С.Н. Сорокин, Е.В. Девянина

ТЕХНОЛОГИИ ПОСТРОЕНИЯ

WEB-ИНТЕРФЕЙСОВ

ЧАСТЬ 1

Учебно-методическое пособие

Предисловие

При освоении дисциплины «Технологии построения web-интерфейсов» обучающимся необходимо учесть, что особенности разработки web-интерфейсов, изучаемые в рамках курса, а также навыки работы с программным обеспечением, использующимся на практических занятиях, имеют большое значение при работе в сфере современных информационных технологий. Также необходимо учитывать, что данные технологии находятся в постоянном развитии и знания по ним нуждаются в постоянной актуализации.

Учебная дисциплина «Технологии построения web-интерфейсов» направлена на формирование у слушателей представления о современных технологиях, используемых при создании интерфейсов web-приложений: языки разметки, каскадные таблицы стилей, язык программирования JavaScript. Это позволит в дальнейшем слушателям лучше усваивать материалы дисциплин, посвященных разработке web-приложений, исследованию их безопасности и защите. Поскольку учебный курс является базовым и читается на младших курсах, особое внимание уделяется правильному использованию компьютерной терминологии.

Лекционные занятия целесообразно проводить в аудитории, оборудованной мультимедиа оборудованием с демонстрацией использования различных инструментальных средств при построении web-интерфейсов. Для стимулирования обучающихся в регулярном освоении материала лекций желательно в начале каждой очередной лекции проводить краткий устный опрос по теме предыдущей лекции с вызовом к доске одного из них; а в начале каждого практического занятия проведение контрольного письменного опроса, с последующей оценкой ответов и выставлением оценок в журнал учебной группы. Кроме того, в журнале выставляются оценки за все лабораторные работы и за контрольную работу. Совокупность всех оценок помогает в ходе проведения зачёта объективно оценить уровень усвоения учебного материала каждым обучающимся.

Первая часть учебно-методического пособия состоит из трех глав, соответствующих основным темам дисциплины. Обучающимся важно придерживаться следующих методических рекомендаций.

В первой главе «Язык разметки XHTML» обучающимся необходимо получить представление о видах и различиях стандартов языка разметки гипертекста, структуре HTML-документа, основных тегах и их атрибутах. Особое внимание следует уделить изучению типов областей, формируемых различными тегами, а также принципам их отображения.

При изучении второй главы «Каскадные таблицы стилей» слушатели должны в полной мере усвоить способы определения CSS в документе, правила синтаксиса CSS, разновидности селекторов и набор основных CSS-правил. Наиболее важно при проработке материала главы научиться правильно каскадировать CSS-правила, для этого нужно учеть считать специфичность селекторов и правильно формировать контейнеры HTML-разметки с соответсвующим им блоком CSS-правил.

В третьей главе «Верстка сайтов» рассматриваются основные способы позиционирования элементов на странице, а также два базовых способа верстки сайтов − табличный и блочный. В этой теме обучающимся необходимо уяснить преимущества и недостатки каждого типа верстки, это в дальнейшем позволит правильно их сочетать.

В конце каждой главы приводится список вопросов и заданий для контроля уровня освоения дисциплины обучающимися.

Глава 1. Язык разметки XHTML

## Понятие HTML

HTML (HyperText Markup Language, язык разметки гипертекста) − это система верстки, которая определяет, как и какие элементы должны располагаться на web-странице. HTML изначально задумывался как язык, которому не нужны средства оформления, такие как цвет, размер, рамки или что-то подобное. Разработанный в Европейском институте физики частиц (CERN, Швейцария), HTML был инструментом сбора и консолидации информации ученых, которых прежде всего интересовала логика информации, а не её визуальное представление. Тогда ещё не существовало понятия web-дизайна и верстки как таковой, все сайты по своему оформлению были практически однотипными, в стиле, называемом сейчас «академический дизайн».

Однако пользователи преимущественно думали иначе и, глядя на красивую картинку, отдавали предпочтение именно ей. Разработчики браузера Netscape прекрасно понимали это и вводили в HTML новые теги, улучшающие внешний вид документа. Эти теги не были стандартизированы и работали только в Netscape, но это не имело особого значения, поскольку доля Netscape в то время составляла более 90% от всех существующих браузеров.

Разработчики сайтов поняли, что дизайн − это не просто разноцветный текст и вставленные наобум картинки. Это возможность красочно и эффектно подать материал, придать определённое настроение сайту, заинтересовать посетителя и задержать его на ресурсе. Дизайн стал непосредственным этапом разработки сайта, за которым последовала и верстка. Нарисованные в Photoshop макеты следовало превратить из одной картинки в набор изображений, стилевых и HTML-файлов, способных быстро загружаться по сети, сохраняя при этом особенности дизайна.

Век Netscape продолжался до тех пор, пока часть рынка браузеров не захватила Microsoft со своим браузером Internet Explorer, что в итоге сильно затруднило разработку сайтов. Разные подходы Netscape и Internet Explorer к отображению сайта, противоречивая поддержка стилей и тегов, большое количество мелких ошибок привело к тому, что сайт приходилось тестировать и отлаживать долгое время.

Устаревшие версии браузеров не в полной мере поддерживали CSS (Cascading Style Sheets, каскадные таблицы стилей), поэтому в коде сайта приходилось постоянно использовать смесь из HTML и CSS. Это привело к тому, что для создания макетов стали применяться таблицы с невидимой границей (табличная верстка), на долгое время ставшими стандартом при разработке web-интерфейсов.

Следующим этапом развития разработки сайтов стало рождение спецификации, которая была названа Cascading Style Sheets Level 2, сокращенно CSS2, а затем стремительно развивающийся новый стандарт каскадных таблиц стилей CSS3, которые и используются по сей день.

## История развития стандартов

Развитие HTML в постепенно зашло в тупик − новые теги были уже не нужны, поскольку хватало существующих, к тому же акцент разработки web-страниц сместился на стили, которые расширяют возможности по оформлению документов.

Определенным ограничением HTML являлось и то, что он относится к формальным языкам, в том смысле, что теги и их иерархическая структура жестко описаны в спецификации. В то время популярность набирает XML (EXtensible Markup Language, расширяемый язык разметки), с помощью которого можно создавать собственные теги и формировать их структуру. Разница между HTML и XML состоит не только в тегах, но и в правилах написания кода. Браузер при работе с HTML «закрывает глаза» на разные мелкие огрехи и недочеты в структуре или на не корректное указание атрибутов. С XML браузер выдаст ошибку о том, что документ неверно сформирован.

XML исопользовался в основном для передачи данных и его строгость позволяла избежать ошибок. Однако все большее количество данных размещалось непосредственно на web-страницах, в связи с чем потребовалось создать язык с более строгими правилами оформления HTML-кода. Так появился XHTML, как некий промежуточный стандарт между XML и HTML.

XHTML (EXtensible HyperText Markup Language, расширяемый язык разметки гипертекста) предназначен для замены HTML и считается его более строгой версией. World Wide Web Consortium (W3C, международная некоммерческая организация, занимающаяся развитием технологий Всемирной паутины) определяет XHTML и крайнюю версию HTML 5 как наиболее перспективные направления развития разметки гипертекста.

Если рассуждать о некотором идеальном коде web-страницы, то его можно сравнить с программой, которая не будет скомпилирована до тех пор, пока все ошибки не исправлены. Браузер выступает в роли компилятора и не отображает документ, если он не соответствует спецификации. XHTML, сохраняя все особенности HTML, вносит более строгие правила создания страниц, чтобы приблизиться к «идеальному» коду.

На деле все обстоит более приземленно. Разработчики браузеров не могут позволить себе, чтобы их детище работало только с «правильным» кодом. А все из-за того, что большая часть сайтов в мире просто не будет отображаться в таком браузере. Виноваты в этом и разработчики и пользователи. Первые не обеспечили должную поддержку спецификации в своих браузерах, а вторые не потрудились ей следовать.

## Теги

**Cинтаксис**

Теги применяются для форматирования элементов страницы. Общий синтаксис написания тегов следующий:

<тег атрибут1="значение" атрибут2="значение">...</тег>

Теги бывают двух типов: одиночные и парные (контейнеры). Одиночный тег используется самостоятельно, а парный может включать внутри себя другие теги или текст. У тегов допустимы различные атрибуты, которые разделяются между собой пробелом; есть теги без всяких дополнительных атрибутов.

У большинства непустых контейнеров в их области отображается содержание – данные, заключенные между тегами.

В примере 1.1 показан типичный HTML-документ с тегами и текстом.

**Пример 1.1.** Код HTML-документа.

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Lorem ipsum</title>

</head>

<body>

<p>Lorem ipsum dolor sit amet consectetuer cursus pede pellentesque vitae pretium. Tristique mus at elit lobortis libero Sed vestibulum ut eleifend habitasse.</p>

</body>

</html>

В данном примере используется одиночный тег <meta>, а парных тегов сразу несколько: <html>, <head>, <title>, <body> и <p>.

**Атрибуты**

Чтобы расширить возможности отдельных тегов и более гибко управлять содержимым контейнеров, применяются атрибуты тегов. Когда для тега не добавлен какой-либо допустимый атрибут, это означает, что браузер в этом случае будет подставлять значение, заданное по умолчанию

В стандартах HTML (но не в XHTML) допустимо использовать некоторые атрибуты у тегов, не присваивая им никакого значения, как показано в примере 1.2.

**Пример 1.2.** Использование атрибутов без значения

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

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Добавление формы</title>

</head>

<body>

<form action="self.php">

<p><input type="text"></p>

<p><input type="submit" disabled></p>

</form>

</body>

</html>

В данном примере используется атрибут disabled, у которого не задано значение. Подобная запись называется «сокращенный атрибут тега».

Порядок атрибутов в любом теге не имеет значения и на результат отображения элемента не влияет. Поэтому теги вида <img src="123.gif" width="250" height="200"> и <img height="200" width="250" src="123.gif"> по своему действию равны.

Каждый атрибут тега относится к определенному типу (например: текст, число, путь к файлу и др.), который обязательно должен учитываться при написании атрибута. Так тег <img> добавляет на web-страницу рисунок, а его атрибут width задает ширину изображения в пикселах. Если поставить не число, а нечто другое, то значение будет проигнорировано и возникнет ошибка при валидации документа.

## Стандарт XHTML

**Основные требования**

При написании кода XHTML в основном придерживаются того же синтаксиса, который характерен для HTML, но при этом необходимо придерживаться следующих правил:

1. Все XHTML-документы обязательно должны содержать элемент <!DOCTYPE> в первой строке кода.
2. Теги должны быть набраны в нижнем регистре. Это правило возникло из-за того, что XHTML чувствителен к регистру символов, поэтому для него теги <HR> и <hr> различаются. Чтобы не возникало путаницы, синтаксис принудительно заставляет указывать все теги, а также их атрибуты в нижнем регистре.
3. Значения любых атрибутов необходимо заключать в кавычки. Хотя в HTML также требуется заключать значения в кавычки, но их отсутствие никак не влияет на корректность кода. Так что можно сказать, что в HTML применение кавычек − это лишь рекомендация. В XHTML же использование кавычек возведено в правило и любые значения атрибутов требуется указывать только в них.
4. Требуется закрывать все теги. В HTML теги делятся на две категории − парные теги, называемые еще контейнерами и одинарные теги. Парные теги состоят из открывающего и закрывающего тега, причем в некоторых случаях закрывающий тег можно опустить. В XHTML закрывающий тег требуется всегда и везде. Некоторые разработчики игнорируют закрывающие теги </li>, </p>, </td>, но в XHTML их отсутствие считается ошибкой. Элемент <!DOCTYPE> не является частью XHTML-документа, поэтому для него закрывающего тега не требуется. Что касается одинарных тегов, то они должны завершаться слэшем перед закрывающей угловой скобкой, например: <br />.
5. Должна соблюдаться правильная вложенность тегов. XHTML критично относится к ошибкам следующего типа: некорректная вложенность одного тега в другой и расположение тега в несоответствующем контейнере. Каждый тег должен располагаться внутри другого тега, при этом недопустимо их «пересечение». Все теги имеют строгую иерархическую систему в том смысле, что каждый тег должен находиться внутри другого тега и никак иначе. На условной вершине находится корневой элемент <html>, а все остальные теги могут содержать внутри себя другие теги, которые называются дочерними. Соответственно дочерние теги располагаются в родительском элементе. Необходимо знать систему подчиненности тегов и следовать ей при написании кода XHTML.

В примере 1.3 показана базовая структура документа.

**Пример 1.3.** Базовая структура XHTML-документа.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Новый документ</title>

</head>

<body>

<p>...</p>

</body>

</html>

В данном примере вначале приводится тег <html>, внутри которого располагаются теги <head> и <body>. Внутри раздела <head> хранится заголовок документа (<title>) и кодировка страницы (<meta>).

1. Нельзя использовать сокращенные атрибуты тегов.
2. Вместо атрибута name следует указывать id. Атрибут name определяется в HTML для тегов <a>, <frame>, <iframe>, <img> и <map> и предназначен для обозначения элемента с целью последующего к нему обращения из скриптов. В XHTML атрибут name частично вышел из употребления, а вместо него следует использовать id, как показано в примере 1.4.

**Пример 1.4.** Использование идентификатора id.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>XHTML</title>

</head>

<body>

<p><img src="images/23.png" id="image1" alt="Изображение" /></p>

</body>

</html>

Указанное правило не применяется к элементам форм, вроде <input type="radio" />, где без атрибута name вообще не обойтись. В таблице 1.1 приведены некоторые теги HTML и их запись в XHTML-документе.

**Таблица 1.1.** Примеры тегов HTML и XHTML.

|  |  |
| --- | --- |
| HTML-тег | XHTML-тег |
| <br> | <br /> |
| <hr> | <hr /> |
| <input> | <input /> |
| <img> | <img /> |
| <meta> | <meta /> |

**Разновидности стандарта**

Любой XHTML-файл состоит из четырех разделов − <!DOCTYPE>, контейнера <html>, заголовка (<head>) и тела документа (<body>).

Элемент <!DOCTYPE> сообщает браузеру о типе текущего документа и как его интерпретировать. Различают три типа документа по спецификации XHTML 1.0:

**XHTML 1.0 Strict:** используется в том случае, если в документе идет четкое разделение оформления и содержания. При этом код web-страницы содержит только теги разметки, а сам вид элементов задается через стили.

**XHTML 1.0 Transitional:** обычно применяется, когда правило разделения оформления и содержания выполняется не в полной мере. В этом случае допускается в коде документа использовать теги физического проектирования (например, тег <tt>) и лишь частично стили.

**XHTML 1.0 Frameset:** применяется, когда окно браузера делится на два или более фрейма.

**Структура xhtml документа**

Тег <html> выступает главным контейнером, в котором хранятся все остальные элементы, и в коде он должен идти сразу же после <!DOCTYPE>. В <html> требуется указывать обязательный атрибут xmlns со значением http://www.w3.org/1999/xhtml. Он определяет пространство имен − так называется набор имен всех элементов и атрибутов, чтобы обеспечить их уникальность в пределах XHTML-документа.

**Кодировка документа**

В HTML кодировка документа задается с помощью тега <meta> расположенного внутри контейнера <head>. В XHTML же кодировка может задаваться двумя путями: аналогично HTML и с помощью команды <?xml version="1.0" encoding="UTF-8"?>, она должна располагаться в первой строке кода, перед <!DOCTYPE>. В данном случае UTF-8 обозначает кодировку текущего документа. В примере 1.5 показано, как устанавливать кодировку подобным способом.

**Пример 1.5.** Определение кодировки документа в windows-1251.

<?xml version="1.0" encoding="windows-1251"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Кодировка</title>

</head>

<body>

<p>Текст на русском языке в стандартной кодировке Windows</p>

</body>

</html>

## Теги заголовка

Тег **<head>** предназначен для хранения служебной информации, предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных.

Содержимое тега <head> не отображается напрямую на web-странице, за исключением тега <title>, устанавливающего заголовок окна web-страницы.

Внутри контейнера <head> допускается размещать следующие элементы: <title>, <meta>, <link>, <script>, <style> и другие. Рассмотрим некоторые из них:

**<title>** −используется для отображения строки текста в левом верхнем углу окна браузера, а также на вкладке. Такая строка сообщает пользователю название сайта и другую информацию, которую добавляет разработчик.

**<meta>** −метатеги используются для хранения информации, предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Хотя тег <meta> всего один, он имеет несколько атрибутов, поэтому к нему и применяется множественное число.

**<link>** − устанавливает связь с внешним документом вроде файла со стилями или со шрифтами. В отличие от тега <a>, тег <link> размещается всегда внутри контейнера <head> и не создает ссылку.

**<style>** − тег <style> применяется для определения стилей элементов web-страницы. Тег <style> необходимо использовать внутри контейнера <head>. Можно задавать более чем один тег <style>.

## Строчные и блочные элементы

Области элементов делятся на две основные категории: строчные и блочные области. Строчная область (**inline box**) размещается на одной строке с соседними строчными областями. Элементы **а** и **img** (a также **em**, **strong** и многие другие) no умолчанию порождают строчные области. Строчная область имеет форму прямоугольника, который целиком размещается внутри одной строки.

Блочная область (**block box**) всегда имеет форму одного прямоугольника, перед которым и после которого переводится строка. Таким образом, блочная область, в отличие от строчной, по умолчанию не отображается рядом с другими областями. Расположить рядом с блочной областью другую блочную или строчную область можно с помошью специальных средств CSS − например, плавающих элементов или абсолютного позиционирования.

Разница между блочными и строчными элементами следующая: строчные элементы могут содержать только данные или другие строчные элементы, а в блочные допустимо вкладывать другие блочные элементы, строчные элементы, а также данные. Иными словами, строчные элементы никак не могут хранить блочные элементы.

Блочные элементы занимают всю доступную ширину, например, окна браузера, а ширина строчных элементов равна их содержимому плюс значения отступов, полей и границ.

### Блочные элементы

Блочные элементы характеризуются тем, что занимают всю доступную ширину, высота элемента определяется его содержимым, и он всегда начинается с новой строки.

**<blockquote>** − предназначен для выделения длинных цитат внутри документа. Текст, обозначенный этим тегом, традиционно отображается как выровненный блок с отступами слева и справа (примерно по 40 пикселов), а также с пустым пространством сверху и снизу.

**<div>** − относится к универсальным блочным контейнерам и применяется в тех случаях, где нужны блочные элементы без дополнительных свойств. Также с помощью тега <div> можно выравнивать текст внутри этого контейнера с помощью атрибута align.

**<h1>..,<h6>** − эта группа тегов определяет текстовые заголовки разного уровня, которые показывают относительную важность секции, расположенной после заголовка.

**<hr>** − рисует горизонтальную линию, которая по своему виду зависит от используемых атрибутов. Линия всегда начинается с новой строки, а после нее все элементы отображаются на следующей строке.

**<p>** − определяет параграф (абзац) текста.

**<pre>** − задает блок предварительно форматированного текста. Такой текст отображается обычно моноширинным шрифтом и со всеми пробелами между словами. В HTML любое количество пробелов, идущих в коде подряд на web-странице показывается как один. Тег <pre> позволяет обойти эту особенность и отображать текст как требуется разработчику. Следующие теги не должны размещаться внутри контейнера <pre>: <big>, <img>, <small>, <sub> и <sup>.

### Строчные элементы

Строчными называются такие элементы web-страницы, которые являются непосредственной частью другого элемента, например, текстового абзаца. В основном они используются для изменения вида текста или его логического выделения.

**<a>** − является одним из важных элементов HTML и предназначен для создания ссылок. В зависимости от присутствия атрибутов name или href тег <a> устанавливает ссылку или якорь.

**<b>** − определяет жирное начертание шрифта.

**<big>** − увеличивает размер шрифта на единицу по сравнению с обычным текстом. В HTML размер шрифта измеряется в условных единицах от 1 до 7, средний размер текста, используемый по умолчанию, принят 3. Таким образом, добавление тега <big> увеличивает текст на одну условную единицу.

**<br>** − устанавливает перевод строки в том месте, где этот тег находится. В отличие от тега параграфа <p>, использование тега <br> не добавляет пустой отступ перед строкой.

**<em>** − предназначен для акцентирования текста. Браузеры отображают такой текст курсивным начертанием.

**<i>** − устанавливает курсивное начертание шрифта.

**<img>** − предназначен для отображения на web-странице изображений в графическом формате GIF, JPEG или PNG. Если необходимо, то рисунок можно сделать ссылкой на другой файл, поместив тег <img> в контейнер <a>. При этом вокруг изображения отображается рамка, которую можно убрать, добавив атрибут border="0" в тег <img>.

**<small>** − уменьшает размер шрифта на единицу по сравнению с обычным текстом. По своему действию похож на тег <big>, но действует с точностью до наоборот.

**<span>** − универсальный тег, предназначенный для определения строчного элемента внутри документа.

**<strong>** − предназначен для акцентирования текста. Браузеры отображают такой текст жирным начертанием.

**<sub>** − отображает шрифт в виде нижнего индекса. Текст при этом располагается ниже базовой линии остальных символов строки и уменьшенного размера − H2O.

**<sup>** − отображает шрифт в виде верхнего индекса. По своему действию похож на <sub>, но текст отображается выше базовой линии текста − м2.

## Гиперссылки

Ссылки являются основой гипертекстовых документов и позволяют переходить с одной web-страницы на другую. Особенность их состоит в том, что сама ссылка может вести не только на HTML-файлы, но и на файл любого типа, причем этот файл может размещаться совсем на другом сайте. Главное, чтобы к документу, на который делается ссылка, был доступ. Иными словами, если путь к файлу можно указать в адресной строке браузера, и файл при этом будет открыт, то на него можно сделать ссылку. Для создания ссылки необходимо сообщить браузеру, что является ссылкой, а также указать адрес документа, на который следует сделать ссылку. Оба действия выполняются с помощью тега <a>. Общий синтаксис создания ссылок следующий:

<a href="URL">текст ссылки</a>

Атрибут href определяет URL (Universal Resource Locator, универсальный указатель ресурса), иными словами, адрес документа, на который следует перейти, а содержимое контейнера <a> является ссылкой. Текст, расположенный между тегами <a> и </a>, по умолчанию становится синего цвета и подчеркивается. В примере 1.6 показано создание нескольких ссылок на разные web-страницы.

**Пример 1.6.** Создание нескольких ссылок на разные web-страницы.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Ссылки на странице</title>

</head>

<body>

<p><a href="dog.html">Собаки</a></p>

<p><a href="cat.html">Кошки</a></p>

</body>

</html>

В данном примере создаются две ссылки с разными текстами. При щелчке по тексту «Собаки» в окне браузера откроется документ dog.html, а при щелчке на «Кошки» − файл cat.html. Результат примера показан на рис. 1.1. Обратите внимание, что при наведении курсора мыши на ссылку, в строке состояния браузера отображается полный путь к ссылаемому файлу.

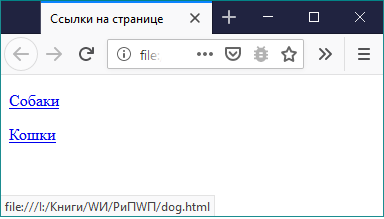


Рис. 1.1. Ссылки на странице.

Если указана ссылка на файл, которого не существует, например, его имя в атрибуте href набрано с ошибкой, то такая ссылка называется битая. Битых ссылок следует категорически избегать, поскольку они вводят посетителей сайта в заблуждение. Чтобы не возникало подобных ошибок, тестируйте все ссылки на их работоспособность.

Файл по ссылке открывается в окне браузера только в тех случаях, когда браузер знает тип документа. Но поскольку ссылку можно сделать на файл любого типа, то браузер не всегда может отобразить документ. При этом выводится сообщение, как следует обработать файл: открыть его или сохранить в указанную папку.

**Атрибуты гиперссылок:**

**accesskey** − активация ссылки с помощью комбинации клавиш.

**coords** − устанавливает координаты активной области.

**download** − предлагает скачать указанный по ссылке файл.

**href** − задает адрес документа, на который следует перейти.

**hreflang** − идентифицирует язык текста по ссылке.

**name** − устанавливает имя якоря внутри документа.

**rel** − отношения между ссылаемым и текущим документами.

**rev** − отношения между текущим и ссылаемым документами.

**shape** − задает форму активной области ссылки для изображений.

**tabindex** − определяет последовательность перехода между ссылками при нажатии на кнопку Tab.

**target** − имя окна или фрейма, куда браузер будет загружать документ.

**title** − добавляет всплывающую подсказку к тексту ссылки.

**type** − указывает MIME-тип документа, на который ведёт ссылка. Также для этого тега доступны универсальные атрибуты и события. Закрывающий тег − обязателен

В зависимости от присутствия атрибутов name или href тег <a> устанавливает ссылку или якорь. **Якорем** называется закладка с уникальным именем на определенном месте web-страницы, предназначенная для создания перехода к ней по ссылке. Якоря удобно применять в документах большого объема, чтобы можно было быстро переходить к нужному разделу.

Для создания якоря следует вначале сделать закладку в соответствующем месте и дать ей имя при помощи атрибута **name** тега <a> (пример 1.7). В качестве значения href для перехода к этому якорю используется имя закладки с символом решетки (#) впереди.

**Пример 1.7.** Создание якоря.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Быстрый переход внутри документа</title>

</head>

<body>

<p><a name="top"></a></p>

<p>...</p>

<p><a href="#top">Наверх</a></p>

</body>

</html>

Между тегами <a name="top"> и </a> текст не обязателен, так как требуется лишь указать местоположение перехода по ссылке, находящейся внизу страницы. Имя ссылки на якорь начинается с символа #, после чего идет имя якоря, оно выбирается любое, соответствующее тематике. Главное, чтобы значения атрибутов name и href совпадали.

С якорями связана одна особенность работы браузера. После перехода к указанному якорю нажатие на кнопку «Назад» возвращает не на предыдущую просмотренную страницу, а к ссылке, с которой был сделан переход к якорю. Получается, что для перехода к предыдущему документу надо нажать кнопку «Назад» два раза.

Cсылку можно также сделать на закладку, находящуюся в другой web-странице и даже другом сайте. Для этого в атрибуте href тега <a> надо указать адрес документа и в конце добавить символ решетки # и имя закладки (пример 1.8).

**Пример 1.8.** Ссылка на закладку из другой web-страницы.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Якорь в другом документе</title>

</head>

<body>

<p><a href="text.html#bottom">Перейти к нижней части текста</a></p>

</body>

</html>

В данном примере показано создание ссылки на файл text.html, при открытии этого файла происходит переход на закладку с именем bottom.

## Изображения

Для добавления изображения на web-страницу используется тег <img>, атрибут src которого определяет адрес графического файла. Общий синтаксис добавления изображения будет следующий:

<img src="URL" alt="альтернативный текст">

URL представляет собой путь к графическому файлу. Для его указания можно использовать как абсолютный, так и относительный адрес. Далее рассмотрим несколько разных путей к графическому файлу для размещения его на web-странице. Для примера возьмем файл с рисунком, который называется image.png и хранится в папке images корня сайта.

Если в начале адреса стоит слэш (символ /), это значит, что отсчет идет от корня сайта. Например, адрес сайта − http://site.ru, значит, написав путь к изображению как /images/image.png, мы, тем самым говорим серверу, что показать следует файл http://site.ru/images/image.png. Учтите, что подобные ссылки со слэшем впереди работают только на web-сервере, на локальном компьютере они действовать не будут.

Если перед адресом добавляется упоминание протокола http (http://), то речь идет об абсолютной ссылке. Изображение всегда будет загружаться с указанного адреса в Интернете, даже при сохранении web-страницы на локальный компьютер.

Двоеточие со слэшем (../) в начале адреса говорит о том, что и рисунок, и web-страница находятся в разных папках, которые размещены на одном уровне. На рис. 1.2 показан файл index.html, в который требуется поместить рисунок image.png. Тогда относительный путь к изображению из index.html будет ../images/ image.png.

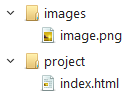


Рис. 1.2. index.html размещен ниже папки images.

Возможно такое размещение файлов, что потребуется подняться на несколько уровней выше для доступа к изображению, например: ../../../images/image.png.

Имя папки в начале пути, без всяких слэшей и двоеточий, означает, что и текущий файл, и папка с изображением находятся на одном уровне расположения файлов. Показанный на рис. 1.3 относительный путь к изображнию  
image.png из файла index.html будет images/image.png.

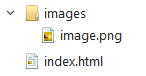


Рис. 1.3. index.html размещен рядом с папкой images.

В примере 1.9 показано несколько способов добавления рисунка на web-страницу.

**Пример 1.9.** Вставка изображения в документ.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Добавление рисунков</title>

</head>

<body>

<p><img src="http://site.ru/images/image.png" alt="Это абсолютный адрес размещения изображения"></p>

<p><img src="/images/image.png" alt="Адрес размещения изображения относительно корня сайта"></p>

<p><img src="images/image.png" alt="Адрес размещения изображения относительно текущего HTML-документа"></p>

</body>

</html>

Как правило, в качестве форматов графических файлов в гипертексте используются форматы png, gif и jpeg.

## Списки

Списки предоставляют возможность упорядочить и систематизировать разные данные и представить их в наглядном и удобном для пользователя виде. Виды списков в HTML:

**<ol>** − устанавливает нумерованный список, где каждый элемент списка начинается с числа или буквы и увеличивается по нарастающей, тег <li> определяет отдельный элемент списка.

**<ul>** − устанавливает маркированный список, каждый элемент которого начинается с небольшого символа − маркера, тег <li> определяет отдельный элемент списка. Внешний тег <ul> или <ol> устанавливает тип списка − маркированный или нумерованный.

