Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Тульский государственный университет»

Кафедра «Информационная безопасность»

СБОРНИК МЕТОДИЧЕСКИХ УКАЗАНИЙ К ЛАБОРАТОНЫМ РАБОТАМ

по дисциплине

ИНТЕРНЕТ-ПРОГРАММИРОВАНИЕ

Направления подготовки:

09.03.04 Программная инженерия

Профиль «Мобильные и веб-приложения»

Квалификация (степень) выпускника: бакалавр

Форма обучения: *очная*

Тула 2023 г.

ЛИСТ СОГЛАСОВАНИЯ
методических указаний по лабораторным работам

Разработчик(и):

Сафронова М.А., доцент ИПМКН, доцент, канд. техн. наук 
(ФИО, должность, ученая степень, ученое звание) *(подпись)*

Лабораторная работа № 1 (часть 2)
Знакомство с HTML

Цель: изучить структуру HTML документа; принципы создания списков, таблиц, гиперссылок; особенности применения тегов и .

Краткая теория

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

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

Для того, чтобы создать HTML документ необходимо:

- открыть любой текстовый редактор (например, блокнот встроенный в Windows);
- набрать произвольный текст и разметить его HTML тэгами;
- сохранить файл с расширением .htm или .html.

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

Структура HTML-документа:

<HTML>

<HEAD>

<TITLE>простое название</TITLE>

</HEAD>

<BODY>

Содержание страницы

</BODY>

</HTML>

Параметры документа задаются через тег <BODY>. Параметры – это специальные символьные команды, которые «понимает» браузер. Тег BODY имеет следующие атрибуты:

- bgcolor – цвет фона задан по схеме RGB;
- background – фон – графический файл;
- text – цвет текста;
- link – цвет ссылки;
- alink – цвет активной ссылки;
- vlink – цвет посещенной ссылки.

Шрифт задается тегом FONT, который имеет следующие атрибуты: face – имена шрифтов, разделенные запятыми, size – размер от 1 до 7 (по умолчанию 3), color – цвет шрифта.

В HTML существуют специальные теги для заголовков: от H1 (самого крупного) до H6 (самого мелкого).

Списки на Web-страницах бывают маркированные (<UL>– начало текста списка, <LI> – начало каждого элемента в списке) и нумерованные (<OL> – начало текста списка, <LI> – начало каждого элемента в списке). Маркером списка может быть рисунок.

Атрибут TYPE позволяет изменить вид маркера. Значения атрибута:

- circle – ○;
- disk – ●;
- square – ■;
- A – A, B, C;
- a – a, b, c;
- I– I, II, III;
- i – i, ii, iii;
- 1 – 1, 2, 3.

Таблицы в HTML.

Элемент <table>служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов<tr>и<td>. Для отображения границ таблицы используется атрибут border. С помощью тэга <th> можно создать табличный заголовок. Текст элемента th центрируется и выделяется жирным шрифтом. С помощью атрибута colspan можно указать на сколько столбцов должна быть растянута указанная ячейка, а с помощью атрибута rowspan – на сколько строк должна быть растянута указанная ячейка.

Ссылки на Web-страницах.

Ссылки являются основой гипертекстовых документов и позволяют переходить с одной веб-страницы на другую. Реализуются с помощью тега A. Общий синтаксис создания ссылок следующий:

текст ссылки

Атрибут hrefопределяет URL адрес документа, на который следует перейти (например"cat.html"), а содержимое контейнера<a>является ссылкой.

Вставка изображений.

Для добавления в HTML документ изображений используется тэг <img>. Атрибут src тэга <img> должен содержать адрес, по которому находится картинка, которая должна быть отображена. Например,

.

Тег .

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

Тег .

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

Задания к лабораторной работе № 1 (часть 2)

Задание 1. Создать HTML-документ (заголовок Моя первая страница) со следующим содержимым:

Заголовок в центре страницы (теги h1) «Я, ФИО, HTML». Аббревиатуру HTML выделить другим цветом.

Три абзаца текста (можно своя автобиография) по 3–4 строки каждый абзац. Для каждого абзаца свой шрифт, размер, цвет (тег font). Для каждого абзаца придумать заголовок (h3), а также заголовок для всего текста (h2). Внутри каждого абзаца выделить несколько слов элементами span.

Список примерно такого вида с любым содержимым

1. Элемент 1
    - Элемент 1.1
    - Элемент 1.2

2. Элемент 2

- Элемент 2.1
- Элемент 2.2
- Элемент 2.3

3. Элемент 3

- Элемент 3.1
- Элемент 3.2

Перед списком вставить заголовок, отражающий его содержимое (**h2**).

Таблица, содержащая результаты сессии. Перед таблицей вставить заголовок (**h2**).

ФИО	Иванов Иван Иванович
Свое фото/Аватарка	Адрес: ул.Ленина, д.5, ком.433

	Дата	Предмет	Оценка
группа	31.12.2022	Информатка	6
	5.01.2022	Физика	9
	11.01.2022	ОИТ	4
	16.01.2022	Математика	6
	22.01.2023	Иностранный язык	8

Задание 2. Создать еще один HTML-документ (заголовок Моя вторая страница) со следующим содержимым:

Два элемента с произвольным текстом. Перед каждым элементом заголовок (**h3**).

Любой рисунок. Перед рисунком оформить заголовок (**h2**).

Элемент , содержащий авторский знак © и ФИО автора.

Задание 3. ФИО в заголовке на первой web-странице и в таблице сделать ссылкой на вторую страницу, а рисунок на второй странице – ссылкой на первую.

Содержание отчета

1. Титульный лист (включая названия дисциплины, номер лабораторной работы)

2. Цель лабораторной работы

3. Формулировка заданий, текст программы по заданиям (с комментариями), скрин экрана результатов выполнения заданий.

Лабораторная работа № 2 (часть 1)
Гиперссылки

Цель: изучить использование различных гиперссылок в HTML документе.

Краткая теория

Ссылки являются основой гипертекстовых документов и позволяют переходить с одной web-страницы (*или закладки в документе*) на другую, а также выполнять действия, связанные с некоторыми прикладными протоколами Web (например, вызывать почтовый клиент по умолчанию, менеджер закачек и т.п.) Ссылка может вести не только на HTML-файл, но и на файл любого другого типа, причем этот файл может размещаться на другом сайте. Как конкретно будет обрабатываться тот или иной тип файлов, зависит от настроек операционной системы и браузера.

Что такое ссылка

Типичная ссылка представляет собой участок текста, щёлкая на который, вы переходите на другую страницу, открываете изображение, начинаете скачивать файл или перемещаетесь к какому-то месту на текущей странице.
Ссылки создаются с помощью тега <a>. Полный список атрибутов тега ссылки можно посмотреть здесь: http://htmlbook.ru/html/a
Основной атрибут - **href** , он определяет URL для перехода по ссылке. Текст, расположенный между тегами <a> и , по умолчанию становится синего цвета и подчеркивается.
Например:

```
<a href="http://htmlbook.ru">HTML Book</a>
```

Тег <a> можно использовать вообще без адреса, то есть без атрибута `href`. Такой тег обозначает "ссылку-заглушку", которая в других условиях может стать обычной ссылкой. Часто ссылки-заглушки используют, чтобы показать, что мы находимся на текущей странице:

```
<ul>
  <li><a>1 страница</a></li>
  <li><a href="2">2 страница</a></li>
  <li><a href="3">3 страница</a></li>
</ul>
```

Когда мы удаляем атрибут `href` у ссылки, то лучше оставить подсказку о том, почему мы это сделали. Подсказку можно добавить с помощью атрибута `title`. Подсказка появится, когда курсор задержится над ссылкой некоторое время. Пример:

```
<a href="http://sibstrin.ru" title="НГАГУ">университет</a>
```

Относительные ссылки

Над сайтом мы в основном работаем на своём компьютере, то есть *локально*. Ссылка на другую страницу сайта должна открывать файл с нашего компьютера, а не откуда-то из интернета. Для таких ссылок нужно использовать относительный путь к файлу, например:

```
<a href="inner.html">Введение</a>
```

В относительном пути **нет** адреса сайта! Чтобы перейти по относительному адресу, браузер должен его "расшифровать". Для этого он обычно использует положение текущей страницы. Например, в папке `c:/blog` есть два файла:

```
c:/blog/
      |-index.html // в браузере открыта эта страница
      |-inner.html
```

В браузере открыта страница `c:/blog/index.html`, и в ней есть ссылка с относительным адресом `inner.html`. Чтобы перейти по этой ссылке, браузер смотрит на расположение открытой страницы и заменяет в нём последнюю часть:

```
c:/blog/ + inner.html // заменяем последнюю часть
c:/blog/inner.html                // открываем этот файл
```

Удалено: index.html

Относительные адреса работают не только для файлов на компьютере, но и для страниц в сети. Если выложить два файла из примера в интернет (не меняя их взаимное расположение), то ссылка всё равно будет работать. Относительный адрес `inner.html` на странице `https://site.ru/blog/index.html` расшифруется так:

```
https://site.ru/blog/ + inner.html
  // заменяем последнюю часть
https://site.ru/blog/inner.html
  // открываем этот адрес
```

Удалено: index.html

Обратите внимание, что в относительных адресах также можно использовать имена вложенных папок, ссылку на родительскую папку `..` (два символа точки), ссылку на текущую папку `.` (один символ точки) и т.п., в общем, все виды путей к файлу из курса Информатики. При составлении путей к файлам нужно использовать прямой слэш "/", а не обратный "\", как в папках Windows.

Абсолютные ссылки

Относительные адреса отлично подходят, если вы делаете сайт на своём компьютере или создаёте навигацию по страницам *своего* же сайта. Но если нужно сделать ссылку на *другой* сайт в интернете, то необходимо использовать уже "обычный" адрес.
Этот "обычный" или полный адрес называется *абсолютным*. Выглядит он, например, так:
`https://site.ru/blog/index.html`
Абсолютные адреса содержат минимум три части: протокол, имя сервера и путь.

```
https:           — протокол
//site.ru        — имя сервера
/blog/index.html — путь от корневой папки сервера к файлу
```

Если в адресе нет имени сервера или протокола, то это относительный адрес:

```
https://site.ru/blog/index.html - абсолютный адрес
               /blog/index.html - относительный адрес
                     index.html - относительный адрес
```

Ссылки на файл

По ссылкам можно не только переходить, но и скачивать файлы. Для этого необходимо в атрибуте `href` прописать ссылку на этот файл. А для того чтобы предотвратить открытие файлов прямо в браузере, у тега `<a>` существует атрибут `download`.

```
<a href="file.pdf" download>Браузер скачает меня, а не будет
читать</a>
```

Если существует стандартная обработка для типа файла, адресуемого ссылкой, браузер может игнорировать указание этого атрибута.

Внутренние ссылки (якоря)

Ссылка-якорь — это обычная ссылка, в адресе которой используется символ `#`, после которого следует идентификатор элемента. Идентификатор создаётся с помощью тега `<a name>` или атрибута `id` у того тега, к которому надо перейти при щелчке по ссылке.

```
<a href="#part1">Глава 1</a>
```

Ссылка-якорь используется для прокрутки к заданной части страницы, в том числе используется и в абсолютных адресах.

Пример. Создадим в конце web-страницы ссылку, которая введет в начало документа. Для этого в начале документа, задаем метку:

```
<body>
<a name="ttt"></a>
```

Затем в конце страницы задаем ссылку:

```
<a href="#ttt">В начало</a>
</body>
```

Тем не менее, этот классический способ устаревает, лучше использовать атрибут `id` элемента:

```
<a id="ttt"></a>
```

Вот так выглядит адрес, состоящий из одного якоря:

```
<a href="#part1">Глава 1</a>
```

При щелчке по такой ссылке браузер найдёт на странице элемент с соответствующим атрибутом `id` и прокрутит окно страницы к нему.

```
<section id="part1">Содержание первой главы</section>
```

Также можно сделать ссылку на метку другого документа. Пример:

```
<a href="text.html#ttt">Как написано на сайте…</a>
```

Атрибуты target и title

С тегом `<a>` можно использовать атрибуты `target` и `title`.

Атрибут `target` указывает окно, в котором нужно открыть ссылку. По умолчанию, при переходе по ссылке документ открывается в текущем окне. Синтаксис следующий:

```
<a target="имя окна">...</a>
```

В качестве значения используется имя окна или фрейма, заданное атрибутом `name`. Если установлено несуществующее имя, то будет открыто новое окно. В качестве зарезервированных имен применяются следующие:

- `_blank` — загружает страницу по ссылке в новое окно (вкладку) браузера.
- `_self` — загружает страницу по ссылке в текущее окно.
- `_parent` — загружает страницу во фрейм-родитель, если фреймов нет, то это значение работает как `_self` (*устарело*).
- `_top` — отменяет все фреймы и загружает страницу в полном окне браузера, если фреймов нет, то это значение работает как `_self` (*устарело*).

Ссылки на адрес электронной почты

Для создания **ссылки на адрес электронной почты** в атрибуте `href` надо указать `mailto:адрес электронной почты`. Например:

```
<a href="mailto:admin@site.ru">администратор</a>
```

Ссылка-картинка

Поместив внутрь элемента `<a>` элемент `<img>`, можно сделать ссылку-изображение:

```
<a href="index.html">
 <img src="image.png" alt="HTML tutorial">
</a>
```

Задание к лабораторной работе № 2 (часть 1)

Создать шаблон страницы сайта, включающий его логические части и подготовить несколько страниц, связанных ссылками, например:

- Главная
- О себе
- Мои работы (для последующей публикации заданий курса)
- Файлы (с возможностью скачать какие-либо файлы на странице)
- Контакты (контактная информация, включающая адрес E-mail)
- Ссылки (страница с внешними ссылками)

В коде использовать разметку HTML5 и относительные ссылки, ссылки-якоря, ссылки на адрес E-mail.

Проверить работу сайта и его валидность.

Содержание отчета

1. Титульный лист (включая названия дисциплины, номер лабораторной работы)

2. Цель лабораторной работы

3. Формулировка заданий, текст программы по заданиям (с комментариями), скрин экрана результатов выполнения заданий.

Лабораторная работа № 3.
Знакомство с html5. Элементы video и audio. Геолокация. Элемент canvas.

Цель: научиться подключать на веб-страницу аудио и видео файлы, определять местоположение пользователя; изучить возможности рисования на веб-странице.

Теория

HTML5 – это новая версия HTML. Наиболее интересные нововведения HTML5:

- поддержка видео и аудио (элементы video и audio);
- возможности рисования на веб-страницах произвольных объектов (элемент canvas);
- улучшение форм (новые значения type для <input> и множество новых элементов и атрибутов);
- добавление семантических тэгов, позволяющих сделать веб-страницы более понятными для поисковых систем, браузеров и других программ и устройств, читающих веб-страницы (элементы footer, header, nav, article и др.);
- DOM хранилища – более функциональная альтернатива cookie.

HTML5 содержит специальный тэг **<video>**, позволяющий встраивать в веб-страницы видео файлы. Атрибут **src** тэга <video> позволяет указать путь к видео файлу, атрибут **controls** отображает в плеере кнопки управления видео, а атрибуты **height** (высота) и **width** (ширина) задают размеры плеера. Например,

<video src=video.ogv" width="300" height="200" controls="controls"></video>

С помощью нового HTML5 элемента **<audio>** можно добавить на веб-страницу музыкальную композицию. Атрибут **src** тэга <audio> позволяет указать путь к аудио файлу, а атрибут **controls** добавляет кнопки управления. Например,

<audio src="stop.ogv" controls="controls"> </audio>

С помощью HTML5 геолокации можно определить местоположение пользователя.

Текущая позиция пользователя может быть определена с помощью метода **getCurrentPosition()** объекта **navigator.geolocation**.

navigator.geolocation.getCurrentPosition(success_function, error_function, options);

Параметры метода:

success_function – имя функции, которая будет вызвана в случае удачного считывания координат;

error_function – имя функции, которая будет вызвана при ошибке;

options – задает настройки, которые будут использованы при считывании координат. Возможные значения:

enableHighAccuracy – если имеет значение true, браузер будет пытаться определить местоположение как можно точнее;

timeout – устанавливает максимально допустимое время для считывания данных (по умолчанию не ограничено);

maximumAge – как долго браузер будет хранить к кэше предыдущее сохраненное значение.

Если пользователь разрешил использовать данные о его местоположении и они были удачно считаны браузером, то в функцию **success_function** в качестве параметров будет передан объект, содержащий свойство **timestamp** (время считывания координат), и объект **coords** со следующими свойствами:

- **latitude** и **longitude** – широта и долгота местоположения;
- **altitude** – высота над уровнем моря в метрах;
- **accuracy** – точность определения широты и долготы (чем больше число, тем меньше точность);
- **altitudeAccuracy** – точность определения высоты (чем больше число, тем меньше точность);
- **heading** – направление пользователя в градусах (т.е. 0 градусов значит, что пользователь направляется на север, 180 на юг и т.д.)
- **speed** – скорость перемещения в метрах в секунду.

В следующем примере определяются широта и долгота местоположения пользователя:

```
<html>

<script>

function getCoordinates() {

navigator.geolocation.getCurrentPosition(showCoordinates);

}

function showCoordinates(position) {

document.write("Широта:" + position.coords.latitude + "<br/>");

document.write("Долгота:" + position.coords.longitude);

}

</script>

<body>

<p>Для считывания координат нажмите на кнопку</p>

<input type="button" value=" Координаты" onclick =
```

"getCoordinates()">

</body>

</html>

Элемент <canvas>

Элемент<canvas>позволяет рисовать на веб-страницах произвольные фигуры с помощью JavaScript (или других клиентских скриптов). Сам по себе canvas ничего не рисует, это холст, который предоставляет возможности для рисования.Создать canvas можно так:

<canvas id='draw' width='300' height='200'>…</canvas>.

Для создания прямоугольников:

fillRect(x,y,ширина,высота) – рисует закрашенный прямоугольник;

strokeRect(x,y,ширина,высота) – не закрашенный прямоугольник;

clearRect(x,y,ширина,высота) – очищает указанную зону (x и y – величина смещения прямоугольника от верхнего левого угла холста в пикселях).

В следующем примере на экран выводятся два прямоугольника: закрашенный с очищенной зоной внутри и не закрашенный. Первые две строки являются стандартными для рисования любого объекта в canvas.

var canvas = document.getElementById("draw");

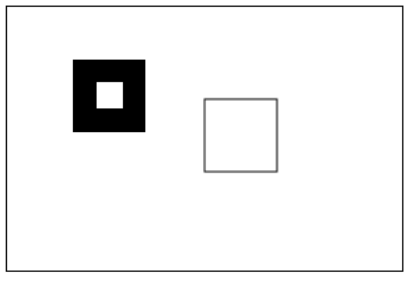
var x=canvas.getContext("2d");

x.fillRect(50,40,55,55);

x.strokeRect(150,70,55,55);

x.clearRect(68,57,20,20);

```
<html>
<body>
<canvas id='draw' width='300' height='200' style='border:1px solid'></canvas>
<script type='text/javascript'>
var canvas=document.getElementById("draw")
var x=canvas.getContext("2d");
x.fillRect(50,40,55,55);
x.strokeRect(150,70,55,55);
x.clearRect(68,57,20,20);
</script>
</body>
</html>
```



Рисование составных фигур

Составные фигуры состоят из нескольких соединенных простых объектов.

moveTo(x,y) – устанавливает координаты точки, из которой начнется рисование следующего объекта;

lineTo(x,y) – рисует прямую линию:

arc(x,y,радиус,нач_угол,конеч_угол) – рисует круг. Угол необходимо задавать в радианах, (радианы=(Math.PI/180)*градусы);

rect(x, y, ширина, высота) – рисует прямоугольник.

```
x.beginPath();

x.moveTo(20,20);

x.lineTo(70,70);

x.lineTo(20,70);

x.closePath();

x.fill();
```

Для раскрашивания нарисованных в canvas фигур предусмотрены свойства: **fillStyle** (применяется к закрашенным фигурам) и **strokeStyle** (к незакрашенным фигурам).

```
x.fillStyle="green";

x.strokeStyle="#FF45FF"

x.fillStyle="rgb(255,73,73)"

x.fillStyle="rgba(0,0,0,0.5)" //(0.5 – прозрачность)
```

Для оформления линий используются: **lineWidth** – ширина линии, **lineCap** – кончики линий (round – закругленные) , **lineJoin** – сглаживание стыков двух линий (round).

Градиент

CreateLinearGradient(x1,y1,x2,y2) – создать линейный градиент, x1 и y1 координаты начальной, x2 и y2 – конечной точки градиента. После создания градиента надо указать цвета перехода с помощью метода **addColorStop(точка,цвет).**

```
var grad =x.createLinearGradient(0,0,0,150);

grad.addColorStop(0.0,'black');

grad.addColorStop(1.0,'white');

x.fillStyle=grad;

x.fillRect(20,20,200,200);
```

Создание теней

shadowOffsetX – смещение тени от объекта по горизонтали (может быть отрицательным).

shadowOffsetY – смещение тени по вертикали (может быть отрицательным)

shadowBlur – величина размытия тени

shadowColor – цвет тени (по умолчанию черный).

```
x.shadowOffsetX=3;

x.shadowOffsetY=3;

x.shadowBlur=5;

x.shadowColor='black';

x.fillStyle='#ffaa00';

x.fillRect(50,40,55,55);
```

Текст

Метод **fillText("текст",x,y)**позволяет отображать в элементе canvas произвольный текст, который может быть оформлен с помощью свойствafont.

```
x.font='15px Verdana';

x.fillStyle='#60016d';

x.fillText("Я студент БГТУ", 10, 40);
```

Вставка изображений

В canvas могут быть вставлены изображения форматов PNG, GIF и JPEG с помощью метода **drawImage('ссылка на картинку',x,y)**

```
<img id='image1' src='ris.jpg' style='display:none;'/>

<canvas id='draw' width='400' height='300' style='border:1px solid'></canvas>

<script>

Var img=document.getElementById('image1');

var canvas=document.getElementById("draw")

var x=canvas.getContext("2d");

x.drawImage(img,10,10);

</script>
```

Задания для лабораторной работы № 3

Задание 1. Создать новую веб-страницу. Вставит заголовок в центре страницы «Я, ФИО, изучаю HTML5». Подсветить ФИО с помощью тега <mark>.

Задание 2. Вставить аудио и видео файлы.

Задание 3. Определить свои координаты с помощью геолокации.

Задание 4. Нарисовать любую картинку (можно домик), используя простые фигуры. Применить возможности для оформления – цвет, тени, градиент. Созданную картинку подписать (свои ФИО).

Лабораторная работа № 4.
Использование новых элементов и атрибутов в html5 формах

Цель: научиться использовать новые элементы и атрибуты в HTML5 формах; изучить особенности применения на странице семантических тэгов.

Теория

Новые типы **input** в HTML5 формах:

input **type=email** – поле, содержащее email адрес (автоматически проверяется перед отправкой на сервер);

input **type=url** – поле, содержащее url адрес (автоматически проверяется перед отправкой на сервер);

input **type=tel** – поле для ввода телефонного номера. С помощью атрибута **pattern** можно установить формат принимаемого телефонного номера, который задается с помощью регулярных выражений;

input **type=number** – поле, которое должно содержать числа. Можно ограничивать диапазон принимаемых чисел с помощью атрибутов **min** (минимальное допустимое число) и **max** (максимальное допустимое число). С помощью атрибута **step** можно задать шаг допустимых чисел (к примеру, если шаг равен 2, то в поле могут вводиться числа 0, 2, 4,6 и т.д.);

input **type=range** – поле, которое может содержать значения в определенном интервале. Отображается как ползунок, который можно перетаскивать мышкой. Можно ограничивать диапазон принимаемых чисел с помощью атрибутов **min** и **max.** Атрибут **step** задает шаг допустимых чисел;

input **type=search** – поле поиска (может использоваться, например, для создания поиска по сайту);

input **type=color** – поле для определения цвета.

Новые элементы в HTML5 формах:

datalist позволяет привязать список к полям формы. Значения списка будут выводиться как поисковые подсказки во время ввода информации в поле, связанное с ним;

keygen позволяет генерировать открытые и закрытые ключи, которые используются для безопасной связи с сервером;

output может использоваться для вывода различной информации. С помощью атрибута **for** можно указать связанные поля.

Новые атрибуты в HTML5 формах:

autofocus делает поле активным после загрузки страницы (может использоваться со всеми типами input);

form указывает форму, которой принадлежит данное поле (может использоваться со всеми типами input);

multiple указывает, что данное поле может принимать несколько значений одновременно (может использоваться с input типов email и file);

novalidate указывает, что данное поле не должно проверяться перед отправкой (может использоваться с form и input);

placeholder отображает текст-подсказку в поле (может использоваться с input следующих типов: text, search, url, tel, email и password);

required указывает, что данное поле должно быть обязательно заполнено перед отправкой.

В HTML5 были добавлены новые элементы ввода, позволяющие удобно выбирать дату и время:

date – дата в формате год-месяц-день_месяца;

time – время;

datetime – дата в формате год-месяц-день_месяца-время (отсчет ведется по глобальному времени);

datetime-local – дата в формате год-месяц-день_месяца-время (отсчет ведется по местному времени);

month – дата в формате год-месяц;

week – дата в формате год-неделя.

Семантические тэги

В HTML5 введены семантические тэги, с помощью которых можно сделать страницы сайтов более понятными для поисковых систем и браузеров:

<footer> – футер;

<header> – заголовочный блок сайта;

<nav> – навигационное меню.

Тэг **<section>** позволяет группировать логически связанное содержимое в документе, тэг **<mark>** – выделить (подсветить) важную часть в тексте.

В HTML5 можно создавать подписи для иллюстраций с помощью тэгов: **<figure>** и **<figcaption>.** Например,

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

</figure>

Задания для лабораторной работы № 4

Задание 1. Открыть свою веб-страницу, созданную в лабораторной работе 3. Создать на странице форму с новыми элементами и атрибутами, перечисленными выше.

Задание 2. Оформить страницу с помощью семантических тэгов: footer, header, nav. Тег footer должен содержать данные автора (ФИО, курс, группа, телефон и email). Телефон и email взять из соответствующих элементов формы.

Задание 3. Тег nav должен содержать свою фотографию (можно взять с самой первой страницы) с соответствующей подписью.

Содержание отчета

1. Титульный лист (включая названия дисциплины, номер лабораторной работы)

2. Цель лабораторной работы

3. Формулировка заданий, текст программы по заданиям (с комментариями), скрин экрана результатов выполнения заданий.

Лабораторная работа № 5
Форма и обработка данных

Цель: научиться разрабатывать формы и обрабатывать данные в них

Теория

1. Простой пример формы

Формы в HTML представляют собой интерфейс с пользователем и основной способ для ввода и отправки данных. Все поля формы помещаются между тегами `<form>` и `</form>`. Например, создадим простейшую форму с текстовым полем ввода и кнопкой:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Формы в HTML5</title>
    </head>
    <body>
        <form method="post"
            action="http://localhost:8080/login.php">
            <input name="login"/>
            <input type="submit" value="Войти" />
        </form>
    </body>
</html>
```

Для настройки форм у элемента `form` определены следующие атрибуты:

- `method`: устанавливает метод отправки данных на сервер. Допустимы два значения: `post` и `get`. Значение `post` позволяет передать данные на web-сервер через заголовки протокола HTTP. А значение `get` позволяет передать данные через URL-адрес запроса.
- `action`: устанавливает адрес, на который передаются данные формы
- `enctype`: устанавливает тип передаваемых данных. Он свою очередь может принимать следующие значения:
    - `application/x-www-form-urlencoded`: кодировка отправляемых данных по умолчанию
    - `multipart/form-data`: эта кодировка применяется при отправке файлов
    - `text/plain`: эта кодировка применяется при отправке простой текстовой информации

В выше использованном примере у формы установлен метод "post", то есть все значения формы отправляются в теле запроса, а адресом служит строка *http://localhost:8080/login.php*.

Адрес здесь указан случайным образом. В реальности по указанному адресу работает web-сервер, который, используя одну из технологий серверной стороны (PHP, NodeJS, ASP.NET и т.д.), может получать запросы и возвращать ответ на них в виде разметки HTML или другого содержимого. В данном же случае мы не будем акцентировать внимание на технологиях серверной стороны, сосредоточимся лишь на тех средствах HTML, которые позволяют отправлять данные на сервер.

Часто web-браузеры запоминают вводимые данные, и при вводе браузеры могут выдавать список подсказок из ранее введенных слов:

Это может быть не всегда удобно, и с помощью атрибута `autocomplete` можно отключить **автодополнение**:

```
<form method="post" autocomplete="off"
            action="http://localhost:8080/login.php">
    <input name="login" />
    <input name="password" />
    <input type="submit" value="Войти" />
</form>
```

Если нам надо включить автодополнение только для каких-то определенных полей, то мы можем применить к ним атрибут `autocomplete="on"`:

```
<form method="post" autocomplete="off"
action="http://localhost:8080/login.php">
    <input name="login" />
    <input name="password" autocomplete="on" />
    <input type="submit" value="Войти" />
</form>
```

Теперь для всей формы, кроме второго поля, будет отключено автодополнение.

2. Элементы форм

Формы состоят из определённого количества элементов ввода. Все элементы ввода помещаются между тегами `<form>` и `</form>`

Наиболее распространенным элементом ввода является элемент `input`. Однако реальное действие этого элемента зависит от того, какое значение установлено у его атрибута `type`. А он может принимать следующие значения:

- `text`: обычное текстовое поле;
- `password`: тоже текстовое поле, только вместо вводимых символов отображаются звездочки, поэтому в основном используется для ввода пароля;
- `radio`: радиокнопка или переключатель. Из группы радиокнопок с одинаковым атрибутом `name` можно выбрать только одну;
- `checkbox`: элемент флажок, который может находиться в отмеченном или неотмеченном состоянии;
- `hidden`: скрытое поле;
- `submit`: кнопка отправки формы;
- `color`: поле для ввода цвета;
- `date`: поле для ввода даты;
- `datetime`: поле для ввода даты и времени с учетом часового пояса;
- `datetime-local`: поле для ввода даты и времени без учета часового пояса;
- `email`: поле для ввода адреса электронной почты;
- `month`: поле для ввода года и месяца;
- `number`: поле для ввода чисел;
- `range`: ползунок для выбора числа из некоторого диапазона;
- `tel`: поле для ввода телефона;
- `time`: поле для ввода времени;
- `week`: поле для ввода года и недели;
- `url`: поле для ввода адреса url;
- `file`: поле для выбора отправляемого файла;
- `image`: создает кнопку в виде картинки.

Кроме элемента `input` в различных модификациях есть еще небольшой набор элементов, которые также можно использовать на форме:

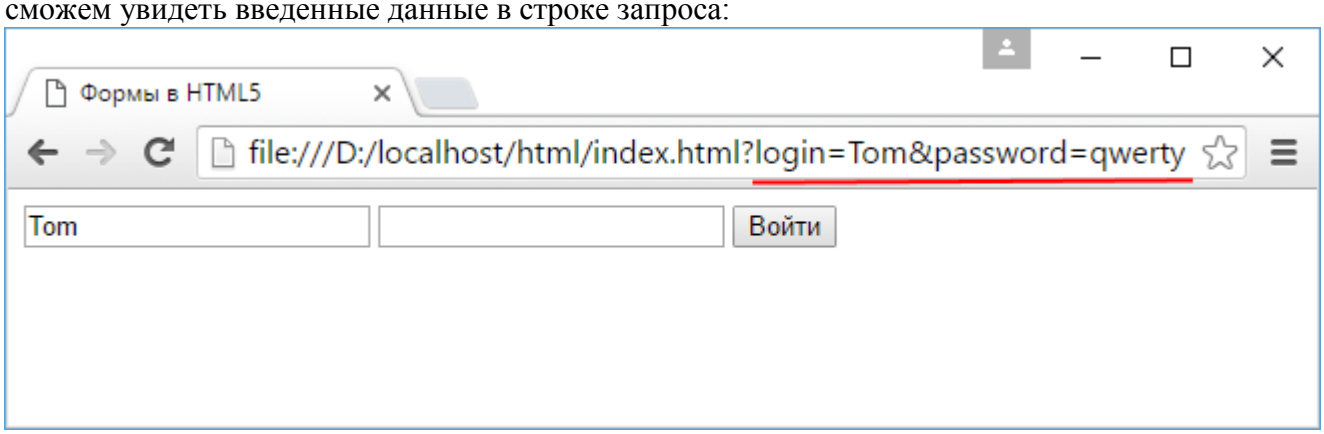
- `button`: создает кнопку;
- `select`: выпадающий список;
- `label`: создает метку, которая отображается рядом с полем ввода;
- `textarea`: многострочное текстовое поле.

У всех элементов ввода можно установить **атрибуты `name` и `value`**. Эти атрибуты имеют важное значение. По атрибуту `name` мы можем идентифицировать поле ввода, а атрибут `value` позволяет установить значение поля ввода. Например:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Формы в HTML5</title>
    </head>
    <body>
        <form method="get" action="index.html">
            <input type="text" name="login" value="Tom"/>
            <input type="password" name="password"/>
            <input type="submit" value="Войти" />
        </form>
    </body>
</html>
```

Здесь текстовое поле имеет значение "`Tom`" (как указано в атрибуте `value`), поэтому при загрузке web-страницы в этом поле мы увидим данный текст.

Поскольку методом отправки данных формы является метод "`get`", то данные будут отправляться через строку запроса. Так как нам в данном случае не важно, как данные будут приниматься, не важен сервер, который получает данные, поэтому в качестве адреса мы установили ту же самую страницу, то есть, файл `index.html`. И при отправке мы сможем увидеть введенные данные в строке запроса:



В строке запроса нас интересует следующий кусочек:

```
login=Tom&password=qwerty
```

При отправке формы браузер соединяет все данные в набор пар "ключ-значение". В нашем случае две таких пары: `login=Tom` и `password=qwerty`. Ключом в этих парах выступает название поля ввода, которое определяется атрибутом `name`, а значением - собственно то значение, которое введено в поле ввода (или значение атрибута `value`).

Получив эти данные, сервер легко может узнать, какие значения в какие поля ввода были введены пользователем.

3. Кнопки

Кнопки представлены элементом `button`. Они обладают широкими возможностями по конфигурации. Так, в зависимости от значения атрибута `type` мы можем создать различные типы кнопок:

- `submit`: кнопка, используемая для отправки формы;
- `reset`: кнопка сброса значений формы;
- `button`: кнопка без какого-либо специального назначения, обычно её нажатие обрабатывается программно кодом на JavaScript;

Если кнопка используется для отправки формы, то есть у нее установлен атрибут `type="submit"`, то мы можем задать у нее ряд дополнительных атрибутов:

- `form`: определяет форму, за которой закреплена кнопка отправки;
- `formaction`: устанавливает адрес, на который отправляется форма. Если у элемента `form` задан атрибут `action`, то он переопределяется;
- `formenctype`: устанавливает формат отправки данных. Если у элемента `form` установлен атрибут `enctype`, то он переопределяется;
- `formmethod`: устанавливает метод отправки формы (`post` или `get`). Если у элемента `form` установлен атрибут `method`, то он переопределяется

Например, определим на форме кнопку отправки и кнопку сброса:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Формы в HTML5</title>
    </head>
    <body>
        <form>
            <p><input type="text" name="login"/></p>
            <p><input type="password" name="password"/></p>
            <p>
                <button type="submit" formmethod="get"
                 formaction="index.html">Отправить</button>
                <button type="reset">Отмена</button>
            </p>
        </form>
    </body>
</html>
```

Кроме элемента `button` для создания кнопок можно использовать элемент `input`, у которого атрибут равен `submit` или `reset`. Например:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Формы в HTML5</title>
    </head>
    <body>
        <form>
            <p><input type="text" name="login"/></p>
            <p><input type="password" name="password"/></p>
```

```
            <p>
                <input type="submit" value="Отправить" />
                <input type="reset" value="Отмена" />
            </p>
        </form>
    </body>
</html>
```

Ещё один элемент `input` с атрибутом `type="image"` позволяет использовать в качестве кнопки изображение:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Форма ввода в HTML5</title>
    </head>
    <body>
<form>
 <p>
  <input type="text" name="search" />
  <input type="image" src="search.png" name="submit" />
 </p>
</form>
</body></html>
```

Кроме наличия изображения в остальном эта кнопка будет аналогична стандартной кнопке отправки `input type="submit"` или `button type="submit"`.

4. Текстовые поля

Однострочное текстовое поле создается с помощью элемента `input`, когда его атрибут `type` имеет значение `text`:

```
<input type="text" name="login" />
```

С помощью ряда дополнительных атрибутов можно настроить текстовое поле:

- `dirname`: устанавливает направление ввода текста;
- `maxlength`: максимально допустимое количество символов в текстовом поле;
- `pattern`: определяет шаблон, которому должен соответствовать вводимый текст;
- `placeholder`: устанавливает текст, который по умолчанию отображается в текстовом поле;
- `readonly`: делает текстовом поле доступным только для чтения;
- `required`: указывает, что текстовое поле обязательно должно иметь значение;
- `size`: устанавливает ширину текстового поля в видимых символах;
- `value`: устанавливает значение по умолчанию в текстовом поле

Применим некоторые атрибуты:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Текстовые поля в HTML5</title>
    </head>
    <body>
        <form>
```

```
            <p><input type="text" name="userName"
              placeholder="Введите имя" size="18" /></p>
            <p><input type="text" name="userPhone"
                 placeholder="Введите номер телефона"
                size="18" maxlength="11" />
            </p>
            <p>
                <button type="submit">Отправить</button>
                <button type="reset">Отмена</button>
            </p>
        </form>
    </body>
</html>
```

В этом примере во втором текстовом поле сразу устанавливаются атрибуты `maxlength` и `size`. При этом `size` - **количество символов, которые помещаются в видимое пространство** поля, больше, чем **допустимое количество символов**. Однако все равно ввести символов больше, чем `maxlength`, мы не сможем.

