

CSS

# Styling before CSS

До появления CSS для стилизации использовались специальные теги и атрибуты. Например, тег `<font>`, `<b>`, `<i>`:

```
<font size="5" color="red" face="Arial">Привет мир</font>  
<i>Привет</i>
```

Привет мир *Привет*

То же самое касается рассмотренных нами атрибутов **bgslice** в таблицах или **width**, **height** для изображений. Сейчас задавать подобные настройки предпочтительно в CSS, а не атрибутах. Это позволяет отделить разметку от стилей.

В данном разделе мы узнаем, как пользоваться таблицами стилей и задавать в них все эти настройки.

# <style> tag

```
<html>
<head>
  <title>Hello CSS</title>
  <style>
    h1 {
      color: green;
    }
  </style>
</head>

<body>
  <h1>Hello CSS!</h1>
</body>
</html>
```

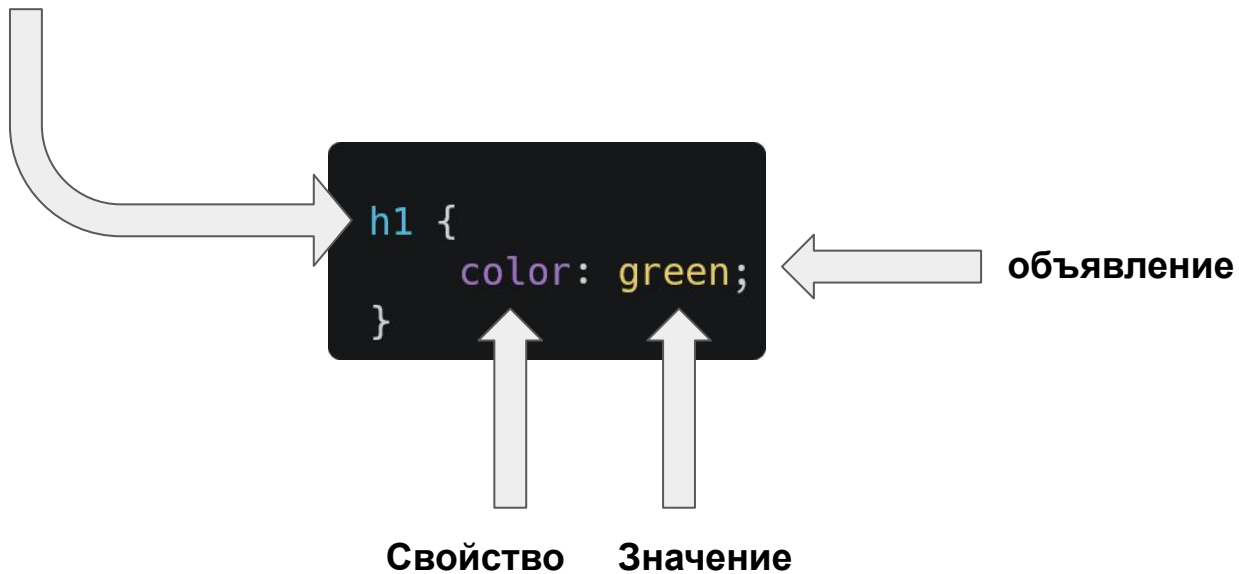


Hello CSS!

Внутри тега <style> располагается код CSS.  
Рассмотрим его подробнее

# CSS code

**Селектор** - указывает, к какому элементу применяется данное правило



# CSS code

Через запятую можно указать несколько селекторов



```
h1, h2, h3 {  
  color: green;  
  font-family: Georgia;  
}
```

Также можно установить несколько свойств, отделив их точкой с запятой

# Moving the CSS code to a separate file

Помимо размещения CSS кода в теге `<style>`, его можно вынести в отдельный файл стилей и подключить к нужной HTML странице.

Элемент **<link>** в HTML-документе сообщает браузеру, где находится файл CSS, используемый для форматирования страницы. Это пустой элемент (он не нуждается в наличии закрывающего тега), который располагается внутри элемента `<head>`. Элемент **<link>** должен использовать два следующих атрибута: **href** (Указывает путь к файлу CSS) и **rel** (определяет отношение между HTML-страницей и связанным файлом. При создании ссылки на файл CSS должно быть указано значение `"stylesheet"`)

```
<html>
<head>
  <title>Hello CSS</title>
  <link rel="stylesheet" href="style.css" />
</head>

<body>
  <h1>Hello CSS!</h1>
</body>
</html>
```

```
h1 {
  color: green;
}
```

Hello CSS!

(В коде HTML-страницы можно использовать более чем одну таблицу стилей. В этом случае для каждого файла CSS необходимо указать отдельный элемент `<link>`)

# CSS Selectors

Ниже перечислены основные виды селекторов CSS

## Пример

Универсальный селектор

```
* {  
    color: green;  
}
```

применяется ко всем  
элементам в документе

Селектор имени тега

```
h1 {  
    color: green;  
}
```

применяется ко всем элементам  
<h1>

# CSS Selectors

## Пример

Селектор класса

```
.className {  
    color: green;  
}
```

применяется ко всем элементам, чей атрибут class имеет значение "className"

Комбинированный селектор класса

```
h2.className {  
    color: green;  
}
```

применяется ко всем элементам с тегом <h2>, чей атрибут class имеет значение "className"

Селектор идентификатора

```
#identifier {  
    color: green;  
}
```

применяется к элементу, чей атрибут id имеет значение "identifier"



# CSS Selectors

## Пример

Дочерний селектор

```
li>a {  
    color: green;  
}
```

применяется к элементам <a>, являющимися прямыми потомками элемента <li>

Селектор потомка

```
li a {  
    color: green;  
}
```

применяется к элементам <a>, являющимися потомками элемента <li> (не обязательно прямыми)

# How the browser sees the selectors

Каким образом браузер проверяет, подходит текущему элементу css селектор или нет?

Если это селектор по *id* или по *class*, то браузер просто сравнивает вышеупомянутые атрибуты элемента с тем, что указано в селекторе.

Но что происходит с комбинированными селекторами? Рассмотрим селектор:



```
.content li a {  
  color: green;  
}
```

**Проверка комбинированных селекторов происходит справа налево.** В данном примере браузер сначала проверит, является ли текущий элемент ссылкой (`<a>`). Далее пройдет выше в дереве элементов и проверит, является ли этот элемент потомком `<li>`. И потом пройдет ещё выше чтобы проверить, является ли элемент потомком некоего элемента с классом `content`. Если на каком-нибудь этапе проверка не прошла, то браузер переходит к обработке следующих стилей и элементов.

# How the browser sees the selectors

Комбинированные селекторы отнимают больше ресурсов у компьютера, потому что для их проверки браузеру приходится делать много действий. Из всего этого следует, что **использование селекторов по id и class предпочтительнее с точки зрения производительности**

```
* {  
    color: green;  
}
```

Универсальный селектор является одним из самых ресурсозатратных, потому что он вынуждает браузер применять стиль буквально к каждому элементу.

# inline styles

Существует еще один способ задать стили CSS к элементу - это атрибут **style**. Он позволяет указывать css правила к конкретному элементу прямо в разметке.

```
<p style="color: blue; font-size: 30px;">Привет мир</p>
```

Привет мир

Задание стилей через **style** по возможности стоит избегать, так как это загромождает разметку и смешивает структуру страницы с ее стилями.

