

# HTML Basics, DOM API

# **Internet and web**

Интернет ≠ Веб

- **Интернет** (появился в 1969) - связь нескольких компьютеров в единую сеть.
- **Веб** (World Wide Web) (появился в 1989) - распределённая система, которая дает доступ к документам связанным между собой.

# **Internet and web**

В **интернете** любая информация может передаваться любыми способами. То есть с помощью любых протоколов (Например HTTP, FTP, SMTP, DTN).

В **Вебе** ключевую роль играют документы, передающиеся по протоколу HTTP.

HTML(HyperText Markup Language) - язык на котором составлены документы в Вебе.

Веб-браузер - программа, которая позволяет эти документы загружать и просматривать.

Веб-сервер - программа, открывающая доступ к вашим HTML документам другим пользователям.

# Web

Чужой компьютер

Веб-сервер

Файловое пространство



<http://123.10.22.198/doc.html>

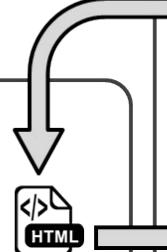
Ваш компьютер

Веб-браузер

Файловое пространство

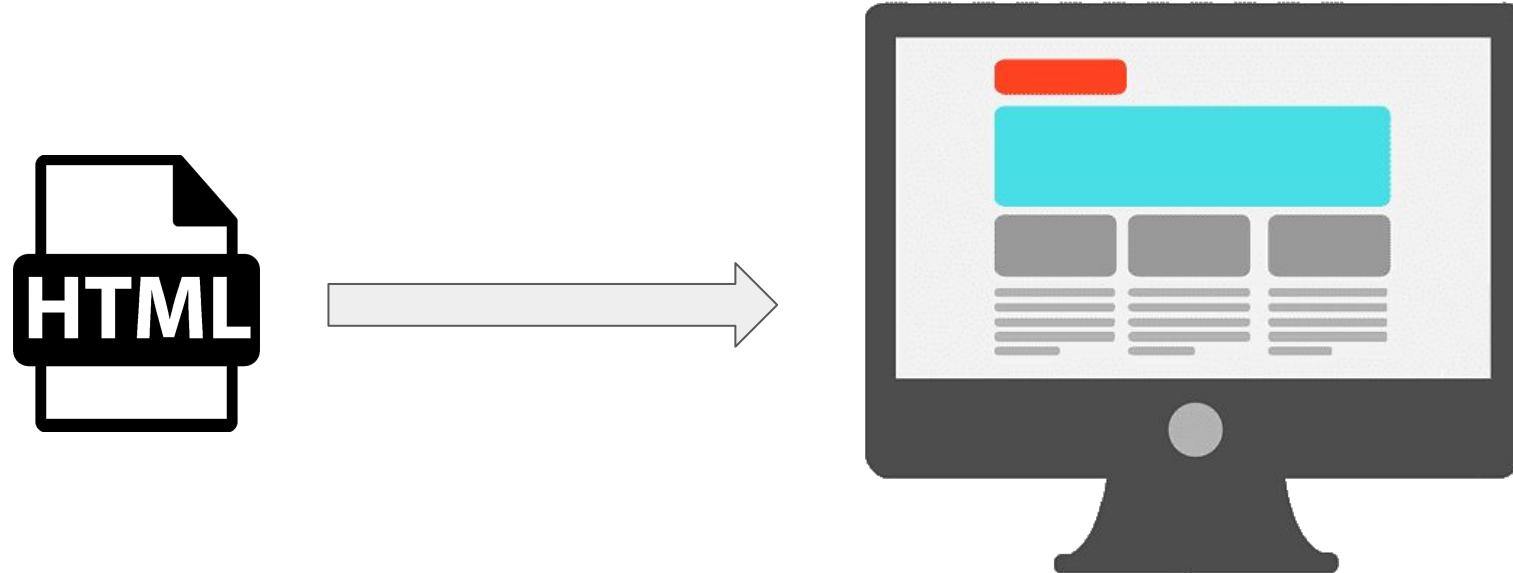


doc.html



# Web

После того, как браузер загрузил HTML документ, он его отображает пользователю.



# **HTML as a language for structuring documents**

The screenshot shows a web-based word processor interface with a toolbar at the top. The document contains three levels of headings: a main title, a subtitle, and a sub-subtitle. The right side of the screen displays the generated HTML code for these elements.

Document example

File Edit View Insert Format Tools Extensions Help Last edit was 4 minutes ago

Normal text Arial 11

1 2 3 4 5 6 7

1. Это основной заголовок

2. Этот текст может быть введением к тексту, помещённому далее на странице. Если текст достаточно длинный, он может быть разделен на несколько разделов с подзаголовками.

3. Это подзаголовок

4. Часто при написании длинных статей используют подзаголовки, помогающие читателям следить за структурой всего написанного. Иногда используются и подзаголовки более низкого уровня.

5. Еще один подзаголовок

6. Вот еще один пример подзаголовка.

7. Document.html

1	<html>
2	<body>
3	<h1>Это основной заголовок</h1>
4	<p>Этот текст может быть введением к тексту, помещённому далее на странице. Если текст достаточно длинный, он может быть разделен на несколько разделов с подзаголовками.</p>
5	<h2>Это подзаголовок</h2>
6	<p>Часто при написании длинных статей используют подзаголовки, помогающие читателям следить за структурой всего написанного. Иногда используются и подзаголовки более низкого уровня.</p>
7	<h3>Еще один подзаголовок</h3>
8	<p>Вот еще один пример подзаголовка.</p>

Это основной заголовок

Этот текст может быть введением к тексту, помещенному далее на странице. Если текст достаточно длинный, он может быть разделен на несколько разделов с подзаголовками.

**Это подзаголовок**

Часто при написании длинных статей используют подзаголовки, помогающие читателям следить за структурой всего написанного. Иногда используются и подзаголовки более низкого уровня.

**Еще один подзаголовок**

Вот еще один пример подзаголовка.

```
Document.html
1 <html>
2   <body>
3     <h1>Это основной заголовок</h1>
4     <p>Этот текст может быть введением к тексту, помещенному далее на странице. Если текст достаточно длинный, он может быть разделен на несколько разделов с подзаголовками.<p>
5       <h2>Это подзаголовок</h2>
6       <p>Часто при написании длинных статей используют подзаголовки, помогающие читателям следить за структурой всего написанного. Иногда используются и подзаголовки более низкого уровня.</p>
7       <h2>Еще один подзаголовок</h2>
8       <p>Вот еще один пример подзаголовка.</p>
9
10  </body>
</html>
```

# Tags and comments

Открывающий тег, закрывающий тег и содержимое между ними называется HTML элементом.

Атрибуты предоставляют дополнительную информацию о содержимом HTML элемента.

```
<открывающийТег имяАтрибута=значение> содержимое </закрывающийТег>
```

Например:

```
<p lang="ru">Текст абзаца</p>
```

Есть html элементы, которые не имеют закрывающего тега. Всё что находится между `<!--` и `-->` игнорируется браузером.

```
<открывающийТег имяАтрибута=значение />
```

Например:

```

```

Например:

```
<h1>Это html</h1>
<!-- <h1>Это комментарий</h1> -->
```

# HTML page structure

Теги подобны контейнерам.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Название страницы</title>
  </head>
  <body>
    <h1>Заголовок</h1>
    <p>Параграф</p>
  </body>
</html>
```

**<!DOCTYPE html>** - служебный тег, обозначающий, что документ соответствует стандарту HTML5.  
(Далее будем опускать для краткости)

**<html></html>** - всё помещенное внутри - html код

**<head></head>** - служебная информация о странице

**<title></title>** - название отображаемое в панели вкладок

**<body></body>** - всё помещенное внутри - должно отобразиться в основном окне браузера

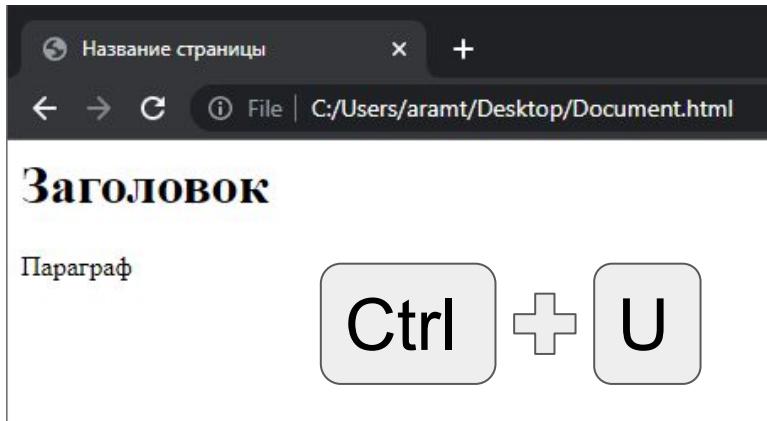
**<h1></h1>** - основной заголовок текста

**<p></p>** - абзац

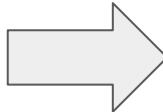
# Viewing HTML source code in the browser

Браузер позволяет посмотреть исходный код любой просматриваемой страницы.

Для этого нужно нажать Ctrl+U.



Ctrl + U



A screenshot of a terminal window titled "Document.html". The command "view-source:file:///C:/Users/aramt/Desktop/Document.html" is entered. The output shows the HTML code:

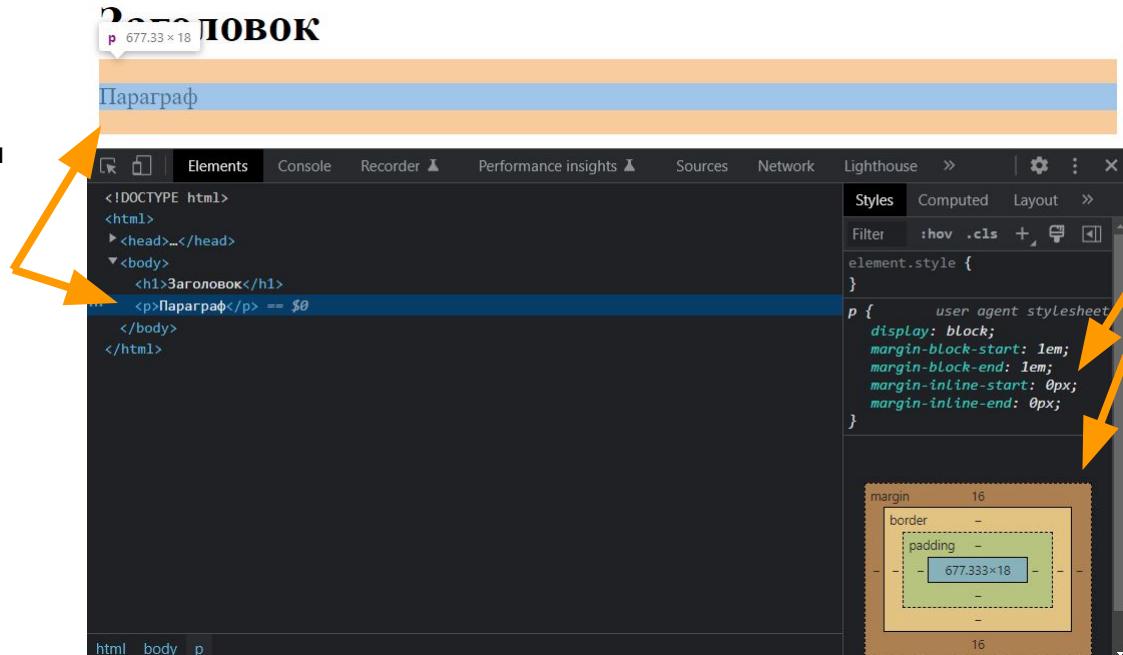
```
Line wrap □
1 <html>
2   <head>
3     <title>Название страницы</title>
4   </head>
5   <body>
6     <h1>Заголовок</h1>
7     <p>Параграф</p>
8   </body>
9 
10 </html>
```

# Developer Tools

Браузер предусматривает специальные инструменты для разработчиков, которые открываются нажатием F12. Нас интересует самая первая вкладка “Elements”.

В левой панели мы видим структуру нашего документа. Там мы можем навести курсор на любой элемент и браузер его подсветит.

Эта панель позволяет динамически менять разметку прямо в браузере (изменения сделанные здесь откатятся при обновлении страницы)



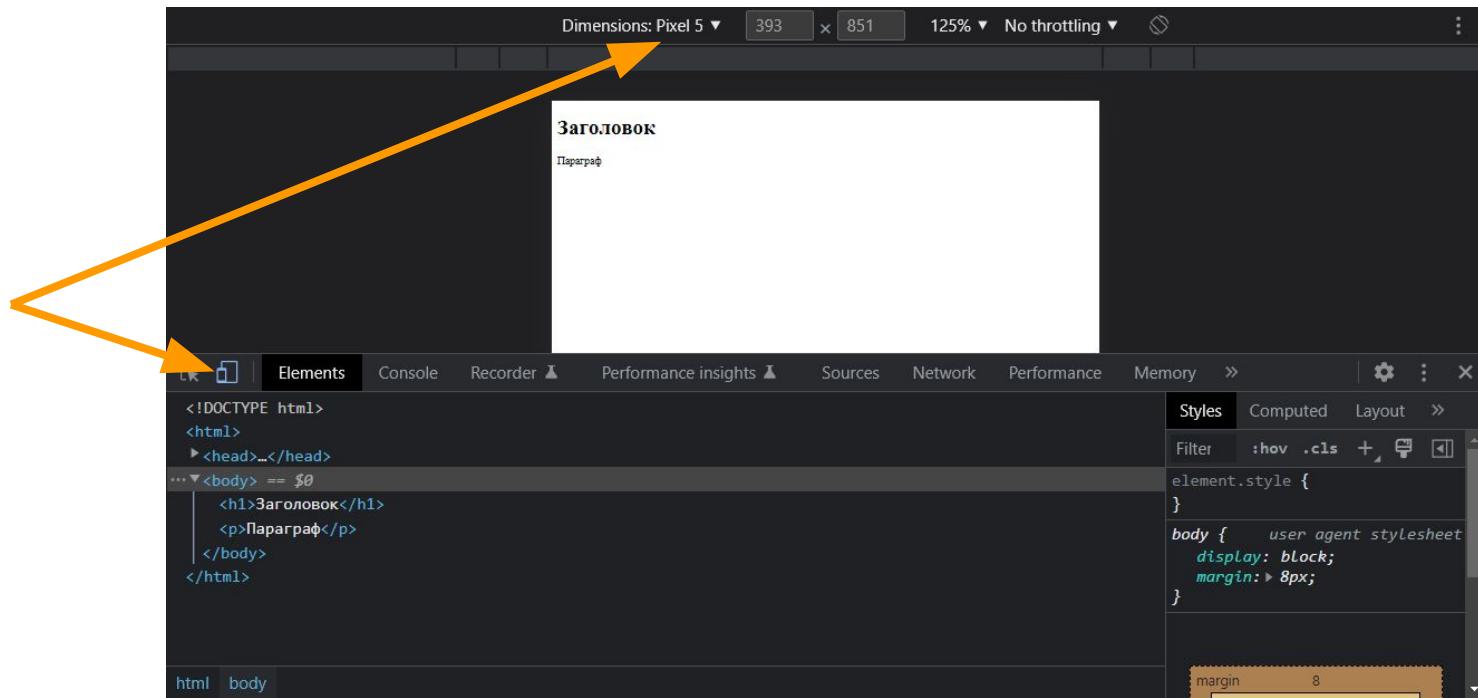
Справа отображаются стили выбранного элемента, а также его размеры вместе с отступами и рамкой.

Эта панель позволяет динамически менять стили прямо в браузере.  
(изменения сделанные здесь откатятся при обновлении страницы)

# Developer Tools

При создании и стилизации страницы бывает полезно увидеть, как она выглядит на мобильных устройствах. Инструменты разработчика поддерживают такую возможность.

При нажатии на иконку девайсов активируется режим симуляции мобильных устройств и в панели сверху можно выбрать соответствующее устройство.



# Block and inline elements

# Block elements

Всегда отображаются в браузерах с новой строки.

Примеры блочных элементов: <h1>, <p>, <ul> и <li>