В данном случае также важно различать атрибуты `value` и `placeholder`, хотя оба устанавливают видимый текст в поле. Однако `placeholder` устанавливает своего рода **подсказку** или приглашение к вводу, поэтому он обычно отмечается серым цветом. В то время как значение `value` представляет введенный в поле текст по умолчанию:

```
<p><input type="text" name="userName" value="Том" /></p>
<p><input type="text" name="userPhone" placeholder="Номер
телефона" /></p>
```

Атрибуты `readonly` и `disabled` **делают текстовое поле недоступным**, однако сопровождаются разным визуальным эффектом. В случае с `disabled` текстовое поле затеняется:

```
<p><input type="text" name="userName" value="Том" readonly /></p>
<p><input type="text" name="userPhone" value="+12345678901"
disabled /></p>
```

Среди атрибутов текстового поля также следует отметить такой атрибут как `list`. Он содержит ссылку на элемент `datalist`, который определяет набор значений, появляющихся в виде подсказки при вводе в текстовое поле. Например:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Текстовые поля в HTML5</title>
    </head>
    <body>
        <form>
            <p><input list="phonesList" type="text" name="model"
               placeholder="Введите модель" /></p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
        <datalist id="phonesList">
            <option value="iPhone 6S" label="54000"/>
```

```
            <option value="Lumia 950">35000</option>
            <option value="Nexus 5X"/>
        </datalist>
    </body>
</html>
```
Атрибут `list` текстового поля указывает на `id` элемента `datalist`. Сам элемент `datalist` с помощью вложенных элементов `option` определяет элементы списка. И при вводе в текстовое поле этот список отображается в виде подсказки.

Для создания **полей поиска** предназначен элемент `input` с атрибутом `type="search"`. Формально он представляет собой простое текстовое поле:
```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Поиск в HTML5</title>
    </head>
    <body>
    <form>
        <input type="search" name="term" />
        <input type="submit" value="Поиск" />
    </form>
    </body>
</html>
```

Для ввода пароля используется элемент `input` с атрибутом `type="password"`. Его отличительной чертой является то, что вводимые символы маскируются точками:
```
<form>
    <p><input type="text" name="login" /></p>
    <p><input type="password" name="password" /></p>
    <input type="submit" value="Авторизация" />
</form>
```

5. Метки и автофокус

Вместе с полями ввода нередко используются **метки**, которые представлены элементом `label`. Метки создают аннотацию или заголовок к полю ввода, указывают, для чего это поле предназначено.
Для связи с полем ввода метка имеет атрибут `for`, который указывает на идентификатор `id` поля ввода:
```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Label в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="login">Логин: </label>
                <input type="text" id="login" name="login" />
            </p>
```

```
            <p>
                <label for="password">Пароль: </label>
                <input type="password" id="password"
                 name="password" />
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```
Так, текстовое поле здесь имеет атрибут `id="login"`. Поэтому у связанной с ним метки устанавливается атрибут `for="login"`. Нажатие на эту метку позволяет перевести фокус на текстовое поле для ввода логина.
Особенно необходим тег `label` с атрибутом `for` при создании "кликабельных" подписей к радиокнопкам и чекбоксам (см. п. 5.7).

Также мы можем установить **автофокус** по умолчанию на какое-либо поле ввода. Для этого применяется атрибут `autofocus`:
```
<form>
    <p>
        <label for="login">Логин: </label>
        <input type="text" autofocus id="login" name="login" />
    </p>
    <p>
        <label for="password">Пароль: </label>
        <input type="password" id="password" name="password" />
    </p>
    <p>
        <button type="submit">Отправить</button>
    </p>
</form>
```
Здесь при запуске страницы фокус сразу же переходит на текстовое поле.

6. Элементы для ввода чисел

Для ввода чисел используется элемент `input` с атрибутом `type="number"`. Он создает числовое поле, которое мы можем настроить с помощью следующих атрибутов:

- `min`: минимально допустимое значение
- `max`: максимально допустимое значение
- `readonly`: доступно только для чтения
- `required`: указывает, что данное поле обязательно должно иметь значение
- `step`: значение, на которое будет увеличиваться число в поле
- `value`: значение по умолчанию

Используем числовое поле:
```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Числовое поле в HTML5</title>
    </head>
    <body>
```

```
<form>
 <p>
  <label for="age">Возраст: </label>
  <input type="number" step="1" min="1" max="100" value="10"
       id="age" name="age"/>
 </p>
 <p>
  <button type="submit">Отправить</button>
 </p>
</form>
</body></html>
```

Здесь числовое поле по умолчанию имеет значение 10 (`value="10"`), минимально допустимое значение, которое мы можем ввести, - `1`, а максимальное допустимое значение - `100`. И атрибут `step="1"` устанавливает, что значение будет увеличиваться на единицу.

В зависимости от браузера визуализация этого поля может отличаться. Но как правило, у большинства современных браузеров, кроме IE 11 и Microsoft Edge, справа в поле ввода имеются стрелки для увеличения/уменьшения значения на величину, указанную в атрибуте `step`.

Как и в случае с текстовым полем мы можем здесь прикрепить список `datalist` с диапазоном возможных значений:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Числовое поле в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="price">Цена: </label>
                <input type="number" list="priceList"
                    step="1000" min="3000" max="100000"
                    value="10000" id="price" name="price"/>
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
        <datalist id="priceList">
            <option value="15000" />
            <option value="20000" />
            <option value="25000" />
        </datalist>
    </body>
</html>
```

Ползунок представляет собой шкалу, на которой мы можем выбрать одно из значений. Для создания ползунка применяется элемент `input` с атрибутом `type="range"`. Во

многом ползунок похож на простое поле для ввода чисел. Он также имеет атрибуты `min`, `max`, `step` и `value`:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Ползунок в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="price">Цена:</label>
                1<input type="range" step="1" min="0" max="100"
                 value="10" id="price" name="price"/>100
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

7. Флажки и переключатели

Флажок (чекбокс) представляет собой элемент, который может находиться в двух состояниях: отмеченном и неотмеченном. Флажок создается с помощью элемента `input` с атрибутом `type="checkbox"`:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Чекбокс в HTML5</title>
    </head>
    <body>
        <h2>Изучаемые технологии</h2>
        <form>
            <p>
                <input type="checkbox" checked
                   name="html5"/>HTML5
            </p>
            <p>
                <input type="checkbox" name="dotnet"/>.NET
            </p>
            <p>
                <input type="checkbox" name="java"/>Java
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

Атрибут `checked` позволяет установить флажок в отмеченное состояние.

Переключатели или **радиокнопки** похожи на флажки, они также могут находиться в отмеченном или неотмеченном состоянии. Только для переключателей можно создать одну группу, в которой одновременно можно выбрать только один переключатель. Например:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Радиокнопки в HTML5</title>
    </head>
    <body>
        <form>
            <h2>Укажите пол</h2>
            <p>
                <input type="radio" value="man" checked
                  name="gender"/>мужской
            </p>
            <p>
                <input type="radio" value="woman"
                  name="gender"/>женский
            </p>
            <h2>Выберите технологию</h2>
            <p>
                <input type="radio" value="html5" checked
                   name="tech"/>HTML5
            </p>
            <p>
                <input type="radio" value="net"
                   name="tech"/>.NET
            </p>
            <p>
                <input type="radio" value="java"
                  name="tech"/>Java
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

Для создания радиокнопки надо указать атрибут `type="radio"`. И теперь другой атрибут `name` указывает не на имя элемента, а на имя группы, к которой принадлежит элемент-радиокнопка. В данном случае у нас две группы радиокнопок: `gender` и `tech`. Из каждой группы мы можем выбрать только один переключатель. Опять же чтобы отметить радиокнопку, у нее устанавливается атрибут `checked`

Важное значение играет атрибут `value`, который при отправке формы позволяет серверу определить, какой именно переключатель был отмечен.

8. Элементы для ввода цвета, url, email, телефона

За **установку цвета** в HTML5 отвечает специальный элемент `input` с типом `color`:

```
<label for="favcolor">Выберите цвет</label>
<input type="color" id="favcolor" name="favcolor" />
```
Элемент отображает выбранный цвет. А при нажатии на него появляется системное диалоговое окно для установки цвета.
Значением этого элемента будет числовой шестнадцатеричный код выбранного цвета.
С помощью элемента `datalist` мы можем задать набор цветов, из который пользователь может выбрать нужный:
```
<label for="favcolor">Выберите цвет</label>
<input type="color" list="colors" id="favcolor"
 name="favcolor" />
<datalist id="colors">
    <option value="#0000FF" label="blue">
    <option value="#008000" label="green">
    <option value="#ff0000" label="red">
</datalist>
```
Каждый элемент `option` в `datalist` должен в качестве значения принимать шестнадцатеричный код цвета, например, `"#0000FF"`. После выбора цвета данный числовой код устанавливается в качестве значения в элементе `input`.

Ряд полей `input` предназначены для ввода таких данных, как url-адрес, адрес электронной почты и телефонного номера. Они однотипны и во многом отличаются только тем, что для атрибута `type` принимают соответственно значения `email`, `tel` и `url`.
Для их настройки мы можем использовать те же атрибуты, что и для обычного текстового поля:

- `maxlength`: максимально допустимое количество символов в поле;
- `pattern`: определяет шаблон, которому должен соответствовать вводимый текст;
- `placeholder`: устанавливает текст, который по умолчанию отображается в поле;
- `readonly`: делает текстовом поле доступным только для чтения;
- `required`: указывает, что текстовое поле обязательно должно иметь значение;
- `size`: устанавливает ширину поля в видимых символах;
- `value`: устанавливает значение по умолчанию для поля;
- `list`: устанавливает привязку к элементу `datalist` со списком возможных значений

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Форма ввода в HTML5</title>
    </head>
    <body>
  <form>
   <p>
    <label for="email">Email: </label>
    <input type="email" placeholder="user@gmail.com"
           id="email" name="email"/>
  </p>
  <p>
   <label for="url">URL: </label>
   <input type="url" id="url" name="url"/>
```

```
  </p>
 <p>
  <label for="phone">Телефон: </label>
  <input type="tel" placeholder="(XXX)-XXX-XXXX"
         id="phone" name="phone"/>
 </p>
 <p>
  <button type="submit">Отправить</button>
 </p>
</form>
</body></html>
```

Основное преимущество подобных полей ввода перед обычными текстовыми полями состоит в том, что поля ввода для email, url, телефона для проверки ввода используют соответствующий шаблон. Например, если мы введем в какое-либо поле некорректное значение и попробуем отправить форму, то браузер может отобразить нам сообщение о некорректном вводе, а форма не будет отправлена.

9. Элементы для ввода даты и времени

Для работы с датами и временем в HTML5 предназначено несколько типов элементов `input`:

- `datetime-local`: устанавливает дату и время;
- `date`: устанавливает дату;
- `month`: устанавливает текущий месяц и год;
- `time`: устанавливает время;
- `week`: устанавливает текущую неделю

Например, используем поле для установки даты:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Форма ввода в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="firstname">Имя: </label>
                <input type="text" id="firstname"
                 name="firstname"/>
            </p>
            <p>
                <label for="date">Дата рождения: </label>
                <input type="date" id="date" name="date"/>
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

И при вводе в поле для даты будет открываться календарик. Действие этого элемента зависит от браузера.
Применение остальных элементов:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Форма ввода в HTML5</title>
    </head>
    <body>
        <form>
            <p>
                <label for="week">Неделя: </label>
                <input type="week" name="week" id="week" />
            </p>
            <p>
                <label for="localdate">Дата и время: </label>
                <input type="datetime-local" id="localdate"
                 name="date"/>
            </p>
            <p>
                <label for="month">Месяц: </label>
                <input type="month" id="month" name="month"/>
            </p>
            <p>
                <label for="time">Время: </label>
                <input type="time" id="time" name="time"/>
            </p>
            <p>
                <button type="submit">Отправить</button>
            </p>
        </form>
    </body>
</html>
```

При использовании этих элементов также надо учитывать, что Firefox поддерживает только элементы `date` и `time`, для остальных создаются обычные текстовые поля. А IE11 совсем не поддерживают эти элементы.

10. Отправка файлов

За выбор файлов на форме отвечает элемент `input` с атрибутом `type="file"`:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Отправка файлов в HTML5</title>
    </head>
    <body>
        <form enctype="multipart/form-data" method="post"
         action="http://localhost:8080/postfile.php">
            <p>
                <input type="file" name="file" />
            </p>
```

```
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```
При нажатии на кнопку "Выберите файл" открывается диалоговое окно для выбора файла. А после выбора рядом с кнопкой отображается имя выбранного файла.
Важно отметить, что для отправки файла на сервер форма должна иметь атрибут `enctype="multipart/form-data"`. При этом форма должна передаваться методом `post` (в теле http-запроса).
С помощью ряда атрибутов мы можем дополнительно настроить элементы выбора файла:

- `accept`: устанавливает тип файл, которые допустимы для выбора;
- `multiple`: позволяет выбирать множество файлов;
- `required`: требует обязательной установки файла

Пример множественного выбора файлов:
```
<form enctype="multipart/form-data" method="post"
action="http://localhost:8080/postfile.php">
    <p>
        <input type="file" name="file" multiple />
    </p>
    <p>
        <input type="submit" value="Отправить" />
    </p>
</form>
```
При нажатии на кнопку также открывается диалоговое окно для выбора файлов, только теперь, зажав клавишу Ctrl или Shift, мы можем выбрать несколько файлов, а после выбора рядом с кнопкой отобразится количество выбранных файлов.

11. Список select

Элемент `select` создает список. В зависимости от настроек это может быть выпадающий список для выбора одного элемента, либо раскрытый список, в котором можно выбрать сразу несколько элементов.
Создадим выпадающий список:
```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Элемент select в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="phone">Выберите модель:</label>
<select id="phone" name="phone">
 <option value="iphone 6s">iPhone 6S</option>
 <option value="lumia 950">Lumia 950</option>
 <option value="nexus 5x">Nexus 5X</option>
 <option value="galaxy s7">Galaxy S7</option>
</select>
            </p>
```

```
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```

Внутри элемента `select` помещаются элементы `option` - элементы списка. Каждый элемент `option` содержит атрибут `value`, который хранит значение элемента. При этом значение элемента `option` не обязательно должно совпадать с отображаемым им текстом. Например:

```
<option value="apple">iPhone 6S</option>
```

С помощью атрибута `selected` мы можем установить выбранный по умолчанию элемент - это необязательно должен быть первый элемент в списке:

```
<select id="phone" name="phone">
    <option value="iphone 6s">iPhone 6S</option>
    <option value="lumia 950">Lumia 950</option>
    <option value="nexus 5x" selected>Nexus 5X</option>
</select>
```

С помощью другого атрибута `disabled` можно запретить выбор определенного элемента. Как правило, элементы с этим атрибутом служат для создания заголовков:

```
<select id="phone" name="phone">
    <option disabled selected>Выберите модель</option>
    <option value="iphone 6s">iPhone 6S</option>
    <option value="lumia 950">Lumia 950</option>
    <option value="nexus 5x" selected>Nexus 5X</option>
</select>
```

Для создания списка с множественным выбором к элементу select надо добавить атрибут multiple:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Элемент select в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="phone">Выберите модель:</label> <br/>

<select multiple id="phone" name="phone">
 <option value="iphone 6s">iPhone 6S</option>
 <option value="lumia 950">Lumia 950</option>
 <option value="nexus 5x">Nexus 5X</option>
 <option value="galaxy s7">Galaxy S7</option>
</select>
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```

Зажав клавишу Ctrl, мы можем выбрать в таком списке несколько элементов.
Select также позволяет группировать элементы с помощью тега <optgroup>:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Элемент select в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="phone">Выберите модель:</label>

<select id="phone" name="phone">
 <optgroup label="Apple">
  <option value="iphone 6s">iPhone 6S</option>
  <option value="iphone 6s plus">iPhone 6S Plus</option>
  <option value="iphone 5se">iPhone 5SE</option>
 </optgroup>
 <optgroup label="Microsoft">
  <option value="lumia 950">Lumia 950</option>
  <option value="lumia 950 xl">Lumia 950 XL</option>
  <option value="lumia 650">Lumia 650</option>
 </optgroup>
</select>
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```

Использование групп элементов применимо как к выпадающему списку, так и к списку со множественным выбором.

12. Многострочное текстовое поле Textarea

Элемент <input type="text"/> позволяет создавать простое однострочное текстовое поле. Однако возможностей этого элемента по вводу текста бывает недостаточно, и в этой ситуации мы можем использовать многострочное текстовое поле, представленное элементом textarea:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Textarea в HTML5</title>
    </head>
    <body>
<form method="get">
 <p>
  <label for="comment">Ваш комментарий:</label><br/>
  <textarea name="comment" id="comment"
    placeholder="Не более 200 символов"
```

```
    maxlength="200"></textarea>
 </p>
 <p>
  <input type="submit" value="Добавить" />
 </p>
</form>
    </body>
</html>
```

С помощью дополнительных атрибутов `cols` и `rows` можно задать соответственно количество столбцов и строк:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Textarea в HTML5</title>
    </head>
    <body>
<form method="get">
 <p>
  <label for="comment">Ваш комментарий:</label><br/>
  <textarea id="comment" name="comment"
   placeholder="Написать комментарий"
   cols="30" rows="7"></textarea>
 </p>
 <p>
  <input type="submit" value="Добавить" />
 </p>
</form>
</body>
</html>
```

Если требуется текст по умолчанию, он набирается между внутри тега `textarea`.

13. Валидация форм

Итак, в нашем распоряжении имеются различные элементы, которые мы можем использовать в форме. Мы можем вводить в них различные значения. Однако нередко пользователи вводят не совсем корректные значения: например, ожидается ввод чисел, а пользователь вводит буквы и т.д. И для предупреждения и проверки некорректного ввода в HTML5 существует механизм валидации.

Преимущество использования валидации в HTML5 заключается в том, что пользователь после некорректного ввода может сразу получить сообщение об ошибке и внести соответствующие изменения в введенные данные.

Для создания валидации у элементов форм HTML5 используется ряд атрибутов:

- `required`: требует обязательного ввода значения. Для элементов `textarea`, `select`, `input` (с типом `text`, `password`, `checkbox`, `radio`, `file`, `datetime-local`, `date`, `month`, `time`, `week`, `number`, `email`, `url`, `search`, `tel`);
- `min` и `max`: минимально и максимально допустимые значения. Для элемента `input` с типом `datetime-local`, `date`, `month`, `time`, `week`, `number`, `range`;
- `pattern`: задает шаблон, которому должны соответствовать вводимые данные. Для элемента `input` с типом `text`, `password`, `email`, `url`, `search`, `tel`

Атрибут `required` требует **обязательного наличия** значения:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Валидация в HTML5</title>
    </head>
    <body>
<form method="get">
 <p>
  <label for="login">Логин:</label>
  <input type="text" required id="login" name="login" />
 </p>
 <p>
  <label for="password">Пароль:</label>
  <input type="password" required id="password"
    name="password" />
 </p>
 <p>
  <input type="submit" value="Отправить" />
 </p>
</form>
</body>
</html>
```

Если мы не введем в эти поля никаких данных, оставив их пустыми, и нажмем на кнопку отправки, то браузер высветит нам сообщения об ошибке, а данные не будут отправлены на сервер. В зависимости от браузера визуализация сообщения может несколько отличаться. Также границы некорректного поля ввода могут окрашиваться в красный цвет.

Для **ограничения диапазона вводимых значений** применяются атрибуты `max` и `min`:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Валидация в HTML5</title>
    </head>
    <body>
        <form method="get">
            <p>
                <label for="age">Возраст:</label>
                <input type="number" min="1" max="100"
                 value="18" id="age" name="age"/>
            </p>
            <p>
                <input type="submit" value="Отправить" />
            </p>
        </form>
    </body>
</html>
```

Атрибут pattern задает **шаблон, которому должны соответствовать данные**. Для определения шаблона используется язык так называемых регулярных выражений. Рассмотрим простейшие примеры:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Валидация в HTML5</title>
    </head>
    <body>
<form method="get">
 <p>
  <label for="phone">Телефон:</label>
  <input type="text" placeholder="+1-234-567-8901"
    pattern="\+\d-\d{3}-\d{3}-\d{4}" id="phone" name="phone" />
 </p>
 <p>
  <input type="submit" value="Отправить" />
 </p>
</form>
</body>
</html>
```

Здесь для ввода номера телефона используется регулярное выражение \+\d-\d{3}-\d{3}-\d{4}. Оно означает, что первым элементом в номере должен идти знак плюс +. Выражение \d представляет любую цифру от 0 до 9. Выражение \d{3} означает три подряд идущих цифры, а \d{4} - четыре цифры подряд. То есть это выражение будет соответствовать номеру телефона в формате "+1-234-567-8901".
Если мы введем данные, которые не соответствуют этому шаблону, и нажмем на отправку, то браузер отобразит ошибку.

Не всегда валидация является желаемой, иногда требуется ее **отключить**. И в этом случае мы можем использовать либо у элемента формы атрибут novalidate, либо у кнопки отправки атрибут formnovalidate:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Валидация в HTML5</title>
    </head>
    <body>
<form novalidate method="get">
 <p>
  <label for="phone">Телефон:</label>
  <input type="text" placeholder="+1-234-567-8901"
   pattern="\+\d-\d{3}-\d{3}-\d{4}" id="phone" name="phone" />
 </p>
 <p>
  <input type="submit" value="Отправить" formnovalidate />
 </p>
</form>
</body>
```

```
</html>
```

14. Элементы fieldset и legend

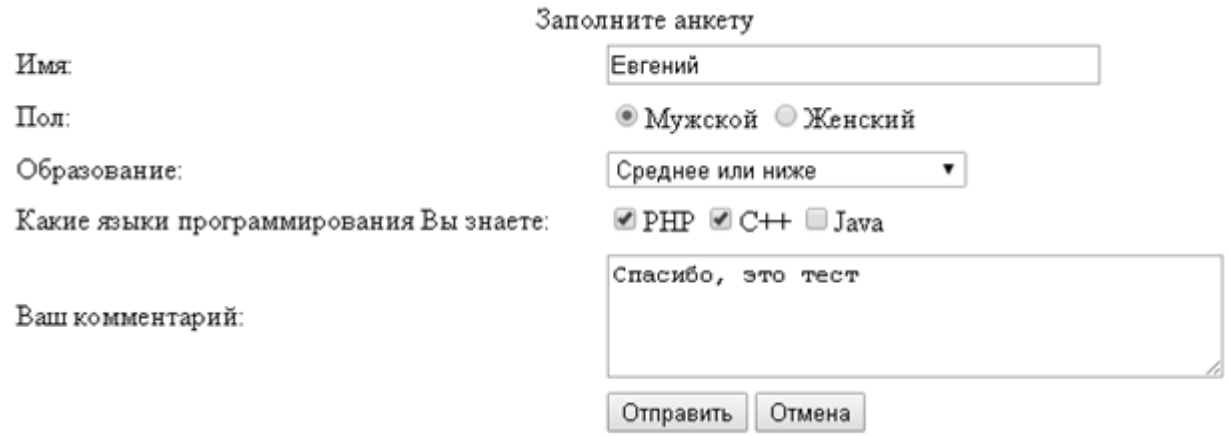
Для группировки элементов формы нередко применяется элемент `fieldset`. Он создает границу вокруг вложенных элементов, как бы создавая из них группу. Вместе с ним используется элемент `legend`, который устанавливает заголовок для группы элементов:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Элементы форм в HTML5</title>
    </head>
    <body>
        <h2>Вход на сайт</h2>
<form>
 <fieldset>
  <legend>Введите данные:</legend>
  <label for="login">Логин:</label><br>
  <input type="text" name="login" id="login" /><br>
  <label for="password">Пароль:</label><br>
  <input type="password" name="password" id="password" /><br>
  <input type="submit" value="Авторизация">
 </fieldset>
</form>
</body>
</html>
```

При необходимости мы можем создать на одной форме несколько групп с помощью элементов `fieldset`.

Задания для лабораторной работы № 4

1. Разработать форму HTML, предназначенную для ввода анкетных данных о студентах Вашей группы, например:



2. Добавьте к форме поля ввода данных HTML5 (например, дата рождения, E-mail).
3. Проверьте на валидность и в работе полученный код.

Содержание отчета

1. Титульный лист (включая названия дисциплины, номер лабораторной работы)

2. Цель лабораторной работы

3. Формулировка заданий, текст программы по заданиям (с комментариями), скрин экрана результатов выполнения заданий.

Лабораторная работа № 6
Блоковая модель

Цель: познакомиться с блоковой моделью, изучить способы размещения элементов на web-странице с использованием HTML+CSS.

Теория

Границы

Стиль границ HTML-элемента (свойство border-style значения solid, dashed, dotted, double).

Цвет границы (свойство border-color).

Толщина границы (свойство border-width значения в пикселях или thin, medium, thick).

Задание стилей для отдельных сторон, названия сторон перечислены на рисунке (рис.1.):

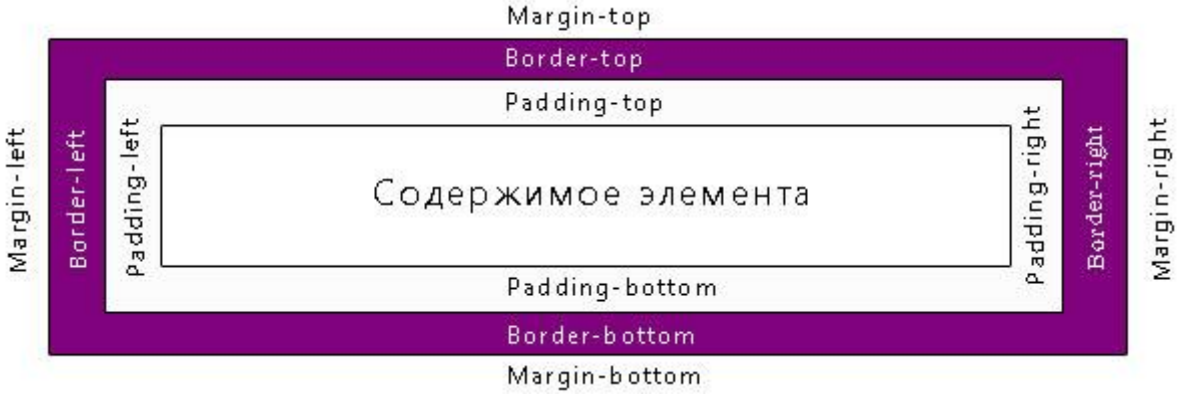


Рис.1. Стили и названия сторон блоковой модели

Способ быстрого задания стилей границ.

Пример1. Border-style:dashed double solid groove – к верхней границе будет применено dashed, к правой double, к нижней solid, а к левой groove.

Пример2. Border-style:dashed double –верхняя и нижняя граница будут оформлены как dashed, а левая и правая граница как double.

Сокращенная форма записи объединяет все свойства оформления границ в одном свойстве border. Порядок следования свойств: border-width, border-style, border-color (можно пропускать неиспользуемые свойства). Пример: border:1px solid.

Отступы (величина в пикселях (px), сантиметрах (cm), процентах (%), em):

- внутренние (свойство padding);
- внешние (свойства margin);
- величина отступа может быть задана отдельно для каждой стороны элемента.

Краткая форма записи: padding: 60px20px40px50px;margin: 30px50px.

Отображение элементов

Существуют два способа отображения элементов: visibility:hidden и display:none).

Размещение элементов

Местоположение элементов задается с помощью следующих CSS свойств:

- top – величина смещения текущего элемента от верхнего края родительского элемента;
- bottom – от нижнего края родительского элемента;
- left – от левого края родительского элемента;
- right – от правого края родительского элемента.

Описанные выше свойства позиционирования не вступят в силу пока Вы не зададите способ размещения. В CSS существуют 4 различных способа размещения элементов:

- статическое (position:static) действует по умолчанию, элементы всегда отображаются там, где они были объявлены (свойства top, bottom, left и right не работают со статичными элементами);
- фиксированное (position:fixed) элементы не изменяют своего местоположения даже при прокрутке окна браузера (свойства top, bottom, left, right работают);
- относительное (position:relative);
- абсолютное (position:absolute).

Наложение элементов (свойство z-index значение – любое число, может быть отрицательное).

Выравнивание элементов.

По центру с помощью margin. Блоковые элементы могут быть выровнены по центру установкой margin с левой и правой стороны значения auto.

С помощью свойств позиционирования

С помощью свойства float. Элемент, выровненный с помощью float, будет прижат к левой или правой границе родительского элемента (в зависимости от заданного значения) и заставит следующие за ним элементы "обтекать" его с противоположной стороны.

Очистить элементы от float можно командой clear:both.

Задания для лабораторной работы № 6

Задание 1. Для предварительного знакомства с блоковой моделью внимательно посмотрите следующий HTML документ и его отображение браузером.

<html>

<head>

```
<style type='text/css'>
#wrap {
margin:0px;
padding:20px;
height:160px;
background-color:yellow;
}
.ex1 {
border:10px red solid;
margin-left:50px;
margin-right:10px;
padding:15px;
background-color:green;
color:white;
}
.ex2 {
border:5px brown solid;
margin-top:30px;
margin-left:250px;
margin-right:70px;
padding:15px;
background-color:green;
color:white;
}
</style>
</head> <body>
<div id='wrap'>
<div class='ex1'>Содержимое первого элемента</div>
<div class='ex2'>Содержимое второго элемента</div>
</div>
</body> </html>
```

На примере изучите внутренние и внешние отступы элементов. Измените цвет элементов, размер элементов, величину отступов и сохраните как свою работу.

Задание 2. Оформите элементы согласно их описанию. Стиль границ, цвет и толщину можно придумать свои.

```
<html>
<head>
<style type='text/css'>
#par1
{
/* Пишите код здесь */
}
#par2
{
/* Пишите код здесь */
}
#par3
{
/* Пишите код здесь */
}
#par4
{
/* Пишите код здесь */
}
#par5
{
/* Пишите код здесь */
}
#par6
```

```
{
/* Пишите код здесь */
}
#par7
{
/* Пишите код здесь */
}
</style>
</head>
<body>
<p id='par1'>Я имею сплошную границу коричневого цвета.</p>
<p id='par2'>Я имею сплошную границу красного цвета слева и справа.</p>
<p id='par3'>Я имею сплошную границу розового цвета сверху, и пунктирную границу голубого цвета слева, справа и снизу.</p>
<p id='par4'>Я имею границу оранжевого цвета толщиной 1 пиксель.</p>
<p id='par5'>Я имею границу красного цвета толщиной 2 пикселя сверху и снизу и пунктирную границу серого цвета слева и справа.</p>
<p id='par6'>Я имею двойную сплошную границу #1435AD цвета. </p>
<p id='par7'>Я имею пунктирную границу #FF8100 цвета и толщиной 1 пиксель сверху и сплошную границу #0B6124 цвета толщиной 2 пикселя снизу. </p>
</body>
</html>
```

Задание 3. Узнайте кодовое слово, используя свойство visibility:hidden, затем свойство display:none). Объясните разницу.

```
<html> <head>
<style type='text/css'>
span
{
font-size:1.6em;
font-family:verdana;
color:green;
}
/* Пишите код здесь */
</style>
```

```
</head>
<body>
<b>
<p>Для того, чтобы узнать кодовое слово скройте элементы со следующими id: code, winter, sky, frog, forest, cloud, hide. </p></b><br />
<hr />
<span id='code'>Б</span><span id='mirror'>А</span><span id='winter'>ДЖ</span><span id='grass'>Л</span><span id='sky'>МЖ</span><span id='bright'>Т</span><span id='forest'>Ь</span><span id='sun'>А</span><span id='frog'>ЗП</span><span id='cloud'>ШН</span><span id='true'>Й</span><span id='hide'>РО</span>
<hr />
</body> </html>
```

Задание 4. Поработайте с элементами div на своей второй web-странице. При помощи свойств margin, border, padding задайте для них различный цвет и толщину отступа, границы полей так, чтобы эти составляющие модели блоковых элементов стали видны. Поэкспериментируйте с разными способами их размещения, не менеее 3 вариантов, сохраните в файлах..

На второй web-странице создайте список следующего вида, где элементы списка являются ссылками на страницы с выполненными заданиями:

- задание (лабораторная 1)
- задание (лабораторная 2)
- задание (лабораторная 3)
- задание 1 (лабораторная 6)
- задание 2 (лабораторная 6)
- задание 3 (лабораторная 6)

Содержание отчета

1. Титульный лист (включая названия дисциплины, номер лабораторной работы)

2. Цель лабораторной работы

3. Формулировка заданий, текст программы по заданиям (с комментариями), скрин экрана результатов выполнения заданий.

Лабораторная работа № 7
Основы работы с CSS3

Цель: научиться применять CSS3 при создании веб-страниц.

Теория

1. Основные понятия

Каскадные таблицы стилей CSS (Cascading Style Sheets) определяют представление документа, его внешний вид.
Стиль в CSS представляет правило, которое указывает браузеру, как надо форматировать элемент. Форматирование может включать установку цветов фона элемента, настройку шрифта, обрамления и т. д.

Определение стиля состоит из двух частей: **селектор**, который указывает на элемент, и **блок объявления** стиля - набор команд, которые устанавливают правила форматирования. Например:

```
div {
   background-color:red;
   width: 100px;
   height: 60px;
}
```

В данном случае селектором является div. Этот селектор указывает, что этот стиль будет применяться ко всем элементам div.
После селектора в фигурных скобках идет блок объявления стиля. Между открывающей и закрывающей фигурными скобками определяются команды, указывающие, как форматировать элемент.

Каждая команда состоит из **свойства** и **значения**. Так, в следующем выражении:
background-color: white;
background-color представляет собой свойство, а red - значение.
Свойство определяет конкретный стиль. Свойств CSS существует множество.
Так, свойство background-color определяет цвет фона. После двоеточия следует значение для этого свойства: white. То есть, для фона элемента устанавливается цвет "white" (белый).
После каждой команды ставится точка с запятой, которая отделяет данную команду от других.
Наборы таких стилей часто называют **таблицами стилей** или CSS (Cascading Style Sheetsили каскадные таблицы стилей). Существует 3 основных способа определения стилей.

2. Атрибут style

Первый способ заключается во встраивании стилей непосредственно в элемент с помощью атрибута style:

```
<h2 style="color:blue;">Стили</h2>
<div
 style="width: 100px; height: 100px; background-color: red;">
</div>
```

Здесь определены два элемента - заголовок h2 и блок div. У заголовка определен синий цвет текста с помощью свойства color. У блока div определены свойства ширины (width), высоты (height), а также цвета фона (background-color).

3. Элемент Style

Второй способ состоит в использования элемента style в документе HTML. Этот элемент сообщает браузеру, что данные внутри являются кодом CSS, а не разметкой HTML:

```
<head>
 <style>
  h2{
   color:blue;
  }
  div{
   width: 100px;
   height: 100px;
   background-color: red;
  }
 </style>
</head>
```

Результат в данном случае будет тем же, что и в предыдущем случае, но стилевые указания будут действовать на *все* теги h2 и div в документе, а не только на те, к которым применён стиль в п. 3.2.

Часто элемент style определяется внутри элемента head, однако может также использоваться в других частях HTML-документа. Элемент style содержит наборы стилей. У каждого стиля указывается вначале селектор, после чего в фигурных скобках идет все те же определения свойств CSS и их значения, что были использованы в предыдущем примере.

Второй способ делает код HTML чище за счет вынесения стилей в элемент style. Но также есть и третий способ, который заключается в вынесении стилей во внешний файл.

4. Файл CSS

Создадим в одной папке с HTML странице текстовый файл, который переименуем в styles.css и определим в нем следующее содержимое:

```
h2 {
 color:blue;
}
div {
 width: 100px;
 height: 100px;
 background-color: red;
}
```

Это те же стили, что были внутри элемента style. Также изменим код web-страницы:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Стили</title>
      <link rel="stylesheet"
        type="text/css" href="styles.css"/>
   </head>
   <body>
      <h2>Стили</h2>
```

```
        <div></div>
    </body>
</html>
```
Здесь уже нет элемента style, зато есть элемент link, который подключает выше созданный файл styles.css из текущей папки.
При таком определении стили легче модифицировать, чем встроенные, способ является **предпочтительным** в HTML5.
Использование стилей во внешних файлах позволяет уменьшить нагрузку на web-сервер с помощью механизма кэширования. К тому же, можно применить один стиль к любому количеству документов HTML.

5. Совмещение способов определения стиля и разрешение конфликтов

Все три способа могут сочетаться в одном документе

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <link rel="stylesheet"
         type="text/css" href="styles.css"/>
        <style>
            div{
                width:200px;
            }
        </style>
    </head>
    <body>
        <div style="width:120px;"></div>
    </body>
</html>
```

Различные стилевые правила, указанные для элемента, могут **конфликтовать** между собой, в этом случае применение конкретного стиля решают **правила приоритета**.
Действует следующая система приоритетов:

- Если у элемента определены встроенные стили (inline-стили), то они имеют высший приоритет, то есть в примере выше итоговой шириной будет 120 пикселей;
- Далее в порядке приоритета идут стили, которые определены в элементе style;
- Наименее приоритетными стилями являются те, которые определены во внешнем файле.

6. Устаревшие (deprecated) атрибуты HTML и стили CSS

Многие элементы, совместимые с HTML4, позволяют устанавливать стили отображения с помощью атрибутов. Например, у ряда элементов мы можем применять атрибуты width и height для установки ширины и высоты элемента соответственно. Однако подобного подхода следует избегать и вместо встроенных атрибутов следует применять стили CSS. Важно чётко понимать, что разметка **HTML** должна предоставлять только **структуру** HTML-документа, а весь его **внешний вид**, стилизацию должны определять стили **CSS**.

7. Валидация кода CSS

В процессе написания стилей CSS могут возникать вопросы, а правильно ли так определять стили, корректны ли они. В этом случае мы можем воспользоваться валидатором CSS, который доступен по адресу https://jigsaw.w3.org/css-validator/

8. Селекторы

Селекторы по тегам содержат имя тега без символов < и > и применяются ко всем подходящим тегам.

```
div{
   width:50px; /* ширина */
   height:50px; /* высота */
   background-color:red; /* цвет фона */
   margin: 10px; /* отступ от других элементов */
}
```

9. Классы

Иногда для одних и тех же элементов требуется различная стилизация. И в этом случае мы можем использовать **классы**.

Для определения селектора класса в CSS перед названием класса ставится **точка**:

```
.redBlock{
 background-color: red;
}
```

Название класса может быть произвольным. Например, в данном случае название класса - "redBlock". Однако при этом в имени класса разрешается использовать буквы, числа, дефисы и знаки подчеркивания, причём, начинаться название класса должно с буквы.

Также нужно учитывать регистр имен: названия "article" и "ARTICLE" будут представлять разные классы.

После определения класса мы можем его применить к элементу с помощью атрибута class. Например:

```
<style>
        div{
           width: 50px;
           height: 50px;
           margin: 10px;
        }
        .redBlock{
           background-color: red;
        }
        .blueBlock{
           background-color: blue;
        }
</style>
<div class="redBlock"></div>
<div class="redBlock"></div>
<div class="blueBlock"></div>
<div class="redBlock"></div>
```

10. Идентификаторы

Для идентификации уникальных на web-странице элементов используются **идентификаторы**, которые определяются с помощью атрибутов id. Например, на странице может быть головной блок или шапка:

```
<div id="header"></div>
```

Определение стилей для идентификаторов аналогично определению классов, только вместо точки ставится символ **решётки** #:

```
<style>
        div{
           margin: 10px;
```

```
        border: 1px solid #222;
    }
    #header{
        height: 80px;
        background-color: #ccc;
    }
    #content{
        height: 180px;
        background-color: #eee;
    }
    #footer{
        height: 80px;
        background-color: #ccc;
    }
</style>
<div id="header">Шапка сайта</div>
<div id="content">Основное содержимое</div>
<div id="footer">Футер</div>
```

Идентификаторы в большей степени относятся к структуре web-страницы и в меньшей степени к стилизации. Для стилизации преимущественно используются классы, нежели идентификаторы.

11. Универсальный селектор

Кроме селекторов тегов, классов и идентификаторов в CSS также есть так называемый универсальный селектор, который представляет знак звездочки (*). Он применяет стили ко всем элементам на HTML-странице:

```
* {
 background-color: red;
}
```

Зачастую применение универсального селектора можно заменить установкой стилей для тега html или body.

12. Стилизация группы селекторов

Иногда определенные стили применяются к целому ряду селекторов. Например, мы хотим применить ко всем заголовкам подчеркивание. В этом случае мы можем перечислить селекторы всех элементов через запятую:

```
<style>
h1, h2, h3, h4 {
 color: red;
}
</style>
<h1>CSS3<h1>
<h2>Селекторы</h2>
<h3>Группа селекторов</h3>
<p>Некоторый текст...</p>
```

Группа селекторов может содержать как селекторы тегов, так и селекторы классов и идентификаторов, например:

```
h1, #header, .redBlock{
    color: red;
}
```

13. Селекторы потомков

Web-страница может иметь сложную организацию, одни элементы внутри себя могут определять другие элементы. Вложенные элементы иначе можно назвать потомками, а контейнер, содержащий эти элементы - родителем.

Например, пусть элемент body на web-странице имеет следующее содержимое:

```
<body>
   <h2>Заголовок</h2>
   <div>
      <p>Текст</p>
   </div>
</body>
```

Внутри элемента body определено три вложенных элемента: h2, div, p. Все эти элемены являются потомками элемента body.

А внутри элемента div определен только один вложенный элемент - p, поэтому элемент div имеет только одного потомка.

Используя специальные селекторы, мы можем стилизовать вложенные элементы или потомков внутри строго определенных элементов. Например, у нас на странице могут быть параграфы внутри блока с основным содержимым и внутри блока футера. Но для параграфов внутри блока основного содержимого мы захотим установить один шрифт, а для параграфов футера другой.

