МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНА МЕТАЛУРГІЙНА АКАДЕМІЯ УКРАЇНИ

Н.В. НЕЧУХАЄВА, В. Г. РАСЧУБКІН

ВИВЧЕННЯ МОВИ HTML З ВИКОРИСТАННЯМ КАСКАДУ СТИЛІВ CSS ТА ЕЛЕМЕНТІВ МОВИ JAVASCRIPT

Частина 2

**Затверджено на засіданні Вченої ради академії
як навчальний посібник. Протокол № 1 від 30.01.2012**

Дніпропетровськ НМетАУ 2012

УДК (004.43)
Нечухаєва Н.В., Расчубкін В.Г. Вивчення мови HTML з використанням каскаду стилів CSS та елементів мови JAVASCRIPT. Частина 2: Навч. посібник (рос. мовою). – Дніпропетровськ: НМетАУ, 2012. – 76 с.

Викладені теоретичні відомості і практичні рекомендації щодо створення Web-документів та їх використання за допомогою провідних інтернет-технологій. Основна увага приділяється особливостям використання інструментальних засобів HTML та JavaScript у середовищі CSS.

Призначений для студентів напряму 6.020105 – документознавство та інформаційна діяльність.

Іл. 4. Бібліогр.: 4 найм.

Друкується за авторською редакцією.

Відповідальний за випуск Г.Г. Швачич, канд. техн. наук, проф.

Рецензенти: О.О. Івлєв, канд. техн. наук, проф. (ДХТУ)
А.Б. Устенко, канд. техн. наук, доц. (ДНУЗТ)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

CSS (Cascading Style Sheets) - это специализированный язык программирования, язык стилей, определяющий отображение HTML-документов. Как и любой другой язык программирования, CSS имеет строго определенный синтаксис, т.е. правила, по которым создаются таблицы стилей и определяется вид того или иного элемента в документе. CSS работает с шрифтами, полями, строками, цветом, фоновыми изображениями, позиционированием элементов и др. CSS предоставляет бóльшие возможности для оформления web-сайтов чем «чистый» HTML; он более проработан и поддерживается всеми браузерами (программами просмотра).

В чём разница между CSS и HTML? HTML используется для структурирования содержимого документа, CSS - для форматирования этого содержимого. По мере развития Web дизайнеры начали искать возможности форматирования онлайновых документов. Чтобы удовлетворить возросшим требованиям пользователей, производители браузеров ввели новые HTML-тэги, которые отличались от оригинальных HTML-тэгов тем, что они определяли внешний вид, а не структуру. Некоторые оригинальные тэги структурирования, например <table>, стали всё больше применяться для дизайна страниц вместо структурирования текста. Кроме того, потребовалась унификация некоторых тэгов дизайна, поддерживаемых только одним из браузеров.

CSS был создан для исправления этой ситуации путём предоставления web-дизайнерам возможностей необходимого дизайна, поддерживаемого всеми браузерами. Одновременно произошло разделение представления (формы) документа и его содержимого, что значительно упростило работу по созданию web-документа.

Пусть перед нами стоит задача создания сайта об интернет-технологиях, выдержанного в одном стиле (тип шрифта, фон, цвет гиперссылок и др.). Если конструировать сайт без CSS, то на каждой странице документа нужно будет указать параметры ссылок и фона, а для каждого параграфа указать еще и атрибуты шрифта.

При необходимости обновления дизайна сайта, в HTML придется заходить на каждую страницу документа и вносить соответствующие поправки. В CSS в отдельном файле создается стиль одинаковый для всех страниц, а на каждой странице пишется единственная строка, подключающая этот стиль, после чего на всех страницах документа дизайн меняется автоматически.

Основные преимущества CSS:

- управление дизайном нескольких документов с помощью одной таблицы стилей;
- более точный дизайн страниц, поддерживаемый всеми браузерами;
- разделение документа на две составляющие: структура и дизайн, благодаря чему исходные коды становятся более простыми и легко читаемыми;
- различные представления для разных носителей информации (экран, печать);
- проработанная техника дизайна и расширенные возможности по сравнению с простым HTML.

1. СИНТАКСИС И СТРУКТУРА CSS

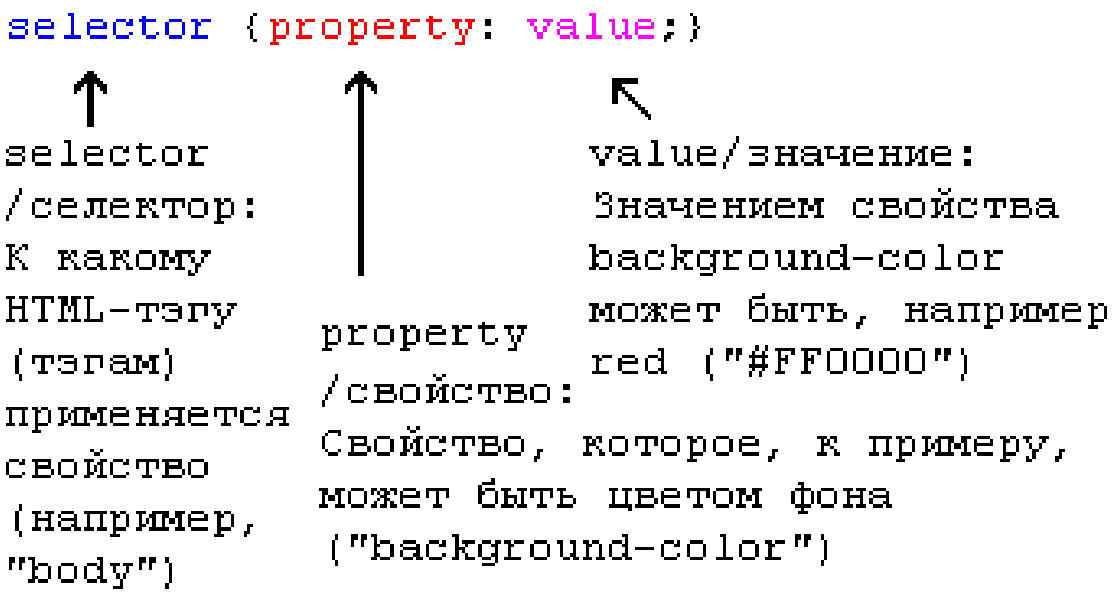
Рассмотрим конкретный пример**.** Допустим, нам нужен красный цвет фона web-страницы. В **HTML** это можно сделать так:

<body bgcolor="#FF0000">

С помощью **CSS** того же результата можно добиться так:

body {background-color: #FF0000;}

Как видите, эти коды более или менее идентичны в HTML и CSS. Этот пример также демонстрирует фундаментальную модель CSS:



Есть три способа применения CSS к HTML-документу. Мы рекомендуем сосредоточиться на методе **external** - то есть внешней таблице стилей.

Метод 1: Инлайн / In-line (атрибут style)

Можно применять CSS к HTML с помощью HTML-атрибута style. Красный цвет фона можно установить так:

```
<html>
<head>
<title>Example</title>
</head>
<body style="background-color: #FF0000;">
<p>This is a red page</p>
</body>
</html>
```

Метод 2: Внутренний (тэг style)

Второй способ вставки CSS-кодов - HTML-тэг <style>. Например:

```
<html>
<head>
<title>Example</title>
<style type="text/css">
body {background-color: #FF0000;}
</style>
</head>
<body>
<p>This is a red page</p>
</body>
</html>
```

Метод 3: Внешний (ссылка на таблицу стилей)

Рекомендуемый метод – создание ссылки на так называемую внешнюю таблицу стилей. В данном пособии мы будем использовать именно этот метод во всех примерах. Внешняя таблица стилей – это просто текстовый файл с

расширением **.css**. Вы можете поместить таблицу стилей на ваш web-сервер или на жёсткий диск, как и другие файлы. Например, ваша таблица стилей называется **style.css** и находится в папке **style**. Это можно проиллюстрировать так:

Задача состоит в том, чтобы создать ссылку из HTML-документа (default.htm) на таблицу стилей (style.css). Это можно сделать одной строчкой HTML-кода:

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

Обратите внимание: путь к вашей таблице стилей указан атрибутом href. Эту строку кода нужно вставлять в разделе header HTML, то есть между тэгами <head> и </head>. Например:

```
<html>
<head>
<title>My document</title>
<link rel="stylesheet" type="text/css" href="style/style.css" />
</head>
<body>
```

Эта ссылка указывает браузеру, что он должен использовать правила отображения HTML-файла из CSS-файла. Самое важное здесь то, что несколько HTML-документов могут ссылаться на одну таблицу стилей. Иначе говоря, один CSS-файл можно использовать для управления отображением множества HTML-документов,

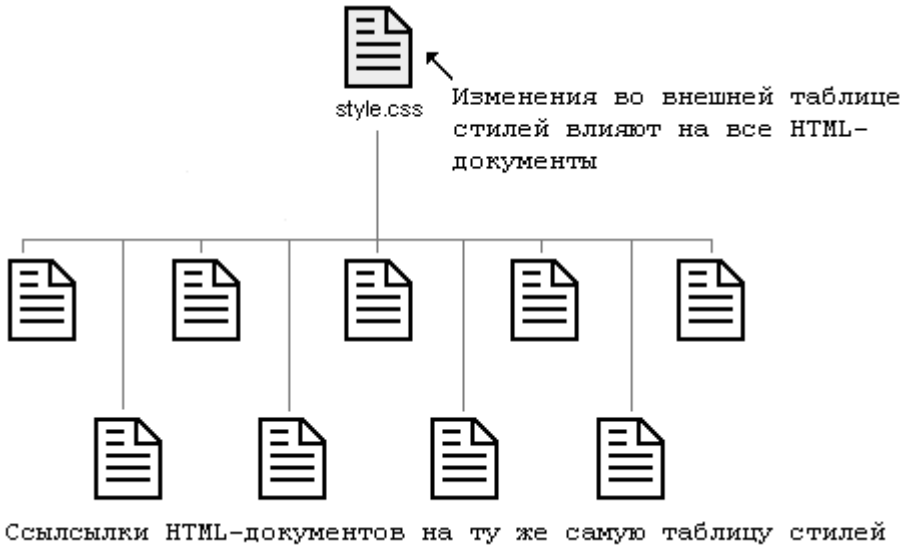


Рис.1.1

что поможет вам сэкономить время и силы. Если вы, например, хотите изменить цвет фона web-сайта из 100 страниц, таблица стилей избавит вас от необходимости вручную изменять все сто HTML-документов. Используя CSS, эти изменения можно сделать за несколько секунд, изменив один код в центральной таблице стилей.

Откройте Notepad (или другой ваш текстовый редактор) и создайте два файла - HTML-файл и CSS-файл - такого содержания:

default.htm

```
<html>
<head>
<title>Мой документ</title>
<link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
<h1>Моя первая таблица стилей</h1>
</body>
</html>
```

style.css

```
body {
  background-color: #FF0000;
}
```

Разместите эти файлы в одной папке. Не забудьте сохранить файлы с правильными расширениями (".css" и ".htm"). Открыв **default.htm** в вашем браузере, вы увидите, что страница имеет красный фон

1.1. Цвет и фон документа

Рассмотрим использование цвета и фона, а также методы позиционирования и управления фоновым изображением на страницах web-сайта. Поясним следующие CSS-свойства:

- color
- background-color
- background-image
- background-repeat
- background-attachment
- background-position
- background

Свойство **color** описывает цвет переднего плана элемента. Например, мы хотим сделать все заголовки документа тёмно-красными. Все заголовки обозначаются HTML-элементом <h1>. В нижеприведённом коде цвет элемента <h1> устанавливается красным.

```
h1 {
color: #ff0000;
}
```

Этот заголовок - красный

Iste quidem veteres inter ponetur honeste, qui vel mense brevi vel toto est iunior anno. Utor permisso, caudaeque pilos ut equinae paulatim vello unum, demo etiam unum, dum cadat elusus ratione ruentis acervi, qui redit in fastos et annis miraturque. Ennius et sapines et fortis et alter Homerus, ut critici dicunt, leviter curare videtur, quo promissa cadant et somnia Pythagorea.Naevius in manibus non est et mentibus haeret paene recens? Adeo sanctum est vetus omne poema. Ambigitur quotiens, sit prior, Pacuvius docti.