```
<h1>Внимание!</h1>  
  
<p>Ниже приведены даты проведения  
вступительных испытаний.</p>  
  
<ul>  
    <li>Подача заявок: 15-25.05.22</li>  
    <li>Онлайн этап: 29.05.22</li>  
    <li>Заключительный этап: 03.06.22</li>  
    <li>Собеседование: 09.06.22</li>  
</ul>
```

## Внимание!

Ниже приведены даты проведения вступительных испытаний.

- Подача заявок: 15-25.05.22
- Онлайн этап: 29.05.22
- Заключительный этап: 03.06.22
- Собеседование: 09.06.22

# Inline elements

Отображаются на той же строке, что и соседние.

Примеры строчных элементов: `<a>`, `<b>`, `<em>` и `<img>`

При заполнении `<a href="list.pdf">`экзаменационного листа`</a>`, убедитесь, что вы корректно указали код регистрации и ФИО. ``  
При несоблюдении правил, описанных в `<em>`настоящей памятке`</em>`, ваша работа `<b>не будет засчитана!</b>`

При заполнении **экзаменационного листа**, убедитесь, что вы корректно указали код регистрации и ФИО.  При несоблюдении правил, описанных в **настоящей памятке**, ваша работа **не будет засчитана!**

# Grouping text and elements into a block

Элемент `<div>` позволяет сгруппировать несколько элементов в единый блок.

Содержимое div будет отображено с новой строки. Никаких никаких других изменений по-умолчанию не произойдёт.

Внимание!

```
<div>
   <b>При несоблюдении правил</b>
</div>
работа аннулируется
```

Внимание!



**При несоблюдении правил**

работа аннулируется

# Grouping text and elements into a line

Элемент `<span>` - строчный эквивалент элемента `<div>`. Используется для окружения части текста или нескольких строчных элементов.

Визуально по-умолчанию применение `span` никак не будет выделено браузером.

Внимание!

```
<span>
   <b>При несоблюдении правил</b>
</span>
работа аннулируется
```

Внимание!  **При несоблюдении правил** работа аннулируется

# **Div and span**

Рассмотренные контейнеры div и span используются для выделения нескольких элементов в единый блок для последующего обращения к нему из CSS.

В чистом HTML они мало чем полезны

# HTML Text tags

# Headers

В языке HTML существует шесть уровней заголовков:

```
<h1>Основной заголовок</h1>
<h2>Заголовок 2-го уровня</h2>
<h3>Заголовок 3-го уровня</h3>
<h4>Заголовок 4-го уровня</h4>
<h5>Заголовок 5-го уровня</h5>
<h6>Заголовок 6-го уровня</h6>
```

## Основной заголовок

### Заголовок 2-го уровня

### Заголовок 3-го уровня

### Заголовок 4-го уровня

### Заголовок 5-го уровня

### Заголовок 6-го уровня

*<h1>* используется для основных заголовков. *<h2>* - для подзаголовков.

Если текст разделен на фрагменты более глубокого уровня, то для них используются заголовки *<h3>* и ниже.

# Paragraphs

Для создания абзаца нужно заключить текст в открывающий и закрывающий теги `<p>`.

```
<p>Абзац1: Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed do eiusmod tempor  
incididunt ut labore et dolore magna aliqua.</p>  
<p>Абзац2: Ut enim ad minim veniam, quis nostrud  
exercitation ullamco laboris nisi ut aliquip ex ea  
commodo consequat.</p>  
<p>Абзац3</p>
```

Абзац1: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Абзац2: Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Абзац3

По умолчанию браузер автоматически отобразит каждый абзац с новой строки, при этом абзацы будут несколько отстоять друг от друга.

# Highlight Tags

`<strong>Полужирный</strong>`  
(или: `<b></b>`)

`<em>Курсив</em>`  
(или `<i></i>`)

`<ins>Подчеркнутый</ins>`  
(или `<u></u>`)

`<del>Зачеркнутый</del>`  
(или `<s></s>`)

`<strong>` используется для обозначения того, что текст, помещенный в него, имеет высокую степень важности. По умолчанию браузеры отображают его содержимое с полужирным начертанием.

`<em>` используется для обозначения логического ударения, которое несколько изменяет значение всего предложения. По умолчанию браузеры отображают его содержимое шрифтом с курсивным начертанием.

Полужирный Курсив Подчеркнутый Зачеркнутый

Несмотря на наличие этих тегов, более предпочтительным является указание таких эффектов через стили CSS.

# Subscripts and superscripts

`<sup>` используется для выделения символов, которые должны быть отображены как надстрочные, к примеру, ряд математических понятий.

`<sub>` используется для отображения символов как подстрочных. Как правило, используются в химических формулах.

```
<p>E = mc<sup>2</sup></p>
```

```
<p>Вода: H<sub>2</sub>O</p>
```

$$E = mc^2$$

Вода: H2O

# Spaces

Сворачивание пробелов - свойство браузера склоняться встречающиеся пробелы и символы перехода на новую строку в один пробел.

Для переноса текста на новую строку существует элемент `<br />`

`<hr />` помимо переноса создает горизонтальную разделяющую линию

```
<p>Привет  
мир</p>  
  
<p>Привет <br/> мир</p>  
  
<p>Привет <hr/> мир</p>
```

Привет мир

Привет  
мир

Привет

---

мир

# Special characters

Некоторые символы зарезервированы браузером и не могут использоваться как текст на HTML странице.

Например, знаки больше (>) и меньше (<) интерпретируются как часть тегов. Чтобы корректно отображать их у себя на странице нужно использовать **мнемоники**: &lt; и &gt;

Также с помощью мнемоник можно вставить на страницу любой символ следующим образом:  
“&#<код символа>;”

```
<p> &#65; &#960; &#128269; </p>
```



(Коды символов unicode вы можете найти в [специальных таблицах](#))

# Special characters

Давайте попробуем вывести в параграфе текст “`<h1> текст </h1>`”

```
<p>  
<h1> текст </h1>  
</p>
```



текст

```
<p>  
    &lt;h1&gt; текст &lt;/h1&gt;  
</p>
```

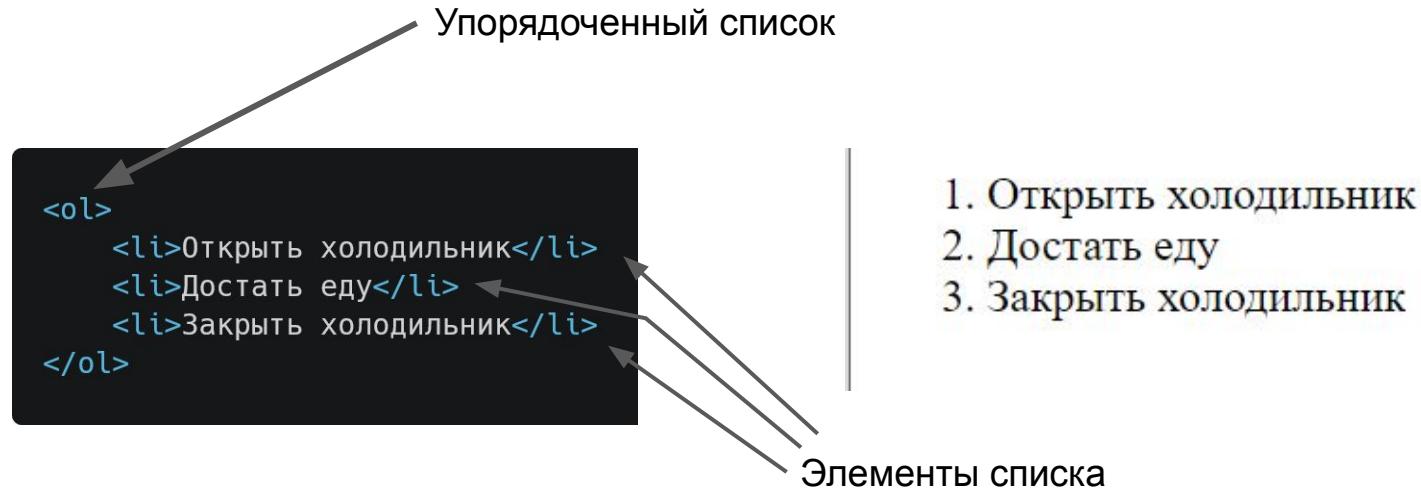


`<h1> текст </h1>`

# Lists

# Ordered list

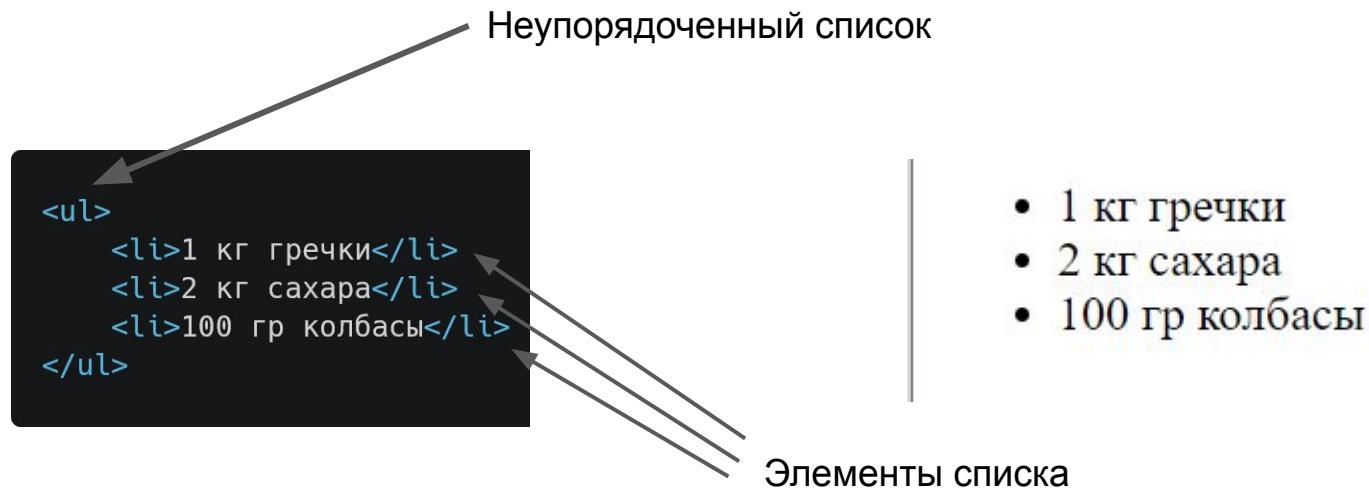
Упорядоченные списки - это списки, каждый элемент которых имеет порядковый номер. Например в качестве такого списка можно представить рецепт какого либо блюда, шаги которого необходимо выполнять в обозначенной последовательности.



По умолчанию браузеры автоматически расставляют нужные отступы для элементов списков.

# Unordered list

Неупорядоченные списки - это списки, рядом с каждым элементом которых помещается маркер (а не цифровые или буквенные символы, обозначающие порядковый номер).



# Nested lists

Внутри элемента `<li>` допустимо создание вложенного списка, или списка второго уровня.