```
<style>
 #main p {
  font-size: 16px;
 }
 #footer p {
  font-size: 13px;
 }
</style>
<div id="main">
 <p>Первый абзац</p>
 <p>Второй абзац</p>
</div>
<div id="footer">
 <p>Текст футера</p>
</div>
```

Для применения стиля к вложенному элементу селектор должен содержать вначале родительский элемент и затем вложенный:

```
#main p{
 font-size: 16px;
}
```

Данный стиль будет применяться только к тем элементам p, которые находятся внутри элемента с идентификатором main.

```
<style>
li .redLink{
 color: red;
}
</style>
<ul>
<li>Просто элемент</li>
<li><span class="redlink">Красный элемент<span></li>
</ul>
```

Здесь стиль применяется к элементам с классом "redLink", которые находятся внутри элемента <li>.

14. Селекторы дочерних элементов

Селекторы дочерних элементов отличаются от селекторов потомков тем, что позволяют выбрать элементы только первого уровня вложенности. Например:

```
<body>
   <h2>Заголовок</h2>
   <div>
      <p>Текст</p>
   </div>
</body>
```

Хотя вложенными в элемент body элементами являются три элемента - h2, div, p, но дочерними для body являются только два - div и h2, так как они находятся в первом уровне вложенности. А элемент p находится на втором уровне вложенности, так как вложен внутрь элемента div, а не просто элемента body.

Для обращения к дочерним элементам используется знак ">":

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы атрибутов в CSS3</title>
      <style>
         .article > p{
               color: red;
         }
      </style>
   </head>
   <body>
      <div class="article">
         <p>Аннотация к статье</p>
         <div class="content">
            <p>Текст статьи</p>
         </div>
      </div>
   </body>
</html>
```

В блоке с классом article есть два параграфа. Селектор .article > p выбирает только те параграфы, который находятся непосредственно в блоке article.

15. Селекторы элементов одного уровня

Селекторы элементов одного уровня или смежных элементов позволяют выбрать элементы, которые находятся на одном уровне вложенности. Иногда такие элементы еще называют **сиблинги** (siblings) или сестринскими элементами. Например:

```
<body>
   <h2>Заголовок</h2>
   <div>
      <p>Текст первого блока</p>
   </div>
   <div>
      <p>Текст второго блока</p>
```

```
    </div>
</body>
```
Здесь элементы h2 и оба блока div являются смежными, так как находятся на одном уровне. А элементы параграфов и заголовок h2 не являются смежными, так как параграфы вложены в блоки div.

Чтобы стилизовать **первый смежный элемент** по отношению к определенному элементу, используется знак плюса "+":

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            h2+div { color: red; }
        </style>
    </head>
    <body>
        <h2>Заголовок</h2>
        <div>
            <p>Текст первого блока</p>
        </div>
        <div>
            <p>Текст второго блока</p>
        </div>
    </body>
</html>
```
Селектор h2+div позволяет определить стиль (в данном случае красный цвет текста) для блока div, который идет непосредственно после заголовка h2.

Итак, этот селектор будет стилизовать блок div, если он будет идти непосредственно после заголовка. Если же между заголовком и блоков div будет находиться еще какой-либо элемент, то к нему не будет применяться стиль, например:

```
<h2>Заголовок</h2>
<p>Элемент между заголовком и блоком div</p>
<div>
    <p>Текст первого блока</p>
</div>
```
Если нам надо стилизовать вообще **все смежные элементы одного уровня**, неважно непосредственно идут они после определенного элемента или нет, то в этом случае вместо знака плюса необходимо использовать знак тильды "~":

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Селекторы в CSS3</title>
        <style>
            h2~div { color: red; }
        </style>
    </head>
    <body>
        <h2>Заголовок</h2>
        <p>Аннотация</p>
        <div>
```

```
        <p>Текст первого блока</p>
      </div>
      <div>
        <p>Текст второго блока</p>
      </div>
  </body>
</html>
```

16. Псевдоклассы

В дополнение к селекторам тегов, классов и идентификаторов нам доступны селекторы **псевдоклассов**, которые несут дополнительные возможности по выбору нужных элементов.

Список доступных псевдоклассов:

- :root: позволяет выбрать корневой элемент web-страницы, обычно корневым элементом является элемент <html>;
- :link: применяется к ссылкам и представляет ссылку в обычном состоянии, по которой еще не совершен переход;
- :visited: применяется к ссылкам и представляет ссылку, по которой пользователь уже переходил;
- :active: применяется к ссылкам и представляет ссылку в тот момент, когда пользователь осуществляет по ней переход;
- :hover: представляет элемент, на который пользователь навел указатель мыши. Применяется преимущественно к ссылкам, однако может также применяться и к другим элементам, например, к параграфам;
- :focus: представляет элемент, который получает фокус, то есть когда пользователь нажимает клавишу табуляции или нажимает кнопкой мыши на поле ввода (например, текстовое поле) ;
- :not: позволяет исключить элементы из списка элементов, к которым применяется стиль;
- :lang: стилизует элементы на основании значения атрибута lang;
- :empty: выбирает элементы, которые не имеют вложенных элементов, то есть являются пустыми.

При применении псевдоклассов перед ними всегда ставится двоеточие. Например, стилизуем ссылки, используя псевдоклассы:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Псевдоклассы в CSS3</title>
<style>
 a:link    {color:blue; text-decoration:none}
 a:visited {color:pink; text-decoration:none}
 a:hover   {color:red; text-decoration:underline}
 a:active  {color:yellow; text-decoration:underline}
 input:hover {border:2px solid red;}
</style>
</head>
  <body>
    <a href="index.html">Учебник по CSS3</a>
    <input type="text" />
  </body>
</html>
```

Селектор :not() позволяет выбрать все элементы кроме определенных, то есть исключить некоторые элементы из выбора.

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
      <style>
         a:not(.blueLink) { color: red; }
      </style>
   </head>
   <body>
      <a>Первая ссылка</a><br/>
      <a class="blueLink">Вторая ссылка</a><br/>
      <a>Третья ссылка</a>
   </body>
</html>
```

Селектор a:not(.blueLink) применяет стиль ко всем ссылкам за исключением тех, которые имеют класс "blueLink". В скобки псевдоклассу not передается селектор элементов, которые надо исключить.

Селектор :lang выбирает элементы на основании атрибута lang:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
      <style>
         :lang(ru) {
            color: red;
         }
      </style>
   </head>
   <body>
      <form>
         <p lang="ru-RU">Я изучаю CSS3</p>
         <p lang="en-US">I study CSS3</p>
         <p lang="de-DE">Ich lerne CSS3</p>
      </form>
   </body>
</html>
```

17. Псевдоклассы дочерних элементов

Особую группу псевдоклассов образуют псевдоклассы, которые позволяют выбрать определенные дочерние элементы:

- :first-child: представляет элемент, который является первым дочерним элементом;
- :last-child: представляет элемент, который является последним дочерним элементом;
- :only-child: представляет элемент, который является единственным дочерним элементом в каком-нибудь контейнере;

- :only-of-type: выбирает элемент, который является единственным элементом определенного типа (тега) в каком-нибудь контейнере;
- :nth-child(n): представляет дочерний элемент, который имеет определенный номер n, например, второй дочерний элемент;
- :nth-last-child(n): представляет дочерний элемент, который имеет определенный номер n, начиная с конца;
- :nth-of-type(n): выбирает дочерний элемент определенного типа, который имеет определенный номер;
- :nth-last-of-type(n): выбирает дочерний элемент определенного типа, который имеет определенный номер, начиная с конца.

Используем **псевдокласс first-child** для выбора первых ссылок в блоках:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
      <style>
         a:first-child{
               color: red;
         }
      </style>
   </head>
   <body>
      <h3>Планшеты</h3>
      <div>
         <a>Microsoft Surface Pro 4</a><br/>
         <a>Apple iPad Pro</a><br/>
         <a>ASUS ZenPad Z380KL</a>
      </div>
      <h3>Смартфоны</h3>
      <div>
         <p>Топ-смартфоны 2016</p>
         <a>Samsung Galaxy S7</a><br/>
         <a>Apple iPhone SE</a><br/>
         <a>Huawei P9</a>
      </div>
   </body>
</html>
```

Стиль по селектору a:first-child применяется к ссылке, если она является первым дочерним элементом любого элемента.

В первом блоке элемент ссылки является первым дочерним элементом, поэтому к нему применяется определенный стиль.

А во втором блоке первым элементом является параграф, поэтому ни к одной ссылке не применяется стиль.

Используем **псевдокласс last-child**:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
```

```
    <style>
      a:last-child{
        color: blue;
      }
    </style>
  </head>
  <body>
    <h3>Смартфоны</h3>
    <div>
      <a>Samsung Galaxy S7</a><br/>
      <a>Apple iPhone SE</a><br/>
      <a>Huawei P9</a>
    </div>
    <h3>Планшеты</h3>
    <div>
      <a>Microsoft Surface Pro 4</a><br/>
      <a>Apple iPad Pro</a><br/>
      <a>ASUS ZenPad Z380KL</a>
      <p>Данные за 2016</p>
    </div>
  </body>
</html>
```

Селектор a:last-child определяет стиль для ссылок, которые являются последними дочерними элементами.

В первом блоке как раз последним дочерним элементом является ссылка. А вот во втором последним дочерним элементом является параграф, поэтому во втором блоке стиль не применяется ни к одной из ссылок.

Селектор :only-child выбирает элементы, которые являются единственными дочерними элементами в контейнерах:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Селекторы в CSS3</title>
    <style>
    p:only-child{
      color:red;
    }
    </style>
  </head>
  <body>
    <h2>Заголовок</h2>
    <div>
      <p>Текст1</p>
    </div>
    <div>
      <p>Текст2</p>
      <p>Текст3</p>
    </div>
    <div>
      <p>Текст4</p>
```

```
    </div>
  </body>
</html>
```
Параграфы с текстами "Текст1" и "Текст4" являются единственными дочерними элементами в своих внешних контейнерах, поэтому к ним применяется стиль - красный цвет шрифта.

Псевдокласс only-of-type выбирает элемент, который является единственным элементом определенного типа в контейнере. Например, единственный элемент div, при этом элементов других типов в этом же контейнере может быть сколько угодно.

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Селекторы в CSS3</title>
    <style>
    span:only-of-type{
      color: green;   /* зеленый цвет */
    }
    p:only-of-type{
      color: red; /* красный цвет */
    }
    div:only-of-type{
       color: blue;    /* синий цвет */
    }
    </style>
  </head>
  <body>
    <div>        Header       </div>
    <p>Единственный параграф и
      <span>элемент спан</span></p>
    <div>          Footer       </div>
  </body>
</html>
```
Хотя для элементов div определен стиль, он не будет применяться, так как в контейнере body находится два элемента div, а не один. Зато в body есть только один элемент p, поэтому он получит стилизацию. И также в контейнере p есть только один элемент span, поэтому он также будет стилизован.

Псевдокласс nth-child позволяет стилизовать каждый второй, третий элемент, только четные или только нечетные элементы и т.д.
Например, стилизуем четные и нечетные строки таблицы:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Селекторы в CSS3</title>
    <style>
       tr:nth-child(odd) { background-color: #bbb; }
       tr:nth-child(even) { background-color: #fff; }
    </style>
  </head>
```

```
   <body>
      <h3>Смартфоны</h3>
<table>
<tr><td>Samsung</td><td>Galaxy S7 Edge</td><td>60000</td></tr>
<tr><td>Apple</td><td>iPhone SE</td><td>39000</td></tr>
<tr><td>Microsoft</td><td>Lumia 650</td><td>13500</td></tr>
<tr><td>Alcatel</td><td>Idol S4</td><td>30000</td></tr>
<tr><td>Huawei</td><td>P9</td><td>60000</td></tr>
<tr><td>HTC</td><td>HTC 10</td><td>50000</td></tr>
<tr><td>Meizu</td><td>Pro 6</td><td>40000</td></tr>
<tr><td>Xiaomi</td><td>Mi5</td><td>35000</td></tr>
</table>
   </body>
</html>
```

Чтобы определить стиль для нечетных элементов, в селектор передается значение "odd":

```
tr:nth-child(odd) {}
```

Для стилизации четных элементов в селектор передается значение "even":

```
tr:nth-child(even) {}
```

Также в этот селектор мы можем передать номер стилизуемого элемента:

```
tr:nth-child(3) { background-color: #bbb; }
```

В данном случае стилизуется третья строка.

Еще одну возможность представляет использование заменителя для номера, который выражается буквой n:

```
tr:nth-child(2n+1) { background-color: #bbb; }
```

Здесь стиль применяется к каждой второй нечетной строке.

Число перед n (в данном случае 2) представляет тот дочерний элемент, который будет выделен следующим. Число, которое идет после знака плюс, показывают, с какого элемента нужно начинать выделение, то есть, +1 означает, что нужно начинать с первого дочернего элемента.

Таким образом, в данном случае выделение начинается с 1-го элемента, а следующим выделяется 2 * 1 + 1 = 3-й элемент, далее 2 * 2 + 1 = 5-й элемент и так далее.

К примеру, если мы хотим выделить каждый третий элемент, начиная со второго, то мы могли бы написать:

```
tr:nth-child(3n+2) { background-color: #bbb; }
```

Псевдокласс :nth-last-child по сути предоставляет ту же самую функциональность, только отсчет элементов идет не с начала, а с конца:

```
tr:nth-last-child(2) {
   background-color: #bbb;
    /* 2 строка с конца, то есть предпоследняя  */
}
tr:nth-last-child(2n+1) {
   background-color: #eee;
    /* нечетные строки, начиная с конца  */
}
```

Псевдокласс :nth-of-type позволяет выбрать дочерний элемент определенного типа по определенному номеру:

```
tr:nth-of-type(2) {
   background-color: #bbb;
}
```

Аналогично работает **псевдокласс nth-last-of-type**, только теперь отсчет элементов идет с конца:

```
tr:nth-last-of-type(2n) {
   background-color: #bbb;
}
```

18. Псевдоклассы форм

Ряд псевдоклассов используется для работы с элементами форм (см. лекцию 5):

- :enabled: выбирает элемент, если он доступен для выбора (то есть у него не установлен атрибут disabled)
- :disabled: выбирает элемент, если он не доступен для выбора (то есть у него установлен атрибут disabled)
- :checked: выбирает элемент, если у него установлен атрибут checked (для флажков и радиокнопок)
- :default: выбирает элементы по умолчанию
- :valid: выбирает элемент, если его значение проходит валидацию HTML5
- :invalid: выбирает элемент, если его значение не проходит валидацию
- :in-range: выбирает элемент, если его значение находится в определенном диапазоне (для элементов типа ползунка)
- :out-of-range: выбирает элемент, если его значение не находится в определенном диапазоне
- :required: выбирает элемент, если у него установлен атрибут required
- :optional: выбирает элемент, если у него не установлен атрибут required

Псевдоклассы enabled и disabled выбирают элементы форм в зависимости от того, установлен ли у них атрибут disabled:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
      <style>
         :enabled {
            border: 2px black solid;
            color: red;
         }
      </style>
   </head>
   <body>
      <p><input type="text" value="Enabled" /></p>
      <p><input type="text" disabled value="Disabled" /></p>
   </body>
</html>
```

Псевдокласс checked стилизует элементы формы, у которых установлен атрибут checked:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
      <style>
```

```
        :checked + span {
            color: red;
            font-weight: bold;  /* выделение жирным */
        }
    </style>
  </head>
<body>
<h2>Выберите технологию</h2>
<p>
 <input type="checkbox" checked
  name="html5"/><span>HTML5</span>
</p>
<p>
 <input type="checkbox" name="dotnet"/><span>.NET</span>
</p>
<p>
 <input type="checkbox" name="java"/><span>Java</span>
</p>
<h2>Укажите пол</h2>
<p>
 <input type="radio" value="man"
  checked name="gender"/><span>мужской</span>
</p>
<p>
<input type="radio" value="woman"
  name="gender"/><span>женский</span>
</p>
</body>
</html>
```

Селектор :checked + span позволяет выбрать элемент, соседний с отмеченным элементом формы, в нашем случае – подписи к элементам, заключённые в тег span. На практике для формирования подписей к флажкам и радиокнопкам следует использовать тег label с атрибутом for.

Псевдокласс :default выбирает стандартный элемент на форме. Как правило, в роли такого элемента выступает кнопка отправки:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Селекторы в CSS3</title>
    <style>
      :default {
        border: 2px solid red;
      }
    </style>
  </head>
  <body>
    <form>
      <input name="login"/>
      <input type="submit" value="Войти" />
    </form>
```

```
   </body>
</html>
```

Псевдоклассы :valid и :invalid стилизуют элементы формы в зависимости от того, проходят они валидацию или нет.

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
      <style>
         input:invalid {
            border: 2px solid red;
         }
         input:valid {
            border: 2px solid green;
         }
      </style>
   </head>
<body>
<form>
 <p><input type="text" name="login"
  placeholder="Введите логин" required /></p>
 <p><input type="password" name="password"
  placeholder="Введите пароль" required /></p>
 <input type="submit" value="Войти" />
</form>
   </body>
</html>
```

В нашем случае обрамление полей ввода логина и пароля станет зелёным, когда в поля формы будет что-либо введено.

Псевдоклассы :in-range и :out-of-range стилизуют элементы формы в зависимости от того, попадает ли их значение в определенный диапазон. Это в первую очередь относится к элементу <input type="number" >

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
      <style>
         :in-range {
            border: 2px solid green;
         }
         :out-of-range {
            border: 2px solid red;
         }
      </style>
   </head>
   <body>
<form>
 <p>
```

```
  <label for="age">Ваш возраст:</label>
  <input type="number" min="1" max="100" value="10"
   id="age" name="age"/>
 </p>
 <input type="submit" value="Отправить" />
</form>
</body>
</html>
```

Здесь атрибуты min и max задают диапазон, в которое должно попадать вводимое в поле значение. В нашем случае рамка поля ввода станет зелёной, если в него будет введено числовое значение от 10 до 100 включительно.

Псевдоклассы :required и :optional стилизуют элемент в зависимости от того, установлен ли у него атрибут required:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы в CSS3</title>
      <style>
         :optional {
            border: 2px solid blue;
         }
         :required {
            border: 2px solid red;
         }
      </style>
   </head>
   <body>
<form>
 <p>
  <label for="login">Логин:</label>
  <input type="text" id="login" name="login" required />
 </p>
 <p>
  <label for="password">Пароль:</label>
   <input type="password" id="password"
    name="password" required />
 </p>
 <p>
  <label for="name">Имя:</label>
   <input type="text" id="name" name="name"/>
 </p>
  <input type="submit" value="Регистрация" />
</form>
</body>
</html>
```

В нашем случае первые два поля ввода будут стилизованы красным обрамлением, а третье – синим.

19. Псевдоэлементы

Псевдоэлементы обладают рядом дополнительных возможностей по выбору элементов web-страницы и отчасти похожи на псевдоклассы. Список доступных псевдоэлементов:

- ::first-letter: позволяет выбрать первую букву из текста;
- ::first-line: стилизует первую строку текста;
- ::before: добавляет сообщение до определенного элемента;
- ::after: добавляет сообщение после определенного элемента;
- ::selection: выбирает выбранные пользователем элементы;

В CSS2 перед псевдоэлементами, как и перед псевдоклассами, ставилось одно двоеточие. В CSS3 для отличия их от псевдоклассов псевдоэлементы стали предваряться двумя двоеточиями. Однако для совместимости с более старыми браузерами, которые не поддерживают CSS3, допустимо использование одного двоеточия: :before.

Стилизуем текст, используя **псевдоэлементы first-letter и first-line**:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Псевдоклассы в CSS3</title>
      <style>
         ::first-letter { color:red; font-size: 25px; }
         ::first-line { font-size: 20px; }
      </style>
   </head>
   <body>
      <p>Но он ничего не видал. Над ним не было ничего уже, кроме неба, — высокого неба.</p>
   </body>
</html>
```

Используем **псевдоэлементы before и after**:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Псевдоклассы в CSS3</title>
<style>
.warning::before{ content: "Важно! "; font-weight: bold; }
.warning::after { content: " Будьте осторожны!";
 font-weight: bold;}
</style>
</head>
<body>
<p><span class="warning">Не пытайтесь засунуть палец в электрическую розетку.</span></p>
</body>
</html>
```

Здесь псевдоэлеметы применяются к элементу с классом warning. Оба псевдоэлемента принимают свойство content, которое хранит вставляемый текст. И также для повышения внимания псевдоэлементы используют выделение текста жирным с помощью свойства font-weight: bold;.

Используем **псевдоэлемент selection** для стилизации выбранных элементов:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Псевдоэлементы в CSS3</title>
      <style>
         ::selection {
            color: white;
            background-color: black;
         }
      </style>
   </head>
<body>
<p>Пседвоэлементы в CSS3 позволяют форматировать текст.</p>
</body>
</html>
```

Результат будет виден при выделении части текста на странице.

20. Селекторы атрибутов

Кроме селекторов элементов мы также можем использовать селекторы их **атрибутов.** Например, у нас есть на web-странице несколько полей input, а нам надо окрасить в красный цвет только текстовые поля. В этом случае мы как раз можем проверять значение атрибута type: если оно имеет значение text, то это текстовое поле, и соответственно его надо окрасить в красный цвет. Определение стиля в этом случае выглядело бы так:

```
input[type="text"]{
   border: 2px solid red;
}
```

После элемента в квадратных скобках идет атрибут и его значение. То есть в данном случае мы говорим, что для текстового поля надо установить границу красного цвета 2 пикселя толщиной сплошной линией.

Например:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы атрибутов в CSS3</title>
      <style>
         input[type="text"]{
            border: 2px solid red;
         }
      </style>
   </head>
   <body>
      <p><input type="text" id="login" /></p>
      <p><input type="password" id="password" /></p>
      <input type="submit" value="Send" />
   </body>
</html>
```

Селекторы атрибутов можно применять не только к элементам, но и классам и идентификаторам. Например:

```
<!DOCTYPE html>
```

```
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Селекторы атрибутов в CSS3</title>
      <style>
         .link[href="http://apple.com"]{
            color: red;
         }
      </style>
   </head>
<body>
<a class="link" href="http://microsoft.com">Microsoft</a> |
<a class="link" href="https://google.com">Google</a> |
<a class="link" href="http://apple.com">Apple</a>
</body>
</html>
```

Специальные символы позволяют конкретизировать значение атрибутов. Например символ ^ позволяет выбрать все атрибуты, которые начинаются на определенный текст. Например, нам надо выбрать все ссылки, которые используют протокол https, то есть ссылка должна начинаться на "https://". В этом случае можно применить следующий селектор:

```
a[href^="https://"]{
   color: red;
}
```

Если значение атрибута должно иметь в конце определенный текст, то для проверки используется символ $. Например, нам надо выбрать все изображения в формате jpg. В этом случае мы можем проверить, оканчивается ли значение атрибута src на текст ".jpg":

```
img[src$=".jpg"]{
   width: 100px;
}
```

Символ "*" (звездочка) позволяет выбрать все элементы с атрибутами, которые в своем значении имеют определенный текст (неважно где - в начале, в середине или в конце):

```
a[href*="microsoft"]{
   color: red;
}
```

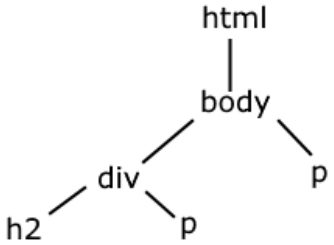
Данный атрибут выберет все ссылки, которые в своем адресе имеют текст "microsoft".

21. Наследование стилей

Для упрощения определения стилей в CSS применяется механизм **наследования стилей.** Этот механизм предполагает, что вложенные элементы могут наследовать стили своих элементов-контейнеров. Например, пусть на web-странице имеются заголовок и параграф, которые должны иметь текст красного цвета. Мы можем отдельно к параграфу и заголовку применить соответствующий стиль, который установит нужный цвет шрифта:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Наследование стилей в CSS3</title>
      <style>
         p {color: red;}
         h2 {color: red;}
      </style>
```

```
    </head>
    <body>
        <h2>Наследование стилей</h2>
        <p>Текст про наследование стилей в CSS 3</p>
    </body>
</html>
```
Однако поскольку и элемент p, и элемент h2 находятся в элементе body, то они наследуют от этого контейнера - элемента body многие стили. И чтобы не дублировать определение стиля, мы могли бы написать так:
```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Наследование стилей в CSS3</title>
        <style>
            body {color: red;}
        </style>
    </head>
    <body>
        <h2>Наследование стилей</h2>
        <p>Текст про наследование стилей в CSS 3</p>
    </body>
</html>
```
В итоге определение стилей стало проще, а результат остался тем же.
Если нам нежелателен унаследованный стиль, то мы его можем переопределить для определенных элементов:
```
body {color: red;}
p {color: green;}
```
При нескольких уровнях вложенности элементы наследуют стили только ближайшего контейнера:
```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Наследование стилей в CSS3</title>
        <style>
            body {color: red;}
            div {color:black;}
        </style>
    </head>
    <body>
        <div>
            <h2>Наследование стилей</h2>
            <p>Текст про наследование стилей в CSS 3</p>
        </div>
        <p>Copyright © MyCorp.com</p>
    </body>
</html>
```
Здесь web-страница имеет следующую структуру:



Для элемента div переопределяется цвет текста. И так как элемент h2 и один из параграфов находятся в элементе div, то они наследуют стиль именно элемента div. Второй параграф находится непосредственно в элементе body и поэтому наследует стиль элемента body. В результате заголовок будет чёрным, а текст абзаца красным.

К наследуемым относятся в основном свойства, определяющие параметры отображения текста: font-size, font-family, font-style, font-weight, color, text-align, text-transform, text-indent, line-height, letter-spacing, word-spacing, white-space, direction и другие.
Также к наследуемым свойствам относятся list-style, cursor, visibility, border-collapse и некоторые другие. Но они используются значительно реже.
Наследуемые свойства можно и нужно задавать через предков, следуя структуре документа.
Например, параметры текста зачастую не меняются в пределах крупных блоков страницы: меню, основного содержания, информационных панелей. Поэтому общие параметры текста (цвет, размер, гарнитура) обычно указывают в стилях этих крупных блоков.
С другой стороны, не ко всем свойствам CSS применяется наследование. Например, свойства, которые представляют отступы (margin, padding) и границы (border) элементов, не наследуются.
Кроме того, браузеры по умолчанию также применяют ряд предустановленных стилей к элементам. Например, заголовки имеют определенную высоту и т.д.

22. Каскадность стилей

Если же к одному и тому же элементу применяется несколько различных стилей, то возникает вопрос, какой же из этих стилей будет в реальности действовать?
В CSS действует **механизм каскадности**, которую можно определить как набор правил, определяющих последовательность применения множества стилей к одному и тому же элементу.
К примеру, у нас определена следующая web-страница:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Каскадность стилей в CSS3</title>
<style>
 .redLink {color: red;}  /* красный цет текста */
 .footer a {font-weight: bold;}  /* выделение жирным */
 a {text-decoration: none;}  /* отмена подчеркивания ссылки */
</style>
</head>
<body>
 <p class="footer">Для просмотра подробной информации
   пройдите по ссылке:
```

```
  <a class="redLink" href="index.php">Основы CSS 3</a>
 </p>
</body>
</html>
```

Здесь определено три стиля и все они применяются к ссылке.

Если к элементу web-страницы применяется несколько стилей, которые не конфликтуют между собой, то браузер объединяет их в один стиль.

Так, в данном случае, все три стиля не конфликтуют между собой, поэтому все эти стили будут применяться к ссылке.

Если же стили конфликтуют между собой, например, определяют разный цвет текста, то в этом случае применяется сложная система правил для вычисления значимости каждого стиля. Все эти правила описаны в спецификации CSS: Calculating a selectors specificity.

Вкратце разберем их.

Для определения стиля к элементу могут применяться различные селекторы, и важность каждого селектора оценивается в баллах. Чем больше у селектора пунктов, тем он важнее, и тем больший приоритет его стили имеют над стилями других селекторов.

- Селекторы тегов имеют важность, оцениваемую в 1 балл;
- Селекторы классов, атрибутов и псевдоклассов оцениваются в 10 баллов;
- Селекторы идентификаторов оцениваются в 100 баллов;
- Встроенные inline-стили (задаваемые через атрибут style) оцениваются в 1000 баллов.

Например:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Каскадность стилей в CSS3</title>
<style>
#index {color: navy;}   /* темно-синий текст */
.redLink {color: red; font-size: 20px;}
  /* красный текст и шрифт 20 пикселей */
a {color: black; font-weight: bold;}
  /* черный цвет и выделение жирным */
</style>
</head>
<body>
 <a id="index" class="redLink" href="index.php">
  Основы CSS 3</a>
</body>
</html>
```

Здесь к ссылке применяется сразу три стиля. Эти стили содержат два не конфликтующих правила:

font-size: 20px;

font-weight: bold;

которые устанавливают высоту шрифта 20 пикселей и выделение ссылки жирным. Так как каждое из этих правил определено только в одном стиле, то в итоге они будут суммироваться и применяться к ссылке без проблем.

Кроме того, все три стиля содержат определение цвета текста, но каждый стиль определяет свой цвет текста. Так как селекторы идентификаторов имеют больший удельный вес, то будет применяться темно-синий цвет, задаваемый селектором:

#index {color: navy;}

Если селектор является **составным**, то происходит сложение баллов всех входящих в селектор подселекторов. Так, рассмотрим следующий пример:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Каскадность стилей в CSS3</title>
<style>
 a {font-size: 18px;}
 .nav li a {color: red;}  /* красный текст */
 #menu a {color: navy;}   /* темно-синий текст */
 .nav .menuItem {color: green;}  /* зеленый текст */
 a.menuItem:not(.newsLink) {color: orange;}
  /* оранжевый текст*/
 div ul li a {color: gray; } /* серый текст */
</style>
</head>
<body>
      <div id="menu">
         <ul class="nav">
            <li><a class="menuItem">Главная</a></li>
            <li><a class="menuItem">Форум</a></li>
            <li><a class="menuItem">Блог</a></li>
            <li><a class="menuItem">О сайте</a></li>
         </ul>
      </div>
   </body>
</html>
```

В стиле определено пять различных селекторов, которые устанавливают цвет ссылок. В итоге браузер выберет селектор #menu a и окрасит ссылки в **тёмно-синий** цвет. Но почему, на каком основании браузер выберет этот селектор?

Рассмотрим, как у нас будут суммироваться баллы по каждому из пяти селекторов:

Селектор	Идентификаторы	Классы	Теги	Сумма
.nav li a	0	1	2	12
#menu a	1	0	1	101
.nav .menuItem	0	2	0	20
a.menuItem:not(.newsLink)	0	2	1	21
div ul li a	0	0	4	4

Итак, мы видим, что для селектора #menu a в колонке "сумма" оказалось больше всего баллов - 101. То есть в нем 1 идентификатор (100 баллов) и один тег(1 балл), которые в сумме дают 101 балл.

К примеру, в селекторе .nav .menuItem два селектора класса, каждый из которых дает 10 баллов, то есть в сумме 20 баллов.

При этом псевдокласс :not в отличие от других псевдокдассов не учитывается, однако учитывается тот селектор, который передается в псевдокласс not.

Правило !important

CSS предоставляет возможность полностью отменить значимость стилей. Для этого в конце стиля указывается значение !important:

```
a {font-size: 18px; color: red !important;}
```

#menu a {color: navy;}
В этом случае вне зависимости от наличия других селекторов с бо́льшим количеством баллов, к ссылкам будет применяться красный цвет, определяемый первым стилем.

23. Единицы измерения в CSS

Все единицы измерения в CSS делятся на *абсолютные* и *относительные*.
Абсолютные единицы измерения привязаны к настоящим физическим размерам и связаны между собой жёсткими пропорциями. Примеры абсолютных единиц измерения:

```
font-size: 1cm;  /* 1 сантиметр  */
font-size: 10mm; /* В 1 сантиметре 10 миллиметров */
font-size: 38px; /* В 1 сантиметре 38 пикселей */
```

Пиксели, px, используют чаще всего, остальные абсолютные единицы почти не применяют.
Относительные единицы измерения описывают значения, которые зависят от других значений. Например, ширина элемента в процентах зависит от ширины родительского элемента, а ширина элемента в em зависит от размера шрифта самого элемента.
К относительным единицам относятся em, rem, vh, vw и некоторые другие, ну и, конечно же, проценты. Каждая из таких единиц решает свой круг задач. Например, проценты используют для "резиновой" вёрстки, а em применяют в вёрстке государственных сайтов с особыми дополнительными требованиями к масштабированию текста.
Перечень всех абсолютных единиц измерения и их соотношений есть в спецификации. Там же, в спецификации, есть перечень всех относительных единиц измерения и описание правил расчёта.

Шпаргалка по основным селекторам

Селектор	CSS	HTML
тега (элемента)	p { color: red; }	Текст применится ко всем абзацам
класса	.red { color: red; }	Текст применится к абзацам с указанным классом
идентификатора	#red { color: red; }	Текст применится к абзацу с указанным идентификатором
псевдокласса	p:hover { color: red; }	Текст применится к абзацам при наведении курсора мыши на содержимое
псевдоэлемента	p::first-letter{ color: red; }	Текст применится к первым буквам абзацев
атрибута	p[align="center"] { color: red; }	Текст применится к центрированным атрибутом align абзацам

Задания для лабораторной работы № 7

1. Разработать валидный стилевой файл для сайта и подключить его к страница, созданным ранее.
2. Предусмотреть в коде селекторы элементов, классы, идентификаторы.

Содержание отчета

1. Титульный лист (включая названия дисциплины, номер лабораторной работы)
2. Цель лабораторной работы

3. Формулировка заданий, текст программы по заданиям (с комментариями), скрин экрана результатов выполнения заданий.

Лабораторная работа № 8
Верстка и макетирование

Цель: научиться осуществлять верстку и макетирование веб-страниц.

Теория

1. Основные понятия

Свойства CSS

1. Цвета в CSS

В CSS широкое распространение находит использование цветов, например, для выделения текста, фона или границ элементов.

Определим красный цвет для фона элемента div:

```
div {
 background-color: red;
}
```

В CSS есть несколько различных свойств, которые в качестве значения требует определенный цвет. Например, за установку цвета текста отвечает свойство color, за установку фона элемента - свойство background-color, а за установку цвета границы - border-color.

Существует также несколько различных способов определения цвета текста.

1. Шестнадцатеричное значение. Оно состоит из отдельных частей, которые кодируют в шестнадцатеричной системе значения интенсивности для красного, зеленого и синего цветов.

Так, в коде #1C4463 первые два символа 1C представляют значение красной компоненты цвета, далее 44 - значение зеленой компоненты цвета и 63 - значение уровня синего цвета. Финальный цвет, который мы видим на web-странице, образуется с помощью смешивания этих значений.

Если каждое из трех двухзначных чисел содержит по два одинаковых символа, то их можно сократить до одного. Например, #5522AA можно сократить до #52A, или #eeeeee до #eee. При этом не столь важно, в каком регистре будут символы.

2. Значение RGB. Значение RGB также представляет последовательно набор значений для красного, зеленого и синего цветов (Red — красный, Green — зеленый, Blue — синий). Значение каждого цвета кодируется тремя числами, которые могут представлять либо процентные соотношения (от 0% до 100% включительно), либо числовые значения от 0 до 255 включительно:

```
background-color: rgb(100%,100%,100%);
```

Здесь каждый цвет имеет значение 100%. И в итоге при смешивании этих значений будет создаваться белый цвет. А при значениях в 0% будет генерироваться черный цвет:

```
background-color: rgb(0%, 0%, 0%);
```

Между 0 и 100% будут находиться все остальные оттенки.

Чаще применяются значения из диапазона от 0 до 255. Например,

```
background-color: rgb(28, 68, 99);
```

3. Значение RGBA. Это то же самое значение RGB плюс компонент прозрачности (Alpha). Компонент прозрачности имеет значение от 0 (полностью прозрачный) до 1 (не прозрачный). Например:

background-color: rgba(28, 68, 99, .6);

4. Значение HSL. HSL представляет собой сокращение: Hue — тон, Saturation — насыщенность и Lightness — осветленность. HSL задает три значения. Первое значение (Hue) угол в круге оттенков, от 0 до 360 градусов. Например, красный = 0 (или 360 при полном обороте круга). Каждый цвет занимает примерно 51°.

Второе значение - Saturation - представляет насыщенность, то указывает, насколько чистым является цвет. Насыщенность определяется в процентах от 0 (полное отсутствие насыщенности) до 100% (яркий, насыщенный цвет).

Третье значение - Lightness - определяет осветленность и указывается в процентах от 0 (полностью черный) до 100 (полностью белый). Для получения чистого цвета применяется значение 50%.

Например:

background-color: hsl(206, 56%, 25%);

Данный цвет является эквивалентом значений #1C4463 и rgb(28, 68, 99)

5. Значение HSLA. Аналогично RGBA здесь к HSL добавляется компонента прозрачности в виде значения от 0 (полностью прозрачный) до 1 (не прозрачный). Например:

background-color: hsl(206, 56%, 25%, .6);

6. Строковые значения. Существует ряд константных строковых значений, например, red (для красного цвета) или green (для зеленого цвета). Так,

color: red;

является эквивалентом

color: #ff0000;

Полный перечень цветов можно найти на странице https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

Существует множество бесплатных онлайн-генераторов цвета, где можно настроить и посмотреть цвет в нужном формате. Например, генератор цвета на mdn.

Прозрачность. Ряд настроек цвета позволяют установить значение для альфа-компоненты, которая отвечает за прозрачность. Но также в CSS есть специальное свойство, которое позволяет установить прозрачность элементов - свойство opacity. В качестве значения оно принимает число от 0 (полностью прозрачный) до 1 (не прозрачный):

```
div{
   width: 100px;
   height: 100px;
   background-color: red;
   opacity: 0.4;
}
```

2. Стилизация шрифтов

Свойство font-family устанавливает **семейство шрифтов**, которое будет использоваться. Например:

```
body {
 font-family: Arial;
}
```

В данном случае устанавливается шрифт Arial.

