается значение переменной `tmpScore`. *(`return tmpScore;`)*.

```
function countMaxScore(elements) {
  var tmpScore = 0;
  for (var i = 0; i < elements.length; i++) {
    if (elements[i].hasAttribute('data-true')) {
```

```
        tmpScore   +=   parseInt(elements[i].getAttrib-
ute('data-true'));
        }
      }
      return tmpScore;
    }
```

По умолчанию язык JavaScript считает значение атрибута `data-true` строкой. Но нужно использовать его как число. В примере для этого применяется функция `parseInt`. Функция `parseInt()` принимает строку в качестве аргумента и возвращает целое число в соответствии с указанным основанием системы счисления.

Пример использования функции `parseInt`:

`parseInt('15')`, где `'15'` – строка, которую нужно перевести в число. Функция преобразует строку в целое число.

`parseFloat('1.6')`, где `'1.6'` – строка, которую нужно перевести в дробное число. Функция преобразует строку в дробное число.

Применим созданную функцию. Для этого создадим в функции `score` переменную `maxScore`. Эта переменная получает значение из написанной функции. Как параметр она принимает коллекцию `elements`.

Пример:

```
        var maxScore = countMaxScore(elements);
```

Функция подсчёта правильных ответов пользователя. Для подсчета правильных ответов пользователя необходимо выполнить следующие этапы (шаги).

1. Создать нужные переменные в функции `score`. `userScore`, в которую будут суммироваться баллы (равна 0); `queName`, где будет храниться название текущего вопроса *(у каждого вопроса своё название, которое записано в атрибуте `name`, чтобы связывать между собой разные варианты ответа)*; `answers` – массив с вариантами ответа на один и тот же вопрос; `i` – переменная-счётчик.

```
    var userScore = 0;
    var queName;
    var answers;
    var i = 0;
```

2. Создать цикл, который берёт следующий вопрос, пока они не закончатся. Он контролирует переменную-счётчик `i`, которая не может быть больше, чем количество ответов на вопросы (`input`).

```
    while (i < elements.length) { }
```

3. В теле цикла выбрать вопрос для проверки. Один вопрос отличается от другого названием. Название содержится в варианте ответа (`input`) в атрибуте `name`. Поэтому выбор вопроса – сохранение в переменную названия текущего вопроса.

```
    queName = elements[i].getAttribute('name');
```

4. Очистить массив `answers`. Так как этот массив хранит ответы на текущий вопрос, для каждого вопроса он должен быть свой.

```
    answers=[];
```

5. Сохранить все возможные ответы на вопрос в массив `answers`.

Для этого создать ещё один цикл. Цикл проверяет два условия: что `i < elements.length` *(так как этот цикл изменяет переменную-счётчик i и должен столкнуться с концом коллекции)* и атрибут `name` ответа на вопрос совпадает с текущим названием вопроса *(так как все варианты ответа на вопрос должны быть расположены подряд)*.

```
   (i < elements.length && elements[i].getAttrib-
ute('name') == queName)
```

Внутри цикла следующему элементу в массиве (`answers [answers.length]`) присваивается текущее значение ответа (`elements[i]`).

Затем счётчик `i` увеличивается на 1 (`i++`).

```
   while (i < elements.length && elements[i].getAt-
tribute('name') == queName) {
            answers[answers.length] = elements[i];
            i++;
        }
```

6. Теперь есть массив со всеми ответами на вопрос. Внутри ответа содержится информация, правильный он или нет, а также отметил ли его пользователь (в случае с текстовым полем – текст, введённый пользователем).

В зависимости от типа ответ на вопрос (значения атрибута `type` в теге `input`) обрабатывают по-разному. Этим будут заниматься отдельные функции, которые пока не созданы. Этим функциям нужно будет передать массив с ответами, если он не пустой. Значение, возвращаемое функцией, необходимо прибавить к переменной `userScore`.

Проверяем, есть ли в массиве элементы, т. е. длина массива должна быть больше 0.

```
    if (answers.length > 0) { }
```

Затем проверяем тип ответа на вопрос. Логика для переключателей (`radio`) и флажков (`checkbox`) одинаковая и её можно объединить.

```
    if (answers[0].getAttribute('type') == 'radio' ||
            answers[0].getAttribute('type') == 'check-
box') {
        userScore += radioCheckAnswer(answers);
    }
```

Текстовое поле подразумевает другую логику. В этом случае требуется другая проверка.

```
  else if (answers[0].getAttribute('type') == 'text')
{
  userScore += textAnswer(answers[0]);
  }
```

Внутри к переменной `userScore` прибавляются значения функций, которые ещё не существуют. Их обязательно нужно написать в дальнейшем.

Цикл завершен, условия выполнены.

7. Результат выводится пользователю в окне `alert`. Символ `\n` – это знак перевода на следующую строку. Символы «+» объединяют несколько строк и переменные в одну строку.

```
    alert("Your score: " + userScore + "\nMax score:
" + maxScore);
```

Функции анализа ответов пользователя состоят из пяти шагов, которые нужно создать.

Функция проверки переключателей и флажков. Принимает как параметр массив с ответами.

1. Создать переменную для сохранения баллов пользователя `tmpScore` (равна 0).

2. В цикле `for` просмотреть каждый ответ.

3. Проверим правильность отмеченных при помощи элементов формы `checkbox` ответов. Специальное слово `checked` пишется к ответу через точку. `Checkbox` – это флажок, переключатель. Такая конструкция возвращает `true`, если ответ отмечен. Это подходит и для переключателей (`radio`), и для флажков (`checkbox`).

```
answers[i].checked
```

4. Если ответ правильный и отмечен пользователем, прибавить к переменной `tmpScore` значение атрибута `data-true`.

```
  tmpScore += parseInt(answers[i].getAttribute('data-
true'));
```

5. После цикла вернуть переменную `tmpScore`.

Код функции:

```
function radioCheckAnswer(answers) {
    var tmpScore = 0;
    for (var i = 0; i < answers.length; i++) {
        if (answers[i].hasAttribute('data-true')
&& answers[i].checked) {
            tmpScore += parseInt(an-
swers[i].getAttribute('data-true'));
        }
    }
    return tmpScore;
}
```

Функция проверки текстового поля. Принимает как параметр один ответ (`answers[0]`).

1. Создать переменную для сохранения баллов пользователя `tmpScore` (равна 0).

2. Создать переменную, куда поместить правильный ответ на вопрос. Правильный ответ хранится в атрибуте `data-ans`. Для того чтобы не было различий в написании, с помощью метода `toLowerCase()` вывести ответ в строчных буквах.

```
   var correct = answer.getAttribute('data-
ans').toLowerCase();
```

3. Сравнить ответ пользователя (тоже в строчных буквах) с правильным. Если ответ правильный, прибавить баллы, как в предыдущей функции.

