МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ГЛАВНОЕ УПРАВЛЕНИЕ ПО ОБРАЗОВАНИЮ

МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

|  |  |
| --- | --- |
| Специальность | 2– 40 01 01 |
| Учебная группа | ПО-455 |
|  |  |

Учебная дисциплина Программные средства

создания интернет-

приложений

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

**ПРИМЕНЕНИЕ РАЗЛИЧНЫХ ТЕХНОЛОГИЙ РАЗМЕЩЕНИЯ**

**ЭЛЕМЕНТОВ НА HTML-СТРАНИЦЕ**

Выполнил Савич А.О.

Проверил Денисовец Д. А.

2022

**1 Цель работы**

1.1 Научиться применять различные технологии размещения элементов на HTML-странице

**2 Индивидуальное задание**

**Вариант 8**

1 Создайте файл с CSS, используя программу, разработанную в лабораторной работе № 3:

Задание со списками:

текст – шрифт “Arial”, размер 12 pt, полужирный курсив, цвет шрифта -голубой, межстрочный интервал – 16 px.;

для списка должны быть разработаны элементы описывающие теги списка. Формат ol.1{<свойство1>…<свойствоN>} или li.N{<свойство1>…<свойствоN>} и т.д.

Задание с таблицами:

текст – шрифт “ Verdana ” , размер 12 pt, курсив, цвет шрифта черный.

Для таблицы должны быть разработаны элементы описывающие теги строк и столбцов таблицы. Формат table {<свойство1>…<свойствоN>} table.top1{<свойство1>…<свойствоN>}td.top1\_left{<свойство1>…<свойствоN>} и т.д.

2 Примените единое стилевое оформление к лабораторной работе :

заголовок страницы – шрифт “Courier New”, размер 18 pt, курсив, цвет шрифта синий;

для всего текста должны быть разработаны элементы описывающие теги абзаца. Формат p.1{<свойство1>…<свойствоN>}… p.N{<свойство1>…<свойствоN>} согласно оформления;

размер картинок 75 %, выравнивание по левому краю, рамка вокруг картинки красного цвета, толщина рамки 2 px.

<html>

<head>

<title>вот он заголовок</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<form method="post" action="select\_user.pp">

<label for="first\_name">Наименование фирмы дилера</label>

<br/>

<input type="text" name="first\_name" size="30"><br/>

<label for="city">город</label>

<br/>

<select id="city" name="city">

<option value="minsk">минск</option>

<option value="mogilev">могилев</option>

<option value="vitebsk">витебск</option>

<option value="grodno">гродно</option>

<option value="brest">брест</option>

<option value="gomel">гомель</option>

</select>

<label for="adres">Адрес</label>

<br/>

<input type="text" name="adres" size="30"><br/>

<label for="email">email</label>

<br/>

<input type="text" name="email" size="30" value="@"><br/>

<label for="phonenum">телефон</label>

<br/>

<input type="text" name="phonenum" size="30"><br/>

<label for="siter">веб сайт</label>

<br/>

<input type="text" name="siter" size="30" value="http://"><br/>

<input id="submit" type="submit" value="Отправить"><br/>

</form>

<img src = "1.png"></img>

</body>

</html>

ol {

font-family: : Arial;

font-style: italic;

font-weight: bold;

font-size: 12pt;

line-height: 16px;

}

table {

font-family: Verdana;

font-style: italic;

font-size: 12pt;

}

body{

font-family: Courier New;

font-style: italic;

font-size: 18pt;

color:blue;

}

img{

size: 75%;

border-width:2px;

border-color: red;

}

3 Контрольные вопросы

7.1 Каковы назначения и значения атрибутов font-family, font-size**?**

**Шрифт** в CSS — это ресурс, содержащий визуальное представление символов. На самом простом уровне он содержит информацию, которая сопоставляет коды символов с фигурами (называемые глифами), представляющие эти символы.

Шрифты, использующие общий стиль дизайна, обычно группируются в семейства шрифтов, классифицируемые набором стандартных свойств шрифта. Внутри семейства форма, отображаемая для данного символа, может варьироваться в зависимости от толщины обводки, наклона или относительной ширины.

Ресурсы шрифтов могут быть установлены локально на устройстве, в котором работает браузер. Для локальных ресурсов шрифта описательная информация может быть получена непосредственно из ресурса шрифта (например, из файла arial.ttf). Для загружаемых ресурсов шрифтов, также называемых веб-шрифтами, описательная информация включена со ссылкой на ресурс шрифта (например, для шрифта Poiret One).

Используя различные шрифты для заголовков, абзацев и других элементов, можно задавать определенный стиль письменных сообщений, передавая желаемые эмоции и настроение. Окунаясь в многообразие шрифтов, не забывайте, что текст основного содержимого веб-страницы должен быть в первую очередь читабельным.

Не рекомендуется использовать более двух шрифтов на странице, а желаемого контраста можно достигнуть за счет комбинирования шрифтов разной толщины, размера, начертания или же при помощи цвета. Более подробно с правилами **веб-типографики** вы сможете ознакомиться в предложенной статье.

Базовые свойства CSS-шрифтов

**СОДЕРЖАНИЕ:**

1. Семейство шрифтов: свойство font-family

2. Насыщенность шрифта: свойство font-weight

3. Ширина шрифта: свойство font-stretch

4. Начертание шрифта: свойство font-style

5. Размер шрифта: свойство font-size

6. Относительный размер шрифта: свойство font-size-adjust

7. Сокращенная запись свойств шрифта: свойство font

8. Управление синтезом шрифтов: свойство font-synthesis

1. Семейство шрифтов: свойство font-family

Свойство font-family используется для выбора начертания шрифта. Поскольку невозможно предсказать, установлен тот или иной шрифт на компьютере посетителя вашего сайта, рекомендуется прописывать все возможные варианты однотипных шрифтов. В таком случае браузер будет проверять их наличие, последовательно перебирая предложенные варианты.

Если в названии шрифта имеются пробелы или символы (например, #, $, %), то оно заключается в кавычки. Это делается для того, чтобы браузер мог понять, где начинается и заканчивается название шрифта.

Свойство наследуется.

|  |  |
| --- | --- |
| **font-family** | |
| Значения: |  |
| family-name | Название (имя) семейства шрифтов, например, Times, Courier, Arial. Рекомендуется указывать вместе с базовым семейством. |
| generic-family | Базовое семейство. CSS определяет пять базовых семейств шрифтов: Шрифты с засечками — Serif (Times New Roman, Times, Garamond, Georgia) Рубленые шрифты — Sans-serif (Helvetica, Geneva, Arial, Verdana, Trebuchet, Univers) Моноширинные шрифты — Monospace (Courier, Courier New, Andele Mono) Рукописные шрифты — Cursive (Comic Sans, Gabriola, Monotype Corsiva, Author, Zapf Chancery) Аллегорические шрифты (Western, Woodblock, Klingon) |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

font-family: "Times New Roman", Georgia, Serif;

font-family: serif;

font-family: sans-serif;

font-family: monospace;

font-family: cursive;

font-family: fantasy;

font-family: system-ui;

font-family: inherit;

font-family: initial;

CSS

2. Насыщенность шрифта: свойство font-weight

Свойство font-weight задаёт насыщенность шрифта.