Шрифт свойства font-family будет работать, только если у пользователя на локальном компьютере имеется такой же шрифт. По этой причине нередко выбираются стандартные шрифты, которые широко распространены, как Arial, Verdana и т.д.

Также нередко применяется указание нескольких шрифтов:

```
body {
 font-family: Arial, Verdana, Helvetica;
}
```

В данном случае основным шрифтом является первый - Arial. Если он на компьютере пользователя не поддерживается, то выбирается второй и т.д.

Если название шрифта состоит из нескольких слов, например, Times New Roman, то название обычно заключается в кавычки:

```
body{
   font-family: "Times New Roman";
}
```

Кроме конкретных стилей также могут использоваться общие универсальные шрифты, задаваемые с помощью значений sans-serif и serif:

```
body{
   font-family: Arial, Verdana, sans-serif;
}
```

Так, если ни Arial, ни Verdana не поддерживаются на компьютере пользователя, то используется sans-serif - универсальный шрифт без засечек.

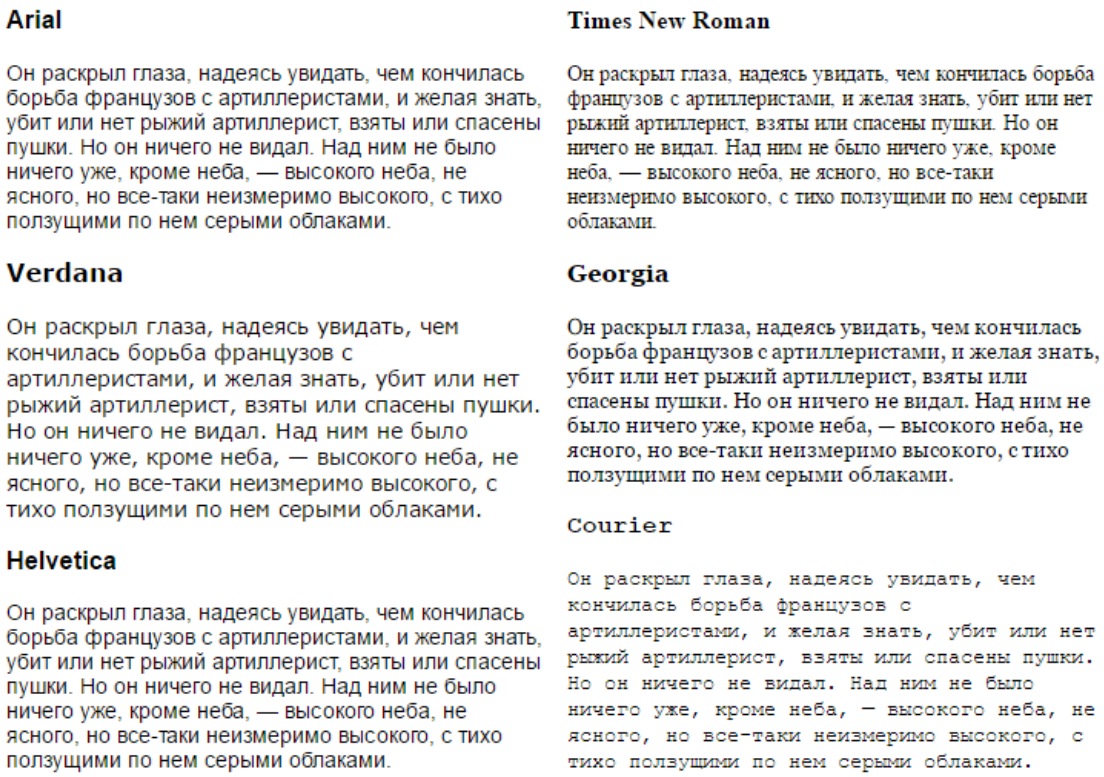
Шрифты с засечками названы так, потому что на на концах основных штрихов имеют небольшие засечки. Считается, что они подходят для больших кусков текста, так как визуально связывают одну букву с другой, делая текст более читабельным. Также шрифты с засечками предпочтительней, если документ предназначен для печати на бумаге.

Распространенные шрифты с засечками: Times, Times New Roman, Georgia, Garamond. Универсальный обобщенный шрифт с засечками представляет значение serif.

Шрифты без засечек, в отличие от шрифтов из первой группы, не имеют засечек по краям букв. Наиболее распространенные шрифты этой группы: Arial, Helvetica, Verdana. Считается, что эти шрифты лучше подходят для заголовков, небольших фрагментов текста и вывода содержимого, не предназначенного для создания бумажных копий. Универсальный обобщенный шрифт без засечек представляет значение sans-serif.

Моноширинный шрифт преимущественно применяется для отображения программного кода (*или другого текста, где существенно выравнивание по символам*) и не предназначен для вывода стандартного текста статей. Своё название эти шрифты получили от того, что каждая буква в таком шрифте имеет одинаковую ширину. Примеры подобных шрифтов: Courier, Courier New, Consolas, Lucida Console. Универсальный обобщенный моноширинный шрифт представляет значение monospace.

Примеры шрифтов:



Свойство font-weight задает **толщину шрифта**. Оно может принимать 9 числовых значений: 100, 200, 300, 400,...900. 100 - очень тонкий шрифт, 900 - очень плотный шрифт.

В реальности чаще для этого свойства используют два значения: normal (нежирный обычный текст) и bold (полужирный шрифт):

font-weight: normal;

font-weight: bold;

Свойство font-style позволяет выделить текст **курсивом**. Для этого используется значение italic:

p {font-style: italic;}

Если надо отменить курсив, то применяется значение normal:

p {font-style: normal;}

Свойство color устанавливает **цвет шрифта**:

```
p {
   color: red;
}
```

Все свойства сразу можно задать универсальным свойством font.

3. Внешние шрифты

Не всегда стандартные встроенные шрифты, как Arial или Verdana, могут быть удобны. Нередко встречается ситуация, когда разработчик хочет воспользоваться возможностями шрифта, которого нет среди встроенных, но он доступен из внешнего файла. Такой шрифт можно подключить с помощью директивы font-face:

```
@font-face {
   font-family: 'Roboto';
   src: url(http://fonts.host.com/fonts/roboto.woff2);
}
```

Свойство font-family задает название шрифта, а свойство src - путь к шрифту.

В данном случае web-страница будет подгружать шрифт, который расположен в интернете по указанному в свойстве src URL-адресу.

В качестве альтернативы мы можем загрузить файл шрифта на локальный компьютер и уже оттуда подгружать его на web-страницу. Как правило, для хранения своих шрифтов на хосте создается вложенная папка fonts:

```
@font-face{
   font-family: 'Roboto';
   src:url('fonts/roboto.ttf');
}
```

После подключения шрифта, его можно использовать в стилях:

```
p{
   font-family: Roboto;
}
```

В данном случе используется шрифт Roboto, созданный компанией Google и определенный в файле в формате woff2. Однако не все браузеры поддерживают данный формат шрифтов.

Существует несколько различных форматов шрифтов:

- TrueType(расширение ttf),
- Open Type (расширение otf),
- Embedded Open Type (расширение eot),
- Web Open Font Format (woff/woff2),
- Scalable Vector Graphic (svg).

Разные браузеры могут поддерживать разные шрифты. И чтобы решить проблему поддержки шрифтов создатели шрифта часто создают сразу несколько форматов. И мы можем сразу эти форматы определить. Например:

```
@font-face {
 font-family:'FontAwesome';
  src: url('https://host.com/fonts/webfont.eot');
  src: url('https://host.com/fonts/webfont.eot?#iefix')
      format('embedded-opentype'),
      url(' https://host.com/fonts/webfont.woff2')
      format('woff2'),
      url(' https://host.com/fonts/webfont.')
      format('woff'),
      url(' https://host.com/fonts/webfont.ttf')
      format('truetype'),
      url(' https://host.com/fonts/webfont.svg')
      format('svg');
}
```

Как и в предыдущем случае свойства font-family и src задают название и путь к шрифту. Но теперь также для совместимости добавляется еще одно свойство src. Второе свойство src устанавливает сразу несколько шрифтов. Первым шрифтом также идет шрифт в формате EOT, но теперь к расширению файла .eot добавляется значение ?#iefix. Это делается для совместимости с версиями Internet Explorer 6–8.

После параметра url следует определение формата шрифта:

```
format('embedded-opentype')
```

Когда браузер начнет загружать web-страницу, на которой таким образом определен шрифт, то он не будет подгружать все шрифты во всех форматах, а загрузит только первый шрифт, который для него окажется понятным.

Упакованный шрифт в формате .zip может содержать не один файл (например, в формате .ttf), а сразу несколько. Дело в том, что каждый шрифт должен определять

отдельный стиль для обычного текста, для текста, выделенного курсивом, для текста, выделенного жирным, для текста, сочетающего выделение жирным и курсивом и т.д.

Чтобы браузер мог автоматически распознавать разные варианты шрифта, к директиве @font-face добавляются свойства font-weight и font-style, которые соответственно устанавливают выделение жирным и выделение курсивом:

```
@font-face {
   font-family: 'Roboto';
   src: url(fonts/Roboto-Regular.ttf);
   font-weight: normal;
   font-style: normal;
}
p {
   font-family: Roboto;
}
```

Поскольку версия шрифта Roboto-Regular.ttf применяется для текста, не выделенного курсивом и жирным, то вместе с ним устанавливаются значения:

```
font-weight: normal;
font-style: normal;
```

4. Высота шрифта

Для установки размера шрифта используется свойство font-size:

```
div {
 font-size: 18px;
}
```

В данном случае высота шрифта составит 18 пикселей. Пиксели представляют наиболее часто используемые единицы измерения. Чтобы задать значение в пикселях, после самого значения идет сокращение "px" (*без пробела после числа!*).

Если к тексту явным образом не применяется высота шрифта, то используются значения браузера по умолчанию. Например, для простого текста в параграфах это 16 пикселей. Это базовый стиль текста.

Базовый стиль для разных элементов текста отличается: если для параграфов это 16 пикселей, то для заголовков h1 это 32 пикселя, для заголовков h2 - 24 пикселя и т.д.

Для измерения шрифта также можно использовать различные единицы измерения, определённые в CSS.

Имеется семь ключевых слов, которые позволяют назначить **размер шрифта относительно базового**:

- medium: базовый размер шрифта браузера (16 пикселей)
- small: 13 пикселей
- x-small: 10 пикселей
- xx-small: 9 пикселей
- large: 18 пикселей
- x-large: 24 пикселя
- xx-large: 32 пикселя

Например:

```
font-size: x-large;
```

Проценты позволяют задать значение относительно базового или унаследованного шрифта. Например:

```
font-size: 150%;
```

В данном случае высота шрифта будет составлять 150% от базового, то есть 16px * 1,5 = 24px

Наследование шрифта **может изменить финальное значение**. Рассмотрим следующую ситуацию:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Шрифты в CSS3</title>

    <style>
      div {font-size: 10px;}
      p {font-size: 150%;}
    </style>
  </head>
  <body>
  <div>
    <p>Однажды в студеную зимнюю пору</p>
  </div>
  </body>
</html>
```

Здесь элемент p наследует от контейнера - блока div шрифт высотой в 10 пикселей. То есть 10 пикселей теперь будет базовым для параграфа.

Далее для элемента p определяется новая высота шрифта в 150%. Это значит, что финальная высота будет равна 10px * 1,5 = 15px.

Единица измерения em во многом эквивалентна процентам. Так, 1em равен 100%, .5em равно 50% и т.д.

5. Форматирование текста

Свойство text-transform изменяет **регистр текста**. Оно может принимать следующие значения:

- capitalize: делает первую букву слова заглавной;
- uppercase: все слово переводится в верхний регистр;
- lowercase: все слово переводится в нижний регистр;
- none: регистр символов слова никак не изменяется

Например:

```
<style>
      p.lowercase {text-transform: lowercase;}
      p.uppercase {text-transform: uppercase;}
      p.capitalize { text-transform: capitalize;}
</style>
```

Свойство text-decoration позволяет добавить к тексту некоторые **дополнительные эффекты**. Это свойство может принимать следующие значения:

- underline: подчеркивает текст;
- overline: надчеркивает текст, проводит верхнюю линию;
- line-through: зачеркивает текст;
- none: к тексту не применяется декорирование

Например:

```
<style>
 p.under {
  text-decoration: underline;
 }
```

```
p.over {
 text-decoration: overline;
}
p.line {
 text-decoration: line-through;
}
p.mixed {
 text-decoration: underline line-through;
}
a.none {
 text-decoration: none;
}
</style>
```

При необходимости мы можем комбинировать значения. Так, в предпоследнем случае применялся стиль:

```
p.mixed { text-decoration: underline line-through; }
```

Два свойства CSS позволяют управлять интервалом между символами и словами текста. Для **межсимвольного интервала** применяется атрибут letter-spacing, а для **интервала между словами** - word-spacing:

```
<style>
 p.smallLetterSpace {
  letter-spacing: -1px;
 }
 p.bigLetterSpace {
  letter-spacing: 1px;
 }
 p.smallWordSpace{
  word-spacing: -1px;
 }
 p.bidWordSpace{
  word-spacing: 1px;
 }
</style>
```

Обычный текст

Над ним не было ничего уже, кроме неба, — высокого неба, не ясного, но все-таки неизмеримо высокого, с тихо ползущими по нем серыми облаками.

letter-spacing: -1px;

Над ним не было ничего уже, кроме неба, — высокого неба, не ясного, но все-таки неизмеримо высокого, с тихо ползущими по нем серыми облаками.

letter-spacing: 1px;

Над ним не было ничего уже, кроме неба, — высокого неба, не ясного, но все-таки неизмеримо высокого, с тихо ползущими по нем серыми облаками.

word-spacing: -1px

Над ним не было ничего уже, кроме неба, — высокого неба, не ясного, но все-таки неизмеримо высокого, с тихо ползущими по нем серыми облаками.

word-spacing: 1px

Над ним не было ничего уже, кроме неба, — высокого неба, не ясного, но все-таки неизмеримо высокого, с тихо ползущими по нем серыми облаками.

С помощью свойства text-shadow можно создать **тени для текста**. Для этого свойства необходимо задать четыре значения: горизонтальное смещение тени относительно текста, вертикальное смещение тени относительно текста, степень размытости тени и цвет отбрасываемой тени. Например:

```
h1{
   text-shadow: 5px 4px 3px #999;
}
```

В данном случае горизонтальное смещение тени относительно букв составляет 5 пикселей, а вертикальное смещение вниз - 4 пикселя. Степень размытости - 3 пикселя, и для тени используется цвет #999.

Если нам надо было бы создать горизонтальное смещение влево, а не вправо, как по умолчанию, то в этом случае надо было бы использовать отрицательное значение. Аналогично для создания вертикального смещения вверх также надо использовать отрицательное значение. Например:

```
h1{
 text-shadow: -5px -4px 3px #999;
}
```

6. Стилизация абзацев

Отдельная группа свойств CSS позволяет стилизовать абзацы текста, например, установить высоту строки или выравнивание текста.

Свойство line-height определяет **межстрочный интервал**. Для его установки можно использовать пиксели, проценты или единицы em. Как правило, применяются либо проценты, либо em. Например:

```
p {
 line-height: 150%;
}
```

Если это свойство не установлено, то по умолчанию обычно используется значение line-height: 120%;

Свойство text-align **выравнивает текст** относительно одной из сторон web-страницы. Оно принимает следующие значения:

- left: текст выравнивается по левой стороне;
- right: текст выравнивается по правой стороне;
- justify: выравнивание по ширине, слова равномерно распределяются по строке;
- center: выравнивание по центру

Например:

```
p {
 text-align: left;
}
```

Свойство text-indent задает **отступ первой строки** абзаца (*красную строку*). Для установки отступа могут применяться стандартные единицы измерения, например, em или пиксели:

```
p {
 text-indent: 35px;
}
```

7. Стилизация списков

CSS предоставляет специальные свойства по стилизации списков. Одним из таких свойств является **list-style-type**. Оно может принимать следующие значения для нумерованных списков:

- decimal: десятичные числа, отсчет идет от 1;
- decimal-leading-zero: десятичные числа, которые предваряются нулем, например, 01, 02, 03, … 98, 99;
- lower-roman: строчные латинские цифры, например, i, ii, iii, iv, v;
- upper-roman: заглавные латинские цифры, например, I, II, III, IV, V…;
- lower-alpha: строчные латинские буквы, например, a, b, c..., z;
- upper-alpha: заглавные латинские буквы, например, A, B, C, … Z;

Для маркированных списков:

- disc: черный диск;
- circle: пустой кружочек;
- square: черный квадратик

Например:

```
ul {
 list-style-type: square;
}
```

Чтобы вообще отключить маркеры у элементов списка, используется значение none:

```
ul {
 list-style-type: none;
}
```

Данное свойство может применяться как ко всему списку, так и к отдельным элементам. Например:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Стилизация списков в CSS3</title>
      <style>
```

```
            .decimal{
               list-style-type: decimal;
            }
            ol{
               list-style-type: lower-roman;
            }
         </style>
      </head>
      <body>
         <ol>
            <li>Элемент 1</li>
            <li class="decimal">Элемент 2</li>
            <li>Элемент 3</li>
            <li>Элемент 4</li>
         </ol>
      </body>
   </html>
```

Браузеры обычно отображают маркеры списка слева от элементов списка. С помощью свойства list-style-position мы можем настроить их **позиционирование**. Это свойство принимает два значения: outside (по умолчанию) и inside (обеспечивает равномерное распределение по ширине).

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Стилизация списков в CSS3</title>
      <style>
         ul.outside{
            list-style-position: outside;
         }
         ul.inside{
            list-style-position: inside;
         }
      </style>
   </head>
   <body>
      <h3>Inside</h3>
<ul class="inside">
<li>Он стал прислушиваться и услыхал звуки приближающегося топота лошадей и звуки голосов...</li>
<li>Он стал прислушиваться и услыхал звуки приближающегося топота лошадей и звуки голосов...</li>
</ul>
<h3>Outside</h3>
<ul class="outside">
<li>Он стал прислушиваться и услыхал звуки приближающегося топота лошадей и звуки голосов...</li>
<li>Он стал прислушиваться и услыхал звуки приближающегося топота лошадей и звуки голосов...</li>
</ul>
</body>
```

```
</html>
```

Свойство list-style-image позволяет **задать в качестве маркера изображение**:

```
<style>
        ul{
           list-style-image:url(phone_touch.png);
        }
</style>
```

Свойство list-style-image в качестве значения принимает путь к изображению url(phone_touch.png), где "phone_touch.png" - это название файла изображения. То есть в данном случае предполагается, что в одной папке с web-страницей находится файл изображения phone_touch.png.

8. Стилизация таблиц

CSS предоставляет ряд свойств, которые помогают стилизовать таблицу:

- border-collapse: устанавливает, как будет стилизоваться граница смежных ячеек;
- border-spacing: устанавливает промежутки между границами смежных ячеек;
- caption-side: устанавливает положение элемента caption;
- empty-cells: задает режим отрисовки для пустых ячеек;
- table-layout: определяет размеры таблицы

Ранее для **установки границы в таблице** широко использовался атрибут border, например:

```
<table border="2px" >
```

Сейчас для стилизации рекомендуется использовать только CSS, поэтому граница также задается через CSS с помощью стандартного свойства border:

```
table {
   border: 1px solid #ccc;  /* граница всей таблицы */
}
tr {
   border: 1px solid #ccc;  /* границы между строками */
}
td, th {
   border: 1px solid #ccc;  /* границы между столбцами */
}
```

При установке границ между столбцами с помощью свойства border-collapse можно установить **общую или раздельную границу между смежными ячейками**:

- collapse: смежные ячейки имеют общую границу;
- separate: смежные ячейки имеют отдельные границы, которые разделяются пространством

Если смежные ячейки имеют раздельные границы, то с помощью свойства border-spacing можно установить **пространство между границами**:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Стилизация таблиц в CSS3</title>
      <style>
      table {
```

```
            border: 1px solid #ccc;
            border-spacing: 3px;
        }
        td, th{
            border: solid 1px #ccc;
        }
        .collapsed{
            border-collapse: collapse;
        }
        .separated{
            border-collapse: separate;
        }
        </style>
    </head>
    <body>
<h3>Collapse</h3>
<table class="collapsed">
<tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
<tr><td>Lumia 950</td><td>Microsoft</td><td>29900</td></tr>
<tr><td>iPhone 6S</td><td>Apple</td><td>52900</td></tr>
<tr><td>Nexus 6P</td><td>Huawei</td><td>49000</td></tr>
</table>
<h3>Separate</h3>
<table class="separated">
<tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
<tr><td>G 5</td><td>LG</td><td>44900</td></tr>
<tr><td>HTC 10</td><td>HTC</td><td>49900</td></tr>
<tr><td>Nexus 5X</td><td>Google/LG</td><td>25000</td></tr>
</table>
</body>
</html>
```

Свойство empty-cells позволяет **стилизовать пустые ячейки** с помощью одного из следующих значений:

- show: пустые ячейки отображаются, значение по умолчанию;
- hide: пустые ячейки не отображаются

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Стилизация таблиц в CSS3</title>
        <style>
        table {
            border: 1px solid #ccc;
            border-spacing: 3px;
        }
        td, th{
            border: solid 1px #ccc;
        }
        .hidden-empty-cells{
            empty-cells: hide;
        }
```

```
        </style>
    </head>
    <body>
<h3>Show</h3>
<table>
<tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
<tr><td>Lumia 950</td><td>Microsoft</td><td>29900</td></tr>
<tr><td>iPhone 6S</td><td></td><td></td></tr>
<tr><td>Nexus 6P</td><td>Huawei</td><td>49000</td></tr>
</table>
<h3>Hide</h3>
<table class="hidden-empty-cells">
<tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
<tr><td>G 5</td><td>LG</td><td>44900</td></tr>
<tr><td>HTC 10</td><td></td><td></td></tr>
<tr><td>Nexus 5X</td><td>Google/LG</td><td>25000</td></tr>
</table>
</body>
</html>
```

Свойство caption-side управляет **позицией заголовка** и может принимать следующие значения:

- top: позиционирование заголовка вверху (значение по умолчанию);
- bottom: позиционирование заголовка внизу

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Стилизация таблиц в CSS3</title>
        <style>
        table {
            border: 1px solid #ccc;
            border-spacing: 3px;
        }

        caption {

            font-weight: bold;
        }
        td, th{
            border: solid 1px #ccc;
        }
        .captionBottom{
            caption-side: bottom;
        }
        </style>
    </head>
    <body>
<h3>Top</h3>
<table>
<caption>Флагманы 2015 года</caption>
<tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
```

```
<tr><td>Lumia 950</td><td>Microsoft</td><td>29900</td></tr>
<tr><td>iPhone 6S</td><td>Apple</td><td>52900</td></tr>
<tr><td>Nexus 6P</td><td>Huawei</td><td>49000</td></tr>
</table>
<h3>Bottom</h3>
<table class="captionBottom">
<caption>Новинки 2016 года</caption>
<tr><th>Модель</th><th>Производитель</th><th>Цена</th></tr>
<tr><td>G 5</td><td>LG</td><td>44900</td></tr>
<tr><td>HTC 10</td><td>HTC</td><td>49900</td></tr>
<tr><td>iPhone SE</td><td>Apple</td><td>37000</td></tr>
</table>
</body>
</html>
```

С помощью свойства table-layout можно **управлять размером таблицы**. По умолчанию это свойство имеет значение auto, при котором браузер устанавливает ширину столбцов таблицы автоматически, исходя из ширины самой широкой ячейки в столбце. А из ширины отдельных столбцов складывается ширина всей таблицы.

Однако с помощью другого значения - fixed можно установить фиксированную ширину:

```
table {
   border: 1px solid #ccc;
   border-spacing: 3px;
   table-layout: fixed;
   width:350px;
}
```

Как правило, содержимое ячеек таблицы **выравнивается** по центру ячейки. Но с помощью свойства vertical-align это поведение можно переопределить. Это свойство принимает следующие значения:

- top: выравнивание содержимого по верху ячейки;
- baseline: выравнивание первой строки текста по верху ячейки;
- middle: выравнивание по центру (значение по умолчанию);
- bottom: выравнивание по низу

Свойство vertical-align применяется только к элементам <th> и <td>:

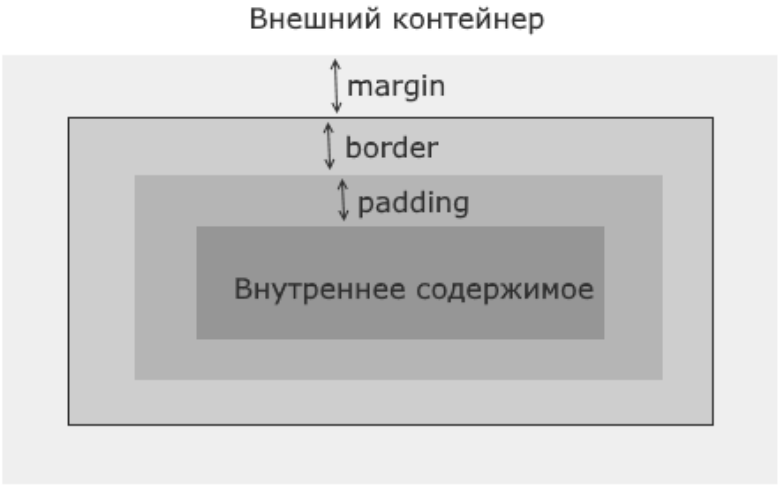
```
td, th{
   border: solid 1px #ccc;
   vertical-align: bottom;
   height: 30px;
}
```

Для **горизонтального выравнивания** текста в ячейках таблицы применим атрибут text-align (см. п. 7.6).

9. Блочная модель

Для браузера элементы страницы представляют собой контейнеры или блоки. Такие блоки могут иметь различное содержимое - текст, изображения, списки, таблицы и другие элементы. Внутренние элементы блоков сами выступают в качестве блоков.

Схематично блочную модель можно представить следующим образом:



Пусть элемент расположен в каком-нибудь внешнем контейнере. Это может быть элемент body, div и так далее. От других элементов он отделяется некоторым расстоянием - **внешним отступом**, которое описывается свойством CSS margin. То есть, свойство margin определяет расстояние от границы текущего элемента до других соседних элементов или до границ внешнего контейнера.

Далее начинается сам элемент. И в начале идет его **граница**, которая в CSS описывается свойством border.

После границы идет **внутренний отступ**, который в CSS описывается свойством padding. Внутренний отступ определяет расстояние от границы элемента до внутреннего содержимого.

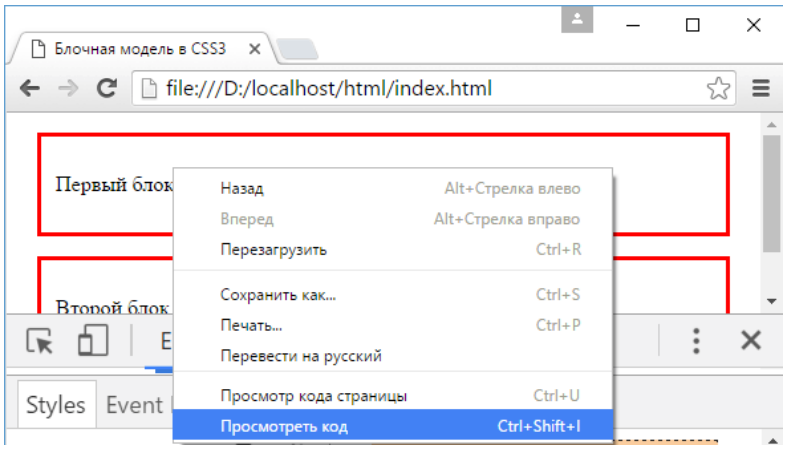
Далее идет внутреннее содержимое, которое также реализует ту же блочную модель и также может состоять из других элементов, которые имеют внешние и внутренние отступы и границу.

Например, определим следующую страницу:

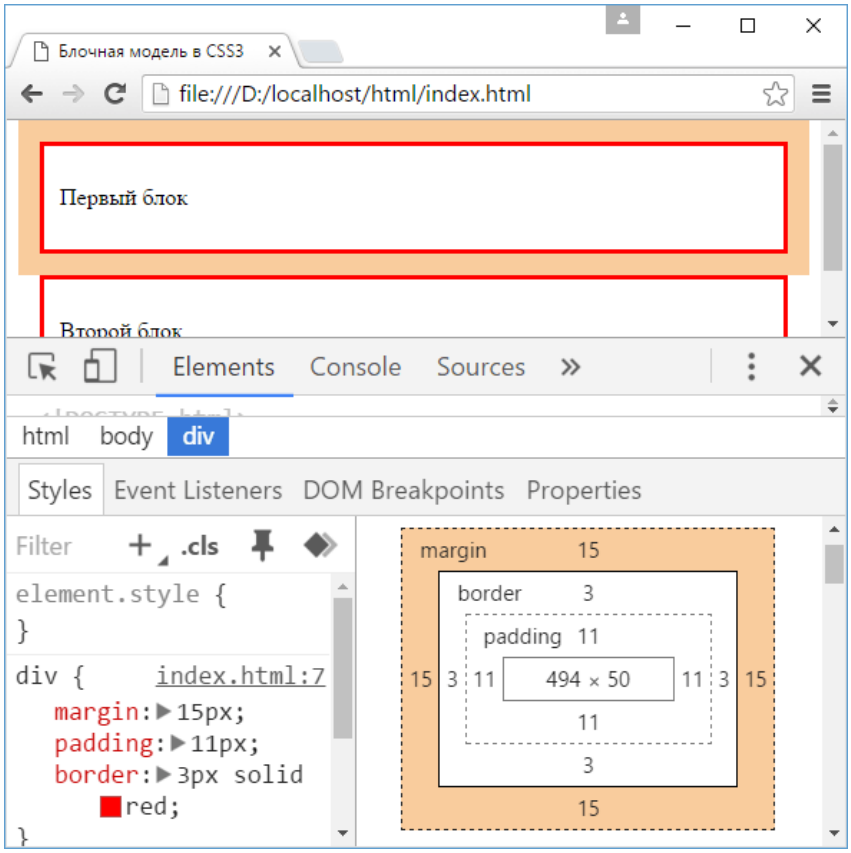
```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Блочная модель в CSS3</title>
<style>
div {
 margin: 15px; /* внешний отступ */
 padding: 11px; /* внутренний отступ */
 border: 3px solid red;
  /* границы шириной в 3 пикселя сплошной красной линией */
}
</style>
</head>
<body>
 <div>
  <p>Первый блок</p>
 </div>
 <div>
  <p>Второй блок</p>
 </div>
</body>
```

</html>

После запуска страницы в браузере мы можем посмотреть блочную модель конкретных элементов. Для этого надо нажать на нужный элемент правой кнопкой мыши и открывающемся контекстном меню выбрать пункт, который позволяет просмотреть исходный код элемента. Для разных браузеров этот пункт может называться по разному. К примеру в Google Chrome это "Посмотреть код":



В Mozilla Firefox аналогичный пункт меню называется "Исследовать элемент". По выбору данного пункта браузер откроет панель, где будет показан код элемента его стили и блочная модель:



В этой модели мы можем увидеть, как задаются отступы элемента, его граница, посмотреть отступы от других элементов и при необходимости динамически поменять значения их стилей.

Если мы явным образом не указываем значения свойств margin, padding и border, то браузер применяет предустановленные значения.

10. Внешние отступы (margin)

Свойство margin определяет отступ элемента от других элементов или границы внешнего контейнера. Существуют специальные свойства CSS для задания отступов для каждой стороны:

- margin-top: отступ сверху;
- margin-bottom: отступ снизу;
- margin-left: отступ слева;
- margin-right: отступ справа

Например:

```
<style>
div {
 margin-top: 30px; /* отступ сверху */
 margin-left: 25px; /* отступ слева */
 margin-right: 20px; /* отступ справа */
 margin-bottom: 15px; /* отступ снизу */
 border: 3px solid red; /* граница */
}
</style>
```

Можно вместо четырех свойств задать одно:

```
div{
 margin: 30px 20px 15px 25px;
 border: 3px solid red;
}
```

Свойство задается в формате:

margin: отступ_сверху отступ_справа отступ_снизу отступ_слева;

Если значения для всех четырех отступов совпадает, то мы можем указать только одно значение:

```
div {
 margin: 25px;
}
```

В этом случае для всех четырех отступов будет использоваться 25 пикселей.

Если казано два значения, первое из них означает отступы по вертикали, а второе – по горизонтали:

```
div {
 margin: 10px 25px;
}
```

Для установки отступов можно использовать точные значения в пикселях (px) или em, либо процентные отношения, либо значение auto (автоматическая установка отступов).

Например:

margin-left: 2em;

Значение 2em определяет расстояние, которое в два раза больше размера шрифта элемента.

При использовании процентов браузеры вычисляют размер отступов на основе ширины элемента-контейнера, в который заключен стилизуемый элемент.

В то же время, если несколько элементов соприкасаются, браузер выбирает наибольший отступ элемента, который затем и используется.

11. Внутренние отступы (padding)

Свойство padding задает внутренние отступы от границы элемента до его внутреннего содержимого. Как и для margin, в CSS имеются четыре свойства, которые устанавливают отступы для каждой из сторон:

- padding-top: отступ сверху;
- padding-bottom: отступ снизу;
- padding-left: отступ слева;
- padding-right: отступ справа

Например:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Блочная модель в CSS3</title>
    <style>
      div.outer{
        margin: 25px;
        padding-top:30px;
        padding-right: 25px;
        padding-bottom: 35px;
        padding-left: 28px;
        border: 2px solid red;
      }
      div.inner{
        height: 50px;
        background-color:blue;
      }
    </style>
  </head>
  <body>
    <div class="outer">
      <div class="inner"></div>
    </div>
  </body>
</html>
```

Для установки значения отступов, как и в margin, могут применяться либо конкретные значения в пикселях, так и процентные значения (относительно размеров элементов).

Для записи отступов также можно использовать сокращенную запись:

padding: отступ_сверху отступ_справа отступ_снизу отступ_слева;

Например:

```
div.outer{
  margin: 25px;
  padding: 30px 25px 35px 28px;
  border: 2px solid red;
}
```

Если все четыре значения совпадают, то можно писать указать только одно значение для всех отступов:

```
div.outer{
  margin: 25px;
  padding: 30px;
  border: 2px solid red;
```

```
}
```

12. Границы (border)

Граница отделяется элемент от внешнего по отношению к нему содержимого. При этом граница является частью элемента.

Для настройки границы могут использоваться сразу несколько свойств:

- border-width: устанавливает ширину границы;
- border-style: задает стиль линии границы;
- border-color: устанавливает цвет границы

Свойство border-width (**ширина границы**) может принимать следующие типы значений:

- Значения в единицах измерения, таких как em, px или cm: border-width: 2px;
- Одно из константных значений thin (тонкая граница - 1px), medium (средняя по ширине - 3px), thick (толстая - 5px): border-width: medium;

Свойство border-color (**цвет границы**) в качестве значения принимает цвет CSS:
border-color: red;

Свойство border-style оформляет **тип линии границы** и может принимать одно из следующих значений:

- none: граница отсутствует;
- solid: граница в виде обычной линии;
- dashed: штриховая линия;
- dotted: линия в виде последовательности точек;
- double: граница в виде двух параллельных линий;
- groove: граница имеет трехмерный эффект;
- inset: граница визуально вдавливается внутрь;
- outset: граница визуально выступает наружу;
- ridge: граница реализует трехмерный эффект

Например:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Блочная модель в CSS3</title>
      <style>
         div{
            width: 100px;
            height:100px;
            border-style: solid;
            border-color: red;
            border-width: 2px;
         }
      </style>
   </head>
   <body>
      <div></div>
   </body>
```

```
</html>
```

При необходимости мы можем определить **цвет, стиль и ширину границы для каждой из сторон**, используя следующие свойства:

```
/* для верхней границы */
border-top-width
border-top-style
border-top-color
/* для нижней границы */
border-bottom-width
border-bottom-style
border-bottom-color

/* для левой границы */
border-left-width
border-left-style
border-left-color
/* для правой границы */
border-right-width
border-right-style
border-right-color
```

Вместо установки по отдельности **цвета, стиля и ширины границы** мы можем использовать одно свойство - border:

```
border: ширина стиль цвет
Например:
border: 2px solid red;
```

Для установки границы **для отдельных сторон** можно использовать одно из свойств:

```
border-top
border-bottom
border-left
border-right
Их использование аналогично:
border-top: 2px solid red;
```

Свойство border-radius позволяет **округлить границу**. Это свойство принимает значение радиуса в пикселях или единицах em.