4. Вернуть переменную `tmpScore`

Код функции:

```
    function textAnswer(answer) {
        var tmpScore = 0;
        var correct = answer.getAttribute('data-
ans').toLowerCase();
        if (answer.value.toLowerCase() == correct) {
            tmpScore += parseInt(answer.getAttrib-
ute('data-true'));
        }
        return tmpScore;
    }
```

Итоговый код веб-страницы и скрипта (рис. 9.3):

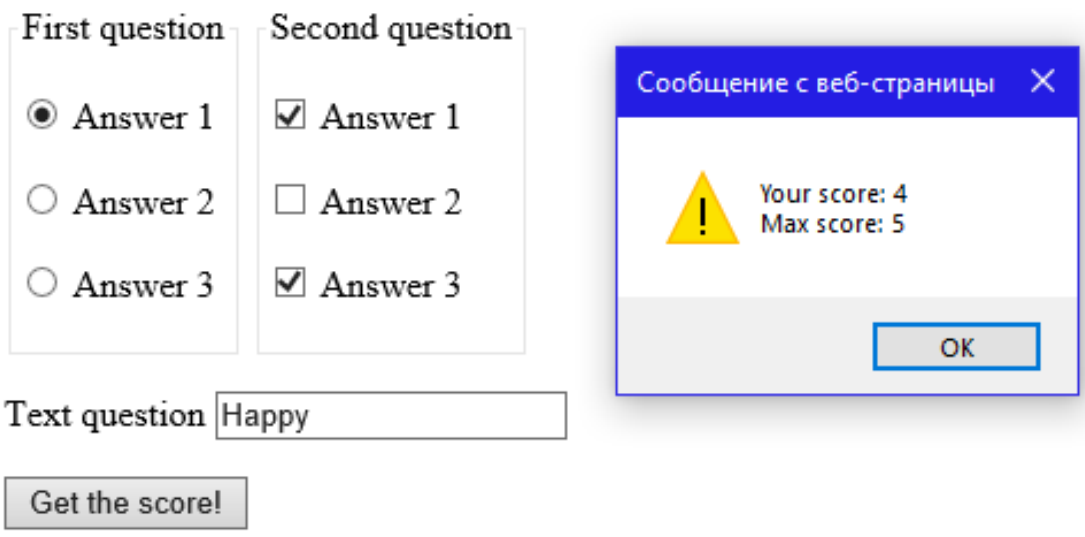


Рис. 9.3. Полная веб-страница

Файл score.js

```
    function countMaxScore(elements) {
       var tmpScore = 0;
       for (var i = 0; i < elements.length; i++) {
         if (elements[i].hasAttribute('data-true')) {
            tmpScore +=
 parseInt(elements[i].getAttribute('data-true'));
            }
       }
       return tmpScore;
    }

    function radioCheckAnswer(answers) {
       var tmpScore = 0;
       for (var i = 0; i < answers.length; i++) {
          if (answers[i].hasAttribute('data-true') &&
answers[i].checked) {
                tmpScore +=
 parseInt(answers[i].getAttribute('data-true'));
            }
       }
       return tmpScore;
    }

    function textAnswer(answer) {
       var tmpScore = 0;
       var correct = answer.getAttribute('data-
ans').toLowerCase();
       if (answer.value.toLowerCase() == correct) {
            tmpScore +=
parseInt(answer.getAttribute('data-true'));
       }
       return tmpScore;
    }
```

```
    function score() {
       var test = document.getElementById('test');
       var elements = test.getElementsByTagName('in-
put');

       var maxScore = countMaxScore(elements);
       var userScore = 0;

       var i = 0;
       var queName;
       var answers;
       while (i < elements.length) {
         queName = elements[i].getAttribute('name');
         answers=[];

         while (i < elements.length && ele-
ments[i].getAttribute('name') == queName){
            answers[answers.length] = elements[i];
                i++;
            }

            if (answers.length > 0) {
                if (answers[0].getAttribute('type')
== 'radio' ||
                    answers[0].getAttribute('type')
== 'checkbox') {
            userScore += radioCheckAnswer(answers);
                } else if (answers[0].getAttrib-
ute('type') == 'text')
    {
                userScore += textAnswer(answers[0]);
                }
            }
       }
       alert("Your score: " + userScore + "\nMax
score: " + maxScore);
    }
```

Файл test.html

```
<!doctype html>
<html>
<head>
   <meta charset="utf-8">
   <title>Test</title>
   <script src="score.js"></script>
</head>

<body>
   <form id="test">
        <fieldset style="display:inline-block">
            <legend>First question</legend>
            <p><label><input type="radio"
name="counter"> Answer 1</label></p>
            <p><label><input type="radio"
name="counter" data-true="1"> Answer 2</label></p>
            <p><label><input type="radio"
name="counter"> Answer 3</label></p>
        </fieldset>
        <fieldset style="display:inline-block">
            <legend>Second question</legend>
            <p><label><input type="checkbox"
name="check"  data-true="1"> Answer 1</label></p>
            <p><label><input type="checkbox"
name="check"> Answer 2</label></p>
            <p><label><input type="checkbox"
name="check"  data-true="1"> Answer 3</label></p>
        </fieldset>
        <p><label>Text question </label><input
type="text" name="text1" data-true="2" data-
ans="happy"></p>
   </form>
<button onClick="score();">Get the score!</but-
ton>
</body>
</html>
```

Задание

Создайте тест, как в лабораторной работе. Вопросы должны быть другими и подразумевать реальные ответы. Должно быть три вопроса с переключателями (`radio`), три вопроса с флажками (`checkbox`) и четыре вопроса с ответом в виде текстового поля.

Имейте в виду, что ответ в виде текстового поля должен быть простым и будет лучше, если он состоит из одного слова.

Контрольные вопросы

1. Как вызвать функцию JavaScript по клику на кнопку?
2. Как добавить к веб-странице скрипт из отдельного файла?
3. Как получить доступ к элементу по его `id` в языке JavaScript?
4. Как записать элемент веб-страницы в переменную по `id`?
5. Как получить атрибут из элемента веб-страницы?
6. Как преобразовать значение атрибута в целое число?
7. Как добавить элемент веб-страницы в массив?
8. Как очистить массив?
9. Как проверить, что в массиве есть элементы?
10. Как проверить, есть ли атрибут с названием у элемента?

Лабораторная работа № 10

СОЗДАНИЕ САЙТА С ИСПОЛЬЗОВАНИЕМ HTML, CSS И JAVASCRIPT

Цель работы

Создать сайт с использованием HTML, CSS и JavaScript.

Ход работы

Основа сайта. Для сайта будет использована блочная вёрстка (см. лабораторную работу № 4). Необходимо создать главную страницу (`Home`), страницу с информацией об авторе (`About`) и страницу с тестом для пользователей (`Test`).

