Теоретический материал по HTML, CSS, JAVASCRIPT

[HTML 5](#_Toc101217300)

[Основная структура Html страницы 5](#_Toc101217301)

[Теги заголовка <head></head> 5](#_Toc101217302)

[Комментарий HTML 6](#_Toc101217303)

[Установка фавиконки 7](#_Toc101217304)

[Спецсимволы 7](#_Toc101217305)

[1. Полезные знаки и символы 7](#_Toc101217306)

[2. Знаки пунктуации 12](#_Toc101217307)

[3. Стрелки 17](#_Toc101217308)

[4. Карточные масти 20](#_Toc101217309)

[5. Деньги 20](#_Toc101217310)

[6. Знаки зодиака 21](#_Toc101217311)

[Добавление и изминение линий 22](#_Toc101217312)

[Изменение шрифта 23](#_Toc101217313)

[Атрибуты тега <body></body> 24](#_Toc101217314)

[Добавление списков 25](#_Toc101217315)

[Ссылки 27](#_Toc101217316)

[Изображения 27](#_Toc101217317)

[Таблицы 29](#_Toc101217318)

[Атрибуты для тега <table>: 29](#_Toc101217319)

[Атрибуты тега <td>: 30](#_Toc101217320)

[Разметка макета с помощью таблицы 31](#_Toc101217321)

[Карта изображений 32](#_Toc101217322)

[Форма 34](#_Toc101217323)

[Связь с <input> - Тег <label> 37](#_Toc101217324)

[Выпадающий список - <select></select> 38](#_Toc101217325)

[Атрибут <textarea></textarea> 40](#_Toc101217326)

[Атрибуты тега <input> </input> 40](#_Toc101217327)

[Атрибуты тега <form> </form> 47](#_Toc101217328)

[Группировка данных в форме 49](#_Toc101217329)

[Подключение отдельного html файла на страницу 50](#_Toc101217330)

[CSS 52](#_Toc101217331)

[Комментарий CSS 52](#_Toc101217332)

[Селекторы 52](#_Toc101217333)

[Все CSS Псевдо-классы 55](#_Toc101217334)

[Все CSS Псевдо-элементы 58](#_Toc101217335)

[Подключение CSS 60](#_Toc101217336)

[Подключение сторонних шрифтов 61](#_Toc101217337)

[Основные свойства CSS 62](#_Toc101217338)

[B 62](#_Toc101217339)

[C 63](#_Toc101217340)

[D 63](#_Toc101217341)

[E 63](#_Toc101217342)

[F 63](#_Toc101217343)

[H 64](#_Toc101217344)

[L 64](#_Toc101217345)

[M 64](#_Toc101217346)

[O 65](#_Toc101217347)

[P 65](#_Toc101217348)

[Q 65](#_Toc101217349)

[R 65](#_Toc101217350)

[T 65](#_Toc101217351)

[U 66](#_Toc101217352)

[V 66](#_Toc101217353)

[W 66](#_Toc101217354)

[Z 66](#_Toc101217355)

[Таблица жирности шрифтов 69](#_Toc101217356)

[Свойства фона 86](#_Toc101217357)

[Фильтры для изображений 94](#_Toc101217358)

[Способы расположения элементов на web – странице 103](#_Toc101217359)

[Подключение иконок типа svg в меню и не только 111](#_Toc101217360)

[Заливка и обводка SVG иконок 112](#_Toc101217361)

[Создание меню/списков 114](#_Toc101217362)

[Управление скроллбаром 120](#_Toc101217363)

[Работа с Input через CSS 125](#_Toc101217364)

[Изменение цвета текста placeholder 125](#_Toc101217365)

[Удаление крестика (очистка содержимого) из поля input type=”search” 126](#_Toc101217366)

[Появление блока при наведении 127](#_Toc101217367)

[Верстка сайта 129](#_Toc101217368)

[I. Указание свойства адаптивности с помощью параметра meta 129](#_Toc101217369)

[II. Использование CSS-свойства @media вместе с правилами адаптивной верстки 130](#_Toc101217370)

[III. Использование разных единиц измерений необходимых для адаптивности макета 135](#_Toc101217371)

[JAVASCRIPT (JS) 138](#_Toc101217372)

[Комментарии 138](#_Toc101217373)

[Подключение javascript 138](#_Toc101217374)

[Атрибуты подключения JS 139](#_Toc101217375)

[Основные методы JS 140](#_Toc101217376)

[1) alert() 140](#_Toc101217377)

[2) confirm() 140](#_Toc101217378)

[3) promt() 141](#_Toc101217379)

[4) write() 142](#_Toc101217380)

[Объявление переменных 145](#_Toc101217381)

[Оператор присванивания: 147](#_Toc101217382)

[Основные способы занесения данных в переменные JS 149](#_Toc101217383)

[Операторы сравнения 150](#_Toc101217384)

[Логические операторы 151](#_Toc101217385)

[Таблица истинности для операции логического и 152](#_Toc101217386)

[Таблица истинности для операции логического или 152](#_Toc101217387)

[Слияние строк 152](#_Toc101217388)

[Условный оператор if 153](#_Toc101217389)

[Проверка состояния/запуска объектов тега <form></form> 154](#_Toc101217390)

[<button></button> 154](#_Toc101217391)

[Checkbox и radiobutton 155](#_Toc101217392)

[Добавление CSS свойства к объекту через JS 158](#_Toc101217393)

[Более сложный способ вывода информации 159](#_Toc101217394)

[Еще один пример вызова функции обработчика события 161](#_Toc101217395)

[Обработчик событий 165](#_Toc101217396)

[Способы вызова функций 165](#_Toc101217397)

[Обращение к фунции 167](#_Toc101217398)

[Коды клавиш для использования в функциях 169](#_Toc101217399)

[Список некоторых методов 176](#_Toc101217400)

[Массивы 176](#_Toc101217401)

[Строки 178](#_Toc101217402)

[Математические (Math) 180](#_Toc101217403)

[Рандомные числа с использованием метода Math 181](#_Toc101217404)

[Циклы 183](#_Toc101217405)

[While 183](#_Toc101217406)

[For 183](#_Toc101217407)

[Цикл for для массивов 184](#_Toc101217408)

[Инструкция break 185](#_Toc101217409)

[Использование массивов 185](#_Toc101217410)

[Объявление и вывод массива: 185](#_Toc101217411)

[Ключи элементов массива 185](#_Toc101217412)

[Заполнение массива 185](#_Toc101217413)

HTML

Основная структура Html страницы

<!DOCTYPE HTML> <!--Позволяет браузеру, в котором открывается данная страница, определить, что для верстки был использован стандарт HTML 5-->

<html lang="rus" >

<head>

<title></title>

</head>

<body>

</body>

</html>

Теги заголовка <head></head>

1. **<title></title>**
2. [**<base>**](#base)
3. **<meta>** - определяет метатеги, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных.

Разрешается использовать более чем один метатег, все они размещаются в контейнере <head>. Как правило, атрибуты любого метатега сводятся к парам «имя=значение», которые определяются ключевыми словами **content**, **name** или **http-equiv**.

**Синтаксис:**

**<head>**

**<meta** content="..."**>**

**</head>**

Атрибуты:

**charset** - Задает кодировку документа.

**content** - Устанавливает значение атрибута, заданного с помощью name или http-equiv.

**http-equiv** - Предназначен для конвертирования метатега в заголовок HTTP.

* **content-type** - Тип кодировки документа.
* **expires** - Устанавливает дату и время, после которой информация в документе будет считаться устаревшей.
* **cache-control** - Способ кэширования документа.
* **refresh** - Загрузить другой документ в текущее окно браузера.

**name** - Имя метатега, также косвенно устанавливает его предназначение.

* **author** - Имя автора документа.
* **description** - Описание текущего документа.
* **keywords** - Список ключевых слов, встречающихся на странице.

Пример:

<meta name=description content=”Обучение, управление проектами, сертификация”>

<meta name=keywords content=”PMP, проект, управление проектами, сертификация”>

1. [**<style></style>**](#style)
2. [**<link>**](#link)
3. [**<script></script>**](#_Подключение_javascript)

Комментарий HTML

Для комментирования кода Html в теге **<body></body>** используется <! --…-->

Установка фавиконки

При загрузке сайта, в адресной строке браузера можно иногда увидеть небольшую картинку, которая связана с тематикой сайта. Тот же рисунок отображается рядом со ссылкой при добавлении ее в закладки браузера.

Размер изображения должен быть 16\*16.

**<head>**

**<link** rel="shortcut icon" href="/images/favicon.ico" type="image/x-icon"**>**

**</head>**

В type указывается **image/png, image/x-icon** или **image/gif** в зависимости от формата изображения.