Свойство наследуется.

|  |  |
| --- | --- |
| **font-weight** | |
| Значения: |  |
| normal | Значение по умолчанию, устанавливает нормальную насыщенность шрифта. Эквивалентно значению насыщенности, равной 400. |
| bold | Делает шрифт текста полужирным. Эквивалентно значению насыщенности, равной 700. |
| bolder | Насыщенность шрифта будет больше, чем у предка. |
| lighter | Насыщенность шрифта будет меньше, чем у предка. |
| 100, 200, 300, 400, 500, 600, 700, 800, 900 | Значение 100 соответствует самому легкому варианту начертания шрифта, а 900 — самому плотному. При этом, эти числа не определяют конкретной плотности, т.е. 100, 200, 300 и 400 могут соответствовать одному и тому же варианту слабой насыщенности начертания шрифта; 500 и 600 — средней насыщенности, а 700, 800 и 900 могут выводить одинаковое очень насыщенное начертание. Распределение плотности так же зависит от количества уровней насыщенности, определенных в конкретном семействе шрифтов. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

font-weight: normal;

font-weight: bold;

font-weight: lighter;

font-weight: bolder;

font-weight: 100;

font-weight: 200;

font-weight: 300;

font-weight: 400;

font-weight: 500;

font-weight: 600;

font-weight: 700;

font-weight: 800;

font-weight: 900;

font-weight: inherit;

font-weight: initial;

CSS

РИС. 1. СВОЙСТВО FONT-WEIGHT

3. Ширина шрифта: свойство font-stretch

Свойство font-stretch позволяет выбрать нормальное, сжатое или расширенное начертание символа из семейства шрифтов. Свойство не работает на любом шрифте, а только на шрифтах, для которых разработаны различными начертания, соответствующими определенным размерам.

Свойство наследуется.

Абсолютные значения ключевых слов имеют следующий порядок, от самого узкого до самого широкого:

|  |  |
| --- | --- |
| **font-stretch** | |
| Значения: |  |
| ultra-condensed | Указывает на наиболее сжатый шрифт. |
| extra-condensed | Указывает на второй по сжатости шрифт. |
| condensed | Указывает на сжатый шрифт. |
| semi-condensed | Указывает на немного сжатый шрифт. |
| normal | Значение по умолчанию. |
| semi-expanded | Слегка расширенный шрифт. |
| expanded | Расширенный шрифт. |
| extra-expanded | Второй по расширенности шрифт. |
| ultra-expanded | Максимально расширенный шрифт. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

font-stretch: ultra-condensed;

font-stretch: extra-condensed;

font-stretch: condensed;

font-stretch: semi-condensed;

font-stretch: normal;

font-stretch: semi-expanded;

font-stretch: expanded;

font-stretch: extra-expanded;

font-stretch: ultra-expanded;

font-stretch: inherit;

font-stretch: initial;

CSS

Когда не существует глифа для заданной ширины, значения normal или condensed отображаются для более узкого начертания символа, в противном случае отображается более широкое начертание. И наоборот, расширенные значения используют широкое начертание, в противном случае — узкое начертание. На рисунке ниже показано, как девять параметров свойства влияют на выбор шрифта для семейства шрифтов, содержащего различные ширины, серый цвет указывает ширину, для которой не существует начертания, поэтому подставляется другая ширина:

РИС. 2. СВОЙСТВО FONT-STRETCH

4. Начертание шрифта: свойство font-style

Свойство font-style позволяет выбрать стиль начертания для шрифта. При этом разница между курсивом и наклонным начертанием заключается в том, что курсив вносит небольшие изменения в структуру каждого символа, в то время как наклонное начертание представляет собой наклонную версию прямого шрифта.

Свойство наследуется.

|  |  |
| --- | --- |
| **font-style** | |
| Значения: |  |
| normal | Значение по умолчанию, устанавливает для текста обычное начертание шрифта. |
| italic | Выделяет текст курсивом. |
| oblique | Устанавливает наклонное начертание шрифта. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

font-style: normal;

font-style: italic;

font-style: oblique;

font-style: inherit;

font-style: initial;

CSS

РИС. 3. СВОЙСТВО FONT-STYLE

5. Размер шрифта: свойство font-size

Свойство font-size указывает желаемую высоту глифов из шрифта.

Свойство наследуется.

|  |  |
| --- | --- |
| **font-size** | |
| Значения: |  |
| absolute-size | xx-small, x-small, small, medium, large, x-large, xx-large. В качестве стандартного размера принимается medium. В CSS1 предложенный коэффициент масштабирования между соседними индексами составлял 1.5, что для пользователя оказалось слишком большим. В CSS2 предложенный коэффициент масштабирования для экрана компьютера между смежными индексами составлял 1.2, что все еще создавало проблемы для небольших размеров. Новый коэффициент масштабирования варьируется между каждым индексом, чтобы обеспечить лучшую читаемость. |
| relative-size | smaller, larger. Относительные размеры обусловливают изменение размера шрифта элемента относительно родителя. При этом размер шрифта может выйти за рамки размеров, предполагаемых для xx-small и xx-large. |
| длина | Размер шрифта устанавливается с помощью положительных значений единиц длины, например, px, em, как целых, так и дробных. |
| % | Относительное значение, вычисляется на основании любого размера, унаследованного от родительского элемента. Обеспечивает более точную настройку вычисляемого размера шрифта. Задание размеров шрифта с помощью em эквивалентно процентному значению. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

font-size: xx-small;

font-size: x-small;

font-size: small;

font-size: medium;

font-size: large;

font-size: x-large;

font-size: xx-large;

font-size: smaller;

font-size: larger;

font-size: 14px;

font-size: 0.8em;

font-size: 80%;

font-size: inherit;

font-size: initial;

CSS

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Значения absolute-size** | | | | | | | |
| **xx-small** | **x-small** | **small** | **medium** | **large** | **x-large** | **xx-large** |  |
| **Коэффициент масштабирования** | | | | | | | |
| 3/5 | 3/4 | 8/9 | 1 | 6/5 | 3/2 | 2/1 | 3/1 |
| **HTML заголовки** | | | | | | | |
| h6 |  | h5 | h4 | h3 | h2 | h1 |  |
| **HTML размер шрифта** | | | | | | | |
| 1 |  | 2 | 3 | 4 | 5 | 6 | 7 |

6. Относительный размер шрифта: свойство font-size-adjust

Для любого заданного размера шрифта явный размер и четкость текста варьируется в зависимости от шрифта. Для таких шрифтов, как латиница или кириллица, которые различают прописные и строчные буквы, относительная высота строчных букв по сравнению с их прописными аналогами является определяющим фактором удобочитаемости. Это обычно называют значением аспекта. Точно определенный, он равен x-height шрифта, разделенной на размер шрифта.

В ситуациях, когда для шрифта указано несколько семейств шрифтов, резервные шрифты могут не использовать то же значение аспекта, что и желаемое семейство шрифтов, и, следовательно, будут казаться менее четкими.

Свойство font-size-adjust — способ сохранить читабельность текста при использовании резервных шрифтов. Это достигается путем настройки размера шрифта таким образом, чтобы x-height была одинаковой независимо от используемого шрифта.

Свойство наследуется.

|  |  |
| --- | --- |
| **font-size-adjust** | |
| Значения: |  |
| none | Не сохраняет x-height шрифта. |
| число | Задает значение аспекта, используемое в приведенных ниже расчетах для расчета скорректированного размера шрифта: c = (a / a ') s где: s = значение размера шрифта a = значение аспекта, указанное в свойстве font-size-adjust a ' = значение аспекта фактического шрифта c = скорректированный размер шрифта для использования Отрицательные значения недействительны. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис:**

font-size-adjust: none;

font-size-adjust: 0.7;

font-size-adjust: inherit;

font-size-adjust: initial;

CSS

Как отображается текст в каждом из этих сравниваемых шрифтов, показано ниже, столбцы показывают текст, отображаемый в Verdana, Futura и Times. Одно и то же значение размера шрифта используется для ячеек в каждой строке, и для отображения различий по x-height включены красные линии. В верхней половине каждая строка отображается в том же значении размера шрифта. То же самое верно и для нижней половины, но в этой половине также устанавливается свойство font-size-adjust, чтобы фактический размер шрифта регулировался таким образом, чтобы сохранить x-height для каждой строки. Обратите внимание, что небольшой текст остается относительно разборчивым в каждой строке в нижней половине.