Цвета можно указывать как шестнадцатеричные значения, например (#ff0000), использовать названия цветов ("red") или rgb-значения (rgb(255,0,0)).

Свойство **background-color** описывает цвет фона элемента. В элементе <body> размещается всё содержимое HTML-документа. Таким образом, для изменения цвета фона всей страницы свойство **background-color** нужно применить к элементу <body>. Можно также применять это свойство к другим элементам, в том числе - к заголовкам и тексту. В следующем примере различные цвета фона применяются к элементам <body> и <h1>.

```
body {
background-color: #FFCC66;
}
h1 {
color: #990000;
background-color: #FC9804;
}
```

Заметьте, что для <h1> устанавливаются два свойства, разделяемые точкой с запятой.

CSS-свойство **background-image** используется для вставки фонового изображения. Воспользуемся изображением бабочки. Можно загрузить это изображение и использовать его на вашем компьютере (щёлкните правой

клавишей мыши на изображении и выберите "сохранить изображение как/save image as"), либо вы можете использовать другое изображение.

Для вставки рисунка бабочки (рис.1.2) в качестве фонового изображения web-страницы просто примените свойство **background-image** в тэге <body> и укажите местоположение рисунка.

Рис 1.2

```
body {background-color: #FFCC66;
background-image: url("butterfly.gif");
}

h1 {
color: #990000;
background-color: #FC9804;
}
```

Обратите внимание, что файл **url("butterfly.gif")** находится в той же папке, что и таблица стилей. Можно ссылаться и на файлы изображений в других папках, используя, например, **url("../images/butterfly.gif"),** или даже на файлы в Internet, указывая полный адрес файла: **url("http://www.html.net/butterfly.gif")**.

В предыдущем примере изображение бабочки повторяется по умолчанию по горизонтали и вертикали, заполняя весь экран. Этим управляет свойство **background-repeat**.

В таблице указаны четыре значения **background-repeat.**

Значение	Описание
Background-repeat: repeat-x	Рисунок повторяется по горизонтали
background-repeat: repeat-y	Рисунок повторяется по вертикали

background-repeat: repeat	Рисунок повторяется по горизонтали и вертикали
background-repeat: no-repeat	Рисунок не повторяется

Например, для отмены повторения/мультипликации фонового рисунка мы должны записать такой код:

```
body {
background-color: #FFCC66;
background-image: url("butterfly.gif");
background-repeat: no-repeat;
}
h1 {
color: #990000;
background-color: #FC9804;
}
```

Свойство **background-attachment** определяет, фиксируется ли фоновый рисунок, или прокручивается вместе с содержимым страницы.

В таблице указаны два значения **background-attachment**. Щёлкните на примере, чтобы почувствовать разницу между scroll и fixed.

Значение	Описание
Background-attachment: scroll	Изображение прокручивается вместе со страницей - разблокировано
Background-attachment: fixed	Изображение блокировано

Например, следующий код фиксирует изображение.

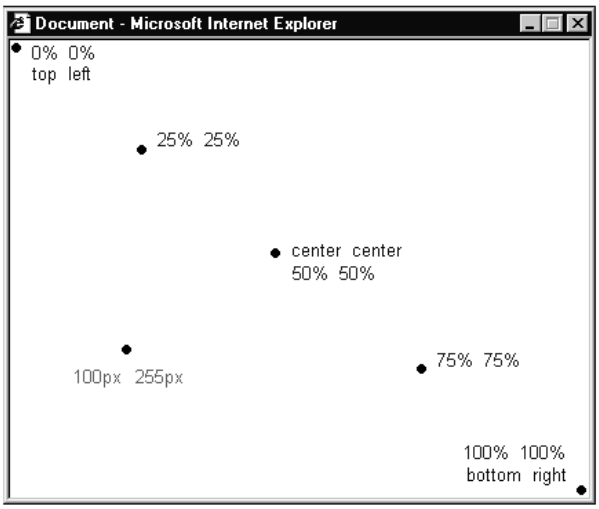
```
body {
background-color: #FFCC66;
background-image: url("butterfly.gif");
background-repeat: no-repeat;
background-attachment: fixed;
```

}

h1 {
color: #990000;
background-color: #FC9804;
}

1.2. Расположение фонового рисунка

По умолчанию фоновый рисунок позиционируется в левом верхнем углу экрана. Свойство **background-position** позволяет изменять это значение по умолчанию, и фоновый рисунок может располагаться в любом месте экрана.



Есть много способов установить значение **background-position**, но все они представляют собой набор координат. Например, значение '100px 200px' располагает фоновый рисунок на 100px слева и на 200px сверху в окне браузера.

Рис.1.3

Координаты можно указывать в процентах ширины экрана, в фиксированных единицах (пикселы, сантиметры, и т. п.), либо использовать слова top, bottom, center, left и right. В таблице дано несколько примеров.

Значение	Описание
background-position: 2cm 2cm	Рисунок расположен на 2 cm слева и на 2 cm сверху
background-position: 50% 25%	Рисунок расположен по центру и на четверть экрана сверху

background-position: top right	Рисунок расположен в правом верхнем углу страницы

В примере кода фоновое изображение располагается в правом нижнем углу экрана:

```
body {
background-color: #FFCC66;
background-image: url("butterfly.gif");
background-repeat: no-repeat;
background-attachment: fixed;
background-position: right bottom;
}
h1 {
color: #990000;
background-color: #FC9804;
}
```

С помощью **background** можно сжимать несколько свойств и записывать стили в сокращённом виде для облегчения чтения таблиц. Например:

```
background-color: #FFCC66;
background-image: url("butterfly.gif");
background-repeat: no-repeat;
background-attachment: fixed;
background-position: right bottom;
```

Того же результата можно достичь одной строкой кода:

```
background: #FFCC66 url("butterfly.gif") no-repeat fixed right bottom;
```

Если свойство отсутствует, оно автоматически получает значение по умолчанию. Например, если отсутствует **background-attachment** или **background-position,** тогда принимается

background: #FFCC66 url("butterfly.gif") no-repeat;

т.е. двум этим неспецифицированным свойствам будут присвоены значения по умолчанию - **scroll** и **top left**.

1.3. Шрифты

Этот раздел посвящен работе со шрифтами с помощью CSS Дано описание следующих CSS-свойств:

- font-family
- font-style
- font-variant
- font-weight
- font-size
- font

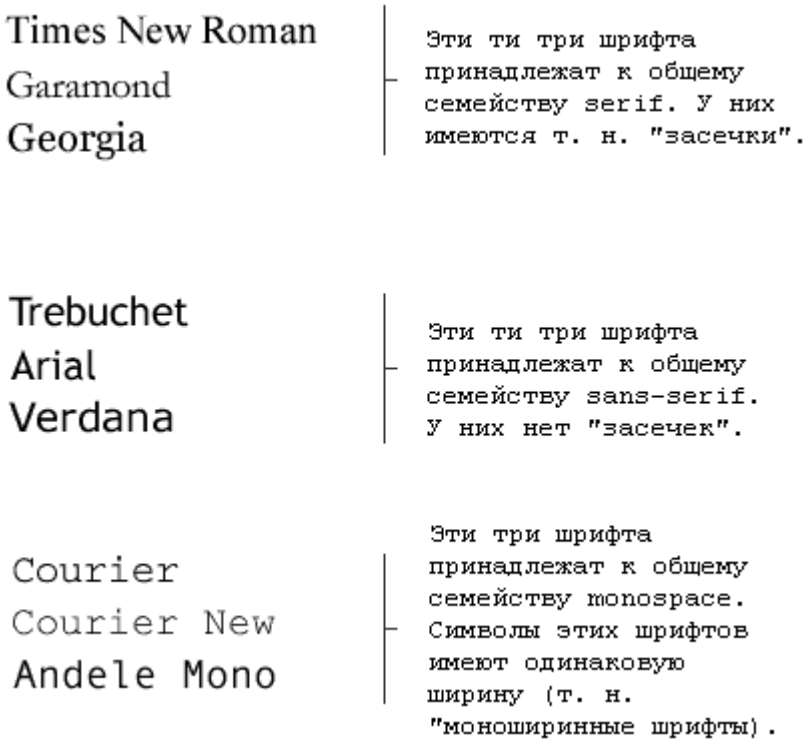
Свойство **font-family** указывает приоритетный список шрифтов, используемых для отображения данного элемента или web-страницы. Если первый шрифт списка не установлен на компьютере, с которого выполняется доступ к сайту, ищется следующий шрифт списка, пока не будет найден подходящий. Для категоризации шрифтов используются два типа имён: имя семейства /**family-name** и общее /**generic family**.

Family-name

Пример family-name (часто называемое просто "шрифт") это, например, "Arial", "Times New Roman" или "Tahoma".

Generic family

Его можно проще описать как группу **family-names**, имеющих характерные общие черты. Пример - sans-serif, набор шрифтов без "засечек /feet".



При указании шрифтов web-сайта начните с предпочтительного шрифта, а затем перечислите альтернативные. Рекомендуем в конце списка указывать родовое имя. Тогда страница будет отображена шрифтом того же семейства, если отсутствуют все специфицированные конкретные шрифты.

Список шрифтов может выглядеть так:

h1 {font-family: arial, verdana, sans-serif;}
h2 {font-family: "Times New Roman", serif;}

Заголовки <h1> будут отображаться шрифтом "Arial". Если он не установлен на пользовательской машине, будет использоваться "Verdana". Если недоступны оба шрифта, для показа заголовков будет использован шрифт семейства **sans-serif**. Обратите внимание, что имя шрифта "Times New Roman" содержит пробелы, поэтому указано в двойных кавычках.

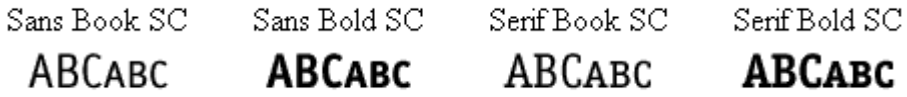
Свойство **font-style** определяет **normal**, **italic** или **oblique**. В примере все заголовки <h2> будут показаны курсивом *italic*.

h1 {font-family: arial, verdana, sans-serif;}
h2 {font-family: "Times New Roman", serif; font-style: italic;}

Заголовок 1 выведен шрифтом Arial

А заголовок 2 - Times New Roman - italic

Свойство **font-variant** используется для выбора между вариантами **normal** и **small-caps**. Шрифт **small-caps** использует малые заглавные буквы (upper case) вместо букв нижнего регистра, например:



Если **font-variant** имеет значение **small-caps**, а шрифт **small-caps** недоступен, браузер, скорее всего, отобразит текст буквами верхнего регистра.

h1 {font-variant: small-caps;}
h2 {font-variant: normal;}

Заголовок 1 - Small Caps

А заголовок 2 - normal

Свойство **font-weight** задает толщину или жирность шрифта. Шрифт может быть **normal** или **bold**. Некоторые браузеры поддерживают даже числовые значения 100-900 (в сотнях) для описания веса шрифта.

p {font-family: arial, verdana, sans-serif;}
td {font-family: arial, verdana, sans-serif; font-weight: bold;}

Текст в ячейках - жирным шрифтом

Размер шрифта устанавливается свойством **font-size**. Для описания размера шрифта используются различные единицы измерения (например,

пикселы и проценты). Воспользуемся самыми удобными единицами измерения, например:

h1 {font-size: 30px;}
h2 {font-size: 12pt;}
h3 {font-size: 120%;}
p {font-size: 1em;}

Заголовок <h1> размер 30px

Заголовок <h2> размер 1cm

Заголовок <h3> размер 120%

Параграф <p> размер 1em

Есть одно отличие в указанных единицах измерения: '**px**' и '**pt**' дают абсолютное значение размера шрифта, а '**%**' и '**em**' - относительные. Многие пользователи не могут читать мелкий текст, по разным причинам. **Чтобы сделать ваш web-сайт доступным для всех**, вы должны использовать относительные значения, такие как '**%**' или '**em**'.

Вот иллюстрация того, как настроить размер шрифта в Mozilla Firefox и Internet Explorer.