Спецсимволы

### 1. Полезные знаки и символы

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид** | **HTML-код** | **CSS-код** | **Описание** |
| **🖂** | &#128386; | \1F582 | Обратная сторона конверта |
| **☃** | &#9731; | \2603 | Снеговик |
| **☦** | &#9766; | \2626 | Православный крест |
| **⚓** | &#9875; | \2693 | Якорь |
| **✆** | &#9990; | \2706 | Знак телефона |
| **☎** | &#9742; | \260E | Телефон |
| **☕** | &#9749; | \2615 | Горячие напитки |
| **✎** | &#9998; | \270E | Карандаш, направленный вправо-вниз |
| **✏** | &#9999; | \270F | Карандаш |
| **✐** | &#10000; | \2710 | Карандаш, направленный вправо-вверх |
| **✑** | &#10001; | \2711 | Незакрашенное острие пера |
| **✒** | &#10002; | \2712 | Закрашенное острие пера |
| **⚜** | &#9884; | \269C | Геральдическая лилия |
| **⛑** | &#9937; | \26D1 | Шлем с белым крестом |
| **⚝** | &#9885; | \269D | Начерченная белая звезда |
| **❄** | &#10052; | \2744 | Снежинка |
| **❤** | &#10084; | \2764 | Закрашенное жирное сердце |
| **❅** | &#10053; | \2745 | Зажатая трилистниками снежинка |
| **❆** | &#10054; | \2746 | Жирная остроугольная снежинка |
| **★** | &#9733; | \2605 | Закрашенная звезда |
| **☆** | &#9734; | \2606 | Незакрашенная звезда |
| **✪** | &#10026; | \272A | Незакрашенная звезда в закрашенном круге |
| **✫** | &#10027; | \272B | Закрашенная звезда с незакрашенным кругом внутри |
| **✯** | &#10031; | \272F | Вращающаяся звезда |
| **❉** | &#10057; | \2749 | Звёздочка с шарообразными окончаниями |
| **❋** | &#10059; | \274B | Жирная восьмиконечная каплеобразная звёздочка-пропеллер |
| **✲** | &#10034; | \2732 | Звёздочка с незакрашенным центром |
| **☀** | &#9728; | \2600 | Закрашенное солнце с лучами |
| **☁** | &#9729; | \2601 | Облака |
| **☂** | &#9730; | \2602 | Зонтик |
| **☑** | &#9745; | \2611 | Галочка в квадрате |
| **☒** | &#9746; | \2612 | Крестик в квадрате |
| **☹** | &#9785; | \2639 | Нахмуренный смайлик |
| **☺** | &#9786; | \263A | Улыбающийся смайлик |
| **☻** | &#9787; | \263B | Закрашенный улыбающийся смайлик |
| **☭** | &#9773; | \262D | Серп и молот |
| **⚑** | &#9873; | \2691 | Закрашенный флаг |
| **⚐** | &#9872; | \2690 | Незакрашенный флаг |
| **☰** | &#9776; | \2630 | Триграмма |
| **✿** | &#10047; | \273f | Закрашенный цветок |
| **❀** | &#10048; | \2740 | Незакрашенный цветок |
| **✾** | &#10046; | \273E | Цветок с шестью лепестками |
| **❁** | &#10049; | \2741 | Закрашенный обведённый цветок |
| **❂** | &#10050; | \2742 | Цветок из точек |
| **✉** | &#9993; | \2709 | Конверт |
| **❦** | &#10086; | \2766 | Сердце в виде цветка |
| **❶** | &#10102; | \2776 | Номер 1 |
| **❷** | &#10103; | \2777 | Номер 2 |
| **❸** | &#10104; | \2778 | Номер 3 |
| **❹** | &#10105; | \2779 | Номер 4 |
| **❺** | &#10106; | \277A | Номер 5 |
| **❻** | &#10107; | \277B | Номер 6 |
| **❼** | &#10108; | \277C | Номер 7 |
| **❽** | &#10109; | \277D | Номер 8 |
| **➒** | &#10130; | \2792 | Номер 9 |
| **➓** | &#10131; | \2793 | Номер 10 |
| **✖** | &#10006; | \2716 | Жирный знак умножения |
| **✘** | &#10008; | \2718 | Жирный крестик |
| **✔** | &#10004; | \2714 | Жирная отметка галочкой |
| **✚** | &#10010; | \271A | Жирный крест |
| **⚛** | &#9883; | \269B | Символ атома |
| **♺** | &#9850; | \267A | Символ переработки |
| **❑** | &#10065; | \2751 | Незакрашенный квадрат с правой нижней тенью |
| **❒** | &#10066; | \2752 | Незакрашенный квадрат с правой верхней тенью |
| **◈** | &#9672; | \25C8 | Алмаз в оправе |
| **◐** | &#9680; | \25D0 | Круг с левой закрашенной половиной |
| **◑** | &#9681; | \25D1 | Круг с закрашенной правой половиной |
| **⁂** | &#8258; | \2042 | Три звездочки |

### 2. Знаки пунктуации

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид** | **HTML-код** | **CSS-код** | **Описание** |
|  | &nbsp; | \00A0 | Неразрывный пробел |
| **­** | &shy; | \00AD | Место возможного переноса |
| **<** | &lt; | \003C | Знак "меньше чем" (начало тега) |
| **>** | &gt; | \003E | Знак "больше чем" (конец тега) |
| **«** | &laquo; | \00AB | Левая двойная угловая скобка |
| **»** | &raquo; | \00BB | Правая двойная угловая скобка |
| **‹** | &lsaquo; | \2039 | Левая угловая одиночная кавычка |
| **›** | &rsaquo; | \203A | Правая угловая одиночная кавычка |
| **"** | &quot; | \0022 | Двойная кавычка |
| **′** | &prime; | \2032 | Одиночный штрих |
| **″** | &Prime; | \2033 | Двойной штрих |
| **‘** | &lsquo; | \2018 | Левая одиночная кавычка |
| **’** | &rsquo; | \2019 | Правая одиночная кавычка |
| **‚** | &sbquo; | \201A | Нижняя одиночная кавычка |
| **“** | &ldquo; | \201C | Левая двойная кавычка |
| **”** | &rdquo; | \201D | Правая двойная кавычка |
| **„** | &bdquo; | \201E | Нижняя двойная кавычка |
| **❜** | &#10076; | \275C | Жирная одинарная верхняя запятая |
| **❛** | &#10075; | \275B | Жирная одинарная повёрнутая верхняя запятая |
| **&** | &amp; | \0026 | Амперсанд |
| **'** | &apos; | \0027 | Апостроф (одинарная кавычка) |
| **§** | &sect; | \00A7 | Параграф |
| **©** | &copy; | \00A9 | Знак copyright |
| **¬** | &not; | \00AC | Знак отрицания |
| **®** | &reg; | \00AE | Знак зарегистрированной торговой марки |
| **¯** | &macr; | \00AF | Знак долготы над гласным |
| **°** | &deg; | \00B0 | Градус |
| **±** | &plusmn; | \00B1 | Плюс-минус |
| **¹** | &sup1; | \00B9 | Верхний индекс "1" |
| **²** | &sup2; | \00B2 | Верхний индекс "2" |
| **³** | &sup3; | \00B3 | Верхний индекс "3" |
| **¼** | &frac14; | \00BC | Одна четверть |
| **½** | &frac12; | \00BD | Одна вторая |
| **¾** | &frac34; | \00BE | Три четверти |
| **´** | &acute; | \00B4 | Знак ударения |
| **µ** | &micro; | \00B5 | Микро |
| **¶** | &para; | \00B6 | Знак абзаца |
| **·** | &middot; | \00B7 | Знак умножения |
| **¿** | &iquest; | \00BF | Перевернутый вопросительный знак |
| **ƒ** | &fnof; | \0192 | Знак флорина |
| **™** | &trade; | \2122 | Знак торговой марки |
| **•** | &bull; | \2022 | Маркер списка |
| **…** | &hellip; | \2026 | Многоточие |
| **‾** | &oline; | \203E | Надчеркивание |
| **–** | &ndash; | \2013 | Среднее тире |
| **—** | &mdash; | \2014 | Длинное тире |
| **‰** | &permil; | \2030 | Промилле |
| **}** | &#125; | \007D | Правая фигурная скобка |
| **{** | &#123; | \007B | Левая фигурная скобка |
| **=** | &#61; | \003D | Знак равенства |
| **≠** | &ne; | \2260 | Знак неравенства |
| **≅** | &cong; | \2245 | Конгруэнтность (геометрическое равенство) |
| **≈** | &asymp; | \2248 | Почти равно |
| **≤** | &le; | \2264 | Меньше чем или равно |
| **≥** | &ge; | \2265 | Больше чем или равно |
| **∠** | &ang; | \2220 | Угол |
| **⊥** | &perp; | \22A5 | Перпендикулярно (кнопка вверх) |
| **√** | &radic; | \221A | Квадратный корень |
| **∑** | &sum; | \2211 | N-ичное суммирование |
| **∫** | &int; | \222B | Интеграл |
| **※** | &#8251; | \203B | Знак сноски |
| **÷** | &divide; | \00F7 | Знак деления |
| **∞** | &infin; | \221E | Знак бесконечности |
| **@** | &#64; | \0040 | Символ собака |
| **[** | &#91; | \005B | Левая квадратная скобка |
| **]** | &#93; | \005D | Правая квадратная скобка |

### 3. Стрелки

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид** | **HTML-код** | **CSS-код** | **Описание** |
| **←** | &larr; | \2190 | Стрелка влево |
| **↑** | &uarr; | \2191 | Стрелка вверх |
| **→** | &rarr; | \2192 | Стрелка вправо |
| **↓** | &darr; | \2193 | Стрелка вниз |
| **↔** | &harr; | \2194 | Стрелка влево-вправо |
| **↵** | &crarr; | \21B5 | Стрелка вниз и влево – знак возврата каретки |
| **⇐** | &lArr; | \21D0 | Двойная стрелка налево |
| **⇑** | &uArr; | \21D1 | Двойная стрелка вверх |
| **⇒** | &rArr; | \21D2 | Двойная стрелка направо |
| **⇓** | &dArr; | \21D3 | Двойная стрелка вниз |
| **⇔** | &hArr; | \21D4 | Двойная стрелка влево-вправо |
| **➠** | &#10144; | \27A0 | Летящая стрела |
| **➤** | &#10148; | \27A4 | Наконечник стрелы |
| **➥** | &#10149; | \27A5 | Изогнутая стрела, указывающая вниз и вправо |
| **➦** | &#10150; | \27A6 | Изогнутая стрела, указывающая вверх и вправо |
| **➳** | &#10163; | \27B3 | Стрела направо |
| **↺** | &#8634; | \21BA | Круглая стрелка с наконечником против часовой стрелки |
| **↻** | &#8635; | \21BB | Круглая стрелка с наконечником по часовой стрелке |
| **⇧** | &#8679; | \21E7 | Толстая полая стрелка вверх |
| **↩** | &#8617; | \21A9 | Стрелка налево с крючком |
| **➫** | &#10155; | \27AB | Наклонённая вниз объёмная стрелка |
| **⬇** | &#11015; | \2B07 | Закрашенная стрелка вниз |
| **⬆** | &#11014; | \2B06 | Закрашенная стрелка вверх |

### 4. Карточные масти

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид** | **HTML-код** | **CSS-код** | **Описание** |
| **♠** | &spades; | \2660 | "Пики" |
| **♣** | &clubs; | \2663 | "Трефы" |
| **♥** | &hearts; | \2665 | "Червы" |
| **♦** | &diams; | \2666 | "Бубны" |
| **♡** | &#9825; | \2661 | Контур "Червы" |
| **♢** | &#9826; | \2662 | Контур "Бубны" |
| **♤** | &#9828; | \2664 | Контур "Пики" |
| **♧** | &#9831; | \2667 | Контур "Трефы" |

### 5. Деньги

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид** | **HTML-код** | **CSS-код** | **Описание** |
| **¢** | &cent; | \FFE0 | Цент |
| **£** | &pound; | \FFE1 | Фунт стерлингов |
| **₽** | &#8381; | \20BD | Российский рубль |
| **¥** | &yen; | \00A5 | Йена или юань |
| **€** | &euro; | \20AC | Евро |
| **$** | &#36; | \0024 | Доллар |
| **₴** | &#8372; | \20B4 | Знак гривны |
| **₹** | &#8377; | \20B9 | Индийская рупия |
| **圓** | &#22291; | \5713 | Китайский юань |
| **₸** | &#8376; | \20B8 | Казахстанский тенге |

### 6. Знаки зодиака

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид** | **HTML-код** | **CSS-код** | **Описание** |
| **♈** | &#9800; | \2648 | Овен |
| **♉** | &#9801; | \2649 | Телец |
| **♊** | &#9802; | \264A | Близнецы |
| **♋** | &#9803; | \264B | Рак |
| **♌** | &#9804; | \264C | Лев |
| **♍** | &#9805; | \264D | Дева |
| **♎** | &#9806; | \264E | Весы |
| **♏** | &#9807; | \264F | Скорпион |
| **♐** | &#9808; | \2650 | Стрелец |
| **♑** | &#9809; | \2651 | Козерог |
| **♒** | &#9810; | \2652 | Водолей |
| **♓** | &#9811; | \2653 | Рыбы |

Добавление и изминение линий

**<hr>** - тег для рисования линии

**Атрибуты и значения:**

align="" – выравнивает линию справа или слева.