```
<ul>
  <li>Муссы</li>
  <li>Пирожные
    <ul>
      <li>Круассаны</li>
    <li>Чизкейки</li>
  </ul>
  </li>
<li>Торты</li>
</ul>
```

- Муссы
- Пирожные
  - Круассаны
  - Чизкейки
- Торты

# Hyperlinks

# Links

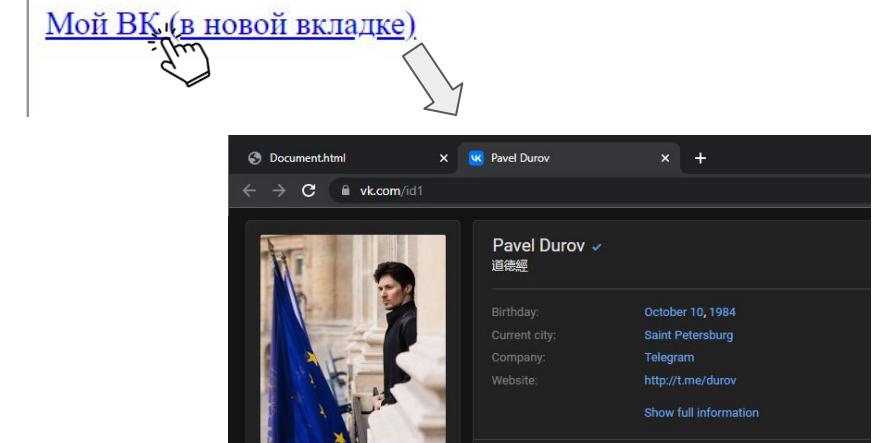
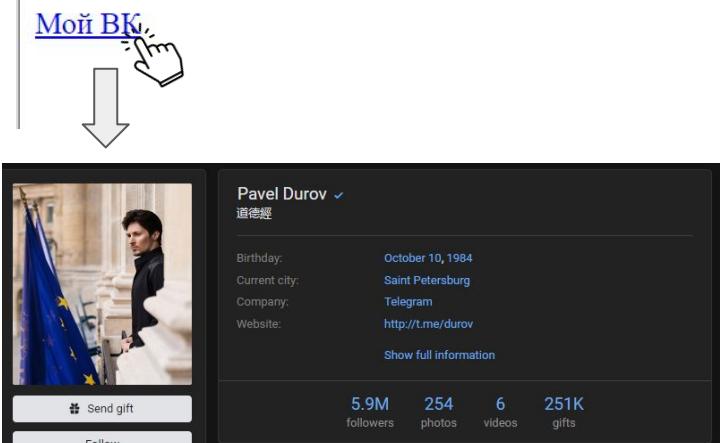
При щелчке по любому слову текста, помещенному между открывающим и закрывающим тегом `<a>` пользователь будет перенаправлен на страницу, адрес которой указан в атрибуте `href`.

```
<a href="адрес ссылки">Текст ссылки</a>
```

Атрибут `target="_blank"` говорит браузеру открыть новую страницу при клике на ссылку в новом окне.

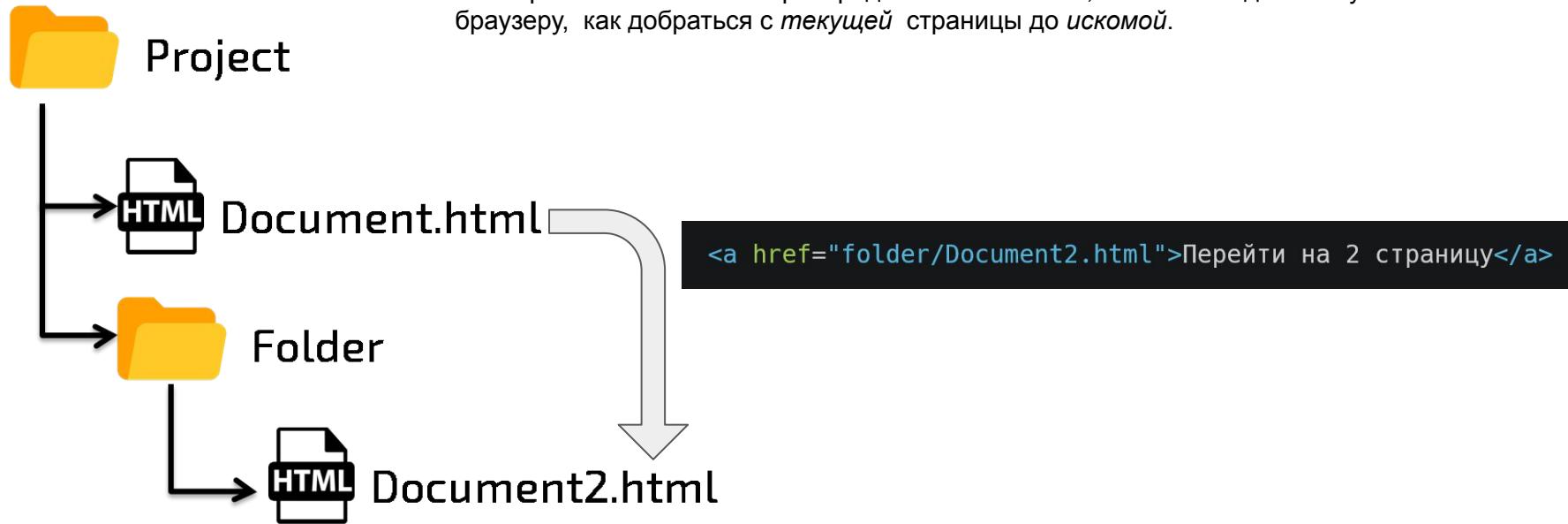
```
<a href="https://vk.com/id1">Мой ВК</a>
```

```
<a href="https://vk.com/id1" target="_blank">Мой ВК (в новой вкладке)</a>
```



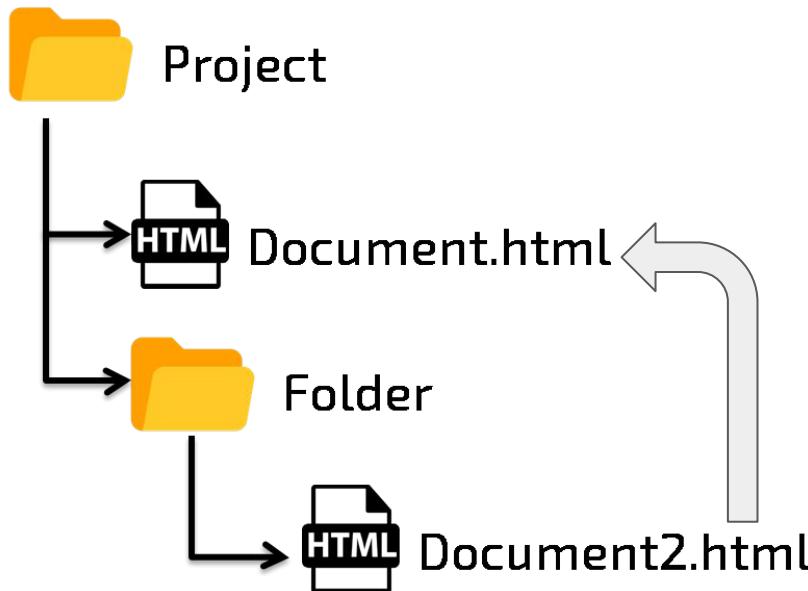
# Relative URLs

При создании ссылок на страницы собственного сайта нет необходимости указывать доменное имя. Вместо этого можно воспользоваться относительными URL-адресами.



# Relative URLs

Введите `../` для указания, что файл находится в папке одним уровнем выше, после чего укажите имя нужного файла.



```
<a href="../Document.html">Перейти на главную</a>
```

(Поднятие на каталог выше можно повторять и несколько раз, например относительный путь `../../../../Document.html` говорит браузеру, что из текущей папки нужно трижды выйти на уровень выше и только затем получить доступ к `Document.html`)

# **Id attribute**

Каждый HTML-элемент может иметь атрибут id.

Он используется для идентификации, поэтому его значение должно быть уникальным для каждого элемента на странице.

Значение атрибута может начинаться с латинской буквы или со знака подчеркивания (но не с цифры или другого символа).

Очень важно, чтобы значения атрибута id не повторялись на странице (в противном случае они не будут уникальными)

```
<p id="myid">Элемент</p>
```

# Link to a specific part of the page

Прежде чем мы сможем создавать ссылку на определенную часть страницы, сначала нужно отметить участок страницы, на который будет указывать ссылка. Это можно сделать с помощью атрибута *id*.

Теперь мы можем указать этот id в атрибуте href ссылки: `href="#<i>id элемента</i>"`

При клике по ссылке, браузер будет прокручивать страницу до нужного элемента.

```
<a href="#end">В конец</a>
```

```
<h1>Л.Н. Толстой - Спецоперация и мир</h1>
```

```
<p>Так говорила в июле 1805 года известная Анна Павловна Шерер,  
фрейлина и приближенная императрицы Марии Феодоровны...</p>
```

```
<p id="end">Конец</p>
```



# Tables

# Simple table

Для создания таблиц используется элемент `<table>`. Контент таблицы описывается построчно.

Открывающий тег `<tr>` обозначает начало новой строки таблицы. После него помещаются элементы , каждый из которых соответствует отдельной ячейке в этой строке. Конец строки обозначается закрывающим тегом `</tr>`.

Каждая ячейка таблицы представляется элементом `<td>`. Конец ячейки обозначается закрывающим тегом `</td>`.

```
<table>
  <tr>
    <th>МИФИ</th>
    <th>МФТИ</th>
  </tr>
  <tr>
    <td>100 очков</td>
    <td>120 очков</td>
  </tr>
</table>
```

Элемент `<th>` используется точно так же, как и элемент `<td>`, однако его назначение - создание заголовка строки или столбца. По умолчанию выделяется браузером жирно.

**МИФИ МФТИ**  
100 очков 120 очков

# Setting the border and color

С помощью атрибута **border** у таблицы можно задать рамку. Значение этого атрибута задает ширину рамки в пикселях.

Атрибут **bgcolor** используется для установки цвета всей таблицы либо отдельных ячеек. Его значением, как правило, является шестнадцатеричный код цвета.

```
<table border="2">
  <tr>
    <th>МИФИ</th>
    <th>МФТИ</th>
  </tr>
  <tr>
    <td bgcolor="#aaaaaa">100 очков</td>
    <td>120 очков</td>
  </tr>
</table>
```

МИФИ	МФТИ
100 очков	120 очков

# Setting the width and spacing

Атрибут **width** используется в открывающем теге `<table>` для указания ширины таблицы. Кроме того, его можно ставить в открывающих тегах элементов `<th>` и `<td>` для задания ширины отдельных ячеек таблицы.

В открывающем теге элемента `<table>` также может быть указан атрибут **cellpadding**, добавляющий промежуток между границами ячейки и ее содержимым, а также атрибут **cellspacing**, задающий величину промежутка между ячейками таблицы.

Внутренние и внешние отступы

```
<table border="2" cellpadding="20px" cellspacing="10px">
  <tr>
    <th width="200px">МИФИ</th>
    <th width="100px">МФТИ</th>
  </tr>
  <tr>
    <td>100 очков</td>
    <td>120 очков</td>
  </tr>
</table>
```

(Значения всех этих атрибутов устанавливаются в пикселях)

The diagram shows a 2x2 grid of four cells. The top-left cell contains 'МИФИ', the top-right 'МФТИ', the bottom-left '100 очков', and the bottom-right '120 очков'. Each cell is a white rectangle with a thin black border. There is a larger gap between the columns and rows, which is the result of the table's border plus cellpadding and cellspacing.

МИФИ	МФТИ
100 очков	120 очков

Ширина ячеек столбца без учёта, внутренних и внешних отступов (cellpadding, cellspacing)

# Content alignment in the table

Для выравнивания контента в ячейках таблиц существует атрибут **align**, который может принимать 3 значения: *left*, *right* или *center*.

```
<table border="2">
  <tr>
    <th width="100px">МИФИ</th>
    <th width="200px">МФТИ</th>
    <th width="100px">ВШЭ</th>
  </tr>
  <tr>
    <td align="left">100 очков</td>
    <td align="center">120 очков</td>
    <td align="right">110 очков</td>
  </tr>
</table>
```

(<th> автоматически использует позиционирование по центру)

МИФИ	МФТИ	ВШЭ
100 очков	120 очков	110 очков

Выравнивание по левому краю, центру и правому краю

(В html5 рекомендуется заменять атрибут **align** на **style="text-align: left | right | center;"**. То есть использовать CSS)

# Merge columns

Иногда может понадобиться, чтобы ячейка таблицы занимала не один, а несколько столбцов. Атрибут **colspan** может быть использован с элементами **<th>** и **<td>** для обозначения, сколько столбцов должна занимать ячейка.

