HTML CSS basics

Каскад стилей

В CSS все стили идут каскадом сверху вниз, что позволяет добавлять новые стили или переписывать уже имеющиеся.

Скажем, мы назначили всем абзацам оранжевый цвет шрифта и размер шрифта в 24 пикселя:

```
p {
    color: orange;
    font-size: 24px;
}
```

А ниже переназначили цвет, изменив значение на зеленый:

```
p { color: green; }
```

Поскольку стиль абзаца, который устанавливает зелёный цвет шрифта, располагается ниже стиля абзаца, который задаёт оранжевый цвет, он будет иметь приоритет в каскаде.

Все абзацы будут иметь зеленый цвет шрифта. Размер шрифта останется 24 пикселя, потому что второй стиль абзаца не определил новый размер шрифта.

Каскад свойств

Каскад так же работает со свойствами внутри отдельных стилей.

Скажем, мы назначили всем абзацам оранжевый цвет шрифта,
а ниже, в том же стиле, переназначили цвет, изменив значение на зеленый:

```
p {
    color: orange;
    color: green;
}
```

Поскольку объявление зелёного цвета шрифта написано после объявления оранжевого цвета, как и прежде, наши абзацы будут иметь зеленый цвет шрифта.

Приоритетность

Каждый селектор в CSS имеет степень приоритетности.

Эта степень + позиция в каскаде определяют отображение стилей.

Мы помним, что существуют селекторы типа, класса и идентификаторы.
Каждый из этих селекторов имеет различную приоритетность.

У селектора типа низкая степень приоритетности.

У селектора класса средняя степень приоритетности.

У идентификаторов высокая степень приоритетности.

Конфликты стилей

Чем выше степень приоритетности селектора, тем больше первенства отдаётся его стилю при возникновении конфликта.
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.

HTML

```
<p id=“important">...</p>
```

CSS

```
#important { color: red; }
p { color: green; }
```

Итак, у нас есть элемент абзаца со значением атрибута id — important.
В CSS этот абзац выбирается двумя различными типами селекторов: селектором типа и идентификатором.

Несмотря на то, что селектор типа указан после идентификатора в каскаде, идентификатор имеет приоритет над селектором типа, следовательно, абзац будет иметь красный цвет шрифта.

Комбинация селекторов

Комбинируя селекторы мы можем конкретизировать, какой элемент или группу элементов мы хотели бы выбрать.

Скажем, мы хотим выбрать все элементы абзаца, которые находятся внутри элемента со значением атрибута класса hotdog и установить для них коричневый цвет шрифта. Однако, если один из этих абзацев содержит значение атрибута класса mustard, мы хотим установить для его шрифта желтый цвет.

HTML

```
<div class="hotdog">
        <p>...</p>
        <p class="mustard">...</p>
</div>
```

CSS

```
.hotdog p { color: brown; }
.hotdog p.mustard { color: yellow; }
```

Когда селекторы комбинируются они должны читаться справа налево. Самый крайний селектор справа, непосредственно перед открытой скобкой, известен как ключевой селектор. Он определяет, к каким именно элементам будут применяться стили.
Любой селектор слева от ключевого будет служить уточнением.

Пробелы в селекторах

В предыдущем комбинированном селекторе **.hotdog p.mustard** есть пробел между классом hotdog и селектором абзаца, но не между селектором абзаца и классом mustard.

Использование пробелов и отказ от них — это большая разница в селекторах.

Отсутствие пробела между селектором абзаца и классом mustard значит, что будут выбраны только абзацы с классом mustard. Если бы селектор абзаца был удалён, а класс mustard содержал пробелы с двух сторон, то был бы выбран любой элемент с классом mustard, а не только абзацы.
Если бы был пробел после селектора абзаца, то выбирались бы любые элементы с классом mustard внутри элемента p.

Лучше всего не писать селектор типа перед селектором класса.
Как правило, мы хотим выбрать любой элемент с данным классом, а не только один тип элемента.