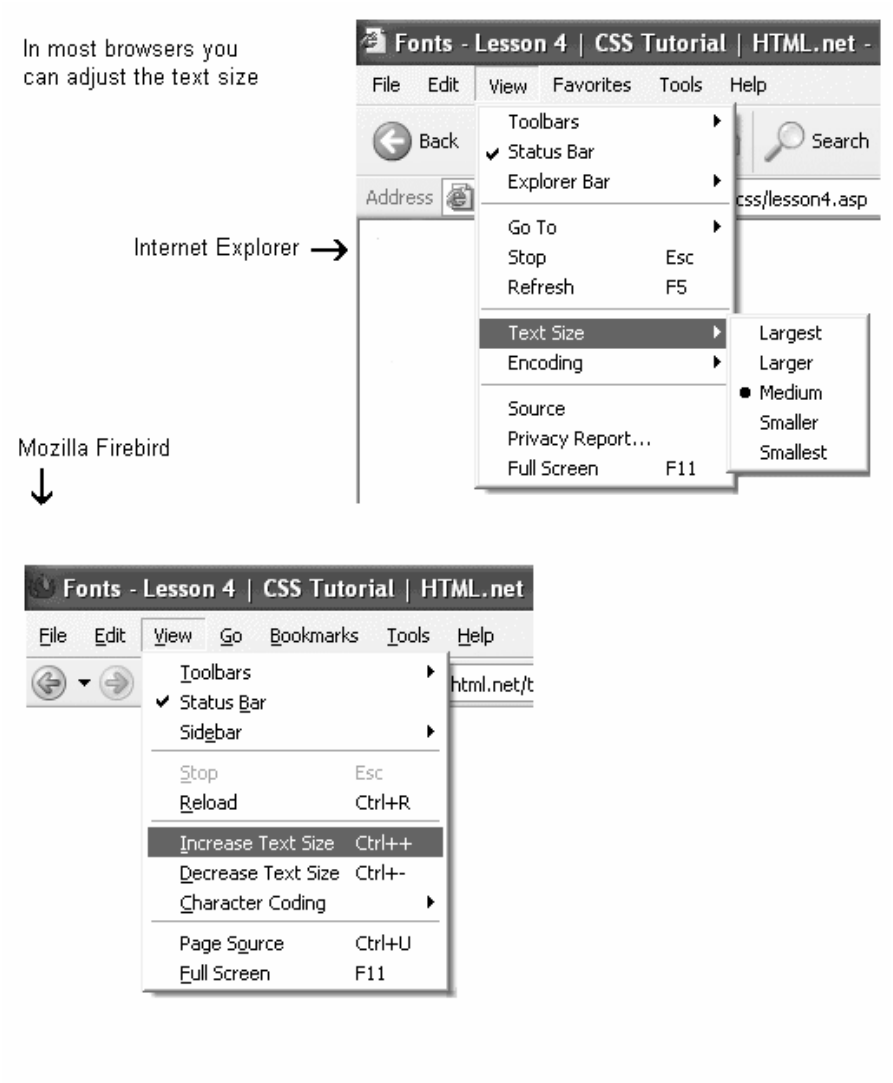


Рис.1.4

Используя сокращенную запись font, можно указывать все свойства шрифта в одном стиле, например:

```
p {
font-style: italic;
font-weight: bold;
font-size: 30px;
font-family: arial, sans-serif;
}
```

Используя сокращённую запись, код можно упростить:

p {
font: italic bold 30px arial, sans-serif;
}

Порядок свойств **font** таков:

font-style | font-variant | font-weight | font-size | font-family

1.4. Форматирование и установка стиля текста

Форматирование и установка стиля текста - ключевая проблема для любого web-дизайнера. Рассмотрим следующие свойства:

- text-indent
- text-align
- text-decoration
- letter-spacing
- text-transform

Свойство text-indent (отступы) выделяет параграф с помощью установки отступа для его первой строки. В следующем примере text-indent **30px** применяется ко всем параграфам <p>:

p {
text-indent: 30px;
}

Interdum volgus rectum videt, est ubi peccat. Si veteres ita miratur laudatque poetas, ut nihil anteferat, nihil illis comparet, errat. Si quaedam nimis antique, si peraque dure dicere credit eos, ignave multa fatetur, et sapit et mecum facit et Iova iudicat aequo.Non equidem insector delendave carmina Livi esse reor, memini quae plagosum mihi parvo Orbilium dictare; sed emendata videri pulchraque et exactis minimum distantia miror. Inter quae verbum emicuit si forte decorum, et si versus paulo concinnior unus et alter, venditque poema.

Indignor quicquam reprehendi, non quia cras

CSS-свойство **text-align** (выравнивание текста) соответствует атрибуту, используемому в старых версиях HTML. Текст может быть выровнен left, right, center (по центру) или justify (по ширине).

В примере текст заголовочных ячеек таблицы <th> выравнивается вправо, а в ячейках данных <td> - по центру. Кроме того, нормальные параграфы - ***justify:***

th {
text-align: right;
}
td {
text-align: center;
}
p {
text-align: justify;
}

Свойство text-decoration (декоративный вариант) позволяет добавлять различные "декоративные эффекты". Например, можно подчеркнуть текст, провести линию по или над текстом и т. д. В примере <h1> подчёркнуты, <h2> - имеют черту над текстом, а <h3> - перечёркнуты.

h1 {
text-decoration: **underline**;
}

h2 {

text-decoration: **overline**;
}

h3 {
text-decoration: line-through;
}

Этот текст подчёркнут

Этот текст надчёркнут

~~**Этот текст перечёркнут**~~

Интервал между буквами текста можно специфицировать свойством letter-spacing. Значение - нужная величина. Например, если необходимо задать интервал **3px** между буквами в параграфах <p> и **6px** - в заголовках <h1>, то используется такой код:

h1 {
letter-spacing: 6px;
}
p {
letter-spacing: **3px;**
}

Свойство text-transform управляет регистром символов. Можно выбрать **capitalize**, **uppercase** или **lowercase**, в зависимости от того, как выглядит текст в оригинальном HTML-коде.

Например, слово "headline" можно показать "HEADLINE" или "Headline". Имеются четыре возможных значения text-transform:

capitalize

Капитализирует каждое слово. Например: "john doe" станет "John Doe".

uppercase

Конвертирует все символы в верхний регистр. Например: "john doe" станет "JOHN DOE".

lowercase

Конвертирует все символы в нижний регистр. Например: "JOHN DOE" станет "john doe".

none

Трансформации нет - текст отображается так же, как в HTML-коде.

Для примера мы используем список имён. Все имена выделены с помощью <li> (list-item). Давайте капитализируем все имена и отобразим все заголовки верхним регистром.

Видите, HTML-код в этом примере в действительности записан в нижнем регистре.

```
h1 {
text-transform: uppercase;
}
li {
text-transform: capitalize;
}
```

этот заголовок - в верхнем регистре

- петер хансон
- макс ларсон
- джо доу
- пола джоунз
- моника левински
- доналд дак

2. ССЫЛКИ И ПСЕВДОКЛАССЫ

Всё изученное на предыдущих занятиях можно применять и для ссылок / links (например, изменять шрифт, цвет, подчёркивание и т. д.). Новым будет то, что в CSS эти свойства можно определять по-разному, в зависимости от того, посетили уже ссылку, активна ли она, находится ли указатель мыши над ссылкой. Это позволяет добавить интересные эффекты на ваш web-сайт. Для этого используются так называемые псевдоклассы. Псевдокласс позволяет учитывать различные условия или события при определении свойств HTML-тэга. Рассмотрим пример.

Как вы знаете, ссылки специфицируются в HTML тэгом <a>. В CSS мы также можем использовать **a** в качестве селектора:

```
a {
color: blue;
}
```

Ссылка может иметь разные состояния: link (просто ссылка) и visited (посещенная). Можно использовать псевдоклассы для установки разных стилей посещённых и непосещённых ссылок.

```
a:link {
color: blue;
}
a:visited {
color: red;
}
```

Используйте |a:link| и |a:visited| для непосещённых и посещённых ссылок, соответственно. Активные ссылки имеют псевдокласс a:active, и a:hover, когда указатель - над ссылкой. Мы рассмотрим каждый из четырёх псевдоклассов на примерах.

Псевдокласс: link

Псевдокласс :link используется для ссылок на страницы, которые пользователь ещё не посещал.

В примере кода непосещённые ссылки - синие.

```
a:link {
color: #6699CC; }
```

Псевдокласс: visited

Псевдокласс :visited используется для ссылок на страницы, которые пользователь посетил. В примере кода посещённые ссылки - фиолетовые.

```
a:visited {
color: #660099;
}
```

Псевдокласс: active

Псевдокласс :active используется для активных ссылок.

В примере активные ссылки имеют жёлтый фон.

```
a:active {
background-color: #FFFF00;
}
```

Псевдокласс: hover

Псевдокласс :hover используется для ссылок, над которыми находится указатель мыши. С помощью псевдокласса :hover можно создавать интересные эффекты. Например, если мы хотим, чтобы ссылки становились оранжевыми и курсивными при прохождении указателя над ними, то CSS –код должен выглядеть так:

```
a:hover {
color: orange;
font-style: italic;
}
```

Свойство text-decoration можно использовать для определения подчёркивания текста. Для удаления подчёркивания установите в text-decoration значение none.

```
a {
text-decoration:none;
}
```

Альтернативно можно также установить text-decoration, наряду с другими свойствами, для всех четырёх псевдоклассов.

```
a:link {
color: blue;
text-decoration:none;
}
a:visited {
color: purple;
text-decoration:none;
}
a:active {
background-color: yellow;
text-decoration:none;
}
a:hover {
color:red;
text-decoration:none;
}
```

2.1. Идентификация и группирование элементов

Иногда необходимо применить особый стиль к определённому элементу или конкретной группе элементов. В этом разделе мы подробно разберём использование class и id для специфицирования свойств выбранных элементов.

Группирование элементов с помощью class

Предположим, у нас есть два списка ссылок сортов винограда - для белого и для красного вина. HTML-код может быть таким:

```
<p>Виноград для белого вина:</p>
<ul>
<li><a href="ri.htm">Рислинг</a></li>
<li><a href="ch.htm">Шардоне</a></li>
<li><a href="pb.htm">Пино Блан</a></li>
</ul>

<p>Виноград для красного вина:</p>
<ul>
<li><a href="cs.htm">Каберне Совиньон</a></li>
<li><a href="me.htm">Мерло</a></li>
<li><a href="pn.htm">Пино Нуар</a></li>
</ul>
```

Далее, мы хотим, чтобы ссылки на белое вино были жёлтого цвета, на красное вино - красного, а остальные ссылки на этой же странице оставались синими. Для достижения этой цели мы разделим ссылки на две категории с помощью присвоения класса каждой ссылке атрибутом class.

Установим классы для предыдущего примера:

```
<p>Виноград для белого вина:</p>
<ul>
<li><a href="ri.htm" class="whitewine">Рислинг</a></li>
<li><a href="ch.htm" class="whitewine">Шардонэ</a></li>
<li><a href="pb.htm" class="whitewine">Пино Блан</a></li>
</ul>
<p>Виноград для красного вина:</p>
<ul>
<li><a href="cs.htm" class="redwine">Каберне Совиньон</a></li>
<li><a href="me.htm" class="redwine">Мерло</a></li>
<li><a href="pn.htm" class="redwine">Пино Нуар</a></li>
</ul>
```

Далее можем определить специальные свойства для ссылок **whitewine** и **redwine**, соответственно.

```
a {
color: blue;
}
a.whitewine {
color: #FFBB00;
}
a.redwine {
color: #800000;
}
```

Результат:

Виноград для белого вина: цвет ссылок желтый

- Рислинг
- Шардоне
- Пино Блан

Виноград для красного вина: цвет ссылок темно-бордовый

- Каберне Совиньон
- Мерло
- Пино Нуар

Ссылка без класса остаётся синей.

Как показано в примере, свойства можно определять для элементов, принадлежащих к определённому классу, с помощью **.имя_класса** в таблице стилей документа.

Элементы и используются для структурирования документа, часто совместно с атрибутами class и id. Рассмотрим их использование, поскольку эти элементы HTML имеют важное значение в CSS.