Главная страница. На главной странице необходимо разместить большое изображение, которое будет случайно меняться при каждой загрузке страницы. Поверх изображения – название сайта. По центру будет расположено большое меню, для того чтобы пользователь мог перейти на другие страницы; снизу – основная часть с описанием сайта и нижняя часть («подвал») для дополнительной информации.

Для начала используем блочную вёрстку, созданную в лабораторной работе № 4.

Начало:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Test</title>
    <link     rel="stylesheet"     type="text/css"
href="style.css">
</head>

<body>
<div id="home">
        <header><p>header</p></header>
        <nav><p><b>nav</b>igation</p></nav>
</div>
<div id="container">
```

```
        <main>
            <p>main</p>
            <section>
                <p>section</p>
                <article>
                    <p>article</p>
                </article>
                <article>
                    <p>article</p>
                </article>
            </section>
        </main>
        <aside><p>aside</p></aside>
        <footer><p>footer</p></footer>
    </div>
    </body>
    </html>
```

Так как главная страница будет немного отличаться от остальных, теги header и nav помещают в тег `<div id="home">…</div>`, а остальные – в тег `<div id="container">…</div>`.

Файл style.css содержит настройки расположения блоков сайта:

```
body{
    width:90%;
    height:800px;
    margin:0 auto;
    text-align:center;
    font: 24px Consolas bold;
    font-family: Consolas;
}
header {
    width:100%;
    background: #dfdfdf;
    border: 10px white solid;
}
```

```
nav {
    width:100%;
    height:auto;
    background: #dfdfdf;
    border: 10px white solid;
}
main {
    width: 76%;
    background: #dfdfdf;
    display: inline-block;
    border: 10px white solid;
    padding: 2%;
}
aside {
    width:18%;
    height: 30%;
    display: inline-block;
    background: #dfdfdf;
    border: 10px white solid;
    margin-right: -20px;
    float: right;
}
footer {
    width:100%;
    height:10%;
    background: #dfdfdf;
    float: left;
    border: 10px white solid;
}
section {
    width: 80%;
    margin: 0 auto;
    background: tomato;
    padding: 2%;
}
```

```
article {
    width: 80%;
    padding: 1%;
    background: #efefef;
    margin: 10px auto;
}
```

Большое изображение на главной странице. Для того чтобы на главной странице картинка была больше, чем на других страницах сайта, необходимо изменить `header` для отображения на главной. Подключимся к нему в файле `style.css`:

```
#home header {}
```

`#home` – обращение к элементу по `id`; `home` – значение атрибута `id` элемента.

`.home` – обращение к элементу по классу; `home` – значение атрибута `class` элемента.

`id` элемента должен быть уникальным и не повторяться на веб-странице, т. е. не может быть двух элементов с одинаковым `id` на одной веб-странице. `class` элемента, наоборот, задаёт общее оформление многим элементам.

Зададим тегу `header` на главной странице отображение по всей ширине с высотой `80vh`. Для того чтобы блок не зависел от родительских элементов, назначим `position: absolute;` Блоку с `id="container"` определим отступ от верха `80vh`.

Для наглядности зададим блоку закругление нижних углов.

```
#home header {
    height: 80vh;
    width: 100vw;
    position: absolute;
 top:0;
 left:0;
    border: none;
 border-radius: 0 0 50px 50px;
}
#container {
    margin-top: 80vh;
}
```

Создадим фоновое изображение. Есть директория `wall` с четырьмя изображениями больших размеров. Возьмем изображение `elephant.jpg` и наложим на элемент `header` (не только для главной). Растянем его по ширине с помощью `background-size: cover;` и выровняем по центру с помощью `background-position: center center;`. Сразу зададим белый цвет текста.

Наложим цвет фона на изображение. Фон – чёрный полупрозрачный цвет, определенный с помощью модели `rgba`. Но его не видно за изображением. Для того чтобы объединить цвета изображения и фона, нужно использовать режим смешивания.

Режим смешивания – это математически рассчитанное браузером наложение основного цвета на цвет фона. ***Работает только в современных браузерах!***

Все режимы смешивания можно разделить на следующие группы:

1) нормальный режим – `normal`*;*

2) режим контраста – `overlay`*,* `soft-light`*,* `hard-light`*;*

3) режим затемнения – `darken`*,* `multiply`*,* `color-burn`*;*

4) режим осветления – `lighten`*,* `screen`*,* `color-dodge`*;*

5) режим сравнения – `difference`*,* `exclusion`*;*

6) компонентный режим – `hue`*,* `saturation`*,* `color`*,* `luminosity`*.*

Используем режим контраста `soft-light`.

Стиль элемента `header` преобразовался:

```
header {
    color: white;
    width:100%;
    background: #dfdfdf;
    border: 10px white solid;
 border-radius: 50px 50px 50px 50px;

    background-image: url("wall/elephant.jpg");
    background-size: cover;
    background-position: center center;
    background-blend-mode: soft-light;
    background-color: rgba(0,0,0,0.3);
}
```

На рис. 10.1 представлен промежуточный вариант главной страницы.

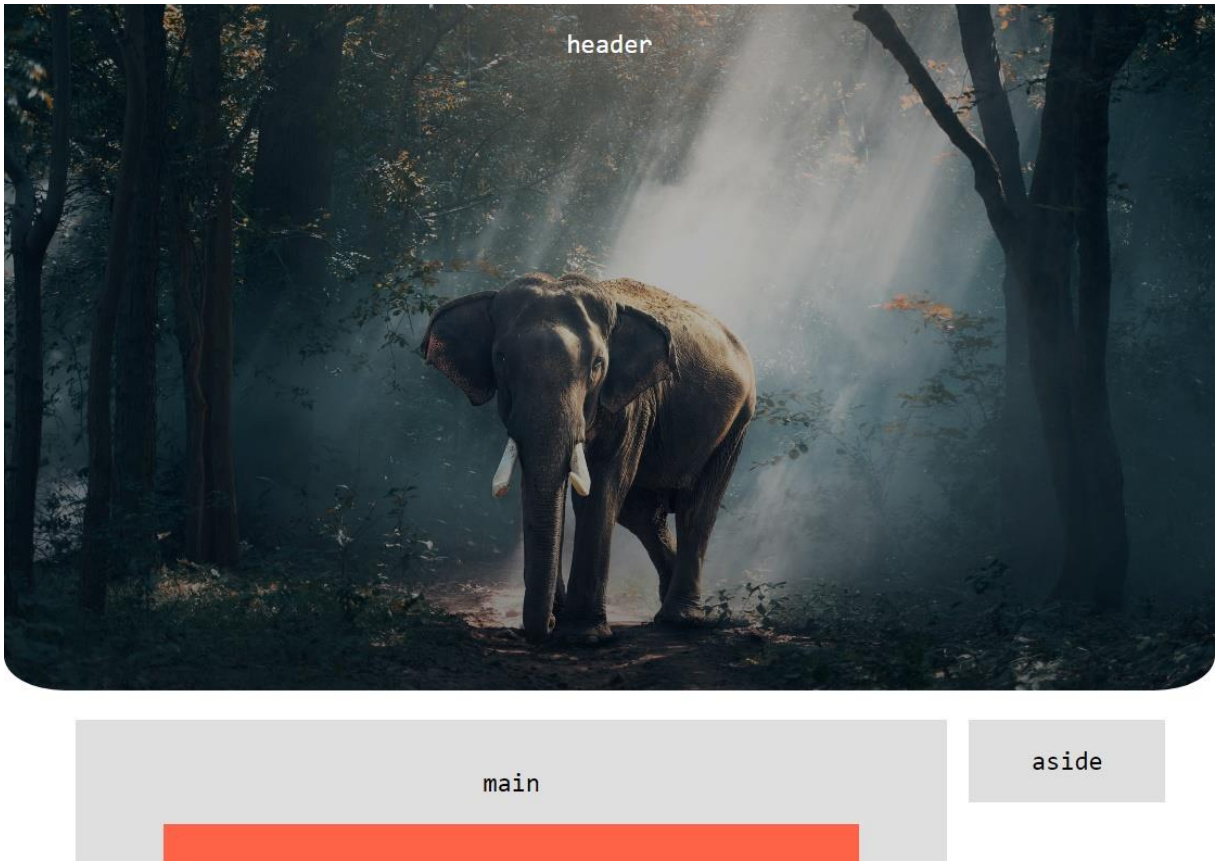


Рис. 10.1. Промежуточный вид главной страницы

Панель навигации на главной странице. В HTML-код встроим панель навигации из лабораторной работы № 4.