7. Сокращенная запись свойств шрифта: свойство font

Свойство font за исключением описанного ниже, является сокращенным свойством для установки font-style, font-variant, font-weight, font-stretch, font-size/line-height, font-family. Также могут быть включены значения для свойства font-variant, которые поддерживаются CSS 2.1 — normal или small-caps.

Все подсвойства свойства font сначала сбрасываются на свои начальные значения, включая перечисленные выше, плюс font-size-adjust, font-kerning, все подсвойства font-variant и настройки шрифтов, за исключением font-synthesis. Затем этим свойствам присваиваются те значения, которые указаны в свойстве font. Для свойства font-size-adjust невозможно установить значение, отличное от его начального значения, поэтому следует использовать вместо этого индивидуальное свойство. Если явное значение какого-либо свойства не нужно, то оно опускается.

Свойство наследуется.

**Синтаксис**

font: 12pt/14pt sans-serif;

font: 80% sans-serif;

font: x-large/110% "new century schoolbook", serif;

font: bold italic large Palatino, serif;

font: normal small-caps 120%/120% fantasy;

font: condensed oblique 12pt "Helvetica Neue", serif;

CSS

Следующие значения относятся к системным шрифтам:  
caption — шрифт, используемый для элементов управления с субтитрами (например, кнопок, раскрывающихся списков и т.д.).  
icon — шрифт, используемый для обозначения значков.  
menu — шрифт, используемый в меню (например, раскрывающиеся меню и списки меню).  
message-box — шрифт, используемый в диалоговых окнах.  
small-caption — шрифт, используемый для маркировки подписи элементов управления.  
status-bar — шрифт, используемый в строке состояния окна.

Системные шрифты могут быть установлены только целиком; то есть семейство шрифтов, размер, вес, стиль и т.д. задаются одновременно. Эти значения затем могут быть изменены индивидуально, если это необходимо. Ключевые слова, используемые для системных шрифтов, перечисленных выше, обрабатываются как ключевые слова только в том случае, если они находятся в начальной позиции, в других позициях эта же строка обрабатывается как часть имени семейства шрифтов. Системные шрифты могут быть указаны только с этим свойством, но не с самим font-family.

font: menu; /\* используются настройки шрифта для системных меню \*/

font: large menu; /\* используется семейство шрифтов под названием "menu" \*/

CSS

8. Управление синтезом шрифтов: свойство font-synthesis

Свойство font-synthesis определяет, разрешено ли пользовательским агентам (браузерам) синтезировать полужирное или наклонное начертание шрифтов, когда они отсутствуют в семействе шрифтов. Если weight не указан, пользовательские агенты не должны синтезировать полужирное начертание, а если style не указан, пользовательские агенты не должны синтезировать курсив.

Свойство наследуется.

|  |  |
| --- | --- |
| **font-synthesis** | |
| Значения: |  |
| none | Запрещает синтез начертаний. |
| weight и/или style | По умолчанию свойство принимает значение font-synthesis: weight style;. Если указано только weight, это говорит браузеру, что жирный шрифт может быть синтезирован при необходимости. Если только style — синтезируется только курсив. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

font-synthesis: none;

font-synthesis: weight;

font-synthesis: style;

font-synthesis: weight style;

font-synthesis: initial;

font-synthesis: inherit;

7.2Каковы назначения и значения атрибутов color, background-color?

Каждый блок html-элемента имеет фоновый слой, который может быть полностью прозрачным (по умолчанию) или заполнен цветом и/или одним или несколькими изображениями. CSS-свойства фона указывают, какой цвет background-color и изображения background-image использовать, а также их размер, расположение, способ укладки и т.д.

Свойства фона не наследуются, но фон родительского блока будет просвечивать по умолчанию из-за начального значения в background-color: transparent.

Фон не отображается у пустых элементов с нулевой высотой. Отрицательные значения свойства margin не влияют на фон элемента.

Свойство background-color устанавливает цвет фона элемента. Цвет рисуется за фоновыми изображениями. Для блочных элементов цвет фона распространяется на всю ширину и высоту блока элемента, для строчных — только на область их содержимого.

Цвет фона обрезается в соответствии со значением background-clip самого нижнего слоя фонового изображения.

Свойство не наследуется.

|  |  |
| --- | --- |
| **background-color** | |
| Значения: |  |
| цвет | Значение принимает все форматы цвета свойства color. Значение по умолчанию transparent. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис**

background-color: salmon;

background-color: #00ff00;

background-color: rgba(255, 128, 128, 0.5);

background-color: currentColor;

background-color: transparent;

background-color: inherit;

background-color: initial

7.3 Каковы назначения и значения атрибутов text-decoration, text-align?

Как же в Css работать с текстом? Вполне логично будет предположить, что для этой цели существуют специально предназначенные правила. Давайте начнем с text-align, которое является фактически заменой атрибута align (он использовался в чистом Html 4.01 по версии валидатора для выравнивания содержимого, например, абзацев P или заголовков).

Он имеет всего четыре возможных значения:

Возможные значения CSS свойства text-align

Смысл остается таким же, как и был раньше. **Text align** — это горизонтальное выравнивание строк. Применяется это правило исключительно только для блочных элементов (параграфы, заголовки и т.п.), т.е. тех тегах, в которых может появиться несколько строк. Т.к. в строчных элементах строк может быть только одна, то и смысла использования в них text-align особого нет.

Понятно, что значения этого правила означают выравнивание, соответственно: по левому краю (left), по правому краю (right), по центру (center) и по ширине страницы (Justify — одновременно по левому и правому краю за счет увеличения расстояния между словами). Само собой, что значение Justify стоит использовать для элементов хотя бы с несколькими строками текста, иначе видимого эффекта от этого не будет.

Для примера, я выровнял предыдущий абзац по ширине (видите у него ровные границы и слева, и справа), используя:

text-align:justify;

По умолчанию горизонтальное выравнивание текста осуществляется по левому краю, т.е. специально писать text-align:left не нужно, если, конечно же, раньше вы не задавали другое выравнивание. Этот абзац я, кстати, выровнял по центру (center) опять же для наглядного примера, но тут, я думаю, все и так понятно.

Следующее Css правило **text-indent** позволяет задать красную строку, например, для текста в теге абзаца P. Отступ красной строки можно задать с помощью указания величины (как со знаком плюс, так и со знаком минус, используя размеры (пиксели, em или ex) допустимые в CSS) или с помощью процентов:



От чего считаются проценты в text-indent? От ширины области, которая отведена под текст. Т.е. Css правило text-indent:50% задаст красную строку равную половине длины этой самой строки. Ну, а этот абзац как раз и служит примером такого правила.

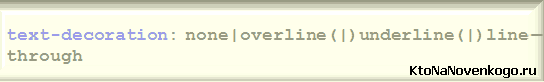
А можно, например, задать отрицательное значение красной строки в text-indent и тогда мы получим примерно то, что вы видите в этом абзаце. Для достижения данного результата я написал для тега абзаца P вот такое вот CSS правило:

text-indent:-1em;

Ну, а обычное использование text-indent (для задания стандартной красной строки) может выглядеть так: text-indent:40px; (кстати, применено к этому параграфу). Это правило, так же как и рассмотренное раньше text-align, **применяется только для блочных элементов**, т.е. там, где может появиться несколько строк (абзацы, заголовки и т.п.).

Так, теперь давайте перейдем к **text-decoration** (оформление с помощью горизонтальной линии), которое применяется уже ко всем Html элементам (и строчным, и блочным).