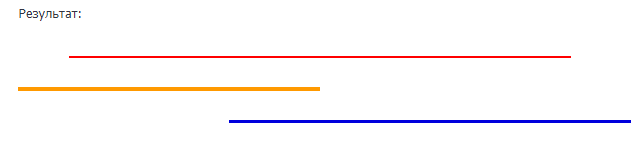
width="50%" – ширина в пикселях или процентах.

size="" – задает толщину горизонтальной линии.

color="#00ff00" – устанавливает цвет линии.

Пример:

<html>  
<head>  
<title>Горизонтальная линия в HTML</title>  
</head>  
<body>  
<hr align="center" width="500" size="2" color="#ff0000" />  
<br />  
<hr align="left" width="300" size="4" color="#ff9900" />  
<br />  
<hr align="right" width="400" size="3" color="#0000dd" />  
</body>  
</html>



Изменение шрифта

**<font></font>** - Закрывающийся тег, необходимый для точечного изменения свойств текста, таких как: гарнитура, размер и цвет шрифта. (**ИСПОЛЬЗОВАНИЕ РЕКОМЕНДУЕТСЯ ЗАМЕНЯТЬ CSS**)

Атрибуты:

* face – **гарнитура шрифта**
* size - **размер шрифта**

|  |  |
| --- | --- |
| Номер шрифта | Размер шрифта в кеглях |
| 1 | 8 |
| 2 | 10 |
| 3 | 12 |
| 4 | 14 |
| 5 | 18 |
| 6 | 24 |
| 7 | 36 |

* color - **цвет шрифта**

Атрибуты тега <body></body>

alink -**Устанавливает цвет активной ссылки**

background - **Задает фоновый рисунок на веб-странице**

bgcolor - **Цвет фона веб-страницы**

bgproperties - **Определяет, прокручивать фон совместно с текстом или нет**

bottommargin - **Отступ от нижнего края окна браузера до контента**

leftmargin - **Отступ по горизонтали от левого края окна браузера до контента**

link - **Цвет ссылок на веб-странице**

rightmargin - **Отступ от правого края окна браузера до контент**

scroll - **Устанавливает, отображать полосы прокрутки или нет**

text - **Цвет текста в документе**

topmargin - **Отступ от верхнего края окна браузера до контента**

vlink - **Цвет посещенных ссылок**

Добавление списков

1. **Упорядоченные списки**

|  |  |
| --- | --- |
| Значение атрибута ***type*** | Тип нумерации пунктов списка |
| ‘1’ | 1, 2, 3, 4, …… |
| ‘i’ | i, ii, iii, iv,…… |
| ‘I’ | I, II, III, IV,.. |
| ‘a’ | a, b, c, d,…. |
| ‘A’ | A, B, C, D,… |

<ol>

<li></li>

…

<li></li>

</ol>

1. **Не упорядоченные списки**

|  |  |
| --- | --- |
| Значение атрибута **type** | Маркер |
| ‘disc’ |  |
| ‘circle’ |  |
| ‘square’ |  |

<ul>

<li></li>

…

<li></li>

</ul>

1. **Списки определений**

<dl>

<dt></dt> **- Используется для обозначения понятия термина**

<dd></dd> - **Используется для определения понятия термина**

</dl>

1. **Вложенные списки**

Пример:

<ul>

<li></li>

<li>

<ul>

<li></li>

…

<li></li>

</li>

</ul>

</ul>

Ссылки

**<base>** - явное задание полного URL – адреса

Пример:

<!DOCTYPE HTML> <html>

<head>

<base href=” www.nit.center”>

</head>

<body>

<a href=”\images\ret.gig”>Ссылка на изображение ret.gif</a>

<a href=”chapter.html”>Переход к следующей главе</a>

</body>

</html>

В данном примере путь к картинке составит: **www.nit.center\images\ret.gig**, а путь к странице**: www.nit.center\ chapter.html**

**----------------------------------------------------------------------------------------------------**

<a href=”**адрес страницы**”**>текст**</a> - **Ссылка на другую страницу**

<a href=”[mailto:**адрес почты**](mailto:адрес%20почты)”>**текст**</a> - **Ссылка на электронную почту**

<a href=”**адрес страницы**” target=”\_blank”**>текст**</a> - **Открытие ссылки в новом окне**

<p id=”**название метки**”>**текст**</p>

<a href=”#**название метки**”>**текст**</a> - **Ссылка на определенную часть страницы**

<p id=”**название метки**”>**текст**</p>

<a href=”**адрес страницы/**#**название метки**”>**текст**</a> - **Ссылка на определенную часть другой страницы**

<a href = **"путь к файлу**" download>**Содержимое элемента**</a> - **Ссылка на скачивание документа**

Изображения

<img> - **Тег для добавления изображения**

Атрибуты:

align - **Определяет как рисунок будет выравниваться по краю и способ обтекания текстом:**

* + bottom - **Выравнивание нижней границы изображения по окружающему тексту**
  + left - **Выравнивает изображение по левому краю окна**
  + middle - **Выравнивание середины изображения по базовой линии текущей строки**
  + right - **Выравнивает изображение по правому краю окна**
  + top - **Верхняя граница изображения выравнивается по самому высокому элементу текущей строки**

alt - **Альтернативный текст для изображения**

border - **Толщина рамки вокруг изображения**

height - **Высота изображени**

hspace - **Горизонтальный отступ от изображения до окружающего контента**

longdesc -**Указывает адрес документа, где содержится аннотация к картинке**

lowsrc - **Адрес изображения низкого качества**

src - **Путь к графическому файлу**

vspace - **Вертикальный отступ от изображения до окружающего контента**

width - **Ширина изображения**

Таблицы

**<table></table>** - тег, для создания области внутри которого будет отображена сама таблица.

## Атрибуты для тега <table>:

align - **Определяет выравнивание таблицы:**

* left - **Выравнивание таблицы по левому краю**
* center - **Выравнивание таблицы по центру**
* right - **Выравнивание по правому краю**

background - **Задает фоновый рисунок в таблиц**

bgcolor - **Цвет фона таблицы**

border - **Толщина рамки в пикселах**

bordercolor - **Цвет рамки**

cellpadding - **Отступ от рамки до содержимого ячейки**

cellspacing - **Расстояние между ячейками**

cols - **Число колонок в таблице**

frame - **Сообщает браузеру, как отображать границы вокруг таблицы:**

* void - **Не отрисовывать границы**
* border - **Граница вокруг таблицы**
* above - **Граница по верхнему краю таблицы**
* below - **Граница снизу таблицы**
* hsides - **Добавить только горизонтальные границы (сверху и снизу таблицы)**
* vsides - **Рисовать только вертикальные границы (слева и справа от таблицы)**
* rhs - **Граница только на правой стороне таблицы**
* ihs - **Граница только на левой стороне таблицы**

height - **Высота таблицы**

rules - **Сообщает браузеру, где отображать границы между ячейками:**

* all - **Линия рисуется вокруг каждой ячейки таблицы**
* groups - **Линия отображается между группами, которые образуются тегами** <thead>**,** <tfoot>**,** <tbody>**,** <colgroup> **или** <col>
* cols - **Линия отображается между колонками**
* none - **Все границы скрываются**
* rows - **Граница рисуется между строками таблицы, созданых через тег** <tr>

summary - **Краткое описание таблицы**

width -**Ширина таблицы**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<tr></tr> - **Новая строка таблицы**

<th></th> - **Ячейка заголовка таблицы**

<td></td> - **Обычная ячейка (столбец) таблицы**

## Атрибуты тега <td>:

abbr - **Краткое описание содержимого ячейки**

align - **Определяет выравнивание содержимого ячейки по горизонтали:**

* left - **Выравнивание содержимого ячейки по левому краю**
* center - **Выравнивание по центру**
* right - **Выравнивание по правому краю**
* justify - **Выравнивание по ширине (одновременно по правому и левому краю)**
* char - **Выравнивает по указанному символу (<td** align="char" char="c"**>**...**</td>)**

background - **Задает фоновый рисунок в ячейке**

bgcolor - **Цвет фона ячейки**

bordercolor - **Цвет рамки**

charoff - **Смещает содержимое ячейки относительно заданного символа**

colspan - **Объединяет горизонтальные ячейки**

headers - **Позволяет связать ячейки с заголовком**

height - **Высота ячейки**

nowrap - **Запрещает перенос строк**

rowspan - **Объединяет вертикальные ячейки**

valign - **Выравнивание содержимого ячейки по вертикали:**

* top - **Выравнивание содержимого ячейки по верхнему краю строки**
* middle - **Выравнивание по середине**
* bottom - **Выравнивание по нижнему краю**
* baseline - **Выравнивание по базовой линии, при этом происходит привязка содержимого ячейки к одной линии**

width - **Ширина ячейки**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<caption></caption> - **Заголовок таблицы**

Пример:

<!—создаем объект таблицы-->

<table>

<!—создаем новую строку-->

<tr>

<!—создаем две ячейки заголовков-->

<th>&nbsp</th><th>&nbsp</th>

</tr>

<!—создаем новую строку-->

<tr>

<!—создаем две обычные ячейки-->

<td>&nbsp</td><td>&nbsp</td>

</tr>

<!—завершаем объект таблицы-->

</table>

## Разметка макета с помощью таблицы

|  |  |  |
| --- | --- | --- |
| Ширина 100% высота 10% | | |
| Ширина 15 % высота 80 % | Ширина определится автоматически  высота 80 % | Ширина 15 % высота 80 % |
| Ширина 100% высота определится автоматически | | |

<html>

<head>

<title>разметка страницы при помощи таблицы</title>

</head>

<body >

<table width='100%' height='100%' valign='top'>

<tr height='10%' bgcolor='silver'>

<td colspan='3'>&nbsp</td>

</tr>

<tr height='80%'>

<td width='15%' bgcolor='green'>&nbsp</td>

<td bgcolor='yellow'>&nbsp</td>

<td width='15%' bgcolor='green'>&nbsp</td>

</tr>

<tr bgcolor='silver'>

<td colspan='3'>&nbsp</td>

</tr>

</body>

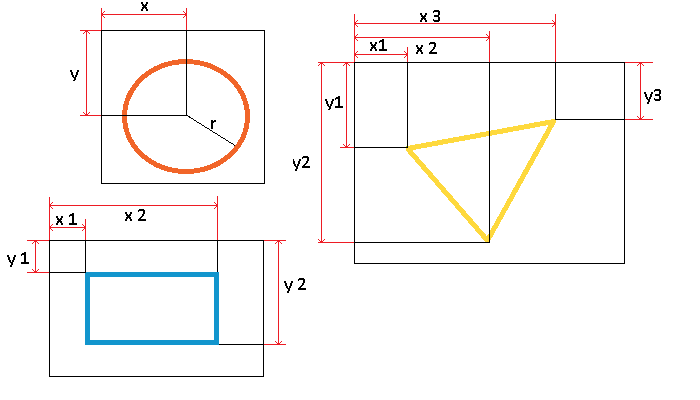
Карта изображений

Тег **<map>** используется для определения изображения-карты. Изображение-карта представляет собой изображение с определённой активной областью. Элемент **<map>** содержит ряд **<area>** тегов, которые определяют интерактивные области в изображении-карте, т.е. при клике на определённую область изображения происходят определённые действия, например, открывается отдельная страница с описанием данного участка изображения.