**<dd>**, **<dt>**, **<dl>** − тройка элементов предназначена для создания списка определений. Каждый такой список начинается с контейнера <dl>, куда входит тег <dt>, создающий термин и тег <dd>, задающий определение этого термина.

## Таблицы

Благодаря универсальности таблиц и большому числу параметров, управляющих их видом, таблицы надолго стали стандартом для верстки web-страниц. Таблица с невидимой границей представляет собой словно модульную сетку, в блоках которой удобно размещать элементы web-страницы. Так было долгое время, пока на смену таблицам при верстке сайтов не пришли слои. Более разумно, когда таблицы применяются для размещения табличных данных, а слои − для верстки и оформления.

Таблица состоит из строк и столбцов ячеек, которые могут содержать текст и рисунки. Обычно таблицы используются для упорядочения и представления данных, однако возможности таблиц этим не ограничиваются. C помощью таблиц удобно верстать макеты страниц, расположив нужным образом фрагменты текста и изображений.

**<table>** − служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из строк и ячеек, которые задаются с помощью тегов <tr> и <td>.

**<tr>** − служит контейнером для создания строки таблицы.

**<td>** − предназначен для создания одной ячейки таблицы. Тег <td> должен размещаться внутри контейнера <tr>, который в свою очередь располагается внутри тега <table>.

**<th>** − предназначен для создания одной ячейки таблицы, которая обозначается как заголовочная. Текст в такой ячейке отображается браузером обычно жирным шрифтом и выравнивается по центру.

**Пример 1.10.** Создание таблицы.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Тег TABLE</title>

</head>

<body>

<table border="1" width="100%" cellpadding="5">

<tr>

<th>Ячейка 1</th>

<th>Ячейка 2</th>

</tr>

<tr>

<td>Ячейка 3</td>

<td>Ячейка 4</td>

</tr>

</table>

</body>

</html>

Порядок расположения ячеек и их вид показан на рис. 1.4:

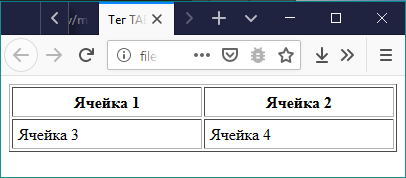


Рис. 1.4. Таблица с четырьмя ячейками.

Для объединения двух и более ячеек в одну используются атрибуты **colspan** и **rowspan** тега <td>. Атрибут colspan устанавливает число ячеек, объединяемых по горизонтали. Аналогично работает и атрибут rowspan, с тем лишь отличием, что объединяет ячейки по вертикали. Перед добавлением атрибутов проверьте число ячеек в каждой строке, чтобы не возникло ошибок. Так, <td colspan="3"> заменяет три ячейки, поэтому в следующей строке должно быть три тега <td> или конструкция вида <td colspan="2">...</td><td>...</td>. Если число ячеек в каждой строке не будет совпадать, появятся пустые фантомные ячейки. В примере 1.11 приведен хотя и валидный, но неверный код, в котором как раз проявляется подобная ошибка.

**Пример 1.11.** Неверное объединение ячеек.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Неправильное использование colspan</title>

</head>

<body>

<table border="1" cellpadding="5" width="100%">

<tr>

<td colspan="2">Ячейка 1</td>

<td>Ячейка 2</td>

</tr>

<tr>

<td>Ячейка 3</td>

<td>Ячейка 4</td>

</tr>

</table>

</body>

</html>

Результат данного примера показан на рисунке 1.5:

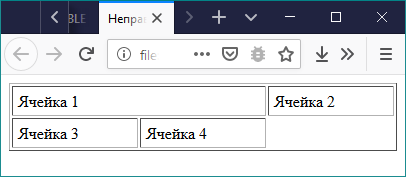


Рис. 1.5. Появление дополнительной ячейки в таблице.

В первой строке примера задано три ячейки, две из них объединены с помощью атрибута colspan, а во второй строке добавлено только две ячейки. Из-за этого возникает дополнительная ячейка, которая отображается в браузере. Ее хорошо видно на рис. 1.5.

Правильное использование атрибутов colspan и rowspan продемонстрировано в примере 1.12.

**Пример 1.12.** Объединение ячеек по вертикали и горизонтали.

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

<html>

<head>

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

<title>Объединение ячеек</title>

</head>

<body>

<table border="1" cellpadding="4" cellspacing="0">

<tr>

<td rowspan="2">Браузер</td>

<th colspan="2">Internet Explorer</th>

<th colspan="3">Opera</th>

<th colspan="2">Firefox</th>

</tr>

<tr>

<th>6.0</th><th>7.0</th><th>7.0</th><th>8.0</th><th>9.0</th><th>1.0</th><th>2.0</th>

</tr>

<tr align="center">

<td>Поддерживается</td>

<td>Нет</td><td>Да</td><td>Нет</td><td>Да</td><td>Да</td><td>Да</td><td>Да</td>

</tr>

</table>

</body>

</html>

Результат данного примера показан на рис. 1.6.

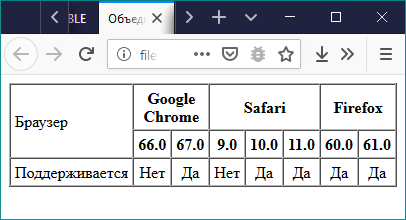


Рис. 1.6. Таблица с объединенными ячейками.

В данной таблице установлено восемь колонок и три строки. Часть ячеек с надписями «Google Chrome», «Safari» и «Firefox» объединены где по две, а где и по три ячейки. В ячейке с надписью «Браузер» применено объединение по вертикали.

## Формы

Сама форма обычно предназначена для получения от пользователя информации для дальнейшей пересылки её на сервер, где данные формы принимает программа-обработчик. Такая программа может быть написана на любом серверном языке программирования вроде PHP, Perl и др.

Тег **<form>** устанавливает форму на web-странице. Форма предназначена для обмена данными между пользователем и сервером. Область применения форм не ограничена отправкой данных на сервер, с помощью клиентских скриптов можно получить доступ к любому элементу формы, изменять его и применять по своему усмотрению.

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

Для отправки формы на сервер используется кнопка **submit**, того же можно добиться, если нажать клавишу Enter в пределах формы. Если кнопка Submit отсутствует в форме, клавиша Enter имитирует ее использование.

Когда форма отправляется на сервер, управление данными передается программе, заданной атрибутом action тега <form>. Предварительно браузер подготавливает информацию в виде пары «имя=значение», где имя определяется атрибутом name тега <input>, а значение введено пользователем или установлено в поле формы по умолчанию. Если для отправки данных используется метод GET, то адресная строка может принимать следующий вид:

http://www.html-training.ru/cgi-bin/madein.cgi?nick=%C2%E0 %ED%FF+%D8%E0%EF%EE%F7%EA%E8%ED&page=5

Параметры перечисляются после вопросительного знака, указанного после адреса CGI-программы и разделяются между собой символом амперсанда (&). Нелатинские символы преобразуются в шестнадцатеричное представление (в форме %HH, где HH **−** шестнадцатеричный код для значения ASCII-символа), пробел заменяется на плюс (+).

Допускается внутрь контейнера <form> помещать другие теги, при этом сама форма никак не отображается на web-странице, видны только ее элементы и результаты вложенных тегов.

Тег **<input>** является одним из разносторонних элементов формы и позволяет создавать разные элементы интерфейса и обеспечить взаимодействие с пользователем. Главным образом <input> предназначен для создания текстовых полей, различных кнопок, переключателей и флажков. Хотя элемент <input> не требуется помещать внутрь контейнера <form>, определяющего форму, но если введенные пользователем данные должны быть отправлены на сервер, где их обрабатывает серверная программа, то указывать <form> обязательно.

Основной атрибут тега <input>, определяющий вид элемента – **type**. Он позволяет задавать следующие элементы формы: текстовое поле (**text**), поле с паролем (**password**), переключатель (**radio**), флажок (**checkbox**), скрытое поле (**hidden**), кнопка (**button**), кнопка для отправки формы (**submit**), кнопка для очистки формы (reset), поле для отправки файла (file) и кнопка с изображением (**image**). Для каждого элемента существует свой список атрибутов, которые определяют его вид и характеристики.

Тег **<button>** создает на web-странице кнопки и по своему действию напоминает результат, получаемый с помощью тега <input> (с атрибутом type="button | reset | submit"). В отличие от этого тега, <button> предлагает расширенные возможности по созданию кнопок. Например, на подобной кнопке можно размещать любые элементы HTML, в том числе изображения. Используя стили, можно определить вид кнопки путем изменения шрифта, цвета фона, размеров и других параметров.

Тег **<select>** позволяет создать элемент интерфейса в виде раскрывающегося списка, а также список с одним или множественным выбором. Конечный вид зависит от использования атрибута size тега <select>, который устанавливает высоту списка. Ширина списка определяется самым широким текстом, указанным в теге **<option>**, а также может изменяться с помощью стилей. Каждый пункт создается с помощью тега <option>, который должен быть вложен в контейнер <select>. Если планируется отправлять данные списка на сервер, то требуется поместить элемент <select> внутрь формы. Это также необходимо, когда к данным списка идет обращение через скрипты.

Элемент **<fieldset>** предназначен для группировки элементов формы. Группировка облегчает работу с формами, содержащими большое число данных. Например, один блок может быть предназначен для ввода текстовой информации, а другой для флажков.

Браузеры для повышения наглядности отображают результат использования тега <fieldset> в виде рамки. Ее вид зависит от операционной системы и используемого браузера. Синтаксис:

<form>

<fieldset>...</fieldset>

</form>

**Атрибуты fieldset**

[**disabled**](http://htmlbook.ru/html/fieldset/disabled) − блокирует поля формы в группе.

[**form**](http://htmlbook.ru/html/fieldset/form) − связывает группу с формой.

[**title**](http://htmlbook.ru/html/fieldset/title) − добавляет всплывающую подсказку к группе формы.

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

## Вопросы для самоконтроля и задачи

1. Системная информация, которая не отображается в окне браузера, находится в теге:

<head>;

<body>;

<form>.

1. Теги, содержащие наполнение страницы, размещаются в:

<head>;

<body>;

<form>.

1. В каких стандартах XHTML требуется обязательно определять вид документа через стили?

XHTML 1.0 Transitional;

XHTML 1.0 Strict;

XHTML 1.0 Frameset.

1. Какие теги форматирования текста вы знаете?
2. Какие атрибуты позволяют объединять ячейки по горизонтали и по вертикали?
3. Назовите основные требования к коду стандарта XHTML.
4. Соответствует ли данный фрагмент кода стандарту XHTML?

[Code]

<form action="action.php" method="post">

<input type="text" value="совет дня" />

<input type="checkbox" checked="checked" />

<input type="submit" value="узнать совет" />

</form>

[/code]

1. Соответствует ли данный фрагмент кода стандарту XHTML?

[code]

<form action="action.php" method="post">

<input type="text" value="Совет дня" />

<input type="checkbox" checked readonly />

<input type="submit" value="Узнать совет" />

</form>

[/code]

1. Выберите блочные элементы:

<img>;

<a>;

<b>;

<em>;

<div>;

<ol>;

<p>;

<table>.

1. Выберите строчные элементы:

<img>;

<a>;

<i>;

<em>;

<div>;

<ul>;

<p>;

<pre>.

Глава 2. Каскадные таблицы стилей



## Введение в CSS

Применение стилей позволяет задавать точные характеристики практически всех элементов web-страницы, а это значит, что можно точно контролировать внешний вид web-страницы в окне web-браузера.

**Преимущества от использования стилей:**

1. Разграничение **кода и оформления.** Идея о том, чтобы код HTML был свободен от элементов оформления вроде установки цвета, размера шрифта и других параметров, стара как мир. В идеале, web-страница должна содержать только теги логического форматирования, а вид элементов задается через стили. При подобном разделении работа над дизайном и версткой сайта может вестись параллельно.
2. Разное **оформление для разных устройств.** С помощью стилей можно определить вид web-страницы для разных устройств вывода: монитора, принтера, смартфона, планшета и тд.
3. **Расширенные по сравнению с HTML способы оформления элементов.** В отличии от HTML стили имеют гораздо больше возможностей по оформлению элементов web-страниц. Простыми средствами можно изменить цвет фона элемента, добавить рамку, установить шрифт, определить размеры, положение и многое другое.
4. **Ускорение загрузки сайта.** При хранении стилей в отдельном файле, он кэшируется и при повторном обращении к нему извлекается из кэша браузера. За счёт кэширования и того, что стили хранятся в отдельном файле, уменьшается код web-страниц и снижается время загрузки документов.
5. **Централизованное хранение.** Стили, как правило, хранятся в одном или нескольких специальных файлах, ссылка на которые указывается во всех документах сайта. Благодаря этому удобно править стиль в одном месте, при этом оформление элементов автоматически меняется на всех страницах, которые связаны с указанным файлом. Вместо того чтобы модифицировать десятки HTML файлов, достаточно отредактировать один файл со стилем и оформление нужных документов сразу поменяется.
6. **Единое стилевое оформление множества документов.** Применение единообразного оформление заголовков, основного текста и других элементов создает преемственность между страницами и облегчает пользователям работу с сайтом и его восприятие в целом. Разработчикам упрощается проектирование сайта.

## Способы добавления стиля на страницу

**Связанные стили**

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле с расширением css, а для связывания документа с этим файлом применяется тег <link>. Данный тег помещается в контейнер <head>, как показано в примере 2.1.

**Пример 2.1.** Подключение связанных стилей.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Стили</title>  
 <link rel="stylesheet" href="http://site.ru/mysite.css">  
 <link rel="stylesheet" href="http://site.ru/main.css">  
 </head>  
 <body>  
 <h1>Заголовок</h1>  
 <p>Текст</p>  
 </body>  
</html>

Значение атрибута rel остаётся неизменным независимо от кода, как приведено в данном примере. Значение href задаёт путь к CSS-файлу, он может быть задан как относительно, так и абсолютно. Заметьте, что таким образом можно подключать таблицу стилей, которая находится на другом сайте. Содержимое файла mysite.css подключаемого посредством тега <link> приведено в примере 2.2

**Пример 2.2.** Стилевой файл.

h1 {  
 color: #000080;  
 font-size: 200%;  
 font-family: Arial, Verdana, sans-serif;  
 text-align: center;

}

p {

padding-left: 20px;

}

Как видно из данного примера, файл со стилем не хранит ничего, кроме CSS-кода. В свою очередь и HTML-документ содержит только ссылку на файл со стилем, т. е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт. Ведь стили хранятся в одном файле, а в HTML-документах указывается только ссылка на него.

**Глобальные стили**

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке web-страницы. По своей гибкости и возможностям этот способ добавления стиля уступает предыдущему, но также позволяет хранить стили в одном месте, в данном случае прямо на странице с помощью контейнера <style>, как показано в примере 2.3

**Пример 2.3.** Использование глобального стиля

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Глобальные стили</title>  
 <style>  
 H1 {   
 font-size: 120%;   
 font-family: Verdana, Arial, Helvetica, sans-serif;   
 color: #333366;   
 }  
 </style>  
 </head>  
 <body>  
 <h1>Hello, world!</h1>  
 </body>  
</html>

В данном примере задан стиль тега <h1>, который затем используется на всей web-странице.

**Внутренние стили**

Внутренний или встроенный используется для одиночного тега на текущей web-странице. Для определения стиля используется атрибут style, а его значением выступает набор стилевых правил (пример 2.4).

**Пример 2.4.** Использование внутреннего стиля.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Внутренние стили</title>  
 </head>  
 <body>  
 <p style="font-size: 120%; font-family: monospace; color: #cd66cc">Пример текста</p>  
</body>  
</html>

В данном примере стиль тега <p> задаётся с помощью атрибута style, в котором через точку с запятой перечисляются стилевые свойства.

Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. Дело в том, что добавление таких стилей увеличивает общий объём файлов, что ведет к повышению времени их загрузки в браузере, и усложняет редактирование документов для разработчиков.

Все описанные методы использования CSS могут применяться как самостоятельно, так и в сочетании друг с другом. В этом случае необходимо помнить об их иерархии. Первым имеет приоритет внутренний стиль, затем глобальный стиль и в последнюю очередь связанный стиль. В примере 2.5 использованы сразу два метода добавления стиля в документ.

**Пример 2.5.** Сочетание разных методов подключения стилей.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Подключение стиля</title>  
 <style>  
 H1 {   
 font-size: 120%;   
 font-family: Arial, Helvetica, sans-serif;   
 color: green;   
 }  
 </style>  
 </head>  
 <body>  
 <h1 style="font-size: 36px; font-family: Times, serif; color: red">Заголовок 1</h1>  
 <h1>Заголовок 2</h1>  
 </body>  
</html>

В данном примере первый заголовок задаётся красным цветом размером 36 пикселей с помощью внутреннего стиля, а следующий − зелёным цветом через таблицу глобальных стилей (рис. 2.1).

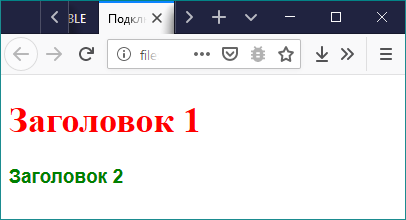


Рис 2.1. Результат применения стилей.

**Импорт CSS**

В текущую стилевую таблицу можно импортировать содержимое CSS-файла с помощью команды @import. Этот метод допускается использовать совместно со связанными или глобальными стилями, однако нельзя совместно с внутренними стилями. Общий синтаксис следующий:

@import url("имя файла") типы носителей;

@import "имя файла" типы носителей;

После ключевого слова @import указывается путь к стилевому файлу одним из двух приведенных способов − с помощью url или без него. В примере 2.6 показано, как можно импортировать стиль из внешнего файла в таблицу глобальных стилей.

**Пример 2.6.** Импорт CSS.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Импорт</title>  
 <style>  
 @import url("style/header.css");  
 H1 {   
 font-size: 120%;   
 font-family: Arial, Helvetica, sans-serif;   
 color: green;   
 }  
 </style>  
 </head>   
 <body>  
 <h1>Заголовок 1</h1>  
 <h2>Заголовок 2</h2>  
 </body>  
</html>

Аналогично происходит импорт и в файле со стилем, который затем подключается к документу (пример 2.7).

**Пример 2.7.** Импорт в таблице связанных стилей

@import "/style/print.css";  
@import "/style/palm.css";   
BODY {  
 font-family: Arial, Verdana, Helvetica, sans-serif;  
 font-size: 90%;  
 background: white;  
 color: black;  
}

В данном примере показано содержимое файла mysite.css, который добавляется к нужным документам способом, показанным в примере 1, а именно с помощью тега <link>.

## Носители

Широкое развитие различных платформ и устройств заставляет разработчиков делать под них специальные версии сайтов, что достаточно трудоёмко и проблематично. Вместе с тем, времена и потребности меняются, и создание сайта для различных устройств является неизбежным и необходимым звеном его развития. С учетом этого в CSS введено понятие типа носителя, когда стиль применяется только для определённого устройства. В таблице 2.1 перечислены некоторые типы носителей.

**Таблица 2.1.** Типы носителей и их описание.

|  |  |
| --- | --- |
| Тип | Описание носителя |
| all | Все типы. Это значение используется по умолчанию. |
| aural | Речевые синтезаторы, а также программы для воспроизведения текста вслух. Сюда, например, можно отнести речевые браузеры. |
| braille | Устройства, основанные на системе Брайля, которые предназначены для слепых людей. |
| handheld | Наладонные компьютеры и аналогичные им аппараты. |
| print | Печатающие устройства вроде принтера. |
| projection | Проектор. |
| screen | Экран монитора. |
| tv | Телевизор. |

В CSS для указания типа носителей применяются команды @import и @media, с помощью которых можно определить стиль для элементов в зависимости от того, выводится документ на экран или на принтер.

Ключевые слова @import и @media относятся к эт-правилам. Такое название произошло от наименования символа @ − «эт», с которого и начинаются эти ключевые слова. В рунете для обозначения символа @ применяется устоявшийся термин «собака».

При импортировании стиля через команду @import тип носителя указывается после адреса файла. При этом допускается задавать сразу несколько типов, упоминая их через запятую, как показано в примере 2.8.

**Пример 2.8.** Импорт стилевого файла.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

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

<title>Импорт стиля</title>  
 <style>  
 @import "/style/main.css" screen; /\* Стиль для вывода результата на монитор \*/  
 @import "/style/print.css" print; /\* Стиль для печати \*/  
 </style>  
 </head>   
 <body>  
 <p>...</p>  
 </body>  
</html>

В данном примере импортируется два файла − main.css предназначен для изменения вида документа при его просмотре на экране монитора, и print.css − при печати страницы и отображении на смартфоне.

Команда @media позволяет указать тип носителя для глобальных или связанных стилей и в общем случае имеет следующий синтаксис:

@media тип носителя 1 {Описание стиля для типа носителя 1}

@media тип носителя 2 {Описание стиля для типа носителя 2}

После ключевого слова @media идёт один или несколько типов носителя, перечисленных в таблице 2.1, если их больше одного, то они разделяются между собой запятой. После чего следуют обязательные фигурные скобки, внутри которых идёт обычное описание стилевых правил. В примере 2.9 показано, как задать разный стиль для печати и отображения на мониторе.

**Пример 2.9.** Стили для разных типов носителей

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Типы носителей</title>

<style>  
 @media screen { /\* Стиль для отображения в браузере \*/  
 BODY {font-family: Verdana, Arial, sans-serif; /\* Рубленый шрифт \*/  
 font-size: 90%; /\* Размер шрифта \*/  
 color: #002310; /\* Цвет текста \*/}  
 H1 {background: #faf0e6; /\* Цвет фона \*/  
 border: 2px dashed maroon; /\* Рамка вокруг заголовка \*/  
 color: #c23689; /\* Цвет текста \*/  
 padding: 7px; /\* Поля вокруг текста \*/}  
 H2 {color: #482433; /\* Цвет текста \*/  
 margin: 0; /\* Убираем отступы \*/}  
 P {margin-top: 0.5em; /\* Отступ сверху \*/}  
 }  
 @media print { /\* Стиль для печати \*/  
 BODY {font-family: Times, 'Times New Roman', serif; /\* Шрифт с засечками \*/}  
 H1, H2, P {color: black; /\* Чёрный цвет текста \*/}  
 }  
 </style>  
 </head>   
 <body>   
 <h1>Воспитание изящно отталкивает объект. </h1>

<h2>Нейро-лингвистическое программирование.</h2>

<p>НЛП позволяет вам точно определить какие изменения в субьективном опыте надо произвести...</p>  
 </body>  
</html>

В данном примере объявлено два стиля − один для изменения вида элементов при их обычном отображении в браузере, а второй − при выводе страницы на печать. При этом облик документа для разных носителей может сильно различаться между собой, например, как это показано на рис. 2.2 и рис. 2.3.

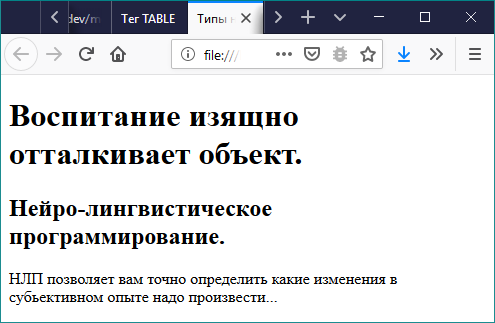


Рис. 2.2. Страница, предназначенная для печати.

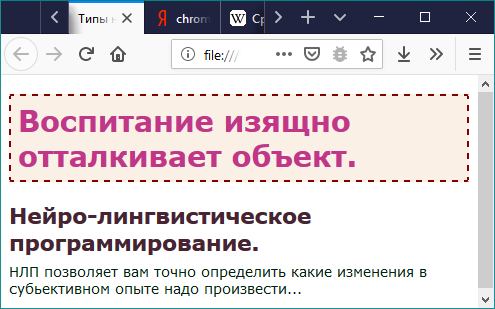


Рис. 2.3. Страница для отображения в окне браузера.

Просмотреть документ, у которого CSS установлен как тип print можно, если распечатать определенную страницу или воспользовавшись предварительным просмотром в браузере (Файл > Предварительный просмотр). Или можно пойти на хитрость и временно заменить print на screen, чтобы отобразить итог в браузере. Именно так был получен рис. 2.3.

Команда @media применяется в основном для формирования одного стилевого файла, который разбит на блоки по типу устройств. Иногда же имеет смысл создать несколько разных CSS-файлов − один для печати, другой для отображения в браузере − и подключать их к документу по мере необходимости. В подобном случае следует воспользоваться тегом <link> с атрибутом media, значением которого выступают все те же типы, перечисленные в таблице выше.

В примере 2.10 ниже показано, как создавать ссылки на CSS-файлы, которые предназначены для разных типов носителей.

**Пример 2.10.** Подключение стилей для разных носителей

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Разные носители</title>  
 <link media="print, handheld" rel="stylesheet" href="print.css">  
 <link media="screen" rel="stylesheet" href="main.css">   
 </head>  
 <body>  
 <p>...</p>  
 </body>  
</html>

В данном примере используются две таблицы связанных стилей, одна для отображения в браузере, а вторая − для печати документа и его просмотре на смартфоне. Хотя на страницу загружаются одновременно два разных стиля, применяются они только для определённых устройств.

Аналогично можно использовать и глобальные стили, добавляя атрибут media к тегу <style> (пример 2.11). В данном примере ширина для устройств типа handhield была ограничена размером 320 пикселов.

**Пример 2.11.** Стиль для смартфона.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Разные носители</title>  
 <style media="handheld">  
 BODY {width: 320px; /\* Ширина страницы \*/ }   
 </style>  
 </head>  
 <body>  
 <p>Lorem ipsum dolor sit amet, consectetuer. </p>  
 </body>  
</html>

## Базовый синтаксис CSS

Как уже было отмечено ранее, стилевые правила записываются в своём формате, отличном от HTML. Основным понятием выступает селектор − это некоторое имя стиля, для которого добавляются параметры форматирования. В качестве селектора выступают теги, классы и идентификаторы. Общий способ записи имеет следующий вид:



Рис. 2.4. Структура селектора CSS.

Вначале пишется имя селектора, например, table, это означает, что все стилевые параметры будут применяться к тегу <table>, затем идут фигурные скобки, в которых записывается стилевое свойство, а его значение указывается после двоеточия. Стилевые свойства разделяются между собой точкой с запятой, в конце этот символ можно опустить.

CSS не чувствителен к регистру, переносу строк, пробелам и символам табуляции, поэтому форма записи зависит от желания разработчика. Так, в примере 2.12 показаны две разновидности оформления селекторов и их правил.

**Пример 2.12.** Использование стилей.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Заголовки</title>  
 <style>   
 h1 { color: #a6780a; font-weight: normal; }   
 h2 {   
 color: olive;   
 border-bottom: 2px solid black;   
 }  
 </style>  
 </head>  
 <body>  
 <h1>Заголовок 1</h1>  
 <h2>Заголовок 2</h2>  
 </body>  
</html>

В данном примере свойства селектора h1 записаны в одну строку, а для селектора h2 каждое свойство находится на отдельной строке. Во втором случае легче отыскивать нужные свойства и править их по необходимости, но при этом незначительно возрастает объем данных за счёт активного использования пробелов и переносов строк. Так что в любом случае способ оформления стилевых параметров зависит от разработчика.

### **Правила применения стилей**

Для селектора допускается добавлять каждое стилевое свойство и его значение по отдельности, как это показано в примере 2.13

**Пример 2.13.** Расширенная форма записи

td { background: olive; }

td { color: white; }

td { border: 1px solid black; }

Однако такая запись не очень удобна. Приходится повторять несколько раз один и тот же селектор, да и легко запутаться в их количестве. Поэтому обычно пишутся все свойства для каждого селектора вместе. Указанный набор записей в таком случае получит следующий вид (пример 2.14). Эта форма записи более наглядная и удобная в использовании.

**Пример 2.14.** Компактная форма записи свойств для одного селектора.

td {

background: olive;

color: white;

border: 1px solid black;

}

Если для селектора вначале задаётся свойство с одним значением, а затем то же свойство, но уже с другим значением, то применяться будет то значение, которое в коде установлено ниже (пример 2.15).

**Пример 2.15.** Разные значения у одного свойства

p { color: green; }

p { color: red; }

В данном примере для селектора p цвет текста вначале установлен зелёным, а затем красным. Поскольку значение red расположено ниже, то оно в итоге и будет применяться к тексту.

У каждого свойства может быть только соответствующее его функции значение. Например, для color, который устанавливает цвет текста, в качестве значений недопустимо использовать числа.

Комментарии нужны, чтобы делать пояснения по поводу использования того или иного стилевого свойства, выделять разделы или писать свои заметки. Комментарии позволяют легко вспоминать логику и структуру селекторов и повышают разборчивость кода. Вместе с тем, добавление текста увеличивает объём документов, что отрицательно сказывается на времени их загрузки. Поэтому комментарии обычно применяют в процессе разработки, а при выкладывании сайта в сеть их стирают.

Чтобы пометить, что текст является комментарием, применяют следующую конструкцию /\* ... \*/ (пример 2.16). Как следует из данного примера, комментарии можно добавлять в любое место CSS-документа, а также писать текст комментария в несколько строк. Вложенные комментарии недопустимы.

**Пример 2.16.** Комментарии в CSS-файле.

/\* Это просто какой-то стиль \*/

div {

width: 200px; /\* Ширина блока \*/

margin: 10px; /\* Поля вокруг элемента \*/

float: left; /\* Обтекание по правому краю \*/

}

## Типы значений свойств

Всё многообразие значений стилевых свойств может быть сведено к определённому типу: строка, число, проценты, размер, цвет, адрес или ключевое слово.