**Группирование с помощью **

Элемент можно назвать нейтральным элементом, который ничего не добавляет к содержимому документа. Но, в сочетании с CSS, может использоваться для визуальных эффектов применимо к отдельным блокам текста. Пример - цитата из Бенджамина Франклина:

** Early to bed, early to rise, make a man healthy, wealthy and wise **

Допустим, мы хотим показать преимущества бодрствования. Отметим эти преимущества с помощью . Каждому блоку span будет присвоен class, который затем можно определить в нашей таблице стилей:

Кто рано ложится и рано встаёт,
будет здоровым,
успешным
и **мудрым**.

В CSS:

span.benefit {
color:red;

}

Можно также использовать **id** для определения стиля -элементов. Необходимо только помнить, что вы должны установить уникальный **id** каждому из трёх -элементов.

**Группирование с помощью **

В то время как используется в элементах уровня блока (см. предыдущий пример), применяется для группирования одного или более блок-элементов. Рассмотрим, например, два списка президентов США, сгруппированных по политической принадлежности:

```
<div id="democrats">
<ul>
<li>Франклин Д. Рузвелт</li>
<li>Гарри Трумэн</li>
<li>Джон Ф. Кеннеди</li>
<li>Линдон Б. Джонсон</li>
<li>Джимми Картер</li>
<li>Билл Клинтон</li>
</ul>
</div>
<div id="republicans">
<ul>
<li>Дуайт Д. Эйзенхауэр</li>
<li>Ричард Никсон</li>
<li>Джэралд Форд</li>
<li>Роналд Рейган</li>
<li>Джордж Буш</li>
<li>Джордж У. Буш</li>
</ul>
</div>
```

В таблице стилей можно использовать такое же группирование:

```
#democrats {
background:blue;
}
#republicans {
background:red;
}
```

Президенты США

- Франклин Д. Рузвелт
- Гарри Трумэн
- Джон Ф. Кеннеди
- Линдон Б. Джонсон (Blue)
- Джимми Картер
- Билл Клинтон

- Дуайт Д. Эйзенхауэр
- Ричард Никсон
- Джэралд Форд
- Роналд Рейган (Red)
- Джордж Буш
- Джордж У. Буш

В этих примерах мы использовали и для определения цвета текста и фона. Но оба элемента несут в себе потенциал и для более сложных операций.

Идентификация элемента с помощью id

Помимо группирования элементов может понадобиться идентифицировать один уникальный элемент. Это реализуется с помощью атрибута id. Особенность id в том, что в документе не может быть более одного элемента с данным конкретным id. Каждый id должен быть уникальным. В других случаях используйте атрибут class. Рассмотрим пример использования id:

<h1>Глава 1</h1>

...

<h2>Глава 1.1</h2>

...

<h2>Глава 1.2</h2>

...

<h1>Глава 2</h1>

...

<h2>Глава 2.1</h2>

...

<h3>Глава 2.1.2</h3>

Это могут быть заголовки документа, разделённого на главы или параграфы. Естественным будет назначить id каждой главе:

<h1 id="c1">Глава 1</h1>

...

<h2 id="c1-1">Глава 1.1</h2>

...

<h2 id="c1-2">Глава 1.2</h2>

...

<h1 id="c2">Глава 2</h1>

...

<h2 id="c2-1">Глава 2.1</h2>

...

<h3 id="c2-1-2">Глава 2.1.2</h3>

Например, заголовок chapter1.2, должен быть красным. Это делается в соответствии с CSS:

#c1-2 {

color: red;

}

Как показано в предыдущем примере, свойства конкретного элемента можно определять с помощью #id в таблице стилей документа.

2.6. Поля и заполнение

Рассмотрим, как можно изменять представление элементов свойствами margin и padding. У элемента есть четыре стороны: right, left, top и bottom. Поля margin - это расстояние от каждой стороны с до соседних элементов (или краёв документа). В качестве первого примера разберём, как определить поля самогó документа, т. е. элемента <body>. На иллюстрации показано, какие поля нам нужны.

CSS-код для этого примера выглядит так:

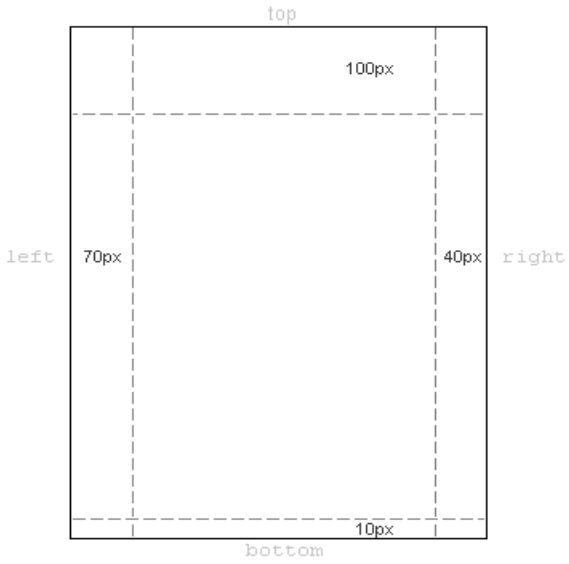
```
body {
margin-top: 100px;
margin-right: 40px;
margin-bottom: 10px;
margin-left: 70px;
}
```

или более элегантно:

```
body {
margin: 100px 40px 10px 70px;
}
```

Таким же образом можно установить поля для любого элемента. Например, можно определить поля для всех параграфов <p>:

```
body {
  margin: 100px 40px 10px 70px;
  }
  p {
```



margin: 5px 50px 5px 50px;
}

Заполнение не влияет на расстояние между элементами, а лишь определяет внутреннее расстояние между рамкой и содержимым элемента. Использование заполнения /padding можно показать на простом примере, где все заголовки имеют цветной фон (желтый или оранжевый):

```
h1 {
background: yellow;
}
h2 {
background: orange;
}
```

Определяя заполнение для заголовков, вы устанавливаете величину поля вокруг текста каждого заголовка:

```
h1 {
background: yellow;
padding: 20px 20px 20px 80px;
}
h2 {
background: orange;
padding-left:120px;
}
```

2.7. Рамки

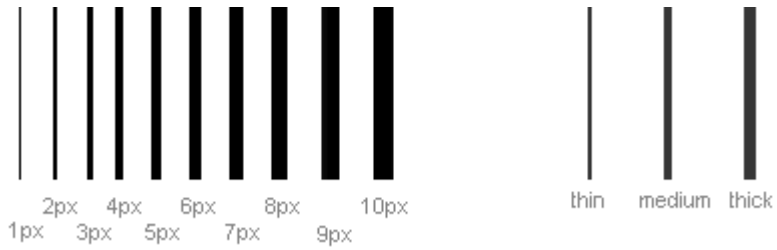
Рамки имеют многообразное применение, например, как декоративный элемент или для отделения двух объектов. CSS предоставляет бесчисленное множество вариантов использования рамок.

- border-width
- border-color

- border-style
- Примеры определения рамок
- border

Толщина рамки [border-width]

Толщина рамки определяется свойством border-width, которое может иметь значения thin, medium и thick, или числовое значение в пикселах. На рисунке показана эта система:

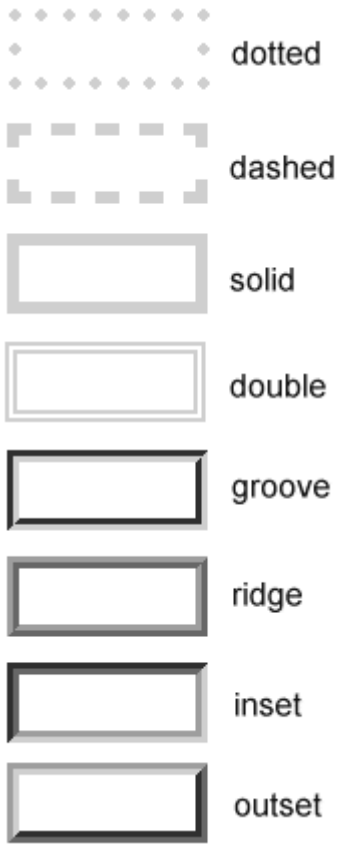


Цвет рамки [border-color]

Свойство border-color определяет цвет рамки. Значения - нормальные значения цвета, например: "#123456", "rgb(123,123,123)" или "yellow" .

Типы рамок [border-style]

Существуют различные типы рамок. Ниже показаны восемь типов рамок и их интерпретация в Internet Explorer 5.5. Все примеры показаны цветом "gold" и толщиной "thick", но могут выводиться с другими параметрами. Значения none или hidden могут использоваться, если вы не хотите отображать рамку.



Примеры определения рамок

Три рассмотренных выше свойства можно объединить в каждом элементе и, соответственно, устанавливать разные рамки. Для иллюстрации покажем документ, где определены разные рамки для <h1>, <h2>, <ul> и <p>.

```
h1 {
border-width: thick;
border-style: dotted;
border-color: gold;
}

h2 {
border-width: 20px;
border-style: outset;
border-color: red;
}
p {
border-width: 1px;
```

```
border-style: dashed;
border-color: blue;
}
ul {
border-width: thin;
border-style: solid;
border-color: orange;
}
```

При необходимости можно объединять несколько свойств в одно, используя слово border, например:

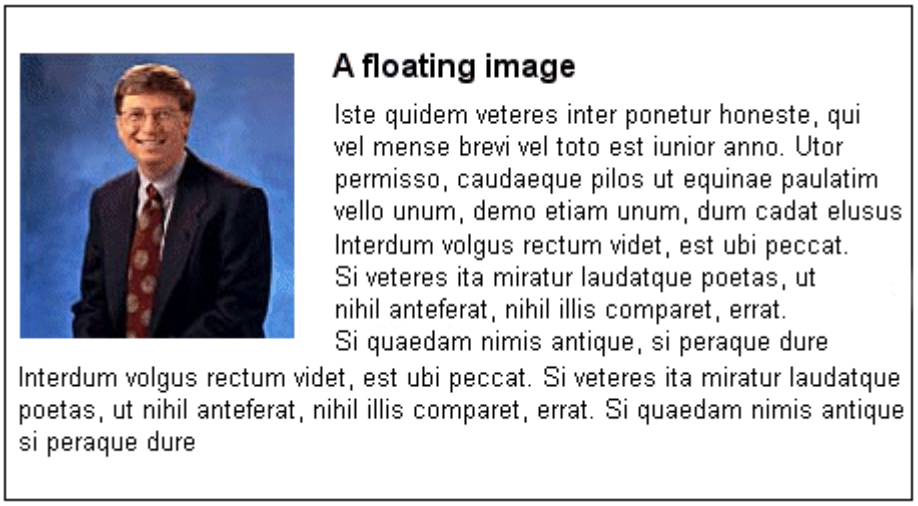
```
p {
border-width: 1px;
border-style: solid;
border-color: blue;
}
```

или

```
p {
border: 1px solid blue;
}
```

2.8. Всплывающие элементы (поплавки)

Если необходимо получить следующий рисунок:



HTML-код для этого примера:

```
<div id="picture">
<img src="bill.jpg" alt="Bill Gates">
</div>
<p>causas naturales et antecedentes,
idciro etiam nostrarum voluntatum...</p>
```

Чтобы рисунок всплывал влево, а текст окружал его, необходимо определить ширину бокса вокруг рисунка, и установить в свойстве float значение left:

```
#picture {
float:left;
width: 100px;
}
```

Поплавки/Floats можно использовать для вывода колонок в документе. Для этого вы определяете необходимые колонки в HTML-коде тэгами таким образом:

```
<div id="column1">
<p>Haec disserens qua de re agatur
et in quo causa consistat non videt...</p>
</div>
<div id="column2">
<p>causas naturales et antecedentes,
idciro etiam nostrarum voluntatum...</p>
</div>
<div id="column3">
<p>nam nihil esset in nostra
potestate si res ita se haberet...</p>
</div>
```

Теперь необходимую ширину колонок установим, например, 33%, а затем установим всплывание каждой влево в свойстве float (float может иметь значения **left**, **right** или **none**.)

```
#column1 {
float:left;
width: 33%;
}
#column2 {
float:left;
width: 33%;
        }

        #column3 {
        float:left;
        width: 33%;
        }
```

КОЛОНКИ

Iste quidem veteres inter ponetur honeste, qui vel mense brevi vel toto est iunior anno. Utor permisso, caudaeque pilos ut equinae paulatim vello unum, demo etiam unum, dum cadat elusus ratione ruentis acervi, qui redit in fastos et annis miraturque.