Атрибут **name** элемента **<map>** является обязательным, он связан с атрибутом **usemap** элемента **<img>** (создает связь между изображением и картой).

Атрибутами тега **<area>** мы указываем как координаты области (атрибут **coords**), так и тип необходимой нам фигуры (атрибут **shape**):

|  |  |
| --- | --- |
| **Значение атрибута** | **Определение** |
| **default** | Весь регион |
| **rect** | Прямоугольная область (**x1,y1,x2,y2**) |
| **circle** | Круглая область (**x,y,r- радиус**) |
| **poly** | Многоугольную область (**x1,y1,x2,y2,...,xn,yn**) |



Пример:

**<html>**

**<head>**

**<title>**Пример использование тега <map>**</title>**

**</head>**

**<body>**

**<p>**Выбери фигуру:**</p>**

**<img** **src = "map.png" usemap = "#figuri" alt = "Доступны 4 фигуры для выбора"** **>**

**<map name = "figuri"** **>**

**<area** **shape = "rect" coords = "25,25,125,125" href = "red.html" alt = "Красный квадрат" >**

**<area** **shape = "circle" coords = "200,75,50" href = "green.html" alt = "Зелёный круг." >**

**<area shape = "poly" coords = "325,25,262,125,388,125" href = "blue.html" alt = "Голубой треугольник" >**

**<area shape = "poly" coords = "450,25,435,60,400,75,435,90,450,125,465,90,500,75,465,60" href = "yellow.html" alt = "Желтая звезда" >**

**</map>**

**</body>**

**</html>**

Форма

**<form></form>**

Элемент **<input>** является основным элементом формы и определяет пользовательское поле для ввода информации. Поле ввода принимает различный вид, в зависимости от значения атрибута **type**, применённого к данному элементу:

<form >

<input type =” ”>

</form>

|  |  |  |
| --- | --- | --- |
| **Значение** | **Описание** | |
| **button** | Определяет кнопку. Как правило, используется в основном совместно с языком программирования *JavaScript* для активации скрипта. |
| **checkbox** включено выключено | Элемент графического пользовательского интерфейса, позволяющий пользователю управлять параметром с двумя состояниями — **☑ включено** и **☐ выключено**. Элемент также называют флажок, флаговая кнопка, чекбокс, галочка. | |
| **colorhtml5** | Элемент управления для задания цвета (определяет палитру цветов). | |
| **datehtml5** | Элемент управления для ввода даты в формате **DD.MM.YYYY** (день, месяц и год). В элементе не указывается время. | |
| **datetime-localhtml5** | Элемент управления для ввода даты в формате **DD.MM.YYYYThh:mm** (день, месяц, год, часы и минуты). В элементе не указывается часовой пояс. | |
| **emailhtml5** | Определяет поле для ввода адреса электронной почты. | |
| **file** | Элемент управления, который позволяет пользователю выбрать файл. | |
| **hidden** | Элемент управления, который определяет скрытое поле ввода. | |
| **image** | Элемент управления, который определяет графическую кнопку *"отправить форму"* (значение **submit**). По аналогии с тегом [**<img>**](https://basicweb.ru/html/tag_img.php) вы должны использовать атрибут [**src**](https://basicweb.ru/html/tag_input_att_src.php), чтобы задать путь к изображению и атрибут [**alt**](https://basicweb.ru/html/tag_input_att_alt.php), чтобы указать альтернативный текст, если изображение не будет загружено по каким-то причинам. Размеры изображения задаются атрибутами [**widthhtml5**](https://basicweb.ru/html/tag_input_att_width.php) (ширина) и [**heighthtml5**](https://basicweb.ru/html/tag_input_att_height.php) (высота). | |
| **monthhtml5** | Элемент управления для ввода месяца и года (**Month YYYY**). В элементе не указывается часовой пояс. | |
| **numberhtml5** | Элемент управления для ввода числа с плавающей точкой. | |
| **password** | Определяет однострочное текстовое поле, предназначенное для ввода пароля (символы, введенные внутри поля скрываются). Используйте атрибут **maxlength** HTML тега [**<input>**](https://basicweb.ru/html/tag_input.php), чтобы задать максимальную длину значения, которое может быть введено в поле. | |
| **radio** выбрано не выбрано | Элемент графического пользовательского интерфейса, который позволяет пользователю выбрать одно значение из предопределенной группы значений. Элемент, как правило, называют **радиокнопка** (*radio button*) или **переключатель**. | |
| **rangehtml5** | Элемент управления для ввода номера (регулятор), в котором ввод точного значения не является важным. Этот тип управления использует следующие значения по умолчанию, если атрибуты не указаны:   * **min = "0"** * **max = "100"** * **value = min + (max - min) / 2**, или **min** (если **max** меньше чем **min**) * **step = "1"** | |
| **reset** | Определяет кнопку сброса содержимого формы до значений, установленных по умолчанию. | |
| **searchhtml5** | Определяет однострочное текстовое поле для ввода строки поиска | |
| **submit** | Определяет кнопку отправки формы | |
| **telhtml5** | Определяет элемент управления для ввода телефонного номера. | |
| **text** | Определяет однострочное текстовое поле (по умолчанию имеет ширину в 20 символов). **Это значение по умолчанию.** | |
| **timehtml5** | Определяет элемент управления для ввода времени без указания часового пояса (**hh:mm**). | |
| **urlhtml5** | Определяет поле для ввода абсолютного URL-адреса. | |
| **weekhtml5** | Определяет элемент управления для ввода порядкого номера недели в году и самого года (Неделя **NN, YYYY**). В элементе не указывается часовой пояс. | |
|  |  | |

Пример:

**<html>**

**<head>**

**<title>**HTML формы**</title>**

**</head>**

**<body>**

**<form>**

Имя: **<input** **type = "text"** **name = "firstname"** **value =**  **"Введите имя"> <br> <br>**

Фамилия: **<input** **type = "text"** **name = "lastname"** **value =**  **"Введите фамилию"> <br> <br>**

Мужчина **<input**  **type = "radio"** **name = "sex"** **value =**  **"male"** **checked>**

Женщина **<input** **type = "radio"** **name = "sex"** **value =**  **"female"> <br> <br>**

О себе: **<br>**

**<input** **type = "checkbox"** **name = "type1"** **value = "low">**Меньше 18**<br>**

**<input** **type = "checkbox"** **name = "type2"** **value = "2old">**От 18 до 35**<br>**

**<input type = "checkbox"** **name = "type3"** **value = "high">**Больше 35**<br>**

**<input** **type = "checkbox"** **name = "type4"** **value = "busy">**Женат /-а**<br>**

**<input type = "checkbox"** **name = "type5"** **value = "cat">**Есть кошка**<br>**

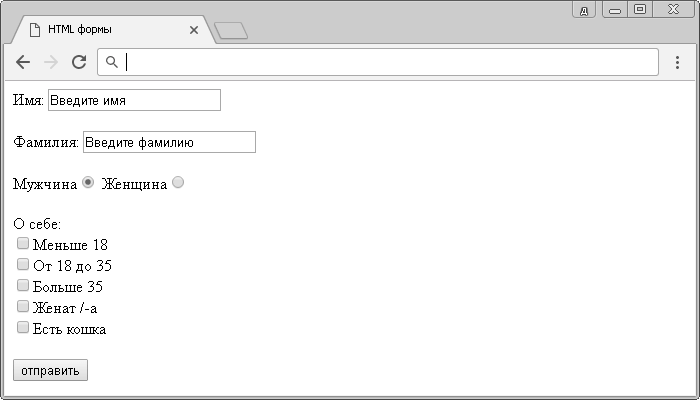
**<br>**

**<input type = "submit"** **name =** **"submit"** **value = "отправить">**

**</form>**

**</body>**

**</html>**



## Связь с <input> - Тег <label>

Тег **<label>** устанавливает связь между определенной меткой, в качестве которой обычно выступает текст, и элементом формы **(<input>, <select>, <textarea>)**.

Такая связь необходима, чтобы изменять значения элементов формы при нажатии курсором мыши на текст. Кроме того, с помощью <label> можно устанавливать горячие клавиши на клавиатуре и переходить на активный элемент подобно ссылкам.

Существует два способа связывания объекта и метки:

1. Использование идентификатора **id** внутри элемента формы и указании его имени в качестве атрибута **for** тега **<label>**
2. Необходимо поместить элемент формы внутри тега **<label>**

**Синтаксис:**

**<input** id="идентификатор"**><label** for="идентификатор"**>**Текст**</label>**

**<label><input** type="..."**>** Текст**</label>**

Пример:

<!DOCTYPE HTML>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Тег LABEL**</title>**

**</head>**

**<body>**

**<form** action="handler.php"**>**

**<p><b>**Lorem ipsum dolor sit amet...**</b></p>** **<p><input** type="checkbox" id="check1"**><label** for="check1"**>**Lorem**</label><Br>**

**<input** type="checkbox" id="check2"**><label** for="check2"**>**Ipsum**</label><Br>**

**<input** type="checkbox" id="check3"**><label** for="check3"**>**Dolor**</label><Br>**

**<input** type="checkbox" id="check4"**><label** for="check4"**>**Sit amet**</label></p>**

**</form>**

**</body>**

**</html>**

## Выпадающий список - <select></select>

Синтаксис:

**<select>**

**<option>**Пункт 1**</option>**

**<option>**Пункт 2**</option>**

**</select>**

Атрибуты:

**accesskey** - Позволяет перейти к списку с помощью некоторого сочетания клавиш.

**autofocus** - Устанавливает, что список получает фокус после загрузки страницы.

**disabled** - Блокирует доступ и изменение элемента.

**form** - Связывает список с формой.

**multiple** - Позволяет одновременно выбирать сразу несколько элементов списка.

**name** - Имя элемента для отправки на сервер или обращения через скрипты.

**required** - Список обязателен для выбора перед отправкой формы.

**size** - Количество отображаемых строк списка.

**tabindex** - Определяет последовательность перехода между элементами при нажатии на клавишу Tab

## Атрибут <textarea></textarea>

**<textarea></textarea>** - Элемент формы для создания области, в которую можно вводить несколько строк текста.

**cols** - Ширина поля в символах.

**rows** - Высота поля в строках текста.

**autofocus** - Автоматическое получение фокуса.

**disabled** - Блокирует доступ и изменение элемента.

**form** - Связывает текстовое поле с формой по её идентификатору.

**maxlength** - Максимальное число введенных символов.

**name** - Имя поля, предназначено для того, чтобы обработчик формы мог его идентифицировать.

**placeholder** - Указывает замещающийся текст.

**readonly** - Устанавливает, что поле не может изменяться пользователем.

**required** - Обязательное для заполнения поле.

**tabindex** - Порядок перехода между элементами при нажатии на клавишу Tab.

**wrap** - Параметры переноса строк.

* **soft** - Длинный текст, который самостоятельно не помещается в поле по ширине, будет автоматически перенесен на новую строку, однако передаваться на сервер будет как одна строка. Нажатие клавиши Enter устанавливает перенос текста, который сохраняется при отправке формы.
* **hard** - Слова в поле переносятся механически, чтобы они поместились в размер области, и при отправке на сервер места автоматического переноса сохраняются. При этом значении обязательно должен присутствовать атрибут cols.
* **off** - Нестандартное значение. Переносы строк отключены. При введении длинного текста без переносов, он будет печататься в одну строку, при этом будет отображаться полоса прокрутки.