# Specificity of css rules

```
<h1 id="myheader">Hello CSS!</h1>
```

```
h1 {  
    color: green;  
}  
  
#myheader {  
    color: red;  
}
```

**Hello CSS!**

\*Применилось правило селектора по идентификатору, так как оно имеет более высокую специфичность

# Specificity of css rules

## Очки специфичности

0 - Универсальный селектор

Применяется то правило, которое имеет наивысшую специфичность

1 - Селектор по имени тега

(Если разные правила имеют одинаковый уровень специфичности, то применяется последнее из них.)

10 - Селектор по классу

100 - Селектор по идентификатору

1000 - inline стили (атрибут style)

# Specificity of css rules

Пример расчета специфичности

| Селектор     | Идент-ор | Класс | Тег | Сумма |
|--------------|----------|-------|-----|-------|
| p            | 0        | 0     | 1   | 1     |
| .class       | 0        | 10    | 0   | 10    |
| p.class      | 0        | 10    | 1   | 11    |
| #id          | 100      | 0     | 0   | 100   |
| #id p        | 100      | 0     | 1   | 101   |
| #id .class   | 100      | 10    | 0   | 110   |
| #id .class p | 100      | 10    | 1   | 111   |

# Specificity of css rules: important

С помощью конструкции **!important**, указанной после значения свойства, можно задать этому правилу наивысшую важность (в том числе большую, чем у inline стилей).

```
<h1 id="myheader" style="color: blue;">Hello CSS!</h1>
```

```
h1 {  
    color: green !important;  
}  
  
#myheader {  
    color: red;  
}
```

Hello CSS!

(Если несколько одних и тех же свойств в разных местах будут иметь **!important**, применится то свойство, селектор которого имеет большую специфичность. То есть **!important** свойства также как и обычные подчиняются специфичности при конкуренции друг с другом.)



Setting the color

# Setting text color

```
<h1>Заголовок</h1>  
<h2>Подзаголовок</h2>  
<p>Текст страницы</p>
```

```
h1 {  
    color: #123FFF;  
}  
h2 {  
    color: rgba(128,255,0,1);  
}  
p {  
    color: cyan;  
}
```

Задать цвет можно тремя способами:

- через шестнадцатеричный код
- через rgba
- через имя цвета

Заголовок

Подзаголовок

Текст страницы

# Style inheritance

Если указать свойства *font-family* или *color* для элемента `<body>`, то они будут применяться к большинству дочерних элементов, поскольку значение этих свойств наследуется дочерними элементами. Это избавляет от необходимости применять свойства к каждому элементу по отдельности, а также упрощает таблицу стилей. Такие свойства называются **наследуемыми**.

Другие же свойства, например, *border* не будут наследоваться дочерними элементами и называются **не наследуемыми**.

```
body {  
  color: red;  
  font-family: Georgia;  
  border: 1px solid black;  
}
```

Наследуется

Не наследуется

**Заголовок страницы**

Текст страницы

```
<body>  
<div>  
  <h2>Заголовок страницы</h2>  
  <p>Текст страницы</p>  
</div>  
</body>
```

Полный список наследуемых свойств можете посмотреть в [стандарте CSS](#) (см. столбец “Inherited?”).

Наследуются в основном свойства, определяющие параметры отображения текста.

# Style inheritance

Чтобы заставить не наследуемые свойства наследовать значения родительских элементов, можно использовать **inherit** в качестве их значения.

```
body {  
    border: 1px solid black;  
}  
  
div, h2, p {  
    border: inherit;  
}
```

Наследуем значение у родителя.



```
<body>  
<div>  
    <h2>Заголовок страницы</h2>  
    <p>Текст страницы</p>  
</div>  
</body>
```

|                           |
|---------------------------|
|                           |
| <b>Заголовок страницы</b> |
|                           |
| Текст страницы            |
|                           |

# Setting background color

```
<h1>Заголовок</h1>  
<h2>Подзаголовок</h2>  
<p>Текст страницы</p>
```

CSS обращается с каждым HTML-элементом так, будто он заключен в блок, а свойство **background-color** задает цвет фона этого блока.

```
h1 {  
    background-color: #123FFF;  
}  
h2 {  
    background-color: rgba(128,255,0,1);  
}  
p {  
    background-color: cyan;  
}
```

Заголовок

Подзаголовок

Текст страницы

# Text styling

# Setting the font

```
<h1>Заголовок</h1>
```

Браузер выбирает первый шрифт из списка, который установлен в системе пользователя.

```
h1 {  
  font-family: Arial, Verdana , sans-serif;  
}
```

**Заголовок**

# Adding custom font

```
<h1>Заголовок</h1>
```

Специальная директива CSS (рассмотрим их подробнее на следующем слайде)

```
@font-face {  
  font-family: 'Exo2';  
  src: url('Exo2.ttf');  
}  
  
h1 {  
  font-family: 'Exo2', sans-serif;  
}
```

Заголовок

url() - это CSS функция используется для включения файла. Она принимает на вход абсолютный URL, относительный URL или URI данных.



# @-rules

**Директивы (@-правила)** — это конструкции, которые позволяют создавать в CSS инструкции для изменения отображения либо поведения элементов страницы. Директива начинается со знака “@”, за которым следует одно из служебных слов. Также после директивы может идти блок, заключённый в фигурные скобки с дополнительными настройками. Примеры популярных директив:

```
@charset "UTF-8";
```

Определяет кодировку, используемую браузером.

```
@import 'another.css';
```

Запрашивает и включает в себя внешний CSS-файл, причём содержимое этого файла будет добавлено непосредственно в то место, где находится директива

# @-rules

```
@font-face {  
  font-family: 'MyWebFont';  
  src:        url('myfont.ttf') format('ttf'),  
            url('myfont.woff') format('woff');  
}
```

Загружает пользовательские шрифты для использования на странице. Поддерживает загрузку разных форматов шрифтов.

```
@media (min-device-width: 100px) and (max-device-width: 480px) {  
  .module { width: 100%;}  
  .footer {display: none;}  
}
```

Директива условных выражений, применяющая определённые стили в зависимости от характеристик экрана.

(Данные стили применяются только если ширина экрана пользователя находится в пределах от 100px и до 480px)

# Setting font size

```
<p id="p1">Параграф 1</p>
```

```
#p1 {  
  font-size: 50px;  
}
```

Параграф 1

## Единицы измерения

Абсолютные

*px(пиксели),  
mm(миллиметры),  
cm(сантиметры),  
pt(пункты),  
pc(пики)*

Относительные

*% (проценты),  
em(единицы em),  
rem(единицы rem)*

# Units

**px (пиксели)** - абсолютная базовая единица измерения. Один px равен одному пикселю на экране (разрешение экрана - это как раз количество таких пикселей, умещенных в длине и ширине экрана)

**mm, cm, pt, pc** - устаревшие абсолютные единицы измерения. Линейным образом переводятся браузером в пиксели, а значит являются бессмысленными.

**% (проценты)** - относительная единица. Свойство указанное в процентах рассчитывается браузером по отношению к свойству родителя. (100% = размер в соответствующем свойстве родителя)

**em** - относительная единица. Свойство указанное в **em** рассчитывается браузером по отношению к текущему размеру шрифта. (1em = текущий размер шрифта). Для font-size **единицы em** аналогичны процентам (1em = 100% и 1.4em = 140%).