```
<table border="2">
  <tr>
    <th>МИФИ</th>
    <th>МФТИ</th>
  </tr>
  <tr>
    <td colspan="2">220 очков</td>
  </tr>
</table>
```

МИФИ	МФТИ
220 очков	

# Merge rows

Кроме того, может понадобиться сделать так, чтобы одна ячейка занимала несколько строк. Атрибут **rowspan** может быть использован с элементами **<th>** и **<td>** для обозначения, на сколько строк должна простираться ячейка.

```
<table border="2">
  <tr>
    <th rowspan="2">МИФИ</th>
    <th>МФТИ</th>
  </tr>
  <tr>
    <td>120 очков</td>
  </tr>
</table>
```

МИФИ	МФТИ
	120 очков

# Tabular layout

```
<table width="100%">
  <tr>
    <td align="left" width="30%">
      
      <h2>Cat studio</h2>
    </td>
    <td align="center" width="20%">
      <a href="/products">Товары</a>
    </td>
    <td align="center" width="20%">
      <a href="/services">Услуги</a>
    </td>
    <td align="center" width="20%">
      <a href="/about">О нас</a>
    </td>
  </tr>
  <tr>
    <td colspan="3">
      <h1>Вас приветствует CatStudio!</h1>
      <p>У нас есть для вас несколько специальных предложений.</p>
    </td>
    <td>
      
    </td>
  </tr>
</table>
```

Таблицы можно использовать для создания макета сайта и позиционирования элементов на странице.

(Табличная разметка распространена при вёрстке email писем, потому что другие способы позиционирования там не работают)



[Товары](#)

[Услуги](#)

[О нас](#)

## Cat studio

# Вас приветствует CatStudio!

У нас есть для вас несколько специальных предложений.



# Forms

# Form structure

```
<form action="script.php" method="post">  
    <!-- элементы формы -->  
</form>
```

Каждый элемент формы должен иметь имя(атрибут *name*) и значение (атрибут *value*). При отправке формы все данные формы будут отправлены на URL указанный в *action* и методом указанным в *method*.

Данные будут иметь вид:

“имяЭлемента1=значение1&имяЭлемента2=значение2&...”

# Text input

Большинство элементов ввода создаются с помощью тега `<input>`.

Устанавливая его атрибут **type**, мы говорим браузеру, какой именно элемент ввода мы хотим. Например `type="text"` создает обычное текстовое поле ввода.

С помощью атрибута **value** мы можем установить значение, которое будет иметь элемент при загрузке страницы.

```
<input type="text" name="username" value="ваше имя"/>
```

ваше имя

С помощью атрибута **placeholder** мы можем задать подсказку для поля ввода, которая исчезнет как только пользователь начнет вводить текст.

```
<input type="text" name="username" placeholder="введите имя"/>
```

введите имя

# Text input

Установив значение **type="password"**, мы можем создать поле ввода пароля.

```
<input type="password" name="userpass"/>
```



.....

Поле многострочного ввода создается с помощью тега **<textarea></textarea>**, а не **<input>**!

```
<textarea name="about" placeholder ="расскажите о себе"></textarea>
```



расскажите о себе

# Autocomplete and autofocus

С помощью атрибута **autocomplete** мы можем указать, должен ли браузер выводить подсказки заполнения поля с учетом данных, которые пользователь вводил ранее. (автозаполнение может быть отключено в настройках браузера)

```
<input type="text" name="username" autocomplete="on" />
```

A screenshot of a web browser interface. At the top is an input field with the placeholder text 'Username'. Below the input field are two dark rectangular buttons containing the names 'Artemon' and 'Aramiko'. To the right of the input field is a light-colored button labeled 'Send'.

```
<input type="text" name="captcha" autocomplete="off" />
```

A screenshot of a web browser interface. At the top is an empty input field with the placeholder text 'Captcha'. To its right is a light-colored button labeled 'Send'.

Атрибут **autofocus** делает так, чтобы при загрузке страницы фокус был на данном элементе ввода. Чтобы пользователь мог сразу начать вводить текст.

```
<input type="text" name="username" autofocus />
```

A screenshot of a web browser interface. At the top is an empty input field with the placeholder text 'Username'. To its right is a light-colored button labeled 'Send'.

```
<input type="text" name="username"/>
```

A screenshot of a web browser interface. At the top is an empty input field with the placeholder text 'Username'. To its right is a light-colored button labeled 'Send'.

# Switches

Переключатели позволяют пользователю выбрать только один вариант из предложенных. Они создаются установкой атрибута `type="radio"` у тега `<input>`

Несколько меток, объединенных одним именем составляют группу.

```
Ваш пол:<br/>
Женский <input type="radio" name="gender" value="f"/>
Мужской <input type="radio" name="gender" value="m"/>
Не скажу <input type="radio" name="gender" value="n" checked/>
```

`value` выбранного элемента будет отправлено на сервер под именем группы

Атрибут `checked` позволяет выбрать по умолчанию определенный переключатель

Ваш пол:  
Женский  Мужской  Не скажу

(В данном случае при отправке формы, на сервер бы отправилось: `gender=n`)

# Switch clickability

Проблема: кликабельны только кружки. При нажатии на надпись ничего не происходит.

Потому что мы никак не связали эти надписи с переключателями.

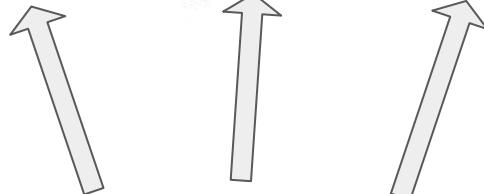


# Form element labels

Метки создаются с помощью элемента `<label>`. Чтобы привязать метку к нужному полю ввода нужно указать `id` этого поля в атрибуте `for` метки.

```
Ваш пол:<br/>
<label for="female">Женский</label> <input type="radio" name="gender" value="f" id="female"/>
<label for="male">Мужской</label> <input type="radio" name="gender" value="m" id="male"/>
<label for="none">Не скажу</label> <input type="radio" name="gender" value="n" checked id="none"/>
```

Ваш пол:  
Женский  Мужской  Не скажу



Теперь тоже кликабельно

Метки можно использовать с любыми элементами форм. При клике на метку произойдёт фокус на привязанному к ней элементу ввода и в случае с переключателями будет выбран соответствующий вариант.

# Checkboxes

С помощью флагков посетители сайта могут выбирать из нескольких вариантов ответа, а также отменять выбор. Создаются они указанием атрибута `type="checkbox"` у тега `<input>`

Несколько меток, объединенных одним именем составляют группу.

```
Какая дата вам удобна?  
25.05 <input type="checkbox" name="date" value="25"/>  
26.05 <input type="checkbox" name="date" value="26" checked />  
27.05 <input type="checkbox" name="date" value="27" checked />
```

`value` выбранных элементов будут отправлены на сервер под именем группы

Атрибут `checked` позволяет выбрать по умолчанию определенные флагки

\*Здесь и далее привязка меток опущена для краткости кода

Какая дата вам удобна?  
25.05  26.05  27.05

(В данном случае при отправке формы, на сервер бы отправилось: `date=26&date=27`)

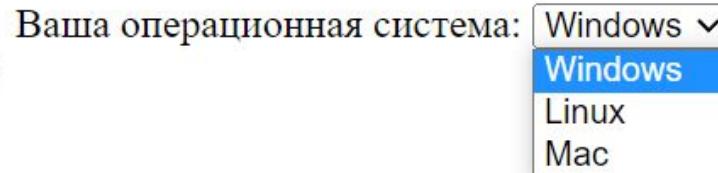
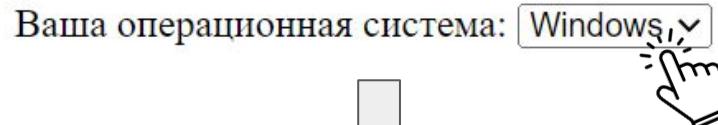
# Drop-down list

Раскрывающийся список появляется при щелчке мышью по элементу формы и позволяет посетителю сайта выбрать один вариант. Для создания раскрывающегося списка используется элемент `<select>`. Он может содержать два и более элементов `<option>`.

Функционирование раскрывающегося списка аналогично переключателям (может быть выбран только один вариант). Список обычно используется вместо переключателей, когда вариантов выбора слишком много.

Ваша операционная система:

```
<select name="os">
    <option value="w" selected>Windows</option>
    <option value="l">Linux</option>
    <option value="m">Mac</option>
</select>
```



# Multiselect list

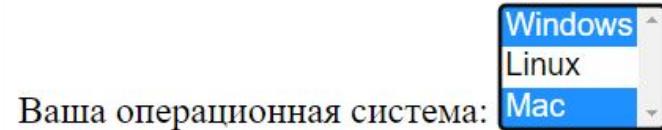
Добавив списку атрибут **multiple** мы можем позволить посетителям сайта выбрать сразу несколько вариантов ответа.

Указав атрибут **size**, мы превращаем раскрывающийся список в поле, отображающее сразу несколько вариантов выбора. Значением этого атрибута должно быть количество пунктов списка, отображаемых за раз.

Функционирование списка с мультивыбором аналогично флажкам.

Ваша операционная система:

```
<select name="os" size="3" multiple>
  <option value="w">Windows</option>
  <option value="l">Linux</option>
  <option value="m">Mac</option>
</select>
```



Ваша операционная система:

\*В браузере мультивыбор осуществляется с помощью зажатого Ctrl.

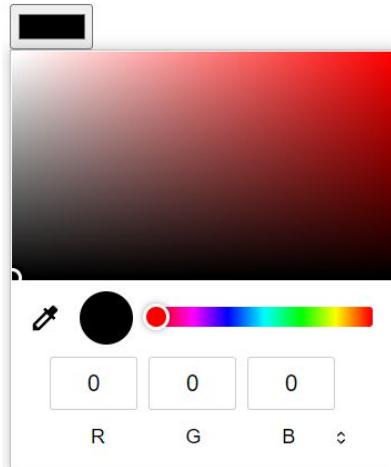
Об этом знают не все пользователи поэтому лучше сопровождать список с мультивыбором соответствующей подсказкой.

# Color and date selection

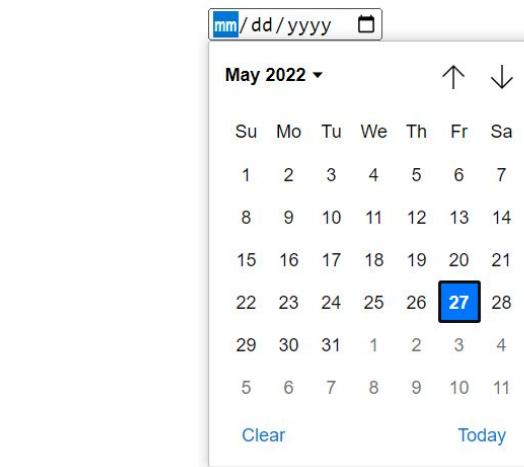
Ещё стоит упомянуть поля выбора цвета и даты.

Создаются они также заданием тегу <input> соответствующего атрибута type:

```
<input type="color" name="color" />
```



```
<input type="date" name="date" />
```



# Uploading a file to the server

Значение атрибута **type="file"** у тега `<input>` создает поле, по внешнему виду напоминающее поле ввода текста, после которого помещается кнопка Обзор (Browse). Когда посетитель щелкает по ней мышью, браузер открывает диалоговое окно, позволяющее выбрать нужный файл на компьютере.

Прикрепите файл:

```
<input type="file" name="document"/>
```

Прикрепите файл:  No file chosen

Для корректной отправки файлов необходимо также установить атрибут **enctype="multipart/form-data"** для формы, а также **method="post"**.

Как вы уже знаете, по умолчанию данные формы отправляются в виде  
`"name1=value1&name2=value2&name3=value3..."`. Этот способ кодирования называется  
**application/x-www-form-urlencoded**. Он используется по умолчанию, потому что отправляет данные в компактном виде и экономит трафик. Но минус такого кодирования состоит в том, что с помощью него нельзя отправлять бинарные данные (файлы). Поэтому для отправки файлов мы должны сменить тип кодирования на **multipart/form-data** (с точки зрения фронтенда разницы между ними нет).

# Submit button

Кнопка подтверждения используется для отправки формы с данными на сервер.  
Она создаётся указанием атрибута **type="submit"** у тега <input>

```
<input type="submit" name="btn" value="Подписаться" />
```

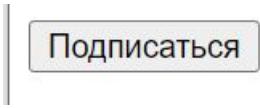
Подписаться

При клике отправляет всю форму по адресу указанному в атрибуте *action* формы.

Без нее форма не отправится, Поэтому такая кнопка **должна присутствовать в каждой форме!**

# Submit button. New style

```
<button>Подписаться</button>
```



Подписаться

Делает то же, что и предыдущая. Отличается лишь тем что внутрь нее можно размещать различные html элементы (например, картинки), а не только текст надписи.

# Hidden field

```
<input type="hidden" name="token" value="bfe3d5a5a367d8e7af45be6b1a5e504f"/>
```

Не отображается на экране у пользователя (но видна в коде страницы).  
Отправляется вместе со всеми остальными полями на сервер при отправке  
формы.

В основном служит для снабжения отправленной пользователем формы  
служебной информацией.

# Html validation

## Атрибут required

```
<input type="text" name="username" required />
<input type="submit" name="btn" value="Отправить" />
```

Please fill out this field.

\*Не даёт отправить форму не заполнив поле

## Атрибуты minlength и maxlength