## Атрибуты тега <input> </input>

**accept** - Устанавливает фильтр на типы файлов, которые вы можете отправить через поле загрузки файлов.

Пример:

**<input** type="file" name="img" accept="image/jpeg,image/png,image/gif"**>**

**accesskey** - Переход к элементу с помощью комбинации клавиш.

**align** - Определяет выравнивание изображения.

|  |  |
| --- | --- |
| <input type="image" src="sample.gif" align="bottom"> | Выравнивание нижней границы изображения по окружающему тексту.. |
| <input type="image" src="sample.gif" align="left"> | Выравнивает изображение по левому краю окна. |
| <input type="image" src="sample.gif" align="middle"> | Выравнивание середины изображения по базовой линии текущей строки. |
| <input type="image" src="sample.gif" align="right"> | Выравнивает изображение по правому краю окна. |
| <input type="image" src="sample.gif" align="top"> | Верхняя граница изображения выравнивается по самому высокому элементу текущей строки. |

**alt** - Альтернативный текст для кнопки с изображением.

**autocomplete** - Включает или отключает автозаполнение.

* **on** - Включает автозаполнение текста.
* **off** - Отключает автозаполнение. Это значение обычно используется для отмены сохранения в браузере важных данных (паролей, номеров банковских карт), а также редко вводимых или уникальных данных (капча).

**autofocus** - Устанавливает фокус в поле формы. В таком поле можно сразу набирать текст без явного щелчка по нему курсором мыши

**border** - Толщина рамки вокруг изображения.

**checked** - Предварительно активированный переключатель или флажок.

Пример:

**<input** type="checkbox" checked**>**

**disabled** - Блокирует доступ и изменение элемента.

Синтаксис:

**<input** type="..." disabled**>**

**form** - Связывает поле с формой по её идентификатору. Такая связь необходима в случае, когда поле располагается за пределами *<form>*, например, при создании её программно или по соображениям

**formaction** - Определяет адрес обработчика формы. Это программа, которая получает данные формы и производит с ними желаемые действия. Атрибут formaction по своему действию аналогично атрибуту action тега <form>.

Пример:

**<input** type="submit" handler.php" value="Отправить"**>**

**formenctype** - Устанавливает способ кодирования данных формы при их отправке на сервер.

* **application/x-www-form-urlencoded** - Вместо пробелов ставится +, символы вроде русских букв кодируются их шестнадцатеричными значениями (например, %D0%9F%D0%B5%D1%82%D1%8F вместо Петя).
* **multipart/form-data** - Данные не кодируются. Это значение применяется при отправке файлов.
* **text/plain** - Пробелы заменяются знаком +, буквы и другие символы не кодируются.

**formmethod** - Сообщает браузеру каким методом следует передавать данные формы на сервер.

* **get** - Этот метод предназначен для передачи данных формы непосредственно в адресной строке в виде пар «имя=значение», которые добавляются к адресу страницы после вопросительного знака и разделяются между собой амперсандом (символ &). Полный адрес к примеру будет http://site.ru/doc/?name=Vasya&password=pup. Объём данных в методе ограничен 4 Кб.
* **post** - Посылает на сервер данные в запросе браузера, объём пересылаемых данных ограничен лишь настройками сервера.

**formnovalidate** - Отменяет встроенную проверку данных на корректность.

Синтаксис:

**<input** type="submit" formnovalidate**>**

**formtarget** - Определяет окно или фрейм в которое будет загружаться результат, возвращаемый обработчиком формы.

* **\_blank** - Загружает страницу в новое окно браузера.
* **\_self** - Загружает страницу в текущее окно.
* **\_parent** - Загружает страницу во фрейм-родитель, если фреймов нет, то это значение работает как \_self.
* **\_top** - Отменяет все фреймы и загружает страницу в полном окне браузера, если фреймов нет, то это значение работает как \_self.

Синтаксис:

**<input** type="submit" formtarget="<имя окна> | \_blank | \_self | \_parent | \_top"**>**

**list** – Указывает на список вариантов, созданный с помощью тега **<datalist>**, которые можно выбирать при наборе текста. Изначально этот список скрыт и становится доступным при получении полем фокуса.

Пример:

**<form>**

**<p><input** list="cocktail"**></p>**

**<datalist** id="cocktail"**>**

**<option>**Аперитивы**</option>**

**<option>**Горячие**</option>**

**<option>**Десертные**</option>**

**<option>**Диджестивы**</option>**

**<option>**Молочные**</option>**

**<option>**Слоистые**</option>**

**</datalist>**

**</form>**

Выбор текста из списка

**max** - Верхнее значение для ввода числа или даты.

**min** - Нижнее значение для ввода числа или даты.

Пример 1:

**<form** action="handler.php"**>**

**<p>**Укажите вашу температуру в градусах Цельсия**</p>**

**<input** type="number" min="35" max="41"

 value="36.6" step="0.1" **/></p>**

**</form>**

Пример:

Дата в формате ГГГГ-ММ-ДД (например: 2012-12-22) для *type="date"*

**<form** action="handler.php"**>**

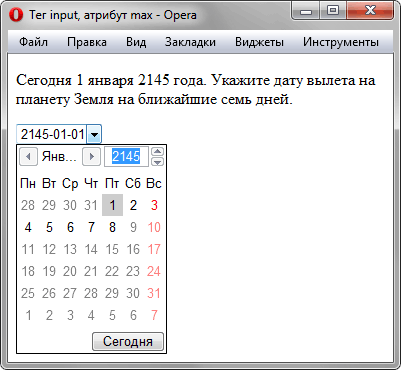
**<p>**Сегодня 1 января 2145 года.

Укажите дату вылета на планету Земля на ближайшие

семь дней.**</p>**

**<p><input** type="date" min="2145-01-01" max="2145-01-08"

 value="2145-01-01" **/></p>**



**maxlength** - Максимальное количество символов разрешенных в тексте.

**multiple** - Позволяет загрузить несколько файлов одновременно.

Синтаксис:

**<input** type="file" multiple**>**

**<input** type="email" multiple**>**

**name** - Имя поля, предназначено для того, чтобы обработчик формы мог его идентифицировать.

**pattern** - Указывает регулярное выражение, согласно которому требуется вводить и проверять данные в поле формы. Если присутствует атрибут *pattern*, то форма не будет отправляться, пока поле не будет заполнено правильно.

|  |  |
| --- | --- |
| **Выражение** | **Описание** |
| \d [0-9] | Одна цифра от 0 до 9. |
| \D [^0-9] | Любой символ кроме цифры. |
| \s | Пробел. |
| [A-Z] | Только заглавная латинская буква. |
| [A-Za-z] | Только латинская буква в любом регистре. |
| [А-Яа-яЁё] | Только русская буква в любом регистре. |
| [A-Za-zА-Яа-яЁё] | Любая буква русского и латинского алфавита. |
| [0-9]{3} | Три цифры. |
| [A-Za-z]{6,} | Не менее шести латинских букв. |
| [0-9]{,3} | Не более трёх цифр. |
| [0-9]{5,10} | От пяти до десяти цифр. |
| ^[a-zA-Z]+$ | Любое слово на латинице. |
| ^[А-Яа-яЁё\s]+$ | Любое слово на русском включая пробелы. |
| ^[ 0-9]+$ | Любое число. |
| [0-9]{6} | Почтовый индекс. |
| \d+(,\d{2})? | Число в формате 1,34 (разделитель запятая). |
| \d+(\.\d{2})? | Число в формате 2.10 (разделитель точка). |
| \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3} | IP-адрес |

Пример:

**<form>**

**<p>**Введите телефон в формате 2-xxx-xxx, где вместо x

должна быть цифра:**</p>**

**<p><input** type="tel" pattern="2-[0-9]{3}-[0-9]{3}"**></p>**

**<p><input** type="submit" value="Отправить"**></p>**

**</form>**

**placeholder** - Выводит подсказывающий текст.

Пример:

**<form** action="handler.php"**>**

**<p><input** type=search placeholder="Введите текст для поиска"**>**

**<input** type="submit" value="Искать"**></p>**

**</form>**

**readonly** - Устанавливает, что поле не может изменяться пользователем.

Синтаксис:

**<input** type="text" readonly**>**

**<input** type="password" readonly**>**

**required** - Обязательное для заполнения поле.

**size** - Ширина текстового поля.

**src** - Адрес графического файла для поля с изображением.

**step** – Устанавливает шаг изменения числа для ползунков и полей ввода чисел.

Синтаксис:

**<input** type="number" step="число"**>**

**<input** type="range" step="число"**>**

**type** - Сообщает браузеру, к какому типу относится элемент формы.

**value** - Значение элемента.

В зависимости от типа элемента атрибут value выступает в следующей роли:

* для кнопок (input type="button | reset | submit") устанавливает текстовую надпись на них;
* для текстовых полей (input type="password | text") указывает предварительно введенную строку. Пользователь может стирать текст и вводить свои символы, но при использовании в форме кнопки Reset пользовательский текст очищается и восстанавливается введенный в атрибуте value;
* для флажков и переключателей (input type="checkbox | radio") уникально определяет каждый элемент, с тем, чтобы клиентская или серверная программа могла однозначно установить, какой пункт выбрал пользователь.
* для файлового поля (input type="file") не оказывает влияние.