Indignor quicquam reprehendi, non quia crasse compositum illepedeve putetur, sed quia nuper, nec veniam antiquis, sed honorem

Brevi vel toto est iunior anno. Utor permisso, caudaeque pilos ut equinae paulatim vello unum, demo etiam unum. Si meliora dies, ut vina, poemata reddit, scire velim, chartis perficit quotus pretium quotus arroget annus. Scriptor abhinc reddit misso annos centum qui decidit, inter perfectos veteresque referri debet an inter vilis atque perfectos novos? Excludat iurgia finis. мEst vetus atque

probus, centum qui perficit annos.o Quid, qui deperiitnihis perfectos uno mense vel?мIste quidem veteres inter ponetur honeste, qui vel mense brevi vel toto est iunior anno.o Utor permisso, caudaeque nisi pilos ut equinae paulatim vello et virtutem, demo etiam unum, dum cadat elusus ratione ruentis acervi, qui redit in fastos et virtutem

aestimat annis miraturque nihil nisi quod.Ennius et sapines et fortis et alter Homerus, ut critici dicunt, leviter curare videtur

Brevi vel toto est iunior anno. Utor permisso, caudaeque pilos ut equinae paulatim vello unum, demo etiam unum. Si meliora dies, ut vina, poemata reddit, scire velim, chartis perficit quotus pretium quotus arroget annus. Scriptor abhinc reddit messe, is non videt quae quamque rem res consequatur. Haec disserens qua de re agatur et in quo causa consistat non videt. Non enim si alii ad alia propensiores sunt propter causas naturales et antecedentes, idciro etiam nostrarum voluntatum atque appetitionum sunt causae naturales at antecedentes; nam nihil esset in nostra potestate si res ita se haberet.

Свойство clear

Свойство clear управляет перемещением всплывающих элементов документа. По умолчанию последовательные элементы смещаются вверх, заполняя пространство, которое освобождается, если бокс всплывает в сторону (текст автоматически смещается вверх вдоль изображения Била Гейтса). Свойство clear может иметь значения **left**, **right**, **both** или **none**. Принцип таков, что если clear, например, имеет для бокса значение both, то верхний край рамки этого бокса всегда будет находиться под нижним краем поля возможных всплывающих сверху боксов.

```
<div id="picture">
<img src="bill.jpg" alt="Bill Gates">
</div>
<h1>Bill Gates</h1>
<p class="floatstop">causas naturales et antecedentes,
idciro etiam nostrarum voluntatum...</p>
```

Чтобы не дать тексту всплывать вверх перед рисунком, необходимо добавить такой код CSS:

```
#picture {
float:left;
width: 100px;
}

.floatstop {
clear:both;
}
```

Если мы хотим расположить его на 100px от верхней границы документа и на 200px слева, мы должны ввести следующий код CSS:

```
h1 {
```

position:absolute;
top: 100px;
left: 200px;
}

2.9. Web-стандарты и проверка кода

W3C это World Wide Web Consortium, независимая организация по разработке стандартов кодов Internet (например, HTML, CSS, XML и др.). Microsoft, The Mozilla Foundation и многие другие являются членами W3C и работают по соглашению о перспективах развития этих стандартов.

Если вы работаете в сфере web-дизайна, то знаете, что страницы выглядят по-разному в различных браузерах. Это может отнимать массу времени при создании страниц, которые будут просматриваться в Mozilla, Internet Explorer, Opera и других существующих браузерах.

Идея стандартизации в том, чтобы заключить соглашение о развитии web-технологий. Это значит, что действуя в рамках стандартов, web-разработчик может быть уверен, что он работает в стиле, универсальном для различных платформ. **Поэтому мы рекомендуем, чтобы вы следили за работой W3C и проверяли ваш CSS на соответствие стандарту**.

CSS validator/ контролер

Для облегчения проверки на соответствие ***CSS-стандарту,*** W3C создал так называемый ***validator,*** который читает ваши таблицы стилей/stylesheet и выдает список предупреждений о нарушениях и ошибках, если ваш CSS их содержит.

Чтобы облегчить проверку ваших таблиц стилей, можно делать это прямо с данной web-страницы, заменив указанный URL на URL вашей таблицы стилей и кликнув для проведения проверки. Затем W3C-сайт проинформирует вас об ошибках.

http://www.html.net/

Click to check stylesheet

Если validator не найдёт ошибок; будет показано изображение, которое вы можете разместить на вашем сайте для иллюстрации того, что вы проверяете код:

3. БАЗОВЫЙ СИНТАКСИС JAVASCRIPT

JavaScript — объектно-ориентированный скриптовый язык программирования. Является диалектом языка ECMAScript[~ 1].

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке
Название «JavaScript» является зарегистрированным товарным знаком компании Oracle Corporation.

Возможности языка JavaScript обладает рядом свойств объектно-ориентированного языка, но реализованное в языке прототипирование обусловливает отличия в работе с объектами по сравнению с традиционными

объектно-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания, что придаёт языку дополнительную гибкость.

Несмотря на схожий с Си синтаксис, JavaScript по сравнению с языком Си имеет коренные отличия:

- объекты, с возможностью интроспекции;
- функции как объекты первого класса;
- автоматическое приведение типов;
- автоматическая сборка мусора;
- анонимные функции.

Однако языку присущи свои минусы:

- модульная система: JavaScript не предоставляет возможности управлять зависимостями и изоляцией областей видимости;
- стандартная библиотека: в частности, отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода/вывода, базовых типов для бинарных данных;
- стандартные интерфейсы к веб-серверам и базам данных;
- система управления пакетами, которая бы отслеживала зависимости и автоматически устанавливала их.

3.1. Способы ввода элементов JavaScript в документ HTML

Ввиду того, что язык HTML уже изучен, можно сразу приступить к рассмотрению простейшей программы на языке Javascript.

```
<script language="javascript">
        alert("Здесь был JavaScript");
</script>
```

Первая строка сообщает браузеру, что все, что находится дальше (то есть, до закрытия тэга) будет скриптом. Причем скрипт будет написан на языке JavaScript. Вообще, язык часто не указывается, и по умолчанию большинство браузеров считает, что это будет именно Javascript, но скрипты также могут быть написаны на языке VisualBasic, Tcl, PerlScript — в общем, на любом встраиваемом языке, который поддерживается браузером. Явное указание языка в виде language="LanguageName" считается устаревшей конструкцией, и вместо нее рекомендуется использовать, например, такое объявление скрипта:

<script type="text/javascript">,

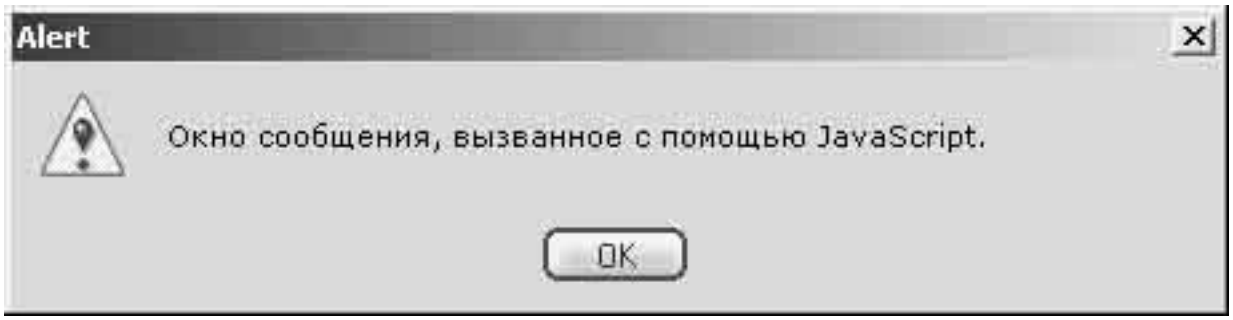
то есть вместо объявления языка указывается MIME-тип содержимого тега.

Следующая строка —

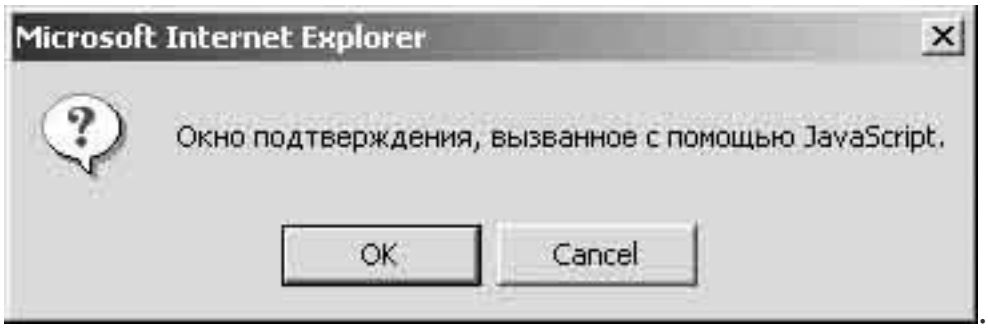
alert("Здесь был JavaScript");

собственно, и есть сам скрипт. Пока что это одна команда — вызов встроенной функции alert(). Эта функция поддерживается всеми браузерами, хотя каждый браузер реагирует на нее по-своему. Так, браузер Opera, например, обязательно кроме самого текста выводит еще и URL сайта, который вызвал открытие окна уведомления, и еще немного дополнительной информации. Internet Explorer выводит окно уведомления с заголовком «Internet Explorer», а Mozilla Firefox — с заголовком «**Javascript application**». Третья строка скрипта — </script> .Это обыкновенный закрывающий тег, ничем не отличающийся от других закрывающих тегов HTML. Он говорит браузеру, что скрипт закончился, и дальше будут идти другие теги языка HTML.

Та последовательность команд, которая выполняла необходимые нам действия, до сих пор не вызывала затруднений в понимании. Но если эта последовательность будет слишком сложна для того чтобы прочесть ее за один раз, следует воспользоваться одним из следующих методов



Окно предупредительных сообщений в браузере IE,FireFox



Окно подтверждения в браузере IE, FireFox

Существует четыре способа подключения скрипта.

1) Первый способ мы только что разобрали.

2)Второй мало чем от него отличается, кроме того, что самого скрипта в HTML-документе не будет. Этот скрипт будет записан в отдельном файле, например, в файле script.js (расширение является обязательным), находящимся рядом с главным HTML-скриптом. Для того чтобы браузер «знал», где лежит этот скрипт, нужно написать, например, такой код:

<script type="text/javascript" src="script.js"></script>

Здесь сам код скрипта загружается из указанного файла.

Более редко используемые способы вставки скрипта:

3) написание скрипта в параметре обработчика события какого-либо HTML элемента (например, onLoad у элемента <body>) — обычно используется исключительно для привязки события к скрипту, уже подключенного любым из предыдущих способов.

4) размещение скрипта прямо в строке браузера (т.е. в строке, в которой вводится адрес)

Комментарии в JS отличаются от комментариев в HTML:

//эта форма комментария

//действует только на одну строку

код скрипта...
/* А эта форма действует на любое количество строк, пока не встретится закрывающий знак */

далее код скрипта...

3.2. Типы и значения в JavaScript

Вернемся к функции **alert.** Как вы думаете, что будет, если написать **alert(123.45);** ? Правильно, браузер выведет окошко с текстом «123.45». То, что мы передавали в функцию, принято называть значениями.

И «"Здесь был JavaScript"», и «123.45», и «true» — все это значения. Имеются в виду строки, числа, логические константы, массивы — все, чем может оперировать JavaScript. Каждое значение имеет свой, четко определенный тип.

Так, «"Здесь был JavaScript"» — это строка, «123.45» — число с плавающей точкой, «true» — логическое значение. В целом Javascript умеет обращаться с 5 примитивными типами (их иногда называют простыми), а также с объектами, массивами и функциями.

Простые типы:

Number (Числа**), Boolean (логические** значения**), String (**строки**), null (**специальный тип — «ничто»**)** и **undefined** (специальный тип — «неопределенность»).