```
<input type="text" name="username" minlength="3" maxlength="10" />
<input type="submit" name="btn" value="Отправить" />
```

Please lengthen this text to 3 characters or more (you are currently using 2 characters).

\*Не даёт вбить менее 3 и более 10 символов

# Html validation

Поле ввода целого числа (`type="number"`)

```
<input type="number" name="mail" min="5" max="10" />
<input type="submit" name="btn" value="Отправить" />
```

A screenshot of a web browser showing a validation error for a number input field. The input field contains the value '4'. To its right is a small orange dropdown arrow icon. Next to the input is a button labeled 'Отправить' (Send). A validation message box is displayed below the input field, containing an exclamation mark icon and the text: 'Value must be greater than or equal to 5.'

Поле ввода email (`type="email"`)

```
<input type="email" name="mail" />
<input type="submit" name="btn" value="Отправить" />
```

A screenshot of a web browser showing a validation error for an email input field. The input field contains the value 'mephi'. To its right is a small orange dropdown arrow icon. Next to the input is a button labeled 'Отправить' (Send). A validation message box is displayed below the input field, containing an exclamation mark icon and the text: 'Please include an '@' in the email address. 'mephi' is missing an '@'.'

Поле ввода ссылок (`type="url"`)

```
<input type="url" name="link" />
<input type="submit" name="btn" value="Отправить" />
```

A screenshot of a web browser showing a validation error for a URL input field. The input field contains the value 'mephi'. To its right is a small orange dropdown arrow icon. Next to the input is a button labeled 'Отправить' (Send). A validation message box is displayed below the input field, containing an exclamation mark icon and the text: 'Please enter a URL.'

# Validation by pattern

Еще одной полезной функцией проверки является атрибут **pattern**, который в качестве значения принимает *регулярное выражение*.

*Регулярное выражение* (regex) - это шаблон, который можно использовать для поиска комбинаций символов в текстовых строках. Синтаксис регулярных выражений по сути является отдельным языком создания шаблонов для строк и не привязан конкретно к HTML, CSS и Javascript. С регулярными выражениями вы можете познакомиться в тех же [статьях на хабре](#) или на сайте [learn.javascript.ru](http://learn.javascript.ru), где им посвящен целый раздел.

```
<input type="text" name="tel" pattern="^\+[0-9]{11}$" />
<input type="submit" name="btn" value="Отправить" />
```

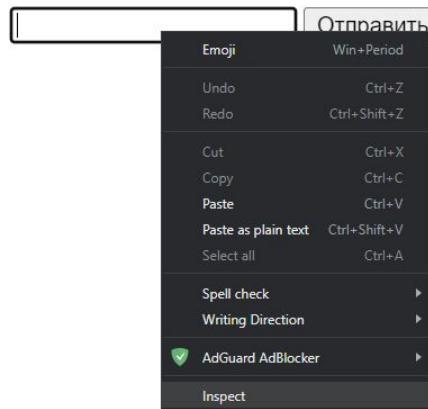
 Please match the requested format.

# Validation and security

Клиентская валидация легко обходится и не заменяет валидацию на стороне сервера!

A screenshot of a web form. A text input field contains the value "azaza". To its right is a button labeled "Отправить" (Send). Below the input field is a yellow rectangular box containing a black exclamation mark icon and the text "Please match the requested format."

Клиентская валидация  
нужна лишь чтобы не  
заставлять пользователя  
 заново заполнять форму  
 из-за одного неправильно  
 заполненного поля.



A screenshot of a web form. The input field is empty and highlighted with a blue border. To its right is a button labeled "Отправить" (Send). Below the input field is a status message: "input 170 x 21.33".

A screenshot of the browser's developer tools Elements tab. It shows the HTML structure of a form. The input field has the name "tel" and a pattern attribute set to "[0-9]{11}". The submit button has the name "btn" and a value attribute set to "Отправить". An orange arrow points from the text "Можно легко заставить браузер удалить проверку" to the "value" attribute of the submit button.

```
<html>
  <head>...</head>
  <body>
    <form action="script.php" method="post">
      <input type="text" name="tel" pattern="[0-9]{11}">
      <input type="submit" name="btn" value="Отправить">
    </form>
  </body>
</html>
```

Можно легко заставить браузер  
удалить проверку

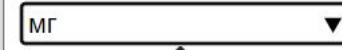
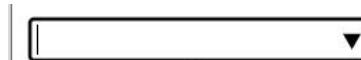
(также можно просто отправить данные напрямую на url обработчика в обход формы)

# Input prompts

Для добавления подсказок к полю ввода нужно создать список подсказок с помощью тега `<datalist>`. Он как и обычный список должен содержать параметры `<option>`, но в отличие от обычного списка нас в `option` интересует только параметр `value` (а не содержимое между его открывающим и закрывающим тегом).

После создания списка мы можем привязать его к инпуту. Для этого надо указать `id` списка в параметре `list` соответствующего поля ввода.

```
<input type="text" list="universities">
<datalist id="universities">
    <option value="МИФИ"></option>
    <option value="МФТИ"></option>
    <option value="МГИМО"></option>
    <option value="МГУ"></option>
    <option value="ВШЭ"></option>
</datalist>
```



# Multimedia and embedding

# Images: <img/> tag

Атрибут *src* - URL картинки

Атрибут *alt* - текст высвечивающийся когда не удалось загрузить картинку (например из-за плохого интернета)

Атрибут *title* - текст всплывающей подсказки при наведении на картинку

Атрибуты *width*, *height* - ширина и высота картинки

Например:

```

```

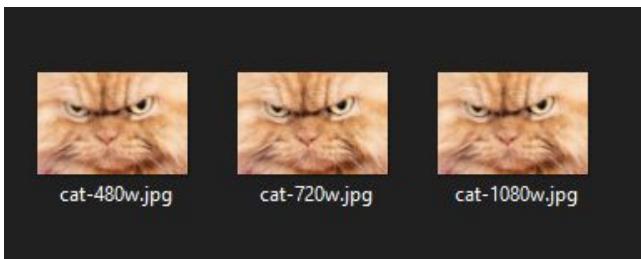


# Adaptive Images

С бурным развитием мобильных устройств оптимизация под мобильные платформы стала по сути стандартом. Для оптимизации изображений в html5 предусмотрены атрибуты **srcset** и **sizes**. Они позволяют задать различные изображения для различных размеров устройств.

Можно конечно просто указать в атрибуте src самую большую картинку и потом масштабировать её с помощью width и height, но такой подход заставляет браузер всегда скачивать большое изображение, что может оказаться на скорости загрузки сайта, особенно на мобильных устройствах с плохим интернетом. Нужно решать эту проблему.

Для этого создадим копии изображения различных размеров.



# Adaptive Images

В атрибуте **src** изображения указываем самое маленькое изображение.

В атрибуте **srcset** через запятую перечисляем размеры наших заготовленных картинок следующим образом: **srcset="<путь к картинке 1> <размер>w, <путь к картинке 2> <размер>w, ..."**

В атрибуте **sizes** мы указываем при каких условиях какие размеры браузеру следует использовать.  
**sizes="(условие 1) <размер>, (условие 2) <размер>, ..."**

В итоге браузер выберет размер, соответствующий первому выполненному условию из этого списка.

```

```

Перечисляем размеры наших картинок

Пишем условия. (Если ширина экрана не превышает 600px, то будет выбран размер картинки 480. Если ширина превышает 600px, но не превышает 840px, то будет выбран размер 720px. Иначе (если ширина превышает 840px) будет выбран размер 1080)

# Embedding video

Атрибут *src* - URL видеофайла

Атрибут *poster* - URL превью для видео

Атрибуты *width* и *height* - ширина и высота блока с видео

Атрибут *controls* - устанавливает стандартные браузерные элементы управления.

Атрибут *loop* - заставляет браузер при окончании видео проигрывать его снова

```
<video src="video.mp4"
       poster="preview.png"
       width="300"
       height="150"
       controls
       loop>
<p>Ваш браузер не поддерживает видео</p>
</video>
```



# Embedding audio

Атрибут *src* - URL аудиофайла

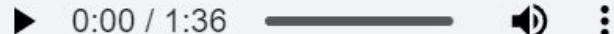
Атрибут *controls* - устанавливает стандартные браузерные элементы управления.

Атрибут *loop* - заставляет браузер при окончании аудио проигрывать его снова

```
<audio src="audio.mp3"  
       controls  
       loop>
```

```
<p>Ваш браузер не поддерживает аудио</p>  

```



# Embedding HTML

Элемент <iframe> позволяет вам встраивать в страницу контент других страниц.

```
<h1>Основная html страница</h1>
<iframe width="300" height="150" src="anotherPage.html"></iframe>
```

```
(anotherPage.html)
<html>
  <body>
    <h1>Другая html страница</h1>
    <p>Hello, world</p>
  </body>
</html>
```

**Основная html страница**

**Другая html  
страница**

Hello, world

# Setting Iframe Restrictions

Iframe позволяет устанавливать ограничения на страницу, которая в нём открыта. Делается это с помощью атрибута **sandbox**. При установленном атрибуте страница превращается в “песочницу”, где применены все ограничения.

Если мы хотим снять некоторые ограничения, то нужно перечислить их через пробел в значении атрибута **sandbox**. Основные снятия ограничений:

1. *allow-same-origin* (Разрешает загружать содержимое фрейма, воспринимая его из того же источника, что и родительский документ)
2. *allow-top-navigation* (Позволяет открывать ссылки фрейма в родительском документе)
3. *allow-forms* (Позволяет содержимому фрейма отправлять формы)
4. *allow-scripts* (Разрешает запуск и выполнение скриптов.)

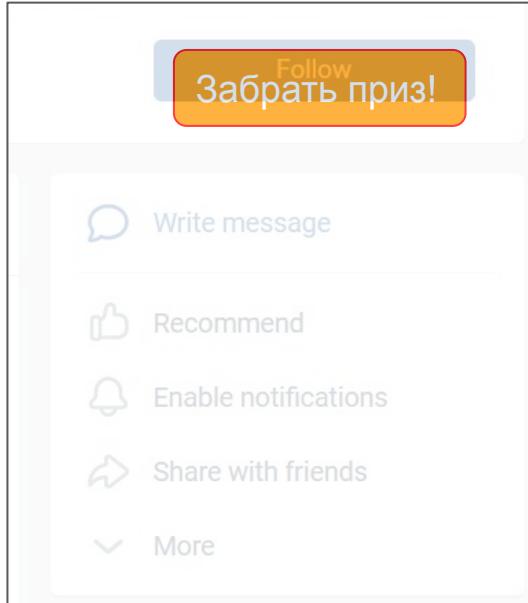
(Полный список вы можете посмотреть [здесь](#) )

Например, следующий фрейм запрещает на открываемой странице всё, кроме форм и навигации:

```
<iframe src="page.html" sandbox="allow-forms allow-top-navigation"></iframe>
```

# Clickjacking attack

Атака “угон клика” была в своё время распространена и основывалась на использовании iframe.



Алгоритм атаки следующий:

1. Пользователя заманивают на мошеннический сайт на странице которого есть кнопка привлекающая внимание.
2. Перед кнопкой помещается прозрачный iframe с сайтом, на котором пользователь авторизован. (прозрачность устанавливается с помощью css)
3. Таким образом щелкая по кнопке, пользователь щёлкает по невидимому iframe перед ней и на авторизованном сайте совершается действие от имени этого пользователя.

Для защиты от clickjacking был придуман заголовок:

**X-Frame-Options: SAMEORIGIN**, при получении которого браузер не позволит открыть ваш сайт в чужом iframe.

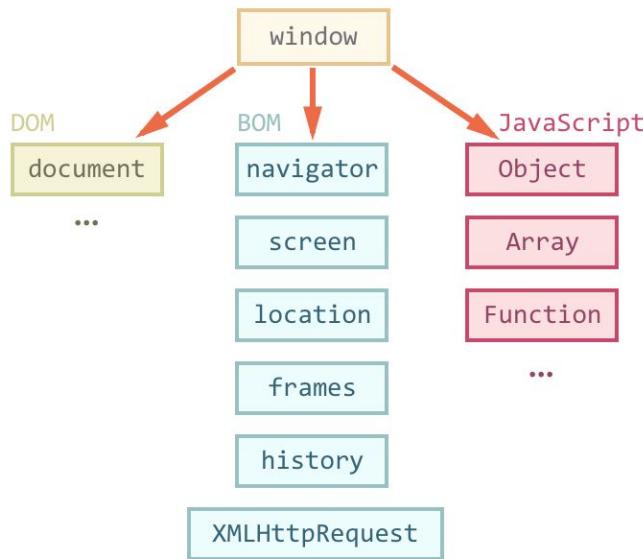
**X-Frame-Options: DENY** не позволит открывать вашу страницу в iframe даже на собственном сайте.

Все современные соц. сети отправляют данный заголовок, так что вы не сможете открыть их у себя в iframe

# DOM API

# Browser environment (window object)

Взаимодействие с браузерным API в JavaScript осуществляется через глобальный объект **window**.



DOM (Document Object Model)

```
> window.document
<  ▾#document
    <!DOCTYPE html>
    <html>
        <head>...</head>
        <body>...</body>
    </html>
```

Все методы браузерного API являются методами объекта **window**

```
> window.XMLHttpRequest
< f XMLHttpRequest() { [native code] }
> window.alert
< f alert() { [native code] }
> window.setInterval
< f setInterval() { [native code] }
```

BOM (Browser Object Model)