```
<div id="home">
     <header><p>header</p></header>
     <nav>
<ul>
            <li><a href="about.html">About</a></li>
            <li><a href="index.html">Home</a></li>
            <li><a href="test.html">Test</a></li>
       </ul>
   </nav>
 </div>
```

В стили CSS необходимо встроить оформление для панели навигации с изменениями для корректного отображения перед изображением.

Стили CSS для меню:

```
    nav {
        width:100%;
        font-size: 1.2em;
     position: fixed;
     top: 0;
        left:0;
        height:58px;
        background-color: rgba(255, 255, 255, 0.75);
        border: none;
        text-align: center;
    }
    nav ul {
        list-style-type: none;
        margin: 0;
        padding-left: 0;
        text-align: center;
    }
    nav ul li {
        display: inline-block; /*show in one line*/
        margin: 5px; /*margin between elements*/
        padding: 10px; /*padding inside element*/
        transition: all 0.5s ease-in-out; /*ainmation*/
        border-bottom: 0px solid red;
    }
    nav a {
        text-decoration:  none;  /*remove  link  under-
line*/
        color: tomato;
        transition: all 0.5s ease-in-out; /*ainmation*/
    }
    nav a:hover {/*link under pointer*/
        color: red;
    }
```

```
nav ul li:hover { /*element under pointer*/
    border-bottom: 5px solid red;
}
```

Зададим размеры и положение панели навигации для главной страницы:

```
#home nav {
    height:78px;
    top: 30vh;
    font-size: 1.5em;
 position: absolute;
}
```

Создание и оформление надписей на главной странице. В HTML-коде напишем текст главной, который будет отображаться поверх изображения. В CSS зададим его шрифт и отступы (см. лабораторную работу № 2).

Пример HTML (рис. 10.2):

```
<header>
  <h1>Site name</h1>
  <h3>Site description</h3>
</header>
```

Пример CSS (рис. 10.2):

```
#home header h1 {
    width: 100%;
    position: absolute;
    font-size: 15vh;
    text-align: center;
}
#home header h3 {
    width: 80%;
    margin: 0 auto;
    position: absolute;
    top: 50vh;
    text-align: right;
}
```

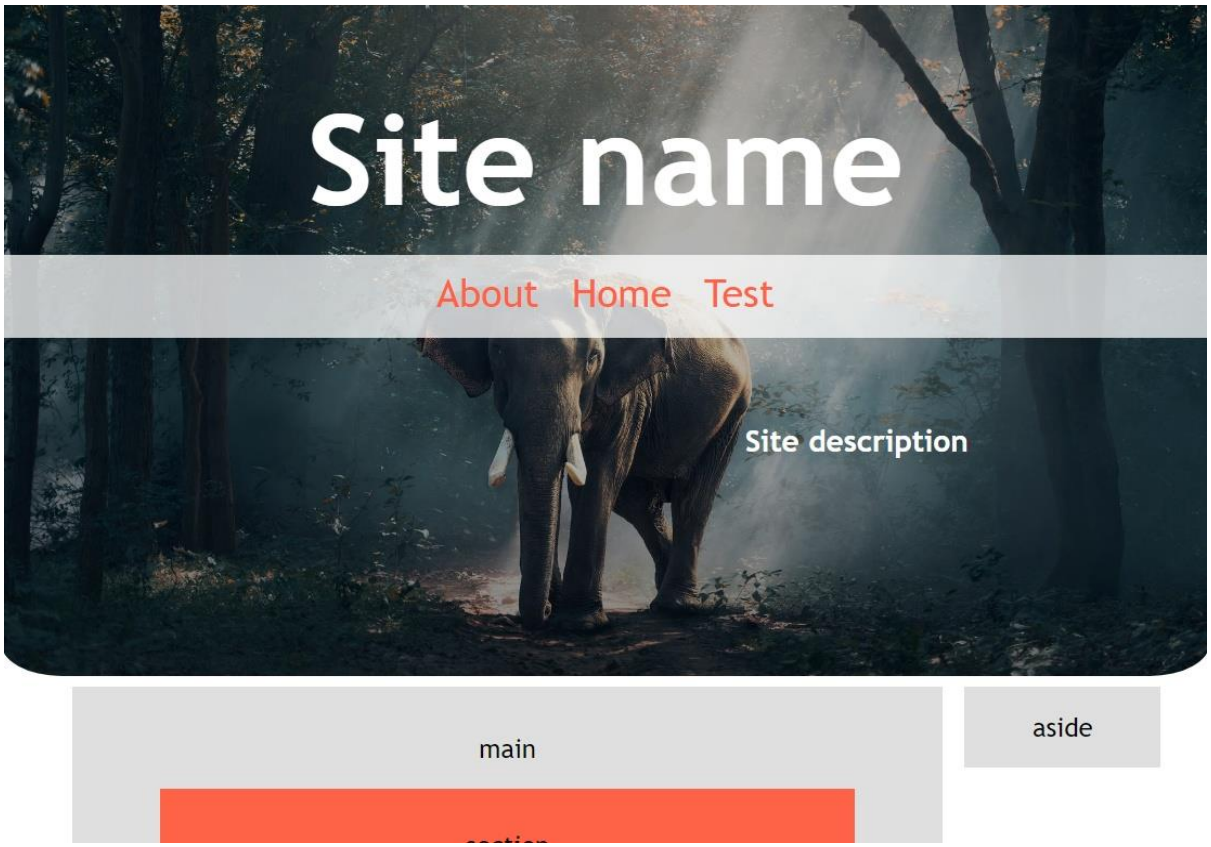


Рис. 10.2. Промежуточный вид главной страницы

Смена изображений в случайном порядке. При каждом обновлении страницы изображение изменяется. Показывается одно из изображений в директории `wall` (задаётся с помощью скрипта JavaScript). Создание меняющегося изображения на сайте (рис. 10.3) включает следующие этапы:

1. Создать файл скрипта `random.js` и указать его в теге `<head></head>`:

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

2. В файле скрипта создать массив. В массив занести ссылки на изображения фона в таком виде: `url("wall/elephant.jpg")`:

```
var img = [];
img[0] = 'url("wall/elephant.jpg")';
img[1] = 'url("wall/nature.jpg")';
img[2] = 'url("wall/sun.jpg")';
img[3] = 'url("wall/fox.jpg")';
```

3. Создать функцию `changeBG()`.

4. Внутри функции получить случайное число. `img.length` – это длина массива с изображениями:

```
var i = Math.floor(Math.random()*img.length);
```

5. Создать переменную, куда поместить тег `header`. Сделать это с помощью метода `getElementsByTagName("header");`.

```
var elem = document.getElementsByTagName("header");
```

6. Взять нулевой элемент из этого массива и присвоить его стилю фонового изображения случайный элемент из массива:

```
elem[0].style.backgroundImage=img[i];
```

Полный код функции:

```
var img = [];
img[0] = "url('wall/elephant.jpg')";
img[1] = "url('wall/nature.jpg')";
img[2] = "url('wall/sun.jpg')";
img[3] = "url('wall/fox.jpg')";

function changeBG() {
var i = Math.floor(Math.random()*img.length);
        var elem = document.getEle-
mentsByTagName("header");
elem[0].style.backgroundImage=img[i];
}
```

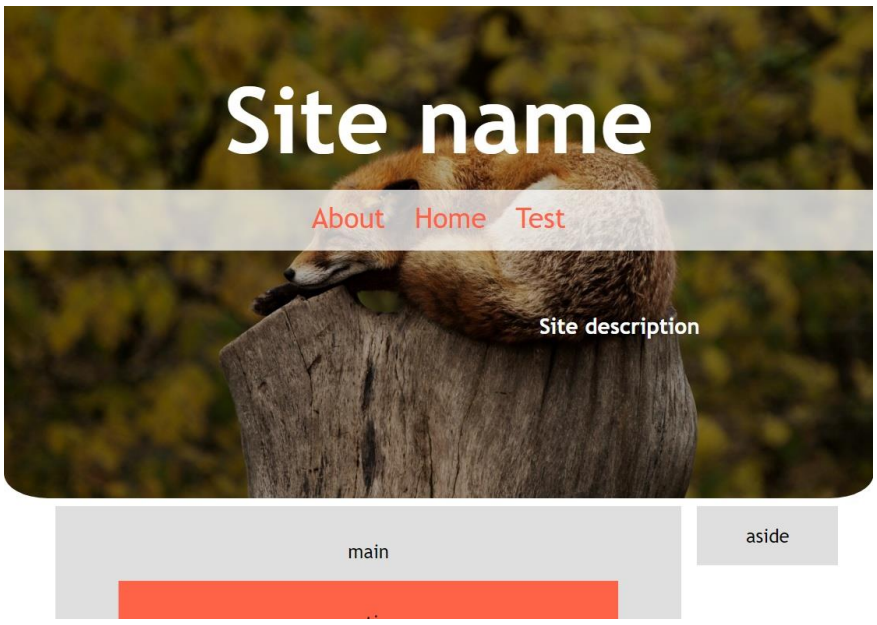


Рис. 10.3. Главная страница со случайным фоном

Страницы «О сайте» и «Тест». Главную страницу необходимо скопировать два раза. Первый файл назовем `about.html`, второй – `test.html`.

Большое изображение следует сделать маленьким, для меню. Из кода HTML удалим блоки с `id=”home”` и с `id=”container”`, а их содержимое – оставим.

В CSS укажем отступ от верхнего края, чтобы меню не закрывало содержимое веб-страницы (рис. 10.4).