С учётом этого наш новый комбинированный селектор будет писаться как **.hotdog .mustard**

Добавочные классы

Элементы в **HTML** могут содержать более одного атрибута **class,** при этом их значения разделяются пробелами.
За счёт этого мы можем применить некоторые стили ко всем элементам одного вида, а другие стили к конкретным элементам этого же вида.

Мы можем связать стили, которые хотим постоянно повторять с одним классом и разделить на дополнительные стили с другим классом.

Скажем, мы хотим, чтобы у всех кнопок был размер шрифта 16 пикселей, но чтобы цвет их фона варьироваться в зависимости от функциональности.
Мы можем создать несколько классов и распределить их по элементам, в зависимости от применения желаемых стилей.

HTML

```
<a class="btn btn-danger">...</a>
<a class="btn btn-success">...</a>
```

CSS

```
.btn { font-size: 16px; }
.btn-danger { background: red; }
.btn-success { background: green; }
```

Позиционирование содержимого

Стили CSS дают нам возможность позиционировать элементы и содержимое на странице практически любым мыслимым образом.

Это вносит структурность в наш дизайн и помогает сделать контент более наглядным.

Позиционирование с помощью float

Один из способов позиционирования элементов на странице — через свойство **float**

Это свойство довольно универсально, хотя разработано было в целях установки обтекаемости объекта текстом.

Свойство **float** убирает элемент из его обычного потока в документе и позиционирует слева или справа от родительского элемента. Все остальные элементы на странице будут его обтекать. Например, абзацы будут обтекать изображение, если для элемента **<img>** установлено свойство **float**

До изобретения **HTML5** с его структурными элементами и активного использования строчно-блочной модели разметки, разработчики применяли свойство **float** к **div**'ам и таким образом создавали макеты с обтекаемыми элементами расположенными рядом или напротив друг друга.

float: left; — элемент прикрепится к родительскому элементу слева и другие элементы будут обтекать его справа.
float: right; — элемент прикрепится к родительскому элементу справа и другие элементы будут обтекать его слева.

Изменение значения display у обтекаемого элемента

Важно понимать, что обтекаемый элемент удаляется из обычного потока страницы и что значение **display**, заданное у него по умолчанию, меняется на **block**.

При этом «ведет» себя элемент как **inline-block**: ширина его содержимого становится по умолчанию его шириной.

Например, элемент, у которого **display** указан как **inline**, такой как строчный **span**, игнорирует любые свойства **height** или **width**. Однако, если для строчного элемента указать **float**, значение **display** изменится на **block** и тогда элемент уже может принимать свойства **height** или **width**.

Когда мы применяем **float** для элемента, то должны следить за тем, как это влияет на значение свойства **display**!

Очистка float

Очистка **float** происходит с помощью свойства **clear**, которое принимает несколько различных значений: **left, right, both**

- значение **left** очистит левые **float**
- значение **right** очистит правые **float**
- значение **both** очистит левый и правый **float**

```
div { clear: both; }
```

Еще одним способом очистки **float** является так называемое «переоборачивание» обтекаемого содержимого и назначение ему **overflow: hidden;**

HTML

```
<div class="wrapper">
        <div class="left-box"></div>
        <div class="right-box"></div>
</div>
```

CSS

```
.wrapper  { overflow: hidden; }
.left-box  { float: left; }
.right-box  { float: right; }
```

Позиционирование с помощью inline-block

Ещё один способ позиционирования контента — применение свойства **display** в сочетании со значением **inline-block**.

Метод с **inline-block**, в первую очередь, предназначен для размещения элементов в линию рядом друг с другом.

Также, свойство **display: inline-block;** позволяет элементам принимать все свойства строчных и блочной модели, включая **height**, **width**, **border**, **padding** и **margin**.

Позиционирование с помощью **display: inline-block** позволяет в полной мере воспользоваться блочной моделью, не беспокоясь об очистке **float**.