```
> window.navigator
<  ▾ Navigator {vendorSub: '', productSub: '20030107', vendor: 'Google Inc.', maxTouchPoints: 0, scheduling: Scheduling, ...}
> window.screen
<  ▾ Screen {availWidth: 1920, availHeight: 1040, width: 1920, height: 1080, colorDepth: 24, ...}
> window.location
<  ▾ Location {ancestorOrigins: DOMStringList, href: 'file:///C:/Users/aramt/Desktop/dom.html', origin: 'file:///', protocol: 'file:', host: '', ...}
```

При использовании свойств и методов объекта **window**, сам **window** можно опустить

```
> XMLHttpRequest
< f XMLHttpRequest() { [native code] }
> alert
< f alert() { [native code] }
> setInterval
< f setInterval() { [native code] }
```

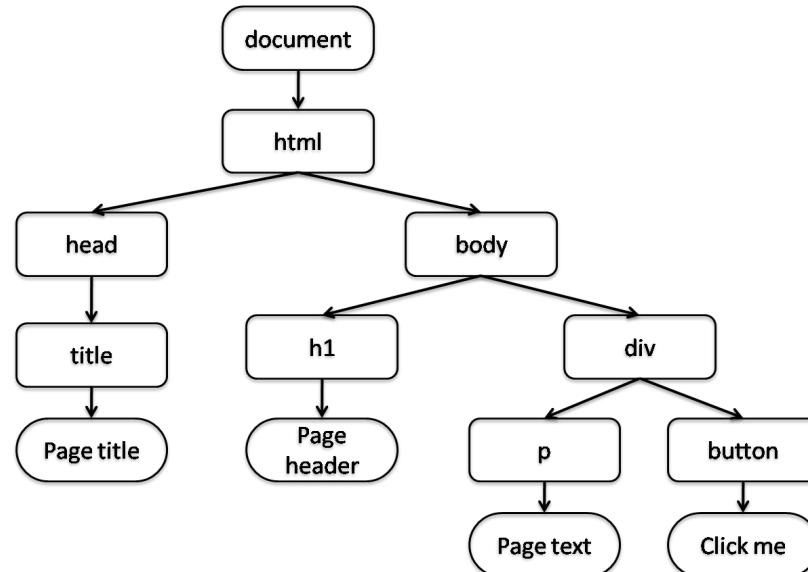
# DOM

В соответствии с **DOM** (Document Object Model), каждый HTML-тег является объектом. Вложенные теги являются детьми родительского элемента.

Все эти объекты можно менять в режиме реального времени при помощи JavaScript.

```
<!DOCTYPE html>

<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>Page header</h1>
    <div>
      <p>Page text</p>
      <button>Click me</button>
    </div>
  </body>
</html>
```



# Navigation through html elements

`document.head`

- получение head элемента страницы

`document.body`

- получение body элемента страницы

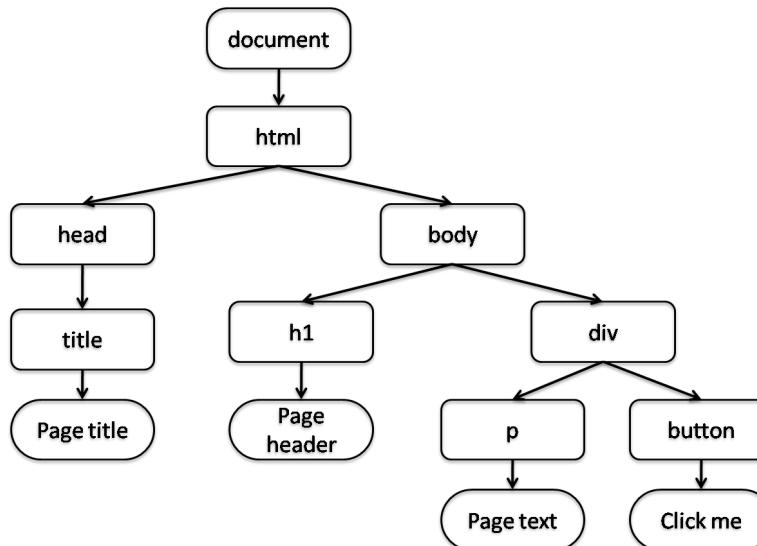
`element.children`

- получение детей элемента

`element.parentElement`

- получение родителя элемента

```
> document.head  
<- ▶ <head>...</head>  
> document.head.children  
<- ▶ HTMLCollection [title]
```



```
> document.body  
<- ▶ <body>...</body>  
> document.body.children  
<- ▶ HTMLCollection(2) [h1, div]  
  
> document.body.children[1].children  
<- ▶ HTMLCollection(2) [p, button]
```

# An example of using the DOM

С помощью DOM мы можем динамически менять нашу HTML страницу.

```
<!DOCTYPE html>  
  
<html>  
  <head>  
    <title>Page title</title>  
  </head>  
  <body>  
    <h1>Page header</h1>  
    <div>  
      <p>Page text</p>  
      <button>Click me</button>  
    </div>  
  </body>  
</html>
```

# Page header

Page text

Click me

```
let header = document.body.children[0]  
let button = document.body.children[1].children[1]  
  
button.onclick = function() {  
  header.innerHTML = "Changed header!"  
}
```

получаем объекты нужных элементов

устанавливаем событие нажатия на кнопку

меняем содержимое заголовка при нажатии на кнопку

# Getting HTML elements

Существуют более удобные способы получения HTML элементов, чем через свойство `children` у `document.body`

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1 id="header">Page header</h1>
    <div>
      <p class="paragraph">Page text</p>
      <button>Click me</button>
    </div>
  </body>
</html>
```

Получение элемента по id

```
> document.getElementById("header")
<-- <h1 id="header">Page header</h1>
```

Получение элементов по классу

```
> document.getElementsByClassName("paragraph")
<-- ▶ HTMLCollection [p.paragraph]
```

Получение элементов по тегу

```
> document.getElementsByTagName("button")
<-- ▶ HTMLCollection [button]
```

# Getting HTML elements

Существуют более удобные способы получения HTML элементов, чем через свойство `children` у `document.body`

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1 id="header">Page header</h1>
    <div>
      <p class="paragraph">Page text</p>
      <button>Click me</button>
    </div>
  </body>
</html>
```

Получение элемента по css селектору

```
> document.querySelector("#header")
<-- <h1 id="header">Page header</h1>
> document.querySelector(".paragraph")
<-- <p class="paragraph">Page text</p>
> document.querySelector("button")
<-- <button>Click me</button>
```

Получение всех элементов по css селектору

```
> document.querySelectorAll("#header")
<-- > NodeList [h1#header]
> document.querySelectorAll(".paragraph")
<-- > NodeList [p.paragraph]
> document.querySelectorAll("button")
<-- > NodeList [button]
```

# Content of the HTML element

После получения объекта HTML элемента мы можем читать/менять его содержимое с помощью свойств `innerText`, `innerHTML`, `outerHTML`.

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1 id="header">Page header</h1>
    <div>
      <p class="paragraph">Page text</p>
      <button>Click me</button>
    </div>
  </body>
</html>
```

```
> let element = document.querySelector("div")
  element.innerText
<-- 'Page text\n\nClick me'
> element.innerHTML
<-- '\n\t\t\t

Page text</p>\n\t\t\t<button>Click me</button>\n\t\t\t'
> element.outerHTML
<-- '<div>\n\t\t\t<button>Click me</button>\n\t\t</div>'


```

Текст внутри тега  
HTML внутри тега  
Полный HTML тега

## Page header

Page text



## Page header

Hello

Данные свойства можно записывать и тем самым менять контент элементов

```
> element.innerHTML = "<p>Hello</p>"
<-- '<p>Hello</p>'
```

Click me

# Attributes of html elements

С помощью методов **getAttribute** и **setAttribute** можно читать и устанавливать любые атрибуты HTML элементу.

```
<a id="mylink" href="https://vk.com" target="_blank">Текст ссылки</a>
```

## Чтение атрибутов

```
> let element = document.querySelector("#mylink")
> element.getAttribute("id")
< 'mylink'
> element.getAttribute("href")
< 'https://vk.com'
> element.getAttribute("target")
< '_blank'
```

## Установка атрибутов

```
> element.innerText = "Новый текст ссылки"
element.setAttribute("href", "https://ok.ru")
element.setAttribute("target", "_self")

element
<  <a id="mylink" href="https://ok.ru" target="_self">Новый
    текст ссылки</a>
```

# Properties of html elements

К многим атрибутам можно обращаться не через `element.getAttribute("propertyname")`, а напрямую через свойство `element.propertyname`

Однако они не всегда эквивалентны. Например:

```
<input type="text" value="myvalue">
```

```
> let element = document.querySelector("input")
```

пользователь меняет  
значение в поле ввода

```
myvalue
```



```
user changed value
```

```
> element.getAttribute("value")
```

```
< 'myvalue'
```

```
> element.value
```

```
< 'myvalue'
```

```
> element.getAttribute("value")
```

```
< 'myvalue'
```

```
> element.value
```

```
< 'user changed value'
```

Атрибут value указывает на  
значение указанного в HTML  
атрибута value

Свойство value всегда  
указывает на текущий текст в  
поле ввода

# Styles of HTML elements

My text

Стили элемента задаются с помощью свойства **style**

```
<div id="mydiv">My text</div>
```

Название css свойств в js заданы в camelCase:

color	-> element.style.color
font-weight	-> element.style.fontWeight
font-size	-> element.style.fontSize
text-decoration	-> element.style.textDecoration

```
> let element = document.querySelector("#mydiv")
> element.style.color = "blue"
< 'blue'
> element.style.fontWeight = "bold"
< 'bold'
> element.style.fontSize = "36px"
< '36px'
> element.style.textDecoration = "underline"
< 'underline'
```

Данные стили (заданные через свойство **style**)  
задаются как inline-стили элемента



```
> element
<  <div id="mydiv" style="color: blue; font-weight: bold; f
ont-size: 36px; text-decoration: underline;"> My text
</div>
```

# Creating and deleting HTML elements

Создать новый HTML элемент можно с помощью метода `document.createElement("названиеетега")`

Чтобы добавить созданный элемент на страницу, нужно установить его в качестве ребенка другому элементу с помощью метода `parent.appendChild(element)`

Чтобы удалить дочерний элемент используется метод `parent.removeChild(element)`

Создание нового элемента

```
> let newElement = document.createElement("input")
newElement.type = "text"
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1 id="header">Page header</h1>
    <div>
      <p class="paragraph">Page text</p>
      <button>Click me</button>
    </div>
  </body>
</html>
```

Добавление элемента на страницу

```
document.body.appendChild(newElement)
<input type="text">
```

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <h1 id="header">Page header</h1>
    <div>
      <p class="paragraph">Page text</p>
      <button>Click me</button>
    </div>
    ... <input type="text"> == $0
    </body>
</html>
```

Удаление элемента со страницы

```
> document.body.removeChild(newElement)
<input type="text">
```

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <h1 id="header">Page header</h1>
    <div>
      <p class="paragraph">Page text</p>
      <button>Click me</button>
    </div>
  </body>
</html>
```

# Processing HTML collections

Для обработки HTML коллекций можно использовать обычный цикл for

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
      <li>Item 4</li>
      <li>Item 5</li>
      <li>Item 6</li>
      <li>Item 7</li>
      <li>Item 8</li>
      <li>Item 9</li>
      <li>Item 10</li>
    </ul>
  </body>
</html>
```

```
> let htmlElements = document.querySelectorAll("li")
  for(let i=0; i<htmlElements.length; i++) {
    htmlElements[i].style.color = "red"
  }
<- 'red'
```

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Item 8
- Item 9
- Item 10



(перекрашивание всех элементов li на странице в красный цвет)

(Также допустимо использовать цикл for of)

# Working with classes

Классы элемента можно получить через метод `getAttribute("class")` или свойство `className`.

