## Наследование и каскадирование

|  |
| --- |
| [Иерархическое дерево](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/1) |
| [Наследование](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/2) |
| [Наследование «на пальцах»](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/3) |
| [Еще немного про наследование](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/4) |
| [Наследуемые свойства](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/5) |
| [Ненаследуемые свойства](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/6) |
| [Принудительное наследование](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/7) |
| [Каскадирование](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/8) |
| [Битва за курочку](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/9) |
| [Битва за курочку. Раунд второй](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/10) |
| [Битва за курочку. Раунд третий](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/11) |
| [Битва за курочку. Борьба накаляется](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/12) |
| [Битва за курочку. Запрещённый приём](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/13) |
| [Еще одна задачка на специфичность](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/14) |
| [Расчет значения специфичности](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/15) |
| [Еще задачка на порядок кода](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/16) |
| [Перекрестное наследование](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/17) |
| [Испытание: взломанный котопрофайл](https://npoed.htmlacademy.ru/htmlcss110/course/9/run/18) |

# Иерархическое дерево [1/18]

HTML-документ представляет собой иерархическое дерево. Это означает, что у каждого элемента (кроме корневого) есть только один родитель, т.е. элемент, внутри которого он располагается. У корневого раздела родитель отсутствует. Рассмотрим простейшую страницу:

<html>

<head></head>

<body>

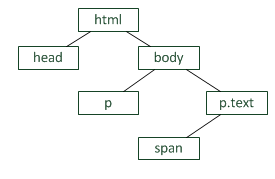
<p>Текст документа</p>

<p class="text">Выделенная <span>строка</span></p>

</body>

</html>

Для этой страницы можно нарисовать такое иерархическое дерево:



Оно схематически отображает структуру вложенности элементов. В данном примере видно, что у элемента span родителем является p.text, а у p.text родитель — body.

Иерархическая структура документа определяет основы концепции наследования.

# Наследование [2/18]

Наследование в CSS — механизм, с помощью которого значения свойств элемента-родителя передаются его элементам-потомкам.

Стили, присвоенные некоторому элементу, наследуются всеми потомками (вложенными элементами), если они не переопределены явно. Например, размер шрифта и его цвет достаточно применить к body, чтобы все элементы внутри имели те же свойства.

Наследование позволяет сократить размер таблицы стилей, но если стилей много, то отследить какой родительский элемент установил некоторое свойство, становится сложнее.

# Наследование «на пальцах» [3/18]

Давайте на простых примерах подробнее разберёмся, в чём же преимущество наследования.

Рассмотрим пример:

<p class="text">Cтрока c выделенным <span>словом</span></p>

Представим, что нам нужно установить красный цвет шрифта для всего текста. Зададим CSS-свойства следующим образом:

.text{

color:red;

}

Благодаря наследованию цвет шрифта в теге span автоматически станет красным:

Cтрока c выделенным словом

А так бы выглядел результат, если бы наследование не работало:

Cтрока c выделенным словом

Нам пришлось бы отдельно прописывать цвет шрифта для тега span. И тогда установка таких простых свойств как стиль шрифта стала бы большой проблемой: нужно было бы задавать свойства для всех возможных вложенных тэгов.

# Еще немного про наследование [4/18]

Наверняка вы обращали внимание, что не все свойства наследуются тегами-потомками от их родителей.

Действительно, было бы странно, если бы свойство border автоматически устанавливалось для всех вложенных элементов.

Например, для этого куска кода:

<p class="bordered">Cтрока c выделенным <span>словом</span></p>

Установим CSS-свойство:

.bordered{

border:1px solid green;

}

Если бы наследовались все свойства, то результат бы выглядел так:

Cтрока c выделенным словом

На самом деле граница будет нарисована только у тега p.

О том, какие именно свойства наследуются, мы расскажем в следующих заданиях.

# Наследуемые свойства [5/18]

К наследуемым свойствам относятся в первую очередь свойства, определяющие параметры отображения текста:

font-size, font-family, font-style, font-weight, color, text-align, text-transform, text-indent, line-height, letter-spacing, word-spacing, white-space, direction и т. д.

Также к наследуемым свойствам относятся list-style, cursor, visibility, border-collapse и некоторые другие. Но они используются значительно реже.

Весь список наследуемых свойств смотрите в [стандарте CSS](http://www.w3.org/TR/CSS21/propidx.html). Значение yes в колонке Inherited?.

Эти свойства можно и нужно задавать через предков, следуя семантике документа.

Например, параметры текста зачастую не меняются в пределах отдельных блоков страницы: меню, основного содержания, информационных панелей. Поэтому общие параметры текста (цвет, размер, гарнитура) обычно указывают в стилях самих блоков.