```
body{
    width: 90%;
    margin: 58px auto;
    text-align:center;
    font: 24px Consolas bold;
    font-family: Trebuchet MS;
}
```

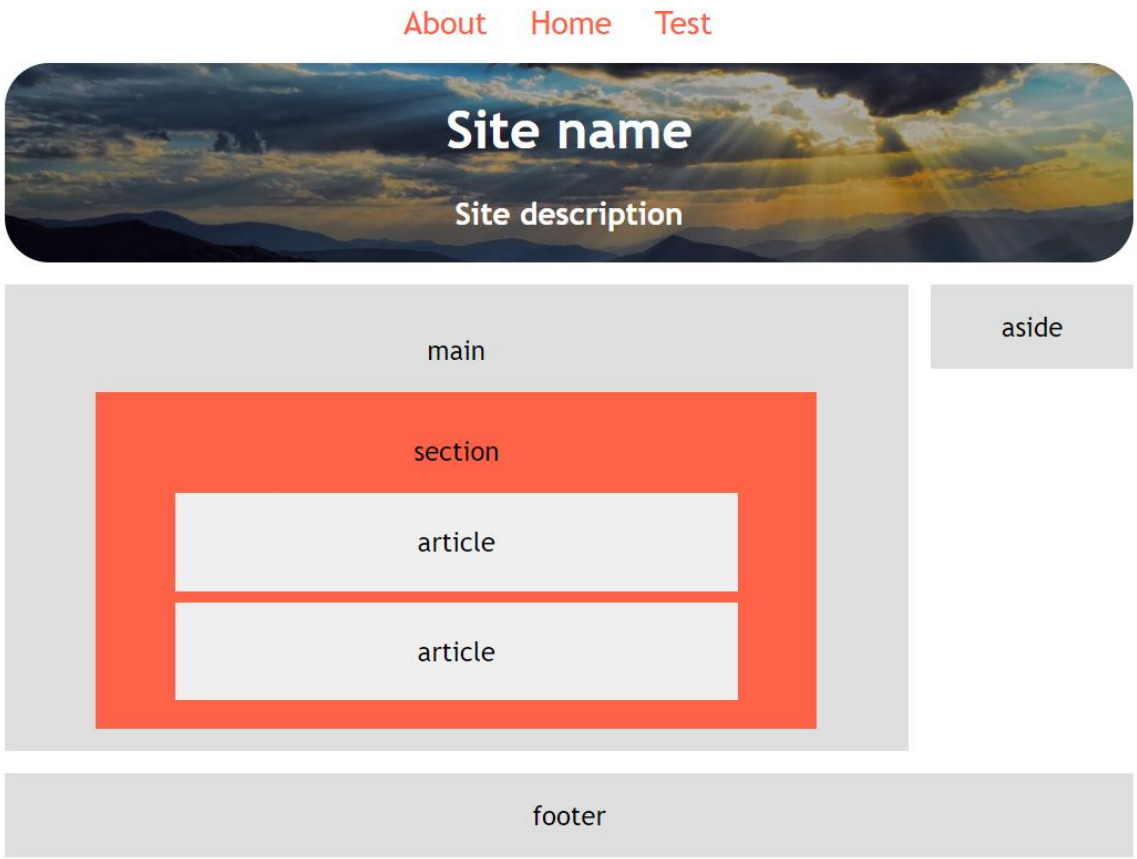


Рис. 10.4. Вид страницы `About` и `Test`

Теперь необходимо изменить основной фон страницы. Для этого используем бесшовную текстуру (рис. 10.5).

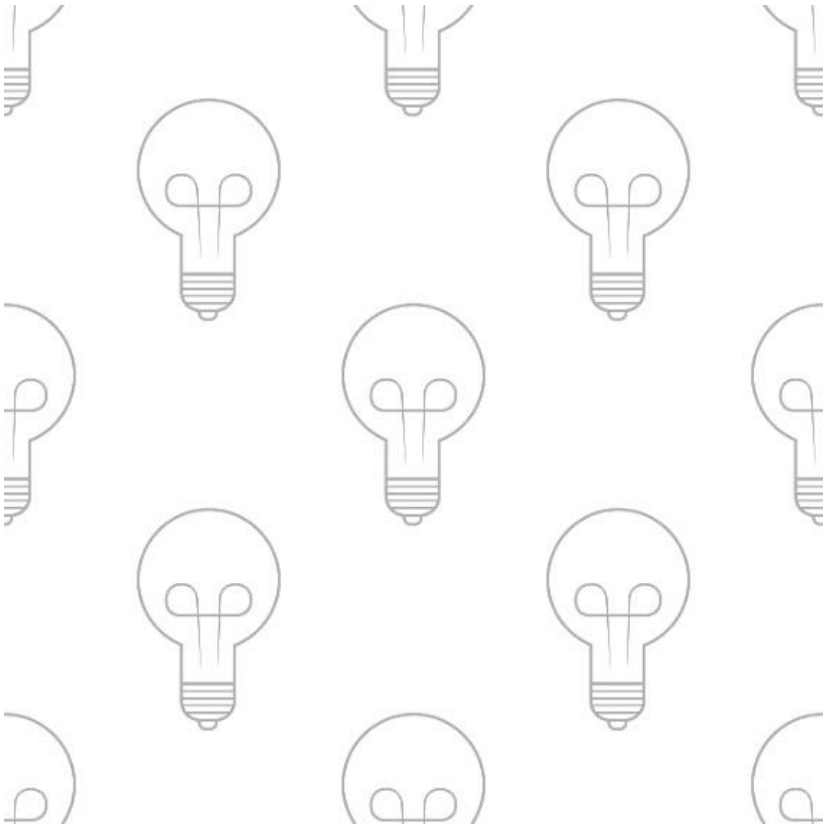


Рис. 10.5. Бесшовная текстура для фона

Универсальное свойство `background` позволяет установить одновременно до пяти характеристик фона. Следует установить все характеристики фона, являющиеся подсвойствами свойства `background`. Если указывается `background-position` внутри `background` и позиция описывается двумя словами (напр., `center left`), то необходимо поставить `"/"` для разделения, иначе свойство может сработать некорректно.

Пример кода приведен ниже, результат показан на рис. 10.6.

```
html {
    background-image: url('bg.jpg');
    background-repeat: repeat;
    background-size: 40%;
}
```

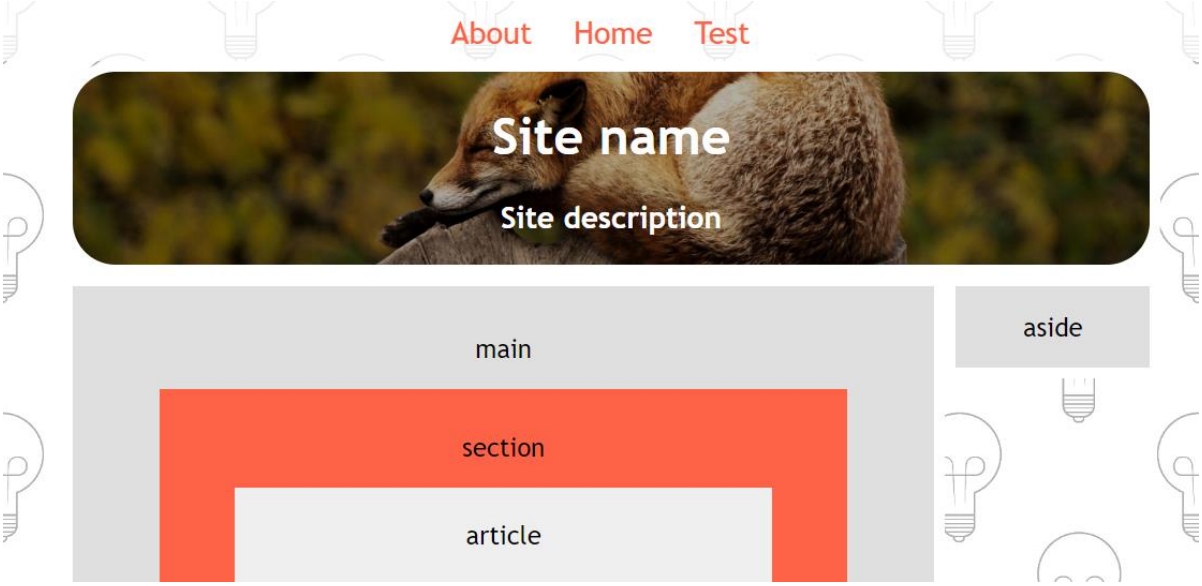


Рис. 10.6. Вид веб-страницы с фоном

Работа с созданным шаблоном. На предыдущих этапах был создан пустой сайт. Теперь необходимо наполнить его информацией.

На страницу «Тест» встроим тест, созданный в лабораторной работе № 9. HTML-форму теста нужно поместить в блок `section`. Блоки `article` в данном случае не нужны.