Стоит заметить, что у специальных типов название и значение совпадают. То есть у значения null тип null, а у значения undefined тип undefined.

3.3. Объекты и массивы

В JavaScript, как и в других алгоритмических языках ООП, существуют объекты и массивы. Можно записать целый массив (то есть — последовательность разных значений) в виде одного значения. Делается это так:

[1, "Здесь был JavaScript", true]— есть массив, содержащий число, строку и логическую константу. И его, как и любое другое значение, можно вывести при помощи функции alert.

Как и в других языках, у массивов номера значений, которые в него входят, называются индексами, а сами значения — элементами

Объекты записываются немного по-другому, например, так: {'a':1, 'b':2}

Каждое значение, которое входит в объект, называют свойством (то есть у объекта из примера будут два свойства с именами «a» и «b», и значениями 1 и 2. Тот текст, которым записывается значение, называется литералом.

Не нужно путать значение и литерал. Так, строка true (без кавычек) — это литерал, а логическое значение «Истина» — это и есть значение данного литерала.

Для того, чтобы браузер мог отличить одни типы от других, каждый тип записывается определенным образом.

- Для записи чисел могут использоваться цифры, десятичная точка и показатель экспоненты (то есть число 1000000 можно записать как 1.0e6). Кроме этого, можно использовать не десятичный, а, например, шестнадцатеричный код — тогда число 12, к примеру, будет записано как 0x0C
- Логические значения записываются строчками true и false, без кавычек.
- Текст внутри кавычек образует строковый литерал — то есть считается значением типа «строка».
- Массивы записываются с использованием квадратных скобок, а объекты — фигурных.

3.4. Переменные в JavaScript

Суть переменной проста — это такая сущность, которая может содержать какое-либо значение. Самое главное, что значение можно туда заносить и просматривать, какое значение там находится. Имя переменной в JavaScript может состоять из букв (латинских), цифр, знака подчеркивания и знака доллара (при этом цифра не может быть первым символом).

Например, такие строки, как qwerty, $_, a1b2c3$$ могут быть именами переменных в JS, а 123a — нет. При этом нужно учитывать, что регистр знаков важен — то есть abcd и aBcD будут разными переменными.

Последние версии JavaScript позволяют использовать любые национальные символы (русский, шведский, тибетский и так далее алфавиты), но для этого они должны быть записаны в Unicode.

Перед тем как переменную использовать, ее нужно объявить (то есть сказать браузеру, что такая переменная будет использоваться). Делается это при помощи описания var:

```
var x, y, z = 3;
```

Можно также просто присвоить значение переменной — в этом случае переменная тоже будет считаться объявленной. Использование же необъявленной переменной может вызвать сообщение об ошибке.

Существуют переменные, которые объявлять не нужно — это так называемые предопределенные переменные (о которых браузер знает с момента запуска). Любой браузер знает о таких переменных, как **window, document, location**, **navigator** и множестве других.

Бывает и так, что переменная уже объявлена, но в нее ничего не занесено. В том примере объявления переменных, который указан выше, значение занесено только в переменную **z**, но неизвестно, что находится в **x** и **y**, т.е. значение переменных не определено. Именно для этого и служит «очень специальный» тип **undefined**. Все не определенные переменные имеют этот тип и содержат это значение.

3.5. Операции и инструкции

Самая простая, и самая главная операция — присваивание. В Javascript эта операция записывается оператором «=». Например,

```
var a, b;
a = 1;
b = a;
```

При этом, как вы видите, тип переменной нигде не указывается — он определяется по типу того значения, которое в переменную занесено. Это называется динамической типизацией — в противовес статической, при которой тип переменной должен указываться при ее объявлении.

Язык JavaScript позволяет выполнять множество операций: сложение, вычитание, умножение, остаток от деления (или деление по модулю), а также логические — AND, OR, NOT, операции сдвигов, сравнения.

Все операции делятся на несколько групп:

- Операции сравнения. Записываются операторами <, >, <=, >=, ==, !=, ===, !==

  В отличие от привычной математической записи, для сравнения используется операция ==, так как использование простого знака равенства может ввести в заблуждение браузер (эта операция будет воспринята как присваивание).

  Также, операция неравенства записывается как != (а не как <>).

  Еще две операции — равенство и неравенство, но с учетом типа переменной (или тождественное равенство и неравенство) — так, например, значения 3 и "3" будут равны, поскольку содержат одно и то же число, но не тождественны, потому что их типы разные (первое — целое число, второе — строка).

- Арифметические операции. Это операции +, -, *, /, %, ++, -- и унарный минус.

Операция % возвращает остаток от целочисленного деления. Унарные операции ++ и --соответственно увеличивают или уменьшают значение переменной (эти операции требуют всего одну переменную). Унарный минус меняет знак числа на противоположный.

- Битовые операции. Так как практически вся вычислительная техника построена на двоичной системе счисления, то большинство языков имеет возможность работать с битовым, двоичным представлением числа. В Javascript для этого есть операции AND (&), OR (|), XOR (^), а также NOT (~). Кроме этого, есть операции побитового сдвига >>, << и >>>

- Логические операции. Эти операции воспринимают переменную как логическое значение, в отличие от битовых, которые с каждым битом работают отдельно. В Javascript есть операции AND (&&), OR (||) и NOT (!). В результате выполнения этих операций получается логическое true или false.

- Строковые операции. Это всего одна операция — «+». В результате ее выполнения две строки объединяются в одну.

- Операции присваивания. Это уже рассмотренная операция =, и операции, полученные комбинацией = и других операций. Например, += : если мы запишем, например, a += 1, то это будет значить то же самое, что и a = a + 1.

Также есть несколько специальных операций, которые будут рассмотрены позже.

Каждая операция получает один, два или три операнда (это могут быть как литералы, так и переменные), а в результате своего действия выдает значение. Так, операция 1+2 выдаст значение 3, которое может быть записано в переменную:

```
a = 1 + 2;
```

(теперь в «a» находится значение 3). При желании, результат операции можно никуда не записывать — например, 2 + 3; (результат операции — число «5», но этот результат нигде не сохранен).

Операция присваивания не является исключением, и тоже возвращает значение. Например, можно записать:

```
a = 1;
d = 1 + c = 2 + b = 3 + a;
```

Эта странная, на первый взгляд, запись, тем не менее нормально выполняется.

Но вместо нее лучше и понятнее записать традиционный оператор

```
a = 1;
b = 3 + a;
c = 2 + b;
d = 1 + c;
```

Иногда бывает так, что операция применяется к двум значениям, или иначе — операндам разного типа. Например, кто-нибудь может попытаться написать

```
var a = "13" - 1;
```

В таком случае JavaScript делает то, что называется «приведением типов», если, конечно, это возможно. То есть, у операндов (обычно одного) меняется тип, при возможности — сохраняя значение, и после замены типа выполняется операция над двумя операндами теперь уже одного типа.

Пример:

1. var a = "13" - 1;
2. var a = "13" + 1;

Первая инструкция занесет в переменную a число «12», а вторая — строку «131».

Это происходит потому что для строк (а первый операнд — строка) нет операции вычитания. Поэтому при вычитании строка преобразуется в число, и результат будет результатом вычитания числа из числа. В случае же сложения

используется операция объединения строк, при этом второй операнд преобразуется также в строку.

Рассмотрим еще один интересный пример

```
var a = true + false;
```

Хотя в этом примере оба операнда — логические, но для них операции сложения не существуют. Поэтому оба операнда преобразуются в числа, и после этого складываются, поэтому результатом будет единица (так как true преобразуется к единице, а false — к нулю).

В случае, если среда JavaScript не может выполнить приведение типов, она выдает ошибку.

Любая программа, и программа на Javascript — не исключение, состоит из списка команд, которые выполняются друг за другом. Обычно это операции над переменными и вызовы функций. Но чаще всего последовательное выполнение каждой команды — не совсем то, что требуется от программы. Одно из полезнейших свойств программ — способность выполнять различные действия в зависимости от состояния переменных. Для того чтобы программа работала по-разному в разных условиях, существуют инструкции.

Самые употребляемые инструкции — это инструкции циклов и условные инструкции.

Первые позволяют выполнять какое-либо действие до тех пор, пока некоторое условие верно (то есть, равно **true**). Вторые позволяют, в зависимости от условия, выполнять различные части кода.

Также имеются инструкции для работы с объектами (например, обхода всех элементов объекта), инструкции для исключений и блочная инструкция.

Для начала, рассмотрим блочную инструкцию, или блок.

Она записывается при помощи двух фигурных скобок: **{ }** (не стоит путать эту запись с литералом объекта). Благодаря этой инструкции любая

последовательность команд, операций и инструкций может выполняться как одна команда — это очень полезно при использовании остальных инструкций.

Для циклических действий в языке JavaScript существуют, во-первых, операторы циклов — это операции **for, while** и **do .. while**.

Оператор **for** записывается таким образом:

for (выражение инициализации; условие; выражение итерации) тело цикла;

Выражение инициализации — это выражение, которое вычисляется перед началом выполнения тела цикла (вы помните, что операция присвоения тоже вычисляется).

Условие — логическое выражение. Цикл выполняется до тех пор, пока это условие истинно. Выражение итерации вычисляется каждый раз после очередного выполнения тела цикла. Тело цикла — одна (и только одна) инструкция. Для того чтобы в теле цикла можно было использовать несколько инструкций, и используется блок. Считается хорошим тоном использовать блочную инструкцию даже в тех случаях, когда тело цикла состоит всего из одной команды.

Например, объявим цикл, который выполняется 30 раз:

```
for (var i = 0; i < 30; i++) {}
```

По сравнению с уже известными конструкциями **for …next** из Visual Basic и Pascal запись « i++» означает , что после каждой итерации значение **i** увеличивается на единицу. Таким образом, переменная **i** последовательно принимает значения от 0 до 29.

В случае, если условия или выражения не нужны, их можно не записывать. Например,

```
for (;;){}
```

Этот цикл будет выполняться «вечно» (то есть до тех пор, пока не произойдет внешнее событие, которое принудительно остановит цикл — например, пользователь закроет браузер).

Инструкция **while** в некотором роде может рассматриваться как упрощенный цикл. Эта инструкция имеет только параметр условия.

Инструкция

```
while (некоторое выражение) {}
```

полностью аналогична

```
for (;некоторое выражение;) {}
```

Оба этих цикла называются циклами с предусловием — то есть условие проверяется до того, как выполняется тело цикла. Если условие ложно до первого выполнения цикла, тело цикла никогда не выполнится

```
for (; false; ) {
        alert ('Это сообщение никогда не выводится');
}
alert ('А это — выведется');
```

Для того чтобы тело цикла выполнилось хотя бы один раз, существует специальный цикл (модификация **while**) — цикл с постусловием. В таком цикле условие проверяется уже после выполнения тела, и, независимо от того, истинное оно или ложное, тело выполнится как минимум один раз.

Например,

```
do {
        alert ('Всего один раз!');
} while (false);
```

Кроме описанных инструкций, циклические вычисления можно производить при помощи инструкции обхода объекта.

Пример:

```
for (variable in object) инструкция;

var a = {'a':10, 'b':20, 'c':30};
for (var index in a) {
        alert ('Просматриваем индекс ' + index);
}
```

В результате выполнения этих команд браузер выведет три окна сообщений, в каждом из которых будет указываться имя очередного свойства объекта.

Условные инструкции служат для того, чтобы, в зависимости от условий, выполнять одно или другое действие.

Таких выражений два **— if** и **switch**.

Аналогично конструкции условного оператора в языках VB и Pascal выражение **if** записывается в виде разветвляющегося процесса (ветвь else иногда можно опустить):

```
if (условие) {
        инструкция если значение верно
} else {
        инструкция если значение ложно
}
```

Для того, чтобы проверить несколько условий, можно каскадировать (то есть поставить друг за другом) инструкции **if**:

```
if (выражение 1) {
} else if (выражение 2) {
} else if (выражение 3) {
...
}else if (выражение n) {
}
```

Для случая, когда нужно использовать выбор из нескольких вариантов такое каскадирование получается громоздким. Вместо него можно использовать инструкцию switch. Она записывается так:

```
switch(выражение) {
        case значение1: инструкция1;
        case значение2: инструкция2;
        default: инструкция;
}
```

При выполнении этой инструкции будет выполняться тот блок кода, который стоит после case со значением, равным выражению.