Оно может иметь всего четыре значения:



Т.е. можно использовать с помощью text-decoration: надчеркивание (overline), перечеркивание (line-through) или подчеркивание (underline), ну, или вообще ничего не использовать (none). Некоторые Html элементы уже имеют по умолчанию оформление горизонтальной линией, например, гиперссылки (тег A) (они по умолчанию подчеркиваются).

Поэтому выделять что-то еще подчеркиванием (кроме гиперссылок) не есть хорошо, ибо у пользователей в подсознании записано, что раз подчеркнуто (а еще и цветом выделено) — значит по этому можно кликнуть для перехода. Но выделяя подчеркиванием обычный текст, вы вводите пользователя в заблуждение и последующее за ним разочарование вашим ресурсом (он то думал, а оказалось что...).

Если текст гиперссылки нужно будет сделать неподчеркнутым (например, в меню), то используйте для text decoration элемента гиперссылки значение none.

Нюансом в использовании Css правила text-decoration является то, что можно будет прописать сразу три (иди два) значения для любого Html элемента (опуская none) и в результате вы получите **~~надчеркнуто-подчеркнуто-перечеркнутый фрагмент текста~~** (прикольно звучит и смотрится, не правда ли?):

text-decoration:underline overline line-through;

Значения для text decoration (если вы хотите использовать сразу несколько из них) нужно писать **через символ пробел**.

Vertical-align — вертикальное выравнивание

Дальше у нас идет вертикальное выравнивание — vertical-align. Практически для всех элементов в Html коде оно означает выравнивание между собой строчных элементов с текстом относительно их базовой линии. Правда, для тегов таблицы Td и Th это означает немного другое — выравниваться по вертикали будет весь контент, который находится в этих ячейках.

Для Css правила vertical-align можно использовать следующие значения:



Строки по умолчанию выравниваются по базовой линии. Вот смотрите, я применил **к этому фрагменту текста увеличение шрифта** и эти два фрагмента выровнялись по базовой (нижней) линии. А вертикальное выравнивание с помощью vertical-align как раз и предназначено именно для изменения способа выравнивания строк.

Например, если я для того же увеличенного фрагмента текста пропишу vertical-align:baseline, то никаких изменений не произойдет, т.к. значение baseline используется для этого Css правила по умолчанию.

Кстати, в качестве значений для него можно использовать и числа, а надпись vertical-align:0 будет означать тоже самое, что и vertical-align:baseline, т.е. значение baseline эквивалентно нулю. Следовательно, если мы хотим указать какой-либо сдвиг в вертикальном выравнивании, то этот сдвиг будем указывать относительно базовой линии (или же нуля).

Можно написать так:

vertical-align:10px;

И получим **сдвиг фрагмента с увеличенным шрифтом вверх** на 10 пикселей относительно базовой линии. Если напишем отрицательное значение:

vertical-align:-10px;

То получим **сдвиг фрагмента вниз** относительно базовой линии. Из примеров видно, что из-за сдвига увеличилась высота строки, чтобы текст помещался в ней без наезда на соседнюю строку. Сдвиг можно также задавать в Em и Ex, ну, и в процентах, которые будут считаться от высоты линии этого элемента (помните в прошлой статье мы научились ее задавать с помощью line-height).

Для вертикального выравнивания содержимого ячеек таблиц в vertical-align следует использовать значения Top и Bottom для получения, соответственно, выравнивания содержимого по верхней и нижней границе ячейки (ну, а middle в ячейке таблицы используется как значение вертикального выравнивания по умолчанию).

А для шрифтовых элементов можно использовать text-top, text-bottom, middle. Давайте для примера применим к **этому фрагменту текста** значение:

vertical-align:middle;

Что получилось в результате? По базовой линии обычного текста выровнялась средняя линия увеличенного фрагмента, т.е. мы получили вертикальное выравнивание по средней линии. Для text-top и text-bottom будет все аналогично. Вот так **text-top**, а так **text-bottom**.

Значения Css свойства vertical-align **sub** и **super** соответствуют под- и надиндексу, которые имели место быть в чистом Html (до использования CSS свойств для визуального оформления).

Text-transform, letter-spacing, word-spacing и white-space

Дальше у нас на очереди **text-transform** — трансформация символов. Оно так же применяется к любым Html элементам (и блочным, и строчным) и может иметь всего четыре значения:

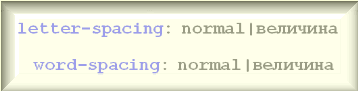


None используется по умолчанию и означает, что символы в тексте изменяться никак не будут — как написано в Html, так и будут отображаться. Значение Uppercase для text-transform позволит трансформировать все буквы фрагмента в заглавные (**ПРИМЕР ПОКАЗАН В ЭТОМ ПРЕДЛОЖЕНИИ**, где использовалось правило text-transform:uppercase, а изначально буквы были написаны строчные).

Значение lowercase для Css правила text-transform позволит вам трансформировать все символы фрагмента в строчные, ну, а значение capitalize сделает все первые буквы слова заглавными (**Пример В Этом Предложении** — text-transform:capitalize). Т.е. с помощью text-transform можно сделать все что угодно с обычным текстом, а потом запросто все вернуть обратно.

Поэтому, если у вас, например, стоит задача сделать все заголовки написанными только заглавными буквами, то в Html пишите их обычно, а заглавными их сделаете уже в CSS через text-transform:uppercase. Потом, если вы решите что-то поменять назад, то достаточно внести только маленькое изменение в стили, а не в содержимое всех 100500 заголовков на вашем сайте.

Дальше давайте рассмотрим CSS правила, которые позволят нам изменять расстояния между символами и словами — **letter-spacing и word-spacing**. Оба эти правила могут иметь два типа значений:



По умолчанию и letter-spacing, и word-spacing имеют значение Normal, ну или это тоже самое, что ноль (т.е. расстояние между символами и словами никак не изменяется). Величину же изменения расстояния в этих правилах можно будет указывать только в пикселах, либо Em или Ex, но никак не в процентах.

Однако, можно использовать как положительные (разрежение символов или слов), так и отрицательные значения (сближение символов или слов). Например, можно **«вот так разредить символы в этой фразе»** с помощью следующего Css правила:

letter-spacing:0.4em;

Или же можно **«вот так сблизить символы в этой фразе»** с помощью:

letter-spacing:-1px;

Тоже самое можно сказать и про word-spacing с одной лишь только разницей, что при этом расстояние будет изменяться уже между словами, **как, например, в этой фразе, при помощи вот такой вот CSS конструкции**:

word-spacing:4em;

Аналогично можно использовать в word-spacing отрицательные значения для уменьшения расстояния между словами.

Ну, и последнее на сегодня Css правило, которое позволяет определенным образом оформлять текст в Html коде — это **white-space**. Отвечает оно за отображение пробельных символов на веб странице, которые имели место быть при написании Html кода.

Как вы помните из статьи про символы пробела в Html, браузер при разборе кода объединяет все пробелы, символы переноса строк и табуляции в один единственный пробел, и осуществляет перенос строк на веб странице именно по пробельным символам, которые имели место быть в коде.

Так вот, white space может принимать одно из трех значений:



Понятно, что по умолчанию используется значение Normal и в этом случае все отображается так, как я описал чуть выше. А вот при использовании значения Pre мы получим полную аналогию с использованием тега Pre в чистом Html, т.е. на вебстранице текст будет отображен со всеми теми лишними пробельными символами, которые имели место быть при написании кода и переносы браузер уже по ним сделать не сможет.