## Атрибуты тега <form> </form>

**accept-charset** - Устанавливает кодировку, в которой сервер может принимать и обрабатывать данные

Пример:

**<form** action="handler.php" accept-charset="windows-1251"**>**

**<p>**...**</p>**

**</form>**

**action** - Адрес программы или документа, который обрабатывает данные формы

Пример 1:

**<form** action="handler.php"**>**

**<p>**...**</p>**

**</form>**

Пример 2:

В качестве обработчика можно указать адрес электронной почты, начиная его с ключевого слова mailto:. При отправке формы будет запущена почтовая программа установленная по умолчанию. В целях безопасности в браузере установлено, что отправить незаметно информацию, введенную в форме, по почте невозможно. Для корректной интерпретации данных используйте атрибут enctype="text/plain" в теге *<form>*.

**<form** action="mailto:vlad@htmlbook.ru" enctype="text/plain"**>**

**<p><input** type="submit" value="Написать письмо"**></p>**

**</form>**

**autocomplete** - Включает авто заполнение полей формы

* on - Включает автозаполнение формы
* off - Отключает автозаполнение. Это значение обычно используется для отмены сохранения в браузере важных данных (паролей, номеров банковских карт), а также редко вводимых или уникальных данных (капча)

Пример:

**<form** autocomplete="on | off"**>**...**</form>**

**enctype** - Способ кодирования данных формы

* **application/x-www-form-urlencoded** - Вместо пробелов ставится +, символы вроде русских букв кодируются их шестнадцатеричными значениями (например, %D0%90%D0%BD%D1%8F вместо Аня)
* **multipart/form-data** - Данные не кодируются. Это значение применяется при отправке файлов
* **text/plain** - Пробелы заменяются знаком +, буквы и другие символы не кодируются

Пример:

**<form** action="handler.php"

 enctype="multipart/form-data" method="post"**>**

**method** - Метод протокола HTTP

* **get** - Этот метод является одним из самых распространенных и предназначен для получения требуемой информации и передачи данных в адресной строке. Пары «имя=значение» присоединяются в этом случае к адресу после вопросительного знака и разделяются между собой амперсандом (символ &). Удобство использования метода get заключается в том, что адрес со всеми параметрами можно использовать неоднократно, сохранив его, например, в закладки браузера, а также менять значения параметров прямо в адресной строке
* **post** - Метод post посылает на сервер данные в запросе браузера. Это позволяет отправлять большее количество данных, чем доступно методу get, поскольку у него установлено ограничение в 4 Кб. Большие объемы данных используются в форумах, почтовых службах, заполнении базы данных, при пересылке файлов и др

Пример:

**<form** action="handler.php" method="post"**>**

**name** - Имя формы.В качестве имени используется набор символов, включая числа и буквы. JavaScript чувствителен к регистру, поэтому при обращении к форме по имени через скрипты соблюдайте то же написание, что и в атрибуте name.

**novalidate** - Отменяет встроенную проверку данных формы на корректность ввод

Пример:

**<form** action="handler.php" novalidate**>**

target - Имя окна или фрейма, куда обработчик будет загружать возвращаемый результат

* **\_blank** - Загружает страницу в новое окно браузера
* **\_self** - Загружает страницу в текущее окно
* **\_parent** - Загружает страницу во фрейм-родитель, если фреймов нет, то это значение работает как \_self.
* **\_top** - Отменяет все фреймы и загружает страницу в полном окне браузера, если фреймов нет, то это значение работает как \_self.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Группировка данных в форме

**<fieldset> </fieldset>** - Группировка элементов формы

**<legend> </legend>** - Заголовок для <fieldset> элемента, с помощью которого группируются элементы формы

Пример:

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>**Группировка данных формы**</title>**

**</head>**

**<body>**

**<form>**

**<fieldset>**

**<legend>**Пожалуйста, представьтесь**</legend>**

Имя: **<input** **type =**  **"text"** **name = "firstname"** **value =**  **"Введите имя"> <br> <br>**

Фамилия: **<input** **type =**  **"text"** **name = "lastname"** **value =**  **"Введите фамилию"> <br> <br>**

Мужчина **<input** **type =**  **"radio"** **name = "sex"** **value =**  **"male"** **checked>**

Женщина **<input** **type =**  **"radio"** **name = "sex"** **value =**  **"female"><br> <br>**

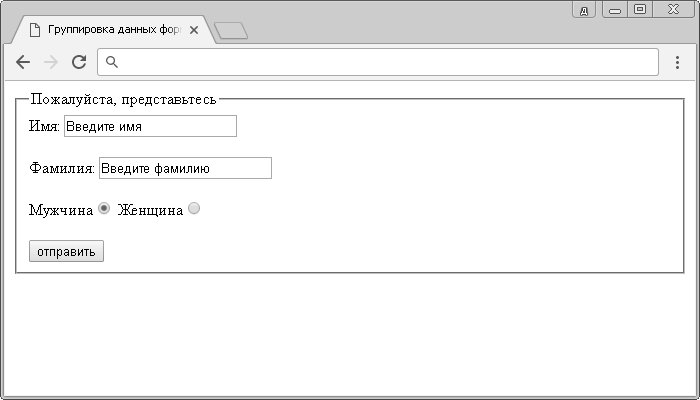
**<input** **type = "submit"** **name = "submit"** **value = "отправить">**

**</fieldset>**

**</form>**

**</body>**

**</html>**



Подключение отдельного html файла на страницу

Зачастую при верстке страниц, необходимо добавление на несколько страниц одних и тех-же блоков. Для того чтобы не терять время и не верстать один и тот же блок много раз, можно создать данный объект в отдельном html – файле и просто подключать его в необходимом месте html – страницы.

В чем преимущество данного подключения?

При разработке может появится необходимость изменить данный блок после добавления его на несколько страниц. И поэтому, чтобы не искать этот блок среди множества созданных страниц и не вносить коррективы по отдельности.

Для начала необходимо вставить строку ниже внутри **<head></head>** чтобы активировать дальнейших скрипт, который мы напишем:

<script src="https://code.jquery.com/jquery-1.10.2.js"></script>

Далее в <body></body> прописывается сам скрипт для подключения отдельной html – страницы внутри текущего тега **<div></div>** :

<script>

        $(function() {

            $("#place\_for\_menu\_header").load("Dop/Menu\_header/menu\_header.html");

        });

    </script>

Где: $("#place\_for\_menu\_header – уникальное имя id контейнера **<div></div>**, а Dop/Menu\_header/menu\_header.html – относительный путь содержимого html – файла который вы хотите вставить.

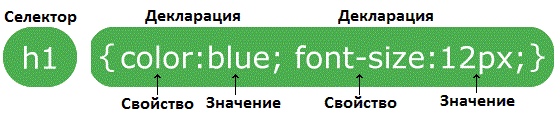
CSS

## Комментарий CSS

Для комментирования кода Html в теге **<head></head>** используется / \*… \* /

## Селекторы

**Синтаксис:**



**Виды селекторов:**

1. **Простые селекторы**
   1. **id**

Селектор id использует атрибут id HTML элемента для выбора определенного элемента.

Идентификатор (id) элемента является уникальным на странице, поэтому селектор id используется для выбора одного уникального элемента!

Чтобы выбрать элемент с определенным идентификатором, напишите символ хеша (#), а затем id элемента.

Пример:

#para1 {  
  text-align: center;  
  color: red;  
}

* 1. **Class**

Селектор class выбирает HTML элементы с определенным атрибутом класса.

Чтобы выбрать элементы с определенным классом, введите символ точки (.), а затем имя класса.

Пример:

.center {  
  text-align: center;  
  color: red;  
}

Вы также можете указать, что класс должен затрагивать только определенные HTML элементы.

Пример:

p.center {  
  text-align: center;  
  color: red;  
}

HTML элементы также могут ссылаться на несколько классов.

Пример:

<p class="center large">This paragraph refers to two classes.</p>

* 1. **Универсальный селектор**

Универсальный селектор (\*) выделяет все элементы HTML на странице.

Пример:

\* {  
  text-align: center;  
  color: blue;  
}

* 1. **Селектор группировки**

Селектор группировки выбирает все HTML элементы с одинаковыми определениями стиля.

В примере элементы h1, h2 и p имеют одинаковые определения стиля):

h1 {  
  text-align: center;  
  color: red;  
}  
  
h2 {  
  text-align: center;  
  color: red;  
}  
  
p {  
  text-align: center;  
  color: red;  
}

1. **Комбинаторные селекторы**
   1. **Селектор потомков**

Селектор потомков соответствует всем элементам, которые являются потомками указанного элемента.

В примере выбираются все элементы <p> внутри <div> элементов:

div p {  
  background-color: yellow;  
}

* 1. **Селектор дочерний (>)**

Пример:

div > p {  
  background-color: yellow;  
}

* 1. **Соседний родственный селектор (+)**

Соседний родственный селектор используется для выбора элемента, который находится непосредственно после другого конкретного элемента.

Родственные элементы должны иметь один и тот же родительский элемент, а "смежные" элементы должны "следовать после".

В примере выбирается первый элемент <p>, который размещается сразу после элемента <div>:

div + p {  
  background-color: yellow;  
}

1. **Псевдо-класс**

**Псевдо-класс** используется для определения особого состояния элемента.

Например, его можно использовать, для:

* Стилизации элемента, когда пользователь наводит на него курсор
* Стилизации посещённых и не посещённых ссылок по-разному
* Стилизации элемента, когда он находится в фокусе

**Синтаксис:**

selector:pseudo-class {  
  property:value;  
}

* 1. **Якорные псевдо-классы**

Пример:

/\* непосещенная ссылка \*/  
a:link {  
  color: #FF0000;  
}  
  
/\* посещенная ссылка \*/  
a:visited {  
  color: #00FF00;  
}  
  
/\* курсор над ссылкой \*/  
a:hover {  
  color: #FF00FF;  
}  
  
/\* выбранная (активная) ссылка \*/  
a:active {  
  color: #0000FF;  
}

* 1. **Псевдо-классы и CSS-классы**

Псевдо-классы можно комбинировать с CSS-классами:

Когда вы наведёте курсор на ссылку в примере, она изменит цвет:

Пример:

a.highlight:hover {  
  color: #ff0000;  
}

## Все CSS Псевдо-классы

|  |  |  |
| --- | --- | --- |
| **Селектор** | **Пример** | **Описание** |
| [**:active**](https://w3schoolsrus.github.io/cssref/sel_active.html) | a:active | Выбирает активную ссылку |
| [**:checked**](https://w3schoolsrus.github.io/cssref/sel_checked.html) | input:checked | Выбирает каждый проверенный элемент <input> |
| [**:disabled**](https://w3schoolsrus.github.io/cssref/sel_disabled.html) | input:disabled | Выбирает каждый отключенный элемент <input> |
| [**:empty**](https://w3schoolsrus.github.io/cssref/sel_empty.html) | p:empty | Выбирает каждый элемент <p>, который не имеет дочерних элементов |
| [**:enabled**](https://w3schoolsrus.github.io/cssref/sel_enabled.html) | input:enabled | Выбирает каждый включенный элемент <input> |
| [**:first-child**](https://w3schoolsrus.github.io/cssref/sel_firstchild.html) | p:first-child | Выбирает все элементы <p>, которые являются первым потомком своего родителя |
| [**:first-of-type**](https://w3schoolsrus.github.io/cssref/sel_first-of-type.html) | p:first-of-type | Выбирает каждый элемент <p> который является первым элементом <p> своего родителя |
| [**:focus**](https://w3schoolsrus.github.io/cssref/sel_focus.html) | input:focus | Выбирает элемент <input> который имеет фокус |
| [**:hover**](https://w3schoolsrus.github.io/cssref/sel_hover.html) | a:hover | Выбирает ссылки при наведении мыши |
| [**:in-range**](https://w3schoolsrus.github.io/cssref/sel_in-range.html) | input:in-range | Выбирает элементы <input> со значением в указанном диапазоне |
| [**:invalid**](https://w3schoolsrus.github.io/cssref/sel_invalid.html) | input:invalid | Выбирает все элементы <input> с недопустимым значением |
| [**:lang(*language*)**](https://w3schoolsrus.github.io/cssref/sel_lang.html) | p:lang(it) | Выбирает каждый элемент <p> со значением атрибута lang, начинающимся с "it" |
| [**:last-child**](https://w3schoolsrus.github.io/cssref/sel_last-child.html) | p:last-child | Выбирает все элементы <p>, которые являются последним дочерним элементом своего родителя |
| [**:last-of-type**](https://w3schoolsrus.github.io/cssref/sel_last-of-type.html) | p:last-of-type | Выбирает каждый элемент <p>, который является последним элементом <p> своего родителя |
| [**:link**](https://w3schoolsrus.github.io/cssref/sel_link.html) | a:link | Выбирает все не посещенные ссылки |
| [**:not(selector)**](https://w3schoolsrus.github.io/cssref/sel_not.html) | :not(p) | Выбирает каждый элемент, который не является элементом <p> |
| [**:nth-child(n)**](https://w3schoolsrus.github.io/cssref/sel_nth-child.html) | p:nth-child(2) | Выбирает каждый элемент <p>, который является вторым дочерним элементом своего родителя |
| [**:nth-last-child(n)**](https://w3schoolsrus.github.io/cssref/sel_nth-last-child.html) | p:nth-last-child(2) | Выбирает каждый элемент <p>, который является вторым дочерним элементом своего родителя, считая от последнего дочернего элемента |
| [**:nth-last-of-type(n)**](https://w3schoolsrus.github.io/cssref/sel_nth-last-of-type.html) | p:nth-last-of-type(2) | Выбирает каждый элемент <p>, который является вторым элементом <p> своего родителя, считая от последнего потомка |
| [**:nth-of-type(n)**](https://w3schoolsrus.github.io/cssref/sel_nth-of-type.html) | p:nth-of-type(2) | Выбирает каждый элемент <p> который является вторым элементом <p> своего родителя |
| [**:only-of-type**](https://w3schoolsrus.github.io/cssref/sel_only-of-type.html) | p:only-of-type | Выбирает каждый элемент <p> который является единственным элементом <p> своего родителя |
| [**:only-child**](https://w3schoolsrus.github.io/cssref/sel_only-child.html) | p:only-child | Выбирает каждый элемент <p> который является единственным потомком своего родителя |
| [**:optional**](https://w3schoolsrus.github.io/cssref/sel_optional.html) | input:optional | Выбирает элементы <input> без атрибута "required" |
| [**:out-of-range**](https://w3schoolsrus.github.io/cssref/sel_out-of-range.html) | input:out-of-range | Выбирает элементы <input> со значением вне указанного диапазона |
| [**:read-only**](https://w3schoolsrus.github.io/cssref/sel_read-only.html) | input:read-only | Выбирает элементы <input> с указанным атрибутом "readonly" |
| [**:read-write**](https://w3schoolsrus.github.io/cssref/sel_read-write.html) | input:read-write | Выбирает элементы <input> без атрибута "readonly" |
| [**:required**](https://w3schoolsrus.github.io/cssref/sel_required.html) | input:required | Выбирает элементы <input> с указанным атрибутом "required" |
| [**:root**](https://w3schoolsrus.github.io/cssref/sel_root.html) | root | Выбирает корневой элемент документа |
| [**:target**](https://w3schoolsrus.github.io/cssref/sel_target.html) | #news:target | Выбирает текущий активный элемент #news (при нажатии на URL, содержащий это имя якоря) |
| [**:valid**](https://w3schoolsrus.github.io/cssref/sel_valid.html) | input:valid | Выбирает все элементы <input> с допустимым значением |
| [**:visited**](https://w3schoolsrus.github.io/cssref/sel_visited.html) | a:visited | Выбирает все посещенные ссылки |

## Все CSS Псевдо-элементы

|  |  |  |
| --- | --- | --- |
| **Селектор** | **Пример** | **Описание** |
| [**::after**](https://w3schoolsrus.github.io/cssref/sel_after.html) | p::after | Вставляет контент после каждого элемента <p> |
| [**::before**](https://w3schoolsrus.github.io/cssref/sel_before.html) | p::before | Вставляет контент перед каждым элементом <p> |
| [**::first-letter**](https://w3schoolsrus.github.io/cssref/sel_firstletter.html) | p::first-letter | Выбирает первую букву каждого элемента <p> |
| [**::first-line**](https://w3schoolsrus.github.io/cssref/sel_firstline.html) | p::first-line | Выбирает первую строку каждого элемента <p> |
| [**::selection**](https://w3schoolsrus.github.io/cssref/sel_selection.html) | p::selection | Выбирает часть элемента, которая выбрана пользователем |

## Подключение CSS

1. **Внешняя таблица стилей –** использование отдельного css файла **(РЕКОМЕНДУЕТСЯ ИСПОЛЬЗОВАНИЕ ЭТОГО МЕТОДА)**

С помощью внешней таблицы стилей вы можете изменить внешний вид всего сайта, изменив только один файл!

Каждая HTML страница должна содержать ссылку на файл внешней таблицы стилей внутри элемента <link>.

Пример подключения стиля:

<head>  
<link rel="stylesheet" href="mystyle.css">  
</head>

Пример файла стиля:

body {  
  background-color: lightblue;  
}  
  
h1 {  
  color: navy;  
  margin-left: 20px;  
}

1. **Внутренняя таблица стилей (Используется для небольшой корректировки стилей html – страницы. НЕ ИСПОЛЬЗУЕТСЯ ДЛЯ ОСНОВНОГО ПОДКЛЮЧЕНИЯ CSS – стилей)**

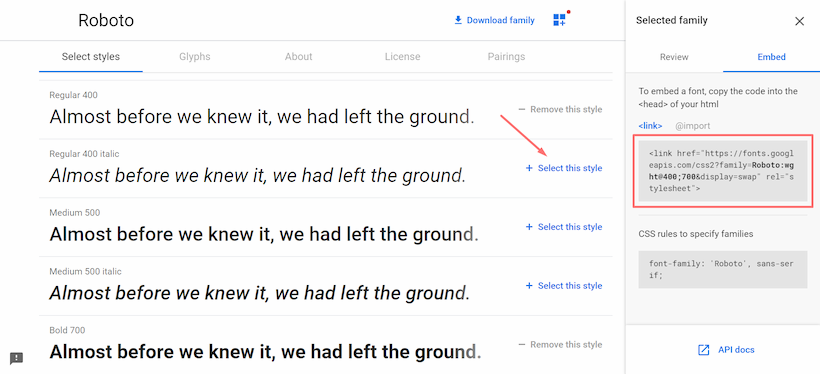
Пример:

<head>  
<style>  
body {  
  background-color: linen;  
}  
  
h1 {  
  color: maroon;  
  margin-left: 40px;  
}  
</style>  
</head>

Подключение сторонних шрифтов

1. Подключение с помощью [Google Fonts](https://fonts.google.com/)

Полученный код необходимо вставить в html файл, внутри тега **<head></head>**



1. Подключение с помощью правила @font-face

* После того, как шрифты добавлены в проект, их нужно подключить в CSS-файле. Для этого используется правило **@font-face**. В самом базовом варианте оно будет включать:
* Название шрифта, которое затем нужно использовать, чтобы задать элементам подключённый шрифт.
* Адрес файла со шрифтом, который нужно подключить, и его формат. Если адресов несколько, их можно указать через запятую. В этом случае важен порядок — браузер будет последовательно пытаться подключить файлы. Первым должен быть самый подходящий формат, а далее — запасные варианты.
* Начертания: жирное, курсивное и так далее. Для каждого начертания нужно отдельное правило **@font-face**.

**Для улучшения производительности правило @font-face лучше всего прописывать в самом начале CSS-файла. Так браузер сможет раньше начать обработку шрифта.**

Пример:

@font-face { font-family: "Roboto";

font-style: normal;

font-weight: 400; /\* Браузер сначала попробует найти шрифт локально \*/

src:

/\* Если не получилось, загрузит woff2 \*/ url("/fonts/roboto.woff2")format("woff2"),

/\* Если браузер не поддерживает woff2, загрузит woff \*/ url("/fonts/roboto.woff") format("woff"); }

/\* Теперь можно использовать шрифт \*/

body { font-family: "Roboto", "Arial", sans-serif; }

Основные свойства CSS

**Для перехода к свойству CSS в документе нажмите на текст c помощью сочетания клавиш: CTRL+клик мыши**.

### B

1. background
2. background-attachment
3. background-clip
4. [background-color](#s4)
5. background-image
6. background-origin
7. [background-position](#s7)
8. [background-repeat](#s8)
9. [background-size](#s9)
10. [border](#s10)
11. border-bottom
12. border-bottom-color
13. border-bottom-left-radius
14. border-bottom-right-radius
15. border-bottom-style
16. border-bottom-width
17. border-collapse
18. [border-color](#s18)
19. [border-image](#s19)
20. border-left
21. border-left-color
22. [border-left-style](#s22)
23. border-left-width
24. [border-radius](#s24)
25. border-right
26. border-right-color
27. border-right-style
28. border-right-width
29. border-spacing
30. border-style
31. border-top
32. border-top-color
33. border-top-left-radius
34. border-top-right-radius
35. border-top-style
36. border-top-width
37. border-width
38. bottom
39. [box-shadow](#s39)
40. [box-sizing](#s40)

### C

1. caption-side
2. clear
3. clip
4. [color](#s44)
5. column-count
6. column-gap
7. column-rule
8. column-width
9. columns
10. content
11. counter-increment
12. counter-reset
13. [cursor](#s53)

### D

1. direction
2. [display](#s55)

### E

1. empty-cells

### F

1. [float](#s57)
2. font
3. font-family
4. font-size
5. font-stretch
6. [font-style](#s62)
7. font-variant
8. [font-weight](#s64)

[fill](#fill)