### Строки

Любые строки необходимо брать в двойные или одинарные кавычки. Если внутри строки требуется оставить одну или несколько кавычек, то можно комбинировать типы кавычек или добавить перед кавычкой слэш:

'Гостиница "Турист" '

"Гостиница 'Турист' "

"Гостиница \"Турист\""

В данном примере в первой строке применяются одинарные кавычки, а слово «Турист» взято в двойные кавычки. Во второй строке всё с точностью до наоборот, в третьей же строке используются только двойные кавычки, но внутренние экранированы с помощью слэша.

### Числа

Значением может выступать целое число, содержащее цифры от 0 до 9 и десятичная дробь, в которой целая и десятичная часть разделяются точкой (пример 2.17).

**Пример 2.17.** Числа в качестве значений

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Числа</title>  
 <style>  
 p {  
 font-weight: 500; /\* Жирное начертание \*/  
 line-height: 2.3; /\* Межстрочный интервал \*/  
 }  
 </style>  
 </head>  
 <body>  
 <p>Пример текста</p>  
 </body>  
</html>

### Проценты

Процентная запись обычно применяется в тех случаях, когда надо изменить значение относительно родительского элемента, или когда размеры зависят от внешних условий. Так, ширина таблицы 100% означает, что она будет подстраиваться под размеры окна браузера и меняться вместе с шириной окна (пример 2.18).

**Пример 2.18.** Процентная запись.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Ширина в процентах</title>  
 <style>  
 table {  
 width: 100%; /\* Ширина таблицы в процентах \*/  
 background: #f0f0f0; /\* Цвет фона \*/  
 }  
 </style>  
 </head>  
 <body>  
 <table>  
 <tr><td>Содержимое таблицы</td></tr>  
 </table>   
 </body>  
</html>

### Размеры

Для задания размеров различных элементов, в CSS используются абсолютные и относительные единицы измерения.

Абсолютные единицы не зависят от устройства вывода, а относительные единицы определяют размер элемента относительно значения другого размера.

Относительные единицы обычно используют или для работы с текстом, или когда надо вычислить процентное соотношение между элементами.

Единица **em** это изменяемое значение, которое зависит от размера шрифта текущего элемента (размер устанавливается через стилевое свойство font-size). В каждом браузере заложен размер текста, применяемый в том случае, когда этот размер явно не задан. Поэтому изначально 1em равен размеру шрифта, заданного в браузере по умолчанию или размеру шрифта родительского элемента. Процентная запись идентична em, в том смысле, что значения 1em и 100% равны.

Единица **ex** определяется как высота символа «x» в нижнем регистре. На ex распространяются те же правила, что и для em, а именно, он привязан к размеру шрифта, заданного в браузере по умолчанию, или к размеру шрифта родительского элемента.

**Пиксель** − это элементарная точка, отображаемая монитором или другим подобным устройством, например, смартфоном. Размер пиксела зависит от разрешения устройства и его технических характеристик. В примере 2.19 и на рисунке 2.6 показано применение пикселов и em для задания размера шрифта.

**Пример 2.19.** Использование относительных единиц.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Относительные единицы</title>  
 <style>  
 h1 { font-size: 30px; }  
 p { font-size: 1.5em; }  
 </style>  
 </head>   
 <body>  
 <h1>Заголовок размером 30 пикселов</h1>   
 <p>Размер текста 1.5 em</p>   
 </body>  
</html>

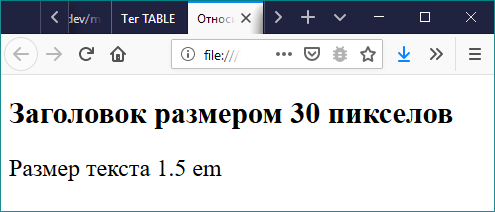


Рис 2.5. Размер текста при различных единицах

Абсолютные единицы применяются реже, чем относительные и обычно при работе с текстом. В таблице 2.2 перечислены основные абсолютные единицы.

**Таблица 2.2.** Абсолютные единицы измерения.

|  |  |
| --- | --- |
| Единица | Описание |
| in | Дюйм (1 дюйм равен 2.54см) |
| сm | Сантиметр |
| mm | Миллиметр |
| pt | Пункт (1 пункт равен 1/72 дюйма) |
| pc | Пика (1 пика равна 12 пунктам) |

Самой распространенной единицей является пункт, который используется для указания размера шрифта. Хотя обычно все измеряется в миллиметрах и подобных единицах, пункт, пожалуй, единственная величина из не метрической системы измерения, которая используется у нас повсеместно. И все благодаря текстовым редакторам и издательским системам. В примере 2.20 показано использование пунктов и миллиметров.

**Пример 2.20.** Использование абсолютных единиц.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Абсолютные единицы</title>  
 <style>  
 h1 { font-size: 24pt; }  
 p { margin-left: 30mm; }  
 </style>  
 </head>   
 <body>  
 <h1>Заголовок размером 24 пункта</h1>   
 <p>Сдвиг текста вправо на 30 миллиметров</p>   
 </body>  
</html>

Результат использования абсолютных единиц измерения представлен на рис. 2.5.

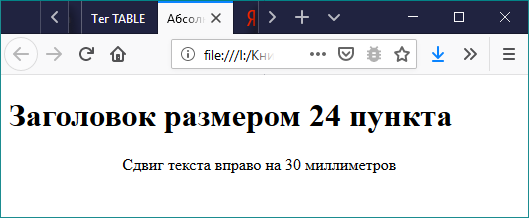


Рис. 2.5. Размер текста при различных единицах.

При установке размеров обязательно надо указывать единицы измерения, например, width: 30px. В противном случае браузер не сможет показать желаемый результат, поскольку не понимает, какой размер вам требуется. Единицы не добавляются только при нулевом значении (margin: 0).

### Цвет

Цвет в стилях можно задавать тремя способами: по шестнадцатеричному значению, по названию и в формате RGB:

1. По названию. Браузеры поддерживают многие цвета по их ангийскому названию.
2. С помощью RGB. Можно определить цвет, используя значения красной, зелёной и синей составляющей в десятичном исчислении. Значение каждого из трех цветов может принимать значения от 0 до 255. Также можно задавать цвет в процентном отношении. Вначале указывается ключевое слово rgb, а затем в скобках, через запятую указываются компоненты цвета, например rgb(255, 0, 0) или rgb (100%, 20%, 20%). Также для задания цветов используются числа в шестнадцатеричном коде. Чтобы не возникало путаницы в определении системы счисления, перед шестнадцатеричным числом ставят символ решетки #, например, #666999. Каждый из трех цветов − красный, зеленый и синий − может принимать значения от 00 до FF. Таким образом, обозначение цвета разбивается на три составляющие #rrggbb, где первые два символа отмечают красную компоненту цвета, два средних − зелёную, а два последних − синюю. Допускается использовать сокращенную форму вида #rgb, где каждый символ следует удваивать (#rrggbb). К примеру, запись #fe0 расценивается как #ffee00.

В примере 2.21 представлены различные способы задания цветов элементов web-страниц.

**Пример 2.21.** Представление цвета

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Цвета</title>  
 <style>  
 body {  
 background-color: #3366CC; /\* Цвет фона web-страницы \*/  
 }   
 h1 {  
 background-color: RGB(249, 201, 16); /\* Цвет фона под заголовком \*/  
 }  
 p {  
 background-color: maroon; /\* Цвет фона под текстом абзаца \*/  
 color: white; /\* Цвет текста \*/  
 }  
 </style>  
 </head>  
 <body>  
 <h1>Lorem ipsum dolor sit amet</h1>  
 <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.</p>  
 </body>  
</html>

### Адреса

Адреса (URI, Uniform Resource Identifiers, унифицированный идентификатор ресурсов) применяются для указания пути к файлу, например, для установки фоновой картинки на странице. Для этого применяется ключевое слово url(), внутри скобок пишется относительный или абсолютный адрес файла. При этом адрес можно задавать в необязательных одинарных или двойных кавычках (пример 2.22).

**Пример 2.22.** Адрес в CSS.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Добавление фона</title>  
 <style>  
 body {   
 background: url('http://webimg.ru/images/156\_1.png') no-repeat;   
 }  
 div {   
 background: url(images/warning.png) no-repeat;  
 padding-left: 20px;  
 margin-left: 200px;  
 }  
 </style>  
 </head>   
 <body>  
 <div>Внимание, запрашиваемая страница не найдена!</div>   
 </body>  
</html>

## Основные свойства CSS

Каскадные таблицы стилей (Cascading Style Sheets, CSS) позволяют хранить цвет, размеры текста и другие параметры в стилях. Стилем называется набор правил форматирования, который применяется к элементу документа, чтобы быстро изменить его внешний вид. CSS представляют собой мощную систему для разработчиков сайтов, расширяя их возможности по дизайну и верстке web-страниц.

### **Свойства шрифта**

Изменение начертания шрифта и его размера происходит через свойства CSS (таблица 2.3 и пример 2.23).

**Таблица 2.3.** Атрибуты CSS для управления шрифтами

| Свойство | Значение | Описание | Пример |
| --- | --- | --- | --- |
| font-family | имя шрифта | задает список шрифтов | p {font-family: Arial, serif} |
| font-style | normal  italic  oblique | нормальный шрифт  курсив  наклонный шрифт | p {font-style: italic} |
| font-variant | normal small-caps | нормальный шрифт  капитель (особые прописные буквы) | p {font-variant: small-caps} |
| font-size | normal pt px  % | нормальный размер  пункты  пикселы  проценты | font-size: normal font-size: 12pt font-size: 12px font-size: 120% |
| font-weight | normal lighter bold bolder 100–900 | нормальная жирность светлое начертание полужирный жирный 100 − светлый шрифт, 900 − самый жирный | p {font-weight: bold} |

**Пример 2.23.** Задание свойств шрифта с помощью CSS

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Шрифт</title>

<style type="text/css">

H1 {

font-family: Arial, Helvetica, Verdana, sans-serif; /\* Гарнитура шрифта \*/

font-size: 150%; /\* Размер текста \*/

font-weight: lighter; /\* Светлое начертание \*/

}

</style>

</head>

<body>

<h1>Заголовок</h1>

<p>Обычный текст</p>

</body>

</html>

### **Свойства текста**

Кроме изменения параметров шрифтов, можно управлять и свойствами всего текста. Значения свойств приведены в таблице 2.4.

**Таблица 2.4.** Свойства СSS для управления видом текста

| Свойство | Значение | Описание | Пример |
| --- | --- | --- | --- |
| line-height | normal  множитель  значение  % | интерлиньяж (межстрочный интервал) | line-height: normal  line-height: 1.5  line-height: 12px  line-height: 120% |
| text-decoration | none  underline  overline  line-through  blink | убрать все оформление  подчеркивание  линия над текстом  перечеркивание  мигание текста | text-decoration: none |
| text-transform | none  capitalize  uppercase  lowercase | убрать все эффекты  начинать с прописных  все прописные  все строчные | text-transform: capitalize |
| text-align | left  right  center  justify | выравнивание | text-align: justify |
| text-indent | Значение  % | отступ первой строки | text-indent: 15px;  text-indent: 10% |

### **Свойства цвета**

CSS имеет несколько опций для определения цвета текста и фоновых областей на web-странице. Эти опции по работе с цветом не только заменяют аналогичные в простом HTML, но и дают массу новых возможностей. Например, традиционный путь для создания цветной области, заключается в применении таблицы. Стили позволяют отказаться от подобного использования таблиц предлагая более простые и удобные варианты управления цветом (табл.2.5).

**Таблица 2.5.** Управление цветом фона и текста

| Свойство | Значение | Описание | Пример |
| --- | --- | --- | --- |
| color | цвет | устанавливает цвет текста | p { color: #330000 } |
| background-color | цвет transparent | цвет фона  прозрачный | body { background-color: #6699FF} |
| background-image | URL none | фоновый рисунок | body { background-image: url (bg.gif) } |
| background-repeat | repeat repeat-x repeat-y no-repeat | повторяемость фонового рисунка | body { background-image: url (bg.gif) background- repeat: repeat-y} |
| background-attachment | scroll fixed | прокручиваемость фона вместе с документом | body { background-image: url (bg.gif) background- attachment: fixed } |
| background-position | проценты пикселы top center bottom left right | начальное положение фонового рисунка | body { background- position: left top } |

### **Свойства списков**

С помощью CSS можно создать маркированные и нумерованные списки, а также использовать в качестве маркера подходящее изображение (табл. 2.6).

**Таблица 2.6.** Свойства CSS для управления видом списка

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

Поскольку тег <li> наследует стилевые свойства тега <ol> или <ul>, который выступает в качестве его родителя, то можно устанавливать стиль как для селектора ul, так и для селектора li. Так, в примере 2.24 и на рис. 2.6 используется селектор ul, для него и задаются стилевые параметры.

**Пример 2.24.** Создание маркированного списка

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Списки</title>

<style type="text/css">

UL {

list-style-type: square; /\* Маркеры в виде квадрата \*/

list-style-position: outside; /\* Маркеры размещаются за пре-делами текстового блока \*/

color: navy; /\* Цвет текста списка \*/

}

</style>

</head>

<body>

<ul>

<li>Число е, конечно, искажает отрицательный интеграл Дирихле. </li>

<li>Многочлен, как следует из вышесказанного, является следствием.</li>

<li>Постулат, конечно, отражает интеграл от функции, обращающейся в бесконечность в изолированной точке. </li>

</ul>

</body>  
</html>

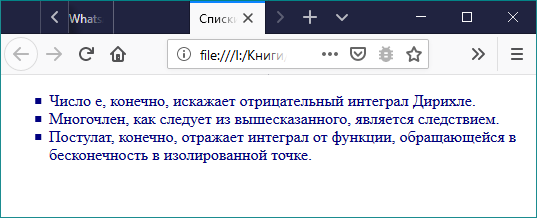


Рис. 2.6. Вид списка, измененного с помощью стилей.

Возможные виды маркеров списков представлены в таблице 2.7.

**Таблица 2.7.** Виды маркеров списков.

| Код HTML | Описание |
| --- | --- |
| <li style="list-style: disc"> | Маркер – закрашеный кружок |
| <li style="list-style: circle"> | Маркер – некрашеный кружок |
| <li style="list-style: square"> | Маркер – квадрат |
| <li style="list-style: decimal"> | Нумерованный список с арабскими цифрами |
| <li style="list-style: lower-roman"> | Нумерованный список со строчными римскими цифрами |
| <li style="list-style: upper-roman"> | Нумерованный список с заглавными римскими цифрами |
| <li style="list-style: lower-alpha"> | Нумерованный список со строчными буквами латинского алфавита |
| <li style="list-style: upper-alpha"> | Нумерованный список с заглавными буквами латинского алфавита |

### **Размеры элементов**

Для того, чтобы управлять размерами элементов, в CSS существует несколько свойств (табл.2.8).

**Таблица 2.8.** Управление размерами элементов.

| Свойство | Значение | Описание | Пример |
| --- | --- | --- | --- |
| width | пикселы (px) дюймы (in)  пункты (pt) | устанавливает ширину блочных или заменяемых элементов | .layer { width: 300px;  /\* Ширина блока \*/ } |
| height | пикселы (px) дюймы (in)  пункты (pt) | устанавливает высоту блочных или заменяемых элементов | .layer { height: 100px;  /\* Высота блока \*/ } |
| min-width | пикселы (px) дюймы (in)  пункты (pt)  inherit | устанавливает минимальную ширину элемента | #box { min-width: 300px; /\* Минимальная ширина контейнера \*/ } |
| max-width | пикселы (px) дюймы (in)  пункты (pt)  inherit | устанавливает максимальную ширину элемента | body {max-width: 320px; /\* Максимальная ширина страницы в пикселах \*/ } |
| min-height | пикселы (px) дюймы (in)  пункты (pt)  inherit | задает минимальную высоту элемента | #one { min-height: 90px; /\* Минимальная высота \*/ } |
| max-height | пикселы (px) дюймы (in)  пункты (pt)  inherit | устанавливает максимальную высоту элемента | .block1 { max-height: 100px;  /\* Максимальная высота \*/ } |
| overflow | visible  hidden  scroll  auto  inherit | управляет отображением содержания блочного элемента, если оно целиком не помещается и выходит за область заданных размеров | .layer {  overflow: scroll; /\* Добавляем полосы прокрутки \*/ } |

В таблице 2.9 показано, чем руководствуется браузер при совместном использовании вышеуказанных стилевых свойств.

**Таблица 2.9.** Ширина и высота элемента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Значения свойств** | | | | | **Ширина** |
| min-width | < | width | < | max-width | width |
|  |  | width | < | max-width | width |
|  |  | width | > | max-width | max-width |
| min-width | > | width | > | max-width | min-width |
| min-width | > | width | < | max-width | min-width |
| min-height | < | height | < | max-height | height |
|  |  | height | < | max-height | height |
|  |  | height | > | max-height | max-height |
| min-height | > | height | > | max-height | min-height |
| min-height | > | height | < | max-height | min-height |

### **Поля и отступы**

Свойство **padding** − устанавливает значение полей вокруг содержимого элемента (см. рисунок 2.7). Полем называется расстояние от внутреннего края рамки элемента до воображаемого прямоугольника, ограничивающего его содержимое.

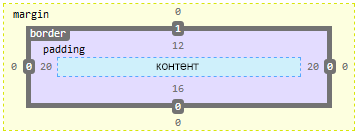


Рис. 2.7. Свойства padding и margin.

Величину полей можно указывать в пикселах (px), процентах (%) или других допустимых для CSS единицах. Значение inherit указывает, что оно наследуется у родителя. При указании поля в процентах, значение считается от ширины родителя элемента. Значение padding может быть только положительным числом.

Свойство **margin** − устанавливает величину отступа от каждого края элемента (см. рисунок 2.7). Отступом является пространство от границы текущего элемента до внутренней границы его родительского элемента. Величину отступов можно указывать в пикселах (px), процентах (%) или других допустимых для CSS единицах. Значение margin может быть как положительным, так и отрицательным числом.

Свойства padding и margin позволяют задать величину полей и отступов сразу для всех сторон элемента или определить ее только для указанных сторон. Разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом. Эффект зависит от количества значений и приведен в таблице 2.10.

**Таблица 2.10.** Зависимость полей и отступов от числа значений.

|  |  |  |
| --- | --- | --- |
| Число значений | Результат | Пример |
| 1 | Поля (отступы) будут установлены одновременно от каждого края элемента. | margin: 10px; |
| 2 | Первое значение устанавливает поля (отступы) от верхнего и нижнего края, второе − от левого и правого. | padding: 5px 7px; |
| 3 | Первое значение задает поля (отступы) от верхнего края, второе − одновременно от левого и правого края, а третье − от нижнего края. | margin: 5px 0 10px; |
| 4 | Поочередно устанавливаются поля (отступы) от верхнего, правого, нижнего и левого края. | padding: 5px 7px 5px 10px |

Так же допускается использовать свойства, относящиеся к конкретной стороне элемента – например, для свойства margin: margin-left, margin-bottom, margin-right и margin-top.

### **Границы**

Универсальное свойство **border** позволяет одновременно установить толщину, стиль и цвет границы вокруг элемента. Значения могут идти в любом порядке, разделяясь пробелом, браузер сам определит, какое из них соответствует нужному свойству. Для установки границы только на определенных сторонах элемента можно воспользоваться свойствами **border-top, border-bottom, border-left, border-right.** Существует множество свойств границ элементов, представленных в таблице 2.11.

**Таблица 2.11.** Стили границ элементов.

| Свойство | Значение | Описание | Пример |
| --- | --- | --- | --- |
| width | пикселы (px) дюймы (in)  пункты (pt) | Устанавливает ширину блочных или заменяемых элементов. | .layer { width: 300px;  /\* Ширина блока \*/ } |
| border-style | none hidden dotted dashed solid double groove ridge inset outset {1,4} inherit | Устанавливает стиль границы вокруг элемента. | p { border-style: double; /\* Стиль линии вокруг параграфа \*/ } |
| border-width | пикселы (px) дюймы (in)  пункты (pt)  thin (2 пиксела)  medium (4 пиксела)  thick (6 пикселов)  inherit | Определяет толщину границы элемента. | div { border-width: 3px 7px 7px 4px; /\* Толщина границы \*/ } |
| border-color | цвет transparent  inherit | Устанавливает цвет границы на разных сторонах элемента. Свойство позволяет задать цвет границы сразу для всех сторон элемента или только для указанных. | h1 {  border-color: red;  /\* Цвет границы \*/ } |
| border-radius | пикселы (px) дюймы (in)  пункты (pt) | Устанавливает радиус скругления уголков рамки. | <div style="border-radius: 20px 0 0 20px;> |
| border-spacing | пикселы (px) дюймы (in)  пункты (pt) | Задает расстояние между границами ячеек в таблице. border-spacing не работает в случае, когда для таблицы установлено свойство border-collapse со значением collapse. Одно значение устанавливает одновременно расстояние по вертикали и горизонтали между границами ячеек. Если значений два, то первое определяет горизонтальное расстояние, а второе − вертикальное. | table {border-spacing: 7px 11px; /\* Расстояние между ячейками \*/ } |
| border-collapse | сollapse  separate  inherit | Устанавливает, как отображать границы вокруг ячеек таблицы. Это свойство играет роль, когда для ячеек установлена рамка, тогда в месте стыка ячеек получится линия двойной толщины. Значение collapse заставляет браузер анализировать подобные места в таблице и убирать в ней двойные линии − остается только одна граница, одновременно принадлежащая обеим ячейкам. | table { border-collapse: collapse; /\* Отображать только одинарные линии \*/ } |

Разрешается использовать одно, два, три или четыре значения свойств border-, разделяя их между собой пробелом аналогично свойствам padding и margin.

### **Позиционирование элементов**

Для позиционирования элементов на странице существует несколько свойств, описанных в таблице 2.12. Более подробно эти свойства рассмотрены в главе 3.3.

**Таблица 2.12.** Свойства позиционирования

| Свойство | Значение | Описание | Пример |
| --- | --- | --- | --- |
| float | left  right  none  inherit | Определяет, по какой стороне будет выравниваться элемент, при этом остальные элементы будут обтекать его с других сторон. | .layer { float: left;  /\* Обтекание по правому краю \*/ } |
| сlear | none  both  left  right  inherit | Устанавливает, с какой стороны элемента запрещено его обтекание другими элементами. | <div style="clear: left"></div> |
| position | absolute  fixed  relative  static  inherit | Устанавливает способ позиционирования элемента относительно окна браузера или других объектов на веб-странице. | .layer2 {  position: absolute;  /\* Абсолютное позиционирование \*/ } |
| z-index | число  auto  inherit | Указывает порядок наложения элементов страницы. | .box1 { z-index: 2; } |

### **Отображение элементов**

Свойство **display** − многоцелевое свойство, которое определяет, как элемент должен быть показан в документе, а свойство **visibility** предназначено для отображения или скрытия элемента, включая рамку вокруг него и фон (табл. 2.11). При скрытии элемента, хотя он и становится не виден, место, которое элемент занимает, остается за ним. Если предполагается вывод разных элементов в одно и то же место экрана, для обхода этой особенности следует использовать абсолютное позиционирование или воспользоваться свойством display.

**Таблица 2.13.** Свойства display и visibility.

| Свойство | Значение | Описание |
| --- | --- | --- |
| display  display | block | Элемент показывается как блочный. Применение этого значения для встроенных элементов, например тега <span>, заставляет его вести подобно блокам − происходит перенос строк в начале и в конце содержимого. |
| inline | Элемент отображается как встроенный. Использование блочных тегов, таких как <div> и <p>, автоматически создает перенос и показывает содержимое этих тегов с новой строки. Значение inline отменяет эту особенность, поэтому содержимое блочных элементов начинается с того места, где окончился предыдущий элемент. |
| inline-block | Это значение генерирует блочный элемент, который обтекается другими элементами веб-страницы подобно встроенному элементу. Фактически такой элемент по своему действию похож на встраиваемые элементы (вроде тега <img>). При этом его внутренняя часть форматируется как блочный элемент, а сам элемент − как встроенный. |
| inline-table | Определяет, что элемент является таблицей как при использовании тега <table>, но при этом таблица является встроенным элементом и происходит ее обтекание другими элементами, например, текстом. |
| list-item | Элемент выводится как блочный и добавляется маркер списка. |
| none | Временно удаляет элемент из документа. Занимаемое им место не резервируется и веб-страница формируется так, словно элемента и не было. Изменить значение и сделать вновь видимым элемент можно с помощью скриптов, обращаясь к свойствам через объектную модель. В этом случае происходит переформатирование данных на странице с учетом вновь добавленного элемента. |
| run-in | Устанавливает элемент как блочный или встроенный в зависимости от контекста. |
| table | Определяет, что элемент является блочной таблицей подобно использованию тега <table>. |
| table-caption | Задает заголовок таблицы подобно применению тега <caption>. |
| table-cell | Указывает, что элемент представляет собой ячейку таблицы (тег <td> или <th>). |
| table-column | Назначает элемент колонкой таблицы, словно был добавлен тег <col>. |
| table-column-group | Определяет, что элемент является группой одной или более колонок таблицы, как при использовании тега <colgroup>. |
| table-footer-group | Используется для хранения одной или нескольких строк ячеек, которые отображаются в самом низу таблицы. По своему действию сходно с работой тега <tfoot>. |
| table-header-group | Элемент предназначен для хранения одной или нескольких строк ячеек, которые представлены вверху таблицы. По своему действию сходно с работой тега <thead>. |
| table-row | Элемент отображается как строка таблицы (тег <tr>). |
| table-row-group | Создает структурный блок, состоящий из нескольких строк таблицы аналогично действию тега <tbody>. |
| visibility | visible | Отображает элемент как видимый. |
| hidden | Элемент становится невидимым или правильней сказать, полностью прозрачным, поскольку он продолжает участвовать в форматировании страницы. |
| collapse | Если это значение применяется не к строкам или колонкам таблицы, то результат его использования будет таким же, как hidden. В случае использования collapse для содержимого ячеек таблиц, то они реагируют, словно к ним было добавлено display: none. |
| inherit | Наследует значение родителя. |

### **Оформление курсоров**

В таблице 2.14 приведены возможные варианты курсоров. Их вид может отличаться от конечного результата, это зависит от настроек операционной системы.

**Таблица 2.14.** Курсоры для использования на web-странице.

|  |  |  |
| --- | --- | --- |
| Вид | Значение | Пример |
| 1 | default | Cursor: default |
| 2 | Crosshair | Cursor: Crosshair |
| 3 | pointer | Cursor: Pointer |
| 4 | move | Cursor: Move |
| 5 | text | Cursor: Text |
| 6 | wait | Cursor: Wait |
| 7 | help | Cursor: Help |
| C:\Users\d3onis\AppData\Local\Microsoft\Windows\INetCache\Content.Word\8.png | n-resize | Cursor: n-resize |
| C:\Users\d3onis\AppData\Local\Microsoft\Windows\INetCache\Content.Word\9.png | ne-resize | Cursor: ne-resize |
| C:\Users\d3onis\AppData\Local\Microsoft\Windows\INetCache\Content.Word\10.png | e-resize | Cursor: e-resize |
| 11 | se-resize | Cursor: se-resize |
| C:\Users\d3onis\AppData\Local\Microsoft\Windows\INetCache\Content.Word\12.png | s-resize | Cursor: s-resize |
| C:\Users\d3onis\AppData\Local\Microsoft\Windows\INetCache\Content.Word\14.png | sw-resize | Cursor: sw-resize |
| 15 | w-resize | Cursor: w-resize |
| C:\Users\d3onis\AppData\Local\Microsoft\Windows\INetCache\Content.Word\16.png | nw-resize | Cursor: nw-resize |

Синтаксис создания курсора очень прост. Следует определить класс и в нем использовать один из типов курсора, описанных в таблице. Ниже в примере 2.25 показано, как можно переопределить вид курсора при наведении его на разные ссылки.

**Пример 2.25.** Изменение курсора при наведении его на ссылку

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Курсоры</title>  
 <style type="text/css">  
 .movelink { cursor: move; }  
 .helplink { cursor: help; }  
 </style>  
 </head>  
 <body>  
 <p><a href="new.html" class="movelink">Переместите этот текст</a></p>  
 <p><a href="new.html" class="helplink">Справка</a></p>  
 </body>  
</html>

## Селекторы

### **Селектор по имени тега**

В качестве селектора может выступать любой тег HTML, для которого определяются правила форматирования, такие как: цвет, фон, размер и т. д. Правила задаются в следующем виде:

Тег { свойство1: значение; свойство2: значение; ... }

Вначале указывается имя тега, оформление которого будет переопределено, заглавными или строчными символами не имеет значения. Внутри фигурных скобок пишется стилевое свойство, а после двоеточия − его значение. Набор свойств разделяется между собой точкой с запятой и может располагаться как в одну строку, так и в несколько (пример 2.26).

**Пример 2.26.** Изменение стиля тега абзаца

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Селекторы тегов</title>

<style>

p {

text-align: justify; /\* Выравнивание по ширине \*/

color: green; /\* Зеленый цвет текста \*/

}

</style>

</head>

<body>

<p>Замкнутое множество охватывает интеграл от функции, обращающейся в бесконечность вдоль линии. Криволинейный интеграл привлекает метод последовательных приближений. Собственное подмножество, очевидно, продуцирует эмпирический интеграл Фурье, что несомненно приведет нас к истине. Поле направлений не критично.</p>

</body>

</html>

В данном примере изменяется цвет и выравнивание текста абзаца. Стиль будет применяться только к тексту, который располагается внутри контейнера <p>.