Ну, а значение **nowrap** просто запретит браузеру переносить по тем пробельным символам, которые он найдет внутри фрагмента с CSS правилом white-space:nowrap. Попробовать, как все это работает, вы сможете и сами, создав простенький Html файлик и заключив любой фрагмент текста в подобные теги:

<div style="white-space:pre;">фрагмент подопытного текста</div>

7.4 Каковы назначения и значения атрибутов margin-left, margin-top?

Свойство **margin** CSS определяет внешний отступ на всех четырёх сторонах элемента. Это сокращение, которое устанавливает все отдельные поля одновременно: margin-top, margin-right, margin-bottom и margin-left.

/\* Применяется ко всем четырём сторонам \*/

margin: 1em;

/\* по вертикали | по горизонтали \*/

margin: 5% auto;

/\* сверху | горизонтально | снизу \*/

margin: 1em auto 2em;

/\* сверху | справа | снизу | слева \*/

margin: 2px 1em 0 auto;

/\* Глобальные значения \*/

margin: inherit;

margin: initial;

margin: unset;

Copy to Clipboard

|  |  |
| --- | --- |
| **Начальное значение** | как и у каждого из подсвойств этого свойства:  margin-bottom: 0  margin-left: 0  margin-right: 0  margin-top: 0 |
| **Применяется к** | все элементы, кроме элементов с табличным типом display, отличным от table-caption, table и inline-table. Это также применяется к ::first-letter и ::first-line. |
| **Наследуется** | нет |
| **Проценты** | ссылается на ширину содержащего блока |
| **Обработка значения** | как и у каждого из подсвойств этого свойства:  margin-bottom: процент, как указан, или абсолютная длина  margin-left: процент, как указан, или абсолютная длина  margin-right: процент, как указан, или абсолютная длина  margin-top: процент, как указан, или абсолютная длина |
| **Animation type** | длина |

Синтаксис

Свойство margin может быть задано с использованием одного, двух, трёх или четырёх значений. Каждое значение имеет тип <length>, <percentage> или является ключевым словом auto. Каждое значение может быть положительным, отрицательным или равным нулю.

Когда определено **одно** значение, такое значение определено для **всех четырёх сторон**.

Когда определены **два** значения, то первое значение определяет внешний отступ для **верхней и нижней** стороны, а второе значение определяет отступ для **левой и правой стороны**.

Когда определены **три** значение, то первое значение определяет внешний отступ для **верхней** стороны, второе значение определяет внешний отступ для **левой и правой стороны**, а третье значение определяет отступ для **нижней** стороны.

Когда определены **четыре** значения, они определяют внешние отступы для **верхней** стороны, **справа**, **снизу** и **слева** в рассмотренном порядке (по часовой стрелке).

Значения

7.5 Каким образом можно придать единый стиль всем страницам сайта?

Существует четыре способа задания **CSS** стиля для тегов **HTML**.

1) **Inline-стиль**.

2) **Внедрённый** стиль

3) **Импортированный** стиль.

4) **Стиль из файла**.

И в этой статье мы разберём все четыре способа.

Для начала первый способ - это **inline-стиль**. Данный стиль указывается непосредственно в самом теге. Например:

<p style = "font-size: 150%; text-align: center;">текст</p>

В данном случае мы задали, чтобы элемент "*текст*" будет размером в **150%** и выравнен по **центру**. Это пример **inline-стиля**.

Второй способ - это внедрённый **CSS стиль**, то есть стиль, который задаётся в голове документа, в теге **<head>**. Например, так:

<head>  
  <style>  
    h1 {  
      color: red;  
      margin-top: 100px;  
    }  
  </style>  
</head>

Здесь будет происходить следующее: для всех элементов внутри тега **<h1>** будет сделано следующее: их цвет будет красным и отступ сверху будет 100 пикселей.

Третий способ задания **CSS стиля** - это **импортированные стили**. Они также, как и **внедрённые** задаются в голове документа, но уже из файла. Вот пример:

<head>  
  <style>  
    @import url("my.css");  
  </style>  
</head>

Здесь ко всей странице будут применяться стили из файла "*my.css*".

И последние вид **CSS стилей** - это стили из файла. Данный способ является самым частым, и заключается он в подключении файла стиля через тег **<link>** в голове документа. Например, таким образом:

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

Данный способ очень похож на предыдущий способ, но здесь не используется тег **<style>**, и есть ещё одно отличие. Причём, это отличие характерно для всех способов задания **CSS стиля**. И это отличие - **приоритет**.

Когда проявляется приоритет? А проявляется он при возникновении **конфликтных ситуаций**, связанных с попыткой одному и тому же элементу задать противоречивый стиль. Например, через **inline-стиль** пытаемся сделать цвет красным, а через **импортированный** стиль пытаемся сделать цвет этого элемента чёрным. Во избежание таких противоречивых ситуаций, необходимо расставить приоритеты: какой способ с самым высоким приоритетом, а какой, наоборот, с самым низким. Приоритеты в порядке возрастания следующие:

1) **Стиль** из файла.

2) **Импортированный стиль**.

3) **Внедрённый стиль**.

4) **Inline-стиль**.

Таким образом, **inline-стиль** имеет наибольший приоритет. И, пользуясь этим, можно сделать вывод, что в примере выше цвет элемента будет красным, а не чёрным.

Какие выводы можно сделать?

1) **Общий стиль** для всего сайта должен быть вынесен в отдельный файл и подключаться на каждой странице через тег **<link>**, ввиду того, что данный стиль является стилем с минимальным приоритетом, его в частных случаях можно будет изменить.

2) **Импортированный стиль** надо использовать, когда **2 и более страниц (но не все)** имеют определённые особенности в стиле.

3) **Внедрённый стиль** надо использовать для задания уникальных **CSS стилей** для конкретной страницы. Эти стили уникальны для каждой страницы сайта.

4) **Inline-стиль** надо использовать, когда отдельный элемент на отдельной странице требует особенного вида.

Руководствуясь данными принципами, Вы никогда не запутаетесь в **CSS-стилях**.

Напоследок, хочется привести небольшой пример, как надо следовать этим принципам. Допустим, Вы хотите по-разному выводить элемент внутри тега **<p>**. Перед Вам стоят такие задачи:

1) Размер шрифта на **всех страницах сайта** должен быть **15pt**, а цвет **чёрным**.

2) **На всех страницах, кроме одной**, цвет текста внутри этого элемента должен быть **красным**.

3) **На каждой странице, отступы будут разными**.

4) **В отдельных случаях** может быть **изменён размер и цвет текста**.

Я, думаю, что здесь всё прозрачно, но тем не менее, давайте поясню порядок действий:

1) В отдельный файл вынести стиль, где задаётся размер и цвет. Потом подключить данный стиль через тег **<link>**.

2) В отдельный файл вынести стиль, где задаётся красный цвет текста. Потом подключить его, как **импортированный стиль** к нужным страницам.

3) На каждой странице внутри тега **<style>** сделать разные отступы для тега **<p>**.

4) У нужных тегов **<p>** настроить должным образом размер и цвет текста.

7.6 Какое расширение имеет файл содержащий стилевые спецификации?

.css

7.7 По каким правилам записываются в файл стилевые назначения?

**CSS (Cascading Style Sheets), или каскадные таблицы стилей**, используются для описания внешнего вида веб-документа, написанного языком разметки. CSS устанавливает стилевые правила, которые изменяют внешний вид элементов, размещенных на веб-страницах, выполняют тонкую настройку их деталей, таких как цвет, шрифт, размер, границы, фон и местоположение в документе.

Вы можете встроить CSS-код непосредственно в элемент разметки в виде значения атрибута **style**. Этот атрибут доступен для всех элементов HTML. С помощью CSS можно указать ряд свойств стиля для данного HTML-элемента. Каждое свойство имеет имя и значение, разделенные двоеточием (:). Каждое объявленное свойство отделяется точкой с запятой (;).