**rem** - относительная единица. Свойство указанное в **rem** рассчитывается браузером по отношению к размеру шрифта элемента `<html>`. (1rem = размер шрифта элемента `<html>`)

# font-weight and font-style

Свойство **font-weight** задает жирность текста. [Возможные значения: “bold” (полужирный), “normal” (обычный), “bolder” (жирнее, чем родитель), “lighter” (тоньше, чем родитель), а также условные единицы от 100 до 900 с шагом в 100]

Свойство **font-style** задает начертание текста. [Возможные значения: “normal” (обычный), “italic” (курсив), “oblique” (наклонный)]

```
<p id="p1">Полужирный</p>
```

```
<p id="p2">Курсив</p>
```

```
#p1 { font-weight: bold; }
```

```
#p2 { font-style: italic; }
```

Полужирный

*Курсив*

# text-decoration

Свойство **text-decoration** добавляет оформление текста. [Возможные значения: “none” (обычный текст), “underline”(подчёркнутый), “line-through”(зачёркнутый), “overline” (подчёркнутый сверху)]

```
<p id="p1">Параграф 1</p>
<p id="p2">Параграф 2</p>
```

```
#p1 {
  text-decoration: underline;
}
#p2 {
  text-decoration: line-through;
}
```

Параграф 1

~~Параграф 2~~

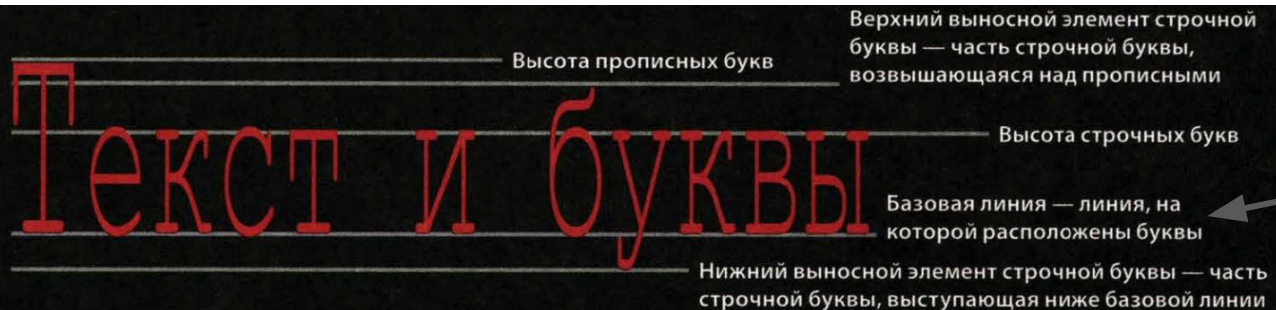
# Line spacing



```
<p id="p1">Lorem ipsum dolor sit amet...</p>
```

```
#p1 {  
  line-height: 40px;  
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.



(Отсчет **line-height** ведется от базовой линии шрифта)

# About displaying text

Можно заметить, что высота блока немного отличается от размера шрифта находящегося внутри текста.

```
<div>Lorem ipsum dolor sit amet...</div>
```

```
div {  
  font-size: 30px;  
}
```

Lorem ipsum dolor sit amet...

div 504 × 34.67

\*Высота блока равна 34.67px, а не 30px, как размер шрифта

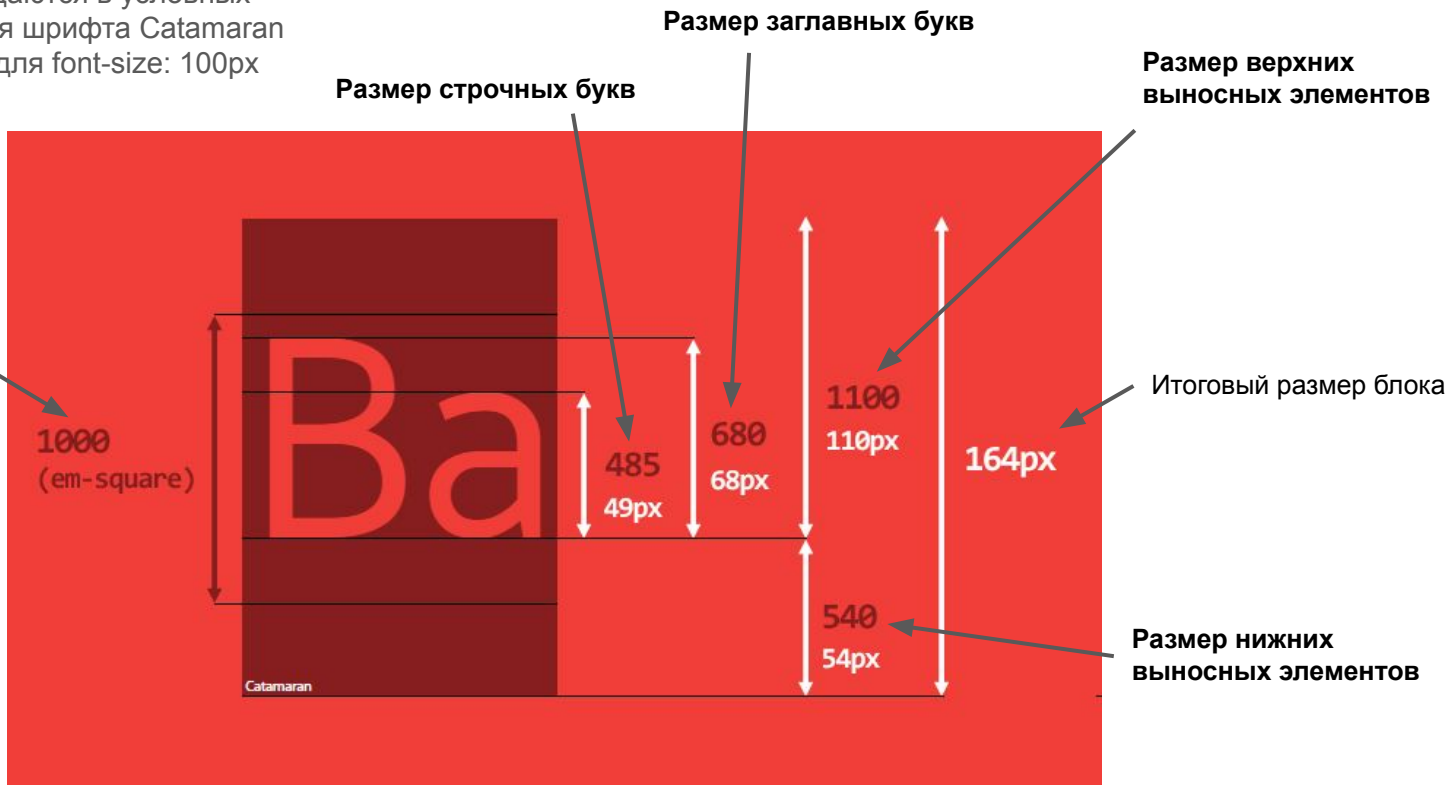
Чтобы разобраться, почему так происходит, нужно понять, какие настройки содержит в себе шрифт и каким образом задается его размер.



# Font parameters

Параметры шрифтов задаются в условных единицах. Например, для шрифта Catamaran с пересчетом в пиксели для font-size: 100px имеем:

**Кегельная площадка**  
(задаваемая в самом шрифте  
площадка, в рамках которой  
будет рисоваться каждый  
символ. Принимается за 1000  
условных единиц)



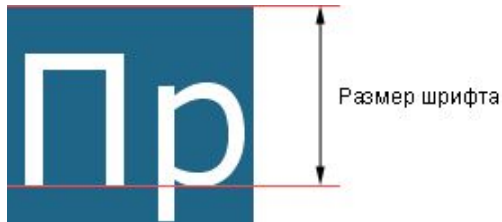
# Font size

Размер шрифта определяется как высота от *базовой линии* до верхней границы *кегельной площадки*:

При этом так называемые верхние выносные и нижние выносные элементы могут выходить за пределы кегельной площадки.

В связи с этим браузер автоматически выбирает высоту блока с запасом, так чтобы в неё умещались все элементы символов.

Мы можем запретить браузеру автоматически рассчитывать высоту строки, установив конкретный *line-height*. Но в таком случае часть элементов шрифта может выходить за пределы блока, что приведет к наложению линий текста одна на другую.