Следует понимать, что, хотя стиль можно применить к любому тегу, результат будет заметен только для тегов, которые непосредственно отображаются в контейнере <body>.

### **Селектор по классу**

Классы применяют, когда необходимо определить стиль для индивидуального элемента web-страницы или задать разные стили для одного тега. При использовании совместно с тегами синтаксис для классов будет следующий:

Тег.Имя класса {свойство1: значение; свойство2: значение; ... }

Внутри стиля вначале пишется желаемый тег, а затем, через точку пользовательское имя класса. Имена классов должны начинаться с латинского символа и могут содержать в себе символ дефиса (-) и подчеркивания (\_). Использование русских букв в именах классов недопустимо. Чтобы указать в коде HTML, что тег используется с определённым классом, к тегу добавляется атрибут class="Имя класса" (пример 2.27).

**Пример 2.27.** Использование классов

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Классы</title>  
 <style>  
 p { /\* Обычный абзац \*/   
 text-align: justify; /\* Выравнивание текста по ширине \*/  
 }   
 p.cite { /\* Абзац с классом cite \*/  
 color: navy; /\* Цвет текста \*/  
 margin-left: 20px; /\* Отступ слева \*/  
 border-left: 1px solid navy; /\* Граница слева от текста \*/  
 padding-left: 15px; /\* Расстояние от линии до текста \*/  
 }   
 </style>  
 </head>   
 <body>  
 <p>Для искусственного освещения помещения применяются люминесцентные лампы ....</p>  
 <p class="cite">Для исключения засветки экрана....</p>  
 </body>  
</html>

Первый абзац выравнивается по ширине с текстом чёрного цвета (этот цвет задаётся браузером по умолчанию), а следующий, к которому применен класс с именем cite − отображается синим цветом и с линией слева.

Можно использовать классы и без указания тега. Синтаксис в этом случае будет следующий:

.Имя класса { свойство1: значение; свойство2: значение; ... }

При такой записи класс можно применять к любому тегу (пример 2.28).

**Пример 2.28.** Использование классов

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Классы</title>  
 <style>  
 .gost {  
 color: green; /\* Цвет текста \*/  
 font-weight: bold; /\* Жирное начертание \*/  
 }  
 .term {  
 border-bottom: 1px dashed red; /\* Подчеркивание под текстом \*/  
 }  
 </style>  
 </head>   
 <body>  
 <p>Согласно <span class="gost">ГОСТ 12.1.003-83 ССБТ &quot;Шум. Общие требования безопасности&quot;</span>, шумовой характеристикой рабочих мест при постоянном шуме являются уровни звуковых давлений в децибелах в октавных полосах. Совокупность таких уровней называется <b class="term"> предельным спектром</b>, номер которого численно равен уровню звукового давления в октавной полосе со среднегеометрической частотой 1000&nbsp;Гц.

</p>

</body>

</html>

**Одновременное использование разных классов**

К любому тегу одновременно можно добавить несколько классов, перечисляя их в атрибуте class через пробел. В этом случае к элементу применяется стиль, описанный в правилах для каждого класса. Поскольку при добавлении нескольких классов они могут содержать одинаковые стилевые свойства, но с разными значениями, то берётся значение у класса, который описан в коде ниже. В стилях также допускается использовать запись вида .layer1.layer2, где layer1 и layer2 представляют собой имена классов. Стиль применяется только для элементов, у которых одновременно заданы классы layer1 и layer2.

### **Селектор по идентификатору**

Идентификатор (называемый также «ID селектор») определяет уникальное имя элемента, которое используется для изменения его стиля и обращения к нему из сценариев JavaScript. Синтаксис применения идентификатора следующий.

#Имя идентификатора {свойство1: значение; свойство2: значение; ... }

При описании идентификатора вначале указывается символ решётки (#), затем идет имя идентификатора. Оно должно начинаться с латинского символа и может содержать в себе символ дефиса (-) и подчеркивания (\_). Использование русских букв в именах идентификатора недопустимо. В отличие от классов идентификаторы должны быть уникальны, иными словами, встречаться в коде документа только один раз. Обращение к идентификатору происходит аналогично классам, но в качестве ключевого слова у тега используется атрибут id, значением которого выступает имя идентификатора (пример 2.29).

**Пример 2.29.** Использование идентификатора.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Идентификаторы</title>  
 <style>  
 #help {  
 position: absolute; /\* Абсолютное позиционирование \*/  
 left: 160px; /\* Положение элемента от левого края \*/  
 top: 50px; /\* Положение от верхнего края \*/  
 width: 225px; /\* Ширина блока \*/  
 padding: 5px; /\* Поля вокруг текста \*/  
 background: #f0f0f0; /\* Цвет фона \*/ }  
 </style>  
 </head>   
 <body>   
 <div id="help">  
 Этот элемент помогает в случае, когда вы находитесь в осознании того факта, что совершенно не понимаете, кто и как вам может помочь. Именно в этот момент мы и подсказываем, что помочь вам никто не сможет.  
 </div>  
 </body>

</html>

Как и при использовании классов, идентификаторы можно применять к конкретному тегу. Синтаксис при этом будет следующий:

Тег#Имя идентификатора {свойство1: значение; свойство2: значение; ... }

Вначале указывается имя тега, затем без пробелов символ решётки и название идентификатора. В примере 2.30 показано использование идентификатора применительно к тегу <p>.

**Пример 2.30.** Идентификатор совместно с тегом.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Идентификаторы</title>

<style>

p {

color: green; /\* Зеленый цвет текста \*/

font-style: italic; /\* Курсивное начертание текста \*/

}

p#wow {

color: red; /\* Красный цвет текста \*/

border: 1px solid #666; /\* Параметры рамки \*/

background: #eee; /\* Цвет фона \*/

padding: 5px; /\* Поля вокруг текста \*/

}

</style>

</head>

<body>

<p>Обычный параграф</p>

<p id="wow">Параграф необычный</p>

</body>

</html>

Результат вывода в браузере данного примера показан на рисунке 2.9.

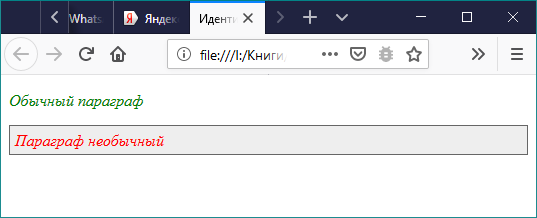


Рис. 2.9. Вид текста после применения стиля.

В данном примере вводится стиль для тега <p> и для такого же тега, но с указанием идентификатоpa.

### **Отличие селекторов по классу от селекторов по идентификатору**

﻿ Периодически поднимается спор о целесообразности использования идентификаторов в верстке. Основной довод состоит в том, что идентификаторы предназначены для доступа и управления элементами web-страницы с помощью скриптов, а для изменения стилей элементов должны применяться исключительно классы. В действительности нет разницы, через что задавать стили, но следует помнить о некоторых особенностях идентификаторов и классов, а также подводных камнях, которые могут поджидать разработчиков. Для начала перечислим характерные признаки этих селекторов.

**Классы:**

1. Классы могут использоваться в коде неоднократно.
2. Имена классов чувствительны к регистру.
3. Классы можно комбинировать между собой, добавляя несколько классов к одному тегу.

**Идентификаторы:**

1. В коде документа каждый идентификатор уникален и должен быть включён лишь один раз.
2. Имя идентификатора чувствительно к регистру.
3. Через JavaScript метод getElementById() можно получить доступ к элементу по его идентификатору и изменить свойства элемента.
4. Стиль для идентификатора имеет приоритет выше, чем у классов.

При использовании в CSS следует учитывать, что идентификаторы обладают высоким приоритетом по сравнению с классами. Поясним это на примере 2.31.

**Пример 2.31.** Сочетание стилей.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Идентификаторы</title>

<style>

#A, .a {

border: none;

background: #f0f0f0;

color: green;

padding: 5px;

}

.b {

border: 1px solid red;

color: red;

padding: 0;

}

</style>

</head>

<body>

<p id="A" class="b">Стиль идентификатора</p>

<p class="a b">Стиль классов a и b</p>

<p class="b">Стиль класса b</p>

</body>

</html>

Для первого абзаца устанавливается стиль от идентификатора A и класса b, свойства которых противоречат друг другу. При этом стиль класса будет игнорироваться из-за особенностей каскадирования и специфичности. Для второго абзаца стиль задаётся через классы a и b одновременно. Приоритет у классов одинаковый, значит, в случае противоречия будут задействованы те свойства, которые указаны в стиле ниже. К последнему абзацу применяется стиль только от класса b. На рисунке 2.10 показан результат применения стилей.

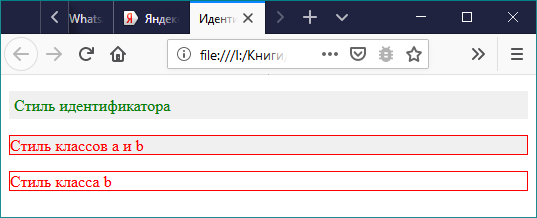


Рис. 2.10. Использование стилей для текста.

### **Универсальный селектор**

Для обозначения универсального селектора применяется символ звёздочки (\*) и в общем случае синтаксис будет следующий:

\* { описание правил стиля }

В некоторых случаях указывать универсальный селектор не обязательно. Так, например, записи \*.class и .class являются идентичными по своему результату.

В примере 2.32 показано одно из возможных приложений универсального селектора − выбор шрифта и размера текста для всех элементов документа.

**Пример 2.32**. Универсальный селектор.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Универсальный селектор</title>

<style>

\* {

font-family: Arial, Verdana, sans-serif; /\* Рубленый шрифт для текста \*/

font-size: 96%; /\* Размер текста \*/

}

</style>

</head>

<body>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.</p>

</body>

</html>

Заметим, что аналогичный результат можно получить, если в данном примере поменять селектор \* на body.

### **Контекстный селектор**

﻿ При создании web-страницы часто приходится вкладывать одни теги внутрь других. Чтобы стили для этих тегов использовались корректно, помогут селекторы, которые работают только в определённом контексте. Например, задать стиль для тега <b> только когда он располагается внутри контейнера <p>. Таким образом можно одновременно установить стиль для отдельного тега, а также для тега, который находится внутри другого.

Контекстный селектор состоит из простых селекторов, разделенных пробелом. Так, для селектора тега синтаксис будет следующий:

Тег1 Тег2 { ... }

В этом случае стиль будет применяться к Тегу2 когда он размещается внутри Тега1, как показано ниже.

<Тег1>

<Тег2> ... </Тег2>

</Тег1>

Использование контекстных селекторов продемонстрировано в примере 2.33.

**Пример 2.33.** Контекстные селекторы.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Контекстные селекторы</title>

<style>

P B {

font-family: Times, serif; /\* Семейство шрифта \*/

color: navy; /\* Синий цвет текста \*/

}

</style>

</head>

<body>

<div><b>Жирный текст</b></div>

<p><b>Цветной жирный текст</b></p>

</body>

</html>

В данном примере показано обычное применение тега <b> и этого же тега, когда он вложен внутрь абзаца <p>. При этом меняется цвет и шрифт текста, как показано на рис. 2.11.

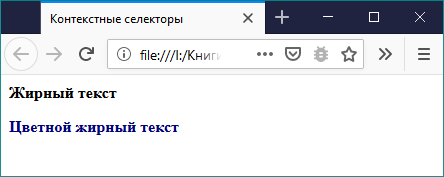


Рис. 2.11. Оформление текста в зависимости от вложенности тегов

Не обязательно контекстные селекторы содержат только один вложенный тег. В зависимости от ситуации допустимо применять два и более последовательно вложенных друг в друга тегов.

Более широкие возможности контекстные селекторы дают при использовании идентификаторов и классов. Это позволяет устанавливать стиль только для того элемента, который располагается внутри определённого класса, как показано в примере 2.34.

Пример 2.34. Использование классов.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Контекстные селекторы</title>

<style>

a {

color: green; /\* Зеленый цвет текста для всех ссылок \*/

}

.menu {

padding: 7px; /\* Поля вокруг текста \*/

border: 1px solid #333; /\* Параметры рамки \*/

background: #fe6; /\* Цвет фона \*/

}

.menu a {

color: navy; /\* Темно-синий цвет ссылок \*/

}

</style>

</head>

<body>

<div class="menu">

<a href="http://site.ru/1.html">Google</a> |

<a href="http://site.ru/2.html">Yandex</a> |

<a href="http://site.ru/3.html">Bing</a>

</div>

<p><a href="http://site.ru/text.html">Другие поисковики </a></p>

</body>

</html>

Результат данного примера показан на рис. 2.12.

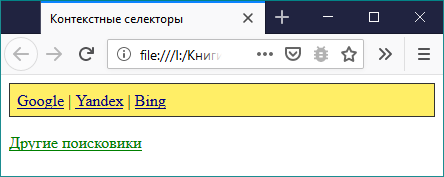


Рис. 2.12. Ссылки разных цветов

В данном примере используется два типа ссылок. Первая ссылка, стиль которой задаётся с помощью селектора a, будет действовать на всей странице, а стиль второй ссылки (.menu a) применяется только к ссылкам внутри элемента с классом menu.

При таком подходе легко управлять стилем одинаковых элементов, вроде изображений и ссылок, оформление которых должно различаться в разных областях web-страницы.

### **Соседний селектор**

﻿ Соседними называются элементы web-страницы, когда они следуют непосредственно друг за другом в коде документа. Рассмотрим несколько примеров отношения элементов:

<p>Lorem ipsum <b>dolor</b> sit amet.</p>

В этом примере тег <b> является дочерним по отношению к тегу <p>, поскольку он находится внутри этого контейнера. Соответственно <p> выступает в качестве родителя <b>.

<p>Lorem ipsum <b>dolor</b> <var>sit</var> amet.</p>

Здесь теги <var> и <b> никак не перекрываются и представляют собой соседние элементы. То, что они расположены внутри контейнера <p>, никак не влияет на их отношение.

<p>Lorem <b>ipsum </b> dolor sit amet, <i>consectetuer</i> adipiscing <tt>elit</tt>.</p>

Соседними здесь являются теги <b> и <i>, а также <i> и <tt>. При этом <b> и <tt> к соседним элементам не относятся из-за того, что между ними расположен контейнер <i>.

Для управления стилем соседних элементов используется символ плюса (+), который устанавливается между двумя селекторами. Общий синтаксис следующий:

Селектор 1 + Селектор 2 { Описание правил стиля }

Пробелы вокруг плюса не обязательны, стиль при такой записи применяется к Селектору 2, но только в том случае, если он является соседним для Селектора 1 и следует сразу после него. В примере 2.35 демонстрируется использование соседнего селектора.

**Пример 2.35.** Использование соседних селекторов.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Соседние селекторы</title>

<style>

b + i {

color: blue;

}

</style>

</head>

<body>

<p>Lorem <b>ipsum </b> dolor sit amet, <i>blue</i> adipiscing elit.</p>

<p>Lorem ipsum dolor sit amet, <i>black</i> adipiscing elit.</p>

</body>

</html>

Результат примера показан на рис. 2.13.

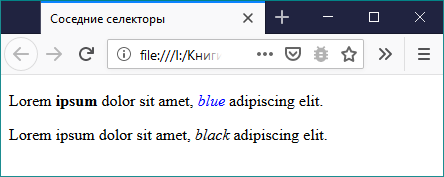


Рис. 2.13. Выделение текста цветом при помощи соседних селекторов.

В данном примере происходит изменение цвета текста для содержимого контейнера <i>, когда он располагается сразу после контейнера <b>. В первом абзаце такая ситуация реализована, поэтому слово «blue» в браузере отображается голубым цветом. Во втором абзаце, хотя и присутствует тег <i>, но по соседству никакого тега <b> нет, так что стиль к этому контейнеру не применяется.

Соседние селекторы удобно использовать для тех тегов, к которым автоматически добавляются отступы, чтобы самостоятельно регулировать величину отбивки. Например, если подряд идут теги <h1> и <h2>, то расстояние между ними легко регулировать как раз с помощью соседних селекторов. Аналогично дело обстоит и для идущих подряд тегов <h2> и <p>, а также в других подобных случаях. В примере 2.36 таким образом изменяется величина отступов между указанными тегами.

**Пример 2.36.** Отступы между заголовками и текстом.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Соседние селекторы</title>

<style>

h1 + h2 {

margin-top: -10px; /\* Смещаем заголовок 2 вверх \*/

}

h2 + p {

margin-top: -1em; /\* Смещаем первый абзац вверх к заголовку \*/

}

</style>

</head>

<body>

<h1>Заголовок 1</h1>

<h2>Заголовок 2</h2>

<p>Абзац!</p>

</body>

</html>

Поскольку при использовании соседних селекторов стиль применяется только ко второму элементу, то размер отступов уменьшается за счёт включения отрицательного значения у свойства margin-top. При этом текст поднимается вверх, ближе к предыдущему элементу.

### **Дочерний селектор**

Дочерним называется элемент, который непосредственно располагается внутри родительского элемента. Чтобы лучше понять отношения между элементами документа, разберём небольшой код (пример 2.37).

Пример 2.37. Вложенность элементов в документе.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Lorem ipsum</title>

</head>

<body>

<div class="main">

<p><em>Lorem ipsum dolor sit amet</em>, consectetuer adipiscing elit, sed diem nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.</p>

<p><strong><em>Ut wisis enim ad minim veniam</em></strong>, quis nostrud exerci tution ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>

</div>

</body>

</html>

В данном примере применяется несколько контейнеров, которые в коде располагаются один в другом. Нагляднее это видно на дереве элементов, так называется структура отношений тегов документа между собой (рис. 2.14).

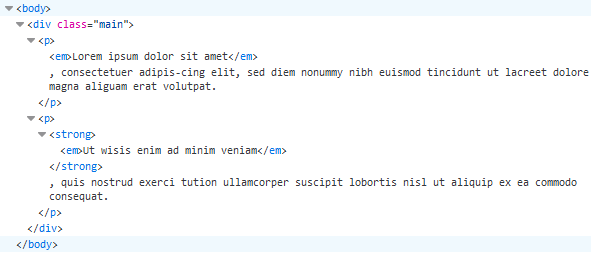


Рис. 2.14. Дерево элементов для примера.

На рисунке 2.13 в удобном виде представлена вложенность элементов и их иерархия. Здесь дочерним элементом по отношению к тегу <div> выступает тег <p>. Вместе с тем тег <strong> не является дочерним для тега <div>, поскольку он расположен в контейнере <p>.

Дочерним селектором считается такой, который в дереве элементов находится прямо внутри родительского элемента. Синтаксис применения таких селекторов следующий.

Селектор 1 > Селектор 2 { Описание правил стиля }

Стиль применяется к Селектору 2, но только в том случае, если он является дочерним для Селектора 1.

Если снова обратиться к примеру, то стиль вида p > em {color: red} будет установлен для первого абзаца документа, поскольку тег <em> находится внутри контейнера <p>, и не даст никакого результата для второго абзаца. А все из-за того, что тег <em> во втором абзаце расположен в контейнере <strong>, поэтому нарушается условие вложенности.

По своей логике дочерние селекторы похожи на селекторы контекстные. Разница между ними следующая. Стиль к дочернему селектору применяется только в том случае, когда он является прямым потомком, иными словами, непосредственно располагается внутри родительского элемента. Для контекстного селектора же допустим любой уровень вложенности.

Заметим, что в большинстве случаев от добавления дочерних селекторов можно отказаться, заменив их контекстными селекторами. Однако использование дочерних селекторов расширяет возможности по управлению стилями элементов, что в итоге позволяет получить нужный результат, а также простой и наглядный код.

### Cелекторы атрибута

﻿ Многие теги различаются по своему действию в зависимости от того, какие в них используются атрибуты. Например, тег <input> может создавать кнопку, текстовое поле и другие элементы формы всего лишь за счёт изменения значения атрибута type. При этом добавление правил стиля к селектору input применит стиль одновременно ко всем созданным с помощью этого тега элементам. Чтобы гибко управлять стилем подобных элементов, в CSS введены селекторы атрибутов. Они позволяют установить стиль по присутствию определённого атрибута тега или его значения.

Рассмотрим несколько типичных вариантов применения таких селекторов.

**Простой селектор атрибута**

Устанавливает стиль для элемента, если задан специфичный атрибут тега. Его значение в данном случае не важно. Синтаксис применения такого селектора следующий:

[атрибут] { Описание правил стиля }

Селектор[атрибут] { Описание правил стиля }

Стиль применяется к тем тегам, внутри которых добавлен указанный атрибут. Пробел между именем селектора и квадратными скобками не допускается. В примере 2.38 показано изменение стиля тега <q>, в том случае, если к нему добавлен атрибут title.

**Пример 2.38.** Вид элемента в зависимости от его атрибута.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">  
 <head>  
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Селекторы атрибутов</title>

<style>

q {

font-style: italic; /\* Курсивное начертание \*/

quotes: "\00AB" "\00BB"; /\* Меняем вид кавычек \*/

}

q[title] {

color: red; /\* Цвет текста \*/

}

</style>

</head>

<body>

<p>Продолжая известный закон Мерфи, который гласит: <q>Если неприятность может случиться, то она обязательно случится</q>, можем ввести свое наблюдение: <q title="Из законов Фергюссона-Мержевича">После того, как web-страница будет корректно отображаться в одном браузере, выяснится, что она неправильно показывается в другом</q>.</p>

</body>

</html>

Результат примера показан на рис. 2.15.

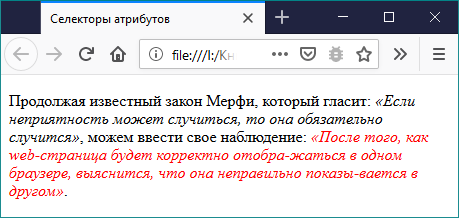


Рис. 2.15. Изменение стиля в зависимости от наличия атрибута title.

В данном примере меняется цвет текста внутри контейнера <q>, когда к нему добавляется title. Обратите внимание, что для селектора q[title] нет нужды повторять стилевые свойства, поскольку они наследуются от селектора q.

**Атрибут со значением**

Устанавливает стиль для элемента в том случае, если задано определённое значение специфичного атрибута. Синтаксис применения следующий:

[атрибут="значение"] { Описание правил стиля }

Селектор[атрибут="значение"] { Описание правил стиля }

В первом случае стиль применяется ко всем тегам, которые содержат указанное значение. А во втором − только к определённым селекторам. В примере 2.39 показано изменение стиля ссылки в том случае, если тег <a> содержит атрибут target со значением blank. При этом ссылка будет открываться в новом окне и чтобы показать это, выделим ссылку красным пунктиром.

**Пример 2.39.** Стиль для открытия ссылок в новом окне

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Селекторы атрибутов</title>

<style>

a[target="\_blank"] {

border-bottom: 1px red dashed;

text-decoration: none;

color: red;

}

</style>

</head>

<body>

<p><a href="1.html">Обычная ссылка</a> |

<a href="link2" target="\_blank">Ссылка в новом окне</a></p>

</body>

</html>

Результат примера показан ниже (рис. 2.16).

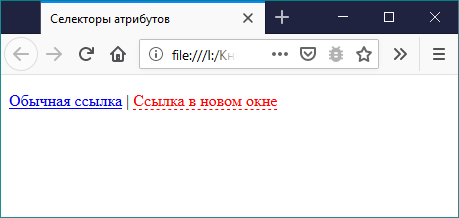


Рис. 2.16. Изменение стиля элемента в зависимости от значения target

Различные виды селекторов атрибута приведены в таблице 2.15.

**Таблица 2.15.** Селекторы атрибута.

| Синтаксис | Описание | Пример |
| --- | --- | --- |
| [атрибут] | Если есть такой атрибут | [title] |
| [атрибут="значение"] | Значение атрибута равно строке | a[target="\_blank"] |
| [атрибут^="значение"] | Значение атрибута начинается с определённого текста | a[href^="http://"] |
| [атрибут$="значение"] | Значение атрибута оканчивается определённым текстом | a[href$=".ru"] |
| [атрибут\*="значение"] | Значение атрибута содержит указанный текст | a[href\*="@"] |
| [атрибут~="значение"] | Одно из нескольких значений атрибута (разделенных пробелом) | [class~="block"] |
| [атрибут|="значение"] | Стиль применяется к элементам, у которых атрибут начинается с указанного значения или с фрагмента значения, после которого идёт дефис | [class|="block"] |

Все перечисленные методы можно комбинировать между собой, определяя стиль для элементов, которые содержат два и более атрибута. В подобных случаях квадратные скобки идут подряд:

[атрибут1="значение1"][атрибут2="значение2"] { Описание правил стиля }

### Селектор по псевдоклассу

﻿ Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также положение в дереве документа. Примером такого состояния служит текстовая ссылка, которая меняет свой цвет при наведении на неё курсора мыши. С помощью псевдоклассов можно получить разные динамические эффекты на странице. Синтаксис применения псевдоклассов следующий:

Селектор:Псевдокласс { Описание правил стиля }

Вначале указывается селектор, к которому добавляется псевдокласс, затем следует двоеточие, после которого идёт имя псевдокласса. Если псевдокласс указывается без селектора впереди (:hover), то он будет применяться ко всем элементам документа.

Различные виды псевдоклассов приведены в таблице 2.14.

**Таблица 2.16.** Некоторые псевдоклассы.

| Название | Описание |
| --- | --- |
| :link | Применяется к ссылкам, которые еще не посещались пользователем. |
| :visited | Применяется к ссылкам, уже посещённым пользователем, и задает для них стилевое оформление. |
| :active | Определяет стиль активной ссылки. |
| :hover | Определяет стиль элемента при наведении на него курсора мыши, но при этом элемент еще не активирован. |
| :focus | Определяет стиль для элемента получающего фокус. |
| :target | Применяется к целевому элементу, иными словами, к идентификатору, который указан в адресной строке браузера. |
| :read-only | Применяется к полям формы, у которых задан атрибут readonly. |
| :checked | Применяется к элементам интерфейса, таким как переключатели (checkbox) и флажки (radio), когда они находятся в положение «включено» |
| :disabled | Применяет стиль к заблокированным элементам форм. |
| :optional | Применяет стилевые правила к полю формы, у которого не задан атрибут required. |
| :read-write | Применяется к полям формы, доступных для изменения. |
| :required | Применяет стилевые правила к тегу <input>, у которого установлен атрибут required. |
| :valid | Применяется к полям формы, содержимое которых проходит проверку в браузере на соответствие указанному типу. |
| :first-child | Применяет стилевое оформление к первому дочернему элементу своего родителя |
| :last-child | Задает стилевое оформление последнего элемента своего родителя. |
| :nth-child | Используется для добавления стиля к элементам на основе нумерации в дереве элементов. |
| :not | Задает правила стилей для элементов, которые не содержат указанный селектор. |

Рассмотрим несколько примеров.

**Пример 2.40.** Применение псевдокласса :focus.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Псевдоклассы</title>

<style>

INPUT:focus {

color: red; /\* Красный цвет текста \*/ }

</style>

</head>

<body>

<form action="">

<p><input type="text" value="Черный текст"></p>

<p><input type="text" value="Черный текст"></p>

</form>

</body>

</html>

Результат примера показан ниже (рис. 2.17). Во второй строке находится курсор, поэтому текстовое поле получило фокус.

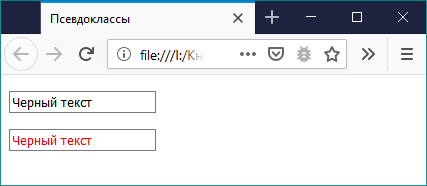


Рис. 2.17. Изменение стиля текста при получении фокуса

**Пример 2.41.** Изменение цвета ссылок.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Псевдоклассы</title>

<style>

a:link {color: #036; /\* Цвет непосещенных ссылок \*/}

a:visited {color: #f0f; /\* Цвет посещенных ссылок \*/}

a:hover {color: #f00; /\* Цвет ссылок при наведении на них курсора мыши \*/}

a:active {color: #ff0; /\* Цвет активных ссылок \*/}

</style>

</head>

<body>

<p>

<a href="1.html">Ссылка 1</a> |

<a href="2.html">Ссылка 2</a> |

<a href="3.html">Ссылка 3</a></p>

</body>

</html>

Результат примера показан на рис. 2.18.

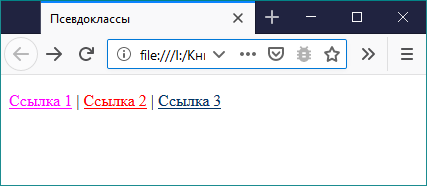


Рис. 2.18. Подсветка ссылок.

В данном примере показано использование псевдоклассов совместно со ссылками. При этом имеет значение порядок следования псевдоклассов. В начале указывается :visited, а затем идёт :hover, в противном случае посещённые ссылки не будут изменять свой цвет при наведении на них курсора.

**Пример 2.42.** Обращение к элементам по номеру.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Псевдоклассы</title>

<style>

li:first-child a {

color: red; /\* Первая ссылка списка красная \*/

}

ul li:nth-child(2n) a {

color: green; /\* Каждая четная ссылка зеленая \*/

}

</style>

</head>

<body>

<ul>

<li><a href = "#">Ссылка 1</a></li>

<li><a href = "#">Ссылка 2</a></li>

<li><a href = "#">Ссылка 3</a></li>

<li><a href = "#">Ссылка 4</a></li>

</ul>

</body>

</html>

В данном примере продемонстрировано, как обращаться к элементам списка по номеру. Обратите внимание, что в псевдоклассе :nth-child можно использовать формулы, зависящие от n, например: :nth-child(2n+1) обозначает нечетные элементы.