# Ненаследуемые свойства [6/18]

В предыдущем задании мы перечислили основные наследуемые свойства. Все остальные относятся к ненаследуемым. Это параметры позиционирования, размеров, отступов, фона, рамок и т. д.

А именно: background, border, padding, margin, width, height, position и др.

Весь список ненаследуемых свойств смотрите в [стандарте CSS](http://www.w3.org/TR/CSS21/propidx.html). Значение no в колонке Inherited?.

Не наследуются они из соображений здравого смысла. Например: если для какого-либо блока установлен внутренний отступ, автоматически выставлять такой же отступ каждому вложенному элементу нет никакой надобности. Эти параметры чаще всего уникальны для каждого отдельного блока.

# Принудительное наследование [7/18]

Для каждого свойства может быть задано значение inherit.

Оно означает, что данное свойство принимает такое же значение, как и у родительского элемента. Значение inherit может быть использовано для усиления наследуемых значений, а также в свойствах, которые обычно не наследуются.

Запись выглядит следующим образом:

p{

background:inherit;

}

В данном случае у тегов p свойство background будет таким же, как и у их родительских тегов.

# Каскадирование [8/18]

CSS расшифровывается как «Cascading Style Sheets» или «каскадные таблицы стилей».

Каскадность обозначает, что к одному и тому же элементу может применяться несколько CSS-правил (наборов CSS-свойств). Среди этих свойств могут быть и конфликтующие между собой. Поэтому существуют инструкции, которые определяют, каким будет финальный набор свойств элемента.

Например, для элемента:

<p class="text" style="color: red;"></p>

CSS-правила существуют как минимум в трёх разных местах:

1. в подключаемом файле style.css для селекторов p или .text;
2. в атрибуте style;
3. в стандартных стилях отображения, встроенных в браузер.

Каскадирование как раз и определяет, какие именно свойства из этих источников применятся к данному абзацу.

Имеется три основные концепции, управляющие порядком, в котором применяются CSS-свойства:

1. важность;
2. специфичность;
3. порядок исходного кода.

Самыми важными для нас являются последние две и подробнее о них мы поговорим отдельно.

# Битва за курочку [9/18]

Перед тем как разбирать сложные правила работы каскадирования, расчёта специфичности и определения приоритетов, потренируемся на котиках.

Разберём пример. Вот HTML-код, в котором есть абзац с двумя классами:

<p class="red blue">Синий или красный?</p>

А вот CSS-код c двумя правилами для этих классов:

.blue{

color:blue;

}

.red{

color:red;

}

Вопрос: какого цвета будет текст абзаца? Какое из CSS-правил приоритетнее?

Ответ: красного цвета, второе правило приоритетнее.

Это происходит потому, что селекторы у правил одинакового типа, и соответственно обладают одинаковой специфичностью. В таком случае более высоким приоритетом обладает то правило, которое расположено в CSS-коде ниже. В нашем случае это правило для класса red.

В этой серии заданий вам нельзя менять значения CSS-свойств, но можно изменять селекторы или менять порядок CSS-правил в коде.

# Битва за курочку. Раунд второй [10/18]

Чуть более сложный пример. Тот же HTML:

<p class="red blue">Синий или красный?</p>

Немного другой CSS:

p.blue{

color:blue;

}

.red{

color:red;

}

В этом случае текст абзаца будет синим. Происходит это потому, что селектор p.blueболее специфичный, чем селектор .red.

Простое объяснение специфичности звучит так:

Чем меньшее количество элементов потенциально может выбрать селектор, тем он специфичнее.

В нашем примере селектор .red выберет все тэги с нужным классом, а селектор p.blue выберет только абзацы с нужным классом.

Заметьте, что в этом задании одно из CSS-правил вынесено в заблокированный HTML-код, а вам нужно победить его, усилив другое CSS-правило.

# Битва за курочку. Раунд третий [11/18]

А теперь посмотрим, как ведут себя контекстные селекторы.

<div class="experiment">

<p class="red blue">Синий или красный?</p>

</div>

И в CSS используем контекстный селектор для второго правила:

p.blue{

color:blue;

}

.experiment .red{

color:red;

}

Какой теперь будет цвет текста?

Кексик и Рудольф помогут разобраться, какой селектор специфичнее.

# Битва за курочку. Борьба накаляется [12/18]

Как вы уже знаете, существуют селекторы не только по классам, но и по id. Они начинаются с решётки #.

HTML:

<div id="experiment-1" class="experiment">

<p class="red blue">Синий или красный?</p>

</div>

CSS:

#experiment-1 .blue{

color:blue;

}

.experiment .red{

color:red;

}

Особенность атрибута id заключается в том, что его значение должно быть уникальным в пределах страницы. То есть, может существовать только один тэг с определенным значением id.

Получается, что селектор по id может выбрать только один элемент. И поэтому он на порядок специфичнее селекторов по тэгам, классам, а также комбинаций этих селекторов.