```
<div>Lorem ipsum dolor sit amet...</div>
```

```
div {  
  font-size: 30px;  
  line-height: 30px;  
}
```

Lorem ipsum dolor sit amet...

div 429.71 × 30

# Letter spacing and word spacing

```
<p id="p1">Lorem ipsum dolor sit amet...</p>
```

```
#p1 {  
  letter-spacing: 3px;  
  word-spacing: 20px;  
}
```

Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed  
do eiusmod tempor incididunt ut  
labore et dolore magna aliqua.  
Ut enim ad minim veniam, quis  
nostrud exercitation ullamco laboris  
nisi ut aliquip ex ea commodo  
consequat.

# Text alignment

```
<p id="p1">Параграф 1</p>
<p id="p2">Параграф 2</p>
<p id="p3">Параграф 3</p>
<p id="p4">Lorem ipsum dolor sit amet...</p>
```

```
#p1 {
  text-align: left;
}
#p2 {
  text-align: right;
}
#p3 {
  text-align: center;
}
#p4 {
  text-align: justify;
}
```

**text-align** выравнивает текст внутри блоков (в нашем случае блоков <p>). Положение же самих этих блоков и их размер остаются неизменными (по умолчанию блочные элементы стремятся растянуться по ширине на всю страницу)

Параграф 1

Параграф 2

Параграф 3

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

# Vertical alignment

Применяются для вертикального выравнивания картинок (а также текста в ячейках таблицы)

```
<p>  Параграф 1</p>
<p>  Параграф 2</p>
<p>  Параграф 3</p>
```

```
#img1 {
    vertical-align: baseline;
}
#img2 {
    vertical-align: text-top;
}
#img3 {
    vertical-align: middle;
}
```



Параграф 1



Параграф 2



Параграф 3

# text-shadow property

С помощью этого свойства можно добавить тень к тексту.

```
<p id="p1">Параграф 1</p>
```

Параграф 1

```
#p1 {  
  text-shadow: 2px 2px 1px gray;  
}
```

Сдвиг тени по горизонтали и вертикали, радиус размытия и цвет.

# Block styling

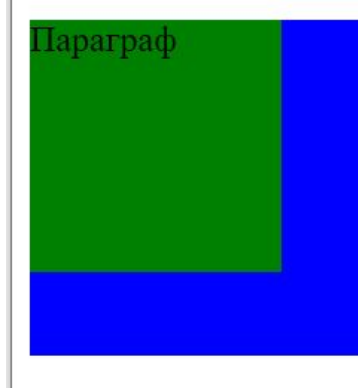
# Block sizes

По умолчанию высота блока задается так, чтобы он был способен вместить контент, а ширина старается принять максимальное доступное значение. Кастомные размеры блока мы можем задать свойствами **width** и **height**.

```
<div>
  <p>Параграф</p>
</div>
```

```
div {
  width: 150px;
  height: 150px;
  background-color: blue;
}
p {
  width: 75%;
  height: 75%;
  background-color: green;
}
```

При использовании процентов размер блока задается относительно размера окна браузера, а если блок заключен в другой блок, то относительно его размера.  
(ширина рассчитывается относительно ширины родителя, высота относительно высоты родителя)



(Если размер окна браузера меньше, чем установленные **width** или **height** блока, то браузер создаст *горизонтальную или вертикальную полосы прокрутки* соответственно.)



# Limiting block size

При изменении размеров окна браузера, ширина блоков может меняться. Жёсткое задание ширины через *width* может быть не всегда удобно, так как размер блока будет оставаться всегда постоянным. Браузер не будет сжимать блок при маленьких размерах окна браузера (появится полоса прокрутки) и не будет растягивать его при больших.

Если мы всё же хотим, чтобы браузер мог менять ширину блока, но в определённых пределах, то нам следует использовать свойства: **min-width**, **max-width**.

Свойство **min-width** определяет наименьшую ширину блока при уменьшении окна браузера, а свойство **max-width** - наибольшую ширину блока при увеличении.

```
<div>Блок</div>
```

```
div {  
  min-width: 400px;  
  max-width: 600px;  
  background-color: #78FFD9;  
}
```

Ширина блока лежит в пределах от 400px до 600px

(Есть также аналогичные свойства для высоты блока: **min-height**, **max-height**)

# Limiting block size



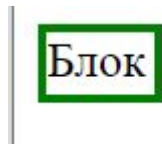
# Borders, Margins and Paddings



# Border

```
<div>Блок</div>
```

```
div {  
  width: 35px;  
  height: 20px;  
  border: 3px solid green;  
}
```

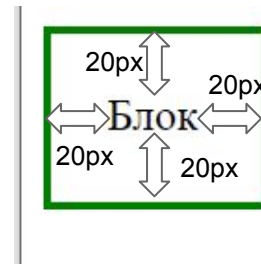


Можно также задать различные границы для каждой стороны блока, используя свойства: **border-top**, **border-right**, **border-bottom**, **border-left**;

# Paddings

```
<div>Блок</div>
```

```
div {  
  width: 35px;  
  height: 20px;  
  border: 3px solid green;  
  padding: 20px;  
}
```

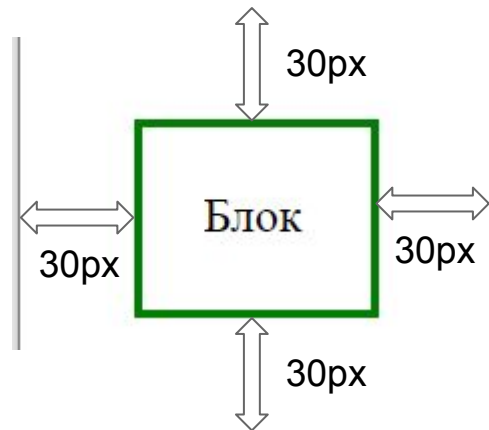


Можно также задать различные отступы для каждой стороны блока, используя свойства: **padding-top**, **padding-right**, **padding-bottom**, **padding-left**;

# Margins

```
<div>Блок</div>
```

```
div {  
  width: 35px;  
  height: 20px;  
  border: 3px solid green;  
  padding: 20px;  
  margin: 30px;  
}
```



Можно также задать различные поля для каждой стороны блока, используя свойства: **margin-top**, **margin-right**, **margin-bottom**, **margin-left**;

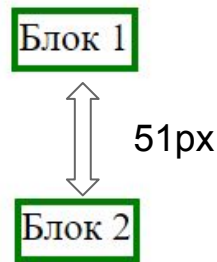
# Fields: Indentation slamming

```
<div id="div1">Блок 1</div>  
<div id="div2">Блок 2</div>
```

```
#div1 {  
  width: 45px;  
  height: 20px;  
  border: 3px solid green;  
  margin: 50px;  
}  
#div2 {  
  width: 45px;  
  height: 20px;  
  border: 3px solid green;  
  margin: 51px;  
}
```

Если один блок располагается поверх другого, то будут использоваться поля, имеющие наибольшее значение.

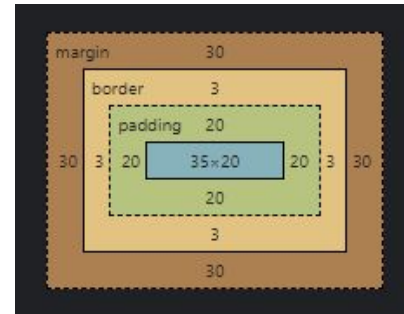
(Такое поведение называется *схлопыванием отступов*)



# Viewing block sizes in browser tools

```
<div>Блок</div>
```

```
div {  
  width: 35px;  
  height: 20px;  
  border: 3px solid green;  
  padding: 20px;  
  margin: 30px;  
}
```





# Margins and Paddings: Stenographic properties

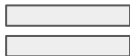
В CSS есть специальные свойства: **margin** и **padding**, которые позволяют быстро определить все поля или отступы. Они принимают значения полей/отступов в следующем порядке: *верхний, правый, нижний, левый* (по часовой стрелке начиная сверху).