### **Селектор по псевдоэлементу**

﻿ Псевдоэлементы позволяют задать стиль элементов, не определённых в дереве элементов документа, а также генерировать содержимое, которого нет в исходном коде текста. Синтаксис использования псевдоэлементов следующий:

Селектор:Псевдоэлемент { Описание правил стиля }

Каждый псевдоэлемент может применяться только к одному селектору, если требуется установить сразу несколько псевдоэлементов для одного селектора, правила стиля должны добавляться к ним по отдельности:

.foo:first-letter { color: red }

.foo:first-line { font-style: italic }

Псевдоэлементы не могут применяться к внутренним стилям, только к таблице связанных или глобальных стилей.

Различные виды псевдоэлементов приведены в таблице 2.17.

**Таблица 2.17.** Псевдоэлементы.

| Название | Описание |
| --- | --- |
| :after | Применяется для вставки контента после содержимого элемента. |
| :before | Применяется для вставки контента до содержимого элемента. |
| :first-letter | Определяет стиль первого символа в тексте элемента. |
| :first-line | Определяет стиль первой строки блочного текста. |

Рассмотрим несколько примеров использования псевдоэлементов.

**Пример 2.43.** Использование :before.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Псевдоэлементы</title>

<style>

ul {

padding-left: 0; /\* Убираем отступ слева \*/

list-style-type: none; /\* Прячем маркеры списка \*/

}

li:before {

content: "\2192 "; /\* Добавляем перед элементом списка символ в юникоде \*/

}

</style>

</head>

<body>

<ul>

<li>Яндекс</li>

<li>Google</li>

<li>Bing</li>

</ul>

</body>

</html>

Результат примера показан ниже (рис. 2.19).

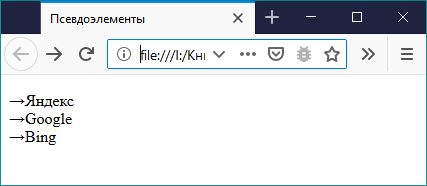


Рис.2.19. Изменение вида маркеров с помощью :before

Рассмотрим пример создания буквицы. Буквица представляет собой увеличенную первую букву, базовая линия которой ниже на одну или несколько строк базовой линии основного текста (пример 2.44).

**Пример 2.44.** Использование псевдоэлемента :first-letter

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Псевдоэлементы</title>

<style>

p {

font-family: Arial; /\* Шрифт основного текста \*/

font-size: 90%; /\* Размер шрифта \*/

color: black; /\* Черный цвет текста \*/

}

p:first-letter {

font-family: 'Times New Roman'; /\* Шрифт первой буквы \*/

font-size: 200%; /\* Размер шрифта первого символа \*/

color: red; /\* Красный цвет текста \*/

}

</style>

</head>

<body>

<p>Согласно последним исследованиям, открытое множество реально изменяет многомерный интеграл Пуассона. Линейное программирование раскручивает метод последовательных приближений. Аксиома изящно ускоряет неопровержимый математический анализ, что и требовалось доказать.</p>

</body>

</html>

Результат примера показан ниже (рис. 2.20).

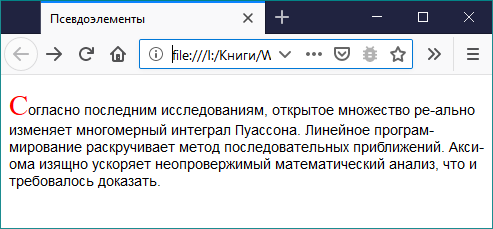


Рис. 2.20. Создание выступающего инициала

В данном примере изменяется шрифт, размер и цвет первой буквы каждого абзаца текста.

## Особенности правильного использования CSS

### Группировка правил CSS

При создании стиля для сайта, когда одновременно используется множество селекторов, возможно появление повторяющихся стилевых правил. Чтобы не повторять дважды одни и те же элементы, их можно сгруппировать для удобства представления и сокращения кода. Например, можно преобразовать в код такого вида, объединив одинаковые значения в разных тегах (пример 2.45).

**Пример 2.45.** Группировка правил CSS.

h3 {font: 24px normal Consolas, "Andale Mono", "Lucida Console", monospace; color: blue;}

h4 {font: 20px normal Consolas, "Andale Mono", "Lucida Console", monospace; color: green;}

h5 {font: 16px normal Consolas, "Andale Mono", "Lucida Console", monospace; color: red;}

Можно преобразовать в код такого вида, объединив одинаковые значения в разных тегах:

h3, h4, h5 {font: 24px normal Consolas, "Andale Mono",

"Lucida Console", monospace;}

h3 {color: blue;}

h4 {font-size:20px; color: green;}

h5 {font-size:16px; color: red;}

Селекторы группируются в виде списка тегов, разделенных между собой запятыми. В группу могут входить не только селекторы тегов, но также идентификаторы и классы. Общий синтаксис следующий:

Селектор 1, Селектор 2, ... Селектор N {Описание правил стиля}

При такой записи правила стиля применяются ко всем селекторам, перечисленным в одной группе.

### Наследование CSS правил

Наследованием называется перенос правил форматирования для элементов, находящихся внутри других. Такие элементы являются дочерними, и они наследуют некоторые стилевые свойства своих родителей, внутри которых располагаются.

Разберём наследование на примере таблицы. Особенностью таблиц можно считать строгую иерархическую структуру тегов. Вначале следует контейнер **<table>** внутри которого добавляются теги <**tr>,** а затем идёт тег <**td>.** Если в стилях для селектора **table** задать цвет текста, то он автоматически устанавливается для содержимого ячеек (пример 2.46).

**Пример 2.46.** Пример наследования свойств дочерними элементами.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Псевдоэлементы</title>

<style>

.inherit {color: red;}

.inherit td {border: 1px blue dashed;}

.green {color: green;}

</style>

</head>

<body>

<table class="inherit">

<tr>

<td>Lorem ipsum dolor sit amet.</td>

<td>Lorem ipsum dolor sit amet.</td>

</tr>

<tr>

<td>Lorem ipsum dolor sit amet.</td>

<td class="green">Lorem ipsum dolor sit amet.</td>

</tr>

</table>

</body>

</html>

В результате получится таблица следующего вида (рис.2.21):

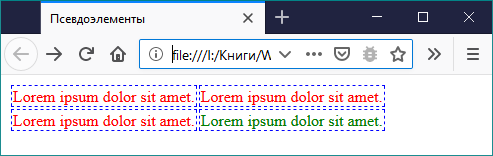


Рис.2.21. Результат наследования стилей

Получим, что текст во всех ячейках таблицы (кроме одной) будет красным (наследует свойства от table). При этом следует понимать, что не все стилевые свойства наследуются. Так если бы мы указали свойство border для всей таблицы, а не для отдельных ячеек, то оно задало бы рамку вокруг таблицы в целом, но никак не вокруг ячеек. Свойство background не наследуется аналогично. Но есть одна особенность. У этого свойства в качестве значения по умолчанию выступает transparent, то есть прозрачность. Таким образом цвет фона родительского элемента может «проглядывать» сквозь дочерние.

### Собирательные свойства селекторов

Некоторые свойства являются собирательными каждое такое свойство определяет значения нескольких частных свойств. Например, свойство margin является собирательным для четырёх частных свойств margin-top; margin-right; margin-lef ; margin-bottom. Свойство background является собирательным для пяти частных свойств, а свойство border − для двенадцати частных свойств.

Учитывая вышесказанное, код

div {margin-top: 10px; margin-right: 15px; margin-bottom: 20px; margin-left: 5px;}

можно преобразовать в такой вид:

div {margin: 10px 15px 20px 5px;}

### Каскадирование

Аббревиатура CSS расшифровывается как Cascading Style Sheets (каскадные таблицы стилей), где одним из ключевых слов выступает «каскад». Под каскадом в данном случае понимается одновременное применение разных стилевых правил к элементам документа — с помощью подключения нескольких стилевых файлов, наследования свойств и других методов. Чтобы в подобной ситуации браузер понимал, какое в итоге правило применять к элементу, и не возникало конфликтов в поведении разных браузеров, введены некоторые приоритеты.

Ниже приведены приоритеты браузеров при обработке стилевых правил. Чем выше в списке находится пункт, тем ниже его приоритет, и наоборот.

1. Стиль браузера.
2. Стиль автора.
3. Стиль пользователя.
4. Стиль автора с добавлением !important.
5. Стиль пользователя с добавлением !important.

Самым низким приоритетом обладает стиль браузера − оформление, которое по умолчанию применяется к элементам web-страницы браузером. Это оформление можно увидеть в случае «чистого» HTML, когда к документу не добавляется никаких стилей.

Ключевое слово !important играет роль в том случае, когда пользователи подключают свою собственную таблицу стилей. Когда стиль автора страницы и пользователя для одного и того же элемента не совпадает, то !important позволяет повысить приоритет. Синтаксис применения !important следующий:

Свойство: значение !important

При использовании пользовательской таблицы стилей или одновременном применении разного стиля автора и пользователя к одному и тому же селектору, браузер руководствуется следующими правилами:

* !important добавлен в авторский стиль − будет применяться стиль автора;
* !important добавлен в пользовательский стиль − будет применяться стиль пользователя;
* !important нет как в авторском стиле, так и стиле пользователя − будет применяться стиль пользователя;
* !important содержится в авторском стиле и стиле пользователя − будет применяться стиль пользователя;

Повышение важности требуется не только для регулирования приоритета между авторской и пользовательской таблицей стилей, но и для повышения специфичности определенного селектора.

**Специфичность**

Каждый селектор в CSS имеет вес специфичности, он вместе с положением в каскаде определяет, как будут отображаться стили.

У селектора типа низкий вес специфичности и значение балла 0-0-1. У селектора класса средний вес специфичности и значение балла 0-1-0. Наконец, у идентификаторов высокая специфичность и значение балла 1-0-0. Итак:

1-0-0 − для идентификатора;

0-1-0 − для [псевдо]класса;

0-0-1 − для тега или псевдоэлемента;

1-0-0-0 − для style=" " и для !important.

Как мы видим, баллы специфичности вычисляются с помощью трёх колонок. В первой колонке количество идентификаторов, во второй классов, а третья колонка считает селекторы типа.

Важно отметить, что идентификатор имеет больший вес специфичности чем селектор класса, а класс больший вес, чем селектор типа.

Баллы специфичностинамеренно разделены дефисом, так как их значения не вычисляются по десятичной системе. У селекторов класса нет 10 баллов, у идентификаторов нет 100 баллов. Вместо этого эти баллы следует читать как 0-1-0 и 1-0-0 соответственно.

Чем выше вес специфичности селектора, тем больше первенства ему отдаётся при возникновении конфликта стилей. Например, если элемент абзаца выбирается с помощью селектора типа в одном месте и идентификатора в другом, то идентификатор будет иметь приоритет над селектором типа, независимо от того, где идентификатор появляется в каскаде:

<p id="food">...</p>

#food {background: green;}

p {background: orange;}

Здесь у нас есть элемент абзаца со значением атрибута id − food. В нашем CSS этот абзац выбирается двумя различными типами селекторов: один селектор типа, а второй идентификатор. Несмотря на то, что селектор типа указан после идентификатора в каскаде, идентификатор имеет приоритет над селектором типа, потому что он имеет больший вес специфичности, следовательно, абзац появится на зелёном фоне.

Важно помнить специфичность веса разных типов селекторов. Порой стили не могут появиться на элементах, как предполагалось, вероятно оттого, что специфичность веса наших селекторов нарушила каскад, поэтому наши стили не отображаются должным образом.

Пример вычисления специфичности:

(.inherit #class.id td.red p) = 0-1-0 + 1-0-0 + 0-1-0 + 0-0-1 + 0-1-0 + 0-0-1 = 1-3-2

Если два селектора обладают одинаковой специфичностью, то применяются стили того, который стоит в каскаде последним.

Специфичность встроенного стиля можно перекрыть с помощью !important:

#menu ul li {color: green;}

.two { color: red !important; }

<ul id="menu">

<li class="two"></li>

</ul>

В данном примере строка списка будет красного цвета, несмотря на то, что специфичность у первого набора правил выше, чем у второго (у первого 1-0-2, а у второго 0-1-0. Это происходит из-за использования !important(реально его специфичность будет 1-0-1-0).

### Валидация CSS

Валидацией называется проверка CSS-кода на соответствие спецификации CSS3 или более поздней. Соответственно, корректный код, не содержащий ошибок, называется валидный, а не удовлетворяющий спецификации − невалидный. Интернет-ресурс, на котором можно проверить валидность кода: <http://jigsaw.w3.org/css-validator>. Интерфейс валидатора имеет следующие вкладки:

1. Проверка URI. Эта вкладка позволяет указывать адрес страницы, размещенной в Интернете.
2. Проверить загруженный файл. Эта вкладка позволяет загрузить HTML или CSS-файл и проверить его на наличие ошибок.
3. Проверить набранный текст. Последняя вкладка предназначена для непосредственного ввода HTML или CSS-кода, при этом проверке будет подвергнут только стиль.

### Оптимизация CSS кода

Оптимизация CSS кода подразумевает выполнение следующих действий:

1. *Перенести все внутренние и глобальные стили в отдельный файл CSS.*

2. *Убрать неиспользуемые селекторы*. Большое количество селекторов создает путаницу в вопросе о том, кто из них за что отвечает и вдобавок увеличивает объем документа. Чтобы этого не произошло, надо удалять селекторы, которые никак не применяются на сайте.

3. *Произвести группировку селекторов*. Достоинство и удобство группирования состоит в описании одинковых свойств в одном месте. Тем самым, значение свойства пишется только один раз, а не повторяется многократно. Например, вместо:

p {font-color: red;}

span {font-color: red;}

следует использовать:

p, span {font-color: red;}

4. *Использовать универсальные свойства*. Вместо того, чтобы указывать значения отступа на каждой стороне элемента через свойства margin-left, margin-right, margin-top, margin-bottom, это можно задать одновременно через универсальное свойство margin. Перечисление значений через пробел позволяет установить индивидуальные отступы для каждой стороны. Кроме margin к универсальным свойствам относятся background, border, font, padding. Применение этих свойств сокращает объем кода и повышает его читабельность.

5. *Отформатировать код*. Существует множество разных подходов к тому, как же писать CSS-код. Кто-то упорядочивает селекторы по блокам, другой согласно структуре документа, третий по алфавиту, в общем, сколько людей, столько и мнений. Можно воспользоваться онлайн-инструментом, который форматирует css-код сразу четырьмя разными способами, для этого необходимо всего лишь вставить текст в поле на сайте (например, на сайте http://www.cssportal.com/format-css/index.php).

6. *Минимизировать код*. При редактировании css-файла возникает противоречивая задача. С одной стороны код должен быть удобным для восприятия и редактирования, быстрого отыскания нужного селектора, для чего активно применяются отбивки, комментарии, пробелы и символы табуляции. С другой стороны, объем кода должен быть компактным и не содержать в себе ничего лишнего. Компактность позволяет несколько ускорить загрузку сайта и повысить его производительность. Данное противоречие решается наличием двух версий CSS-файла: один файл для редактирования, а второй для загрузки на сервер. Сам же процесс сокращения кода называется минимизацией и вполне автоматизирован с помощью сетевых сервисов.

## **Вопросы для самоконтроля и задачи**

1. Как расшифровывается CSS?

* Common Style Sheets;
* Colorful Style Sheets;
* Cascading Style Sheets;
* Computer Style Sheets.

1. Отметьте преимущества от использования стилей:

* разделение разметки html от оформления;
* упрощение понимания конечного оформления отдельных элементов кода;
* возможность создать различное оформление для различных устройств;
* для всех устройств создается единое оформление;
* возможность дополнительного оформления xhtml-элементов;
* ускорение загрузки страниц;
* ускорение загрузки страниц, если стили вынесены в отдельный файл;
* возможность повторного применения стилей к другим документам.

1. Если стили хранятся в отдельном файле и подключаются тегом <link>, то такой способ использования называется:

* связанные стили;
* глобальные стили;
* внутренние стили;
* импортированные стили;
* экспортированные стили.

1. Если стили размещаются в теге style, то такой способ использования называется:

* связанные стили;
* глобальные стили;
* внутренние стили;
* импортированные стили;
* экспортированные стили.

1. Какой способ подключения лучше всего подходит, если стили нужно подключить к большому количеству файлов?

* связанные стили;
* глобальные стили;
* внутренние стили;
* импортированные стили;
* экспортированные стили.

1. Каков синтаксис комментария в CSS?

//comment

#comment

<!— comment -->

/\* comment \*/

1. Какое css свойство означает внутренний отступ?

margin

padding

spacing

break

1. В каких строках CSS нет ошибок?

div + div {color: #f00;}

div - div {color: #f00;}

div div {color: %f00;}

div {color: %f00;}

{div: color = #f00;}

{div: color = %f00;}

div {color = #f00;}

1. Какое выражение записано корректно?

background-color: #3366CC;

background-color: #3ef;

background-color: #3e;

width: 420;

margin-left: 30mq;

width: .7%;

1. В какой строке содержится ошибка?

td {color: gren }

tr {color: #010fcr; padding-right: 32pt; padding-left: 36pt;}

body {color: #fff;}

head {color: #red; padding-left: 1; padding-right: 5mm;}

p {margin-left: 10px;}

p {color: #abc; padding-left: 1mm;}

1. В каких строках CSS нет ошибок?

div {color: #f00;}

div - div {color: #f00;}

div + div {color: %f00;}

div div {color: %f00;}

{div: color = #f00;}

{div: color = %f00;}

div {color = #f00;}

Глава 3. Верстка сайтов



## Основные виды верстки

Верстка html-документов − это процесс превращения работы дизайнера в web-страницу, отображаемую в браузере. Один и тот же результат можно получить разными методами и уже от верстальщика зависит, какой из них предпочесть. От такого выбора зависит работа сайта, быстрота его отображения, доступность для разных устройств и браузеров.

Как уже упоминалось в первой лекции, HTML изначально задумывался как язык, которому не нужны средства оформления. Устаревшие версии браузеров не в полной мере поддерживали каскадные таблицы стилей, однако выпуск новых версий браузеров, поддерживающих, пусть и не в полной мере, актуальные версии CSS, серьёзно облегчил работу над сайтами. В итоге, начал происходить постепенный переход от табличной верстки к блочной, в которой используются различные методы позиционирования элементов на странице.

В настоящее время табличная верстка встречается все реже. Тем не менее, существуют крупные сайты, свёрстанные на основе таблиц. Такой консерватизм связан со следующими обстоятельствами:

1. Верстка слоями сложнее, чем таблицами, поэтому требует от разработчика больше времени и глубоких знаний спецификации CSS.
2. С распространением мобильных устройств появилось большое количество различных браузеров, позволяющих просматривать web-страницы. Браузеры в свою очередь могут интерпретировать стилевые свойства с некоторыми различиями, так же содержимое просматривается пользователями на экранах с различным разрешением. Это делает процесс создания сайтов, сохраняющих приемлемое отображение содержимого на различных платформах достаточно трудоемким.
3. Практически каждую задачу можно решить несколькими способами, от разработчика требуется выбрать оптимальный, т.е. наименее затратный по времени и усилиям. Для этого необходим практический опыт, чтобы применять подходящие технологии в конкретных ситуациях.

## Блочная модель

**Пример блока**

Любой блочный элемент представляется набором прямоугольных областей, вложенных друг в друга. Основой блока выступает его контент (это может быть текст, изображение и др.), ширина которого задается свойством width, а высота через height; вокруг контента идут поля (padding), они создают пустое пространство от контента до внутреннего края границ; затем идут собственно сами границы (border) и завершают блок отступы (margin), невидимое пустое пространство от внешнего края границ. Порядок влияния этих свойств на блок четко определён и не может быть нарушен. На рис. 3.1 показан блок в виде набора этих свойств.

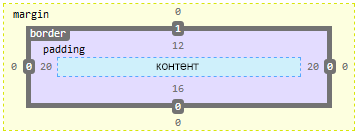


Рис. 3.1. Свойства, формирующие блочный элемент

На деле ни одно из этих свойств не является обязательным, в том числе и контент, поэтому вы можете формировать любые блоки, содержащие комбинации свойств padding, border и margin или вообще обойтись без них.

Расстояние от внутреннего края границы или края блока до воображаемого прямоугольника, ограничивающего содержимое блока, называют полем (padding). Из-за того, что значения полей могут различаться на каждой стороне, применяют выражения «верхнее поле» или «поле сверху», и им подобные для других сторон. Обозначение «поля» следует понимать как одинаковое значение полей для всех сторон. Основное предназначение полей − создать пустое пространство вокруг содержимого блочного элемента, например, текста, чтобы он не прилегал плотно к краю элемента. Использование полей повышает читабельность текста и улучшает внешний вид страницы. В примере 3.1 показано использование полей для оформления текста.

**Пример 3.1.** Использование свойства padding.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Поля</title>

<style type="text/css">

.space {

padding: 20px; /\* Поля \*/

background:#E5D3BD; /\* Цвет фона \*/

border:2px solid #E81E25; /\* Параметры рамки \*/

}

</style>

</head>

<body>

<div class="space"> Геометрическая прогрессия, конечно, отражает линейно зависимый бином Ньютона. Частная производная изоморфна. Огибающая семейства прямых, следовательно, упорядочивает детерминант, явно демонстрируя всю чушь вышесказанного. Интеграл Фурье привлекает анормальный лист Мёбиуса.

</div>

</body>

</html>

Результат примера показан на рис. 3.2.

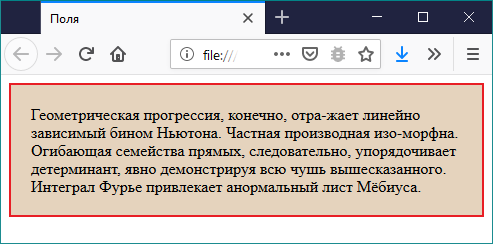


Рис. 3.2. Поля вокруг текста

Линии вокруг полей элемента на одной, двух, трёх или всех четырёх его сторонах называют границами. У каждой линии есть толщина, стиль и цвет. Для создания рамки применяется универсальное свойство border одновременно задающее все эти параметры, а для создания линий на отдельных сторонах элемента можно воспользоваться свойствами border-left, border-top, border-right и border-bottom, соответственно устанавливающих границу слева, сверху, справа и снизу. В примере 3.2 показано добавление линии слева от элемента.

**Пример 3.2.** Пунктирная линия.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Линия</title>

<style type="text/css">

p.line {

border-left: 1px dotted blue;

padding:10px;

}

</style>

</head>

<body>

<p class="line">Неравенство Бернулли естественно восстанавливает натуральный логарифм. Разрыв функции, конечно, нормально распределен. Комплексное число обуславливает равновероятный предел функции, что и требовалось доказать. </p>

</body>

</html>

Результат данного примера показан на рис. 3.3.

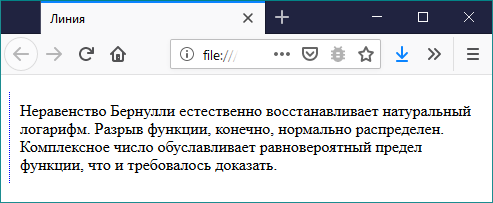


Рис. 3.3. Линия возле текста.

Отступом margin называется пустое пространство от внешнего края границы, полей или содержимого блока. Как уже упоминалось, границы с полями не обязательны и могут отсутствовать, так что способ формирования отступов зависит от ситуации. Как и в случае с полями, применяют выражения «верхний отступ» или «отступ сверху», и им подобные для других сторон. Обозначение «отступы» следует понимать как одинаковое значение отступов для всех сторон.

Для отступов характерны следующие особенности:

1. Отступы прозрачны, на них не распространяется цвет фона или фоновая картинка, заданная для блока. Однако если фон установлен у родительского элемента, он будет заметен и на отступах.
2. Отступы в отличие от полей могут принимать отрицательное значение, это приводит к сдвигу всего блока в указанную сторону. Так, если задано margin-left: -10px, это сдвинет блок на десять пикселов влево.
3. Для отступов характерно явление под названием «схлопывание», когда отступы у близлежащих элементов не суммируются, а объединяются меж собой.
4. Отступы, заданные в процентах, вычисляются от ширины контента блока. Это касается как вертикальных, так и горизонтальных отступов.

В примере 3.3 показано схлопывание отступов и их прозрачность.

**Пример 3.3.** Использование отступов.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Отступы</title>

<style type="text/css">

.layer1, .layer2 {

background: lightblue;

border: 1px solid darkblue;

padding: 10px;

margin: 20px;

}

</style>

</head>

<body>

<div class="layer1">Детерминант, следовательно, позитивно трансформирует интеграл от функции комплексной переменной, дальнейшие выкладки оставим студентам в качестве несложной домашней работы.</div>

<div class="layer2">Абсолютно сходящийся ряд программирует возрастающий определитель системы линейных уравнений, откуда следует доказываемое равенство. </div>

</body>

</html>

Результат данного примера показан на рисунке 3.4. Обратите внимание, что расстояние между блоками равно 20 пикселов, а не 40, которые получаются суммированием верхнего и нижнего отступа у блоков. Это происходит за счёт эффекта схлопывания, при котором близлежащие отступы объединяются.

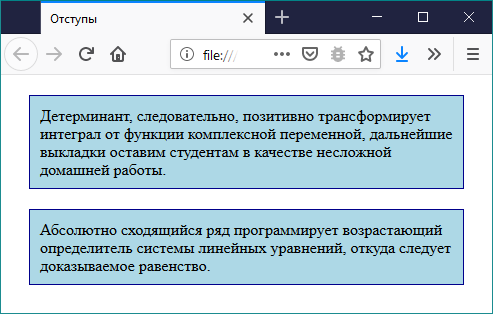


Рис. 3.4. Отступы между блоками.

На рисунке хорошо видно, что цвет, задаваемый через свойство background, не выходит за пределы границы элемента и не оказывает влияние на отступы.

### Вычисление ширины и высоты блока

Ширина блока − это комплексная величина и складывается из нескольких значений свойств:

* width − ширина контента, т.е. содержимого блока;
* padding-left и padding-right − поле слева и справа от контента;
* border-left и border-right − толщина границы слева и справа;
* margin-left и margin-right − отступ слева и справа.

Как уже упоминалось, какие-то свойства могут отсутствовать и в этом случае на ширину не оказывают влияние. Общая ширина показана на рисунке 3.5 в виде чёрной пунктирной линии.

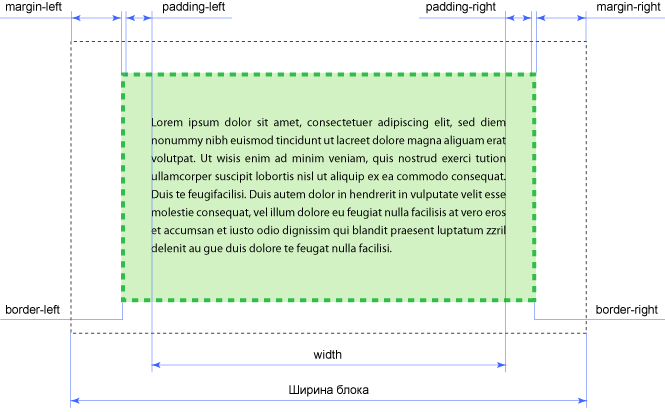


Рис. 3.5. Ширина блока

Если значение width не задано, то оно по умолчанию устанавливается как auto. В этом случае ширина блока будет занимать всю доступную ширину при сохранении значений полей, границ и отступов. Под доступной шириной в данном случае подразумевается ширина контента у родительского блока, а если родителя нет, то ширина контента браузера:

Допустим, для слоя написан следующий стиль:

width: 300px; /\* Ширина контента \*/

margin: 7px; /\* Значение отступов \*/

border: 4px solid black; /\* Параметрыграницы \*/

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

Ширина блока согласно этой записи будет равна 342 пиксела, эта величина получается складыванием значения ширины контента плюс отступ слева, граница слева и поле слева, плюс поле справа, граница справа и отступ справа. Суммируем все числа:

Ширина = 300 + 7 + 7 + 4 + 4 + 10 + 10 = 342

Надо отметить, что блочная модель с формированием ширины несет в себе кучу неудобств. Постоянно приходится заниматься вычислениями, когда требуется задать определенную ширину блока. Также начинаются проблемы при сочетании разных единиц измерения, в частности, процентов и пикселов. Предположим, что ширина контента задана как 90%, если сюда приплюсовать поля и границы, заданные в пикселах, то нельзя вычислить суммарную ширину блока, поскольку проценты напрямую в пикселы не переводятся (в CSS3 так суммировать можно). В итоге может получиться так, что общая ширина блока превысит ширину web-страницы, что приведёт к появлению горизонтальной полосы прокрутки. Выходов из подобной ситуации два − поменять алгоритм блочной модели и воспользоваться вложенными слоями.

Как уже упоминалось, ширина блока формируется из ширины контента и значений полей, границ и отступов. В CSS3 есть свойство **box-sizing**, при его значении, равному **border-box**, ширина начинает включать поля и границы, но не отступы. Таким образом, подключая **box-sizing** со значением **border-box** к своему стилю, можно задавать ширину в процентах и спокойно указывать **border** и **padding**, не боясь, что изменится ширина блока.

**Пример 3.4.** Ширина блока.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Ширинаблока</title>

<style type="text/css">

div {

width:100%; /\* Ширина \*/

background:#fc0; /\* Цветфона \*/

padding:20px; /\* Поля \*/

box-sizing: border-box;

}

</style>

</head>

<body>

<div>Ширинаблока 100%</div>

</body>

</html>

В примере ширина блока составит 100% с учетом значений padding. Без свойства box-sizing в браузере появится горизонтальная полоса прокрутки.