```
<div class="mydiv block main">My text</div>
```

```
> let element = document.querySelector("div")
    element.getAttribute("class")
<- 'mydiv block main'
> element.className
<- 'mydiv block main'
```

С помощью `setAttribute("class", "...")` и `className` можно также менять классы элемента

```
element.setAttribute("class", "mydiv block main newclass")
<- undefined
> element.className = "mydiv block main newclass"
<- 'mydiv block main newclass'
```

Удобнее работать через список классов, находящийся в свойстве `classList`

```
> element.classList
<- DOMTokenList(3) [ 'mydiv', 'block', 'main', value:
  'mydiv block main' ]
```

Добавление нового класса →  
Удаление класса →  
Проверка наличия класса →

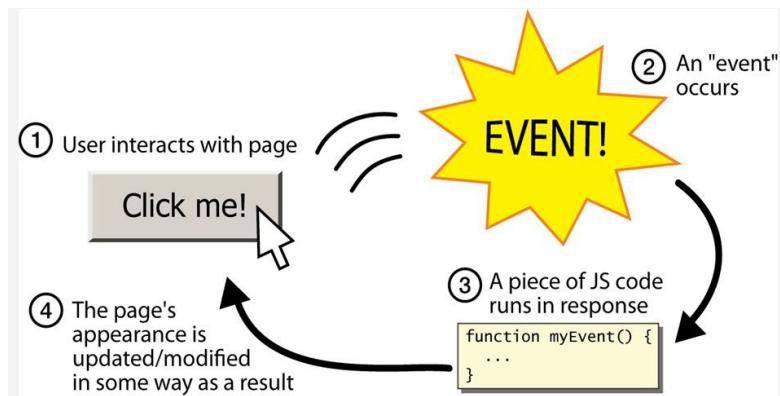
```
> element.classList.add("newclass")
<- undefined
> element.classList.remove("newclass")
<- undefined
> element.classList.contains("block")
<- true
```

# Browser Events

Событие – это сигнал от браузера о том, что что-то произошло. Все DOM-узлы подают такие сигналы.

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.

С помощью обработчиков JavaScript-код может реагировать на действия пользователя.



Распространённые события:

- События мыши:  
`click` (клик на элементе левой кнопкой мыши)  
`contextmenu` (клик на элементе правой кнопкой мыши)  
`mousemove` (изменение положения курсора мыши)
- События клавиатуры:  
`keydown` (нажатие клавиши на клавиатуре)  
`keyup` (отпускание клавиши на клавиатуре)
- События на элементах управления  
`submit` (отправка формы)  
`focus` (фокусировка на элементе ввода)

# Setting the event handler

Название свойства обработчика события строится по принципу “on” + <название события>

Установить обработчик события можно через атрибут прямо в html

```
<div onclick="myHandler()">My text</div>
```

Главное, чтобы при совершении события, функция была определена в скрипте

```
> function myHandler() {  
    console.log("Element clicked!")  
}
```

Установить обработчик события можно через JS свойство полученного элемента

```
<div>My text</div>
```

```
> let element = document.querySelector("div")  
  
element.onclick = function() {  
    console.log("Element clicked!")  
}
```

My text



Element clicked!

# Setting the event handler

Более предпочтительно навешивать обработчики с помощью методов

```
element.addEventListener("eventname", handler)  
element.removeEventListener("eventname", handler)
```

```
<div>My text</div>
```

Можно навесить несколько обработчиков на событие



```
Click handler1  
Click handler2
```



```
Click handler1
```

Установка обработчиков

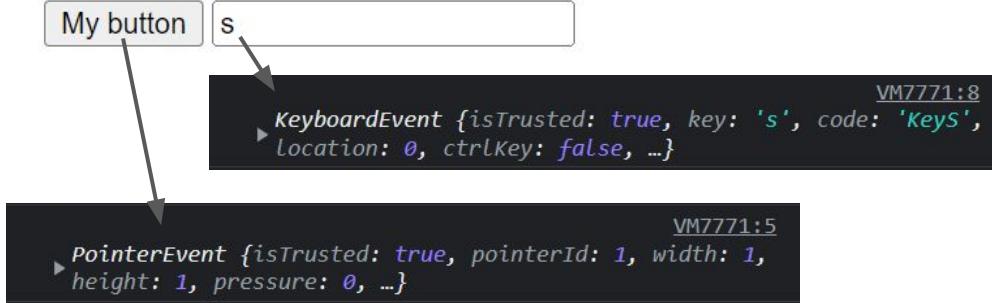
```
> let element = document.querySelector("div")  
  
function handler1() {  
  console.log("Click handler1")  
}  
function handler2() {  
  console.log("Click handler2")  
}  
  
> element.addEventListener("click", handler1)  
> element.addEventListener("click", handler2)
```

Удаление обработчиков

```
> element.removeEventListener("click", handler2)  
< undefined
```

# Event object

В качестве входного аргумента в функцию-обработчик передается объект события, содержащий много полезной информации о данном событии.



Получение координат клика относительно документа (без учёта прокрутки, с учетом прокрутки) и относительно элемента.

```
function buttonHandler(e) {
  console.log(e.pageX + " " + e.pageY)
  console.log(e.clientX + " " + e.clientY)
  console.log(e.offsetX + " " + e.offsetY)
}
```

50	21
50	21
40	11

```
> let button = document.querySelector("button")
let input = document.querySelector("input")

function buttonHandler(e) {
  console.log(e)
}
function inputHandler(e) {
  console.log(e)
}

button.addEventListener("click", buttonHandler)
input.addEventListener("keypress", inputHandler)
< undefined
```

(объект, с которым связано событие, может быть получен как **e.target**)

Получение нажатой клавиши и её кода.

```
function inputHandler(e) {
  console.log(e.key)
  console.log(e.keyCode)
}
```

s
115

# Canceling the default action

В объекте события есть метод `preventDefault`, с помощью которого можно отменить дефолтную реакцию браузера на событие.

Предотвращение отправки формы при клике на кнопку отправки

```
> let form = document.querySelector("form")  
  
    function formSubmitHandler(e) {  
        e.preventDefault()  
    }  
  
    form.addEventListener("submit", formSubmitHandler)  
< undefined
```

Предотвращение появления контекстного меню при правом щелчке мыши по элементу

```
> let body = document.body  
  
    function rightClickHandler(e) {  
        e.preventDefault()  
    }  
  
    body.addEventListener("contextmenu", rightClickHandler)  
< undefined
```

Предотвращение выделения текста в блоке

```
> let div = document.querySelector("div")  
  
    function selectHandler(e) {  
        e.preventDefault()  
    }  
  
    div.addEventListener("selectstart", selectHandler)  
< undefined
```

# Bubbling event

**Всплытие события:** когда на элементе происходит событие, обработчики сначала срабатывают на нём, потом на его родителе, затем выше и так далее, вверх по цепочке предков.

```
<style>
section {
  width: 200px;
  height: 150px;
  border: 1px solid red;
  margin: 10px;
}
div {
  width: 150px;
  height: 100px;
  border: 1px solid green;
  margin: 10px;
}
button {
  width: 100px;
  height: 50px;
  margin: 10px;
}
</style>
```



```
<section>
  <div>
    <button>My button</button>
  </div>
</section>
```

```
> let section = document.querySelector("section")
let div = document.querySelector("div")
let button = document.querySelector("button")

function sectionHandler() {
  console.log("Section click")
}
function divHandler() {
  console.log("Div click")
}
function buttonHandler() {
  console.log("Button click")
}

section.addEventListener("click", sectionHandler)
div.addEventListener("click", divHandler)
button.addEventListener("click", buttonHandler)
< undefined
```

Button click	VM9884:12
Div click	VM9884:9
Section click	VM9884:6

(событие срабатывает сначала на внутреннем элементе, а потом на родительских)

# Cancelling Event Bubbling

С помощью метода объекта события **stopPropagation** можно остановить всплытие события к родительским элементам.

```
<style>
section {
    width: 200px;
    height: 150px;
    border: 1px solid red;
    margin: 10px;
}
div {
    width: 150px;
    height: 100px;
    border: 1px solid green;
    margin: 10px;
}
button {
    width: 100px;
    height: 50px;
    margin: 10px;
}
</style>
```



```
<section>
  <div>
    <button>My button</button>
  </div>
</section>
```

остановка  
всплытия

```
> let section = document.querySelector("section")
let div = document.querySelector("div")
let button = document.querySelector("button")

function sectionHandler() {
  console.log("Section click")
}

function divHandler() {
  console.log("Div click")
}

function buttonHandler(e) {
  e.stopPropagation()
  console.log("Button click")
}

section.addEventListener("click", sectionHandler)
div.addEventListener("click", divHandler)
button.addEventListener("click", buttonHandler)
< undefined
```

Button click

VM9952:13

lodash

# lodash

<https://lodash.com/>

**Lodash** - это современная библиотека утилит JavaScript, обеспечивающая модульность, производительность и дополнительные возможности.

Проще всего подключить библиотеку через CDN, добавив в HEAD соответствующий скрипт:

```
<script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
```

Все методы lodash доступны через специальный объект “\_” (нижнее подчёркивание)

Выводим в консоль версию библиотеки

```
<html>
  <head>
    <title>Page title</title>
    <script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
  </head>
  <body>

    <script>
      console.log(_.VERSION)
    </script>

  </body>
</html>
```

4.17.21



# lodash

<https://lodash.com/docs/4.17.15>

В библиотеке содержится много полезных методов.

Все методы делятся на 13 секций.

Покопавшись в документации, вы можете найти интересные методы для себя

```
> _.camelCase('font-size')
< 'fontSize'

> _.random(3, 10)
< 5

> _.mean([1, 2, 3, 4])
< 2.5
```

← Перевод строки в camelCase

← Генерация случайного числа в  
определённых пределах

← Получение среднего от чисел в массиве

⊕ Array

⊕ Collection

⊕ Date

⊕ Function

⊕ Lang

⊕ Math

⊕ Number

⊕ Object

⊕ Seq

⊕ String

⊕ Util

⊕ Properties

⊕ Methods

# cloneDeep and merge

Метод **cloneDeep** позволяет сделать глубокую копию объекта.

```
> let obj1 = {  
    info: {  
        name: "Mike",  
        age: 25  
    },  
    sayHello: function() {  
        console.log("Hello")  
    }  
}  
  
let obj2 = _.cloneDeep(obj1) ← Создание глубокой копии объекта
```

```
obj2.info.age = 100 ← Изменение одного объекта не скажется на другом  
← 100  
  
obj1.info.age ←  
← 25
```

Метод **merge** позволяет сделать глубокое копирование свойств одного объекта в другой.

```
> let obj1 = {  
    users: [{  
        name: "Mike"  
    }, {  
        name: "Ben"  
    }]  
}  
let obj2 = {  
    users: [{  
        age: 29  
    }, {  
        age: 33  
    }]  
}  
  
_.merge(obj1, obj2) ← Глубокое копирование свойств второго объекта в первый  
  
obj1  
← ▼{users: Array(2)} ⓘ  
  ▼users: Array(2)  
    ►0: {name: 'Mike', age: 29}  
    ►1: {name: 'Ben', age: 33}  
      length: 2  
      ► [[Prototype]]: Array(0)  
      ► [[Prototype]]: Object
```

# pick and pickBy

Методы **pick** и **pickBy** позволяют создать новый объект, включив туда нужные свойства исходного объекта.

**pick** включает в новый объект свойства, указанные в списке, переданным вторым аргументом

```
> let obj1 = {  
    a: 100,  
    b: 200,  
    c: 300,  
    d: 400,  
    e: 500  
}  
  
let obj2 = _.pick(obj1, ['a', 'b', 'c'])  
  
obj2  
< ▶ {a: 100, b: 200, c: 300}
```

**pickBy** вызывает переданную вторым аргументом функцию для каждого свойства. Те свойства, для которых она вернула true, будут включены в новый объект

```
> let obj1 = {  
    a: 100,  
    b: 200,  
    c: 300,  
    d: 400,  
    e: 500  
}  
  
let obj2 = _.pickBy(obj1, (val, prop) => {  
    return prop != 'c'  
})  
  
obj2  
< ▶ {a: 100, b: 200, d: 400, e: 500}
```

(Похожий эффект имеют методы **omit** и **omitBy**. Только в них указывается, какие свойства НЕ включать в новый объект.)

# debounce

**Debounce** - это метод, который гарантирует, что функция в вашем веб-приложении не вызывается слишком часто, что, в свою очередь, оптимизирует производительность вашего приложения.

Например, если в вашем приложении есть поле поиска, где результаты обновляются при каждом новом введенном символе, то это может привести к большому количеству лишних вызовов арі. С помощью **debounce** можно сделать так, что при быстром вводе пользователем текста в поле поиска, обращение к арі выполнится лишь единожды.

```
<input type="text">
```

```
> let input = document.querySelector("input")

function apiSearch() {
    console.log("call api")
}

input.addEventListener("input", _.debounce(apiSearch, 1000))
< undefined
```

Событие input будет вызываться при каждом введенном символе, но обращение к арі будет происходить только один раз, когда пройдёт 1000 мс с последнего введенного символа

moveTo

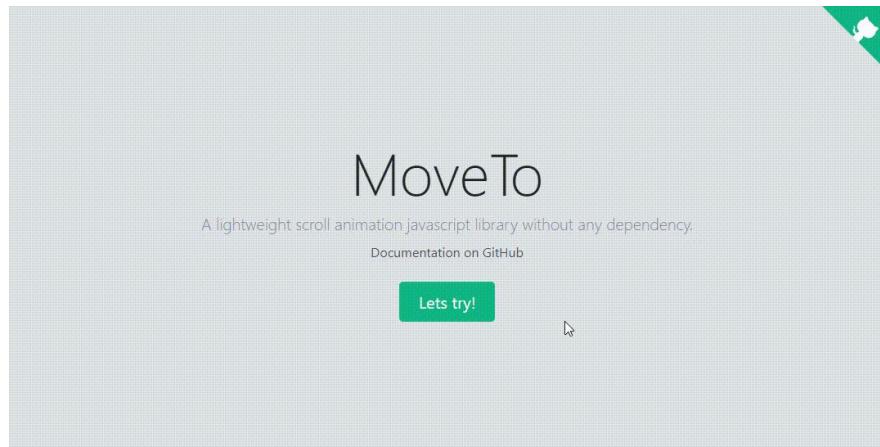
# MoveTo

<https://github.com/hsnaydd/moveTo>

**MoveTo** - это небольшая библиотека, написанная на языке JavaScript, используя функции которой, можно очень быстро и просто создавать эффект скролла.

Подключить можно также через CDN, добавив в HEAD:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/moveTo/1.8.2/moveTo.min.js"></script>
```



# MoveTo

Option	Default	Description
tolerance	0	The tolerance of the target to be scrolled, can be negative or positive
duration	800	Duration of scrolling, in milliseconds
easing	easeOutQuart	Ease function name
container	window	The container been computed and scrolled
callback	noop	The function to be run after scrolling complete. Target passes as the first argument

```
body, html {  
    height: 100%;  
}  
#top {  
    height: 100%;  
    background: #F49095;  
}  
#bottom {  
    height: 100%;  
    background: #9386F4;  
}
```

```
<div id="top">  
    <h1>Начало страницы</h1>  
    <button onclick="moveToBottom()>В конец</button>  
</div>  
<div id="bottom">  
    <h1>Окончание страницы</h1>  
    <button onclick="moveToTop()>В начало</button>  
</div>
```

## Начало страницы

В конец

```
const moveTo = new MoveTo({  
    tolerance: 0,  
    duration: 800,  
    easing: 'easeOutQuart',  
    container: window  
});  
const bottomTarget = document.getElementById('bottom');  
const topTarget = document.getElementById('top');  
  
function moveToBottom() {  
    moveTo.move(bottomTarget);  
}  
function moveToTop() {  
    moveTo.move(topTarget);  
}
```

# Tippy.js

# Tippy.js

<https://atomiks.github.io/tippyjs/>

Tippy.js - это библиотека для всплывающих подсказок, всплывающих окон, выпадающих окон и меню в Интернете на базе Popper.