```
    <main>
        <p>Here is the test</p>
        <section>
            <p>Test</p>
            <form id="test">
        <fieldset style="display:inline-block">
            <legend>First question</legend>
            <p><label><input type="radio" name="coun-
ter"> Answer 1</label></p>
            <p><label><input type="radio" name="coun-
ter" data-true="1"> Answer 2</label></p>
            <p><label><input type="radio" name="coun-
ter"> Answer 3</label></p>
        </fieldset>
        <fieldset style="display:inline-block">
            <legend>Second question</legend>
```

```
                <p><label><input       type="checkbox"
name="check"  data-true="1"> Answer 1</label></p>
                <p><label><input       type="checkbox"
name="check"> Answer 2</label></p>
                <p><label><input       type="checkbox"
name="check"  data-true="1"> Answer 3</label></p>
        </fieldset>
            <p><label>Text   question   </label><input
type="text"    name="text1"    data-true="2"    data-
ans="happy"></p>
        </form>
    <button onClick="score();">Get the score!</button>
            </section>
        </main>
```

Для того чтобы работала проверка теста, в теге <head>…</head> обязательно следует подключить ссылку на скрипт проверки.

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

Пример готовой страницы с тестом представлен на рис. 10.7.

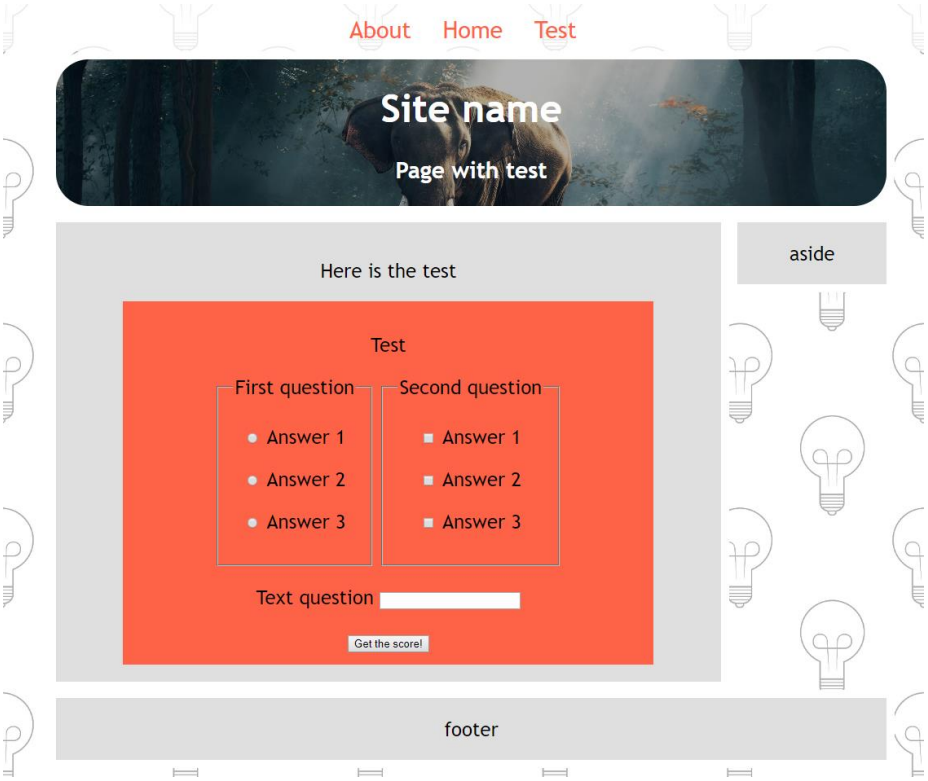


Рис. 10.7. Страница с тестом

Сайт можно продолжить наполнять и изменять контент после того, как разработан дизайн всех страниц. Вместо слова `footer` поместим год создания сайта и имя автора, вместо слова `aside` – рекламные изображения.

Полный код сайта

Файл style.css

```
html {
    background-image: url('bg.jpg');
    background-repeat: repeat;
    background-size: 40%;
}
body{
    width: 90%;
    margin: 58px auto;
    text-align:center;
    font: 24px Consolas bold;
    font-family: Trebuchet MS;
}
header {
    color: white;
    width:100%;
    background: #dfdfdf;
    border: 10px white solid;
    border-radius: 50px 50px 50px 50px;

    background-size: cover;
    background-position: center center;
    background-blend-mode: soft-light;
    background-color: rgba(0,0,0,0.3);
}
#home nav {
    height:78px;
    top: 30vh;
    font-size: 1.5em;
    position: absolute;
}
```

```
    nav {
        width:100%;
        top: 0;
        font-size: 1.2em;
        left:0;
        height:58px;
        background-color: rgba(255, 255, 255, 0.75);
        border: none;
        position: fixed;
        text-align: center;
    }
    nav ul {
        list-style-type: none;
        margin: 0;
        padding-left: 0;
        text-align: center;
    }
    nav ul li {
        display: inline-block; /*show in one line*/
        margin: 5px; /*margin between elements*/
        padding: 10px; /*padding inside element*/
        transition: all 0.5s ease-in-out; /*ainmation*/
        border-bottom: 0px solid red;
    }
    nav a {
        text-decoration:  none;  /*remove  link  under-
line*/
        color: tomato;
        transition: all 0.5s ease-in-out; /*ainmation*/
    }
    nav a:hover {/*link under pointer*/
        color: red;
    }
    nav ul li:hover { /*element under pointer*/
        border-bottom: 5px solid red;
    }
```

```
main {
    width: 76%;
    background: #dfdfdf;
    display: inline-block;
    border: 10px white solid;
    padding: 2%;
}
aside {
    width:18%;
    height: 30%;
    display: inline-block;
    background: #dfdfdf;
    border: 10px white solid;
    margin-right: -20px;
    float: right;
}
footer {
    width:100%;
    height:10%;
    background: #dfdfdf;
    float: left;
    border: 10px white solid;
}
section {
    width: 80%;
    margin: 0 auto;
    background: tomato;
    padding: 2%;
}
article {
    width: 80%;
    padding: 1%;
    background: #efefef;
    margin: 10px auto;
}
```

```
    #home header {
        height: 80vh;
        width: 100%;
        position: absolute;
        top:0;
        left:0;
        border: none;
        border-radius: 0 0 50px 50px;
    }
    #container {
        margin-top: 80vh;
    }
    #home header h1 {
        width: 100%;
        position: absolute;
        font-size: 15vh;
        text-align: center;
    }
    #home header h3 {
        width: 80%;
        margin: 0 auto;
        position: absolute;
        top: 50vh;
        text-align: right;
    }
```

Файл index.html

```
    <!doctype html>
    <html>
    <head>
        <meta charset="utf-8">
        <title>Home</title>
        <link     rel="stylesheet"     type="text/css"
href="style.css">
        <script src="random.js"></script>
    </head>
```