```
padding-top: 1px;  
padding-right: 2px;  
padding-bottom: 3px;  
padding-left: 4px;
```



```
padding: 1px 2px 3px 4px;
```

```
margin-top: 1px;  
margin-right: 2px;  
margin-bottom: 3px;  
margin-left: 4px;
```



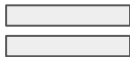
```
margin: 1px 2px 3px 4px;
```

Такие свойства называются **стенографическими**. Они не несут нового функционала, а лишь являются более кратким синтаксисом классических свойств.

# Margins and Padding: Stenographic properties

Если для **margin** или **padding** задать только 2 отступа, то последующие два будут установлены аналогичные.

```
padding-top: 1px;  
padding-right: 2px;  
padding-bottom: 1px;  
padding-left: 2px;
```



```
padding: 1px 2px;
```

```
margin-top: 1px;  
margin-right: 2px;  
margin-bottom: 1px;  
margin-left: 2px;
```



```
margin: 1px 2px;
```

# Overflowing content

Свойство **overflow** сообщает браузеру, что делать, если объем контента блока превышает его размер.

Возможные значения:

- *visible* (Стоит по умолчанию. Не влезающий контент отображается вне блока)
- *hidden* (Всё, что не влезает в размеры блока станет невидимым)
- *scroll* (Добавляются полосы прокрутки для просмотра не влезающего содержимого. Причем полосы прокрутки появляются всегда, даже если контент не выходит за пределы блока.)
- *auto* (То же самое, что и *scroll*, но полосы прокрутки появляются только при необходимости, т.е. когда контент выходит за пределы блока)

Существуют также свойства **overflow-x** and **overflow-y**, которые позволяют управлять отдельно поведением при переполнении по вертикали и по горизонтали. Принимают они те же самые значения, что и обычный **overflow**.

- **overflow-x** указывает что делать с левым и правым краями контента
- **overflow-y** указывает что делать с верхним и нижним краями контента

*Особенность:* Если использовать значение *visible* для одного из этих свойств (*overflow-x* или *overflow-y*) и что-то отличное от *visible* для другого, то *visible* для первого будет интерпретироваться браузером как *auto*.

```
<p id="p1">Lorem ipsum dolor sit amet...</p>
```

```
#p1 {  
  width: 300px;  
  height: 50px;  
  background-color: #78FFD9;  
  overflow: ;  
}
```

visible

hidden

scroll

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

# Content centering

Чтобы поместить блок в центре страницы (или в центре элемента, в котором он находится) нужно присвоить значение **auto** свойствам `margin-left` и `margin-right`.

```
<div id="div1">
  <div id="div2">Блок</div>
</div>
```

```
#div1 {
  border: 3px solid red;
}
#div2 {
  width: 35px;
  height: 20px;
  border: 3px solid green;
  margin: 0 auto 0 auto;
}
```



(Помимо полей **auto** нужно обязательно указать ширину блока (`width`), который мы хотим центрировать. В противном случае он займет всю ширину страницы и не центрируется.)

\*можно более кратко: `margin: 0 auto;`

# Block shadows and border rounding

```
<div>Блок</div>
```

```
div {  
  width: 100px;  
  height: 50px;  
  border: 3px solid green;  
  
  border-radius: 5px;  
  box-shadow: 5px 5px 3px gray;  
}
```

Блок

Радиус скругления углов рамки

Сдвиг тени по горизонтали и вертикали, радиус размытия и цвет.

# Setting background image

```
<div>Блок</div>
```

```
div {  
  width: 50px;  
  height: 50px;  
  color: white;  
  background-image: url("background.jpg");  
}
```



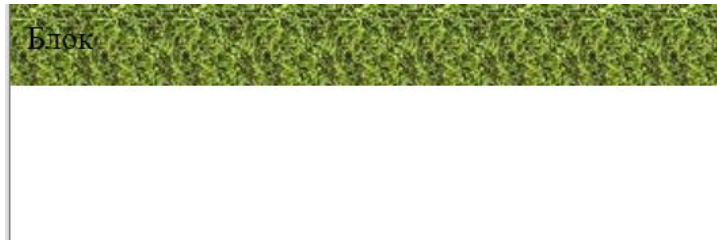
# Setting background image: Repeating

```
<div>Блок</div>
```

```
body {  
  background-image: url("background.jpg");  
  background-repeat: repeat; (по  
}  
                                умолчанию)
```

```
body {  
  background-image: url("background.jpg");  
  background-repeat: repeat-x;  
}
```

```
body {  
  background-image: url("background.jpg");  
  background-repeat: no-repeat;  
}
```





# Setting background image: Position

Если фоновое изображение не повторяется, то можно использовать свойство **background-position**, чтобы указать, в каком месте контейнера оно должно быть помещено.

```
<div>Блок</div>
```

```
body {  
  background-image: url("background.jpg");  
  background-repeat: no-repeat;  
  background-position: center top;  
}
```

```
body {  
  height: 50px;  
  background-image: url("background.jpg");  
  background-repeat: no-repeat;  
  background-position: 25% 50%;  
}
```

Блок



Блок



# Setting background image: Size

```
<div>Блок</div>
```

**background-size**

Блок



```
div {  
  width: 50px;  
  height: 50px;  
  background-image: url("cat.jpg");  
  background-repeat: no-repeat;  
}
```

**background-size:** 200px auto;

Блок



**background-size:** cover;

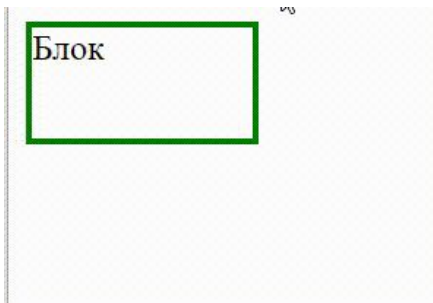
Блок



# Mouse pointer styles

```
<div>Блок</div>
```

```
div {  
  width: 100px;  
  height: 50px;  
  border: 3px solid green;  
  
  cursor: url("rosatom.gif"), default;  
}
```



























Предпочтительный формат: gif, размер: 32x32

# Mouse pointer styles

Можно также использовать встроенные указатели мыши:

**cursor:** <название стандартного курсора>;

|  |   |  |  |
|--|---|--|--|
|  auto       |  move          |  no-drop     |  col-resize |
|  all-scroll |  pointer       |  not-allowed |  row-resize |
|  crosshair  |  progress      |  e-resize    |  ne-resize  |
|  default    |  text          |  n-resize    |  nw-resize  |
|  help       |  vertical-text |  s-resize    |  se-resize  |
|  inherit    |  wait          |  w-resize    |  sw-resize  |

# Pseudo-classes, pseudo-elements

# Pseudo-elements: First letter or string

**Псевдоэлемент** - это ключевое слово, добавляемое к селектору, которое позволяет стилизовать определенную часть выбранного элемента.

Псевдоэлементы **::first-letter** и **::first-line** позволяют задать стили для первой буквы и первой строки текста соответственно:

```
<p id="p1">Lorem ipsum dolor sit amet...</p>
```

```
#p1::first-letter {  
  color: red;  
}  
#p1::first-line {  
  text-decoration: underline;  
}
```

Lorem ipsum dolor sit amet, consectetur   
adipiscing elit, sed do eiusmod tempor  
incididunt ut labore et dolore magna aliqua. Ut  
enim ad minim veniam, quis nostrud  
exercitation ullamco laboris nisi ut aliquip ex  
ea commodo consequat.

# Pseudo-elements `::before` и `::after`

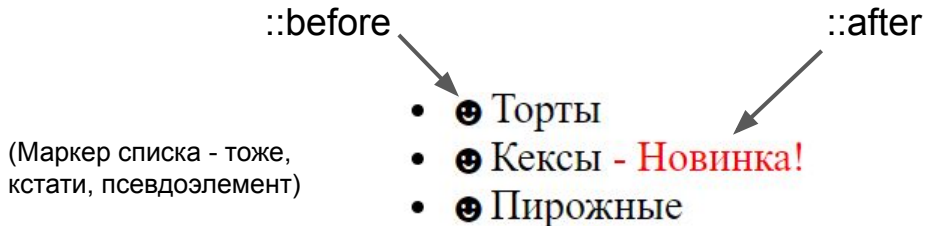
Псевдоэлементы `::before` и `::after` позволяют добавлять содержимое (стили) до и после элемента, к которому были применены.

Содержимое этих псевдоэлементов задаётся с помощью свойства **content**.

Вместе с этим свойством к ним можно применять и обычные стили CSS такие как цвет, размер шрифта, фоновый цвет и т.п.

```
<ul>
  <li>Торты</li>
  <li class="new">Кексы</li>
  <li>Пирожные</li>
</ul>
```

```
li::before {
  content: "☺";
}
.new::after {
  content: " - Новинка!";
  color: red;
}
```



Использование псевдоэлементов `::before` (`::after`) аналогично тому, что перед (после) содержимого указанного элемента будет добавлен текстовый контейнер `<span>`, в который будет помещено значение свойства **content** псевдоэлемента и которому будут применены остальные стили заданные в псевдоэlemente.

Однако нужно понимать, что элементы `::before` и `::after` не будут сгенерированы, т.е. не будут видны в коде страницы, поэтому они и называются псевдоэлементами. (Весь список псевдоэлементов можно [посмотреть на MDN](#))

# Pseudo-classes: Link Styles

**Псевдокласс** - это ключевое слово, добавленное к селектору, которое позволяет указать стили для особого состояния выбранного элемента.

С помощью псевдоклассов **:link** и **:visited** можно указать стили для непосещенных и посещенных ссылок соответственно.

```
<a href="#">Ссылка</a>
```

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

\*До посещения (Состояние: не посещена)

Ссылка

\*После посещения (Состояние: посещена)

Ссылка

(Заметьте, что псевдоклассы используют одно двоеточие, а не два, как псевдоэлементы. Хотя раньше псевдоэлементы тоже использовали одно)



# Pseudo-classes: Focus, click and hover over an element

С помощью псевдоклассов **:hover**, **:focus** и **:active** можно изменять внешний вид элементов в ответ на действия пользователя.

```
<p>Параграф</p>
<input type="text" value="текст"/>
<button>Кнопка</button>
```

```
p:hover {
  color: red;
}

input:focus {
  color: green;
}

button:active {
  background-color: yellow;
}
```

**:hover** Применяется, когда пользователь наводит на элемент указатель мыши.

**:focus** Применяется, когда элемент имеет фокус.

**:active** Применяется, когда элемент активируется пользователем, например путем нажатия кнопки или щелчка по ссылке. Стиль применяется в промежуток между нажатием и отпусанием пользователем кнопки мыши.

Параграф

текст

Кнопка

# Pseudo-classes: First and last child elements

Псевдоклассы **:first-child** и **:last-child** позволяют задать стиль для первого или последнего соответственно дочернего элемента из группы братских элементов (то есть таких, у которых один родитель).

```
<ul>
  <li>Торты</li>
  <li>Кексы</li>
  <li>Пирожные</li>
  <li>Пироги</li>
</ul>
```

```
li:first-child {
  background-color: green;
}
li:last-child {
  background-color: red;
}
```

- Торты
- Кексы
- Пирожные
- Пироги

(Весь список псевдоклассов можно [посмотреть на MDN](#))

# Layout

# Key Concepts

CSS обращается с каждым HTML-элементом так, будто он заключен в блок.

Блочные элементы располагаются на новой строке и ведут себя как основные структурные единицы макета, а встроенные элементы окружаются текстом.

**БЛОЧНЫЕ  
ЭЛЕМЕНТЫ**  
РАСПОЛАГАЮТСЯ  
НА НОВОЙ СТРОКЕ

Примеры:

`<h1>` `<p>` `<ul>` `<li>`

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit.

- Lorem ipsum dolor sit
- Consectetur adipisicing
- Elit, sed do eiusmod

**ВСТРОЕННЫЕ  
ЭЛЕМЕНТЫ**  
ОКРУЖЕНЫ  
ТЕКСТОМ

Примеры:

`<img>` `<b>` `<i>`

*Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.*



Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



# Element positioning



```
graph TD; A[Element positioning] --> B[Нормальный поток]; A --> C[Относительное позиционирование]; A --> D[Абсолютное позиционирование];
```

Нормальный поток  
(используется по-умолчанию)

**position:** static;

+Плавающие элементы  
(Свойство **float**)

Относительное  
позиционирование

**position:** relative;

Абсолютное  
позиционирование

**position:** absolute;

+Фиксированное  
позиционирование

**position:** fixed;

# Normal flow (default)

```
<body>
  <h1>Lorem ipsum</h1>
  <p id="p1">Lorem ipsum dolor sit amet...</p>
  <p id="p2">Ut enim ad minim veniam...</p>
  <p id="p3">Duis aute irure dolor in...</p>
</body>
```

```
body {
  width : 460px;
}
h1 {
  background-color : #dedede;
  padding: 10px;
}
p {
  width : 350px;
}
```

Каждый последующий блочный элемент на странице располагается ниже, чем предыдущий.

## Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

(Все последующие примеры, демонстрирующие схемы позиционирования, будут использовать эту структуру.)



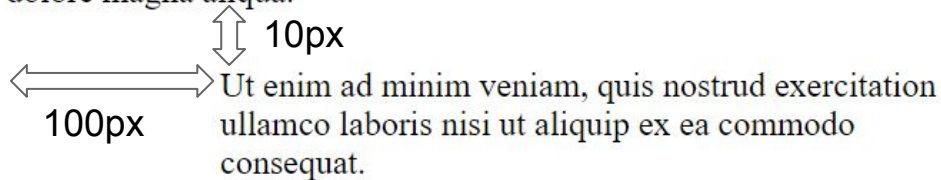
# Relative positioning

Положение элемента задается (свойствами **top**, **left**, **right**, **bottom**) относительно места, в котором он бы находился при использовании схемы нормального потока.