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Блочная модель в CSS3</title>
      <style>
         div{
            width: 100px;
            height:100px;
            border: 2px solid red;
            border-radius: 30px;
         }
      </style>
```

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

Теперь каждый угол будет скругляться по радиусу в 30 пикселей.

Так как у элемента может быть максимально четыре угла, то мы можем указать четыре значения для установки радиуса у каждого углов:

```
border-radius: 15px 30px 5px 40px;
```

Вместо общей установки радиусов для всех углов, можно их устанавливать по отдельности. Так, предыдущее значение border-radius можно переписать следующим образом:

```
border-top-left-radius: 15px;
 /* радиус для верхнего левого угла */
border-top-right-radius: 30px;
 /* радиус для верхнего правого угла */
border-bottom-right-radius: 5px;
 /* радиус для нижнего правого угла */
border-bottom-left-radius: 40px;
 /* радиус для нижнего левого угла */
```

Также border-radius поддерживает возможность создания **эллиптических углов.** То есть угол не просто скругляется, а использует два радиуса, образуя в итоге душу эллипса:

```
border-radius: 40px/20px;
```

В данном случае полагается, что радиус по оси X будет иметь значение 40 пикселей, а по оси Y - 20 пикселей.

13. Размеры элементов

Размеры элементов задаются с помощью свойств width (ширина) и height (высота).

Значение по умолчанию для этих свойств - auto, то есть браузер сам определяет ширину и высоту элемента. Можно также явно задать размеры с помощью единиц измерения (пикселей, em) или с помощью процентов:

```
width: 150px;
width: 75%;
height: 15em;
```

Пиксели определяют точные ширину и высоту. Единица измерения em зависит от высоты шрифта в элементе. Если размер шрифта элемента, к примеру, равен 16 пикселей, то 1em для этого элемента будет равен 16 пикселям. То есть если у элемента установить ширину в 15em, то фактически она составит 15 * 16 = 230 пикселей. Если же у элемента не определён размер шрифта, он будет взят из унаследованных параметров или значений по умолчанию.

Процентные значения для свойства width вычисляются на основании ширины элемента-контейнера. Если, к примеру, ширина элемента body на web-странице составляет 1000 пикселей, а вложенный в него элемент имеет ширину 75%, то фактическая ширина этого блока составляет 1000 * 0.75 = 750 пикселей. Если пользователь изменит размер окна браузера, то ширина элемента body и соответственно ширина вложенного в него блока div тоже изменится.

Процентные значения для свойства height работают аналогично свойству width, только теперь высота вычисляется по высоте элемента-контейнера.

Например:

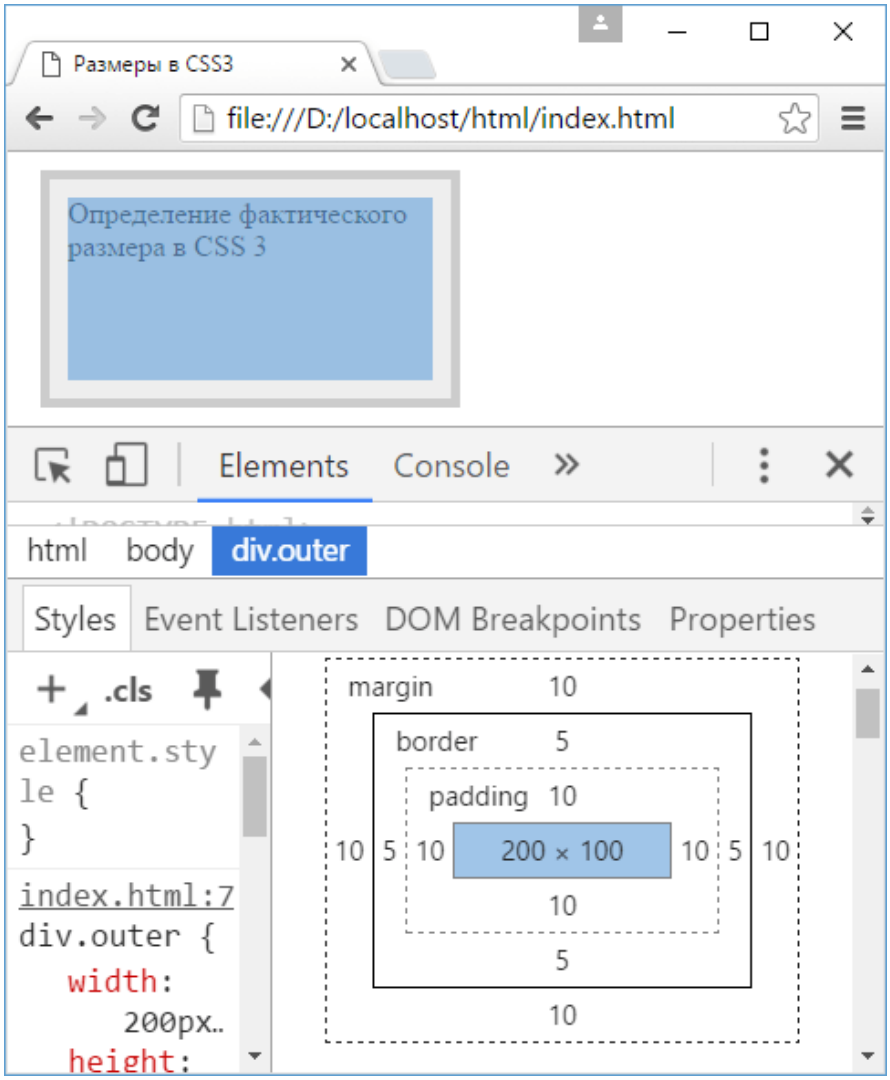
```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Размеры в CSS3</title>
      <style>
      div.outer{
         width: 75%;
         height: 200px;
         margin: 10px;
         border: 1px solid #ccc;
         background-color: #eee;
      }
      div.inner{

         width: 80%;
         height: 80%;
         margin: auto;
         border: 1px solid red;
         background-color: blue;
      }
      </style>
   </head>
   <body>
      <div class="outer">
         <div class="inner"></div>
      </div>
   </body>
</html>
```

Нужно учитывать, что фактические размеры элемента могут **отличаться** от тех, которые установлены в свойствах width и height. Например:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Размеры в CSS3</title>
      <style>
      div.outer{
         width: 200px;
         height: 100px;
         margin: 10px;
         padding: 10px;
         border: 5px solid #ccc;
         background-color: #eee;
      }
      </style>
   </head>
   <body>
      <div class="outer">
         Определение фактического размера в CSS 3
      </div>
```

```
    </body>
</html>
```



Как видно на скриншоте, в реальности значение свойства width (200px) - определяет только ширину внутреннего содержимого элемента, а под блок самого элемента будет выделяться пространство, ширина которого равна ширине внутреннего содержимого (свойство width) + внутренние отступы (свойство padding) + ширина границы (свойство border-width) + внешние отступы (свойство margin). То есть элемент будет иметь ширину в 230 пикселей, а ширина блока элемента с учетом внешних отступов составит 250 пикселей.

Подобные расчеты следует учитывать при определении размеров элементов.

С помощью дополнительного набора свойств можно установить минимальные и максимальные размеры:

- min-width: минимальная ширина;
- max-width: максимальная ширина;
- min-height: минимальная высота;
- max-height: максимальная высота

```
min-width: 200px;
width:50%;
max-width: 300px;
```

В данном случае ширина элемента равна 50% ширины элемента-контейнера, однако при этом не может быть меньше 200 пикселей и больше 300 пикселей.

Свойство box-sizing позволяет **переопределить установленные размеры** элементов. Оно может принимать одно из следующих значений:

- content-box: значение свойства по умолчанию, при котором браузер для определения реальных ширины и высоты элементов добавляет к значениям свойств width и height ширину границы и внутренние отступы
- padding-box: указывает браузеру, что ширина и высота элемента должны включать внутренние отступы как часть своего значения. Например, пусть у нас есть следующий стиль:

width: 200px;
height: 100px;
margin: 10px;
padding: 10px;
border: 5px solid #ccc;
background-color: #eee;
box-sizing: padding-box;

Здесь реальная ширина внутреннего содержимого блока будет равна 200px (width) - 10px (padding-left) - 10px (padding-right) = 180px.
Не все современные браузеры поддерживают данное свойство.

- border-box: указывает браузеру, что ширина и высота элемента должны включать внутренние отступы и границы как часть своего значения. Например, пусть у нас есть следующий стиль:

width: 200px;
height: 100px;
margin: 10px;
padding: 10px;
border: 5px solid #ccc;
background-color: #eee;
box-sizing: border-box;

Здесь реальная ширина внутреннего содержимого блока будет равна 200px (width) - 10px (padding-left) - 10px (padding-right) - 5px (border-left-width) - 5px (border-right-width) = 170px.

Определим два блока, которые отличаются только значением свойства box-sizing:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Размеры в CSS3</title>
    <style>
    div{
      width: 200px;
      height: 100px;
      margin: 10px;
      padding: 10px;
      border: 5px solid #ccc;
      background-color: #eee;
    }
    div.outer1{
      box-sizing: content-box;
    }
    div.outer2{
      box-sizing: border-box;
```

```
        }
        </style>
    </head>
    <body>
        <div class="outer1">
            Определение фактического размера в CSS 3
        </div>
        <div class="outer2">
            Определение фактического размера в CSS 3
        </div>
    </body>
</html>
```

В первом случае при определении размеров блока к свойствам width и height будут добавляться толщина границы, а также внутренние и внешние отступы, поэтому первый блок будет иметь большие размеры.

14. Фон элемента

Фон элемента описывается в CSS свойством background. Фактически это свойство представляет сокращение набора следующих свойств CSS:

- background-color: устанавливает цвет фона background-color: #ff0507;
- background-image: в качестве фона устанавливается изображение background-image: url(image.png);

Это свойство принимает одно значение: ключевое слово url, после которого в скобках идет путь к файлу изображения. В данном случае имеется в виду, что в одной папке рядом с web-страницей находится файл image.png.
Также можно использовать абсолютные адреса URL, например: background-image: url(http://localhost.com/someimage.png);
Либо можно использовать относительные адреса - относительно HTML-документа или корневого каталога сайта: background-image: url(../images/someimage.png);
/* путь относительно html-документа */

- background-repeat: устанавливает режим повторения фонового изображения по всей поверхности элемента
- background-size: устанавливает размер фонового изображения
- background-position: указывает позицию фонового изображения
- background-attachment: устанавливает стиль прикрепления фонового изображения к элементу
- background-clip: определяет область, которая вырезается из изображения и используется в качестве фона
- background-origin: устанавливает начальную позицию фонового изображения

Например:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Блочная модель в CSS3</title>
        <style>
        div{
```

```
            width: 250px;
            height: 200px;
            margin: 10px;
        }

        .colored{
            background-color: #ff0507;
        }

        .imaged{
            background-image: url(image.png);
        }
        </style>
    </head>
    <body>
        <div class="colored">Первый блок</div>
        <div class="imaged">Второй блок</div>
    </body>
</html>
```

Первый блок окрашен в оттенок красного цвета, а второй блок устанавливает в качестве фона изображение. Все содержимое блока накладывается поверх фона.

CSS должным образом масштабирует изображение, чтобы наиболее оптимально вписать его в пространство элемента. Однако в связи с масштабированием изображение может не полностью покрывать поверхность элемента, и поэтому для полного покрытия автоматически CSS начинает повторять изображение.

С помощью свойства background-repeat можно изменить **механизм повторения**. Оно может принимать следующие значения:

- repeat-x: повторение по горизонтали;
- repeat-y: повторение по вертикали;
- repeat: повторение по обеим сторонам (действие по умолчанию);
- space: изображение повторяется для заполнения всей поверхности элемента, но без создания фрагментов;
- round: изображение должным образом масштабируется для полного заполнения всего пространства;
- no-repeat: изображение не повторяется

Например:

```
div{
    width: 200px;
    height: 150px;
    background-image: url(image.png);
    background-repeat: round;
}
```

Свойство background-size позволяет установить размер **фонового изображения**. Для установки размера можно использовать либо единицы измерения, например, пиксели, либо проценты, либо одно из предустановленных значений:

- contain: масштабирует изображение по наибольшей стороне, сохраняя аспектное отношение;
- cover: масштабирует изображение по наименьшей стороне, сохраняя аспектное отношение;

- auto: значение по умолчанию, изображение отображается в полный размер

Если нужно масштабировать изображение таким образом, чтобы оно оптимальнее было вписано в фон, то для обеих настроек можно установить значение 100%:

background-size: 100% 100%;

Если задаются точные размеры, то вначале указывается ширина, а потом высота изображения:

background-size: 200px 150px; /* ширина 200 пикселей, высота 150 пикселей */

Можно задать точное значение для одного измерения - ширины или высоты, а для другого задать автоматические размеры, чтобы браузер сам выводил точные значения:

background-size: 200px auto; /* ширина 200 пикселей, автоматическая высота */

Например:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Блочная модель в CSS3</title>
    <style>
    div{
      width: 200px;
      height: 150px;
      margin: 10px;

      border: black solid 1px;
      background-image: url(image.png);

    }
    .imaged1{

      background-size: cover;
    }
    .imaged2{

      background-size: 140px 110px;
    }
    </style>
  </head>
  <body>
    <div class="imaged1"></div>
    <div class="imaged2"></div>
  </body>
</html>
```

Во втором случае изображение будет масштабироваться до размеров 140x110. Поскольку у нас еще остается место на элементе, то по умолчанию изображение будет повторяться для заполнения всей поверхности.

Свойство background-position управляет **позицией фонового изображения** внутри элемента. Оно может принимать отступы от верхнего левого угла элемента в единицах измерения, например, в пикселях в следующем формате:

background-position: отступ_по_оси_X отступ_по_оси_Y;

Например:

```
  <style>
    div{
```

```
            width: 300px;
            height: 250px;
            margin: 10px;

            border: 1px solid #ccc;
            background-color: #eee;
            background-image: url(image.png);
            background-repeat: no-repeat;
            background-position: 20px 15px;
        }
</style>
```

Кроме того, данное свойство может принимать одно из следующих значений:

- top: выравнивание по верхнему краю элемента;
- left: выравнивание по левому краю элемента;
- right: выравнивание по правому краю элемента;
- bottom: выравнивание по нижнему краю элемента;
- center: изображение располагается по центру элемента

Например:

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

Здесь изображение выравнивается по верху и правому краю, то есть будет располагаться в правом верхнем углу элемента.

Свойство background-attachment определяет, как **фоновое изображение будет прикреплено к элементу**. Это свойство может принимать следующие значения:

- fixed: фон элемента фиксирован вне зависимости от прокрутки внутри элемента;
- local: по мере прокрутки внутри элемента фон изменяется;
- scroll: фон фиксирован и не меняется при прокрутке, но в отличие от fixed несколько элементов могут использовать свой фон, тогда как при fixed создается один фон для всех элементов

Например:

```
div{
    width: 300px;
    height: 250px;
    overflow:scroll;    /* добавляем прокрутку  */
    border: 1px solid #ccc;
    background-image: url(image.png);
    background-size: 512px 384px;
    background-attachment: scroll;
    background-repeat: no-repeat;
}
```

Свойство background-origin указывает **позицию на изображении, с которой будет начинаться фоновое изображение** для элемента. Оно может принимать следующие значения:

- border-box: фон у элемента устанавливается начиная с его внешней границы, определяемой свойством border;
- padding-box: фон устанавливается с учетом внутренних отступов;
- content-box: фон устанавливается по содержимому элемента

Используем все три значения:

```
<!DOCTYPE html>
<html lang="ru">
```

```
<head>
  <meta charset="utf-8">
  <title>Блочная модель в CSS3</title>
  <style>
  div{
    width: 200px;
    height: 200px;
    margin: 10px;
    display: inline-block;
     /* располагаем блоки в ряд */
    color: #eee;

    padding:15px;
    border: 15px solid rgba(23,23,23,0.2);

    background-image: url(image.jpg);
    background-size: cover;
    background-repeat: no-repeat;
  }
  .borderBox {background-origin: border-box;}
  .paddingBox {background-origin: padding-box;}
  .contentBox {background-origin: content-box;}
  </style>
</head>
<body>
<div class="borderBox">          Текст       </div>
<div class="paddingBox">          Текст       </div>
<div  class="contentBox">          Текст       </div>
</body>
</html>
```

Свойство background-clip определяет, **какая часть изображения используется для фона**. Он принимает те же значения:

- border-box: изображение обрезается по границам элемента;
- padding-box: из изображения исключается та часть, которая находится под границами элемента;
- content-box: изображение обрезается по содержимому с учетом внутренних отступов

Например, к предыдущей разметке можно применить следующие стили:

```
div{
  width: 200px;
  height: 200px;
  margin: 10px;
  display: inline-block;

  color: #eee;
  padding:15px;
  border: 15px solid rgba(23,23,23,0.1);

  background-image: url(image.jpg);
  background-size: cover;
  background-repeat: no-repeat;
```

```
}
.borderBox{background-clip: border-box;}
.paddingBox{background-clip: padding-box;}
.contentBox{background-clip: content-box;}
```

Свойство background является сокращением всех ранее рассмотренных свойств CSS в формате:

background: <background-color> <background-position> <background-size> <background-repeat> <background-origin> <background-clip> <background-attachment> <background-image>

Например, если у нас есть следующий набор свойств:

```
background-image: url(image.jpg);
background-color: #eee;
background-repeat: no-repeat;
background-clip: border-box;
background-origin: border-box;
background-attachment: local;
```

То мы их можем сократить следующим образом:

```
background: #eee no-repeat border-box local url(image.jpg);
```

15. Создание тени у элемента

Свойство box-shadow позволяет создать у элемента тень. Это свойство может принимать сразу несколько значений:

box-shadow: hoffset voffset blur spread color inset

- hoffset: горизонтальное смещение тени относительно элемента. При положительном значении тень смещается вправо, а при отрицательном – влево;
- voffset: вертикальное смещение тени относительно элемента. При положительном значении тень смещается вниз, а при отрицательном – вверх;
- blur: необязательное значение, которое определяет радиус размытия тени. Чем больше это значение, тем более размытыми будут края тени. По умолчанию имеет значение 0;
- spread: необязательное значение, которое определяет направление тени. Положительное значение распространяет тень вовне во всех направлениях от элемента, а отрицательное значение направляет тень к элементу;
- color: необязательное значение, которое устанавливает цвет тени;
- inset: необязательное значение, которое заставляет рисовать тент внутри блока элемента

Например:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Тени в CSS3</title>
      <style>
      div{
         width: 128px;
         height: 96px;
         margin: 20px;
         border: 1px solid #ccc;
         background-color: #eee;
         box-shadow: 10px 4px 10px 3px #888;
```

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

Через запятую можно определить несколько различных теней:

```
box-shadow: 5px 3px 8px 3px #faa,
        10px 4px 10px 3px #888 inset;
```

16. Контуры элементов

Концепция контуров похожа на использование границ у элементов. Не стоит путать или заменять границы контурами, они имеют разное значение. Контуры полезны тем, что позволяют выделить элемент, чтобы привлечь к нему внимание в какой-то ситуации. Контуры располагаются вне элемента сразу за его границами.

Контур в CSS 3 представлен свойством outline, хотя данное свойство является сокращением следующих свойств:

- outline-color: цвет контура;
- outline-offset: смещение контура;
- outline-style: стиль контура. Свойство принимает те же значения, что и border-style:
  - none: контур отсутствует;
  - solid: контур в виде обычной линии;
  - dashed: штриховая линия;
  - dotted: линия в виде последовательности точек;
  - double: контур в виде двух параллельных линий;
- outline-width: толщина контура

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Контур в CSS3</title>
    <style>
    div{
      width: 128px;
      height: 96px;
      margin: 20px;
      border: 1px solid #ccc;
      background-color: #eee;
      outline-color: red;
      outline-style: dashed;
      outline-width: 2px;
    }
    </style>
  </head>
  <body>
    <div>Контур в CSS3</div>
  </body>
</html>
```

С помощью свойства outline данное определение контура можно сократить следующим образом:

outline: red dashed 2px;

17. Обтекание элементов

Как правило, все блоки и элементы на web-странице в браузере появляются в том порядке, в каком они определены в коде HTML. Однако CSS предоставляет специальное свойство float, которое позволяет установить обтекание элементов, благодаря чему мы можем создать более интересные и разнообразные по своему дизайну web-страницы.

Это свойство может принимать одно из следующих значений:

- left: элемент перемещается влево, а все содержимое, которое идет ниже его, обтекает правый край элемента;
- right: элемент перемещается вправо;
- none: отменяет обтекание и возвращает объект в его обычную позицию

При применении свойства float для стилизуемых элементов, кроме элемента img, рекомендуется установить свойство width.

Итак, представим, что нам надо на странице вывести слева от основного текста изображение, справа должен быть сайдбар, а все остальное место должно быть занято основным текстом статьи. Определим интерфейс страницы сначала без свойства float:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Обтекание в CSS3</title>
      <style>

      .image {
         margin:10px;
         margin-top:0px;
      }
      .sidebar{
         border: 2px solid #ccc;
         background-color: #eee;
         width: 150px;
         padding: 10px;
         margin-left:10px;
         font-size: 20px;
      }
      </style>
   </head>
<body>
<div>
<div class="sidebar">Л. Толстой. Война и мир. Том второй. Часть третья</div>
<img src="image.png" class="image" alt="Война и мир" />
<p>Старый дуб, весь преображенный, раскинувшись шатром сочной, темной зелени, млел, чуть колыхаясь в лучах вечернего солнца...</p>
<p>«Нет, жизнь не кончена в 31 год, – вдруг окончательно, бесповоротно решил князь Андрей...</p>
</div>
</body>
</html>
```

В данном случае мы получим последовательное размещение элементов на странице.

Теперь на той же странице применим свойство float, изменив стили следующим образом:

```
.image {
   float:left; /* обтекание слева */
   margin:10px;
   margin-top:0px;
}
.sidebar{
   border: 2px solid #ccc;
   background-color: #eee;
   width: 150px;
   padding: 10px;
   margin-left:10px;
   font-size: 20px;
   float: right; /* обтекание справа */
}
```

Соответственно изменится и размещение элементов на странице.

Элементы, к которым применяется свойство float, еще называют floating elements или плавающими элементами.

Иногда возникает необходимость **запретить обтекания**. Подобная задача актуальна, если какой-то блок должен переноситься вниз на новую строку, а не обтекать плавающий элемент. Например, футер, как правило, должен находиться строго внизу и растягиваться по всей ширине страницы. Если же перед футером находится плавающий элемент, то футер может обтекать этот элемент, что не желательно.

Для запрета обтекания элементов в CSS применяется свойство clear, которое указывает браузеру, что к стилизуемому элементу не должно применяться обтекание.

Свойство clear может принимать следующие значения:

- left: стилизуемый элемент может обтекать плавающий элемент справа. Слева же обтекание не работает;
- right: стилизуемый элемент может обтекать плавающий элемент только слева. А справа обтекание не работает;
- both: стилизуемый элемент может обтекать плавающие элементы и относительно них смещается вниз;
- none: стилизуемый элемент ведет себя стандартным образом, то есть принимает участие в обтекании справа и слева

Например, пусть на web-странице будет определен футер:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Обтекание в CSS3</title>
      <style>
      .image {
         float:left;
         margin:10px;
         margin-top:0px;
      }
      .footer{
         border-top: 1px solid #ccc;
```

```
        }
        </style>
    </head>
    <body>
        <img src="image.png" class="image" alt="Дубы" />
        <div class="footer">Copyright © MyCorp. 2016</div>
    </body>
</html>
```

Наличие обтекания будет создавать некорректное отображение, при котором футер смещается вверх.

Изменим стиль футера:

```
.footer{
    border-top: 1px solid #ccc;
    clear: both;
}
```

Теперь футер не будет обтекать изображение, а будет уходить вниз.

18. Прокрутка элементов

Нередко можно столкнуться с ситуацией, когда содержимое блока занимает гораздо больше места, чем сам определено шириной и высотой блока. В этой ситуации по умолчанию браузер все равно отображает содержимое, даже если оно выходит за границы блока.

Однако свойство overflow позволяет настроить поведение блока в подобной ситуации и добавить возможность прокрутки. Это свойство может принимать следующие значения:

- auto: если контент выходит за границы блока, то создается прокрутка. В остальных случаях полосы прокрутки не отображаются;
- hidden: отображается только видимая часть контента. Контент, который выходит за границы блока, не отображается, а полосы прокрутки не создаются;
- scroll: в блоке отображаются полосы прокрутки, даже если контент весь помещается в границах блока, и таких полос прокрутки не требуется;
- visible: значение по умолчанию, контент отображается, даже если он выходит за границы блока

Рассмотрим применение двух значений:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Прокрутка в CSS3</title>
        <style>
        .article1{
            width: 300px;
            height: 150px;
            margin:15px;
            border: 1px solid #ccc;
            overflow: auto;
        }
        .article2{
            width: 300px;
            height: 150px;
            margin:15px;
```

```
            border: 1px solid #ccc;
            overflow: hidden;
        }
        </style>
    </head>
<body>
<div class="article1">
<p>Старый дуб, весь преображенный, раскинувшись шатром сочной, темной зелени, млел, чуть колыхаясь в лучах вечернего солнца. Ни корявых пальцев, ни болячек, ни старого недоверия и горя – ничего не было видно. Да, это тот самый дуб», – подумал князь
Андрей, и на него вдруг нашло беспричинное весеннее чувство радости и обновления.</p>
</div>
<div class="article2">
<p>Старый дуб, весь преображенный, раскинувшись шатром сочной, темной зелени, млел, чуть колыхаясь в лучах вечернего солнца. Ни корявых пальцев, ни болячек, ни старого недоверия и горя – ничего не было видно. Да, это тот самый дуб», – подумал князь
Андрей, и на него вдруг нашло беспричинное весеннее чувство радости и обновления.</p>
</div>
</body>
</html>
```

Свойство overflow управляет полосами прокрутки как по вертикали, так и по горизонтали. С помощью дополнительных свойств overflow-x и overflow-y можно определить прокрутку соответственно по горизонтали и по вертикали. Данные свойства принимают те же значения, что и overflow:

```
overflow-x: auto;
overflow-y: hidden;
```

19. Линейный градиент

Градиенты представляют плавный переход от одного цвета к другому. В CSS3 имеется ряд встроенных градиентов, которые можно использовать для создания фона элемента.

Градиенты в CSS не представляют какого-то специального свойства. Они лишь создают значение, которое присваивается свойству background-image.

Линейный градиент распространяется по прямой от одного конца элемента к другому, осуществляя плавный переход от одного цвета к другому.

Для создания градиента нужно указать его начало и несколько цветов, например:

```
background-image: linear-gradient(left,black,white);
```

В данном случае началом градиента будет левый край элемента, который обозначается значением left. Цвета градиента: черный (black) и белый (white). То есть начиная с левого края элемента до правого будет плавно идти переход из черного цвета в белый.

В использовании градиентов есть один недостаток - многообразие браузеров вынуждает использовать *префикс вендора*:

```
-webkit-
/* Для Google Chrome, Safari, Microsoft Edge,
   Opera выше 15 версии  */
-moz- /* Для Mozilla Firefox  */
```

-o- /* Для Opera старше 15 версии (Opera 12) */

Так, полноценное использование градиента будет выглядеть следующим образом:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Стилизация таблиц в CSS3</title>
<style>
div {
 width: 200px;
 height: 100px;
 background-color: #eee;
 background-image: linear-gradient(left,black,white);
 background-image: -o-linear-gradient(left,black,white);
 background-image: -moz-linear-gradient(left,black,white);
 background-image: -webkit-linear-gradient(left,black,white);
}
p{
 margin: 0;
 padding-top: 30px;
 text-align: center;
 color:white;
}
</style>
</head>
<body>
<div><p>Линейный черно-белый градиент</p></div>
</body>
</html>
```

Для установки начала градиента можно использовать следующие значения: left (слева направо), right (справа налево), top (сверху вниз) или bottom (снизу вверх). Например, вертикальный градиент будет выглядеть следующим образом:

```
background-image: linear-gradient(bottom,black,white);
```

Также можно задать диагональное направление с помощью двух значений:

```
background-image: linear-gradient(top left,black,white);
```

Кроме конкретных значений типа top или left также можно указать угол от 0 до 360, который определит направление градиента:

```
background-image: linear-gradient(30deg,black,white);
```

После величины угла в градусах указывается слово deg.

К примеру, 0deg означает, что градиент начинается в левой части и перемещается в правую часть, а при указании 45deg он начинается в нижнем левом углу и перемещается под углом 45° в верхний правый угол.

После определения начала градиента, можно указать применяемых цветов или опорные точки. Цветов не обязательно должно быть два, их может быть и больше:

```
background-image: linear-gradient(top, red, #ccc, blue);
```

Все применяемые цвета распределяются равномерно. Однако можно также указать конкретные места фона для цветовых точек. Для этого после цвета добавляется второе значение, которое и определяет положение точки.

```
background-image: linear-gradient(left, #ccc, red 20%, red 80%, #ccc);
```

С помощью repeating-linear-gradient можно создавать **повторяющиеся** линейные градиенты. Например:

```
background-image: repeating-linear-gradient(left, #ccc 20px,
 red 30px, rgba(0, 0, 126, .5) 40px);
background-image: -moz-repeating-linear-gradient(left,
 #ccc 20px, red 30px, rgba(0, 0, 126, .5) 40px);
background-image: -webkit-repeating-linear-gradient(left,
 #ccc 20px, red 30px, rgba(0, 0, 126, .5) 40px);
```

В данном случае градиент начинается с левого края элемента с полоски серого цвета (#ccc) шириной 20 пикселей, далее до 30 пикселей идет переход к красному цвету, а затем до 40 пикселей выполняется обратный переход к светло-синему цвету (rgba(0, 0, 126, .5)). После этого браузер повторяет градиент, пока не заполнит всю поверхность элемента.

20. Радиальный градиент

Радиальные градиенты в отличие от линейных распространяются от центра наружу по круговой схеме. Для создания радиального градиента достаточно указать цвет, который будет в центре градиента, и цвет, который должен быть снаружи. Эти цвета передаются в функцию radial-gradient(). Например:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Стилизация таблиц в CSS3</title>
<style>
div {
 width: 200px;
 height: 200px;
 border-radius: 100px;
 background-color: #eee;
 background-image: radial-gradient(white, black);
 background-image: -moz-radial-gradient(white, black);
 background-image: -webkit-radial-gradient(white, black);
}
p{
 margin: 0;
 padding-top: 60px;
 text-align: center;
 color: #eee;
}
</style>
   </head>
   <body>
      <div><p>Радиальный градиент</p></div>
   </body>
</html>
```

Как и в случае с линейным градиентом здесь также надо использовать префиксы вендоров для поддержки браузерами.

Радиальный градиент может иметь две формы: круговую и эллиптическую. Эллиптическая форма представляет распространение градиента в виде эллипса и задается с помощью ключевого слова ellipse:

```
background-image: radial-gradient(ellipse, white, black);
```

Поскольку это значение для градиента по умолчанию, то оно может опускаться при использовании.

Круговая форма представляет распространение градиента в виде кругов от центра во вне. Для этого используется ключевое слово circle:

background-image: radial-gradient(circle, white, black);

Как правило, центр радиального градиента расположен в центре элемента, но это поведение можно переопределить, указав значение для параметра background-position:

background-image: radial-gradient(25% 30%, circle, white, black);

Числа 25% 30% означают, что центр градиента будет находиться на расстоянии в 25% от левой границы и в 30% от верхней границы элемента.

С помощью специальных дополнительных значений можно задать размер градиента:

- closest-side: градиент распространяется из центра только до ближайшей к центру стороне элемента. То есть градиент остается внутри элемента;
- closest-corner: ширина градиента вычисляется по расстоянию из его центра до ближайшего угла элемента, поэтому градиент может выйти за пределы элемента;
- farthest-side: градиент распространяется из центра до самой дальней стороны элемента;
- farthest-corner: ширина градиента вычисляется по расстоянию из его центра до самого дальнего угла элемента

background-image: radial-gradient(25% 30%, circle
farthest-corner, white, black);

Верстка и макетирование

1. Блочная вёрстка

Как правило, web-страница состоит из множества различных элементов, которые могут иметь сложную структуру. Поэтому при создании страницы возникает необходимость нужным образом позиционировать эти элементы, стилизовать их так, чтобы они располагались на странице нужным образом. То есть возникает вопрос создания макета страницы, ее вёрстки.

Существуют различные способы, стратегии и виды вёрстки. Изначально распространенной была **вёрстка на основе таблиц**, так как таблицы позволяют при необходимости легко и просто разделить всё пространство страницы на строки и столбцы. Строками и столбцами довольно удобно управлять, в них легко позиционировать любое содержимое. Именно это определило популярность табличной вёрстки.

Однако табличная вёрстка создает не самые гибкие по дизайну страницы, что является особенно актуальным аспектом в современном мире, где нет одного единственного разрешения экрана, зато есть большие экраны на телевизорах, малые экраны на планшетах, очень маленькие экраны на смартфонах и т.д. Все это многообразие экранов табличная вёрстка оказалась не в состоянии удовлетворить. Поэтому постепенно ей на смену пришла **блочная вёрстка.**

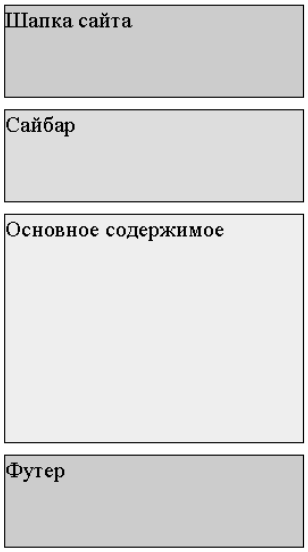
Блочная вёрстка - это условное название способов и приемов вёрстки, когда в большинстве web-страниц для разметки используется CSS-свойство float, а основным строительным элементов web-страниц является элемент , то есть, раздел (блок) документа. Используя свойство float и элементы div или другие элементы, можно создать структуру страницы из нескольких столбцов, как при табличной вёрстке, которая будет значительно гибче.

Теперь используем его для создания **двухколоночной web-страницы**. Допустим, вверху и внизу у нас будут стандартно шапка и футер, а в центре - две колонки: колонка с меню или сайдбар и колонка с основным содержимым.

В начале определим все блоки. При работе с элементами, которые используют обтекание и свойство float, важен их порядок. Так, код плавающего элемента, у которого устанавливается свойство float, должен идти **перед элементом, который обтекает** плавающий элемент. То есть блок сайдбара будет идти до блока основного содержимого:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Блочная вёрстка в HTML5</title>
      <style>
         div{
            margin: 10px;
            border: 1px solid black;
            font-size: 20px;
            height: 80px;
         }
         #header{
            background-color: #ccc;
         }
         #sidebar{
            background-color: #ddd;
         }
         #main{
            background-color: #eee;
            height: 200px;
         }
         #footer{
            background-color: #ccc;
         }
      </style>
   </head>
   <body>
      <div id="header">Шапка сайта</div>
      <div id="sidebar">Сайбар</div>
      <div id="main">Основное содержимое</div>
      <div id="footer">Футер</div>
   </body>
</html>
```

Пока что получается следующая страница:



Высота, граница и отступы блоков в данном случае добавлены произвольно, чтобы идентифицировать пространство блока и отделять его от других.

Далее, чтобы переместить блок сайдбара влево по отношению к блоку основного содержимого и получить эффект обтекания, нам надо указать у блока сайдбара свойство float: left и предпочтительную ширину. Ширина может быть фиксированной, например, 150px или 8em. Либо также можно использовать проценты, например, 30% от ширины контейнера body.

С одной стороны, блоками с фиксированной шириной легче управлять, с другой же процентные значения ширины позволяют создавать более гибкие, "резиновые" блоки, которые изменяют размеры при изменении размеров окна браузера.

Последним шагом является установка отступа блока с основным содержимым от блока сайдбара. Поскольку при обтекании обтекающий блок может обтекать плавающий элемент и справа и снизу, если плавающий элемент имеет меньшую высоту, то нам надо установить отступ, как минимум равный ширине плавающего элемента. Например, если ширина сайдбара равна 150px, то для блока основного содержимого можно задать отступ в 170px, что позволит создать пустое пространство между двумя блоками.

При этом не стоит у блока основного содержимого указывать явным образом ширину, так как браузеры расширяют его автоматически, чтобы он занимал все доступное место.

Принимая во внимание сказанное, изменим стили блоков сайдбара и основного содержимого следующим образом:

```
#sidebar{
   background-color: #ddd;
   float: left;
   width: 150px;
}
#main{
   background-color: #eee;
   height: 200px;
   margin-left: 170px;
    /* 150px (ширина сайдбара) + 10px + 10px (2 отступа) */
}
```

У нас получится сайдбар по левую сторону от основного блока.

Высота блоков в данном случае указана условно для большей наглядности, в реальности, как правило, высоту будет автоматически устанавливать браузер.

Создание правого сайдбара будет аналогично, только теперь нам надо установить у сайдбара значение float: right, а у блока основного содержимого - отступ справа:

```
#sidebar{
   background-color: #ddd;
   float: right;
   width: 150px;
}
#main{
   background-color: #eee;
   height: 200px;
   margin-right: 170px;
}
```

При этом разметка HTML остается такой же, блок сайдбара по прежнему должен предшествовать блоку основного содержимого.

Подобным же образом мы можем добавить на страницу и **большее количество колонок** и сделать более сложную структуру. Например, добавим на web-страницу второй сайдбар:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Блочная вёрстка в HTML5</title>
      <style>
         div{
            margin: 10px;
            border: 1px solid black;
            font-size: 20px;
            height: 80px;
         }
         #header{
            background-color: #ccc;
         }
         #leftSidebar{
            background-color: #ddd;
         }
         #rightSidebar{
            background-color: #bbb;
         }
         #main{
            background-color: #eee;
            height: 200px;
         }
         #footer{
            background-color: #ccc;
         }
      </style>
   </head>
   <body>
      <div id="header">Шапка сайта</div>
      <div id="leftSidebar">Левый сайдбар</div>
      <div id="rightSidebar">Правый сайдбар</div>
      <div id="main">Основное содержимое</div>
      <div id="footer">Футер</div>
```

```
    </body>
</html>
```

Здесь также код обоих сайдбаров должен идти до блока с основным содержимым, который обтекает их.

Теперь изменим стили обоих сайдбаров и основного блока:

```
#leftSidebar{
    background-color: #ddd;
    float: left;
    width: 150px;
}
#rightSidebar{
    background-color: #bbb;
    float: right;
    width: 150px;
}
#main{
    background-color: #eee;
    height: 200px;
    margin-left: 170px;
    margin-right: 170px;
}
```

Вновь у обоих плавающих блоков - сайдбаров нам надо установить ширину и свойство float - у одного значение left, а у другого right.

2. Вложенные плавающие блоки

Нередко встречается ситуация, когда к вложенным в обтекающий блок элементам также применяется обтекание. Например, блок основного содержимого может включать блок собственно содержимого и блок меню. В принципе к таким блокам будут применяться все те же правила, что были рассмотрены ранее.

Определим сначала последовательно все блоки web-страницы:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Блочная вёрстка в HTML5</title>
        <style>
            div{
                margin: 10px;
                border: 1px solid black;
                font-size: 20px;
                height: 80px;
            }
            #header{
                background-color: #ccc;
            }
            #sidebar{
                background-color: #bbb;
                float: right;
                width: 150px;
            }
            #main{
                background-color: #fafafa;
```

```
            height: 200px;
            margin-right: 170px;
        }
        #menu{
            background-color: #ddd;
        }
        #content{
            background-color: #eee;
        }
        #footer{
            background-color: #ccc;
        }
    </style>
</head>
<body>
    <div id="header">Шапка сайта</div>
    <div id="sidebar">Правый сайдбар</div>
    <div id="main">
        <div id="menu">Меню</div>
        <div id="content">Основное содержимое</div>
    </div>
    <div id="footer">Футер</div>
</body>
</html>
```

В главном блоке по-прежнему вложенные блоки идут последовательно: сначала блок меню, а потом блок основного текста.

Теперь применим обтекание к блоку меню:

```
#menu{
    background-color: #ddd;
    float: left;
    width: 160px;
}
#content{
    background-color: #eee;
    margin-left: 180px;
}
```

У плавающего элемента, которым является блок меню, устанавливаются свойства float и width. А у обтекающего его блока content устанавливается отступ слева.

Аналогично можно сделать блок меню справа:

```
#menu{
    background-color: #ddd;
    float: right;
    width: 160px;
}
#content{
    background-color: #eee;
    margin-right: 180px;
}
```

3. Выравнивание столбцов по высоте

При блочной вёрстке мы можем столкнуться с **проблемой высоты столбцов**, которая особенно наглядна, если плавающие блоки имеют определенный фон. Рассмотрим проблему на примере:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Блочная вёрстка в HTML5</title>
    <style>
      body, h2, p{
        margin:0;
        padding:0;
      }
      body{
        font-size: 18px;
      }
      #header{
        background-color: #eee;
        border-bottom: 1px solid #ccc;
        height: 80px;
      }
      #menu{
        background-color: #ddd;
        float: left;
        width: 150px;
      }
      #main{
        background-color: #f7f7f7;
        border-left: 1px solid #ccc;
        margin-left: 150px;
        padding: 10px;
      }
      #footer{
        border-top: 1px solid #ccc;
        background-color: #dedede;
      }
    </style>
  </head>
  <body>
    <div id="header"><h2>Сайт MySyte.com</h2></div>
    <div id="menu">
      <ul>
        <li><a href="#">Главная</a></li>
        <li><a href="#">Блог</a></li>
        <li><a href="#">Контакты</a></li>
        <li><a href="#">О сайте</a></li>
      </ul>
    </div>
    <div id="main">
      <h2>What is Lorem Ipsum?</h2>
      <p>Lorem Ipsum is simply dummy text of the
```

```
            printing and typesetting industry.
            Lorem Ipsum has been the industry...</p>
        </div>
        <div id="footer">
          <p>Copyright © MySyte.com, 2016</p>
        </div>
      </body>
    </html>
```

Здесь уже установлено обтекание, и оно работает, но в зависимости от количества содержимого одна колонка может быть длиннее другой:

В данном случае, если в главном блоке много текста, то блок с меню имеет недостаточную высоту.