Если же ни одно из значений не подходит, то выполняется блок после default (если он есть). Очень часто блок default не используется.

Впрочем, эта инструкция хранит в себе подводные камни. А именно — после того, как switch начнет выполнение тех инструкций, которые находятся после case с соответствующим значением, выполнение продолжится до конца инструкции switch.

Пример

```
var a = 1, b;
switch (a) {
        case 1: b = 1;
        case 2: b = 2;
        default: b = 0;
}
alert(b);
```

Результат: вывод окна с текстом «0».

Для того, чтобы в b содержалось правильное значение, нужно после каждого присваивания писать инструкцию break:

```
var a = 1, b;
switch (a) {
```

```
        case 1: b = 1;
                break;
        case 2: b = 2;
                break;
        default: b = 0;
        }
alert(b);
```

Теперь переменная b примет, как и ожидалось, значение 1 .

Инструкция **break (** инструкции прерывания и продолжения циклов) используется для того, чтобы завершить выполнение инструкции **switch,** и **для прерывания циклов**. Чаще всего это прерывание совершается по условию — к примеру, бесконечный цикл может завершаться при достижении переменной определенного значения (теперь вы понимаете, зачем могут понадобиться бесконечные циклы).

Инструкция **continue** похожа на **break** с той разницей, что **break** завершает выполнение цикла (причем все, что находится в теле цикла ниже инструкции **break,** не выполняется), а **continue** просто начинает новую итерацию цикла — как бы пропускает все операции, идущие ниже, как будто их нет.

Есть еще одна инструкция — это так называемая пустая. Записывается она в виде обычной точки с запятой — «;». Такая инструкция используется там, где ничего выполнять не нужно. Кроме того, этим же символом одна инструкция (или выражение) отделяется от другого.

3.6. Методы и функции JS

Рассмотрим один из простейших примеров, демонстрирующий использование метода

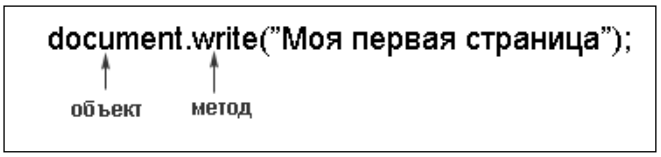
```
<html>
<script language="javascript">
<!-- код скрипта -->
 document.write("Моя первая страница.");
```

</script>
</ html>

Откроем эту страницу в браузере, она выглядит так:

Моя первая страница.

Давайте разберемся, как это работает. Браузер читает нашу html-страницу, видит оператор для выполнения document.write ("Моя первая страница."); и выполняет его. Рассмотрим, из чего состоит сам оператор (инструкция):



объект. метод( )

Те скрипты, которые мы рассматривали до этого времени, были очень небольшими, можно сказать – на один экран редактора. И весь код был виден одновременно. Но если представить, что скрипт занимает не пятнадцать строчек, а все полторы тысячи – как в нем разобраться? Или следующее: та последовательность команд, которая выполняла необходимые нам действия, до сих пор не вызывала затруднений в понимании. Но если эта последовательность будет слишком сложна для того чтобы прочесть ее за один раз...

Все эти проблемы решаются при помощи функций.

Что же такое функция? Если не вдаваться в подробности (а на первых порах эти подробности будут только мешать), функция представляет из себя некоторый блок кода, которому дали имя. Например, вот типичное объявление функции:

```
function diff(a,b) {
        if (a>b) {
                return a – b;
        } else {
                return b – a;
```

```
        }
}
```

В этом коде объявляется функция под именем **diff** (именно так к ней можно будет обращаться в тексте программы). Эта функция ожидает два параметра – **a** и **b**.

При вызове любые значения, которые были переданы функции, запишутся в переменные **a** и **b** внутри тела функции (кода между фигурными скобками), и к ним можно будет обращаться по этим именам.

Оператор **return** (это еще один оператор, который можно применять только внутри тела функции) завершает выполнение блока кода и возвращает значение, если оно есть.

Эта функция может быть вызвана вот так:

```
var c = diff(4, 7);
var d = diff(c, 12);
```

Как вы видите, функция, которая была объявлена выше, вызывается с двумя параметрами, и возвращает значение, которое можно записать в переменную или использовать любым другим образом.

При вызове функции нужно помнить, что эта функция должна быть объявлена или в текущем блоке кода, или в одном из предыдущих, иначе браузер выдаст ошибку. В приведенном примере один и тот же код был применен два раза, причем действия, которые он выполнил, зависели от того, что именно было передано в функцию.

В чем-то такие рекурсивные функции похожи на циклы (и большую часть циклов можно переписать в виде рекурсивных функций, и наоборот). В рекурсивной функции обязательно есть условие и какой-нибудь изменяемый параметр.

Функция в JavaScript тоже является **типом** и ее можно записать **в переменную**. Поначалу это кажется странным, особенно для тех, кто уже сталкивались с языками вроде C, Java или PHP.

Пример:

```
function fnc(a) {
        return a + 1;
}

var f = fnc;
```

Результат: созданная функция **fnc( )** запишется в переменную **f.**

При этом возможен как вызов функции напрямую, при помощи имени **fnc( ),** так и посредством переменной **f,** просто написав скобки за именем: f( )

Поскольку функция теперь является значением переменной, то это значение можно записывать в другие переменные и передавать в другие функции.

Некоторые стандартные функции.

Их, на удивление, не так уж и много. Большая часть полезных функций разнесена по встроенным объектам языка (и браузера). Но существует несколько таких функций, которые невозможно было отнести к какому-нибудь объекту, это:

- **eval()**: позволяет выполнить строку как javascript-код. Это бывает полезно, когда код приходит из внешнего источника (например, от сервера), или если он формируется в ходе выполнения скрипта.
- **parseInt(), parseFloat()**: эти функции предназначены для того, чтобы превращать строку в число (с учетом системы счисления). Например, можно свободно работать с двоичными, восьмеричными или (для оригиналов) с пятеричными числами.

Первая работает с целыми числами, вторая – с числами с плавающей точкой.

- **encodeURI(), decodeURI(), encodeURIComponent(), decodeURIComponent()**: эти функции предназначены для работы с URI (Uniform Resource Identifier). Этот стандарт требует, чтобы некоторые символы отсутствовали в URI, и указанные функции для преобразования этих символов.
- **isNaN()** : бывают случаи, когда числовая операция ни при каких условиях не может вернуть число. Например, при попытке преобразования строки «hello» с помощью функции parseInt. В таком случае используется специальное нечисловое значение (так называемое «не-число», NaN). Для проверки, является ли аргумент этим не-числом, и используется указанная функция.
- **isFinite()** : очень похожа на предыдущую, но кроме NaN также проверяет Infinite (то есть бесконечность).

В этом списке нет функции **alert().** Действительно, хотя мы и использовали эту функцию без предварительного объявления, и не указывали объект, к которому она относится, эта функция не является стандартной функцией языка. Кроме стандартных функций, без указания объекта могут вызываться функции объекта **window**. Вообще, любые глобальные переменные и функции являются переменными и функциями объекта **window.**

Также, все встроенные объекты языка JavaScript одновременно являются функциями (и наоборот). Это легче будет понять, после изучения как именно создаются объекты. Так что вполне допустимы вызовы вида

```
var a  = String(23);
```

Параметры функций. Области видимости. Значение и ссылка.

Кроме параметров внутри функции также могут использоваться глобальные переменные, и создаваться локальные.

Рассмотрим конструкцию:

```
function x(a,b,c) {
}
```
Что будет результатом x(1,2);

Нет, браузер не выдаст ошибку, хотя указано меньше параметров, чем предполагалось. Просто при вызове этой функции параметры **a** и **b** примут, соответственно, значения 1 и 2, а вот параметр **c** окажется **undefined**

Так, у любой функции после создания есть свойства и методы. Например, для объявленной выше функции x можно вызвать метод **toString():**

```
var a = x.toString();
```

И в результате в переменой a окажется строка «function x(a,b,c){}»

Внутри тела функции, кроме пришедших извне операндов, могут объявляться и другие переменные (при помощи оператора var). В этом случае переменная будет локальной – то есть она не будет видна снаружи функции.

Если же мы используем необъявленную внутри функции переменную, то это будет либо глобальная переменная (точнее, внешняя – будет использоваться переменная из той функции, внутри которой объявлена текущая, или из функции, в которой объявлена функция, в которой объявлена текущая функция, и так далее), либо свойство объекта window. Стоит заметить, что если объявлена локальная переменная, то доступ к глобальной переменной с тем же именем закрывается – все действия переменных будут происходить с локальной переменной. При этом аргументы функции автоматически становятся локальными переменными.

JavaScript также поддерживает отслеживание некоторых изменений и действий пользователя на странице. Каждое такое изменение или действие порождает событие, которое программист может перехватить и выполнить какие-то необходимые действия. Примеры событий - клик мыши на элементе страницы, получение фокуса ввода элементом формы, окончание загрузки или выгрузки страницы.

Список событий, доступных в каждом отдельном браузере обычно можно узнать на сайте компании-разработчика или в справочной документации. Список основных обработчиков определён стандартом W3C, но каждый разработчик браузера волен добавлять обработчики по своему желанию. Например, событие onAbort поддерживается IE, но не является стандартным и не поддерживается браузерами Netscape.

Наиболее популярные и полезные события перечислены в таблице:

События	
События форм и элементов страницы	
onchange()	Элемент теряет фокус ввода, а содержимое элемента изменилось за время, пока элемент был в фокусе.
onselect()	Какая-то часть текста внутри элемента становится выделенной.
ousubmit()	В форме нажата кнопка "Отправить", но отправка формы на сервер ещё не производилась.
События мыши	
onclick()	Произведён клик кнопкой мыши на элементе управления. Событие возникает после того, как кнопка мыши была отпущена.
onmousedown()	Нажата кнопка мыши.
onmousemove()	Указатель мыши движется внутри области отображения элемента.
onmouseout()	Указатель мыши вышел из области отображения элемента.
oumouseover()	Указатель мыши находится внутри области отображения элемента.
onmouseup()	Отжата кнопка мыши.

События окна (объект Window)	
onblur()	Элемент управления теряет фокус ввода, т.е. курсор переходит к другому элементу.
onfocus()	Отображаемый элемент получил фокуса ввода. Для текстовых полей это событие означает, что курсор уже находится в данном элементе.
onload()	Завершена загрузка страницы.
onunload()	Производится выход из документа (закрытие или перенаправление страницы на другой адрес).
События клавиатуры	
onkeydown()	Нажата кнопка на клавиатуре.
onkeypress()	Кнопка на клавиатуре нажата и не отпускается дольше, чем интервал повторения. Длительность интервала повторения является системным параметром и зависит от настроек операционной системы пользователя.
onkeyup()	Отпущена ранее нажатая кнопка.

Чтобы при возникновении события выполнить какие-то действия, необходимо предварительно указать обработчик этого события. Это можно сделать двумя путями: программно или непосредственно в HTML-коде.

Пример :

```
<head>

    <script type="text/javascript" language="javascript">

        //пример программного добавления события onload():

            function initVariables()
            {
```

```
            alert('документ загружен');
            return true;
        }

        window.onload = initVariables;

    </script>
    </head>
```

Результат: открытие нового окна, загрузка документа и выдача сообщения «документ загружен»

Аналогичным образом могут быть присвоены обработчики событий другим элементам (ссылкам, элементам списка, кнопкам и пр.).

Некоторые элементы уже имею встроенные обработчики по-умолчанию. Например, клик по кнопке типа **"submit"** отправляет форму на сервер, а клик по ссылке перенаправляет браузер по указанному адресу. Но иногда возникает необходимость отключить стандартный обработчик. Сделать это очень просто - достаточно в обработчике вернуть значение **false** или воспользоваться свойствами объекта **even**t.