Подключение:

```
<script src="https://unpkg.com/@popperjs/core@2"></script>  
<script src="https://unpkg.com/tippy.js@6"></script>
```



```
body, html {  
    height: 100%;  
}  
.page {  
    height: 100%;  
    background: #F49095;  
    padding-left: 10px;  
}  
.buttons {  
    display: flex;  
    column-gap: 15px;  
}
```

```
<div class="page">  
    <h1>Страница</h1>  
    <div class="buttons">  
        <button id="btn1">Кнопка 1</button>  
        <button id="btn2">Кнопка 2</button>  
        <button id="btn3">Кнопка 3</button>  
    </div>  
</div>
```

```
tippy('#btn1', {  
    content: 'Tip 1',  
});  
tippy('#btn2', {  
    content: 'Tip 2',  
});  
tippy('#btn3', {  
    content: 'Tip 3',  
    duration: 1000,  
    arrow: false,  
    delay: [1000, 200],  
});
```

# Overlay Scrollbars

# Overlay Scrollbars

<https://kingsora.github.io/OverlayScrollbars/#!overview>

**Overlay Scrollbars** - библиотека для кастомизации полос прокрутки элементов.

## Подключение:

```
<link rel="stylesheet"  
      href="https://cdnjs.cloudflare.com/ajax/libs/OverlayScrollbars/1.13.3/css/OverlayScrollbars.min.css" />  
  
<script src="https://cdnjs.cloudflare.com/ajax/libs/OverlayScrollbars/1.13.3/js/OverlayScrollbars.min.js"></script>
```

## Violet:

Fixed handle size, the track is visible. The scrollbars look like a slider.

Lorem ipsum dolor sit amet, consetetur s  
diam nonumy eirmod tempor invidunt ut  
magna aliquyam erat, sed diam voluptua  
accusam et justo duo dolores et ea rebur  
gubergren, no sea takimata sanctus est L  
sit amet. Lorem ipsum dolor sit amet, cor  
elitr, sed diam nonumy eirmod tempor in  
dolore magna aliquyam erat, sed diam v  
et accusam et justo duo dolores et ea re  
gubergren, no sea takimata sanctus est L  
sit amet. Lorem ipsum dolor sit amet, cor

### Thin:

The scrollbars are only visible if you hover over the host-element. Flexible handle size, the track is visible. The handles are very thin.

## Minima

The scrollbars are only visible during scrolling. (due to the Flexible handle size, the track is invisible. The handles goe

# Overlay Scrollbars

```
body, html {  
    height: 100%;  
}  
.page {  
    height: 100%;  
    background: #F49095;  
    padding-left: 10px;  
}  
.container {  
    width: 200px;  
    height: 150px;  
    background: #BEA3F4;  
    overflow: hidden;  
}
```

```
OverlayScrollbars(document.querySelectorAll(".container"), {  
    scrollbars: {  
        autoHide: 'scroll'  
    }  
});
```

```
<div class="page">  
    <h1>Страница</h1>  
    <div class="container">  
        Lorem ipsum dolor sit amet,  
        consectetur adipiscing elit...  
    </div>  
</div>
```

## Страница

Loreum 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  
Duis aute irure

Duis aute irure

# Swiper

# Swiper

<https://swiperjs.com/>

Swiper - библиотека для внедрения слайдера (карусели).

Подключение:

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/swiper@8/swiper-bundle.min.css" />
```

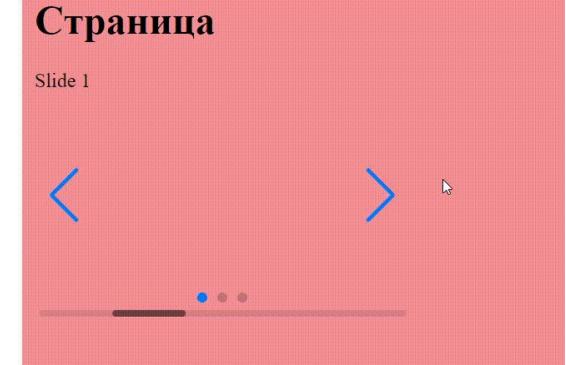
```
<script src="https://cdn.jsdelivr.net/npm/swiper@8/swiper-bundle.min.js"></script>
```

```
body, html {
    height: 100%;
}
.page {
    height: 100%;
    background: #F49095;
    padding-left: 10px;
}
.swiper {
    margin-left: 0;
    width: 300px;
    height: 200px;
}
```

```
<div class="page">
    <h1>Страница</h1>
    <!-- Slider main container -->
    <div class="swiper">
        <!-- Additional required wrapper -->
        <div class="swiper-wrapper">
            <!-- Slides -->
            <div class="swiper-slide">Slide 1</div>
            <div class="swiper-slide">Slide 2</div>
            <div class="swiper-slide">Slide 3</div>
            ...
        </div>
        <!-- If we need pagination -->
        <div class="swiper-pagination"></div>

        <!-- If we need navigation buttons -->
        <div class="swiper-button-prev"></div>
        <div class="swiper-button-next"></div>

        <!-- If we need scrollbar -->
        <div class="swiper-scrollbar"></div>
    </div>
</div>
```



```
const swiper = new Swiper('.swiper', {
    // Optional parameters
    direction: 'horizontal',
    loop: true,

    // If we need pagination
    pagination: {
        el: '.swiper-pagination',
    },

    // Navigation arrows
    navigation: {
        nextEl: '.swiper-button-next',
        prevEl: '.swiper-button-prev',
    },

    // And if we need scrollbar
    scrollbar: {
        el: '.swiper-scrollbar',
    },
});
```

# LazySizes

# Lazysizes

<https://github.com/aFarkas/lazysizes>

**Lazysizes** - библиотека, реализующая “ленивую” загрузку изображений, iframe-ов, скриптов и виджетов на страницу.

“Ленивая” значит, что при загрузке вашей страницы будут загружаться не все элементы, а только те, которые попадают в поле зрения пользователя. Оставшиеся элементы будут подгружаться, когда понадобятся (при прокрутке страницы)

Подключение:

```
<script src="http://afarkas.github.io/lazysizes/lazysizes.min.js"></script>
```

Использование:

адрес картинки плохого качества

адрес оригинальной картинки

класс `lazyload`, который говорит библиотеке, что данную картинку нужно загружать лениво

```

```

iMask.js

# imask.js

<https://imask.js.org/>

imask.js - библиотека для установки маски (шаблона) на поля ввода в формах.

Подключение:

```
<script src="https://unpkg.com/imask"></script>
```

RegExp (Russian postal code) `/^{\{1-6\}}d{\{0,5\}}$/`

[source](#)

Pattern (Phone) `+{\{7\}}(000)000-00-00`

unmasked: [source](#)

Number in range [-10000, 10000]

number: 0 [source](#)

Date 'dd.mm.yyyy' in range [01.01.1990, 01.01.2020]

date: - [source](#)

# imask.js

```
body, html {  
    height: 100%;  
}  
.page {  
    height: 100%;  
    background: #F49095;  
    padding-left: 10px;  
    padding-top: 10px;  
}
```

```
<div class="page">  
    <form>  
        <input type="text" id="phone" placeholder="phone" />  
        <input type="text" id="age" placeholder="age" />  
        <input type="text" id="date" placeholder="date" />  
    </form>  
</div>
```



```
IMask(document.querySelector("#phone"), {  
    mask: '+{7}(000)-000-00-00'  
})  
IMask(document.querySelector("#age"), {  
    mask: Number,  
    min: 0,  
    max: 100  
})  
IMask(document.querySelector("#date"), {  
    mask: Date,  
    lazy: false,  
    autofocus: true,  
    blocks: {  
        d: {mask: IMask.MaskedRange, placeholderChar: 'd', from: 1, to: 31, maxLength: 2},  
        m: {mask: IMask.MaskedRange, placeholderChar: 'm', from: 1, to: 12, maxLength: 2},  
        Y: {mask: IMask.MaskedRange, placeholderChar: 'y', from: 1900, to: 2999, maxLength: 4}  
    }  
})
```

anime.js

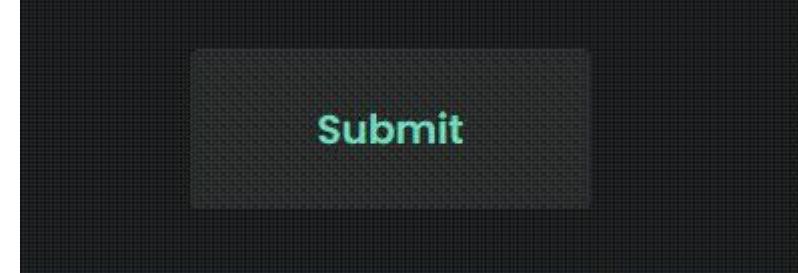
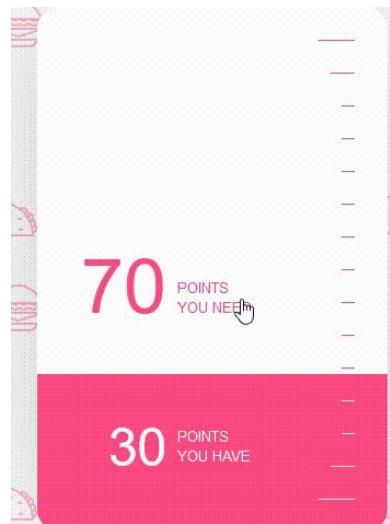
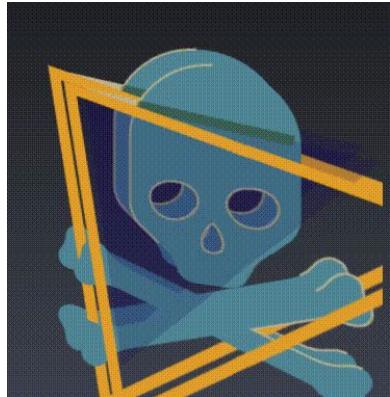
# animejs

<https://animejs.com/>

Anime.js - это легкая библиотека анимации JavaScript с простым и мощным API.

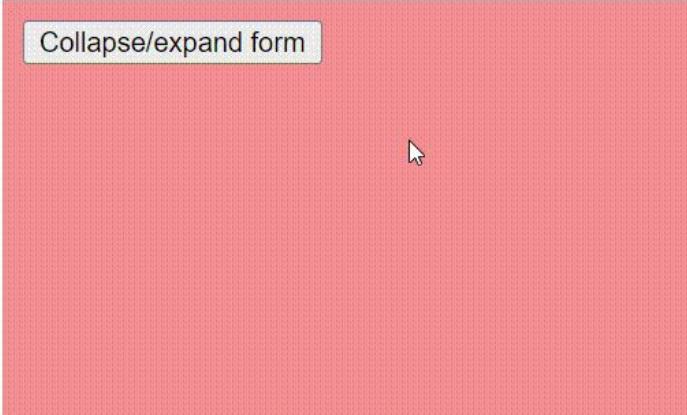
Подключение:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/animejs/3.2.1/anime.min.js"></script>
```



# animejs

```
body, html {  
    height: 100%;  
}  
.page {  
    height: 100%;  
    background: #F49095;  
    padding-left: 10px;  
    padding-top: 10px;  
}  
.container {  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
    gap: 10px;  
    width: 300px;  
    margin-top: 10px;  
    padding-top: 20px;  
    border: 2px solid black;  
}
```



```
<div class="page">  
    <button onclick="toggleHandler()">Collapse/expand form</button>  
    <form class="container" style="opacity: 0; height: 0; overflow: hidden; ">  
        <input type="text" placeholder="email">  
        <input type="text" placeholder="password">  
        <input type="submit" value="send">  
    </form>  
</div>
```

```
function toggleHandler() {  
    let form = document.querySelector("form")  
    if(form.style.opacity == 0) {  
        anime({  
            targets: form,  
            opacity: 1,  
            height: form.scrollHeight,  
            duration: 500,  
            easing: 'easeInOutQuad'  
        })  
    } else {  
        anime({  
            targets: form,  
            opacity: 0,  
            height: 0,  
            duration: 500,  
            easing: 'easeInOutQuad'  
        })  
    }  
}
```