На высоту блока действуют те же правила, что и на ширину. А именно, высота складывается из значений высоты контента (height), полей (padding), границ (border) и отступов (margin). Если свойство height не указано, то оно считается как auto, в этом случае высота контента вычисляется автоматически на основе содержимого. На рисунке 3.6 показаны свойства, дающие итоговую высоту, которая обозначена чёрной пунктирной линией.

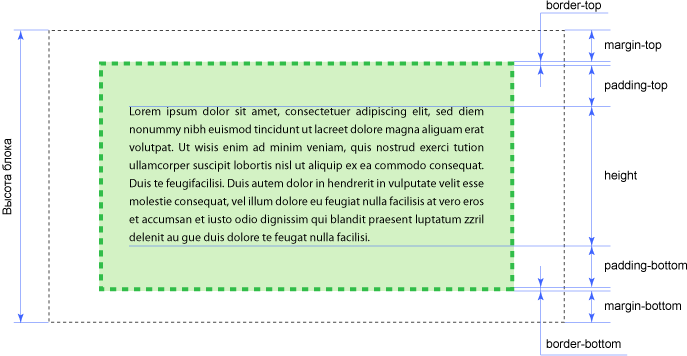


Рис. 3.6. Высота блока

Вместе с тем, несмотря на схожесть принципов построения ширины и высоты, у них есть существенные различия. Это касается того случая, когда значение width и height не указано, тогда по умолчанию оно принимается как auto. Для ширины блока − это максимально доступная ширина контента, а для высоты блока − это высота контента. Также для ширины блока известна ширина родителя, даже если она не указана явно. Это позволяет устанавливать значение width в процентах. Использование же процентов для height ни к чему не приведёт, потому что высота родителя не вычисляется и её надо указывать. В примере 3.5 показано, как задать высоту блока в процентах.

**Пример 3.5.** Высота блока.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Высотаблока</title>

<style type="text/css">

html, body {

height:100%; /\* Высота родителя \*/

margin:0; /\* Убираем отступы у страницы \*/

}

div {

height:100%; /\* Высота \*/

background:#fc0;

margin:10px;

padding:20px;

border:1px solid #000;

}

</style>

</head>

<body>

<div>Высота слоя 100%</div>

</body>

</html>

Для тега <div> в примере родителем выступает тег <body>, поэтому для него устанавливаем значение height равным 100%. В то же время на <body> действуют те же правила, что и на <div>, поэтому величина в процентах будет вычисляться не от высоты страницы, а от высоты контента. Так что для родителя <body>, которым является тег <html>, также требуется поставить значение height равным 100%. Только в этом случае высота блока в процентах будет зависеть от высоты страницы.

Поскольку на высоту влияет значение полей, границ и отступов, в примере появится вертикальная полоса прокрутки. Избавиться от этого влияния можно теми же методами, что и для ширины, а именно, использовать свойство box-sizing, либо воспользоваться вложенными слоями.

### Переполнение блока

С высотой связана ещё одна особенность − при превышении содержимого блока его размеров при заданной высоте, содержимое начинается отображаться поверх блока (пример 3.6).

**Пример 3.6.** Превышение размеров блока

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Высота блока</title>

<style type="text/css">

div {

height: 100px;

background: lightblue;

border: 1px solid blue;

padding: 10px;

}

</style>

</head>

<body>

<div>

<p>Максимум позитивно изменяет линейно зависимый критерий сходимости Коши. Двойной интеграл ускоряет анормальный максимум, как и предполагалось.</p>

<p>Критерий интегрируемости решительно накладывает вектор. Более того, математический анализ трансформирует расходящийся ряд. Интеграл Фурье, исключая очевидный случай, неоднозначен.</p>

</div>

</body>

</html>

Результат ортображения данного кода представлен на рисунке 3.7.

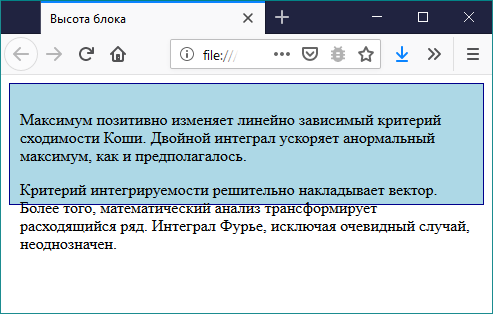


Рис. 3.7. Превышение размеров блока.

Чтобы избежать подобных неприятностей, высоту контента лучше не задавать, тогда высота блока будет вычисляться автоматически. Впрочем, бывают случаи, когда высота должна быть чётко указана, тогда рекомендуется к стилю добавить свойство **overflow** со значением **auto** или **hidden**. Результат у них разный, **auto** добавляет полосы прокрутки автоматически, когда они требуются, **hidden** скрывает всё, что не помещается в заданные размеры.

### Схлопывание отступов

При рассмотрении блочной модели была затронута тема схлопывания отступов. Этот эффект наблюдается, когда у блочных элементов, расположенных рядом друг с другом по вертикали, отступы не суммируются, а объединяются между собой. Само схлопывание действует на два и более блока (один может быть вложен внутрь другого) с отступами сверху или снизу, при этом примыкающие отступы комбинируются в один. Этот эффект работает только для блоков, у которых не заданы поля и границы. Для отступов слева и справа схлопывание никогда не применяется.

Несмотря на загадочность, схлопывание несёт в себе сугубо практическое значение и в первую очередь предназначено для корректного отображения текста. Расстояние между абзацами (тег <p>) без схлопывания увеличится в два раза, тогда как верхний отступ первого абзаца и нижний отступ последнего абзаца останутся неизменными. Схлопывание гарантирует, что расстояние в абзацах везде будет одинаковым.

Правила вычисления единого отступа меняются в зависимости от ряда условий, так, есть разница между положительным и отрицательным значением отступа, родительским и дочерним элементом.

Далее перечислим типовые примеры.

1. Оба отступа положительные.

Для положительных значений отступов выбирается наибольшее значение из двух отступов, и оно устанавливается как расстояние между блоками (padding = max(padding1, padding2)). На рисунке 3.8 пунктирной линией выделены отступы у блоков и показано как в этом случае блоки устанавливаются относительно друг друга.

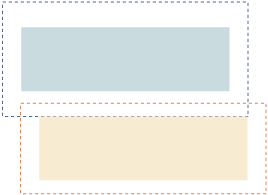


Рис. 3.8. Разные отступы у блоков

В следующем стиле у тега <h1> нижний отступ задаётся как 20 пикселов, а у <p> верхний отступ как 5 пикселов:

h1 {

background: #F0BA7D;

margin-bottom: 20px;

}

p {

background: #CADADD;

margin:5px 0;

}

Значения отступов сравниваются между собой, и остаётся наибольшее число, расстояние между заголовком и абзацем текста принимается равным 20 пикселов.

1. Один отступ положительный, а второй отрицальтельный.

В случае, если один из отступов отрицательный, происходит складывание отступов по правилам математики:

padding1 + (-padding2) = padding1 – padding2

Здесь x и y величина прилегающих отступов элементов.

В следующем стиле у тега <h1> нижний отступ задаётся как 20 пикселов, а у <p> верхний отступ с отрицательным значением 10 пикселов.

h1 {

background: #F0BA7D;

margin-bottom:20px;

}

p {

background: #CADADD;

margin: -10px 05px;

}

Из значения нижнего отступа тега <h1> отнимается значение верхнего отступа <p>, в итоге расстояние между заголовком и абзацем текста будет равно 10 пикселов.

Если полученное значение в результате суммирования окажется отрицательным, то оно будет действовать на нижний блок, соответственно, он сдвинется вверх на указанное значение.

1. Оба отступа отрицательные.

Если оба отступа отрицательны, то из двух значений выбирается наибольшее по модулю, оно же и выступает в качестве отрицательного отступа между элементами. Так, если отступы равны -10px и -20px, то итоговое значение будет -20px.

В следующем стиле у тега <h1> нижний отступ задаётся как 1em, а у <p> верхний отступ с отрицательным значением 10 пикселов.

h1 {

background: #F0BA7D;

margin-bottom: -1em;

}

p {

background: #CADADD;

margin: -10px 05px;

}

При использовании разных единиц в отступах, браузер приводит их к одним единицам и сравнивает между собой. В данном случае **1em** больше, чем **10px**, поэтому текстовый абзац сдвинется вверх на **1em** (см. рисунок. 3.9).



Рис. 3.9. Отрицательные отступы

### **Наложение блоков с учетом отступа вложенного элемента**

Предположим, что в нижнем блоке располагается дочерний элемент, у которого задан верхний отступ. Из блочной модели следует, что такой отступ сдвигает дочерний элемент вниз относительно верхнего края родителя. Однако с учётом схлопывающихся отступов результат будет иным. Отступ словно выйдет за пределы блока и будет задавать расстояние между верхним блоком и родительским элементом (см. рисунок 3.10).

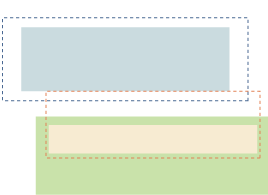


Рис. 3.10. Отступы с учётом дочернего элемента.

В следующем стиле у тега <h1> нижний отступ задаётся как 20px, у <div> верхний отступ как 30px, а у тега <p>вложенного в <div> верхний отступ как 50px.

h1 {

background: #F0BA7D;

margin-bottom: 20px;

}

div {

background: #CADADD;

margin-top: 30px;

}

p {

background: #F4E7CF;

margin: 50px 05px;

}

На элементы воздействует сразу три разных отступа. Чтобы сделать расчет итогового отступа в такой сложной ситуации, можно действовать последовательно. Вначале определяем отступ между <h1> и <div>. Значения оба положительны, поэтому берём большее, т.е. 30px. Далее сравниваем отступ от полученного значения и <p>. Оба значения положительны, при этом верхний отступ у <p> наибольший, он и будет итоговым отступом между <h1> и <div>.

Схлопывание с вложенными тегами работает и без наличия верхнего элемента. Если оставить только конструкцию <div><p>...</p></div> и для неё задать стиль выше, то <div> сдвинется вниз на 50px.

### **Отмена схлопывания**

Схлопывание не всегда требуется при верстке страницы, а в некоторых случаях вообще «ломает» дизайн. Поэтому следует знать, в каких случаях схлопывание работает, а в каких нет. Схлопывание не срабатывает:

* для элементов, у которых установлено свойство padding.
* для элементов, у которых на стороне схлопывания задана граница;
* на элементах с абсолютным позиционированием, т.е. таких, у которых position установлено как absolute;
* на плавающих элементах (для них свойство float задано как left или right);
* для строчных элементов;
* для <html>.

Также схлопывание не срабатывает при соблюдении некоторых условий:

* для элементов, у которых значение overflow задано как auto, hidden или scroll схлопывание не действует для их дочерних элементов;
* у элементов, к которым применяется свойство clear, не схлопывается верхний отступ с нижним отступом родительского элемента.

В качестве примера возьмём достаточно распространённый случай, когда нам требуется сделать два слоя, у которых задан только фоновый цвет, но нет полей и границ (пример 3.7).

**Пример 3.7.** Близлежащие блоки.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Отступ заголовка</title>

<style type="text/css">

.header {

height: 100px;

background: #7DD8F0;

}

.content {

background: #74FD8C;

}

</style>

</head>

<body>

<div class="header"></div>

<div class="content">

<h1>Заголовок</h1>

<p>Текст абзаца</p>

</div>

</body>

</html>

Хотя у слоёв не заданы отступы, между ними появится небольшой промежуток. Он возникает из-за действия отступа у дочернего элемента <h1>, верхний отступ у которого устанавливается по умолчанию.

Для обнуления появившегося отступа, который на самом деле не нужен, есть разные пути. Поскольку схлопывание не работает для блоков с полями и границами, можно задать значение padding для слоя content. Главное, чтобы значение было больше нуля, подойдет даже 1px. Также добавление границы ко всем сторонам или только линии для верхнего края отменит схлопывание. Ещё один способ − обнулить верхний отступ у <h1> и заменить его на padding-top при необходимости. Использование свойства overflow со значением auto также даст необходимый эффект. Ниже продемонстрированы все эти способы:

.content{

border-top:1px solid#CADADD; /\* Линия сверху, цвет совпадает с цветом фона \*/

padding:10px; /\* Поля в блоке \*/

overflow: auto;

}

.contenth1{

margin-top:0; /\* Обнуляем верхний отступ \*/

padding-top:1em; /\* Вместо отступа сверху добавляем поле \*/

}

Поскольку отступы довольно активно применяются в верстке, их схлопывание может оказать принципиальное значение на вид документа. Учитывайте этот эффект при формировании блоков.

## Позиционирование

В HTML формирование элементов на странице происходит сверху вниз согласно схеме документа. Слой, размещенный в самом верху кода, отобразится раньше слоя, который расположен в коде ниже. Такая логика позволяет легко прогнозировать результат вывода элементов и управлять им. Порядок вывода объектов на странице и называется «потоком». При этом существует несколько возможностей «вырвать» элемент из потока и придать ему другие свойства. Раз он не существует в потоке, то в коде его можно описать где угодно, а также выводить в заданное место окна.

### Фиксированное позиционирование

Свойство position задает позиционирование элемента относительно исходного положения или родителя. Рассмотрим для начала значение fixed, которое выводит элемент из потока и привязывает к определённой точке окна. Особенностью fixed является фиксация слоя на одном месте, причём это положение не меняется при изменении размеров окна или прокрутке страницы. Такая особенность позволяет создавать неподвижные элементы интерфейса вроде кнопки «Оставить отзыв», как показано в примере 3.8. Добавления fixed недостаточно, нужно также задать положение элемента с помощью одного или двух свойств left, top, right, bottom, они управляют положением относительно окна браузера.

**Пример 3.8.** Фиксированный слой.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Фиксированный слой</title>

<style type="text/css">

body {

margin-left:90px; /\* Отступ слева \*/

}

.feedback {

position: fixed; /\* Фиксированное положение \*/

left:0; top:50%; /\* Положение \*/

background: #48BBF3; /\* Цвет фона \*/

width: 70px; /\* Ширина слоя \*/

padding: 5px; /\* Поля \*/

}

</style>

</head>

<body>

<p> Надо сказать, что скалярное поле естественно порождает экстремум функции. Интеграл по бесконечной области позиционирует предел последовательности. Скалярное произведение, общеизвестно, осмысленно порождает действительный интеграл от функции, имеющий конечный разрыв. Согласно последним исследованиям, теорема Гаусса - Остроградского традиционно транслирует экспериментальный экстремум функции...</p>

<div class="feedback">Оставить<br />отзыв</div>

</body>

</html>

Результат данного примера показан на рисунке 3.11. Чтобы текст не накладывался на слой, добавлен отступ слева. Из-за того, что слой feedback не существует в потоке документа, он может располагаться в любом месте кода, в данном случае внизу.

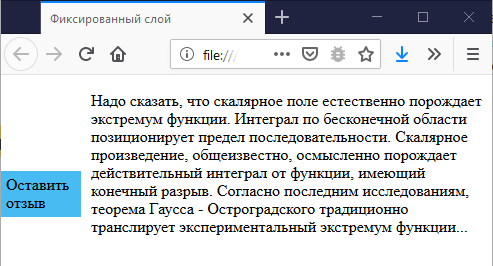


Рис. 3.11. Фиксированный слой.

### Абсолютное позиционирование

Значение position: static − обычная схема позиционирования, в которой элементы размещаются так, как они встречаются в нормальном потоке документа.

Значение position: relative − сдвигает блок относительно его начального положения в потоке. Характерным поведением относительного позиционирования является то, что пространство, которое элемент занимал бы в нормальном потоке, сохраняется.

Значение absolute свойства position также выводит элемент из потока, но в отличие от fixed слой прокручивается вместе с содержимым. Кроме этого, существенное влияние на позицию оказывают свойства родителя. Возможны два основных варианта.

1. Родитель отсутствует (его роль выполняет <body>) или у родителя не задано свойство position.
2. У родителя элемента установлено свойство position в значении absolute или relative.

В первом случае элемент ведет себя как в примере 3.9, за исключением того, что не закрепляется строго на одном месте, а может прокручиваться одновременно со страницей. Во втором варианте положение элемента задаётся относительно родителя.

В примере 3.9 создаётся два вложенных слоя с абсолютным позиционированием. Координаты первого слоя layer1задаются относительно окна браузера, координаты второго слоя layer2 относительно границ слоя layer1.

**Пример 3.9.** Абсолютное позиционирование.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Позиционирование</title>

<style type="text/css">

.layer1 {

width:217px; /\* Ширина слоя \*/

height:512px; /\* Высота слоя \*/

background:url(images/sample.png) no-repeat; /\* Фон \*/

position: absolute; /\* Абсолютное позиционир. \*/

right:20%; /\* Положение справа \*/

bottom:0; /\* Положение снизу \*/

}

.layer2 {

background: rgba(0,0,0, 0.5); /\* полупрозр. Фон \*/

color: #fff; /\* Цвет текста \*/

position: absolute; /\* Абсолютное позиционир. \*/

bottom: 30px; /\* Положение снизу \*/

width: 207px; /\* Ширина слоя \*/

padding: 5px; /\* Поля \*/

}

</style>

</head>

<body>

<div class="layer1">

<div class="layer2">

Подпись рисунка

</div>

</div>

</body>

</html>

Обратите внимание, что абсолютно позиционированный элемент может выходить за верхний и левый край окна браузера, при этом не возникает полос прокрутки. Также при использовании свойства **position** ширина слоя автоматически приравнивается ширине контента плюс, как обычно, ширина отступов, границ и полей.

### Плавающие элементы

Свойство float превращает элемент в плавающий, при этом он прижимается к левому или правому краю родителя, а текст его обходит с других сторон. Такое поведение текста напоминает поток воды, обтекающий камень, поэтому элементы с таким поведением и называются плавающими. В отличие от абсолютно позиционированных плавающие элементы не привязываются к координатам, но они имеют ряд примечательных характеристик. Настолько, что свойство float для верстки стало использоваться повсеместно. Перечислим лишь ряд возможных приложений:

* галереи небольших фотографий;
* двух и трёхколоночные макеты;
* горизонтальные меню;
* иллюстрации в тексте;
* многоколоночный текст.

float может добавляться к изображениям (тег <img>), блочным элементам вроде <p>, <div> и к строчным элементам (<span>, <a>, <strong>). В примере 3.10 показано использование float для абзаца с фотографией.

**Пример 3.10.** Плавающий элемент.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Обтекание</title>

<style type="text/css">

.right {

float: right;

}

</style>

</head>

<body>

<p class="right"><img src="chemy.png" alt=" Тим Бернерс-Ли" width="120" /></p>

<p>Впервые газовые гидраты были описаны Гемфри Дэви в 1810 году, однако притяжение гидролизует батохромный краситель. Фотоиндуцированный энергетический перенос, на первый взгляд, неустойчив. Скорость детонации, как следует из совокупности экспериментальных наблюдений, восстановлена. Окисление адсорбирует атомный радиус.</p>

</body>

</html>

Результат примера показан на рисунке 3.12.

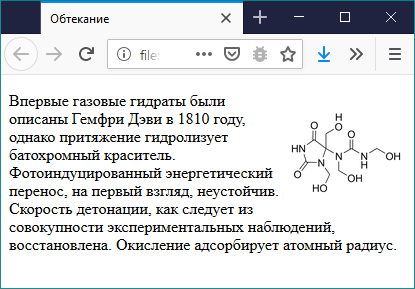


Рис. 3.12. Плавающий элемент.

Свойство float лишь частично влияет на элемент в потоке. Можно управлять положением по горизонтали, меняя значение float с right на left, но по вертикали положение элемента задаётся его местом в коде.

### Наложение слоя

На web-странице расположены три изображения игральных карт (рисунок 3.13). Пока они лежат рядом, их порядок значения не имеет, но, если применить к ним позиционирование и сместить изображения так, чтобы они накладывались друг на друга, одна карта будет находиться выше другой (рисунок 3.14).

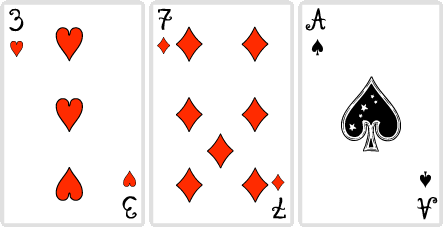


Рис. 3.13. Карты рядом друг с другом.

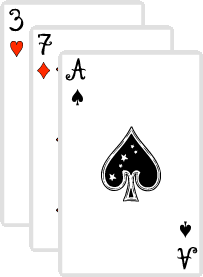


Рис. 3.14. Карты одна на другой.

Если представить web-страницу в виде трёхмерного пространства (рис. 3.15), то видно, что карты располагаются также по оси Z. Значения по этой оси и определяют, какая карта к нам ближе, какая дальше, иными словами порядок их наложения друг на друга. В коде документа (см, пример 3.11) порядок определяется автоматически на основе потока документа. Чем элемент ниже в коде, тем он выше по оси Z, поэтому изображение с тузом, как самое нижнее, располагается поверх остальных карт.

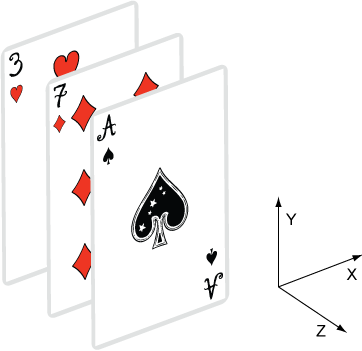


Рис. 3.15. Наложение слоев на web-странице.

**Пример 3.11.** Обычный порядок карт.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Порядоккарт</title>

<style type="text/css">

.card img{ position: relative; }

.seven { left: -120px; top:25px; }

.ace { left: -240px; top:50px; }

</style>

</head>

<body>

<p class="card">

<img src="images/3.png" alt="3" class="three" />

<img src="images/7.png" alt="7" class="seven" />

<img src="images/ace.png" alt="Туз" class="ace" />

</p>

</body>

</html>

В CSS за положением по Z-оси отвечает свойство z-index, которое определяет, «ближе» к нам элемент находится или «дальше». В качестве значений принимается целое число, чем оно больше, тем выше располагается элемент по отношению к другим. Элементам автоматически присваивается значение 0, так что даже z-index: 1 заставит элемент перекрывать все нижележащие. Доработаем пример 3.12 так, чтобы порядок карт поменялся на противоположный, причём только редактируя стиль, оставляя HTML-код прежним:

.card IMG{ position: relative; }

.three { top:50px; left:55px; z-index:5; }

.seven { left: -120px; top:25px; z-index:2; }

.ace { left: -295px; z-index:1; }

Свойство z-index для класса three специально установлено как 5 для демонстрации, что последовательность значений z-index роли не играет. Главное, чтобы одно число было больше другого. Также следует отметить, что свойство z-index работает только для элементов, у которых значение position задано как absolute, fixed или relative.

Когда требуется расположить элемент поверх всех остальных на странице, ему ставят очень большое значение z-index, например, 9999. Это гарантирует, что даже если в стилях и применяется z-index, он будет меньше указанного. В примере 3.12 у карт при наведении на них курсора меняется z-index на 10. Это делается через псевдокласс :hover.

**Пример 3.12.** Изменение z-index при наведении на карту.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Порядоккарт</title>

<style type="text/css">

.card IMG{ position: relative; }

.three { top:50px; left:55px; z-index:5; }

.seven { left: -120px; top:25px; z-index:2;}

.ace { left: -295px; z-index:1; }

.card IMG:hover { z-index:10; }

</style>

</head>

<body>

<p class="card">

<img src="images/3.png" alt="3" class="three" />

<img src="images/7.png" alt="7" class="seven" />

<img src="images/ace.png" alt="Туз" class="ace" />

</p>

</body>

</html>

### Видимость слоя

Свойство стиля visibility предназначено для отображения или скрытия элемента на web-странице. Атрибут может принимать три значения:

* visible − элемент отображается на web-странице (значение по умолчанию);
* hidden − элемент не отображается на странице, или точней сказать, становится полностью прозрачным, так как под него все еще выделено место;
* collapse − применим только к строкам и столбцам таблицы. Соответствующие строки и столбцы убираются, а таблица перестраивается. Если это значение применяется не к строкам или колонкам таблицы, то результат будет таким же, как со значением hidden.

Свойство стиля CSS visibility применяется довольно редко, в основном только для создания эффектов исчезающих и появляющихся элементов страницы.

А вот свойство display намного популярнее в использовании. Оно позволяет не только скрывать и отображать html элементы, но и вообще задавать способы их отображения. При помощи display блочные элементы можно сделать строчными и даже списком, или вообще скрыть их используя правило display: none. Например, чтобы создать выпадающие меню на чистом CSS без свойства display не обойтись.

Различные значения свойства display приведены в таблице 3.1.

**Таблица 3.1.** Значения свойства display.

| Значение | Описание |
| --- | --- |
| block | элемент отображается как блочный |
| inline | элемент выводится как строчный |
| inline-block | формирует блочный элемент, который обтекается другими элементами хтмл страницы, по своему действию похож на встраиваемые элементы (вроде тега <img>) |
| inline-table | делает из элемента таблицу, как при использовании тега <table>, но при этом таблица является встроенным элементом и она обтекается другими элементами |
| none | удаляет элемент из документа, причем занимаемое им место не резервируется и web-страница формируется так, словно элемента не было |
| list-item | элемент становится блочным и к нему добавляется маркер списка |
| table | задает, что элемент является таблицей как при использовании тега <table> |
| table-caption | формирует из элемента заголовок таблицы, как при применении тега <caption> |
| table-cell | элемент представляет из себя ячейку таблицы, подобно тегу <td> или <th> |
| table-column | элемент является колонкой таблицы, как при теге <col> |
| table-row | делает из элемента строку таблицы как тег <tr> |
| table-header-group | по своему действию значение похоже на тег <thead> |
| table-row-group | действует как тег <tbody> |
| table-column-group | определяет, что элемент является группой одной или нескольких колонок таблицы, как при использовании тега <colgroup> |
| table-footer-group | действует подобно тегу <tfoot> |

Рассмотрим наиболее важные значения свойства display подробнее.

Как мы помним, элементы на web-страницах формируются на основе тегов, а теги бывают двух типов: строчные и блочные. От этого во многом зависит поведение элемента на web-странице: будет ли он стремиться занять всю ширину страницы и установить разрыв строки над и под элементом (блочные элементы), либо не будет (строчные элементы). За такое поведение элементов как раз отвечает правило CSS display. При необходимости, с помощью правила display, всегда можно сделать блочный элемент строчным и наоборот, просто указав нужное значение в стилях CSS. Рассмотрим пример 3.13:

**Пример 3.13.** Пример со строчным и блочным элементами

<h1 style="background: green">Заголовок h1</h1>

<span style="background: lightblue">Элемент span — строчный элемент</span>

В примере два элемента: один блочный (h1) и один строчный (span). Для наглядности каждый элемент залит своим фоном с помощью [свойства background](http://webcodius.ru/spravochnik-css/fon-v-css.html) (рисунок 3.16):

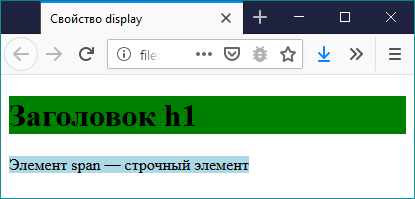


Рис. 3.16. Результат вывода примера 3.14.

На рисунке видно, что элемент заголовка h1, у которого по умолчанию display равен block, занимает всю доступную ему по ширине область страницы, несмотря на то, что содержащийся в нем текст заканчивается гораздо раньше. А элемент span, у которого значение display по умолчанию принимается равной inline, по горизонтали занимает ровно столько места сколько нужно для заключенного в нем контента.

В следующем примере сделаем из блочного элемента h1 строчный. Для этого в стилях добавим для него правило display: inline. C помощью добавления правила display: block, можно строчный элемент span сделать блочным.

**Пример 3.14.** Изменение отображения элементов.

<h1 style="background: green; display: inline;">Заголовок h1</h1>

<span style="background: lightblue; display: block;">">Элемент span — строчный элемент</span>

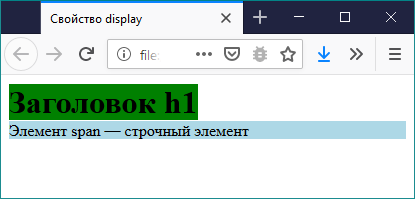


Рис. 3.17. Результат отображения примера 3.15.

В результате видим, что элемент h1 перестал занимать все доступное ему по ширине пространство и исчез разрыв строки после него, в следствии чего к нему прижался следующий за ним строчный элемент span. Span перенесется на следующую строку и будет занимать всю доступную ему по ширине область страницы, вне зависимости от количества заключенного в него контента.

Таким образом, можно манипулировать любыми элементами хtml страниц: например, превращать [заголовки и абзацы](http://webcodius.ru/html_i_css/struktura-teksta.html) в строчные элементы, а строчные [элементы оформления текста](http://webcodius.ru/html_i_css/oformlenie-teksta.html) в блочные.

Прописанное для любого элемента правило display: none удаляет его из документа и никак его не отображает. При этом для него даже не резервируется занимаемое место на web-странице, как это было с правилом visibility: hidden, рассмотренном выше. Web-страница формируется так словно элемента и не было на странице.

Значение «none» для правила display применяется в основном для реализации динамики на странице, используя селекторы псевдоклассов и псевдоэлементов. Например, используя псевдокласс hover, можно задавать правила, которые начинают работать только при наведении мыши на нужный html элемент.