```
    <body>
    <script type="text/javascript">
        window.onload=changeBG;
    </script>
        <div id="home">
            <header><h1>Site   name</h1><h3>Site   de-
scription</h3></header>
            <nav>
              <ul>
               <li><a href="about.html">About</a></li>
               <li><a href="index.html">Home</a></li>
               <li><a href="test.html">Test</a></li>
              </ul>
          </nav>
        </div>
        <div id="container">
        <main>
            <p>main</p>
            <section>
                <p>section</p>
                <article>
                    <p>article</p>
                </article>
                <article>
                    <p>article</p>
                </article>
            </section>
        </main>
        <aside><p>aside</p></aside>
        <footer><p>footer</p></footer>
        </div>
    </body>
    </html>
```

Файл about.html

```
    <!doctype html>
    <html>
```

```
    <head>
        <meta charset="utf-8">
        <title>About</title>
        <link     rel="stylesheet"     type="text/css"
href="style.css">
        <script src="random.js"></script>
    </head>

    <body>
    <script type="text/javascript">
        window.onload=changeBG;
    </script>
        <header><h1>Site    name</h1><h3>About    this
site</h3></header>
            <nav>
              <ul>
               <li><a href="about.html">About</a></li>
               <li><a href="index.html">Home</a></li>
               <li><a href="test.html">Test</a></li>
              </ul>
          </nav>
        <main>
            <p>main</p>
            <section>
                <p>section</p>
                <article>
                    <p>article</p>
                </article>
                <article>
                    <p>article</p>
                </article>
            </section>
        </main>
        <aside><p>aside</p></aside>
        <footer><p>footer</p></footer>
    </body>
    </html>
```

Файл test.html

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Test</title>
    <link     rel="stylesheet"     type="text/css"
href="style.css">
    <script src="random.js"></script>
    <script src="score.js"></script>
</head>

<body>
<script type="text/javascript">
    window.onload=changeBG;
</script>
    <header><h1>Site     name</h1><h3>Page     with
test</h3></header>
        <nav>
          <ul>
           <li><a href="about.html">About</a></li>
           <li><a href="index.html">Home</a></li>
           <li><a href="test.html">Test</a></li>
          </ul>
      </nav>
    <main>
        <p>Here is the test</p>
        <section>
            <p>Test</p>
            <form id="test">
        <fieldset style="display:inline-block">
            <legend>First question</legend>
            <p><label><input          type="radio"
name="counter"> Answer 1</label></p>
            <p><label><input          type="radio"
name="counter" data-true="1"> Answer 2</label></p>
```

```
                <p><label><input          type="radio"
name="counter"> Answer 3</label></p>
            </fieldset>
            <fieldset style="display:inline-block">
                <legend>Second question</legend>
                <p><label><input       type="checkbox"
name="check"  data-true="1"> Answer 1</label></p>
                <p><label><input       type="checkbox"
name="check"> Answer 2</label></p>
                <p><label><input       type="checkbox"
name="check"  data-true="1"> Answer 3</label></p>
            </fieldset>
            <p><label>Text   question   </label><input
type="text"    name="text1"    data-true="2"    data-
ans="happy"></p>
        </form>
    <button onClick="score();">Get the score!</button>
            </section>
        </main>
        <aside><p>aside</p></aside>
        <footer><p>footer</p></footer>
    </body>
    </html>
```

Задание

Создайте сайт по примеру, приведенному в лабораторной работе. На веб-страницах разместите выполненные задания из предыдущих лабораторных работ.

Контрольные вопросы

1. Что на сайте отвечает за смену фонового изображения при обновлении страницы?
2. Как сделать большое изображение на главной странице?
3. Как сделать фоновое изображение из бесшовной текстуры?

ЗАКЛЮЧЕНИЕ

В практикуме рассмотрены технологии представления информации в Интернете: HTML, CSS, JavaScript. С их помощью информация, которую необходимо донести до пользователя, представляется в специальном структурном виде, понятном браузеру; оформляется по современным требованиям дизайна; может иметь логику для решения простых задач пользователя.

В рамках лабораторных работ было создано несколько веб-страниц, на которых представлены: блоки для размещения текста разного назначения, ссылки для связи веб-страниц между собой, панели навигации, изображения, формы для ввода и обработки информации, технологии использования анимации, замена содержимого элемента при обновлении страницы.

В практикуме рассмотрено создание статических веб-страниц, не требующих баз данных и работы с сервером.

Для закрепления изученного материала студентами даны задания по пройденной теме, а также контрольные вопросы.

Работая с практикумом, студент получает базовые знания HTML, CSS и JavaScript, учится применять и использовать их для создания полноценного сайта со статическими веб-страницами.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. *Никсон, Р.* Создаем динамические веб-сайты с помощью PHP, MySQL, Javascript и CSS / Р. Никсон. – СПб. : Питер, 2013. – 560 с. – ISBN 978-5-496-00187-8.

2. *Флэнаган, Д.* Javascript. Подробное руководство / Д. Флэнаган. – СПб. : Символ-Плюс, 2013. – 1080 с. – ISBN 978-5-93286-215-5. – ISBN 978-0-596-80552-4.

3. *Фрейен, Б.* Responsive Web Design with HTML5 and CSS3 / Б. Фрейен. – СПб. : Питер, 2014. – 304 с. – ISBN 978-5-496-00185-4.

4. *Пилгрим, М.* Погружение в HTML5 : пер. с англ. / М. Пилгрим. – СПб. : БХВ-Петербург, 2011. – 304 с. – ISBN 978-5-9775-0688-5.

5. *Бейтс, М.* CoffeeScript. Второе дыхание Javascript / М. Бейтс. – М. : ДМК-Пресс, 2016. – 310 с. – ISBN 978-5-97060-240-9.

6. *Мальцев, А.* ИТ Шеф: JavaScript – методы alert, prompt и confirm [Электронный ресурс] / А. Мальцев. – Режим доступа: https://itchief.ru/lessons/javascript/javascript-methods-alert-prompt-confirm (дата обращения: 12.07.2019).

7. *Назарова, Е.* HTML5BOOK [Электронный ресурс] / Е. Назарова. – Режим доступа: https://html5book.ru/ (дата обращения: 13.07.2019).

8. *Кантор, И.* Язык JavaScript [Электронный ресурс] / И. Кантор. – Режим доступа: https://learn.javascript.ru/ (дата обращения: 13.07.2019).

ОГЛАВЛЕНИЕ

Учебное издание

ОЗЕРОВА Марина Игоревна

ОСНОВЫ ИНФОРМАЦИОННОГО ДИЗАЙНА

Практикум

Редактор Е. А. Лебедева
Технический редактор Е. В. Невская
Корректор Н. В. Пустовойтова
Компьютерная верстка Л. В. Макаровой
Выпускающий редактор А. А. Амирсейидова

Подписано в печать 28.12.20.
Формат 60×84/16. Усл. печ. л. 8,14. Тираж 50 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.