3.7. Примеры применения JavaScript

Пример 1. Метод **write**

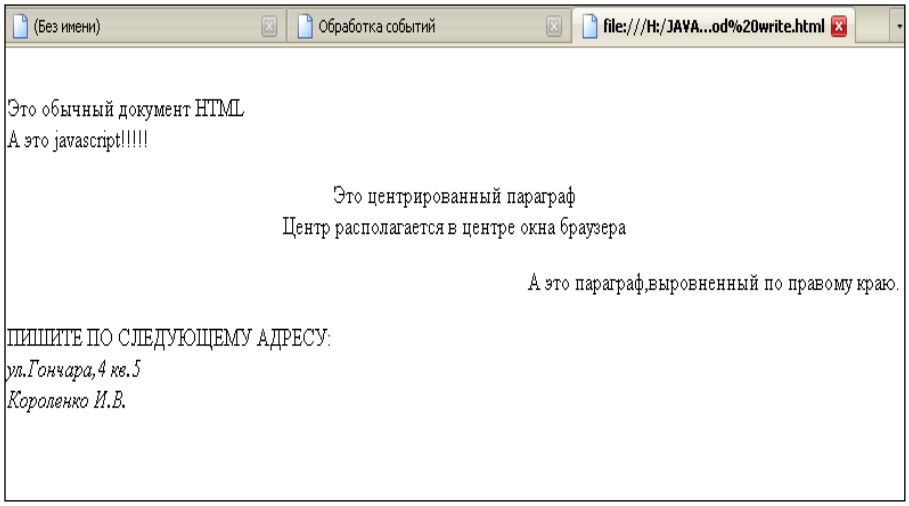
```
<html>
<body>
<br> Это обычный документ HTML<br>

<!-- код скрипта -->
<script language="javascript">
document.write("А это javascript !");
</script>
</ html>
<!--снова HTML -->
```

<P ALIGN="center">Это центрированный параграф.
Текст располагается в центре окна браузера</P>

<P ALIGN="right">А это параграф, выровненный по правому краю.</P>
Пишите по следующему адресу:
<ADDRESS> ул.
Гончара, 4 кв.5
Короленко И.В.
</ADDRESS>

</body>
</html>



Пример 2. Метод **Promt**

После первой фразы в круглых скобках поставьте запятую за пределами кавычек, а после нее впишите вторую фразу (тоже в кавычках). Это делается таким образом:

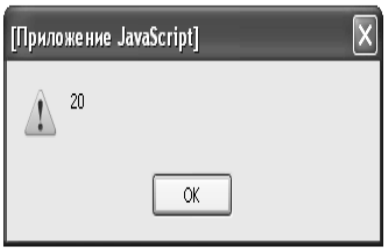
```
<html>
<head>
<title>Простая страница</title>
<script language="JavaScript">
prompt("Как вас зовут?", "Введите здесь ваше имя");
</script>
</head>
<body>
</body>
</html>
```

Следующая единица ввода - это текст по умолчанию, который должен появиться в соответствующем поле. Добавить его очень просто, а заодно это позволит избавиться от появления undefined в браузере Internet Explorer.

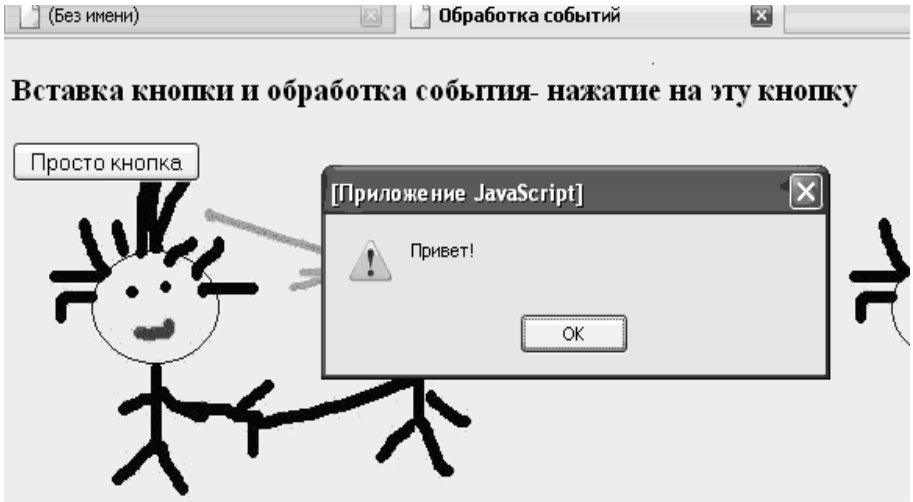
Сохраните вашу страницу и еще раз обновите вид в браузере. Обратите внимание, что поле ввода больше не является пустым, а содержит текст, указанный во второй паре кавычек

Пример 3. Простейшие вычисления и события в javascript

```
html>
<head>
<title>Пример 1</title>
</head>
</body>
<script type="text/javascript">
var r=5;
var z=r+10;
var y=z+r
alert(y);
</script>
</body>
</html>
```



Пример 4.Обработка событий в javascript



```
<html>
<form>
<title> Обработка событий </title>
<h3>Вставка кнопки и обработка события- нажатия на эту кнопку</h3>

<body>
<body bgcolor =”#ff0099”>
</body>

<input type="button" value="Просто кнопка"
onClick="alert('Привет!')">
</form>
</html>
```

Самостоятельно: добавить вторую кнопку по центру, выводящую текст До свидания!

Пример 5. История посещений

Использование объекта History предоставляет возможность возвращаться на URL, который был посещен перед этим (что эквивалентно щелчку на кнопке BACK), и переходить на URL, посещенный перед этим (что эквивалентно щелчку на кнопке FORWARD).Список посещенных URL содержится в менюGO.
Делается это, используя методы объекта History: **back()** и **forward().** Для этого в HTML-тег <a> включается следующая строка:

```
<a href="javascript:history.back()">НАЗАД</a>
```

или

```
<a href="javascript:history.forward()">ВПЕРЕД</a>
```

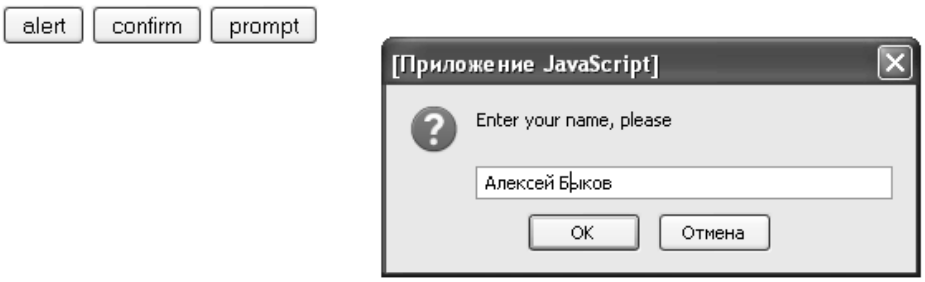
Если необходимо вернуться на несколько позиций списка меню **GO**, то используется метод **go( )**, в скобках указывается целочисленный аргумент

(отрицательное значение которого соответствует количеству шагов НАЗАД, положительное -ВПЕРЕД). Например, для возврата на три позиции назад указывается go(-3), вперед - go(3). Обратите внимание: если нет посещенных перед этим URL, то этот алгоритм не будет работать

Самостоятельно : добавить метод GO( ), соответствующий возврату на 2 шага разад.

Пример 6. Использование условной конструкции **if** и методов языка. Вывод окна сообщений

Используя методы alert, confirm, prompt можно выводить сообщения пользователю. Сообщение, выводимое alert, используется для вывода предупреждений пользователю. Метод confirm используется для сообщений, требующих принятия решения пользователем. При использовании prompt окно сообщений содержит само сообщение и поле ввода текста, который при нажатии кнопки "OK" может передаваться серверу или использоваться при вызове другого скрипта.



Для этого в теле документа в тэге <FORM> используются обработчики событий onClick и методы alert, prompt, а для confirm используется функция test, описанная в тэгах <SCRIPT>.

```
<script>

<script language="JavaScript">

function test() {

  if (confirm("If you want to close the window, press 'OK'?")) { window.close()} }
```

</script>

<form>

<input name=kuku type=submit value="alert" onClick="alert('Message for users')">

<input name=tutu type=submit value="confirm" onClick="test()">

<input name=nunu type=submit value="prompt" onClick="prompt('Enter your name, please',')">

</form>

Пример 7. Использование переменных и функций

<script type="text/javascript">

var str = "Простой текст"; //Создаём переменную str, которой придаём значение Простой текст

var func = function (text){return text.toUpperCase()} //Создаём переменную func, которая становится функцией с одним параметром text, значение text возвращается так, что все буквы становятся заглавными

alert (str + func("...и ещё")); //с помощью диалогового окна мы показываем результат смешания(сложения) значений переменных

</script>

Пример 8. Обработчик событий. Для изменения цвета и стиля шрифта данного текста нажмите и отпускайте кнопку мыши. Цвет надписи будет изменяться с синего на красный.

```
<html>

<head>

<meta http-equiv=Content-Type

content="text/html; charset=windows-1251">

</head>

<body lang=RU>

<H2> Изменение вида данного элемента</H2>

<b><p onmousedown="this. style.fontStule='italic';

this. style.color='red'" onmouseup="this.style.fontStyle='';

this.style.color='blue'">

Для изменения цвета и стиля шрифта данного текста нажмите и отпускайте кнопку мыши</p></b>

</body>

</html>
```

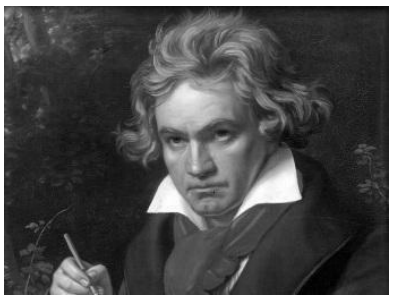
Пример 9. Смена графических объектов, обтекаемых текстом

Изменение вида данного элемента

Предварительно сохраните в текущей папке портреты И. Баха и Л. Бетховена файлы “bah.jpg"

и “bet.jpg" соответственно)

Для смены графического объекта переместите на него мышку. В результате изображение Иоганна Себастьяна Баха сменит Людвиг ван Бетховен, и так заданное количество раз.



Иоганн Себастьян Бах сменит Людвига ван Бетховена. Людвиг ван Бетховен сменит Иоганна Себастьяна Баха. Иоганн Себастьян Бах сменит Людвига ван Бетховена. Людвиг ван Бетховен сменит Иоганна Себастьяна Баха и.т.д.

```
<html>

<head>

<meta http-equiv=Content-Type

content="text/html; charset=windows-1251">

</head>

<body lang=RU>

<H2>Смена графических обьектов,обтекаемых текстом</H2>

<b>Для смены графического обьекта переместите на него мышку</b>

<p>Иоганн Себастьян Бах сменит Людвига ван Бетховена.

<a href="#"

OnMouseOver="B_B.src='bah.jpg'"

OnMouseOut="B_B.src='bet.jpg'">

<img id=B_B src="bah.jpg" border=0

align="left"></a>

Людвиг ван Бетховен сменит Иоганна Себастьяна Баха.

</html>
```

ЛИТЕРАТУРА

1. Муссиано Ч., Кеннеди Б. HTML и XHTML. Подробное руководство, 6-е издание. – М.: Издательство «Символ-Плюс», 2008.
2. Эрик А., Мейер И. CSS. Каскадные таблицы стилей. Подробное руководство, 3-е издание. – М.: Издательство «Символ-Плюс», 2008.
3. Рева О. Н. Использование HTML, JavaScript и CSS. Руководство Web-дизайнера. – М.: Издательство «Эксмо», 2008.
4. http://www.w3schools.com/html.

Навчальне видання

Нечухаєва Наталія Вікторівна
Расчубкін Віталій Геннадійович

ВИВЧЕННЯ МОВИ HTML З ВИКОРИСТАННЯМ КАСКАДУ СТИЛІВ CSS ТА ЕЛЕМЕНТІВ МОВИ JAVASCRIPT

Частина 2

Навчальний посібник

(російською мовою)

Тем. план 2012, поз.246

Підписано до друку 26.09.2012. Формат 60х84 1/16. Папір друк. Друк плоский. Облік.-вид. арк. 4,47. Умов. друк. арк. 4,41. Тираж 100 пр. Замовлення №

Національна металургійна академія України
49600, м. Дніпропетровськ-5, пр. Гагаріна,4

Редакційно-видавничий відділ НМетАУ