Наиболее распространенным решением данной проблемы является **оборачивание плавающего элемента и блока, его обтекающего, в отдельный элемент**, у которого устанавливается фон. Затем этот фон используется наименьшим по длине блоком. В итоге получается иллюзия, что блоки имеют равную длину, а фон у блоков установлен корректно.

Так, изменим выше созданную страницу следующим образом:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Блочная вёрстка в HTML5</title>
    <style>
      body, h2, p{
        margin:0;
        padding:0;
      }
      body{
        font-size: 18px;
      }
```

```
        #header{
            background-color: #eee;
            border-bottom: 1px solid #ccc;
            height: 80px;
        }
        #wrapper{
            background-color: #ddd;
        }
        #menu{
            float: left;
            width: 150px;
        }
        #main{
            background-color: #f7f7f7;
            border-left: 1px solid #ccc;
            margin-left: 150px;
            padding: 10px;
        }
        #footer{
            border-top: 1px solid #ccc;
            background-color: #dedede;
        }
    </style>
</head>
<body>
    <div id="header"><h2>Сайт MySyte.com</h2></div>
    <div id="wrapper">
        <div id="menu">
            <ul>
                <li><a href="#">Главная</a></li>
                <li><a href="#">Блог</a></li>
                <li><a href="#">Контакты</a></li>
                <li><a href="#">О сайте</a></li>
            </ul>
        </div>
        <div id="main">
            <h2>What is Lorem Ipsum?</h2>
            <p>Lorem Ipsum is simply dummy text of the
            printing and typesetting industry.
            Lorem Ipsum has been the industry...</p>
        </div>
    </div>
    <div id="footer">
        <p>Copyright © MySyte.com, 2016</p>
    </div>
</body>
</html>
```

В данном случае плавающий блок menu и обтекающий его блок main обертываются в один элемент wrapper, в котором устанавливается фон для элемента menu. А элемент main, как наибольший элемент, может использовать собственную установку фона.

4. Свойство display

Кроме свойства float, которое позволяет изменять позицию элемента, в CSS есть еще одно важное свойство - display. Оно позволяет управлять блоком элемента и также **влиять на его позиционирование относительно соседних элементов.**

Это свойство может принимать следующие значения:

- inline: элемент становится строчным, подобно словам в строке текста;
- block: элемент становится блочным, как параграф;
- inline-block: элемент располагается как строка текста;
- list-item: элемент позиционируется как элемент списка обычно с добавление маркера виде точки или порядкового номера;
- run-in: тип блока элемента зависит от окружающих элементов;
- flex: позволяет осуществлять гибкое позиционирование элементов;
- table, inline-table: позволяет расположить элементы в виде таблицы;
- none: элемент не виден и удален из разметки HTML

Итак, значение block позволяет определить **блочный** элемент. Такой элемент визуально отделяется от соседних элементов переносом строки, как, например, элемент параграфа p или элемент div, которые по умолчанию являются блочными и при визуализации web-страницы визуально переносятся на новую строку.

Однако элемент span, в отличие от элемента div, по умолчанию блочным не является. Поэтому посмотрим, какие с ним произойдут изменения при применении значения block:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <link href="styles.css" rel="stylesheet">
    <title>Свойство display в CSS3</title>
    <style>
      span{
        color: red;
      }
      .blockSpan{
        display: block;
      }
    </style>
  </head>
  <body>
    <div>Это <span>строчный</span> элемент span</div>
    <div>Это <span class="blockSpan">блочный</span>
    элемент span</div>
  </body>
</html>
```

Здесь определено два элемента span, но один из них является блочным, так как к нему применяется стиль display: block;. Поэтому этот элемент span переносится на новую строку.

В отличие от блочных элементов строчные встраиваются в строку, так как имеют для свойства display значение inline. Элемент span как раз по умолчанию имеет стиль display: inline, поэтому и встраивается в строку, а не переносится на следующую, как параграфы или div. Произведём обратную процедуру - сделаем блочный элемент div строчным:

```
<!DOCTYPE html>
```

```
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Свойство display в CSS3</title>
    <style>
      div{
        display: inline;
      }
    </style>
  </head>
  <body>
    <div>Первый строчный элемент div.</div>
    <div>Второй строчный элемент div.</div>
  </body>
</html>
```

Следует учитывать, что при применении значения inline браузер игнорирует некоторые свойства, такие как width, height, margin.

Еще одно значение - inline-block - представляет элемент, который обладает "смесью" признаков блочного и строчного элементов. По отношению к соседним внешним элементам такой элемент расценивается как строчный. То есть он не отделяется от соседних элементов переводом строки. Однако по отношению к вложенным элементам он рассматривается как блочный. К такому элементу **применяются** свойства width, height, margin.

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Свойство display в CSS3</title>
    <style>
      span{
        width: 100px;
        height: 30px;
        background-color: #aaa;
        text-align: center;
      }
      .inineBlockSpan{
        display: inline-block;
      }
    </style>
  </head>
  <body>
<p>Проехав с полверсты в хвосте <span>колонны</span>, он
остановился</p>
<p>Проехав с полверсты в хвосте <span
 class="inineBlockSpan">колонны</span>, он остановился</p>
</body>
</html>
```

Первый элемент span является строчным, у него значение inline, поэтому для него бессмысленно применять свойства width и height. А вот второй элемент span имеет значение inline-block, поэтому к нему применяются и ширина, и высота, и, при необходимости, можно установить отступы.

Значение run-in определяет элемент, который зависит от соседних элементов. И здесь есть три возможных варианта:

- Элемент окружен блочными элементами, тогда фактически он имеет стиль display: block, то есть сам становится блочным;
- Элемент окружен строчными элементами, тогда фактически он имеет стиль display: inline, то есть сам становится строчным;
- Во всех остальных случаях элемент считается блочным

Значение table, по сути, **превращает элемент в таблицу**. Рассмотрим его применение на примере списка:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Свойство display в CSS3</title>
    <style>
      ul{
        display: table;
        margin: 0;
      }
      li{
        list-style-type: none;
        display: table-cell;
        padding: 10px;
      }
    </style>
  </head>
  <body>
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ul>
  </body>
</html>
```

Здесь список превращается в таблицу, а каждый элемент списка - в отдельную ячейку. Для этого у элемента списка устанавливается стиль display: table-cell. Фактически вместо этого списка мы могли бы использовать стандартную таблицу.

Значение none позволяет **скрыть элемент** на web-странице:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Свойство display в CSS3</title>
    <style>
      .invisible{
        display: none;
      }
    </style>
  </head>
```

```
  <body>
    <p>Первый параграф</p>
    <p class="invisible">Второй параграф</p>
    <p>Третий параграф</p>
  </body>
</html>
```

5. Создание панели навигации

Панель навигации играет важную роль на сайте, так как обеспечивает переходы между страницами сайта или на внешние ресурсы. Рассмотрим, как создать простую панель навигации.

Фактически панель навигации - это набор ссылок, часто в виде ненумерованного списка. Панели навигации бывают самыми различными: вертикальными и горизонтальными, одноуровневыми и многоуровневыми, но в любом случае в центре каждой навигации находится элемент <a>. Поэтому при создании панели навигации мы можем столкнуться с рядом трудностей, которые вытекают из ограничений элемента ссылки. А именно, элемент <a> является строчным, а это значит, что мы не можем указать для него ширину, высоту, отступы. По ширине ссылка автоматически занимает то место, которое ей необходимо.

Распространенное решение данной проблемы для создания **вертикального меню** состоит в том, чтобы сделать ссылку блочным элементом.

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Панель навигации в HTML5</title>
    <style>
      ul.nav{
        margin-left: 0px;
        padding-left: 0px;
        list-style: none;
      }
      ul.nav a {
        display: block;
        width: 7em;
        padding:10px;
        background-color: #f4f4f4;
        border-top: 1px dashed #333;
        border-right: 1px dashed #333;
        border-left: 5px solid #333;
        text-decoration: none;
        color: #333;
      }
      ul.nav li:last-child a {
        border-bottom: 1px dashed #333;
      }
    </style>
  </head>
  <body>
    <ul class="nav">
      <li><a href="#">Главная</a></li>
```

```
            <li><a href="#">Контакты</a></li>
            <li><a href="#">О сайте</a></li>
        </ul>
    </body>
</html>
```

После установки свойства display: block мы можем определить у блока ссылки ширину, отступы и т.д.

Для создания **горизонтального меню** есть два метода. Первый заключается в применении свойства float и создании из ссылок плавающих элементов, которые обтекают друг друга с слева. И второй способ состоит в создании строки ссылок с помощью установки свойства display: inline-block.

Алгоритм создания панели навигации с помощью float разделяется на два этапа. На первом этапе у элемента li, в который заключена ссылка, устанавливается свойство float: left;. Это позволяет расположить все элементы списка в ряд при достаточной ширине, когда правый элемент списка обтекает левый элемент списка.

Второй этап заключается в установке у элемента ссылки display: block, что дает нам возможность устанавливать ширину, отступы, вообщем все те признаки, которые характерны для блочных элементов.

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Панель навигации в HTML5</title>
        <style>
            ul.nav{
                margin-left: 0px;
                padding-left: 0px;
                list-style: none;
            }
            .nav li {
                float: left;
            }
            ul.nav a {
                display: block;
                width: 5em;
                padding:10px;
                margin: 0 5px;
                background-color: #f4f4f4;
                border: 1px dashed #333;
                text-decoration: none;
                color: #333;
                text-align: center;
            }
            ul.nav a:hover{
                background-color: #333;
                color: #f4f4f4;
            }
        </style>
    </head>
    <body>
```

```
        <ul class="nav">
            <li><a href="#">Главная</a></li>
            <li><a href="#">Блог</a></li>
            <li><a href="#">Контакты</a></li>
            <li><a href="#">О сайте</a></li>
        </ul>
    </body>
</html>
```

Для создания горизонтальной панели навигации с помощью inline или inline-block нам надо сделать каждый элемент li строчным, то есть установить для него display: inline. После этого для элемента ссылки, которая располагается в элементе li, мы можем установить display: inline-block:

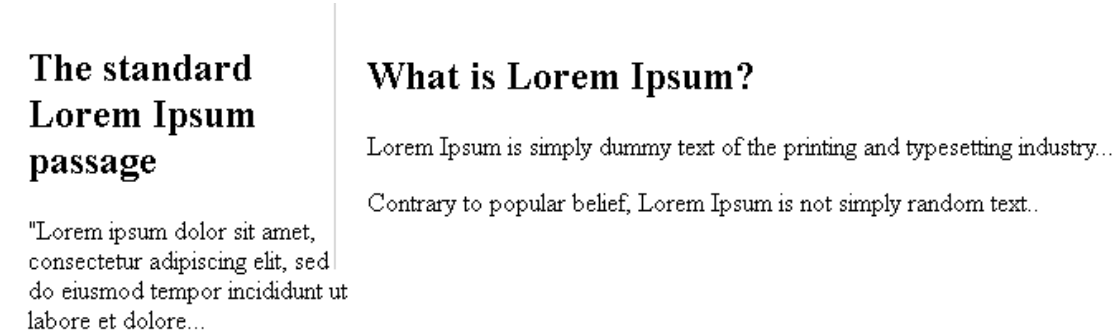
```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Панель навигации в HTML5</title>
        <style>
            ul.nav{
                margin-left: 0px;
                padding-left: 0px;
                list-style: none;
            }
            .nav li {
                display: inline;
            }
            ul.nav a {
                display: inline-block;
                width: 5em;
                padding:10px;
                background-color: #f4f4f4;
                border: 1px dashed #333;
                text-decoration: none;
                color: #333;
                text-align: center;
            }
            ul.nav a:hover{
                background-color: #333;
                color: #f4f4f4;
            }
        </style>
    </head>
    <body>
        <ul class="nav">
            <li><a href="#">Главная</a></li>
            <li><a href="#">Блог</a></li>
            <li><a href="#">Контакты</a></li>
            <li><a href="#">О сайте</a></li>
        </ul>
    </body>
</html>
```

6. Выравнивание плавающих элементов

При работе с плавающими элементами и свойством float довольно часто можно столкнуться с проблемой выпадения из страницы плавающих элементов. У этой проблемы есть различные аспекты и их решения. Рассмотрим эти аспекты.

Например, пусть у нас задан следующий блок:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Блочная вёрстка в HTML5</title>
    <style>
      #sidebar{
        float: left;
        width: 25%;
        padding: 10px;
      }
      #main{
        border-left: 1px solid #ccc;
        width:75%;
        padding: 15px;
        margin-left: 25%;
      }
    </style>
  </head>
  <body>
    <div id="sidebar">
      <h2>The standard Lorem Ipsum passage</h2>
      <p>"Lorem ipsum dolor sit amet, consectetur
        adipiscing elit, sed do
        eiusmod tempor incididunt ut labore et
        dolore...</p>
    </div>
    <div id="main">
      <h2>What is Lorem Ipsum?</h2>
      <p>Lorem Ipsum is simply dummy text of the
        printing and typesetting industry...</p>
      <p>Contrary to popular belief, Lorem Ipsum
        is not simply random text..</p>
    </div>
  </body>
</html>
```



Как видно на скриншоте, **буквы "вылезают" из плавающего блока за границу**, несмотря на то, что в плавающем блоке должен быть установлен еще и внутренний отступ в 10 пикселей от правой границы.

Почему так происходит? Зачастую браузеры своеобразно интерпретируют размеры элемента. В частности, у всех элементов по умолчанию для свойства box-sizing используется значение content-box, то есть при определении ширины и высоты элемента браузер будет прибавлять к значению свойств width и height также и внутренние отступы padding и ширину границы. В итоге это может привести к выпадению плавающих элементов из тех блоков, которые для них предназначены. Поэтому часто для всех элементов рекомендуется устанавливать для свойства box-sizing значение border-box, чтобы все элементы измерялись одинаково, а их ширина представляла только значение свойства width. В таких случаях добавляется следующий стиль:

```
* {
 box-sizing: border-box;
}
```

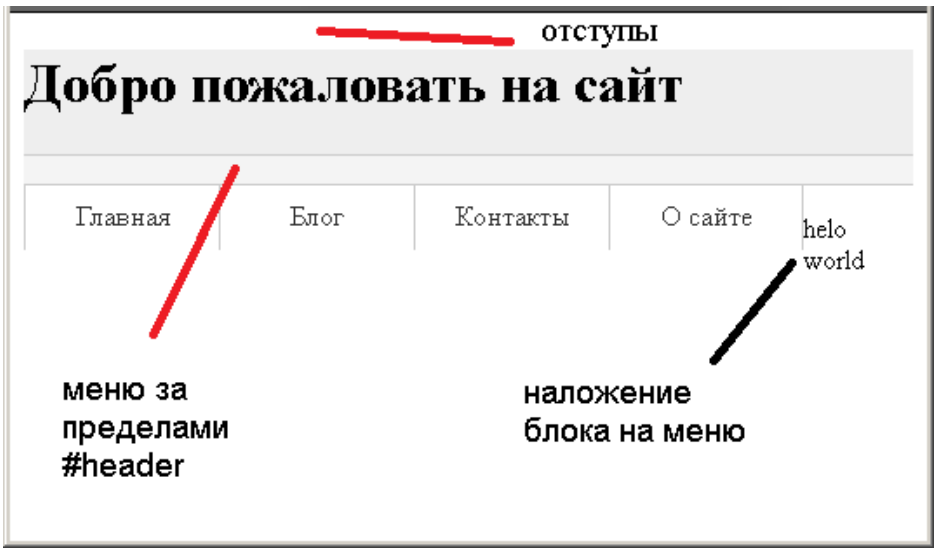
То есть, значение box-sizing: border-box; устанавливается для всех элементов, и все они интерпретируются браузером одинаково. К примеру, добавим этот стиль в выше определенную страницу и мы получим уже несколько результат.

Рассмотрим другую проблему, которая связана с **позиционированием плавающих элементов в контейнере**:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <title>Блочная вёрстка в HTML5</title>
      <style>
         *{
            box-sizing: border-box;
         }
         #header{
            background-color: #eee;
         }
         #nav{
            background-color: #f4f4f4;
            border-top: 1px solid #ccc;
            border-bottom: 1px solid #ccc;
         }
         #nav ul{
            margin-left: 0px;
            padding-left: 0px;
            list-style: none;
         }
         #nav li {
            float: left;
         }
         #nav ul a {
            display: block;
            width: 7em;
            padding:10px;
            border-left: 1px solid #ccc;
            text-decoration: none;
```

```
            color: #333;
            text-align: center;
        }
        #nav ul li:last-child a {

            border-right: 1px solid #ccc;
        }
        #nav ul a:hover{
            background-color: #aaa;
            color: #f4f4f4;
        }
    </style>
</head>
<body>
    <div id="header">
        <h1>Добро пожаловать на сайт</h1>
        <div id="nav">
            <ul>
                <li><a href="#">Главная</a></li>
                <li><a href="#">Блог</a></li>
                <li><a href="#">Контакты</a></li>
                <li><a href="#">О сайте</a></li>
            </ul>
        </div>
    </div>
    <div id="content"><p>helo world</p></div>
</body>
</html>
```



Несмотря на то, что панель навигации определена в блоке с идентификатором header, визуально она явно не находится в элементе head. Также можно увидеть, что появляются лишние отступы, а следующий после заголовка блок накладывается на меню.

Проблема отступов заключается в том, что браузер определяет для элементов встроенные стили по умолчанию в тех случаях, когда стили не указаны явно.Самое просто решение этой проблемы - **сброс** наиболее часто используемых стилей для большинства элементов, например:

```
html, body, div, span, h1, h2, h3, h4, h5, h6, p, a, img, dl, dt, dd, ol, ul, li, form, table, caption, tr, th, td, article, aside, footer, header {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    vertical-align: baseline;
}
```

Вторая проблема, то есть, наложение элемента div с основным контентом на плавающий блок навигационной панели, решается обычно установкой для этого элемента div следующего стиля:

```
#content {
 clear: both;
}
```

Третья проблема, связанная с "выпадением" плавающих элементов меню из границ блока-контейнера, имеет два основных варианта решения. Первое решение состоит в добавлении к элементу, который представляет панель навигации, следующего стиля:

```
ul:after {
    content: " ";
    display: table;
    clear: both;
}
```

Второе решение состоит в том, чтобы сделать сам блок панели навигации плавающим:

```
#nav {
    background-color: #f4f4f4;
    border-top: 1px solid #ccc;
    border-bottom: 1px solid #ccc;
    float:left;
    width: 100%;
    clear: both;
}
```

С учётом сказанного, изменим стили для web-страницы (код HTML остается прежним):

```
* {
    box-sizing: border-box;
}
html, body, div, span, h1, h2, h3, h4, h5, h6, p, a, img, dl, dt, dd, ol, ul, li, form, table, caption, tr, th, td, article, aside, footer, header {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    vertical-align: baseline;
}
#header{
    background-color: #eee;
}
#header h1{
    font-size: 1.3em;
    padding: 15px;
}
```

```
#nav{
   background-color: #f4f4f4;
   border-top: 1px solid #ccc;
   border-bottom: 1px solid #ccc;
}
#nav ul{
   margin-left: 0px;
   padding-left: 0px;
   list-style: none;
}
#nav li {
   float: left;
}
#nav ul a {
   display: block;
   width: 7em;
   padding: 10px;
   border-left: 1px solid #ccc;
   text-decoration: none;
   color: #333;
   text-align: center;
}
#nav ul li:last-child a {
   border-right: 1px solid #ccc;
}
#nav ul a:hover{
   background-color: #aaa;
   color: #f4f4f4;
}
#nav ul:after {
   content: " ";
   display: table;
   clear: both;
}
#content{
   clear: both;
   padding: 15px;
}
```

Теперь web-страница будет выглядеть иначе, как и должна.

7. Создание макета

Используем полученные сведения, создадим макет web-страницы. Для начала определим базовую структуру web-страницы:

```
<!DOCTYPE html>
<html lang="ru">
   <head>
      <meta charset="utf-8">
      <link href="styles.css" rel="stylesheet">
      <title>Блочная вёрстка в HTML5</title>
   </head>
   <body>
      <div id="header">
```

```
        <h1>MySyte.com - Сайт о Lorem Ipsum</h1>
        <div id="nav">
          <ul>
            <li><a href="#">Главная</a></li>
            <li><a href="#">Блог</a></li>
            <li><a href="#">Форум</a></li>
            <li><a href="#">Контакты</a></li>
            <li><a href="#">О сайте</a></li>
          </ul>
        </div>
      </div>
      <div class="wrapper">
        <div id="sidebar1" class="aside">
          <h2>The standard Lorem Ipsum passage</h2>
          <p>"Lorem ipsum dolor sit amet, consectetur
           adipiscing elit, sed do eiusmod tempor
           incididunt ut labore et dolore magna
           aliqua..."</p>
          </div>
        <div id="sidebar2" class="aside">
          <h2>1914 translation by H. Rackham</h2>
          <p>It is a long established fact that a reader
          will be distracted by the readable
          content of a page when looking at its
          layout.</p>
          <h3>Options</h3>
          <ul>
            <li>Item1</li>
            <li>Item2</li>
            <li>Item3</li>
          </ul>
        </div>
        <div id="article">
          <h2>What is Lorem Ipsum?</h2>
          <p>Lorem Ipsum is simply dummy text of the
            printing and typesetting industry...</p>
          <p>Contrary to popular belief, Lorem Ipsum is
            not simply random text. It has roots in a
            piece of classical Latin literature from 45
            BC, making it over 2000 years old. Richard
            McClintock, a Latin professor at
            Hampden-Sydney College in Virginia...</p>
        </div>
      </div>
      <div id="footer">
        <p>Contacts: admin@mysyte.com</p>
        <p>Copyright © MySyte.com, 2016</p>
      </div>
    </body>
</html>
```

В начале идет шапка сайта - блок с header, который содержит заголовок страницы и панель навигации. Далее идет блок wrapper, в котором два сайдбара и блок основного содержимого страницы. Сайдбары условно тоже содержат некоторое содержимое, но главное, что они определены до основного блока. И в самом низу небольшой футер.

В начале web-страницы определено подключение файла styles.css, который будет стилизовать web-страницу. Поэтому создадим в одном каталоге с web-страницей файл styles.css и определим в нем следующее содержимое:

```
* {
 box-sizing: border-box;
}
html, body, div, span, h1, h2, h3, h4, h5, h6, p, a, ul, li {
   margin: 0;
   padding: 0;
   border: 0;
   font-size: 100%;
   vertical-align: baseline;
}
body {
   font-family: Verdana, Arial, sans-serif;
   background-color: #f7f7f7;
}
#header{
   background-color: #f4f4f4;
}
#header h1 {
   font-size: 24px;
   text-transform: uppercase;
   font-weight: bold;
   padding: 30px 30px 30px 10px;
   clear: both;
}
#nav {
   background-color: #eee;
   border-top: 1px solid #ccc;
   border-bottom: 1px solid #ccc;
}
#nav li {
   float: left;
   list-style: none;
}
#nav a {
   display: block;
   color: black;
   padding: 10px 25px;
   text-decoration: none;
   border-right: 1px solid #ccc;
}
#nav li:last-child a {
   border-right: none;
}
#nav a:hover {
   font-weight: bold;
```

```
}
#nav:after {
    content: " ";
    display: table;
    clear: both;
}
.wrapper{
    background-color: #f7f7f7;
}
.aside h2 {
    font-size: 0.95em;
    margin-top: 15px;
}
.aside h3 {
    font-size: 0.85em;
    margin-top: 10px;
}
.aside p, .aside li {
    font-size: .75em;
    margin-top: 10px;
}
.aside li{
    list-style-type: none;
}
#sidebar1 {
    float: left;
    width: 20%;
    padding: 0 10px 0 20px;
}
#sidebar2 {
    float: right;
    width: 20%;
    padding: 0 20px 0 10px;
}
#article{
    background-color: #fafafa;
    border-left: 1px solid #ccc;
    border-right: 1px solid #ccc;
    margin-left: 20%;
    margin-right: 20%;
    padding: 15px;
    width: 60%;
}
#article:after{
    clear:both;
    display:table;
    content:'';
}
#article h2{
    font-size: 1.3em;
    margin-bottom:15px;
}
```

```
#article p{
   line-height: 150%;
   margin-bottom: 15px;
}
#footer{
   border-top: 1px solid #ccc;
   font-size: .8em;
   text-align: center;
   padding: 10px 10px 30px 10px;
}
#nav ul, #header h1, .wrapper, #footer p {
   max-width: 1200px;
   margin: 0 auto;
}
.wrapper, #nav, #header, #footer{
   min-width: 768px;
}
```

Первые три стиля сбрасывают стилевые настройки по умолчанию для используемых нами элементов, а также устанавливают стиль элемента body.

Пара стилей #header управляет отображением шапки (хедера) и заголовка страницы.

Набор стилей #nav управляет созданием горизонтальной панели навигации.

Ранее уже обсуждалось создание горизонтальной панели навигации. В принципе здесь ничего нового не добавляется: для элементов <li> устанавливается обтекание (float: left;), благодаря чему они размещаются в ряд, а каждая ссылка делается блочным элементом (display: block;)

Далее идет настройка средней части страницы, в частности, сайдбаров (#sidebar1, #sidebar2).

Стиль класса wrapper позволяет установить фоновый цвет для боковых панелей. Для каждого сайдбара определяется ширина в 20% от ширины страницы. Процентные значения позволят автоматически подстраивать ширину блоков под ширину окна браузера при его расширении или сужении.

Далее следуют стили блока основного содержимого (#article) и футера (#footer).

Поскольку боковые панели имеют ширину в 20% каждая, то для главного блока устанавливается ширина в 60% и отступы справа и слева в 20%.

И в конце идет пара довольно важных стилей

```
#nav ul, #header h1, .wrapper, #footer p {
   max-width: 1200px;
   margin: 0 auto;
}
.wrapper, #nav, #header, #footer{
   min-width: 768px;
}
```

В начале для ряда селекторов определяется максимальная ширина в 1200 пикселей. Это значит, что основные элементы страницы не выйдут за пределы 1200 пикселей. А автоматический внешний отступ слева и справа позволит центрировать содержимое элементов. То есть при ширине браузера в 1400 пикселей эти элементы с шириной в 1200 пикселей будут размещаться посередине, а справа и слева будут отступы шириной в (1400-1200)/2 = 100 пикселей.

Второй стиль позволят сделать фиксированную минимальную ширину для ряда элементов. То есть в итоге при сжатии окна браузера сайдбары и основной блок будут

выглядеть более менее, а при сжатии окна менее 768 пикселей образуется полоса прокрутки.

В итоге у нас получится макет сайта из 3 колонок, проверьте его в работе.

Данная модель размеров не идеальна. Более гибкие макеты можно получить на основе *адаптивной вёрстки*.

Наиболее популярным является деление макетов по ширине и количеству колонок. Выделяют следующие типы макетов, связанных с шириной:

- фиксированные;
- резиновые;
- эластичные;
- адаптивные;
- комбинированные.

Фиксированный макет располагается по центру окна браузера, а его ширина ограничивается заданными размерами в пикселах.

При создании **резинового макета** ширина колонок в процентах задается таким образом, что макет занимает всю свободную ширину окна браузера.

Эластичный макет по своему виду может не отличаться от фиксированного или резинового макета. Размер элементов задаѐтся в em, привязанных к размеру шрифта. Значение em можно использовать не для всех элементов, оставляя ширину некоторых фиксированной.

Адаптивный макет подстраивается под разрешение монитора и окна браузера, меняя при необходимости ширину макета, число колонок, размеры изображений и текста. Для этого заготавливается несколько стилевых правил или файлов под разный диапазон разрешений, выбор правил происходит через скрипты или CSS3, которые и определяют нужную для этого информацию о пользователе.

Комбинированный макет предполагает использование разной ширины для отдельных частей страницы, например, шапку и подвал делают резиновыми, а контент фиксированным.

Пример макета с тремя колонками, где первая колонка задана в %, третья в пикселах, а вторая – то, что осталось.

```
<!DOCTYPE html>
<html lang="ru">
<head>
<style>
.header { background: #D5BAE4; }
.layout { position: relative;}
.layout DIV { position: absolute; }
.col1 { background: #C7E3E4; width: 30%; }
.col2 { background: #E0D2C7; left: 30%; right: 200px; }
.col3 { background: #ECD5DE; right: 0; width: 200px; }
</style>
<title>Три колонки</title>
</head>
<body>
<div class="header">Шапка сайта</div>
<div class="layout">
<div class="col1">Колонка 1</div>
<div class="col2">Колонка 2. Пример макета с тремя колонками, где первая колонка задана в %, третья в пикселах, а вторая - то, что осталось.</div>
<div class="col3">Колонка 3</div>
</div>
```

```
</body>
</html>
```

8. Позиционирование

CSS предоставляет возможности по позиционированию элемента, то есть мы можем поместить элемент в определенное место на странице

Основным свойством, которые управляют позиционированием в CSS, является свойство position. Это свойство может принимать одно из следующих значений:

- static: стандартное позиционирование элемента, значение по умолчанию;
- absolute: элемент позиционируется относительно границ элемента-контейнера, если у того свойство position не равно static;
- relative: элемент позиционируется относительно его позиции по умолчанию. Как правило, основная цель относительного позиционирования заключается не в том, чтобы переместить элемент, а в том, чтобы установить новую точку привязки для абсолютного позиционированния вложенных в него элементов;
- fixed: элемент позиционируется относительно окна бразуера, это позволяет создать фиксированные элементы, которые не меняют положения при прокрутке

Не следует одновременно применять к элементу свойство float и любой тип позиционирования, кроме static (то есть тип по умолчанию).

Рассмотрим **абсолютное позиционирование**. Область просмотра браузера имеет верхний, нижний, правый и левый края. Для каждого из этих четырех краев есть соответствующее свойство CSS: left (отступ от края слева), right (отступ от края справа), top (отступ от края контейнера сверху) и bottom (отступ снизу). Значения этих свойств указываются в пикселях, em или процентах. Необязательно задавать значения для всех четырех сторон. Часто устанавливают только два значения - отступ от верхнего края top и отступ от левого края left.

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Блочная вёрстка в HTML5</title>
    <style>
      .header {
        position: absolute;
        left: 100px;
        top: 50px;
        width: 430px;
        height: 100px;
        background-color: rgba(128, 0, 0, 0.5);
      }
    </style>
  </head>
  <body>
    <div class="header"></div>
    <p>HELLO WORLD</p>
  </body>
</html>
```

Здесь элемент div с абсолютным позиционированием будет находиться на 100 пикселей слева от границы области просмотра и на 50 снизу.

При этом не столь важно, что после этого элемента div идут какие-то другие элементы. Данный блок div в любом случае будет позиционироваться относительно границ области просмотра браузера.

Если элемент с абсоютным позиционированием располагается в другом контейнере, у которого в свою очередь значение свойства position не равно static, то элемент позиционируется относительно границ контейнера:

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Позиционирование в HTML5</title>
    <style>
      .outer {
        position: absolute;
        left: 80px;
        top: 40px;
        width: 430px;
        height: 100px;
        border: 1px solid #ccc;
      }
      .inner{
        position: absolute;
        left: 80px;
        top: 40px;
        width: 50px;
        height: 50px;
        background-color: rgba(128, 0, 0, 0.5);
      }
    </style>
  </head>
  <body>
    <div class="outer">
      <div class="inner"></div>
    </div>
  </body>
</html>
```

Относительное позиционирование также задается с помощью значения relative. Для указания конкретной позиции, на которую сдвигается элемент, применяются те же свойства top, left, right, bottom:

```
<!DOCTYPE html>
<html lang="ru">
```

```
<head>
    <meta charset="utf-8">
    <title>Позиционирование в HTML5</title>
    <style>
        .outer {
            position: relative;
            left: 80px;
            top: 40px;
            width: 300px;
            height: 100px;
            border: 1px solid #ccc;
        }
        .inner{
            position: absolute;
            left: 80px;
            top: 40px;
            width: 50px;
            height: 50px;
            background-color: rgba(128, 0, 0, 0.5);
        }
    </style>
</head>
<body>
    <div class="outer"><div class="inner"></div></div>
</body>
</html>
```

По умолчанию при совпадении у двух элементов границ, **поверх** другого отображается тот элемент, который определен в разметке HTML **последним**. Однако свойство z-index позволяет **изменить порядок следования элементов при их наложении**. В качестве значения свойство принимает число. Элементы с большим значением этого свойства будут отображаться поверх элементов с меньшим значением z-index. Например:

```
<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <title>Позиционирование в HTML5</title>
        <style>
            body{
                margin:0;
                padding:0;
            }
            .content{
                position: relative;
                top: 15px;
                left: 20px;
                width: 250px;
                height: 180px;
                background-color: #eee;
                border: 1px solid #ccc;
            }
```

```
            .redBlock{
                position: absolute;
                top: 20px;
                left:50px;
                width: 80px;
                height: 80px;
                background-color: red;
            }
            .blueBlock{
                position: absolute;
                top: 80px;
                left: 80px;
                width: 80px;
                height: 80px;
                background-color: blue;
            }
        </style>
    </head>
    <body>
        <div class="content">
            <div class="redBlock"></div>
            <div class="blueBlock"></div>
        </div>
    </body>
</html>
```

Теперь добавим к стилю блока redBlock новое правило:

```
.redBlock{
    z-index: 100;

    position: absolute;
    top: 20px;
    left:50px;
    width: 80px;
    height: 80px;
    background-color: red;
}
```

Здесь z-index равен 100. Но это необязательно должно быть число 100. Так как у второго блока z-index не определен и фактически равен нулю, то для redBlock мы можем установить у свойства z-index любое значение больше нуля.

Теперь первый блок будет накладываться на второй, а не наоборот, как было ранее.

9. Фиксированное позиционирование

Фиксированное позиционирование является распространенным способом удержать в области просмотра браузера некоторые элементы. Достаточно часто на различных сайтах можно увидеть фиксированную панель навигации, которая не изменяет своего положения вне зависимости от прокрутки.

Для фиксированного позиционирования у элементов нужно установить значение fixed для свойства position. После этого с помощью стандартных свойств left, right, top и bottom можно определить конкретную позицию фиксированного элемента.

Создадим фиксированную панель навигации:

```
<!DOCTYPE html>
```

```
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Позиционирование в HTML5</title>
    <style>
      body{
        margin:0;
        padding:0;
      }
      .toolbar{
        position: fixed;
        top: 0;
        left: 0;
        right: 0;
        background-color: #222;
        border-bottom: 1px solid #ccc;
      }
      .toolbar a{
        color: #eee;
        display: inline-block;
        padding: 10px;
        text-decoration: none;
        font-family: Verdana;
      }
      .content{
        margin-top: 50px;
        padding: 10px;
      }
    </style>
  </head>
  <body>
    <div class="toolbar">
      <a href="#">Главная</a>
      <a href="#">Блог</a>
      <a href="#">Контакты</a>
      <a href="#">О сайте</a>
    </div>
    <div class="content">Lorem Ipsum is simply dummy text of the printing and
typesetting industry.
      Lorem Ipsum has been the industry....</div>
  </body>
</html>
```

Чтобы растянуть фиксированный блок от левой до правой границы страницы, устанавливаются три свойства:

top: 0;
left: 0;
right: 0;

Для нижележащего блока с основным содержанием фиксированный элемент фактически не существует в разметке, так как блок с фиксированным, как и с абсолютным позиционированием не участвуют в стандартном потоке HTML. Поэтому по умолчанию оба блока будут накладываться друг на друга и размещаться в одной точке. И нам надо

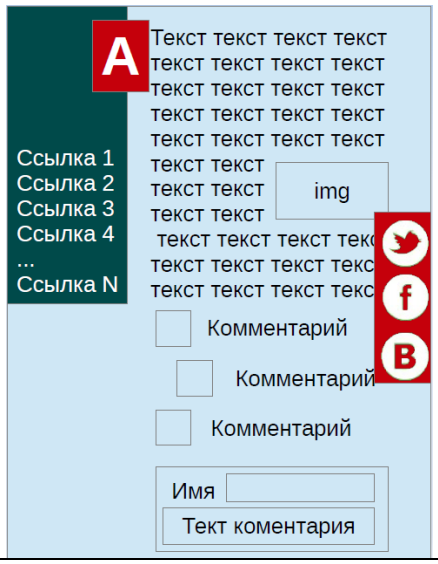
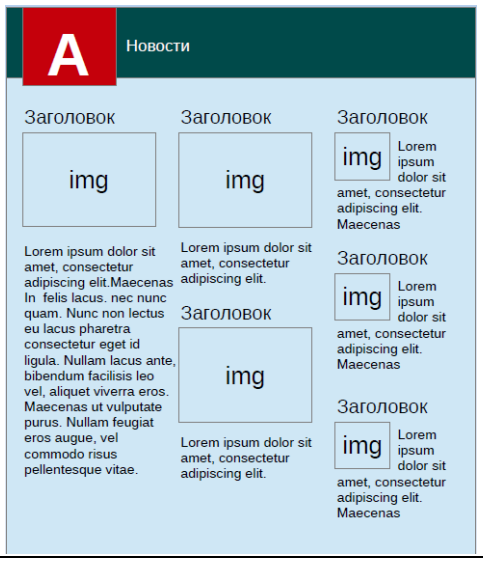
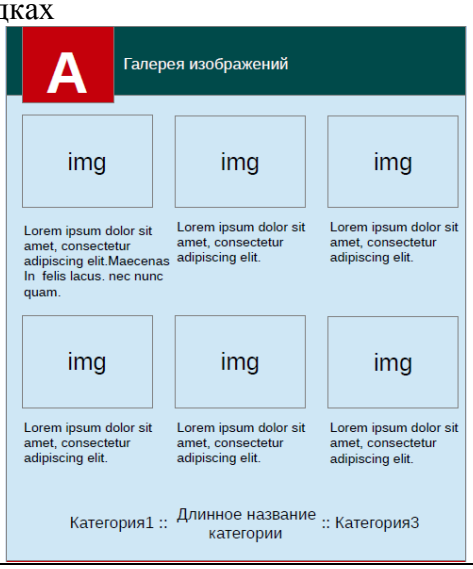
должным образом разместить блок содержимого относительно фиксированного блока, например, установив нужный отступ:

margin-top: 50px;

Фактически отступ идет от границ области просмотра браузера, поэтому высота отступа должна быть больше высоты фиксированного элемента.

Задания для лабораторной работы № 8

1. Выполните вёрстку макета сайта по эскизам следующих страниц.

<table>
<tr>
<td>Страница с меню и формой для комментариев
 
</td>
<td>Страница с трёхколоночным макетом и изображениями
 
</td>
</tr>
<tr>
<td>Страница для галереи изображений с возможностью открытия миниатюр в новых вкладках
 
</td>
<td>Ваш собственный макет</td>
</tr>
</table>

2. Оформите отчет.

Содержание отчета

1. Титульный лист (включая названия дисциплины, номер лабораторной работы)
2. Цель лабораторной работы

3. Формулировка заданий, текст программы по заданиям (с комментариями), скрин экрана результатов выполнения заданий.

Лабораторная работа № 9
Основы JavaScript

Цель: научиться использовать возможности JavaScript при создании веб-страниц.

Теория

1. Основные понятия

Синтаксис языка

Язык JavaScript чувствителен к регистру.

Приложение JavaScript представляет собой набор операторов языка (команд), последовательно обрабатываемых встроенным в браузер интерпретатором. Каждый оператор можно располагать в отдельной строке. В этом случае разделитель ‘;’, отделяющий один оператор от другого, не обязателен. Его используют только в случае задания нескольких операторов на одной строке. Любой оператор можно расположить в нескольких строках без всякого символа продолжения. Например, следующие два вызова функции alert эквивалентны:

```
… alert("Подсказка");
alert(
"Подсказка"
);
…
```

Нельзя перемещать на другую строку единый строковый литерал - он должен располагаться полностью на одной строке текста программы или разбит на два строковых литерала, соединенных операцией конкатенации ‘+’:

```
…
alert("Подсказка");// правильно
alert("Под
сказка");          // не правильно
alert("Под" +
"сказка");      // правильно (но браузер выведет текст одной строкой!)
…
```

Пробельные символы в тексте программы являются незначащими,если только они не используются в строковых литералах.

В JavaScript строковые литералы можно задавать двумя равноправными способами - последовательность символов, заключенная в двойные или одинарные кавычки:

```
"Анна"
'Анна'
```

В строковых литералах можно использовать ESC-последовательности, которые начинаются с символа обратной наклонной черты, за которойследует обычный символ. Некоторые подобные комбинации трактуются как один специальный символ.

Таблица 1.

Esc-последовательности	Символ
\b	Возврат на один символ
\f	Переход на новую страницу
\n	Переход на новую строку

\r	Возврат каретки
\t	Горизонтальная табуляция Ctrl-I
\'	Апостроф
\"	Двойные кавычки
\\	Обратная наклонная черта

ESC-последовательности форматирования используются при отображении информации в диалоговых окнах, отображаемых функциями alert(), prompt() и confirm(), а также, если методом document.write() записывается содержимое элемента pre.

Комментарии в программе JavaScript двух видов - однострочные и многострочные:

// комментарий, расположенный на одной строке.