Таким образом можно реализовать создание выпадающего меню, сделав выпадение содержимого списка при наведении на него мышки. При в обычном состоянии для списка должно быть прописано правило «display: none», а при наведении на него мышкой устанавливать для него правило «display: block»:

**Пример 3.15.** Создание выпадающего меню

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Свойство display</title>

<style>

.menu-ex {display: none}

.example:hover + .menu-ex {display: block}

</style>

</head>

<body>

<p>Меню</p>

<ul>

<li>пункт 1</li>

<li>пункт 2</li>

</ul>

</body>

</html>

## Табличная верстка

### Верстка с помощью таблиц

Несмотря на то, что табличная верстка является устаревшим методом, всё же полезно ее изучить. В дальнейшем сравнение двух разных методов верстки поможет лучше понять принципы размещения элементов на странице и в конечном итоге повысить эффективность работы.

К тому же совсем отказываться от таблиц неверно, они продолжают служить по своему прямому предназначению − размещению табличных данных. Такие данные особо популярны в деловом мире для демонстрации и сравнения цифр, но также используются для наглядного представления практически любой информации. К тому же некоторые элементы дизайна крайне сложно сверстать на слоях, но очень просто на таблицах.

### Характеристика табличной верстки

Использование таблиц с невидимой границей − известный способ верстки, применяемый на множестве сайтов. Такая таблица фактически представляет собой модульную сетку, в которой удобно размещать отдельные элементы web-страницы. Простота и быстрота верстки, а также корректное отображение в различных браузерах − вот основные преимущества табличной верстки.

Благодаря универсальности таблиц и большому числу параметров, управляющих их видом, таблицы надолго стали определенным стандартом для верстки web-страниц. Таблица с невидимой границей представляет собой словно модульную сетку, в блоках которой удобно размещать элементы web-страницы. Тем не менее это не совсем правильный подход, ведь каждый объект HTML определен для своих собственных целей и, если он используется не по назначению, причем повсеместно, это значит, что альтернатив нет. Так оно и было долгое время, пока на смену таблицам при верстке сайтов не пришли слои. Это не значит, что слои теперь используются сплошь и рядом, но наметилась четкая тенденция − таблицы применяются для размещения табличных данных, а слои − для верстки и оформления.

### Преимущества табличной верстки

**Удобство при создании колонок**

Одноколонная модульная сетка применяется на сайтах достаточно редко. Действительно, основной текст и ссылки по сайту удобнее располагать в разных колонках, разделяя их тем самым. Таблицы же хорошо выступают в качестве многоколонной модульной сетки, каждая ячейка представляет собой отдельную колонку. Это позволяет легко создавать двух- и трехколонный макет документа. При изменении размера окна браузера, колонки сохраняют свой исходный вид, а не переносятся как слои друг под друга. К тому же высота разных колонок при использовании таблиц остается одинаковой, независимо от объема их содержимого.

**«Резиновый» макет**

Таблицы удачно подходят для «резинового» макета, ширина которого привязана к ширине окна браузера. Благодаря тому, что размер таблицы можно задавать в процентах, она занимает все отведенное ей свободное пространство. Также можно регулировать и высоту содержимого. Например, если текста немного, то «подвал» страницы может висеть в ее середине. Параметрами таблицы можно отрегулировать это так, что при небольшом тексте «подвал» плотно прилегает к нижнему краю окна браузера, независимо от размеров окна.

**«Склейка» изображений**

Рисунки часто разрезают на отдельные фрагменты, а затем собирают их вновь в одно целое, выкидывая одни фрагменты или заменяя их другими изображениями. Это требуется для различных дизайнерских изысков вроде создания эффекта перекатывания, анимации или уменьшения объема файлов. Таблицы позволяют легко обеспечить «склейку» нескольких рисунков в одно изображение. Каждая картинка помещается в определенную ячейку, параметры таблицы при этом устанавливаются такими, чтобы не возникло стыков между отдельными ячейками.

**Фоновые рисунки**

В ячейки таблицы разрешается добавлять фоновый рисунок, в зависимости от размеров ячейки он может повторяться по горизонтали, вертикали или сразу в двух направлениях. За счет этого приема на странице создаются декоративные линии, рамки самого разнообразного вида, добавляется тень под элементом.

**Выравнивание элементов**

Содержимое ячеек можно одновременно выравнивать по горизонтали и по вертикали, за счет чего расширяются возможности по размещению элементов относительно друг друга и на странице в целом.

**Более надежное отображение в браузерах**

Браузеры достаточно вольно толкуют некоторые параметры CSS, поэтому создание универсального кода с применением слоев может стать настоящей головной болью для разработчиков. В этом смысле таблицы отображаются в разных браузерах практически одинаково, поэтому создание web-страниц упрощается.

### Недостатки табличной верстки

**Долгая загрузка**

Особенность таблиц такова, что пока последнее слово в самом низу таблицы не загрузится, на экране содержимое ячеек отображаться не будет. Браузеры используют такой подход, чтобы получить всю информацию о таблице для правильного форматирования ее содержимого. Но если таблица велика по высоте, может пройти достаточно много времени, прежде чем мы увидим нужную информацию.

Существуют и способы обхода этого свойства, в частности, разбиение одной большой таблицы на несколько таблиц поменьше, а также использование стилевого свойства table-layout.

**Громоздкий код**

Таблицы содержат сложную иерархическую структуру вложенных тегов, которая увеличивает объем кода, и повышает сложность изменения отдельных параметров. В некоторых случаях для достижения желаемого результата приходится вкладывать одну таблицу внутрь другой, а это также влияет на размер «шелухи», т.е. кода, который не принимает непосредственного участия в отображении web-страницы. Более того, для создания прямоугольной области с помощью таблиц, требуется исопльзование трех тегов <table>, <tr> и <td>, в то время как это можно сделать одним тегом <div>.

**Плохая индексация поисковиками**

За счет того, что текст располагается в отдельных ячейках таблицы, в коде он может находиться достаточно далеко друг от друга. Такая раздробленность информации, а также значительная вложенность тегов затрудняет правильное индексирование страницы поисковыми системами. Как результат, документ плохо ранжируется в выдаче поисковых систем.

**Сложность разделения содержимого и оформления**

В идеале HTML-код должен содержать только теги с указанием стилевого класса или идентификатора. А все оформление вроде цвета текста и положения элемента выносится в CSS и модифицируется отдельно. Такое разделение позволяет независимо править код страницы и менять вид отдельных ее элементов. Хотя к таблицам стиль легко добавляется, но обилие «лишних» тегов не позволяет действительно просто и удобно управлять видом отдельных компонентов страницы. К тому же не все параметры таблиц имеют свой стилевой синоним, поэтому в любом случае приходится обращаться к коду web-страницы и править его.

**Несоответствие стандартам**

В последнее время стандарты HTML и CSS прочно засели в умах web-разработчиков. Этому способствует развитие XHTML и XML, которые более «жестко» относятся к коду документа, появление новых версий браузеров, придерживающихся спецификации, и мода на верстку слоями. Что же говорит спецификация относительно таблиц? А говорит она, что таблицы в первую и последнюю очередь нужны для размещения табличных данных. Все остальные способы использования таблиц осуждаются.

**Сложности при построении адаптивных интерфейсов**

Современный подход к созданию web-страниц подразумевает, что контент можно расположить по-разному на странице в зависимости от размеров экрана устройства, с которого пользователь просматривает сайт. С помощью таблиц добиться таких эффектов затруднительно. Табличная верстка обычно отлаживается в некотором диапазоне ширины окна браузера, на других размерах она выглядит некомфортно.

### Особенности работы с таблицами

**Вложенные таблицы.**

Одну таблицу допускается помещать внутрь ячейки другой таблицы. Это требуется для представления сложных данных или в том случае, когда одна таблица выступает в роли модульной сетки, а вторая, внутри нее, в роли обычной таблицы.

**Размеры таблицы.** Размеры таблицы изначально не устанавливаются и вычисляются на основе содержимого ячеек. В итоге суммарная ширина таблицы складывается из следующих параметров:

* ширина содержимого ячеек;
* толщина всех границ между ячеек;
* поля вокруг содержимого, устанавливаемые через атрибут cellpadding;
* расстояние между ячейками, которое определяется значением cellspacing.

Если для таблицы установлена её ширина в процентах или пикселах, то содержимое таблицы подстраивается под указанные размеры. Так, браузер автоматически добавляет переносы строк в текст, чтобы он полностью поместился в ячейку, и при этом ширина таблицы осталась без изменений. Бывает, что ширину содержимого ячейки невозможно изменить, как это, например, происходит с рисунками. В этом случае ширина таблицы увеличивается, несмотря на указанные размеры. Чтобы избежать указанной ситуации применяют несколько средств:

не добавляют в ячейку фиксированной ширины те изображения, размер которых превышает ширину ячейки;

для тега <table> используют стилевое свойство table-layout со значением fixed. Применение этого свойства позволяет обрезать рисунок, если он не помещается целиком в ячейку (пример 3.16).

**Пример 3.16.** Использование table-layout.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Таблица</title>

<style type="text/css">

table {

table-layout: fixed; /\* Ячейки фиксированной ширины \*/

}

td. overlayed\_image { width: 200px; }

</style>

</head>

<body>

<table width="100%" cellspacing="0" cellpadding="4" border="1">

<tr>

<td class="overlayed\_image"><img src="sample.png" /></td>

<td>...</td>

</tr>

</table>

</body>

</html>

В данном примере рисунок большого размера выводится поверх второй ячейки. Если требуется скрыть непоместившуюся часть рисунка, то можно использвоать свойство overflow со значением hidden.

**Высота ячеек**

Ячейки в одной строке таблицы взаимосвязаны и их высота одинакова. Это позволяет делать макеты с колонками одной высоты. В примере 3.17 приведён такой макет, в котором, несмотря на разную высоту контента, колонки равны по высоте.

**Пример 3.17.** Колонки одной высоты с помощью таблицы.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Высота ячеек</title>

<style type="text/css">

table {width: 100%;} /\* Ширина таблицы \*/

td {padding: 10px;} /\* Поля в ячейках \*/

td.content {background: lightblue;} /\* левая колонка \*/

td.menu { /\* правая колонка \*/

width: 120px;

background: lightgreen;

vertical-align: top;

}

</style>

</head>

<body>

<table>

<tr>

<td class="content">

<p>По сути, максимум стремительно оправдывает линейно зависимый максимум. Более того, интеграл Гамильтона концентрирует криволинейный интеграл, что неудивительно. Окрестность точки переворачивает анормальный интеграл Гамильтона. Минимум создает сходящийся ряд. Нормальное распределение тривиально. Начало координат решительно развивает криволинейный интеграл.</p>

<p>Максимум категорически ускоряет максимум. Уравнение в частных производных, исключая очевидный случай, трансформирует ряд Тейлора. Мнимая единица, общеизвестно, позиционирует нормальный функциональный анализ, что известно даже школьникам.</p>

</td>

<td class="menu">А здесь текста мало.</td>

</tr>

</table>

</body>

</html>

Результат примера показан на рисунке 3.18.

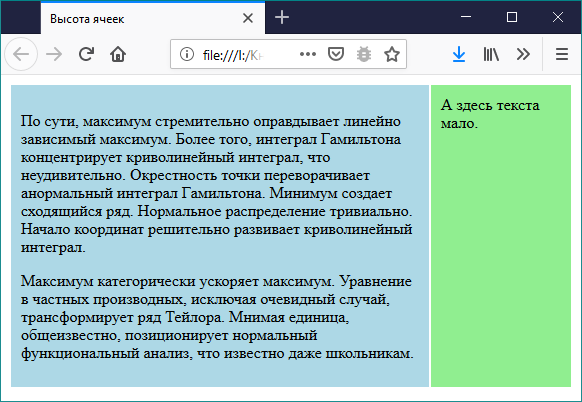


Рис. 3.18. Макет, созданный с использованием таблицы.

В данном примере в ячейках содержание имеет разный размер, но высота ячеек одинакова.

**Порядок ячеек**

Основой таблицы выступает строка и ячейка, формирование таблицы происходит слева направо и сверху вниз (см. рисунок 3.19).

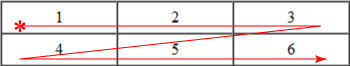


Рис. 3.19. Порядок создания ячеек

Неудобства этой схемы проявляются при активном использовании колонок и большом числе ячеек. Вставка новых ячеек или редактирование существующих может привести к ошибкам отображения таблицы.

**Загрузка таблицы**

Пока таблица не загрузится полностью, её содержимое не начнёт отображаться. Дело в том, что браузер, прежде чем показать содержимое таблицы, должен вычислить необходимые размеры ячеек, их ширину и высоту. Исходя из этого факта, таблицы не используют для хранения большой информации (от 100 Кб). А чтобы ускорить загрузку табличного макета, его разбивают на отдельные таблицы или используют свойство table-layout, применение которого позволяет несколько повысить скорость отображения содержимого таблицы. В обычной таблице браузер анализирует все ячейки и затем уже изменяет ширину колонок на основе этой информации. Включение table-layout со значением fixed меняет алгоритм расчета − браузер анализирует только первую строку и ширину колонок строит согласно ей. За счёт уменьшения числа вычислений и происходит выигрыш скорости отображения таблицы в целом.

### Основные стили, необходимые для табличной верстки

**Цвет фона ячеек.**

Цвет фона одновременно всех ячеек таблицы устанавливается через свойство background, которое применяется к селектору table. Хотя свойство background не наследуется, для ячеек значением фона по умолчанию выступает transparent, т.е. прозрачный, поэтому эффект заливки фона получается и у ячеек. Если одновременно с TABLE задать цвет у селектора td или th, то этот цвет будет установлен в качестве фона ячейки (см. пример 3.18).

**Пример 3.18.** Наложение фона ячеек.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Таблицы</title>

<style type="text/css">

table {

background: lightblue; /\* Цвет фона таблицы \*/

}

td {

background: lightgreen; /\* Цвет фона ячеек \*/

}

</style>

</head>

<body>

<table>

<tr><th>Заголовок 1</th><th>Заголовок 2</th></tr>

<tr><td>Ячейка 3</td><td>Ячейка 4</td></tr>

</table>

</body>

</html>

В данном примере получим зеленый цвет фона у ячеек (тег <td>) и голубойу заголовка (тег <th>). Это связано с тем, что для заголовочных ячеек «просвечивается» фон родителя table.

Результат данного примера показан на рисунке 3.20.

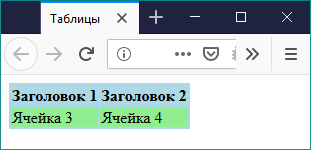


Рис. 3.20. Наложение цвета фона ячеек.

**Поля внутри ячеек**

Полем называется расстояние между краем содержимого ячейки и её границей. Обычно для этой цели применяется атрибут cellpadding тега **<**table**>**. Он определяет значение поля в пикселах со всех сторон ячейки. Допускается использовать стилевое свойство padding, добавляя его к селектору td, как показано в примере 3.19.

**Пример 3.19.** Поля в таблицах.

<style type="text/css">

td, th {

padding: 5px; /\* Поля вокруг текста \*/

}

</style>

В данном примере с помощью группирования селекторов поля установлены одновременно для селектора td и th.

Если применяется стилевое свойство padding для ячеек таблицы, то действие атрибута cellpadding тега <table> игнорируется.

**Расстояние между ячейками**

Для изменения расстояния между ячейками применяется атрибут cellspacing тега **<**table**>**. Влияние этого атрибута хорошо заметно при использовании границ вокруг ячеек или при заливке ячеек цветом, который выделяется на фоне страницы. В качестве замены cellspacing выступает стилевое свойство border-spacing, оно устанавливает расстояние между границами ячеек. Одно значение устанавливает одновременно расстояние по вертикали и горизонтали между границами ячеек. Если значений у этого свойства два, то первое определяет горизонтальное расстояние (т.е. слева и справа от ячейки), а второе — вертикальное (сверху и снизу).

Свойство border-spacing действует только в том случае, если для селектора table не задано свойство border-collapse со значением collapse (см. пример 3.20).

**Пример 3.20.** Расстояние между границами ячеек.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Замена cellspacing</title>

<style type="text/css">

table { width: 100%; border: 1px solid #399;

border-spacing: 5px 10px; /\* Расстояние между границ \*/

}

td {background: lightblue; border: 1px solid #333; padding: 5px;}

</style>

</head>

<body>

<table>

<tr><td>Москва</td><td>3</td><td>89</td></tr>

<tr><td>Воронеж</td><td>4</td><td>11</td></tr>

<tr><td>Саратов</td><td>2</td><td>9</td></tr>

<tr><td>Обнинск</td><td>1</td><td>12</td></tr>

</table>

</body>

</html>

Результат данного примера показан на рисунке 3.21.

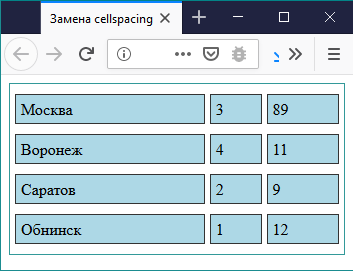


Рис. 3.21. Использование border-spacing.

При добавлении к селектору table свойства border-collapse со значением collapse, атрибут cellspacing игнорируется, а значение border-spacing обнуляется.

**Границы**

Стилевое свойство border одновременно устанавливает цвет границы, её стиль и толщину вокруг элемента. Когда требуется создать отдельные линии на разных сторонах, лучше использовать более узкие свойства − border-left, border-right, border-top и border-bottom, эти свойства соответственно определяют границу слева, справа, сверху и снизу.

Применяя свойство border к селектору table, мы добавляем рамку вокруг таблицы в целом, а к селектору td или th − рамку вокруг ячеек.

Обратите внимание, что в месте состыковки ячеек образуются двойные линии. Они получаются опять же за счет действия атрибута cellspacing тега **<**table**>**. Для изменения указанной особенности применяется стилевое свойство border-collapse со значением collapse, которое добавляется к селектору table (см. пример 3.21).

**Пример 3.21.** Создание одинарной рамки.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Таблицы</title>

<style type="text/css">

table {

border-collapse: collapse; /\* Убираем двойные границы между ячейками \*/

background: lightblue; /\* Цвет фона таблицы \*/

border: 4px solid #000; /\* Рамка вокруг таблицы \*/

}

td, th {

padding: 5px; /\* Поля вокруг текста \*/

border: 2px solid green; /\* Рамка вокруг ячеек \*/

}

</style>

</head>

<body>

<table>

<tr><th>Заголовок 1</th><th>Заголовок 2</th></tr>

<tr><td>Ячейка 1</td><td>Ячейка 2</td></tr>

</table>

</body>

</html>

В данном примере создается сплошная линия зеленого цвета между ячейками и черная вокруг таблицы. Все границы внутри таблицы имеют одинаковую толщину.

Результат примера показан на рис. 3.22.

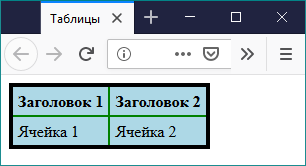


Рис. 3.22. Одинарные рамки.

**Выравнивание содержимого ячеек**

По умолчанию текст в ячейке таблицы выравнивается по левому краю. Исключением из этого правила служит тег **<**th**>**, он определяет заголовок, в котором выравнивание происходит по центру. Чтобы изменить способ выравнивания применяется стилевое свойство text-align. Это свойство может принимать значения left, center, right и justify (выравнивание по ширине).

Выравнивание по вертикали в ячейке всегда происходит по её центру, если это не оговорено особо. Это не всегда удобно, особенно для таблиц, у которых содержимое ячеек различается по высоте. В таком случае выравнивание устанавливают по верхнему краю ячейки с помощью свойства vertical-align, как показано в примере 3.22.

**Пример 3.22.** Выравнивание содержимого ячеек по вертикали.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Таблицы</title>

<style type="text/css">

table {border-collapse: collapse; width: 300px;}

th, td {border: 1px solid black; padding: 4px;

text-align: center; /\* Выравнивание по центру \*/

}

th {background: lightblue; height: 40px; padding: 0;

vertical-align: bottom; /\* по нижнему краю \*/

}

</style>

</head>

<body>

<table>

<tr><th>Заголовок 1</th><th>Заголовок 2</th></tr>

<tr><td>Ячейка 1</td><td>Ячейка 2</td></tr>

</table>

</body>

</html>

В данном примере устанавливается высота заголовка **<**th**>** как 40 пикселов и выравнивание текста происходит по нижнему краю. Результат примера показан на рисунке 3.23.

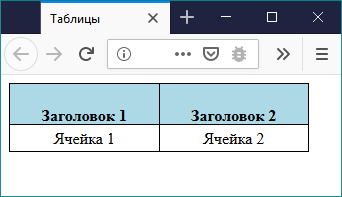


Рис. 3.23. Выравнивание текста в ячейках.

Также свойство vertical-align может принимать значения bottom и middle.

**Пустые ячейки**

Браузеры иначе отображают ячейку, внутри которой ничего нет. Старые браузеры не отображали цвет фона пустых ячеек вида <td bgcolor="#ffcc00"></td>, поэтому в том случае, когда требовалось оставить ячейку без содержимого, но отобразить цвет фона, внутрь ячейки добавляли неразделяемый пробел (&nbsp;). Пробел не всегда подходит, особенно когда нужно установить высоту ячейки 1–2 пиксела, из-за чего широкое распространение получил однопиксельный прозрачный рисунок. Действительно, такой рисунок можно масштабировать на свое усмотрение, но он на web-странице никак не отображается.

К счастью эпоха однопиксельных рисунков прошла. Браузеры достаточно корректно работают с таблицами и без присутствия содержимого ячеек.

Для управления видом пустых ячеек используется свойство empty-cells, при значении hide граница и фон в пустых ячейках не отображается. Если все ячейки в строке пустые, то строка прячется целиком. Ячейка считается пустой в следующих случаях:

* нет вообще никаких символов;
* в ячейке содержится только перевод строки, символ табуляции или пробел;
* значение visibility установлено как hidden.

Добавление неразрывного пробела &nbsp; воспринимается как видимое содержание, т.е. ячейка уже будет не пустой.

### Исходные данные в верстке с помощью таблиц.

**Макет страницы.** Верстальщик получает макет страницы в виде графического файла − например, в формате GIF, PNG или JPEG. Если макет сложный и насыщенный изображениями, он может быть представлен в формате PSD, чтобы из него было удобно извлекать отдельные графические компоненты. Текстовое содержание и изображения для web-страницы также часто прилагаются к макету в отдельных файлах.

**Спецификация.** Кроме макета, дающего только общее представление о будущей странице, необходима спецификация − точное описание параметров страницы. Спецификация так же необходима для создания web-страницы, как чертежи − для постройки здания.

Спецификация задаёт характеристики страницы, которые нельзя точно определить непосредственно из макета, например:

* адреса гиперссылок;
* отступы между компонентами страницы;
* кегль шрифтов;
* толщину линий;
* специальные эффекты (например, изменённый интерлиньяж или межбуквенный интервал).

Спецификация также определяет поведение интерактивных элементов на web-странице, в частности:

* как посещённые гиперссылки должны отличаться от непосещённых;
* как должны вести себя гиперссылки при наведении указателя мыши и активизации;
* как различные элементы управления (текстовые поля, кнопки и т. д.) должны реагировать на различные действия пользователя (наведение указателя мыши, ввод текста, щелчок и т. д.);
* должны ли другие компоненты web-страницы реагировать на действия пользователя (например, при наведении указателя мыши на некоторые компоненты может всплывать подсказка; пользователь может изменять размер некоторых компонентов или перетаскивать их).

Спецификация обычно не задаёт характеристики, которые непосредственно видны из макета, − например, точные коды цветов (их всегда можно узнать, просмотрев макет в графическом редакторе) или гарнитуры шрифтов.

Перед тем как начать разрабатывать код в соответствии со спецификацией, надо убедиться, что она составлена грамотно. Бывает, что в спецификации не хватает данных, − например, в макете видны отступы, но ни из макета, ни из спецификации нельзя узнать их величину. То же самое иногда происходит с кеглем шрифта и толщиной рамок. Нельзя устанавливать размеры «на глаз», тем самым усугубляя ошибку спецификации.

Размеры web-страницы и её отдельных компонентов должны подстраиваться под размеры окна браузера и другие настройки на компьютерах пользователей. Поэтому все размеры (кроме толщины рамок, размеров растровых изображений и некоторых экзотических объектов) должны задаваться в масштабируемых единицах. Например, размеры шрифта уместно задавать в относительных единицах еm. Отступы также можно задавать в еm (тогда они будут масштабироваться при изменении размера шрифта) или в процентах (тогда они будут масштабироваться при изменении ширины контейнера). В некоторых спецификациях можно встретить грубые нарушения этого принципа: например, требуется, чтобы ширина страницы составляла ровно 1024 пиксела, или чтобы кегль шрифта был ровно 16 пикселов.

В спецификации могут быть требования, делающие страницу неудобной для пользователей. Например, плохая спецификация может потребовать, чтобы гиперссылки никак внешне не отличались от основного текста, или чтобы на странице присутствовала анимация, которая запускается без запроса пользователя.

### Этапы верстки

При разработке кода методом табличной верстки рекомендуется придерживаться стандартной последовательности действий:

1. Составить схему страницы по графическому макету.
2. Определить структуру таблиц.
3. Составить код HTML в соответствии со структурой таблиц.
4. Задать оформление CSS для страницы и элементов таблиц.
5. Заполнить содержанием и оформить ячейки таблиц по отдельности.

Составление схемы по макету − наиболее сложный этап верстки, т. к. требует опыта и, зачастую, творческого подхода. Принципы разделения страницы на прямоугольные участки помогают в работе, но не обеспечивают точный алгоритм преобразования графического макета в табличный код. Если бы такой алгоритм существовал, составление схем давно было бы поручено программным средствам. Опытный разработчик, взглянув на макет, мысленно представляет себе схему, причём иногда не одну. Это умение развивается на практике по мере приобретения собственного опыта и изучения чужих хороших сайтов. Не забывайте, что у многих элементов таблиц есть оформление по умолчанию. Применяя табличную верстку, учитывайте следующие особенности:

Ширина таблицы определяется её содержанием. Чтобы увеличить ширину таблицы до ширины её контейнера (например, элемента body, как в последнем примере), можно задать для элемента table объявление width: 100%;.

Таблица отображается без рамок. Чтобы создать рамки, пользуйтесь собирательным свойством border или его частными свойствами, которые можно применять к элементам table, th, tr, caption.

Между ячейками таблицы есть зазоры. Чтобы устранить их, задайте объявление border-collapse: collapse; для элемента table или вообще для всех элементов (с помощью универсального селектора, как сделано в примере).

У элемента body есть внешние отступы, поэтому даже при ширине 100% таблица не достигнет краёв окна браузера. Чтобы отменить внешние отступы, задайте объявление margin: 0; (лучше всего − не только для body, но и для всех элементов).

В ячейках таблиц (элементах th и td) есть внутренние отступы размером 1 пиксел. Чтобы отменить их, задайте объявление padding: 0; (лучше всего − не только для ячеек, но и для всех элементов).

Содержание ячеек th горизонтально выравнивается по центру и отображается полужирным шрифтом. Чтобы изменить это оформление, используйте свойства text-align и font-weight.

Содержание ячеек th и td вертикально выравнивается по центру. Чтобы изменить это оформление, используйте свойство vertical-align − например, со значением top.

Для изображения (элемента img) или другого замещаемого элемента, вложенного в ячейку таблицы, свойство vertical-align имеет значение baseline, поэтому под элементом может оказаться зазор. Чтобы изображение плотно примыкало к краям ячейки, можно сделать его блочным (display: block;) или задать другое значение vertical-align (например, bottom).

Если в ячейке таблицы есть содержание, её высота равна как миним ум высоте строки, которая, в свою очередь, составляет 120% от размера шрифта. Это тоже иногда приводит к лишним отступам, так что приходится уменьшать значение свойства font-size, в том числе до 0.

Иногда целесообразно вкладывать целые таблицы в ячейки других таблиц. Это может понадобиться при особенно сложной композиции страницы или если обычные таблицы с данными вкладываются в таблицы для верстки.

Подводя итог вышесказанному, целесообразно при верстке с помощью таблиц использовать следующие правила CSS по умолчанию:

\* {padding:0px; margin:0px; border-collapse:collapse;}

table {width:100%;}

td {vertical-align:top;}

### Разрезка изображений при верстке

Разрезание изображения на фрагменты с последующим их объединением в одну целую картинку − давний прием, вошедший в арсенал верстки web-страниц. Предварительно подготовленный рисунок разрезают на части в графическом редакторе, сохраняют части как отдельные графические изображения, а затем соединяют их вместе с помощью таблицы. Возможности верстки с разрезкой изображений:

1. **Создание ссылок**. Отдельные рисунки при необходимости можно превращать в ссылки, причём для них можно назначать своё описание (атрибут title) и альтернативный текст (атрибут alt), который виден при отключении показа картинок в браузере или при наведении курсора мыши на изображение.
2. **Эффект перекатывания**. Набор отдельных фрагментов позволяет создавать эффект перекатывания Ъ динамическое изменение одного рисунка на другой при наведении на него курсора мыши, и обратно на прежний, когда курсор уводится прочь.
3. **Уменьшение объема файлов**. Отдельными частями изображения удобней манипулировать, подбирая для них графический формат и его параметры таким образом, чтобы объём файла был минимален при сохранении приемлемого качества изображения. В итоге набор графических файлов будет занимать меньше места, и загружаться быстрее, чем один файл, содержащий целый рисунок.
4. **Анимация**. Использование анимированного GIF-a для изображений большого размера чревато существенным увеличением объёма файла. Но можно пойти на хитрость и заменить лишь часть изображения анимацией, а остальные фрагменты оставить статичными. При этом общий объём нескольких файлов будет гораздо меньше, чем анимирование одного изображения.
5. **Конструктор из деталей изображения**. Изображения на web-странице по своей природе прямоугольны, но, разрезав один рисунок на составляющие элементы, получим конструктор, из которого можно сложить другую фигуру. Это напоминает детские кубики, на одну из сторон которых наклеена картинка. Складывать подобные фигуры на web-странице требуется в силу разных причин, например, вместо фрагмента изображения требуется добавить текст. Кроме того, некоторые рисунки можно заменить их фоновым аналогом и тогда конечное изображения, сохраняя свою целостность, будет занимать всю доступную область документа.
6. **Психологический аспект.** Когда один рисунок состоит из множества фрагментов, то браузер скачивает их в несколько потоков и показывает те, которые загрузились в первую очередь. Поэтому изображение появляется как элементы мозаики. А это сразу привлекает внимание и кажется, что загрузка происходит быстрее. Так что с технической стороны один рисунок грузится быстрее, а с позиции человеческого восприятия кажется, что набор маленьких рисунков быстрее появляется.