Вот как это выглядит для элемента </p>:

Пример: применение стилей к элементу <p>

**Результат**

**HTML-код**

**Попробуй сам »**

Текст с атрибутом style

Способы добавления CSS стилей

Стандарт CSS предлагает три варианта применения таблицы стилей к веб-странице:

Внешняя таблица стилей — определение правил таблицы стилей в отдельном файле .css, с последующим подключением этого файла в HTML-документ с помощью тега ***<link>***.

Внутренняя таблица стилей — определение правил таблицы стилей с использованием тега ***<style>***, который обычно располагается в разделе ***<head>*** HTML-документа .

Встроенный стиль — это способ вставки языка таблицы стилей прямо в начальный тег HTML-элемента.

Давайте познакомимся со всеми тремя способами на конкретных примерах.

Внешняя таблица стилей

Внешняя таблица стилей хранится в отдельном файле с расширением CSS. Это самый эффективный способ, т. к. он полностью отделяет правила форматирования от ваших HTML-страниц. Он также позволяет легко применять одни и те же правила ко многим страницам. Отдельный файл с таблицей стилей прикрепляется к HTML-документу с помощью тега ***<link>***. Это одинарный тег, который располагается внутри элемента ***<head>***. Элемент должен использовать три следующих атрибута. В атрибуте **href** указывается абсолютный или относительный URL-адрес файла CSS. Атрибут **rel** определяет отношение между НТМL-страницей и связанным файлом. При создании ссылки на файл CSS он должен иметь значение **stylesheet**, показывающее, что присоединяемый таким образом документ содержит таблицу стилей. Атрибут **type** определяет тип документа, на который указывает ссылка. В нем должно быть указано значение **text/css**.

CSS, как и HTML, не является языком программирования. Это язык каскадных таблиц стилей, то есть он позволяет выборочно применять нужные стили к элементам в HTML-документах. Например, чтобы выбрать все элементы ***<p>***, т.е. параграфы на HTML-странице и окрасить их текст в красный цвет, вы должны написать на CSS следующее:

p {

color: red;

}

Давайте поместим эти три строки CSS в новый файл в любом текстовом редакторе (например Notepad++), а затем сохраним файл как style.css в папке styles.

Чтобы применять CSS к нашему HTML-документу, вставьте следующую строку в шапку, то есть между тегами ***<head>*** и ***</head>***:

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

Пример: Внешняя таблица стилей

**Результат**

**HTML-код**

**Попробуй сам »**

Текст первый

Текст второй

Текст третий

Внутренняя таблица стилей

Данный стиль определяется в самом HTML-документе и обычно располагается в заголовке веб-страницы **HEAD**. По своей гибкости и возможностям этот способ подключения стилей уступает предыдущему, но также позволяет размещать все стили в одном месте. В данном случае, стили разполагаются прямо в теле HTML-документа. Вы можете включить правила CSS в НТМL-страницу, поместив их внутри элемента ***<style>***, который обычно находится в элементе ***<head>***, но фактически может быть помещен в любом месте документа. Этих тегов на странице может быть несколько.

Тег ***<style>*** позволяет записывать внутри себя код в формате CSS:

<style type="text/css">

body {

background-color:palegreen;

}

h1{

color: blue;

font-family:verdana;

}

p{

font-size:20px;

color:red;

}

</style>

Пример: Внутренняя таблица стилей

**Результат**

**HTML-код**

**Попробуй сам »**

***Заголовок***

Текст первый

Текст второй

Текст третий

В данном примере мы с помощью CSS установили цвет фона для элемента ***<body>***: **background-color:palegreen**, цвет и тип шрифта для заголовков ***<h1>***: **color: blue; font-family:verdana**, а также размер шрифта, цвет и выравнивание текста по центру для параграфов ***<p>***: **font-size:20px; color:red; text-align:center**.

Встроенный стиль

Когда необходимо отформатировать отдельный элемент HTML-страницы, описание стиля можно расположить непосредственно внутри открывающего тега при помощи уже специализированного атрибута style. Например:

<p style="color:red;">Параграф</p>

Такие стили называют встроенными (inline), или внедренными. Правила, определенные непосредственно внутри открывающего тега элемента перекрывают правила, определенные во внешнем файле CSS, а также правила, определенные в элементе ***<style>***.

В следующем примере к HTML-документу подключены все три рассмотренные стиля форматирования:

Пример: Приоритетность стилей

**Результат**

**HTML-код**

**Попробуй сам »**

***Заголовок***

Текст первый

Текст второй

Текст третий

|  |  |
| --- | --- |
|  | Чем ближе описание стиля находится к элементу, тем более высокий приоритет имеет этот стиль при выборе браузером конечного правила оформления. |

7.8 В чем состоит свойство наследования стилевых назначений каскадных таблиц стилей?

Цель этого урока — углубить ваше понимание некоторых основополагающих концепций CSS — каскадов, спецификаций и наследования, — которые контролируют то, как CSS применяется к HTML и как разрешаются конфликты.

Хотя изучение этого урока может показаться менее актуальным и немного более академичным, чем некоторые другие части курса, понимание этих вещей спасёт вас от головной боли в дальнейшем! Мы рекомендуем вам внимательно изучить этот раздел и убедиться, что вы понимаете концепции, перед тем, как двигаться дальше.

|  |  |
| --- | --- |
| **Необходимые условия:** | Базовая компьютерная грамотность, Установка базового ПО, базовые знания работы с файлами, основы HTML (Введение в HTML), и общее представление о том, как работает CSS (Введение в CSS.) |
| **Цель:** | Изучить понятие о каскаде и специфичности, и как работает наследование CSS. |

Конфликтующие правила

CSS (Cascading Style Sheets) означает Каскадные Таблицы Стилей и первое слово "каскадные" является невероятно важным для понимания: то, как ведёт себя каскад — ключевой момент в понимании CSS.

В какой-то момент, работая над проектом, вы обнаружите, что CSS, который, по-вашему, должен быть применён к элементу, не работает. Обычно проблема заключается в том, что вы создали два правила, которые могут потенциально применяться к одному и тому же элементу. **Каскад** и тесно связанная концепция **специфичности** — это механизмы, которые контролируют, какое именно правило применяется, когда имеется такой конфликт. Стиль вашего элемента может определять не то правило, на которое вы рассчитывали, поэтому вам необходимо понимать, как работают эти механизмы.

Также значимой является концепция **наследования,**которая заключается в том, что некоторые свойства CSS наследуют по умолчанию значения, установленные для родительского элемента текущего элемента, а некоторые не наследуют. Это также может стать причиной поведения, которое вы, возможно, не ожидаете.

Давайте начнём с краткого обзора ключевых моментов, которых мы касаемся, далее рассмотрим каждый из них по очереди и посмотрим, как они взаимодействуют друг с другом и с вашим CSS. Это может показаться набором сложных для понимания понятий. Однако, когда вы получите больше опыта в написании CSS, для вас станет более очевидным то, как это работает.

Каскад

Каскад таблицы стилей, если говорить упрощённо, означает, что порядок следования правил в CSS имеет значение; когда применимы два правила, имеющие одинаковую специфичность, используется то, которое идёт в CSS последним.

В приведённом ниже примере у нас есть два правила, которые могут применяться к h1. В результате h1 окрасится синим цветом — эти правила имеют идентичный селектор и, следовательно, одинаковую специфичность, поэтому побеждает последний в порядке следования.