Уникальное позиционирование элементов

Рано или поздно возникнет надобность точно позиционировать элемент, чего не возможно будет добиться с помощью флотирования или строчно-блочной модели.

Свойство **position** определяет, как элемент позиционируется на странице и будет ли он отображаться в обычном потоке документа.

Данное свойство применяется в сочетании с показателями смещения блока:
top, right, bottom, left, которые точно определяют, где элемент будет расположен путём перемещения его в разных направлениях.

По умолчанию у каждого элемента значение **position** установлено как **static**
Это означает, что элемент существует в обычном потоке документа и не принимает какие-либо свойства для его смещения.
Значение **static** наиболее часто переписывается значениями **relative** или **absolute**

Относительное позиционирование

Значение **relative** для свойства **position** позволяет элементам отображаться в обычном потоке страницы, резервируя место для элемента как предполагалось и не позволяя другим элементам его обтекать.
Однако, оно также позволяет модифицировать положение элемента с помощью свойств смещения.

Для относительно позиционированных элементов важно знать, что свойства смещения блока определяют, куда элемент будет перемещён, учитывая его исходное положение.

Когда мы позиционируем элемент с помощью свойств смещения, элемент перекрывает элемент под ним, а не сдвигает его вниз, как это делают свойства **margin** или **padding**.

Также, указание **position: relative;** обязательно для родительского элемента, внутри которого размещается абсолютно позиционированный элемент.
Данное сочетание свойств обеспечит «прикрепление» абсолютно позиционированного элемента к родителю.

Абсолютное позиционирование

Значение **absolute** для свойства **position** отличается от значения **relative** тем, что элемент с абсолютным позиционированием не появляется в обычном потоке документа – его исходное пространство и положение не резервируется.

Для абсолютно позиционированных элементов свойства смещения определяют в каком направлении элемент будет перемещён относительно его ближайшего относительно позиционированного родителя.
Если относительно позиционированного родителя не существует, то абсолютно позиционированный элемент будет позиционироваться относительно элемента **body**!

Поскольку абсолютно позиционированный элемент не располагается в обычном потоке страницы, то он будет перекрывать любые окружающие его элементы.

Кроме того, исходное положение такого элемента не сохраняется
и другие элементы в праве занять это место.

Фиксированное позиционирование

Значение **fixed** свойства **position** задаёт элементу фиксированное позиционирование.

Фиксированное позиционирование похоже на абсолютное, но есть и отличия:

- фиксированный элемент так же выпадает из потока
- фиксированный элемент привязывается к определенной точке в окне браузера и остается на ней всегда, даже при прокрутке страницы
- фиксированный элемент можно позиционировать с помощью свойств **top, left, right,bottom,** но точка отсчета всегда привязана к окну браузера

Можете представить себе фиксированный элемент, как стикер, который наклеен на монитор. Такие элементы часто применяют для создания навигационных панелей, привязанных к верхней или нижней части окна браузера.

z-index

Любые позиционированные элементы на веб-странице могут накладываться друг на друга в определенном порядке, имитируя тем самым третье измерение, перпендикулярное экрану. Каждый элемент может находиться как ниже, так и выше других объектов веб-страницы.

Размещением элементов по z-оси управляет **z-index**
Это свойство работает только для элементов, у которых значение **position** задано как **absolute, fixed** или **relative**

В качестве значения используются целые числа. Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше.
При равном значении **z-index**, на переднем плане находится тот элемент, который в коде **HTML** описан ниже.

Хотя спецификация и разрешает использовать отрицательные значения **z-index** и ноль, отсчёт «слоёв» рекомендуется начинать с **1**
Кроме числовых значений применяется **auto** — порядок элементов в этом случае строится автоматически, исходя из их положения в коде **HTML** и принадлежности к родителю, поскольку дочерние элементы имеют тот же номер, что их родительский элемент.
Значение **inherit** указывает, что оно наследуется у родителя.