```
/*
    комментарий, расположенныйна
    нескольких строках.
*/
```

Ссылка на объект осуществляется по имени, заданному параметром name тэга HTML, с использованием точечной нотации. Например, пусть в документе задана форма с двумя полями ввода:

```
<form name="form1">
Фамилия: <input type = "text" name = "student" size = 20>
Курс: <input type = "text" name = "course" size = 2>
</form>
```

Для получения фамилии студента, введенного в первом поле ввода, в программе JavaScript следует использовать ссылку document.form.student.value, а чтобы определить курс, на котором обучается студент, необходимо использовать ссылкуdocument.form.course.value.

Переменные и литералы в JavaScript

В JavaScript все переменные вводятся с помощью одного ключевого слова var. Синтаксическая конструкция для ввода в программе новой переменной с именем name1 выглядит следующим образом:

```
var name1;
```

Объявленная таким образом переменная name1 имеет значение 'undefined' до тех пор, пока ей не будет присвоено какое-либо другое значение, которое можно присвоить и при ее объявлении:

```
var  name1 = 5;
var  name1 = "новая строковая переменная";
```

JavaScript поддерживает четыре простых типа данных:

- Целый
- Вещественный
- Строковый
- Логический (булевый)

Для присваивания переменным значений основных типов применяются литералы – буквальные значения данных соответсвующих типов.

Выражения JavaScript

Выражение – комбинация переменных, литералов и операторов, в результате вычисления которой получается одно единственное значение. Переменные в выражениях должны быть инициализированы.

1. Присваивание

Оператор присваивания (=) рассматривается как выражение присваивания, которое вычисляется равным выражению правой части, и в то же время он присваивает вычисленное значение выражения переменной заданной в левой части:

var name2=10;

2. Арифметическое выражение

Вычисляемым значением арифметического выражения является число. Создаются с помощью арифметических операторов.

Таблица 2.

Оператор	Действие
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления целых чисел
++	Увеличение значения на единицу
--	Уменьшение значения на единицу

3. Логическое выражение

Вычисляемым значением логического выражения может быть true или false. Для создания используются операторы сравнения или логические операторы, применяемые к переменным любого типа.

Таблица 3.

Операторы сравнения	Значение	Логические Операторы	Значение
= =	Равно	&&	логическое И
! =	Не равно	||	логическое ИЛИ
>=	Больше или равно	!	логическое НЕ

<=	Меньше или равно		
>	Строго больше		
<	Строго меньше		

4. Строковые выражения

Вычисляемым значением строкового выражения является число. В JavaScript существует только один строковый оператор – оператор конкатенации (сложения) строк:

string1 = “Моя ” + “строка”

Управляющие конструкции языка JavaScript

Операторы JavaScript

Операторы служат для управления потоком команд в JavaScript. Блокиоператоров должны быть заключены в фигурные скобки.

1. Операторы выбора

 условный оператор if

Эта управляющая структура используется, когда необходимо выполнить некий программный код в зависимости от определенных условий. Также предусмотрена конструкция if-else (если-тогда-иначе).

```
if (условие_1)
{
 оператор_1;              // эти операторы выполняются, если условие_1верно
 оператор_2;
 }
else
{
оператор_3;       // эти операторы выполняются, если условие_1 ложно
оператор_4;
 }
```

Условие для проверки (вопрос компьютеру) записывается сразу после слова if в круглых скобках. После этого в фигурных скобках пишется то, что будет предприниматься в случае выполнения условия. Далее else и снова в фигурных скобках то, что выполнится в случае, если условие не сработает. Количество различных действий между фигурными скобками неограниченно, фактически можно выполнить две различные программы. При сравнении можно использовать логические выражения. Например:

```
<script language="JavaScript">var x
= 5;
var y = 10;if
(x>y) {
    alert('x - максимальное число')
}
else
{
    alert('y - максимальное число')
}
```

```
</script>
```

- оператор выбора switch

Это фактически несколько условных операторов, объединенных в одном. В данном операторе вычисляется одно выражение и сравнивается со значениями, заданными в блоках case. В случае совпадения выполняются операторы соответсвующего блока case.

```
switch (выражение) {
case значение1:
        оператор_1;
        break;
case значение2:
        оператор_2;
        break;
…..
default:
        оператор;
}
```

Если значение выражения не равняется ни одному из значений, заданных в блоках case, то вычисляется группа операторов блока default, если этот блок задан, иначе происходит выход из оператора switch. Необязательный оператор break, задавемый в блоках case, выполняет безусловный выход их оператора switch.

2. Операторы цикла

Оператор цикла повторно выполняет последовательность операторов JavaScript, определенных в его теле, пока не выполниться некоторое заданное условие.

- цикл for (цикл со счетчиком) for

```
(i=1; i<10; i++){
    <тело цикла>
}
```

Первый параметр (i=1) определяет счетчик и указывает его начальное значение. Этот параметр называется начальным выражением, поскольку в нем задается начальное значение счетчика (начальное значение в данном случае равно единице). Это выражение инициализации выполняется самымпервым и всего один раз.

Второй параметр (i<10) - это условие, которое должно быть истинным, чтобы цикл выполнялся, как только условие цикла становится ложным, работа цикла завершается. Он называется условием цикла. Проверка условия цикла осуществляется на каждом шаге; если условие истинно, то выполняется тело цикла (операторы в теле цикла). Цикл в данном случае выполнится только девять раз так как задано условие i<10.

Третий параметр (i++) - это оператор, который выполняется при каждом последовательном прохождении цикла. Он называется выражением инкремента, поскольку в нем задается приращение счетчика (приращение счетчика в данном случае равно единице). Пример автоматической прорисовки нескольких линий с помощью цикла for.

```
<script language="JavaScript" type="text/JavaScript"> for (var
i=1; i<10; i++){
    document.write("<hr align='center' width='100'>");
}
</script>
```

- цикл while (цикл с предусловием)while

```
(условие)
{
    <тело цикла>
```

}

Пока значение условия - true (истинно), выполняется тело цикла. Тело цикла может быть представлено простым или составным оператором.

Оператор while содержит в скобках все необходимые параметры условия цикла (логическое выражение). После определения всех параметров цикла вводится открывающая фигурная скобка, символизирующая начало тела цикла. Закрывающая фигурная скобка вводится в конце тела цикла. Все операторы, введенные в скобках, выполняются при каждом прохождении цикла.

```
<script language="JavaScript">var
 i=1;
 while(i<=10){
 document.write('число='+i+'<br>');i=i+2;
 }
</script>
```

 прерывание и перезапуск цикла

Оператор прерывания break позволяет прервать выполнение цикла иперейти к слудующему за ним выражению:

```
a = 10;
i = 1;
while (a<100){
a = a * i;
if (i>4) break;
++i;
}
```

Если значение i превысит 4, то прерывается выполнение цикла. Оператор перезапуска continue позволяет перезапустить цикл, т.е. оставить невыполненными все последующиевыражения, входящие в телоцикла, и запустить выполнение цикла с самого начала.

```
a = 10;
i = 1;
while (a<100){
++i;
if (i>2 && i<11) continue;a =
a * i;
}
```

Создание и вызов функций в JavaScript

В JavaScript функцией называется именованная часть программногокода, которая выполняется только при обращении к ней посредствомуказания ее имени. Функции создаются с помощью ключевого словаfunction. Обычно функции располагают в секции <head>. Такое расположение функций в HTML-документе гарантирует их полную загрузку до того момента, когда их можно будет вызвать из секции <body>.

После названия функции (func_name) ставятся двойные круглыескобки, программный код при этом заключается в фигурные скобки:

```
<script language="JavaScript">
 function func_name()
 {
      программный код функции (тело функции)
  }
</script>
```

Для того, чтобы вызвать функцию в нужном месте, необходимопросто

указать ее имя в тексте:

```
<script language="JavaScript">
func_name();
</script>
```

Второй вариант вызова функции непосредственно в HTML теге:

```
<a href="javascript:func_name()">Текст ссылки</a>
```

Ниже приведен код страницы HTML, после загрузки которой каждые три секунды будет появляться сообщение, генерируемое вызовом функции myMessage():

```
<script>
function myMessage()
{
alert("My Message")
}
</script>
<body onload='setTimeout ("myMessage()",3000)'>
<p>Каждые три секунды будет появляться сообщение</p>
</body>
```

Метод setTimeout() запускает выполнение кода JavaScript, задаваемого первым строковым параметром, через определенный промежуток времени после выполнения метода.

Интервал задается в миллисекундах (1000 соответствует 1 секунде).

Стандартные объекты и функции ядра JavaScript

Объект Array

Массив - упорядоченный набор однородных данных, к элементам которого можно обращаться по имени и индексу. Язык JavaScript не имеет встроенного типа данных для создания массивов, поэтому для решения используется объект Array и его методы.

Для создания объекта Array вызывается оператор new и конструктор массива - системная функция (ее имя совпадает с именем объекта), инициализирующая элементы массива:

```
m=new Array();
```

Заполнение массива происходит позже. Например:

```
<script language="JavaScript">
 //создание нового массива
m=new Array();
 //заполнение массива
m[0]=1;
m[1]=2;
m[2]=4;
m[3]=56;
</script>
```

В приведенном выше примере с помощью команды new создается массив m, а затем происходит его заполнение - каждому элементу присваивается определенное значение.

```
m=new Array(1,2,4,56);
```

Вызывается команда new и сразу задаются значения всех элементов массива.

```
<script language="JavaScript">
 //создание нового массива и его заполнение
```

m=new Array(1,2,4,56)
</script>

Объявление строковых массивов проводится тем же способом, что иобъявление числовых массивов.

Таблица 4.

Методы объекта Array	Действие
join()	Объединяет все элементы массива в одну строку суказанием разделителя.
reverse()	Изменяет порядок элементов в массиве - первый элемент становится последним, последний - первым
sort()	Выполняет сортировку элементов массива
split()	Разделяет строку на составные части
concat()	Объединяет два массива в один
slice()	Выделяет часть массивы
toString()	Возвращает строку - результат конкатенации всех элементов массива. Элементы массива в строке разделены запятой.
Свойство length	Возвращает длину массива (число элементов в нем).

Пусть определены два массива:

array1 = new Array("Первый","Второй","Третий");array2 =

new Array("Один","Два","Три");

Тогда метод join() первого массива array1.join() возвратит строку:

"Первый,Второй,Третий"

Метод sort() первого массива array1.sort() упорядочит элементы массива array1 (переставив их местами непосредственно в самом массиве array1) в алфавитном порядке:

array1[0] = "Второй";

array1[1] = "Первый";

array1[2] = "Третий";

Поскольку некоторые методы массива возвращают массив, то к нему можно сразу же применить какой-либо метод, продолжив "точечную" нотацию. Например, array1.concat(array2).sort() объединит два массива в один новый и отсортирует его.

Объект Date

Используется для представления дат в программах JavaScript. Время храниться в виде числа миллисекунд, прошедших от 1 января 1970 года.

Данный объект создается также, как и любой объект в JavaScript – спомощью оператора new и конструктора, в данном случае Date():

date1 = new Date(); // значением переменной date1 будет текущая дата Параметром конструктора может быть строка, в которой записана нужная дата:

date1 = new Date("january 14, 2000, 12:00:00");

Можно задать список параметров:

date1 = new Date(2000, 1, 14, 12, 0, 0);

Объект Math

В свойствах данного объекта хранятся основные математические константы, а его методы вычисляют основные математические функции. При обращении к данному объекту, создавать его не надо, но необходимо явно указывать его имя Math. Например:

p = Math.PI; // хранится значение числа пи.

Объект String

Можно явно создавать строковый объект, используя оператор new и конструктор:

myString = new String("Hello!");

Данный объект имеет единственное свойство length, хранящее длину строки, содержащейся в строковом объекте, и два типа методов: одни непосредственно влияют на содержание самой строки, вторые возвращают отформатированный HTML-вариант строки.

Стандартные функции верхнего уровня

В JavaScript существуют несколько функций, для вызова которых не надо создавать никакого объекта, она находятся вне иерархии объектов.

Функция parseFloat(parameter) анализирует значение переданного ей строкового параметра на соответствие представлению вещественногочисла.

Функция parseInt(parameter, base) пытается возвратить целое число по основанию, заданному вторым параметром.

Эти функции полезны при анализе введенных пользователем данных в полях формы до их передачи на сервер.

Функции Number(object) и String(object) преобразуют объект, заданный в качестве его параметра в число или строку.

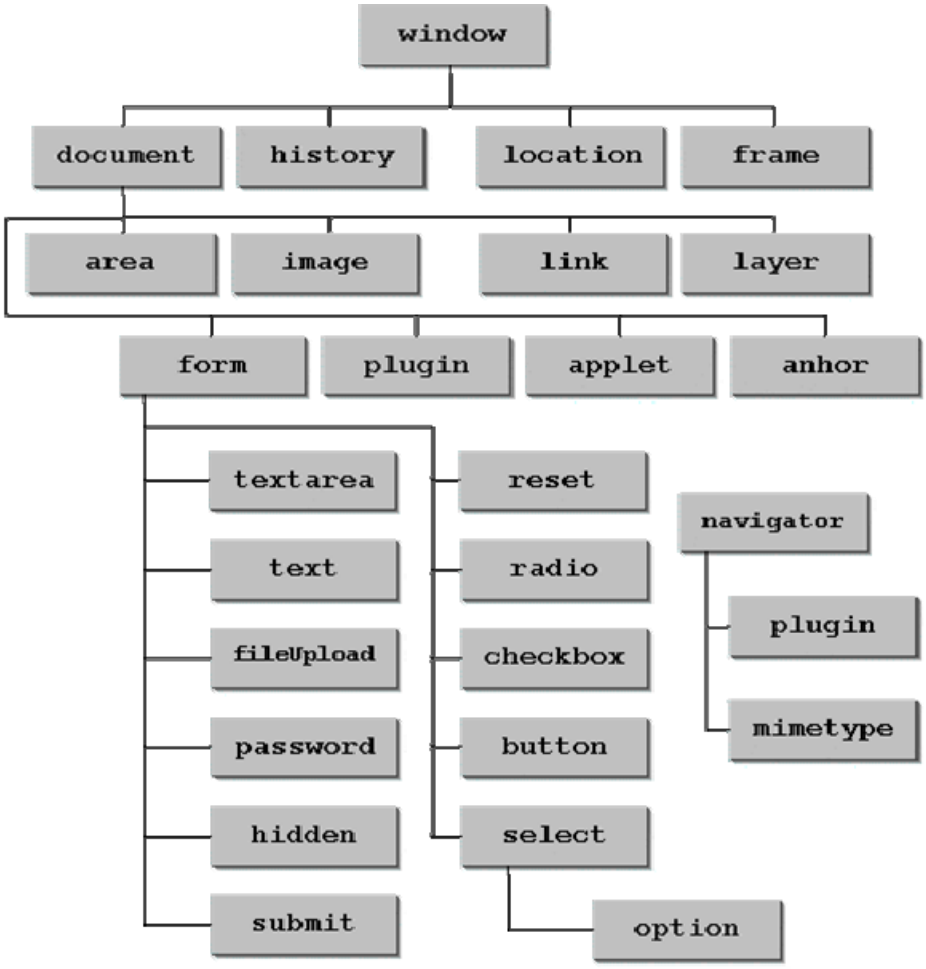
Объекты клиента

При интерпритации страницы HTML браузером создаются объекты JavaScript, свойтсва которых представляют значения параметров тэгов языка HTML.

Иерархия объектов

Созданные объекты существуют в виде иерархической структуры, отражающей структуру самой HTML-страницы. На верхнем уровне расположен объект window, представляющий собой активное окно браузера. Далее вниз по иерархической лестнице следуют объекты frame, document, location и history и т.д.

Значения свойств объектов отражают значения соответствующих параметров тэгов страницы или установленных системных параметров. На рисунке показана структура объектов клиента (браузера).



Особняком стоит объект navigator с двумя дочерними (подчиненными) объектами. Он относится к самому браузеру, и его свойства позволяют определить характеристики программы просмотра. Каждая страница в добавление к объекту navigator обязательно имеет еще четыре объекта:

- window — объект верхнего уровня, свойства которого применяются ко всему окну, в котором отображается документ;
- document — свойства которого определяются содержимым самого документа: связи, цвет фона, формы и т. д.;
- location — свойства которого связаны с url-адресом отображаемого документа;
- history — представляет адреса ранее загружавшихся HTML-страниц.

Кроме указанных объектов страница может иметь дополнительные объекты, зависящие от ее содержимого, которые являются дочерними объектами объекта document. Если на странице расположена форма, то все ее элементы являются дочерними объектами этой формы. Для задания точного имени объекта используется точечная нотация с полным указанием всей цепочки наследования объекта. Наиболее общий объект высшего уровня находится слева в выражении, и слева направо происходит переход к более частным объектам, являющимся при этом наследниками высших в иерархии объектов.

Кроме этих классов объектов пользователь может создавать и свои собственные. Но обычно большинство программ используют эту систему классов и не создают новых.

Объект navigator

Этот объект применяется для получения информации о версиях.

Синтаксис:

navigator.name_properties

Методы и события, догадаться не определены для этого объекта. Да и свойства только для чтения, так как ресурс с информацией о версии недоступен для редактирования.

Свойства

- appCodeName - кодовое имя браузера;
- appName - название браузера;
- appVersion - информация о версии браузера;
- userAgent - кодовое имя и версия браузера;

Ниже приведен пример использования объекта navigator.

```
<html>
<head>
<title> navigator </title>
<script language="javascript">
var firstn = window.prompt("Введите ваше имя: ","ваше имя")function
welcome(){
    var appname=navigator.appNamevar
    appver=navigator.appVersion
    window.alert("Привет, " + firstn + ". Вы используете " + appname + ".
    Версия " + appver + ". Спасибо за визит.");
 }
function writename(){
    document.write(firstn + ".");
}
</script>
</head>
<body onLoad="welcome()">
Добро пожаловать,
<script language="JavaScript">
writename();
</script>
</body>
</html>
```

Объект window

Объект window создается автоматически при запуске браузера, так как для отображения документа необходимо окно. Одно из назначений объекта окна - это создание нового окна. Новое окно браузера создается с помощью метода window.open(). Метод window.open() имеет ряд дополнительных аргументов, которые позволяют задать местоположение окна, его размер и тип, а также указывают, должно ли окно иметь полосы прокрутки, полосу команд и т. п. Помимо этого можно задавать и имя окна.

В общем виде данный метод можно представить следующим образом:

window.open('url', 'name', 'parameters')

Рассмотрим синтаксис более подробно:

- Первый параметр метода window.open() - это url документа, загружаемого в окне. Если его не заполнить, то окно останется пустым.
- Второй параметр определяет название окна (name). Это имя может

использоваться для обращения к созданному окну.

- Третий параметр представляет список необязательных опций, разделенных запятой. С их помощью Вы определяете вид нового окна: наличие в нем панелей инструментов, строки состояния и других элементов.

Приведем таблицу с описанием параметров нового окна, задаваемого третьим параметром (parameters) метода open().

Таблица 5.

Параметр	Значение	Описание
fullscreen	yes: 1 / no: 0	указывает, показывается ли новое окно на полный экран или как обычное окно. По умолчанию показывается обычное окно
channelmode	yes: 1 / no: 0	позволяет указать, отображается ли полоса каналов
toolbar	yes: 1 / no: 0	позволяет указать, отображается ли полоса кнопок
location	yes: 1 / no: 0	позволяет указать, отображается ли полоса для ввода адреса
directories	yes: 1 / no: 0	позволяет указать, отображается ли полоса кнопок для выбора каталогов
status	yes: 1 / no: 0	позволяет указать, отображается ли полоса статуса
menubar	yes: 1 / no: 0	позволяет указать, отображается ли полоса меню
scrollbars	yes: 1 / no: 0	задает отображение горизонтальной и вертикальной полос прокрутки
resizable	yes: 1 / no: 0	позволяет указать, может ли окно изменять свой размер
width	yes: 1 / no: 0	задает ширину окна в пикселах. Минимальное значение - 100
height	yes: 1 / no: 0	задает высоту окна в пикселах. Минимальное значение - 100
top	yes: 1 / no: 0	задает вертикальную координату левого верхнего угла окна
left	yes: 1 / no: 0	задает горизонтальную координату левого верхнего угла окна

Объект window использует три метода отображения сообщений:

- метод prompt() – выводит диалоговое окно с полем ввода, куда пользователь

может ввести информацию

- метод alert() – выводит на экран окно - сообщение с кнопкой ОК и определенным программистом текстом
- метод confirm() – выводит диалоговое окно с кнопками ОК и Cancel. Дает возможность пользователю продолжить или отменить предложенную операцию.

Сообщение, которое вы хотите вывести на экран, набирается вкавычках внутри круглых скобок.

Данный скрипт запрашивает имя посетителя и выдает приветствие свведенным именем.

```
<script language="JavaScript">
 name=window.prompt ("Введите, пожалуйста, свое имя", "Ваше имя");
 window.alert ("Вас зовут, " + name);
</script>
```

В этом фрагменте кода метод prompt имеет следующие параметры: текстзапроса и значение, заполняющее поле ввода по умолчанию; переменнаяname - имя переменной, куда сохраняется введенная информация (имя может быть любым).

Объект document

Объект document имеет дело прежде всего с телом HTML-страницы. Он имеет несколько дочерних объектов (коллекций): all, images, link, anchor и form. Пользуясь объектной моделью построения документа можно, например, обратиться к любой картинке на странице через следующий синтаксис:

document.images.name.src

Для document не существует никаких событий. Некоторые свойства и методы перечислены в таблице, из методов наиболее употребимы write и writeln .

Таблица 6.

Свойства	Назначение
bgColor	Устанавливает цвет фона текущего документа. Этот цвет может иметь шестнадцатеричное представление #rrggbb или соответствующее название. Синтаксис: document.bgColor="#e7e6d8"
fgColor	Устанавливает цвет текста документа. Аналогичен по функциям свойству bgColor
referrer	Указывает url документа, на который ссылается пользовательв настоящее время. Например, если кто-то обратился по адресу: http://www.nm.org/welcome.htm с сервера

	http://www.someplace.com, то свойством referrer будет: http://www.someplace.com, если это свойство было в странице вышеупомянутого расположения; в противном случае оно обращается в null
location	Соответствует адресу url текущего документа

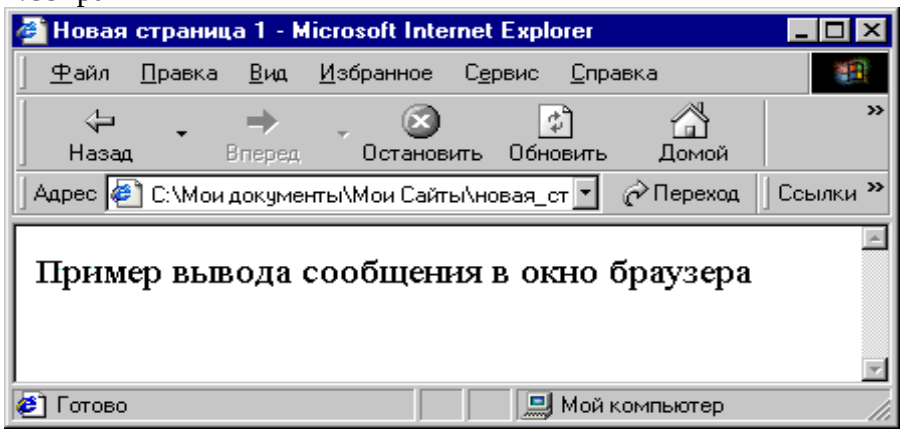
Таблица 7.

Методы	Назначение
write() writeln()	Записывает HTML-текст в текущий документ и должен вызываться, когда документ открывается для записи. Синтаксис: document.write('somestring'), где somestring может быть одной строкой, переменной или же несколькими связанными строками в формате HTML, которые выводятся на экран
lastModified()	Показывает дату последней модификации документа:date1 = document. lastModified
open()	Открывает документ для записи дополнительных строк вформате HTML: document.open()
close()	Закрывает документ, который вызывался методом document.open(): document.close().

Методы write() / writeln().

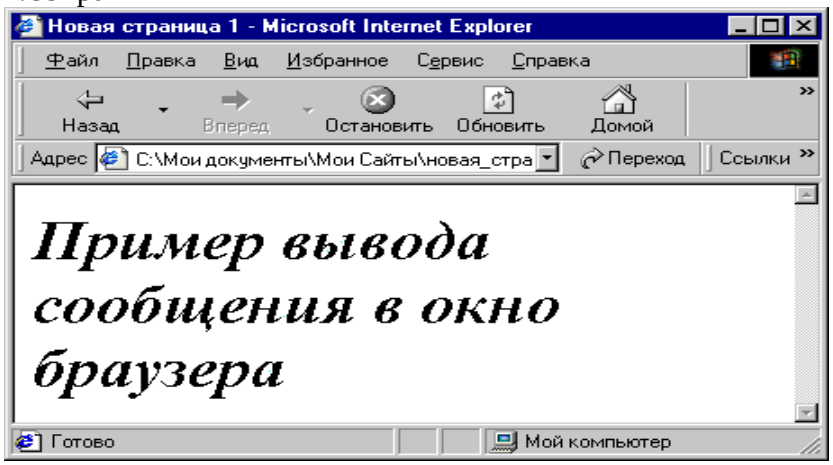
Вызов метода document.write() с указанием определенных параметров приводит к отображению текста в окне браузера. В качестве параметра при вызове метода document.write() мы указываем строку, которую хотели бы увидеть на экране.

```
<script language="JavaScript">
 document.write('Пример вывода сообщения в окно браузера')
</script>
```



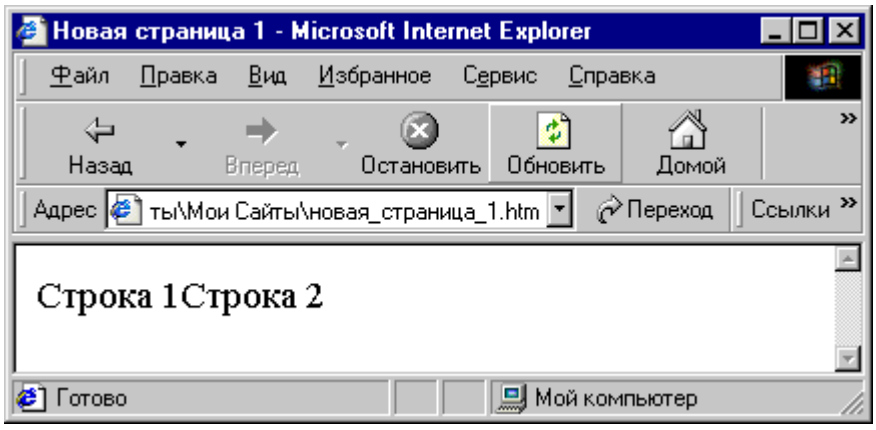
Выводимая строка может содержать и тэги языка HTML. В этом случае браузер выведет данную строку точно так же, как если бы она была размещена непосредственно в HTML документе.

<script language="JavaScript">
document.write('<h1>< i>Пример вывода сообщения в окно браузера</i></h1>')
</script>



При написании скрипта, содержащего несколько команд document.write() подряд, при выводе в браузер текст окажется на одной строке

<script language="JavaScript">
document.write('Строка 1') ;
document.write('Строка 2') ;
</script>



Для размещения каждого куска текста в новом абзаце можно использовать 2 способа:

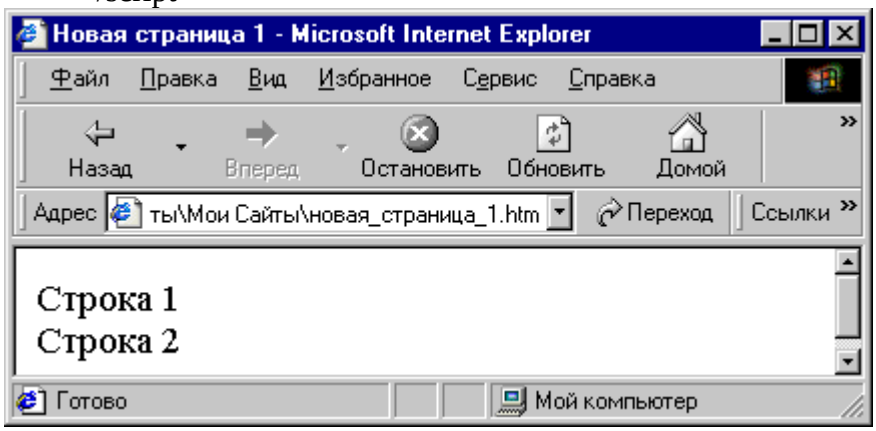
1. либо тег <p>, либо тег <br>, который включается в составвыводимой строки;
2. используя метод document.writeln().

Следующие примеры приведут к одинаковому результату:

<script language="JavaScript">
document.write('Строка 1
') ;
document.write('Строка 2
') ;
</script>

и

```
<script language="JavaScript">
document.writeln('Строка 1') ;
document.writeln('Строка 2') ;
</script>
```



Объект location

Объект location содержит информацию о местонахождении текущего документа, т.е. его интернет-адрес. Его также можно использовать для перехода на другой документ и перезагрузки текущего документа.

Таблица 8.

Свойство	Описание
hash	Имя "якоря" в интернет-адресе документа, если оно есть.
host	Имя компьютера в сети интернет вместе с номеромпорта, если он указан.
hostname	Имя компьютера в сети Интернет.
href	Полный интернет-адрес документа.
pathname	Путь и имя файла, если они есть.
port	Номер порта. Если не указан, возвращает номер 80 - стандартный порт, через который работает протокол http.
protocol	Идентификатор протокола. Если не указан, возвращается "http:".
search	Строка параметров, если она есть.

Таблица 9.

Метод	Описание
assign(url)	Загружает документ, адрес которого передан в качестве параметра. Поддерживаетс только IE начиная с 4.0
reload()	Перезагружает документ с Web-сервера.
replace(url)	Загружает документ, адерс которого передан в качестве параметра, и заменяет в списке истории Web-обозревателя адрес предыдущего документа адресом нового.

Пользуясь объектом location, можно загрузить другой документ на место текущего. Для этого просто необходимо присвоить значение нового интернет-адреса свойству href.

document.location.href = "http://www.---.ru";

Если вы хотите полностью заменить текущий документ, чтобы дажеадрес его не появлялся в списке истории, воспользуйтесь методом replace:

document.location.replace("http://www.--.ru");

Объект form

Каждая форма в документе, определенная тегом <form>, создает объект form, порождаемый объектом document. Ссылка на этот объект осуществляется с помощью переменной, определенной в атрибуте name тега <form>. В документе может быть несколько форм, поэтому для удобства ссылок и обработки в объект document введено свойство-массив forms, в котором содержатся ссылки на все формы документа. Ссылка на первую форму задается как document.forms[0], на вторую - document.forms[1] и т.д. Вместо индекса в массиве forms можно указывать имя формы. Например, если в документе присутствует единственная форма со значением атрибута name=form1, то любой из следующих операторов JavaScript содержит ссылку на эту форму:

document.forms[0];

document.forms["form1"];

document.form1;

Последний оператор возможен в силу того, что объект document порождает объект form (как и все остальные объекты, соответствующие элементам HTML страницы) и ссылку на него можно осуществлять по обычным правилам наследования языка JavaScript.

Все элементы формы порождают соответствующие объекты, подчиненные объекту родительской формы. Таким образом, для ссылки на объект text (с параметром name = text1) формы form1 можно пользоваться любым из нижеприведенных операторов:

document.forms[0].text1;

document.forms["form1"].text1;

document.form1.text1;

Кроме имени элементы формы, имеют свойство value, значение которого определяется смыслом атрибута value элемента формы.Например, для элементов text и textarea значением этого свойства будет строка содержимого полей ввода этих элементов; для кнопки подтверждения - надпись на кнопке и т.д. Обратиться к свойству value можно по тому же принципу, например: document.form1.text1.value

Обработка событий

Использование языка JavaScript при обработке событий значительно расширило возможности языка HTML. Чаще всего программы создаются для обработки информации, вводимой пользователем в поля форм. Возможности управления элементами форм обеспечиваются главным образом за счет функций обработки событий, которые могут быть заданы для всех элементов формы. События делятся на несколько категорий:

- события, связанные с документами (события документа) - загрузка и выгрузка документов;
- события, связанные с гиперсвязью (события гиперсвязи) - помещение указателя мыши на гиперсвязь и активизация гиперсвязи;
- события, связанные с формой (события формы) –
    - щелчки мыши на кнопках;
    - получение и потеря фокуса ввода и изменение содержимогополей ввода, областей текста и списков;
    - выделение текста в полях ввода и областях текста;

События, связанные с документами, возникают при загрузке и выгрузке документа, в то время как события гиперсвязей возникают при ихактивизации или при помещении на них указателя мыши. Чтобы обеспечить перехват события, необходимо написать функцию-обработчик события. В качестве обработчиков событий могут быть заданы целые функции языка JavaScript или только группы из одного или нескольких операторов. В таблице перечислены имена событий и условия их возникновения:

Таблица 10.

Имя события	Атрибут HTML	Условие возникновения события
Blur	onBlur	Потеря фокуса ввода элементом формы
Change	onChange	Изменение содержимого поля ввода или области текста, либо выбор нового элементасписка

Click	onClick	Щелчок мыши на элементе формы или гиперсвязи
Focus	onFocus	Получение фокуса ввода элементом формы
Load	onLoad	Завершение загрузки документа
Unload	onUnload	Выгрузка текущего документа и начало загрузки нового
MouseOver	onMouseOver	Помещение указателя мыши на гиперсвязь
MouseOut	onMouseOut	Помещение указателя мыши не на гиперсвязь
Select	onSelect	Выделение текста в поле ввода или областитекста
Submit	onSubmit	Передача данных формы

Атрибут onClick

Атрибут onClick может использоваться в следующих тегах HTML:

- . . .
- <input type="checkbox" onClick=" function()">
- <input type="radio" onClick="function()">
- <input type="reset" onClick=" function()">
- <input type="submit" onClick=" function()">
- <input type="button" onClick=" function()">

Операторы языка JavaScript, заданные в атрибуте onClick, выполняются при щелчке мыши на таких объектах как гиперсвязь, кнопкаперезагрузки формы или контрольный переключатель. Для контрольных переключателей и селекторных кнопок событие Click возникает не толькопри выборе элемента, но и при разблокировании.

Разберем пример использования атрибута onClick для кнопок, определенных тегами

<input type="button"> в контейнере <form> . . . </form>:

```
…
<script language="JavaScript">
function but1() {
alert("Вы нажали первую кнопку");
}
function but2() {
alert("Вы нажали вторую кнопку");
}
</script>
…
<form>
<input type="button" value="Первая кнопка" onClick="but1()">
```

```
<input type="button" value="Вторая кнопка" onClick="but2()">
</form>
…
```

Работа с меню

Список в форме задается с помощью объекта select, обработка событий выполняется с помощью следующих параметров: onChange - вызывается при изменении выбора; onBlur - вызывается при снятии фокуса с объекта; onFocus - вызывается при перемещении фокуса на объект.

Рассмотрим следующий пример:

```
…
<script language="JavaScript">
function selectBlur()
{
    document.myForm7.myText.value="Вы нажали поле вне списка ";
}
function selectFocus()
{
    document.myForm7.myText.value="Вы нажали ту же кнопку ";
}
function selectChange()
{
    document.myForm7.myText.value="Вы нажали другую кнопку ";
}
</script>
…
<form name="myForm7">
<input type="text" name="myText" size=40 value="Город"><br>
<select name="script" multiple onBlur="selectBlur()"
onFocus="selectFocus()" onChange="selectChange()">
<option value="town1" selected>Париж
<option value="town2">Лондон
<option value="town3">Рим
<option value="town4">Берлин
</select>
</form>
…
```

Управление логикой программного кода при помощи событий

В объектно-ориентированном программировании нет единой структуры управления работой программы. Есть независимые друг от

друга объекты. Когда пользователь щелкает, например, по ссылке на экране, браузер передает событие Click объекту, тега <a>. Для события “щелчок мыши” в этом объекте предусмотрен стандартный обработчик — он загружает в окно новый документ.

Давайте попробуем “перехватить” это событие:

документ page1

Если щелкнуть по ссылке, на экране возникнет надпись “Хода нет?”. Событие перехвачено, но, при закрытии окна alert, видим, что браузер по- прежнему грузит документ page1.htm. При помощи атрибута onClick мы установили в объекте, “отвод” на собственный обработчик. Но когда скрипт нашего обработчика выполнен, управление возвращается к стандартному обработчику, и это вызывает загрузку документа page1.htm.

Отключение стандартной обработки кодируется так:

документ page1

Оператор return указывает возвращаемое функцией значение. Если ее операнд true, то документ загружается, если false, нет.

Подтверждение активизации гиперсвязи.

Аналогичный пример управления логикой программного кода при помощи событий рассмотрен и в следующем примере. Гиперссылка обычно всегда срабатывает по клику мыши, но иногда нужно, чтобы пользователь был уверен, что хочет перейти по ссылке в следующий документ. Для этого существует метод confirm(), который отображает на экране окно сообщения с кнопками "Ok" и "Cancel". Для перехвата события в теге <a href= ... > мы применим событие onClick. Рассмотрите пример подтверждения активизации гиперсвязи:

 Подтверждение активизации гиперсвязи

Определение событий формы

Объект form имеет два обработчика событий: onSubmit и onReset. В эти обработчики событий, задаваемые в пределах дескриптора <form>, добавляется группа операторов JavaScript или функция, управляющая формой.

Если вы добавите оператор (или функцию) в обработчик onSubmit, то он (или она) вызывается до отправки данных в сценарий CGI. Для того чтобы отменить отправку данных на обработку сценарием CGI, обработчик событий onSubmit должен возвратить значение false. Если же он возвращает значение true, то данные отправляются на сервер. В некоторых случаях необходимо добавить в форму кнопку reset,запускающую обработчик событий onReset.

Для формы одним из важных действий на странице является проверка правильности заполнения полей пользователем на машине клиента до

пересылки их на сервер. В следующем примере разъясняется, как выполнять эту процедуру.

Рассмотрим скрипт, который будет проверять правильность заполнения формы. Необходимо проверить нет ли пустых строк иправильно ли введен e-mail:

```
<html>
<head>
<title>пример формы</title>
<script language="JavaScript">
        function doSend(){
        var v=document.user.e.value.indexOf("@",1)
        if(document.user.f.value==""){
        alert('Вы должны заполнить поле ФИО')
        document.user.f.focus()
         }
        if(document.user.a.value==""){
        alert('Вы должны заполнить поле адреса')
        document.user.a.focus()
        }
        if(document.user.e.value==""){
        alert('Вы должны заполнить поле e-mail')
        document.user.e.focus()
        }
       if(v==-1){
       alert('Адрес e-mail указан неверно')
       document.user.e.select()
       document.user.e.focus()
       }
       else
       document.user.submit()
       }
       </script>
</head>
<body>
<p align="center"><font size=6>Данные о пользователе</font>
<form name="user">
<b>Пожалуйста, укажите данные о себе:</b>
<br>
ФИО<input type="text" name="f" size="30"><br>
Адрес<input type="text" name="a" size="35"><br>e-
mai<input type="text" name="e" size="30"><br>
<input type="button" value="Послать" onClick="doSend()">
<input type="reset" value="Отменить">
</form>
```

```
</p>
</body>
</html>
```

Вставка звука

Если вам необходимо озвучить страницу, вот простейшая инструкция:

```
<bgsound src="music/gimn.mid" loop=infinite>
```

С помощью JavaScript можно разнообразить страницы сайта.

Пример скрипта проигрывания музыки при наведении на заголовоктекста:

```
…
<script>
function playHome() { document.all.sound.src =
"music/file.mid"}
</script>
…
<bgsound id=sound>
<h1 onMouseover=playHome()>Заголовок с музыкой</h1>
…
```

DHTML

DHTML (динамический HTML) – это набор средств, которые позволяют создавать более интерактивные Web-страницы без увеличения загрузки сервера. Другими словами, определенные действия посетителя ведут к изменениям внешнего вида и содержания страницы без обращения к серверу.

DHTML построен на объектной модели документа (Document Object Model, DOM), которая расширяет традиционный статический HTML- документ. DOM обеспечивает динамический доступ к содержимомудокумента, его структуре и стилям. В DOM каждый элемент Web- страницы является объектом, который можно изменять. DOM не определяет новых тэгов и атрибутов, а просто обеспечивает возможность программного управления всеми тэгами, атрибутами и каскадными таблицами стилей (CSS).

Каждой гиперссылке, заголовку или текстовому параграфу можно присвоить имя, атрибуты стиля или цвета текста и указать это имя в сценарии, имеющемся на данной странице. Сценарий может быть написан на любом существующем скриптовом языке, но здесь подразумевается использование языка JavaScript. Элементы страницы впоследствии могут изменяться в результате определенного события, например при наведении курсора мыши, при щелчке, нажатии клавиши на клавиатуре либо после истечения заданного промежутка времени. Изменяться может не толькостиль и цвет текста, но и весь объект, текст или рисунок.

Объединение JavaScript и CSS

1. Пример изменения цвета текста.