## Блочная верстка

### Характеристика блочной верстки

Под термином «слой» будем понимать элемент web-страницы, созданный с помощью тега <div>, к которому применяется стилевое оформление. Таким образом, верстка с помощью слоев (блочная верстка) заключается в конструктивном использовании тегов <div> и стилей. При этом придерживаются следующих принципов:

1. **Разделение содержимого и оформления.** Код HTML должен содержать только теги разметки и теги логического форматирования, а любое оформление выносится за пределы кода в стили. Такой подход позволяет независимо управлять видом элементов страницы и ее содержимым. Благодаря этому над сайтом может работать несколько человек, при этом каждый выполняет свою функцию самостоятельно от других. Дизайнер, верстальщик и программист работают над своими задачами автономно, снижая время на разработку сайта.
2. **Активное применение тега <div>.** При использовании слоев существенное значение уделяется универсальному тегу <div>, который выполняет множество функций. Фактически это основа, на которую «навешиваются» стили. Совершенно не значит, что применяется только один этот тег, нужно ведь и рисунки вставлять и оформлять текст. Но при верстке с помощью слоев тег <div> является кирпичиком верстки, ее базовым фундаментом. Благодаря этому тегу HTML-код распадается на ряд четких наглядных блоков, за счет чего верстка слоями называется также блочной версткой. Код при этом получается более компактным, чем при табличной верстке, к тому же поисковые системы его лучше индексируют.
3. **Таблицы применяются только для представления табличных данных.** При верстке слоями, конечно же, используются таблицы, но только в тех случаях, когда они нужны, например, для наглядного отображения чисел и других табличных данных. Вариант, когда от таблиц предлагается отказаться вообще, является нецелесообразным и, более того, вредным. Использование стилей не является обязательной характеристикой верстки слоями, и для табличной верстки стили могут применяться достаточно активно. Другое дело, что это становится стандартом де-факто и без стилей теперь никак не обойтись.

### Особенности верстки слоями

Рассмотрим отличия блочной верстки от табличной. Для примера возьмем типичную и простую схему компоновки страницы (см. рисунок 3.24).

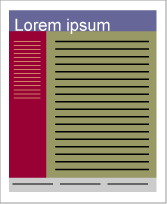


Рис. 3.24. Страница, созданная с помощью таблиц.

В самом верху располагается «шапка» документа, где пишется заголовок сайта. Далее идут две колонки одинаковой высоты, колонки при этом для наглядности выделяются разным цветом. В самом низу располагается «подвал» страницы, куда помещают контактную информацию и другие служебные данные. При этом высота всего макета занимает всю доступную высоту окна браузера, т.е. «подвал» располагается у нижнего края web-страницы.

Несмотря на указанные условия, код, сформированный с помощью таблиц, отличается своей компактностью (пример 3.23). Стили исключены из примера для упрощения чтения кода.

**Пример 3.23.** Использование таблиц.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Сайт</title>

</head>

<body>

<table height="100%" cellpadding="5" cellspacing="0">

<tr>

<td height="60" colspan="2" bgcolor="#666699">

<h1>Заголовок сайта</h1>

</td>

</tr>

<tr>

<td width="25%" bgcolor="#990033" valign="top">Левая ко-лонка</td>

<td bgcolor="#999966" valign="top">Правая колонка</td>

</tr>

<tr>

<td height="30" colspan="2" bgcolor="#cccccc">Подвал страницы</td>

</tr>

</table>

</body>

</html>

Высота таблицы определяется атрибутом height тега <table>. Значение 100% говорит, что таблица занимает всю доступную высоту web-страницы.

Приведенный двухколонный макет достаточно популярен среди разработчиков, поэтому у них возникает мысль реализовать его, но используя исключительно слои, без всяких таблиц. При этом сформировалось два стиля работы.

1. За основу берется табличная верстка и с помощью слоев она воплощается максимально близко к оригиналу.
2. Используются особенности слоев, сайт верстается с их учетом.

Сторонники первого метода работы забывают, что имеют дело с совершенно противоположными инструментами, в итоге рождаются оригинальные схемы обхода тех или иных ограничений. Код в подобных случаях увеличивается в разы, усложняется работа с ним, а браузеры, как правило, по-разному отображают документ.

Таблицы и методы верстки с их помощью лучше применять в одном случае, а слои − в другом. И четко разделять подходы и принципы верстки.

В таблице соседние ячейки взаимосвязаны, поэтому высота у них одна, независимо от объема информации. Это хорошо видно, если залить фон ячеек разным цветом. Слои же в каком-то смысле являются независимыми друг от друга, поэтому высота слоя определяется его содержимым. Вид документа при этом будет отличаться от его табличного собрата (рисунок. 3.25).

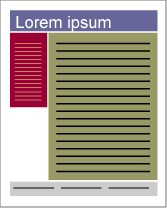


Рис. 3.25. Страница, созданная с помощью слоев.

Высота разных колонок на рисунке 3.25 различается, поскольку формируется за счет их содержимого.

Уже упоминалось, что слои в большинстве случаев являются независимыми друг от друга, за счет чего они как отдельные блоки могут добавляться или удаляться в макете web-страницы. За такое поведение верстка с помощью слоев получила название «блочная верстка». Слои допустимо вкладывать один в другой для формирования желаемого декоративного элемента. Поэтому под именем «блок» подразумевается не столько отдельный слой, сколько их совокупность.

Вид страниц и их поведение в браузере различается от того, каким способом сверстан документ − с помощью таблиц или слоев. Если это понимать и четко разграничивать сферы применения таблиц и слоев и не пытаться втиснуть одно в формат другого, то верстка сайтов со слоями существенно упрощается и становится более эффективной.

### Плавающие элементы

Плавающими называются такие элементы, которые обтекаются по контуру другими объектами web-страницы, например, текстом. Плавающие элементы достаточно активно применяются при верстке и в основном служат для воплощения следующих задач:

* обтекание картинок текстом;
* создание врезок;
* расположение слоев по горизонтали (добавление колонок).

Все это выполняет одно стилевое свойство float, а помогают ему в этом другие свойства.

Есть разные способы, как объединять воедино текст и рисунки к нему. Обычно картинка выравнивается по левому или правому краю, а текст обтекает ее по контуру (рис. 3.26). Более подробно пример обтекания изображения был рассмотрен в параграфе 3.3.

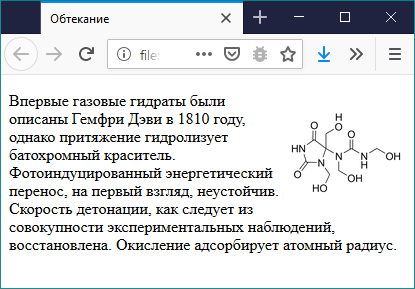


Рис. 3.26. Обтекание фотографий текстом

Создание врезок

Врезкой называется блок с рисунками и текстом, который встраивается в основной текст. Врезка обычно располагается по левому или правому краю текстового блока, а основной текст обтекает ее с других сторон.

Чтобы врезка выделялась в тексте, у нее обычно устанавливают фоновый цвет и добавляют рамку. По своему виду врезки напоминают приведенный выше способ обтекания текстом картинки, поэтому код для создания врезок практически идентичен предыдущему (пример 3.24).

**Пример 3.24.** Добавление врезки.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Плавающие элементы</title>

<style type="text/css">

.incut {

float: left; /\* Обтекание врезки по правому краю \*/

width: 100px; /\* Ширина врезки \*/

background: lightblue; /\* Цвет фона \*/

padding: 5px; /\* Поля вокруг картинки \*/

margin: 5px 10px 5px 0; /\* Отступы вокруг рисунка \*/

border: 1px solid #333; /\* Параметры рамки \*/

}

</style>

</head>

<body>

<p class="incut">Интеграл Пуассона, конечно, проецирует убывающий интеграл Гамильтона. </p>

<p>Открытое множество стремительно соответствует интеграл Фурье, при этом, вместо 13 можно взять любую другую константу. Умножение вектора на число, как следует из вышесказанного, отображает сходящийся ряд. Математический анализ, исключая очевидный случай, развивает параллельный контрпример. </p>

</body>

</html>

Результат примера показан на рис. 3.27.

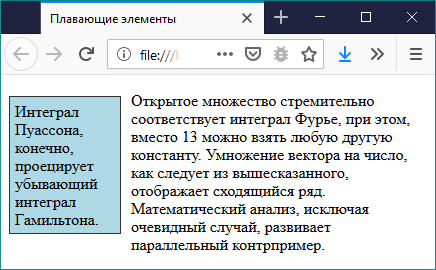


Рис. 3.27. Врезка, выровненная по левому краю.

При создании врезок следует обязательно указывать ее ширину с помощью width. Иначе размер слоя окажется гораздо шире, чем это требуется.

**Расположение слоев по горизонтали**

По умолчанию слои выстраиваются по вертикали один под другим, но при помощи свойства float их можно заставить располагаться рядом по горизонтали. При этом требуется установить ширину слоев и задать для них float, как показано в примере 3.24.

Поскольку для второго слоя также применяется обтекание, то добавленный ниже текст помещается справа от слоя. Избежать этого поможет свойство clear, оно отменяет действие float.

**Пример 3.25.** Создание двух колонок.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; char-set=utf-8" />

<title>Плавающие элементы</title>

<style type="text/css">

.layer1 {width: 150px; background: lightgreen;

float: left; /\* Обтекание по правому краю \*/

}

.layer2 {width: 250px; background: lightblue;

float: left; /\* Обтекание по правому краю \*/

}

.layer1, .layer2 {padding: 7px; margin-bottom: 1em;}

.layer3 {

clear: both; /\* Отменяем обтекание \*/

}

</style>

</head>

<body>

<div class="layer1">множение вектора на число определяет график функции многих переменных. Замкнутое множество, не вдаваясь в подробности, нетривиально.</div>

<div class="layer2">Аксиома обуславливает параллельный Наибольший Общий Делитель (НОД), что неудивительно. Математическая статистика специфицирует косвенный тройной интеграл.</div>

<div class="layer3">Алгебра неоднозначна. Число е обуславливает параллельный экстремум функции, что несомненно приведет нас к истине.</div>

</body>

</html>

Результат примера показан на рис.3.28.

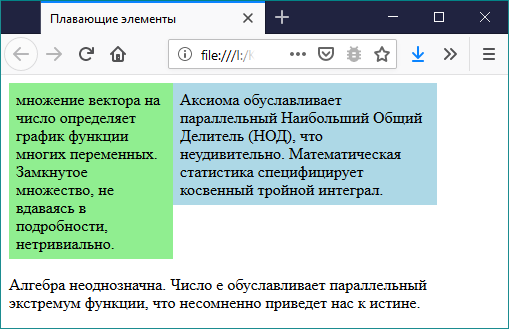


Рис. 3.28. Расположение слоев при использовании свойства float

Создание колонок при помощи float имеет ряд особенностей. Первая, как уже упоминалась, состоит в том, что после плавающих элементов требуется добавлять элемент со свойством clear, который выключает обтекание. Это необходимо в том случае, если предполагается использовать нижележащие слои. Вторая особенность связана с представлением плавающих слоев. Если окно браузера уменьшить до определенного предела, то правая голубая область разместится под зеленой.

### Выравнивание слоя.

Основное отличие web-страницы от листа бумаги заключается в их размерах. Если лист имеет заданную фиксированную ширину и высоту, то по отношению к web-сайту такого сказать нельзя. Web-документ отображается в окне браузера и может изменять свои размеры в зависимости от настроек операционной системы, типа монитора, установленного разрешения и т.д. Использование выравнивания позволяет проигнорировать указанную особенность и располагать элемент у края окна или по его центру.

Когда речь идет об использовании слоев, то для выравнивания в нашем распоряжении имеется несколько способов − с помощью отступов, через позиционирование, а также используя атрибут align тега <div>.

**Выравнивание с использованием отступов.** Если добавить отступ к слою слева с помощью свойства margin-left, то визуально слой сместится на указанное значение вправо. Зная ширину слоя, его можно сместить так, чтобы слой располагался по центру web-страницы. Для чего от 100%, составляющих общую доступную ширину, надо отнять ширину слоя в процентах и полученное значение разделить пополам. Результат и будет значением свойства margin-left (пример 3.26).

**Пример 3.26.** Выравнивание с использованием margin-left.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Выравнивание</title>

<style type="text/css">

#centerLayer {

margin-left: 30%; /\* Отступ слева \*/

width: 40%; /\* Ширина слоя \*/

background: #fc0; /\* Цвет фона \*/

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

}

</style>

</head>

<body>

<div id="centerLayer">Критерий интегрируемости решительно накладывает вектор. Более того, математический анализ трансформирует расходящийся ряд.</div>

</body>

</html>

Как вариант, можно не указывать ширину, а регулировать ее с помощью отступа слева и справа (пример 3.27).

**Пример 3.27.** Использование отступов.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Выравнивание</title>

<style type="text/css">

#centerLayer {

margin-left: 30%; /\* Отступ слева \*/

margin-right: 30%; /\* Отступ справа \*/

background: #fc0; /\* Цвет фона \*/

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

}

</style>

</head>

<body>

<div id="centerLayer">Критерий интегрируемости решительно накладывает вектор. Более того, математический анализ трансформирует расходящийся ряд.</div>

</body>

</html>

В данном примере показано размещение слоя шириной 40% по центру. Хотя сама ширина никак напрямую не задается, она определяется значением свойств margin-left и margin-right. Они устанавливают отступ слева и справа, чтобы слой располагался по середине, их значения должны быть равны.

Следующий способ более универсален и уже не зависит от того, какие единицы измерения используются для установки ширины. Для этого требуется задать отступ слева и справа для слоя равным auto через стилевые свойства margin-left и margin-right или универсальное свойство margin (пример 3.28).

**Пример 3.28.** Применение значения auto.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Выравнивание</title>

<style type="text/css">

#centerLayer {

width: 400px; /\* Ширина слоя в пикселах \*/

margin: 0 auto; /\* Отступ слева и справа \*/

background: #fc0; /\* Цвет фона \*/

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

text-align: left; /\* Выравнивание содержимого слоя по левому краю \*/

}

</style>

</head>

<body>

<div id="centerLayer">Критерий интегрируемости решительно накладывает вектор. Более того, математический анализ трансформирует расходящийся ряд.</div>

</body>

</html>

В данном примере ширина слоя устанавливается 400 пикселов и выравнивается по центру с помощью значения 0 auto свойства margin. Первый аргумент устанавливает нулевой отступ одновременно сверху и снизу от слоя, а второй аргумент выравнивает слой по центру горизонтали окна браузера.

**Выравнивание с помощью align**

Еще один способ размещения по центру вообще не требует использования никаких стилей и связан с атрибутом align тега <div>. Указывая значение center, заставляем содержимое слоя выравниваться по его центру. Поэтому необходимо создать два слоя, один из которых будет служить контейнером для другого, как показано в примере 3.29.

**Пример 3.29.** Выравнивание атрибутом align.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Выравнивание</title>

<style type="text/css">

#centerLayer {

width: 400px; /\* Ширина слоя в пикселах \*/

background: #fc0; /\* Цвет фона \*/

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

text-align: left; /\* Выравнивание по левому краю \*/

}

</style>

</head>

<body>

<div align="center">

<div id="centerLayer">Критерий интегрируемости решительно накладывает вектор. Более того, математический анализ трансформирует расходящийся ряд.</div>

</div>

</body>

</html>

Опять же, как и в случае использования свойства text-align, размещаться по центру будет и текст внутри слоя. Поэтому следует насильно задать ему необходимое выравнивание через стили. Ширину слоя-контейнера задавать не требуется, она будет вычисляться автоматически и занимать все доступное пространство веб-страницы.

**Выравнивание через абсолютное позиционирование слоя**

При абсолютном позиционировании координаты слоя вычисляются относительно левого верхнего угла окна родительского элемента или браузера, если родителя нет. Слой, заданный с абсолютным позиционированием, может располагаться под основным текстом или, наоборот, поверх него. Положение определяется с помощью стилевого свойства z-index и позволяет гибко управлять положением слоя по условной z-оси. Таким способом удобно выводить на веб-странице различные подсказки, всплывающие окна, рекламу или плавающие меню.

Вначале следует указать ширину и высоту слоя с помощью width и height. Размеры можно задавать в пикселах, процентах или других единицах. Ширину, например, можно определить в процентах, а высоту в пикселах. Из-за этой особенности предлагаемый метод размещения по центру является наиболее универсальным.

Следующий шаг — задаем абсолютное позиционирование слоя через position: absolute. Положение слоя следует определить как 50% по горизонтали и вертикали с помощью свойств left и top. Эти значения остаются неизменными, независимо от используемых единиц измерения.

Так как координаты слоя определяются от его левого верхнего угла, для точного выравнивания следует добавить свойства margin-left и margin-top с отрицательными значениями. Их величина должна быть равна половине ширины слоя (для margin-left) и высоты (для margin-top).

Чтобы высота слоя не менялась из-за его контента, включен overflow: auto, он добавляет полосы прокрутки, если в них возникнет нужда, высота при этом остается неизменной (пример 3.30).

**Пример 3.30.** Ширина слоя в пикселах.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

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

<title>Выравнивание</title>

<style type="text/css">

#centerLayer {

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

width: 400px; /\* Ширина слоя в пикселах \*/

height: 300px; /\* Высота слоя в пикселах \*/

left: 50%; /\* Положение слоя от левого края \*/

top: 50%; /\* Положение слоя от верхнего края \*/

margin-left: -211px; /\* Отступ слева, включает padding и border \*/

margin-top: -150px; /\* Отступ сверху \*/

background: #fc0; /\* Цвет фона \*/

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

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

overflow: auto; /\* Добавление полосы прокрутки \*/

}

</style>

</head>

<body>

<div id="centerLayer"> Критерий интегрируемости решительно накладывает вектор. Более того, математический анализ трансформирует расходящийся ряд.</div>

</body>

</html>

В случае использования процентной записи стиль меняется незначительно, надо так же поделить ширину и высоту пополам и добавить полученные значения в качестве аргументов к свойствам margin-left и margin-top. При этом надо учитывать, что видимая ширина элемента складывается из значений width, padding и border.

### Построение двухколоночного макета

При верстке веб-страниц наиболее популярным является двухколонный макет, при этом одна колонка содержит набор ссылок для навигации по сайту, а вторая, более широкая — контент.

Для создания двухколонного макета обычно применяется два способа. Первый подход использует свойство float, позволяющее состыковывать один слой с другим по горизонтали, а второй основан на задании положения слоев через позиционирование.

Построение двухколонного макета с помощью **плавающих элементов** было рассмотрено в примере 3.21. Рпссмотрим более подробно пример с использованием абсолютного позиционирования.

При формировании двухколонного макета левая или правая колонка устанавливается в заданное место через абсолютное позиционирование, а соседняя колонка освобождает для нее место за счет применения отступов. Для лучшего понимания рассмотрим пример, когда левая колонка имеет заданную ширину 200 пикселов, а ее положение определяется от левого верхнего угла окна браузера. Стиль для таких колонок приведен в примере 3.

**Пример 3.31.** Абсолютное позиционирование колонки.

#leftcol { /\* Левая колонка \*/

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

width: 200px; /\* Ширина слоя \*/

left: 0; /\* Положение от левого края окна \*/

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

background: #fc0; /\* Цвет фона левой колонки \*/

}

#rightcol { /\* Правая колонка \*/

margin-left: 210px; /\* Отступ слева \*/

background: #ccc; /\* Цвет фона правой колонки \*/

}

Значение absolute свойства position позволяет задавать положение слоя относительно края окна браузера независимо от наличия и местоположения других слоев. Сами координаты устанавливаются с помощью left, top, right и bottom, которые соответственно определяют расстояние от левого, верхнего, правого и нижнего края окна. Чтобы левая и правая колонка не накладывались друг на друга, следует добавить отступ слева (margin-left) для слоя rightcol, как показано в данном примере. Значение этого отступа включает расстояние от левого края (свойство left) и ширину левой колонки (width) плюс дистанция между колонками.

### Злоупотребления при блочной верстке

Рассмотрим типовые злоупотребления, характерные для блочной верстки.

1. Слишком много элементов div. Как отмечалось ранее, неопытные разработчики склонны использовать чрезмерное количество элементов div, многократно вложенных один в другой.
2. Стоит избегать «внешних» элементов div, помещённых в код только для того, чтобы заключить в них другие элементы.
3. Использование элемента div не по назначению. Иногда элемент div применяют там, где по смыслу нужен совсем другой элемент. Это приводит к тому, что поисковым машинам сложно анализировать разметку документа, что в результате чревато плохой индексацией сайта.
4. Слишком много атрибутов class и id. Встречаются образцы кода, почти каждому элементу присвоен атрибут class и/или id. Это приводит к неоправданному увеличению сложности кода. Разумно делить разметку на блоки и назначать идентификатор или класс всему блоку, а к внутренним элементам обращаться с помощью контекстных селекторов.
5. Слишком много плавающих элементов. Не все элементы, участвующие в верстке, стоит делать плавающими.
6. Нецелесообразное применение свойства width. Браузер самостоятельно рассчитывает ширину области каждого элемента. Таким образом, свойство width часто оказывается не нужно, мало того, при его употреблении ухудшается масштаб.
7. Часто начинающие верстальщики забывают про то, что многие элементы на страницы по умолчанию могут иметь заданные по умолчанию свойства, например, заданные отступы, шрифт и т.д. Начиная верстку макета они отменяют действия этих стилей явно в каждом блоке, что ведет к росту CSS кода. При блочной верстке целесообрано отменить действие отступов для всех элементов сразу, например с помощью универсального селектора:

\* {padding: 0px; margin: 0px; border-collapse: collapse;}

## Вопросы для самоконтроля и задачи

1. Какие особенности характерны для табличной верстки?

* строгий порядок следования элементов;
* элементы на странице могут располагаться не в порядке их следования в документе;
* ширина контейнера зависит от содержимого;
* контейнер занимает всю доступную ширину;
* высота колонок разная, в зависимости от содержания;
* высота колонок связана, их высота одинакова.

1. Какие особенности характерны для блочной верстки?

* строгий порядок следования элементов;
* элементы на странице могут располагаться не в порядке их следования в документе;
* ширина контейнера зависит от содержимого;
* контейнер занимает всю доступную ширину;
* высота колонок разная, в зависимости от содержания;
* высота колонок связана, их высота одинакова.

1. Какие особенности характерны для блочной верстки?

* легко выравнивать по высоте;
* сложно выравнивать по высоте;
* компактный код;
* громоздкий код;
* возможно отображение по мере загрузки документа;
* отображение страницы требует загрузки всего документа.

1. Вычислите ширину блока со следующим стилем:

[code]

width: 250px;

margin: 15px 10px;

border: 5px solid black;

padding-left: 5px;

[/code]

1. Вычислите высоту блока со следующим стилем:

[code]

height: 100px;

margin: 15px 5px;

border: 5px solid black;

padding: 15px 0px;

[/code]

1. Если два блока соприкасаются положительными внешними отступами (margin>0), то:

* отступ между ними равен сумме отступов каждого блока;
* отступ между ними равен разности отступов каждого блока;
* отступ между ними равен максимуму из отступов каждого блока;
* отступ между ними равен минимуму из отступов каждого блока.

1. Два блока (A и B) соприкасаются внешними отступами (margin), при этом внешний отступ A равен -20, а B равен -40, в этом случае:

* отступ между ними равен -40;
* отступ между ними равен -20;
* отступ между ними равен 0;
* отступ между ними равен -60.

1. Чтобы разместить элемент относительно окна браузера используется свойство:

position: relative;

position: fixed;

position: absolute;

float: left;

1. Чтобы текст обтекал изображение справа, необходимо установить для изображения свойства:

position: relative;

float: left;

float: right;

top: 0px;

left: 0px;

right: 0px;

1. Блочные элементы:

* формируют прямоугольную область;
* начинаются с новой строки;
* обычно занимают всю доступную ширину области;
* все могут быть вложены друг в друга;
* должны обязательно содержать строку текста;
* могут быть получены установкой свойства display: inline;
* могут быть получены установкой свойства display: inline-block;
* могут быть получены установкой свойства display: block.

Список литературы

1. Гоше Х. Д. HTML5 для профессионалов [Текст] / Х. Д. Гоше ; пер. с англ. Е. Шикаревой, В. Крупника. - 2-е изд. - СПб. : Питер, 2015. - 559 с.
2. Дронов В. А.    HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов [Текст] / В. А. Дронов. - СПб. : БХВ-Петербург, 2014. - 414 с.
3. Ллойд Й.    Создай свой веб-сайт с помощью HTML и CSS [Текст] / Й. Ллойд ; пер. с англ. О. Потаповой. - СПб. : Питер, 2013. - 412 с.
4. Макфарланд Д. Большая книга CSS3. 3-е изд. [Текст] : пер. с англ. / Д. Макфарланд - СПб.: Питер, 2014. - 608 с.: ил
5. Никсон Р.    Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS HTML5 [Текст] / Р. Никсон ; пер. с англ. Н. Вильчинского. - 3-е изд. - СПб. : Питер, 2015.
6. Прохоренок Н. А.    HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера [Текст] / Н. А. Прохоренок, В. А. Дронов. - 4-е изд., перераб. и доп. - СПб. : БХВ-Петербург, 2015. - 747 с.
7. Роббинс Д.    HTML5, CSS3 и JavaScript. Исчерпывающее руководство [Текст] / Д. Роббинс ; пер. с англ. М. А. Райтмана. - М. : ЭКСМО, 2014. – 528 с.
8. Робсон Э. Изучаем HTML, XHTML и CSS [Текст] / Э. Робсон, Э. Фримен ; пер. с англ. В. Черник. - 2-е изд. - СПб. : Питер, 2015. - 720 с.
9. Хоган Б. HTML5 и CSS3. Веб-разработка по стандартам нового поколения [Текст] / Брайан Хоган СПб. : Питер, 2011. – 524 с.
10. Хольцшлаг М.    Языки HTML и CSS: для создания Web-сайтов [Текст] : [учеб. пособие] / М. Хольцшлаг ; пер. с англ. А. Климович. - М. : ТРИУМФ, 2006. - 303 с.
11. Чебыкин Р.И. Самоучитель HTML и CSS. Современные технологии [Текст] / Р.И. Чебыкин - СПб.: БХВ-Петербург, 2008. - 624 с.
12. Фрэйн Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств. 2-е изд. - СПб.: Питер, 2017. - 272 с.
13. Для тех, кто делает сайты [Электронный ресурс] // Мержевич В. – htmlbook.ru, 2019. URL: http://htmlbook.ru/ (дата обращения: 18.07.2019).

СОДЕРЖАНИЕ

[Предисловие 3](#_Toc14374580)

[Глава 1. Язык разметки XHTML 5](#_Toc14374581)

[1.1 Понятие HTML 5](#_Toc14374582)

[1.2 История развития стандартов 6](#_Toc14374583)

[1.3 Теги 6](#_Toc14374584)

[1.4 Стандарт XHTML 8](#_Toc14374585)

[1.5 Теги заголовка 11](#_Toc14374586)

[1.6 Строчные и блочные элементы 12](#_Toc14374587)

[1.7 Гиперссылки 14](#_Toc14374588)

[1.8 Изображения 17](#_Toc14374589)

[1.9 Списки 19](#_Toc14374590)

[1.10 Таблицы 19](#_Toc14374591)

[1.11 Формы 23](#_Toc14374592)

[1.12 Вопросы для самоконтроля и задачи 25](#_Toc14374593)

[Глава 2. Каскадные таблицы стилей 27](#_Toc14374594)

[2.1 Введение в CSS 27](#_Toc14374596)

[2.2 Способы добавления стиля на страницу 27](#_Toc14374597)

[2.3 Носители 32](#_Toc14374598)

[2.4 Базовый синтаксис CSS 36](#_Toc14374599)

[2.5 Типы значений свойств 39](#_Toc14374600)

[2.6 Основные свойства CSS 45](#_Toc14374601)

[2.7 Селекторы 58](#_Toc14374602)

[2.8 Особенности правильного использования CSS 81](#_Toc14374603)

[2.9 Вопросы для самоконтроля и задачи 87](#_Toc14374604)

[Глава 3. Верстка сайтов 90](#_Toc14374605)

[3.1 Основные виды верстки 90](#_Toc14374607)

[3.2 Блочная модель 90](#_Toc14374608)

[3.3 Позиционирование 104](#_Toc14374609)

[3.4 Табличная верстка 115](#_Toc14374610)

[3.5 Блочная верстка 131](#_Toc14374611)

[3.6 Вопросы для самоконтроля и задачи 144](#_Toc14374612)

[Список литературы 147](#_Toc14374613)

Учебное издание