```
#p2 {  
  position: relative;  
  top: 10px;  
  left: 100px;  
}
```

## Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.



100px 10px  
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

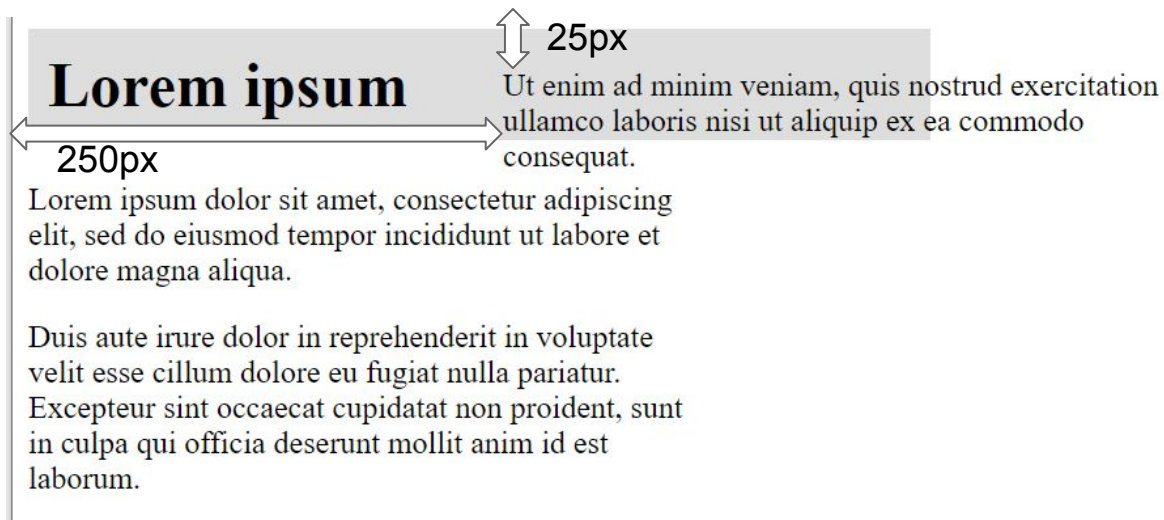


# Absolute positioning

Блок исключается из нормального потока и не влияет на положение других элементов страницы. (Для них этого блока словно не существует.)

Свойства смещения блока (**top**, **bottom**, **left** или **right**) указывают, где должен быть расположен элемент относительно контейнера (*первого позиционированного [т.е. не static] элемента-предка*).

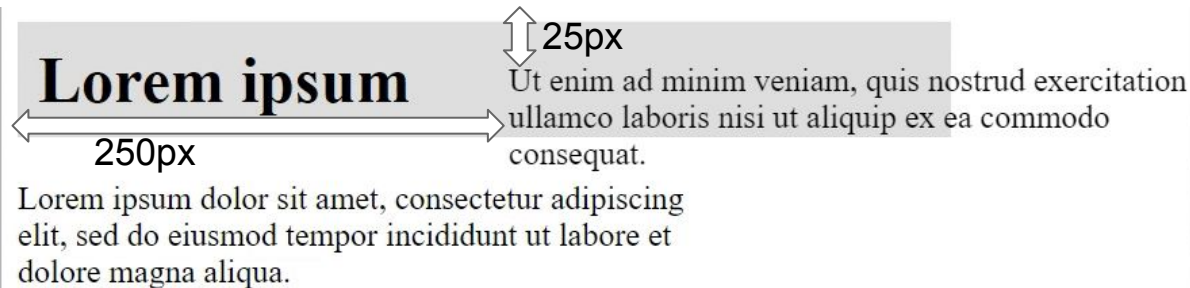
```
#p2 {  
  position: absolute;  
  top: 25px;  
  left: 250px;  
}
```



# Fixed positioning

Форма абсолютного позиционирования, при которой элемент позиционируется относительно окна браузера. Таким образом, при прокрутке страницы элемент остается на том же месте.

```
#p2 {  
  position: fixed;  
  top: 25px;  
  left: 250px;  
}
```



# Overlapping elements

Элементы (relative, absolute, fixed), которые указаны в HTML-коде позднее, располагаются поверх тех, которые указаны ранее. Порядок можно изменить свойством **z-index**. Чем выше этот параметр, тем ближе данный элемент к переднему плану.

Свойство не работает с элементами нормального потока! Только с relative, absolute, fixed.

(Рассмотрим пример про абсолютного позиционирование)

```
h1 {  
  position: absolute;  
  left: 200px;  
  z-index: 10;  
}  
  
h1 {  
  position: absolute;  
  left: 200px;  
  z-index: 10;  
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

**Lorem ipsum**

um, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

**Lorem ipsum**

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

# Floating elements (Float property)

Свойство **float** позволяет поместить элемент, позиционированный с помощью схемы нормального потока, с левого или с правого края элемента-контейнера. Все, что находится внутри контейнера, будет обтекать плавающий элемент.

```
<h1>Lorem ipsum</h1>

<blockquote>
  "Я мыслю, значит я существую". – Декарт
</blockquote>

<p id="p1">Lorem ipsum dolor sit amet...</p>
...
```

```
blockquote {
  float: right;
  width: 70px;
}
```

## Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

"Я  
мыслю,  
значит я  
существую".  
- Декарт

Вместе со свойством **float** нужно обязательно указать ширину блока (width)

# Using float to place elements next to each other

Во многих макетах блоки помещаются друг рядом с другом. Для этого обычно используется свойство float.

```
p {  
  float: left;  
  width: 150px;  
  margin-left: 3px;  
}
```

## Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

# Cleaning floating elements (Property clear)

Свойство **clear** позволяет указать, что ни один элемент, находящийся в том же элементе-контейнере, не должен касаться левой или правой стороны блока.

```
p {  
  float: left;  
  width: 150px;  
  margin-left: 3px;  
}  
  
#p3 {  
  clear: left;  
}
```

## Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

# Problem with the size of the parent elements of floating blocks

Если в элементе-контейнере содержатся плавающие элементы, то браузер может исключать их из размера контейнера.

```
body {  
  border: 1px solid black;  
}
```

## Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Граница вокруг контейнера <body> отобразилась неверно, так как при расчете его размера были проигнорированы дочерние плавающие элементы



# Problem with the size of the parent elements of floating blocks

Для решения этой проблемы нужно добавить в конец контейнера (после последнего плавающего элемента) пустой элемент со свойством *clear: both*;

(Теперь размеры контейнера `<body>` рассчитываются правильно)

```
<body>
  <h1>Lorem ipsum</h1>
  <p id="p1">Lorem ipsum dolor sit amet...</p>
  <p id="p2">Ut enim ad minim veniam...</p>
  <p id="p3">Duis aute irure dolor in...</p>
  <div class="clear"></div>
</body>
```

```
body {
  border: 1px solid black;
}

.clear {
  clear: both;
}
```

## Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.



# Display property

Свойство **display** определяет, как элемент должен быть показан в документе.

Основные возможные значения:

- *block* (Заставляет встроенный элемент вести себя как блочный)
- *inline* (Заставляет блочный элемент вести себя как строчный)
- *inline-block* (Генерирует блочный элемент, который обтекается другими элементами веб-страницы подобно строчному элементу)
- *none* (Скрывает элемент на странице, исключая из потока документа)
- *flex* (будет рассмотрено на следующей лекции)
- *grid* (будет рассмотрено на следующей лекции)

Полный список значений свойства **display** вы можете найти [на MDN](#).