```
<html>
<head>
<title>Простая страница</title>
</head>
<body>
<h1 style="color:red">Добро пожаловать на нашу страницу!</h1>
<p>Здесь много интересной  информации. Здесь много
      интересной информации. Здесь много интересной
      информации. Здесь много интересной
```

```
информации. 3десь много интересной
информации. </p>
</body>
</html>
```

Данный пример применения CSS позволяет сделать заголовок красного цвета. Допустим, вы хотите, чтобы текст заголовка только тогда становился красным, когда пользователь наводит на него курсор. Этого можно добиться с помощью CSS и JavaScript.

Шаг 1. Удаление существующей информации о стиле

Это действие может показаться вам шагом назад, но оно действительно необходимо:

```
<html>
<head>
    <title>Простая страница</title>
</head>
<body>
<h1>Добро пожаловать на нашу страницу! </h1>
<p>3десь много интересной  информации. 3десь много
    интересной информации. 3десь много интересной
    информации. 3десь много интересной
    информации. 3десь много интересной
    информации. </p>
</body>
</html>
```

Шаг 2. Добавление идентификатора

Поскольку вам нужно как-то обращаться к элементу, с которым будут производиться манипуляции, необходимо в тэг <hl> добавить атрибут id - это краткое обозначение, позволяющее указать нужный элемент:

```
<html>
<head>
<title>Простая страница</title>
</head>
<body>
<hl id="head1">Доброо пожаловать на нашу страницу!</h1>
<p>3десь много интересной  информации. 3десь много
    интересной информации. 3десь много интересной
    информации. 3десь много интересной
    информации. 3десь много интересной
    информации. </p>
    </body>
</html>
```

Шаг 3. Добавление обработчика событий

Следующий шаг — добавление обработчика событий. Этому действию соответствует событие onMouseover. Также следует указать имя функции, которая будет вызываться при выполнении события:

```
<html>
<head>
<title>Простая страница</title>
</head>
<body>
<h1 id="head1" onMouseover ="colorchange ()">Добро пожаловать нанашу
страницу!</h1>
```

3десь много интересной информации. 3десь много
интересной информации. 3десь много интересной
информации. 3десь много интересной
информации. 3десь много интересной
информации.
</body>
</html>

Шаг 4. Написание сценария JavaScript

Вам потребуется единственная строка, состоящая из следующихчастей:

- имя объекта на странице, с которым должен выполняться вашсценарий - в данном случае head1;
- применяемый аспект JavaScript - в данном случае style;
- атрибут стиля, который будет изменяться - color;
- новое значение, принимаемое атрибутом стиля - red.

Соедините это, и получится следующая строка:

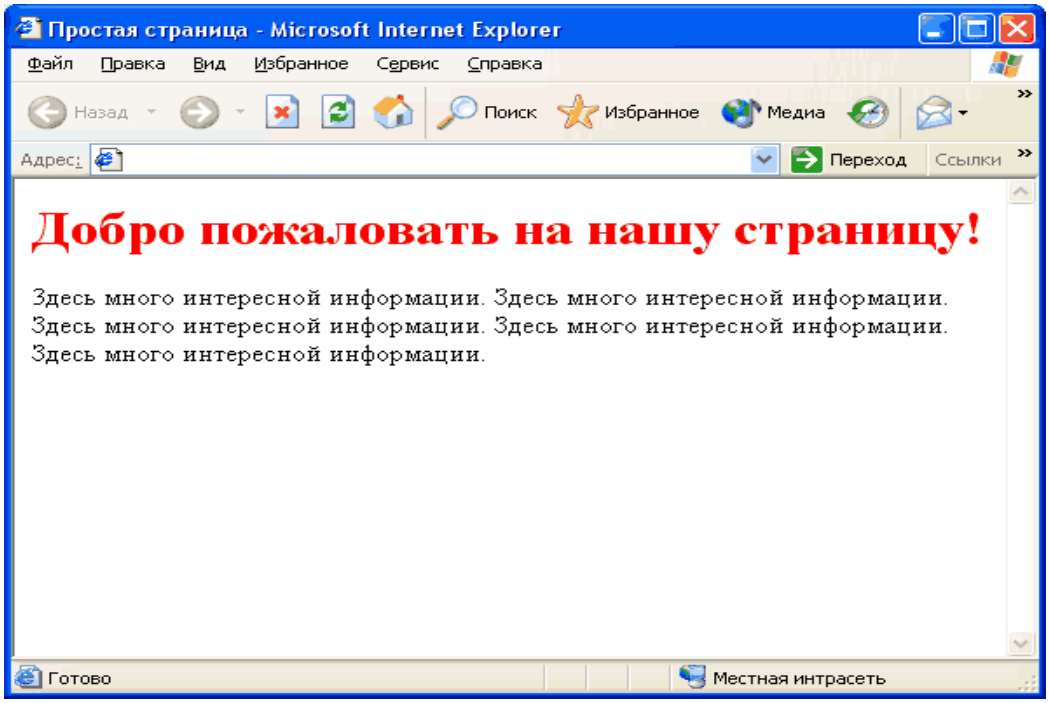
Head1.style.color = "red"

Добавьте ее в функцию и сохраните файл. В окончательном вариантестраница должна выглядеть так:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
function colorchange()
{
head1.style.color = "red";
}
</script>
</head>
<body>
<h1 id="head1" onMouseover="colorchange()">Добро пожаловать нанашу
страницу!</h1>
<p>3десь много интересной информации. 3десь много
интересной информации. 3десь много интересной
информации. 3десь много интересной
информации. 3десь много интересной
информации. </p>
</body>
</html>
```

Откройте эту страницу в браузере и посмотрите, что происходит, когда вы наводите курсор на заголовок. Если все было сделано правильно, то цвет заголовка изменится.

Но обратите внимание, что цвет текста не становится прежним, когда вы убираете курсор с заголовка.



2. Пример использования атрибута text-decoration.

Используя в сценариях JavaScript атрибуты, название которых пишется через дефис, убирайте дефис и пишите оба слова слитно, причем второе слово должно начинаться с заглавной буквы. Таким образом, text- decoration в сценариях должно выглядеть как textDecoration.

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
function addunderline()
{
head.style.textDecoration = "underline";
}
function removeunderline()
{
head.style.textDecoration = "none";
}
</script>
</head>
<body>
<h1 id="head" onMouseover="addunderline()"
onMouseout="removeunderline()">Добро пожаловать на нашустраницу!
</h1>
<p>3десь много интересной информации. Здесь много
        интересной информации. Здесь много
        интересной информации. Здесь много
        интересной информации.</p>
</body>
</html>
```

Необходимо обратить внимание на прописную букву в слове textDecoration — если все слово набрать в нижнем регистре, сценарий выполняться не будет. Теперь

при наведении курсора заголовок станетподчеркнутым, а затем, если убрать курсор, вернется в прежнее состояние.

3. Пример точного позиционирования текста

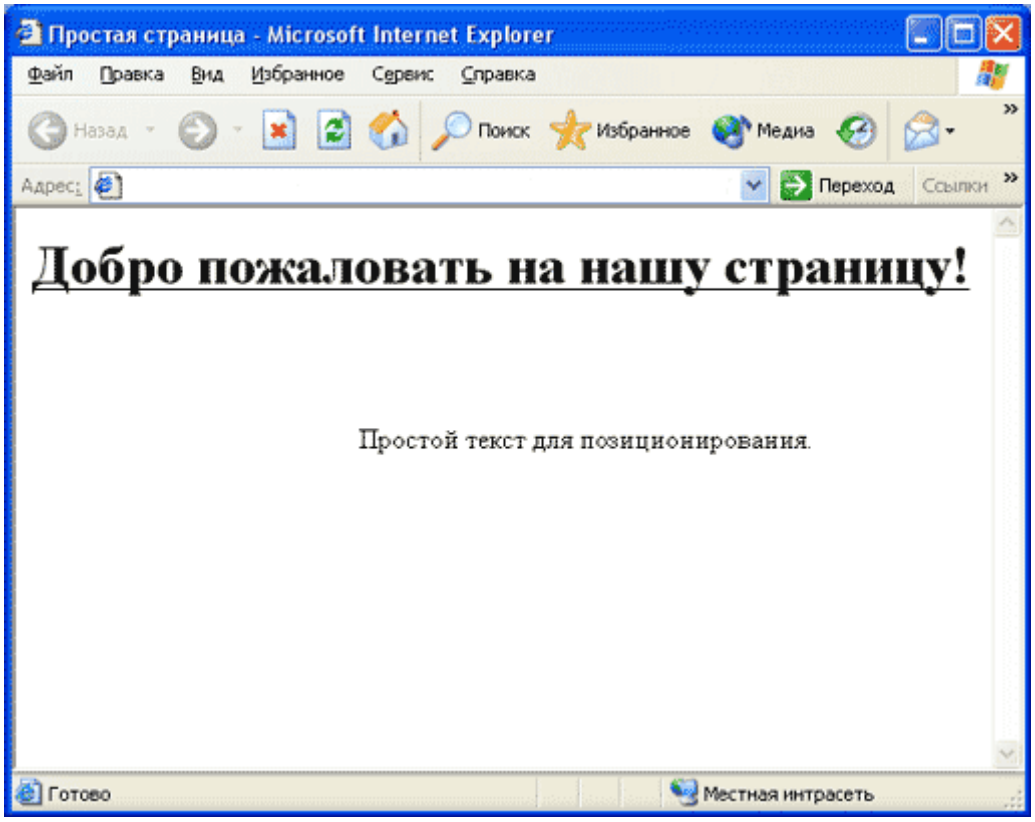
Сначала необходимо рассмотреть, каким образом осуществляется позиционирование текста.

Наиболее часто применяются следующие атрибуты позиционирования:

- position - имеет два интересующих нас значения: absolute и relative (по умолчанию значение static). Для значения absolute в качестве точки отсчета используется верхний левый угол окна браузера, и все параметры местоположения отмеряются от него. В свою очередь, дляrelative точкой отсчета является то место, в котором разместился бы элемент страницы, если бы не было представлено никакой информации о местоположении;
- top - используется для указания вертикального смещения элемента от точки отсчета. Величина смещения может выражаться в различных единицах (пикселы, дюймы, сантиметры, миллиметры и т.п.). В наших примерах используются пикселы. Положительное значение top соответствует смещению элемента страницы вниз, в то время как отрицательное - по направлению к верхней границе окна браузера;
- left - подобен атрибуту top, но применяется для указания горизонтального направления. Положительное значение соответствует сдвигу элемента вправо, отрицательное - влево.

```
<html>
<head>
<title>Простая страница</title>
</head>
<body>
<h1 style=" text-decoration: underline">Добро пожаловать на нашу
страницу!</h1>
<p style="position:absolute; top:125px; left:200px">Простой текст для
позиционирования.</p>
</body>
</html>
```

С помощью CSS текст расположен со значением absolute, то есть его положение отсчитывается от верхнего левого угла окна браузера. Значение атрибута top равно 125px, таким образом, текст будет расположен на 125 пикселов ниже верхнего края страницы. Значение атрибута left равно 200px, то есть текст будет сдвинут на 200 пикселов от левого края окна браузера.



Создание анимационных объектов

Анимация - это процесс «оживления» объекта

Анимация включает в себя две составляющие: расстояние между соседними кадрами, называемое скачком (jump), и временной промежуток между двумя последовательными скачками, называемый интервалом (interval). При больших скачках и длительных интервалах анимация выглядит медленной и грубой. Движение объектов кажется неестественным и воспринимается как мелькание. При малых скачках и кратких интервалах анимация выглядит более плавной, хотя, если чересчур увлечься, движение покажется нарочитым.

1. Пример перемещения текста слева направо

Сначала следует ввести текст в тэге , ограничивающем текст,добавить идентификатор id.

```
<html>
<head>
<title>Простая страница</title>
</head>
<body>
<div id="anim">
Текст, шагом марш!
</div>
```

```
</body>
</html>
```

Затем воспользуемся CSS, чтобы поместить текст в начальное положение:

```
<html>
<head>
<title>Простая страница</title>
</head>
<body>
<div id="anim" style="position:absolute; left:10; top:10">
Текст, шагом марш!
</div>
</body>
</html>
```

Далее начинаем работать над сценарием JavaScript. Поскольку не нужно, чтобы текст вечно двигался вправо, надо предусмотреть возможность контролирования этого процесса. Чтобы запустить сценарий на выполнение только при условии, если текст находится, например, менеечем в 500 пикселах от левой границы экрана, удобнее всего воспользоваться оператором if. Для этого понадобится атрибут CSS pixelLeft.

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
function moveTxt()
{
if (anim.style.pixelLeft < 500)
{
}
}
</script>
</head>
<body>
<div id="anil" style="position:absolute; left:10; top:10">
Текст, шагом марш!
</div>
</body>
</html>
```

Теперь рассмотрим операторы, управляющие анимацией. Прежде всего нужно задать скачок. Каждый раз текст будет перемещаться вправо на 50 пикселей. Атрибут pixelLeft используется не только для определения положения текста, но и для изменения положения на 50 пикселей:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
function moveTxt()
{
if (anim.style.pixelLeft < 500)
{
anim.style.pixelLeft +=50;
}
}
</script>
</head>
<body>
<div id="anim" style="position:absolute; left:10; top:10">
Текст, шагом марш!
</div>
</body>
</html>
```

Далее речь пойдет об интервале. Он задается с помощью метода setTimeout, позволяющего вновь запустить функцию после истечения определенного промежутка времени. Давайте установим интервал до повторного запуска функции moveTxt(), равным 5000 мс:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
<!-- Маскируемся!
function moveTxt()
{
if (anim.style.pixelLeft < 500)
{
anim.style.pixelLeft +=50;
setTimeout("moveTxt()", 5000);
}
}
</script>
</head>
<body>
<div id="anim" style="position:absolute; left:10; top:10">
Текст, шагом марш!
</div>
</body>
</html>
```

Процесс будет повторяться до тех пор, пока условие оператора if не станет ложным. Последнее, что нужно сделать, - запустить сценарий на выполнение. Для этого следует воспользоваться событием onLoad:

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
function moveTxt()
{
if (anil.style.pixelLeft < 500)
{
anil.style.pixelLeft +=2;
setTimeout("moveTxt()", 50);
}
}
</script>
</head>
<body onLoad="moveTxt()">
<div id="anil" style="position:absolute; left:10; top:10">
Текст, шагом марш!
</div>
</body>
</html>
```

2. Пример движения текста по диагонали

```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
function moveTxt()
{
if (anim.style.pixelTop < 500)
{
anim.style.pixelTop + = 2;
anim.style.pixelLeft +=2;
setTimeout("moveTxt()", 50);
}
}
</script>
</head>
<body onLoad="moveTxt()">
<div id="anim" style="position:absolute; left:10; top:10">
Текст, шагом марш!
</div>
```

```
</body>
</html>
```

Слои

Позиционирование слоя

Для создания слоев следует использовать тег или . Эти теги взаимозаменяемы и различаются лишь внешним видом в браузере. Если требуются отступы до и после текста, следует использовать элемент
. При размещении текста внутри параграфа применяется тег .

В Таблице 11 перечислены наиболее важные атрибуты.

Таблица 11.

id	Имя слоя, используемое для указания его в <script>
left	Позиция слоя по x координате
top	Позиция слоя по y координате
position	Задает относительную или абсолютную позицию относительно других объектов
z-index	Позиция слоя при наложении нескольких объектов друг на друга
width	Ширина слоя в пикселах или %
height	Высота слоя в пикселах или %
bgColor	Цвет фона слоя
background	Картинка фона
src	Внешний html документ, содержащийся в слое

Пример наложения текста:

```
<html>
<body>
Слой1 наверху
<div style="position:relative; font-size:50px; z-index:2; color: navy">
Слой 1
</div>
<div style="position:relative; top:-55; left:5; color:orange; font-size:80px;z-index:1">
Слой 2
</div>
Слой 2 наверху
<div style="position:relative; font-size:50px; z-index:3; color: navy">
Слой 1
</div>
<div style="position:relative; top:-55; left:5; color:orange; font-size:80px;z-index:4">
Слой 2
</div>
</body>
```

</html>

Тип позиционирования слоя определяется параметром position, положение элемента - двумя координатами top и left.

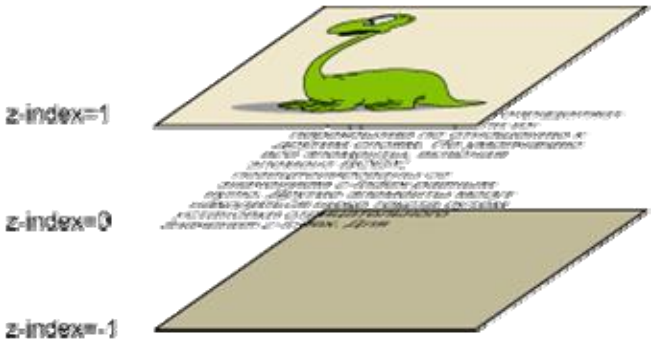
Кроме тегов и абсолютное позиционирование поддерживают следующие элементы: <applet>, <input>, <button>, <object>, <select>, <fieldset>, <iframe>, <table>, <img>, <textarea>.

Параметр position:relative используется для смещения слоя относительно родительского элемента. Установка этого значения не изменяет размещение элемента, но если установлены значения свойств торили left, то слой смещается от своего нормального положения в документе.В то время как свойство position указывает тип системы координат, параметры top и left определяют точную позицию слоя. Значения этихпараметров могут определяться в процентном отношении или пикселах,принимать положительные и отрицательные величины.

Это даетвозможность размещать контент выше или ниже на странице независимоот физической позиции кода HTML. То есть, в верхней части страницы можно поместить слой, который описан внизу HTML-документа.

Свойство z-index

Свойство z-index определяет порядок слоев, или их перекрытие по отношению к другим слоям. По умолчанию все слои позиционированы со значением z-index равным нулю. Другие слои могут размещаться ниже путем установки отрицательного значения z-index. Для слоев, у которых z- index не установлен, это значение назначается неявно в соответствии с их положением в документе. Поэтому слой, который помещен в документ позже, размещается выше остальных элементов, позиционированныхранее.



Свойства visibility и display

Для отображения или скрытия слоя используется свойство visibility. Он может принимать значения visible, установленное по умолчанию, для показа слоя, и hidden, которое его прячет.

Например, скрытый блок текста можно оформить следующим образом:

Спрятанный слой

При этом, когда используется данное свойство для скрытия элемента, соотвествующий данному элементу блок занимает прежнее положение на странице, но сома содержимо не отображается.

Чтобы на странице не оставалось пустого блока, соотвествующего скрываемому элементу, можно использовать свойство display со значением none. Для отображения элемента display равно block.

Динамическое управление слоями

Сценарии JavaScript позволяют динамически управлять параметрами установленных слоев. Это позволяет получить такие эффекты, как скрытие и отображение слоя, изменение порядка отображения, перемещение слоя в окне браузера. Все эти эффекты достигаются с помощью изменения соответствующих стилевых параметров установленных слоев.

Для обращения к слоям из сценариев JavaScript, удобнее всего каждому слою дать собственное имя при помощи параметра id. Например:

```
<div id="div1">
...
</div>
```

Для того, чтобы скрыть отображение слоя div1, можно использовать следующую команду:

```
div1.style.visibility='hidden';
```

Для повторного отображения слоя следует выполнить следующееприсвоение:

```
div1.style.visibility='visible';
```

Пример динамической смены слоев: в данном примере дляотображения некоторого слоя следует нажать на соответствующую ссылку. Эту идею можно применить и для организации выпадающихменю.

```
<html>
<head>
<style type="text/css">div
{
position: absolute;
top: 20;
left: 160; visibility:
hidden;
}
</style>
<script language="JavaScript">
function show_d(d)
{
div1.style.visibility='hidden';
div2.style.visibility='hidden';
div3.style.visibility='hidden';
div4.style.visibility='hidden';
div5.style.visibility='hidden';
d.style.visibility='visible';
}
</script>
</head>
<body>
<a href="javascript:void(0)" onClick="show_d(div1);">
показать слой 1
</a><br>
<a href="javascript:void(0)" onClick="show_d(div2);">
показать слой 2
</a><br>
<a href="javascript:void(0)" onClick="show_d(div3);">
показать слой 3
</a><br>
```

```
<a href="javascript:void(0)" onClick="show_d(div4);">
показать слой 4
</a><br>
<a href="javascript:void(0)" onClick="show_d(div5);">
показать слой 5
</a><br>
<div id="div1">
<h3>Слой номер один</h3>
```
Некоторый текст, на слое расположенный. Его можно скрыть ипоказать. Текст может содержать несколько строк.
```
</div>
<div id="div2">
<h3>Слой номер два</h3>
```
Содержит свой текст. Если показывается, то текст на других слоях невиден.
```
</div>
<div id="div3">
<h3>Слой номер три</h3>
```
Тоже текст. При работе со слоями надо следить, чтобы текст одного слоя не "выглядывал" из-под другого слоя при самых различных размерах окна браузера и используемых шрифтах.
```
</div>
<div id="div4">
<h3>Слой номер четыре</h3>
Здесь нет текста.
</div>
<div id="div5">
<h3>Слой номер пять</h3>И
тут тем более нет.
</div>
</body>
</html>
```

Динамическое изменение цвета фона ячеек

Использование стилей и управление ими с помощью JavaScript позволяет менять вид ячейки "на ходу", при выполнении определенных условий, таких как наведение курсора на ссылку или саму ячейку.

Рассмотрим самый простой прием - цвет фона ячейки меняется, когда курсор мыши наводится на нее. Наведение мыши на область отслеживается событием onMouseover, а вывод мыши за ее пределы - событием onMouseout. Поскольку цвет фона меняется у той же самой ячейки, на которую наводим курсор мыши, то изменение стиля делается с помощью метода this.style.background.

```
…
<table width=60% border=1 cellspacing=0 cellpadding=4
bordercolor=#333333 align=center>
<tr>
<td align=center bgcolor=#cccccc
onMouseover="this.style.background='#ffcc33'"
onMouseout="this.style.background='#cccccc'"><a href="#">Пункт1</a></td>
<td align=center bgcolor=#cccccc><a href="#">Пункт 2</a></td>
```

```
</tr>
</table>
…
```

Задания для лабораторной работы № 9

Задание 1. Размещение скриптов в HTML-документе.

1. Создайте простой HTML-документ.
2. Добавьте два абзаца с произвольным текстом.
3. Организуйте между двумя абзацами вывод приветсвенного сообщения в диалоговом окне, задав необходимые команды внутри тэга <script>.
4. Добавьте команду вывода аналогичного приветсвенного сообщенияв окно браузера после закрытия диалогового окна.

Задание 2. Размещение скриптов в HTML-документе.

1. Создайте простой HTML-документ.
2. Добавьте два абзаца с произвольным текстом.
3. Организуйте между двумя абзацами вывод приветсвенного сообщения в диалоговом окне, задав необходимые команды JavaScript во внешем файле. Для этого:
   - создайте новый текстовый файл,
   - поместите в него код JavaScript,

- сохраните файл с именем main.js следующим образом: укажитетип файла “Все файлы”, кодировку “UTF-8”.

4. Добавьте ссылку на внешний скриптовый файл из рабочего HTML-документа.

Задание 3. Размещение скриптов в HTML-документе.

1. Создайте простой HTML-документ.
2. Сохраните документ с именем Ex3.html в рабочей папке.
3. Добавьте в документ код JavaScript так, чтобы в диалоговом окне появлялось поле с надписью "Введите сюда своё имя" и созначением по умолчанию в поле "Введите имя". Для этого используйте метод promt(...) объекта window. Для хранения введенного значения заведите новую переменную.
4. Организуйте вывод введенного значения имени в окно браузера в виде: "Ваше имя <…..>".
5. Дополните код, чтобы в новом диалоговом окне появилось надпись "Начать заново? " При положительном ответе появлялось диалоговоеокно: "Не надоело? ", при отказе – "Ну и правильно!". Используйте для написания методы alert(…) и confirm(…) объекта window.

Задание 4. Операторы управления, функции.Объекты ядра JavaScript.

1. Рассмотрите пример скрипта:

```
<html>
<head>
<title>if</title>
</head>
<body>
<script language="JavaScript" type="text/JavaScript">var x, y;
x=parseInt(prompt("Введите значение x",")); // метод parseInt()
переводит строку в целое
y=parseInt(prompt("Введите значение y",")); // число
if(x<y)
{
alert("Максимальное число - y")
}
else {
alert("Максимальное число - x")
```

```
}
</script>
</body>
</html>
```

1. Допишите скрипт так, чтобы при введении пользователем одинаковых чисел, открывалось сообщение "Введенные числа равны!".
2. Напишите скрипт, в котором пользователя просят ввести правильный пароль. При вводе правильного пароля, в окне браузера появляется сообщение о том, что пароль верен. При вводе неправильного пароля – выпадает сообщение о неправильно введенном пароле. Для выполнения задания введите переменную password, в которую сохраните верное значение пароля.

Задание 5. Операторы управления, функции.Объекты ядра JavaScript.

1. Рассмотрите пример скрипта:

```
<html>
<head>
<title>for</title>
</head>
<body>
<h1>Пример простой</h1>
<script language="JavaScript" type="text/JavaScript">function line()
{
document.writeln("<hr align='center' width='100'>");
}
for (var i=1; i<10; i++)line();
</script>
</body>
</html>
```

2. Создайте вариант прорисованных линий со следующим условием:
   - десять линий должны располагаться друг под другом,
   - первая должна быть длинной 10 пикселей,
   - каждая последующая на 10 пикселей больше.

Задание 6. Операторы управления, функции.Объекты ядра JavaScript.

1. Создайте простой HTML-документ.
2. Добавьте в документ код JavaScript так, чтобы в окне браузера была выведена таблица степеней двойки вида:

Степень	Результат
2^0	1
2^1	2
2^2	4
2^3	8

2^4	16
2^5	32

Для этого в сценарии используйте метод write(…) объекта document для формирования содержимого страницы. На каждой итерации цикла for сформируйте очередную строку таблицы, в первую ячейку которойзаносится соответствующая степень двойки, а во вторую результат ее возведения в указанную степень. Для выполнения этого действия используется встроенный объект Math и его метод pow(…), возводящий первый параметр в степень, заданную вторым параметром. Обратите внимание, что метод write(…) может вызываться с любым количеством фактических параметров. Результатом его работы в любом случае являетсявывод в документ строки, полученной конкатенацией всех параметров,переданных в метод.

Задание 7. Операторы управления, функции.Объекты ядра JavaScript.

1. Рассмотрите пример скрипта:

```
<html>
<head>
<title>array</title>
</head>
<body>
<script language="JavaScript">
year=new Array("декабрь","январь","февраль","март","апрель","май",
"июнь","июль","август","сентябрь","октябрь","ноябрь");
summer=new Array();  //летние месяцы
summer=year.slice(6,9);
document.write(summer+"<br>");
</script>
</body>
</html>
```

2. Создайте массив, содержащий названия школьных предметов. Выделите из него два массива. Пусть к первому относятся предметы из раздела точных наук, а ко второму - из раздела гуманитарных наук. Для создания и вывода в окно браузера новых массивов используйте метод slice(…) и wtite(…) объекта document. Оформите исполняющий скрипт в виде отдельной функции, описанной в разделе <head> и вызванной в разделе <body>.

Задание 8. Операторы управления, функции.Объекты ядра JavaScript.

1. Создайте простой HTML-документ.
2. Добавьте скрипт, на основе которого будут выполняться следующие условия:

- если на страницу зашел пользователь через браузер Microsoft Internet Explorer, перенаправьте его автоматически на другую страницу.
- если на страницу зашел пользователь через любой другой браузер, перенаправьте его на другую страницу.

Для выполнения задания используйте свойство appName объекта navigator.

Задание 9. Объекты клиентских приложений.Обработка событий.

1. Рассмотрите скрипт:

```
<html>
<head>
<title>document</title>
</head>
<body>
<script language="JavaScript" type="text/JavaScript"> document.write("Спасибо, что пришли к нам на курсы!");
</script>
</body>
</html>
```

2. Допишите скрипт так, чтобы
   - цвет фона документа был #E7E6D8,
   - цвет шрифта – красный,
   - внизу выводилась дата последней модификации документа, используйте для этого слияние методов wtite(…) и lastModified(…) объекта document.

Задание 10. Объекты клиентских приложений.Обработка событий.

1. Рассмотрите пример скрипта открытия нового окна на странице:

```
<html>
<head>
<title>window</title>
</head>
<body>
<h1>Создание нового окна</h1>
<hr>
<script language="JavaScript" type="text/JavaScript">
window.open("http://www.google.com","","toolbar=no,scrollbars=yes,widt
h=250, height=250, resizable=yes, top=100, left=500")
</script>
</body>
</html>
```

2. Измените скрипт так, чтобы выполнялись следующие условия:

- открытие нового окна происходило при нажатии на ссылку с текстом: «Щелкните на ссылке для получения справочной информации»,
- размеры окна - 500x500,
- есть возможность изменения размеров окна.

Для выполнения задания используйте написание функции.

Задание 11. Объекты клиентских приложений.Обработка событий.

1. Создайте страницу с переадресацией на другой адрес (redirect).
2. Изменитескрипт так, чтобы переадресация на другой адрес была сзадержкой 5 секунд.

Задание 12. Объекты клиентских приложений.Обработка событий.

1. Создайте HTML-документ, в котором будет 2 ссылки:
   - первая ссылка должна ссылаться на PDF файл; при нажатии на нее выпадает сообщение с предупреждением о том, что для загрузки документа требуется программа типа Acrobat, и продолжить загрузку или нет; используйте для написания метод confirm(…) для подтверждения загрузки;
   - вторая ссылка должна содержать такой код, чтобы при наведении на нее мыши менялся цвет фона документа на красный.

Задание 13. Объекты клиентских приложений.Обработка событий.

1. Создайте HTML-документ, содержащий любую картинку.
2. Добавьте скрипт с условиями:
   - при наведении курсора мыши на картинку она увеличивается,
   - при отведении курсора мыши – уменьшается до исходногоразмера.

Постройте скрипт через использование функций и событий MouseOver и MouseOut.

Задание 14. Объекты клиентских приложений.Обработка событий.

1. Создайте HTML-страницу содержащую следующую форму заполнения данных:

Ваше имя: *

Пароль *

Подтверждение пароля*

Электронный адрес: *

Тема сообщения:

Сообщение:

Отправить Очистить

* - необходимые для заполнения поля

2. Добавьте скрипт, проверяющий следующие данные:

- заполнено ли поле имени,
- введен ли пароль и содержит ли он больше 4-х символов.Используйте для этого свойство length данного поля,
- совпадают ли значения, введенные в оба поля для паролей,
- заполнено ли поле электронного адреса и содержит ли оносимвол @,
- заполнено ли поле сообщения и содержит ли оно больше 10 символов,

3. При несоблюдении условий, курсор должен установиться в тополе, где пользователем введено неверное значение.
4. Сохраните документ с именем Ex15.html в рабочей папке.

Задание 15. Объединение JavaScript и CSS.

1. Рассмотрите скрипт:

```
<head>
<title>h1</title>
<script language="JavaScript">
function colorchange()
{
head.style.color = "red";
}
</script>
</head>
<body>
<h1 id="head" onmouseover="colorchange()">Добро пожаловать на нашу страницу!</h1>
</body>
</html>
```

2. Допишите скрипт страницы таким образом, чтобы красный цветисчезал после отвода курсора мыши с заголовка.

Задание 16. Объединение JavaScript и CSS.

1. Рассмотрите скрипт:

```
<html>
<head>
<title>text decoration</title>
<script language="JavaScript">
function addunderline()
```

```
{
head.style.textDecoration = "underline";
}
function removeunderline()
{
head.style.textDecoration = "none";
}
</script>
</head>
<body>
<h1 id="head" onMouseover="addunderline()"
onMouseout="removeunderline()">
Добро пожаловать на нашу страницу!
</h1>
</body>
</html>
```

2. Допишите скрипт страницы таким образом, чтобы на одинарный щелчок мыши появлялось полоса над заголовком, а на двойной щелчок – текст зачеркивался. Используйте события onclick, ondblclick и значения рассматриваемого свойства overline и line- through.

Задание 17. Объединение JavaScript и CSS.

1. Создайте HTML-документ, содержащий любое изображение.
2. Поместите изображение в тег . Задайте для него абсолютное позиционирование со смещением вниз и влево на 500 пикселей.

Задание 18. Слои. Движущиеся элементы.

1. Рассмотрите скрипт:

```
<html>
<head>
<title>simple animation</title>
<script language="JavaScript">
function moveTxt()
{
if (anil.style.pixelLeft < 500)
{
```

```
anil.style.pixelLeft +=50;
setTimeout("moveTxt()", 5000);
}
}
</script>
</head>
<body onLoad="moveTxt()">
<div id="anil" style="position:absolute; left:10; top:10">
Текст, шагом марш!
</div>
</body>
</html>
```

2. Измените скрипт страницы:
    - добейтесь плавного передвижения текста;
    - измените направление текста - задайте направление сверхувниз при помощи атрибута pixelTop.

Задание 19. Слои. Движущиеся элементы.

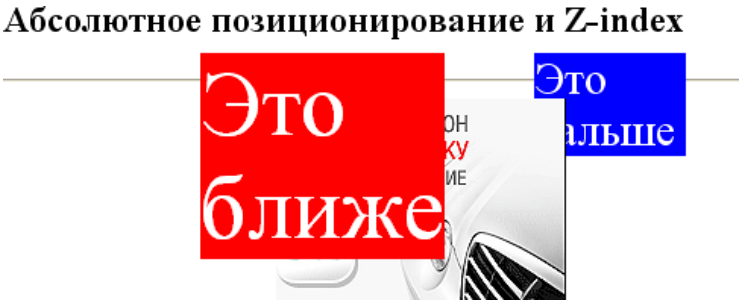
1. Рассмотрите скрипт:

```
<head>
<title>anima1</title>
<script language="JavaScript">
function moveTxt()
{
if (anim.style.pixelTop <500)
{
anim.style.pixelTop +=2;
anim.style.pixelLeft +=2;
setTimeout("moveTxt()", 50);
}
}
</script>
</head>
<body onLoad="moveTxt()">
<div id="anim" style="position:absolute; left:10; top:10">
Текст, шагом марш!
</div>
</body>
</html>
```

2. Измените направление текста. Задайте направление с верхнего правого угла экрана (приблизительно) по диагонали к середине экрана.

Задание 20. Слои. Движущиеся элементы.

1. Создайте HTML-страницу, на которой будет три слоя. Верхний и нижней представляют из себя статичные квадраты разного цвета с текстом, а меду ними должна проплывать любая картинка слева направо.



2. Оформите отчет по всем заданиям работы.

Содержание отчета

1. Титульный лист (включая названия дисциплины, номер лабораторной работы)
2. Цель лабораторной работы
3. Формулировка заданий, текст программы по заданиям (с комментариями), скрин экрана результатов выполнения заданий.