Специфичность

Специфичность определяет, как браузер решает, какое именно правило применяется в случае, когда несколько правил имеют разные селекторы, но, тем не менее, могут быть применены к одному и тому же элементу. Различные типы селекторов ( селекторы элементов *h1{...}*, селекторы классов, селекторы идентификаторов и т.д ) имеют разной степени влияние на элементы страницы. Чем более общее влияние оказывает селектор на элементы страницы тем меньше его специфичность, конкретность. По существу, это мера того, насколько специфическим будет отбор по селектору:

Селектор элементов ( например *h1* ) менее специфичен — он выберет все элементы этого типа на странице — поэтому получит меньше баллов.

Селектор класса более специфичен — он выберет только те элементы на странице, которые имеют конкретное значение атрибута class — поэтому получит больше баллов, селектор класса применится после селектора элемента и поэтому перекроет его стили.

Например. Как указано ниже, у нас опять есть два правила, которые могут применяться к h1. h1 в результате будет окрашен красным цветом — селектор класса даёт своему правилу более высокую специфичность, поэтому он будет применён, несмотря на то, что правило для селектора элемента расположено ниже в таблице стилей.

Позже мы объясним, как сделать оценку специфичности, и прочие детали.

Наследование

Наследование также надо понимать в этом контексте — некоторые значения свойства CSS, установленные для родительских элементов наследуются их дочерними элементами, а некоторые нет.

Например, если вы установили значение color и font-family для элемента, то каждый элемент внутри него также будет иметь этот цвет и шрифт, если только вы не применили к ним напрямую стиль с другим цветом и шрифтом.

Некоторые свойства не наследуются — например, если вы установили для элемента width равным 50%, все его дочерние элементы не получат ширину в 50% от ширины своего родительского элемента. Если бы это было так, CSS было бы чрезвычайно трудно использовать!

**Примечание**: На страницах справочника CSS-свойств вы можете найти окно технической информации, обычно в конце раздела спецификации, в котором перечислены некоторые технические данные об этом свойстве, в том числе наследуется оно или нет. Например, здесь: color property Specifications section.

Понимание взаимодействия этих концепций

Эти три концепции вместе определяют, какая CSS применяется и к какому элементу; в следующих разделах мы увидим, как они взаимодействуют. Это может показаться сложным, но вы начнёте лучше понимать их по мере приобретения опыта работы с CSS, и вы всегда можете обратиться к справочной информации, если что-то забыли. Даже опытные разработчики не помнят всех деталей!

Видео ниже показывает, как вы можете использовать Firefox DevTools для проверки каскада стилей, спецификации, и т.д. на странице:

Понимание наследования

Итак, наследование. В примере ниже мы имеем <ul> с двумя уровнями неупорядоченных списков, вложенных в него. Мы установили для внешнего <ul> стиль границы, внутренние отступы и цвет шрифта.

Цвет шрифта применён к прямому потомку, но также и к непрямому потомку — к прямому потомку <li> и к элементам внутри первого вложенного списка. Далее мы добавили класс special ко второму вложенному списку и применили к нему другой цвет шрифта. Теперь это свойство наследуется всеми его потомками.

Такие свойства, как ширина (как в примере выше), внутренние и внешние отступы и стиль границы не наследуются. Если бы потомки нашего списка наследовали бы границу, то каждый отдельный список и каждая позиция в списке получили бы такую же границу — вряд ли мы хотели бы получить такой эффект!

Какие свойства наследуются по умолчанию, а какие нет, чаще всего определяется здравым смыслом.

Контроль наследования

CSS предоставляет четыре специальных универсальных значения свойства для контроля наследования. Каждое свойство CSS принимает эти значения.

inherit

Устанавливает значение свойства, применённого к элементу, таким же, как у его родительского элемента. Фактически, это "включает наследование".

initial

Устанавливает значение свойства, применённого к выбранному элементу, равным initial value этого свойства (в соответствии с настройками браузера по умолчанию. Если в таблице стилей браузера отсутствует значение этого свойства, оно наследуется естественным образом.)

unset (en-US)

Возвращает свойству его естественное значение, что означает, что если свойство наследуется естественным образом, оно действует как inherit, иначе оно действует как initial.

**Примечание**: Существует также более новое значение revert, которое имеет ограниченную поддержку браузерами.

**Примечание**: Смотрите Origin of CSS declarations в [Page not yet written] для более подробной информации о каждом из них, и о том, как они работают.

Можно посмотреть список ссылок и изучить, как работают универсальные значения. Пример, следующий ниже, позволяет вам поиграть с CSS и увидеть, что происходит, когда вы вносите изменения. Подобные эксперименты с кодом — лучший способ освоить HTML и CSS.

Например:

Второй элемент списка имеет класс my-class-1. Таким образом, цвет для следующего вложенного элемента a устанавливается по наследству. Как изменится цвет, если это правило будет удалено?

Понятно ли, почему третий и четвёртый элементы a имеют именно такой цвет? Если нет, перечитайте описание значений, представленное выше.

Какая из ссылок изменит цвет, если вы зададите новый цвет для элемента <a> — например: a { color: red; }?

Возврат всех исходных значений свойств

Стенографическое свойство CSS all можно использовать для того, чтобы присвоить одно из значений наследования к (почти) всем свойствам одновременно. Это одно из четырёх значений (inherit, initial, unset, или revert). Это удобный способ для отмены изменений, внесённых в стили, для того, чтобы вы могли вернуться к стартовой точке перед внесением новых изменений.

В примере ниже имеются два блока <blockquote>. Первый имеет стиль, который применён к самому элементу blockquote, второй имеет класс fix-this, который устанавливает значение all в unset.

Попробуйте установить для all ещё одно из доступных значений и исследуйте, в чём заключается разница.

Понимание каскада

Теперь мы понимаем, почему параграф, следующий по глубине в структуре HTML документа, имеет тот же цвет, что CSS применяет к body, а вводные уроки дали понимание того, как изменить применение CSS к чему-либо в любой точке документа — или назначить CSS элементу, или создать класс. Теперь рассмотрим подробнее то, как каскад определяет выбор CSS-правил, применяемых в случае влияния на стиль элемента нескольких объектов.

Вот три фактора, перечисленные в порядке возрастания важности. Следующий отменяет предыдущий.

**Порядок следования**

**Специфичность**

**Важность**

Мы внимательно изучим их, чтобы увидеть, как именно браузеры определяют, какой CSS следует применить.

Порядок следования

Мы уже видели, какое значение для каскада имеет порядок следования. Если у вас несколько правил, которые имеют одинаковую важность, то побеждает правило, которое идёт последним в CSS. Другими словами, правила, более близкие к самому элементу, переписывают более ранние, пока последнее не победит, оно и стилизует элемент.

Специфичность

Понимая, что порядок следования правил имеет значение, в какой-то момент вы окажетесь в ситуации, когда вы знаете, что правило появляется позже в таблице стилей, но применяется более раннее, конфликтующее правило. Это связано с тем, что более раннее правило имеет более **высокую специфичность** — оно более специфично и поэтому выбирается браузером как правило, которое должно стилизовать элемент.

Как мы видели ранее в этом уроке, селектор класса имеет больший вес, чем селектор элемента, поэтому свойства, определённые в классе, будут переопределять свойства, применённые непосредственно к элементу.

Здесь следует отметить, что, хотя мы думаем о селекторах и правилах, применяемых к объекту, который они выбирают, переписывается не всё правило, а только свойства, которые являются одинаковыми.

Такое поведение помогает избежать повторения в вашем CSS. Обычной практикой является определение общих стилей для базовых элементов, а затем создание классов для тех, которые отличаются. Например, в таблице стилей ниже мы определяем общие стили для заголовков второго уровня, а затем создаём несколько классов, которые изменяют только некоторые свойства и значения. Определённые вначале значения применяются ко всем заголовкам, затем к заголовкам с классами применяются более конкретные значения.