# Display: block

Преобразует строчные элементы в блочные.

```
<p>  
  обратитесь к <a href="doc.html">документации</a>  
</p>
```

```
a {  
  display: block;  
}
```

Обратитесь к  
документации

```
a {  
  display: block;  
}
```

Обратитесь к документации

# Display: inline

Преобразует блочные элементы в строчные.

```
<ul>
  <li>Услуги</li>
  <li>Контакты</li>
  <li>О нас</li>
</ul>
```

```
li {
  display: inline;
}
```

Услуги Контакты О нас

```
li {
  display: inline;
}
```

- Услуги
- Контакты
- О нас

# Display: inline-block

**display:inline-block** в отличие от *display:inline* позволяет задать ширину и высоту элемента (width и height), а также поля и отступы (padding, margin)

В то же время **display:inline-block** в отличие от *display:block* не добавляет разрыв строки после элемента, поэтому элемент может располагаться рядом с другими элементами.

```
<p>  
  Обратитесь к <a href="doc.html">документации</a>  
</p>
```

```
a {  
  background-color: red;  
  width: 150px;  
  height: 40px;  
  margin-left: 10px;  
  display: inline-block;  
}
```

Обратитесь к

[документации](#)

# Display: Hiding elements

```
<ul>
  <li>Услуги</li>
  <li class="soon">Контракты</li>
  <li>Сотрудничество</li>
  <li>О нас</li>
</ul>
```

```
.soon {
  display: none;
}
```

- Услуги
- Сотрудничество
- О нас

Свойство **display:none** полностью исключает элемент из нормального потока (static).

Оно действует так, будто элемент вообще отсутствует (хотя посетитель может увидеть содержимое блока при просмотре исходного кода).

# Hiding elements: (visibility: hidden)

```
<ul>
  <li>Услуги</li>
  <li class="soon">Контракты</li>
  <li>Сотрудничество</li>
  <li>О нас</li>
</ul>
```

```
.soon {
  visibility: hidden;
}
```

- Услуги
- Сотрудничество
- О нас

Свойство **visibility: hidden** действует так, что на месте элемента появится пустое пространство.

# Hiding elements: (Changing transparency)

```
<ul>
  <li>Услуги</li>
  <li class="soon">Контракты</li>
  <li>Сотрудничество</li>
  <li>О нас</li>
</ul>
```

```
.soon {
  opacity: 0.5;
}
```

- Услуги
- Контракты
- Сотрудничество
- О нас

(Этот метод позволяет *взаимодействовать с элементом*. Например, в данном случае можно скопировать текст скрытого элемента через CTRL+C, если выделить курсором эту пустоту.

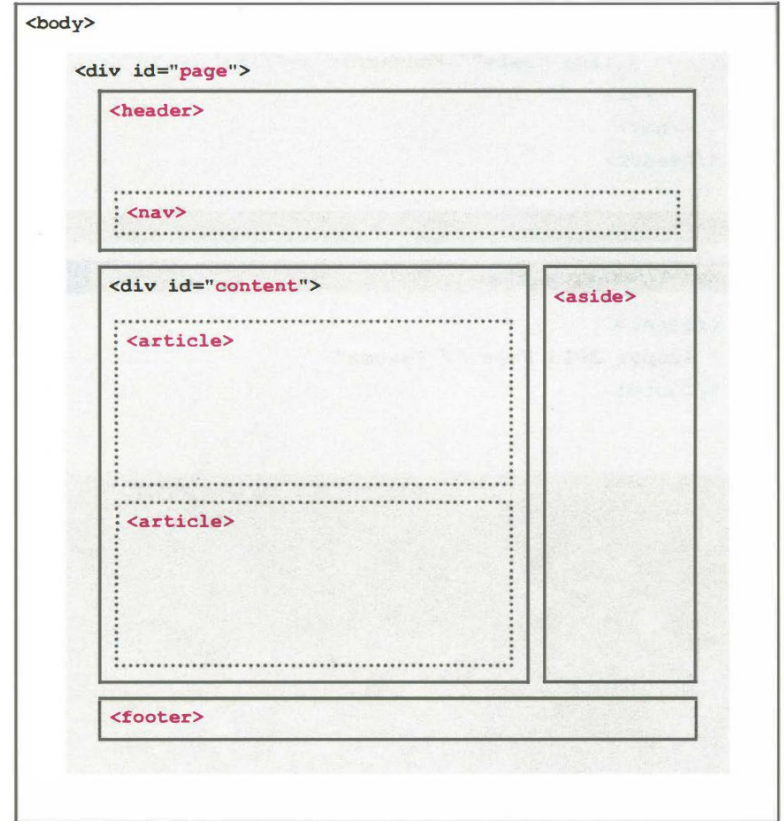
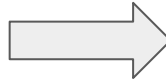
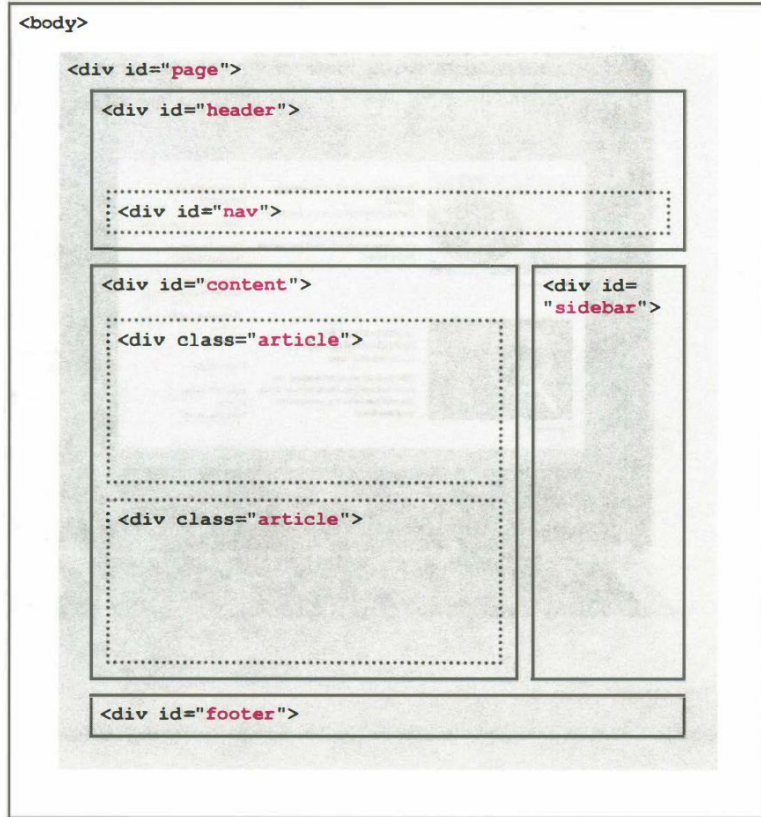
В случае с *visibility: hidden* элемент просто заменяется на пустоту эквивалентного размера и взаимодействовать с ней нельзя.

```
.soon {
  opacity: 0;
}
```

- Услуги
- Сотрудничество
- О нас

# Layouts in HTML5

В html5 появились специальные семантические контейнеры:





# Спасибо за внимание

Более подробно о [CSS](#) можно почитать на [MDN](#)

# Темы для докладов

1) Critical Rendering Path.

[https://developer.mozilla.org/en-US/docs/Web/Performance/Critical\\_rendering\\_path](https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path)

<https://web.dev/learn/performance/understanding-the-critical-path>

2) SVG animation.

<https://habr.com/ru/articles/450924/>