Кексик и Рудольф демонстрируют его работу.

# Битва за курочку. Запрещённый приём [13/18]

Битва подходит к концу, и чтобы удержать победу, Рудольф решается на запрещённый приём:

Он прописывает стили с помощью атрибута style прямо в тэге курочки.

CSS-правила, которые прописаны в style обладают наивысшим приоритетом. Такой способ задания стилей не приветствуется в профессиональной вёрстке сайтов и годится только для создания быстрых прототипов. Поэтому мы и назвали этот приём запрещённым.

Однако существует способ переопределить из подключаемых CSS-файлов даже стили, заданные в атрибуте style. Для этого нужно использовать ключевое слово !important. Оно задаёт CSS-свойству усиленный приоритет. Вот пример:

HTML:

<p style="color:red;" class="blue">Синий или красный?</p>

CSS:

.blue{

color:blue !important;

}

Цвет текста в этом примере будет синим.

При вёрстке не рекомендуется часто использовать !important. По возможности старайтесь обходиться без него.

# Еще одна задачка на специфичность [14/18]

Как вы уже догадались, самым главным механизмом для определения приоритетов стилей является специфичность. Поэтому давайте еще немного потренируемся работать с ней.

Рассмотрим пример: на полу в коробке сидит кот

<div id="floor"><span class="cat-in-box">Кексик</span></div>

Допустим в стилях существуют следующие определения:

span{

background-color:#27ae60; /\* Зеленый \*/

}

div span{

background-color:#2980b9; /\* Синий \*/

}

#floor .cat-in-box{

background-color: #34495e; /\* Мокрый асфальт \*/

}

.cat-in-box{

background-color: #8e44ad; /\* Фиолетовый \*/

}

#floor span{

background-color: #c0392b; /\* Красный \*/

}

div .cat-in-box{

background-color: #e67e22; /\* Оранжевый \*/

}

А теперь вопрос на засыпку: какого цвета будет коробка? Сначала сделайте предположение, а затем проверьте.

Почему именно такой и как это определяется мы расскажем в следующем задании.

# Расчет значения специфичности [15/18]

Вы вдоволь наигрались со специфичностью, а теперь пришло время изучить полные правила её вычисления.

Специфичность селектора разбивается на 4 группы — a, b, c, d:

* если стиль встроенный, т.е. определен как style="...", то а=1, иначе a=0;
* значение b равно количеству идентификаторов (тех, которые начинаются с #) в селекторе;
* значение c равно количеству классов, псевдоклассов и селекторов атрибутов;
* значение d равно количеству селекторов типов элементов и псевдо-элементов.

После этого полученное значение приводится к числу (обычно в десятичной системе счисления). Селектор, обладающий большим значением специфичности, обладает и большим приоритетом.

Посчитаем специфичность в нашем примере:

|  |  |  |
| --- | --- | --- |
| **Селектор** | **a,b,c,d** | **Число** |
| span | 0,0,0,1 | 1 |
| div .cat-in-box | 0,0,1,1 | 11 |
| **#floor .cat-in-box** | **0,1,1,0** | **110** |
| div span | 0,0,0,2 | 2 |
| .cat-in-box | 0,0,1,0 | 10 |
| #floor span | 0,1,0,1 | 101 |

Остюда сразу видно, что в нашем примере самым приоритетным является селектор #floor .cat-in-box.

# Еще задачка на порядок кода [16/18]

Здесь все просто.

Если два CSS-правила применяются к одному и тому же элементу и имеют одинаковую специфичность, то более приоритетным будет то правило, которое появится в коде позже другого.

С этим фактом вы уже познакомились на битве за курочку. А сейчас просто закрепим этот материал, выполнив несколько простейших головоломок на изменение порядка кода.

# Перекрестное наследование [17/18]

При создании стилей для сходных по внешнему виду или функциональности элементов, которые могут использоваться на странице неоднократно, очень удобно пользоваться перекрёстным наследованием.

Приём этот заключается в следующем:

1. создается базовый стиль для таких элементов;
2. определяются вспомогательные стили, которые применяются к элементам по мере надобности;
3. элемент наследует базовый стиль и один или несколько вспомогательных.

**Пример:**

На странице используются кнопки разного назначения: для отправки форм, для сброса информации в полях формы, как элементы навигации и т.д.

Можно вынести общее оформление (размеры, отступы и т.д.) для всех кнопок в отдельное CSS-правило для класса, например, .button.

А затем создать дополнительные CSS-правила, в которых будут определены только различающиеся свойства этих кнопок, например, цвет фона. Для этих правил можно использовать такие названия классов: .button-send, .button-clear, .button-navigation.

Каждая кнопка в HTML-коде будет иметь два класса: общий и дополнительный.

<a class="button button-send">Отправить</a>