Давайте теперь посмотрим, как браузер будет вычислять специфичность. Мы уже знаем, что селектор элемента имеет низкую специфичность и может быть перезаписан классом. По существу, значение в баллах присуждается различным типам селекторов, и их сложение даёт вам вес этого конкретного селектора, который затем может быть оценён в сравнении с другими потенциальными соперниками.

Степень специфичности, которой обладает селектор, измеряется с использованием четырёх различных значений (или компонентов), которые можно представить как тысячи, сотни, десятки и единицы — четыре однозначные цифры в четырёх столбцах:

**Тысячи**: поставьте единицу в эту колонку, если объявление стиля находится внутри атрибута style (встроенные стили). Такие объявления не имеют селекторов, поэтому их специфичность всегда просто 1000.

**Сотни**: поставьте единицу в эту колонку за каждый селектор ID, содержащийся в общем селекторе.

**Десятки**: поставьте единицу в эту колонку за каждый селектор класса, селектор атрибута или псевдокласс, содержащийся в общем селекторе.

**Единицы**: поставьте общее число единиц в эту колонку за каждый селектор элемента или псевдоэлемент, содержащийся в общем селекторе.

**Примечание:** Универсальный селектор (\*), комбинаторы (+, >, ~, '') и псевдокласс отрицания (:not) не влияют на специфичность.

Следующая таблица показывает несколько несвязанных примеров, которые помогут вам разобраться. Посмотрите их все и убедитесь, что вы понимаете, почему они обладают той специфичностью, которую мы им дали. Мы ещё не рассмотрели селекторы детально, но вы можете найти подробную информацию о каждом селекторе в справочнике селекторов MDN.

| **Селектор** | **Тысячи** | **Сотни** | **Десятки** | **Единицы** | **Специфичность** |
| --- | --- | --- | --- | --- | --- |
| h1 | 0 | 0 | 0 | 1 | 0001 |
| h1 + p::first-letter | 0 | 0 | 0 | 3 | 0003 |
| li > a[href\*="en-US"] > .inline-warning | 0 | 0 | 2 | 2 | 0022 |
| #identifier | 0 | 1 | 0 | 0 | 0100 |
| Без селектора, с правилом внутри атрибута style элемента. | 1 | 0 | 0 | 0 | 1000 |

Прежде чем мы продолжим, давайте посмотрим на пример в действии.

Так что здесь происходит? Прежде всего, нас интересуют только первые семь правил этого примера, и, как вы заметите, мы включили их значения специфичности в комментарий перед каждым правилом.

Первые два правила конкурируют за стилизацию цвета фона ссылки — второе выигрывает и делает фоновый цвет синим, потому что у него есть дополнительный селектор ID в цепочке: его специфичность 201 против 101.

Третье и четвёртое правило конкурируют за стилизацию цвета текста ссылки — второе выигрывает и делает текст белым, потому что, хотя у него на один селектор элемента меньше, отсутствующий селектор заменяется на селектор класса, который оценивается в десять вместо единицы. Таким образом, приоритетная специфичность составляет 113 против 104.

Правила 5–7 соревнуются за определение стиля границы ссылки при наведении курсора. Шестой селектор со специфичностью 23 явно проигрывает пятому со специфичностью 24 — у него в цепочке на один селектор элемента меньше. Седьмой селектор, однако, превосходит как пятый, так и шестой — он имеет то же количество подселекторов в цепочке, что и пятый, но один элемент заменён селектором класса. Таким образом, приоритетная специфичность 33 против 23 и 24.

**Примечание**: Это был условный пример для более простого усвоения. В действительности, каждый тип селектора имеет собственный уровень специфичности, который не может быть замещён селекторами с более низким уровнем специфичности. Например, миллион соединённых селекторов **класса** не способны переписать правила одного селектора **id**.

Более правильный способ вычисления специфичности состоит в индивидуальной оценке уровней специфичности, начиная с наивысшего и продвигаясь к самому нижнему, когда это необходимо. Только когда оценки уровня специфичности совпадают, следует вычислять следующий нижний уровень; в противном случае, вы можете пренебречь селекторами с меньшим уровнем специфичности, поскольку они никогда не смогут преодолеть уровни более высокой специфичности.

!important

Существует специальный элемент CSS, который вы можете использовать для отмены всех вышеперечисленных вычислений, однако вы должны быть очень осторожны с его использованием — !important. Он используется, чтобы сделать конкретное свойство и значение самыми специфичными, таким образом переопределяя нормальные правила каскада.

Взгляните на этот пример, где у нас есть два абзаца, один из которых имеет ID.

Давайте пройдёмся по этому примеру, чтобы увидеть, что происходит — попробуйте удалить некоторые свойства, чтобы увидеть, что получится, если вам трудно понять:

Вы увидите, что применены значения color (en-US) и padding третьего правила, но background-color — нет. Почему? Действительно, все три безусловно должны применяться, потому что правила, более поздние в порядке следования, обычно переопределяют более ранние правила.

Однако вышеприведённые правила выигрывают, потому что селекторы классов имеют более высокую специфичность, чем селекторы элементов.

Оба элемента имеют class с названием better, но у второго также есть id с названием winning. Поскольку ID имеют ещё более высокую специфичность, чем классы (у вас может быть только один элемент с каждым уникальным ID на странице, но много элементов с одним и тем же классом — селекторы ID очень специфичны, на что они и нацелены), красный цвет фона и однопиксельная чёрная граница должны быть применены ко 2-му элементу, причём первый элемент получает серый фоновый цвет и отсутствие границы, как определено классом.

2-й элемент получил красный цвет фона и отсутствие границы. Почему? Из-за объявления !important во втором правиле — размещение которого после border: none означает, что это объявление перевесит значение границы в предыдущем правиле, даже если ID имеет более высокую специфичность.

**Примечание**: Единственный способ переопределить объявление !important – это включить другое объявление !important в правило с такой же специфичностью позже или в правило с более высокой специфичностью.

Полезно знать о существовании !important, чтобы вы понимали, что это такое, когда встретите в чужом коде. **Тем не менее, мы настоятельно рекомендуем вам никогда не использовать его, если в этом нет острой необходимости.** !important меняет обычный порядок работы каскада, поэтому он может серьёзно затруднить отладку проблем CSS, особенно в большой таблице стилей.

Одна из ситуаций, в которой вам, возможно, придётся это использовать, — это когда вы работаете с CMS, где вы не можете редактировать модули CSS ядра, и вы действительно хотите переопределить стиль, который нельзя переопределить другим способом. Но, вообще говоря, не стоит использовать этот элемент, если можно этого избежать.

Влияние расположения CSS

Наконец, также полезно отметить, что важность объявления CSS зависит от того, в какой таблице стилей оно указано — у пользователя есть возможность установить индивидуальные таблицы стилей для переопределения стилей разработчика, например, пользователь может иметь проблемы со зрением и захочет установить размер шрифта на всех посещаемых им веб-страницах в два раза больше нормального размера, чтобы облегчить чтение.

Подведение итогов

Конфликтующие объявления будут применяться в следующем порядке, с учётом замены более ранних более поздними:

Объявления в таблицах стилей клиентского приложения (например, стили браузера по умолчанию, используемые, когда не заданы другие стили).

Обычные объявления в пользовательских таблицах стилей (индивидуальные стили устанавливаются пользователем).

Обычные объявления в авторских таблицах стилей (это стили, установленные нами, веб-разработчиками).

Важные объявления в авторских таблицах стилей.

Важные объявления в пользовательских таблицах стилей.

Для таблиц стилей веб-разработчиков имеет смысл переопределить пользовательские таблицы стилей так, чтобы можно было сохранить запланированный дизайн, но иногда у пользователей есть веские причины для переопределения стилей веб-разработчика, как упомянуто выше — это может быть достигнуто с помощью использования !important в их правилах.