[fill-opacity](#fill_opacity)

[filter](#filter)

### H

1. height

### L

1. left
2. letter-spacing
3. line-height
4. list-style
5. [list-style-image](#s70)
6. [list-style-position](#s71)
7. [list-style-type](#s72)

[line-height](#lineheight)

### M

1. [margin](#s73)
2. margin-bottom
3. margin-left
4. margin-right
5. margin-top
6. [max-height](#ы68)
7. max-width
8. min-height
9. min-width

### O

1. opacity
2. orphans
3. [outline](#s84)
4. outline-color
5. [outline-offset](#s86)
6. outline-style
7. outline-width
8. [overflow](#s89)
9. overflow-x
10. overflow-y

### P

1. [padding](#s92)
2. padding-bottom
3. padding-left
4. padding-right
5. padding-top
6. page-break-after
7. page-break-before
8. page-break-inside
9. [position](#s100)

### Q

1. quotes

### R

1. resize
2. right

### T

1. tab-size
2. table-layout
3. [text-align](#s106)
4. text-align-last
5. [text-decoration](#s108)
6. text-decoration-color
7. text-decoration-line
8. text-decoration-style
9. [text-indent](#s112)
10. text-overflow
11. [text-shadow](#s114)
12. text-transform
13. top
14. transform
15. transform-origin
16. transition
17. [transition-delay](#s120)
18. transition-duration
19. transition-property
20. transition-timing-function

### U

1. unicode-bidi

[user-select](#user_select)

### V

1. [vertical-align](#s125)
2. visibility

### W

1. white-space
2. widows
3. width
4. word-break
5. word-spacing
6. word-wrap
7. writing-mode

### Z

1. [z-index](#s134)
2. **color –** Определяет цвет текста элемента

Синтаксис:

color: цвет

Пример:

**<style>**

.letter {

color: red; /\* Цвет символа \*/

font-size: 200%; /\* Размер шрифта \*/

}

**p** {

color: **rgb(**49, 151, 116**)**; /\* Цвет текста \*/

}

**</style>**

1. **line-height** - Устанавливает интерлиньяж (межстрочный интервал) текста, отсчет ведется от базовой линии шрифта. При обычных обстоятельствах расстояние между строками зависит от вида и размера шрифта и определяется браузером автоматически. Отрицательное значение межстрочного расстояния не допускается.

Синтаксис:

line-height: множитель | значение | проценты | normal | inherit

**Значения:**

Любое число больше нуля воспринимается как множитель от размера шрифта текущего текста. Например, значение 1.5 устанавливает полуторный межстрочный интервал. В качестве значений принимаются также любые единицы длины, принятые в CSS — пикселы (px), дюймы (in), пункты (pt) и др. Разрешается использовать процентную запись, в этом случае за 100% берется высота шрифта.

**normal** - Расстояние между строк вычисляется автоматически.

**inherit** - Наследует значение родителя.

1. **font-weight -** Устанавливает насыщенность шрифта. Значение устанавливается от 100 до 900 с шагом 100.

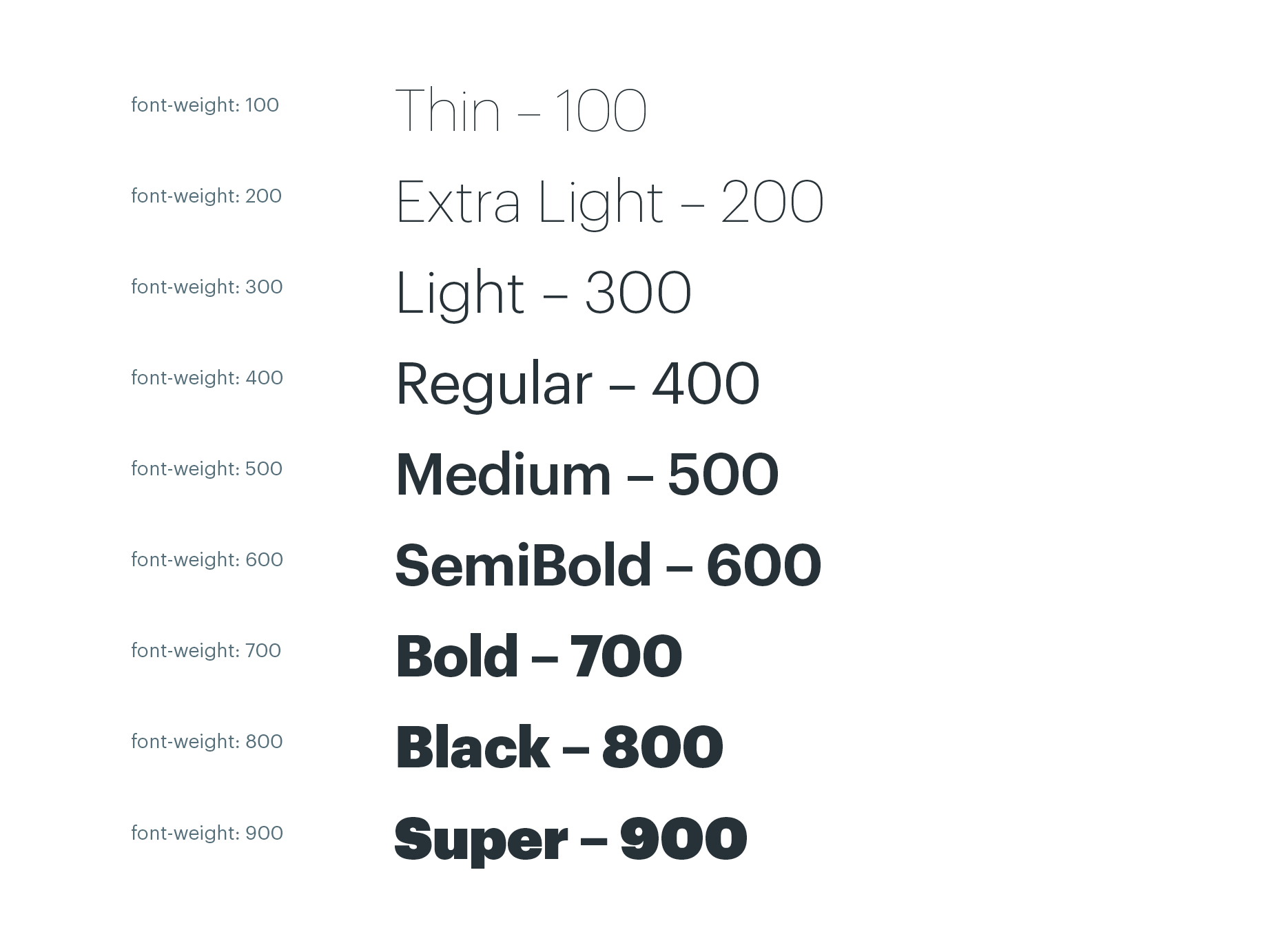
Синтаксис:

font-weight: bold|bolder|lighter|normal|100|200|300|400|500|600|700|800|900

Значения:

Насыщенность шрифта задаётся с помощью ключевых слов: **bold** — полужирное начертание, **normal** — нормальное начертание. Также допустимо использовать условные единицы **от 100 до 900**. Значения **bolder** и **lighter** изменяют жирность относительно насыщенности родителя, соответственно, в большую и меньшую сторону.

Таблица жирности шрифтов



1. **text-align -** Определяет горизонтальное выравнивание текста в пределах элемента.

Синтаксис:

text-align: **center** | justify | left | right | start | end

**Атрибуты:**

**center** - Выравнивание текста по центру. Текст помещается по центру горизонтали окна браузера или контейнера, где расположен текстовый блок. Строки текста словно нанизываются на невидимую ось, которая проходит по центру веб-страницы. Подобный способ выравнивания активно используется в заголовках и различных подписях, вроде подрисуночных, он придает официальный и солидный вид оформлению текста. Во всех других случаях выравнивание по центру применяется редко по той причине, что читать большой объем такого текста неудобно.

**justify** - Выравнивание по ширине, что означает одновременное выравнивание по левому и правому краю. Чтобы произвести это действие браузер в этом случае добавляет пробелы между словами.

**left** - Выравнивание текста по левому краю. В этом случае строки текста выравнивается по левому краю, а правый край располагается «лесенкой». Такой способ выравнивания является наиболее популярным на сайтах, поскольку позволяет пользователю легко отыскивать взглядом новую строку и комфортно читать большой текст.

**right** - Выравнивание текста по правому краю. Этот способ выравнивания выступает в роли антагониста предыдущему типу. А именно, строки текста равняются по правому краю, а левый остается «рваным». Из-за того, что левый край не выровнен, а именно с него начинается чтение новых строк, такой текст читать труднее, чем, если бы он был выровнен по левому краю. Поэтому выравнивание по правому краю применяется обычно для коротких заголовков объемом не более трех строк. Мы не рассматриваем специфичные сайты, где текст приходится читать справа налево, там возможно подобный способ выравнивания и пригодится.

**auto** - Не изменяет положение элемента.

**inherit** - Наследует значение родителя.

**start** - Аналогично значению left, если текст идёт слева направо и right, когда текст идёт справа налево.

**end** - Аналогично значению right, если текст идёт слева направо и left, когда текст идёт справа налево.

1. **vertical-align** - Выравнивает элемент по вертикали относительно своего родителя, окружающего текста или ячейки таблицы.

**Синтаксис:**

vertical-align: baseline|bottom|middle|sub|super|text-bottom|text-top|top|inherit | значение | проценты

Значения:

**baseline** - Выравнивает базовую линию текущего элемента по базовой линии родителя. Если родительский элемент не имеет базовой линии, то за нее принимается нижняя граница элемента.

**bottom** - Выравнивает основание текущего элемента по нижней части элемента строки, расположенного ниже всех.

**middle** - Выравнивание средней точки элемента по базовой линии родителя плюс половина высоты родительского элемента.

**sub** - Элемент изображается как подстрочный, в виде нижнего индекса. Размер шрифта при этом не меняется.

**super** - Элемент изображается как надстрочный, в виде верхнего индекса. Размер шрифта остается прежним.

**text-bottom** - Нижняя граница элемента выравнивается по самому нижнему краю текущей строки.

**text-top** - Верхняя граница элемента выравнивается по самому высокому текстовому элементу текущей строки.

**top** - Выравнивание верхнего края элемента по верху самого высокого элемента строки.

**inherit** - Наследует значение родителя.

В качестве значения также можно использовать проценты, пикселы или другие доступные единицы. Положительное число смещает элемент вверх относительно базовой линии, в то время как отрицательное число опускает его вниз. При использовании процентов, отсчет ведется от значения свойства **line-height**, при этом 0% аналогично значению **baseline**.

Для выравнивания по вертикали в ячейках таблицы применяются следующие значения.

**baseline** - Выравнивает базовую линию ячейки с базовой линией первой текстовой строки или другого вложенного элемента.

**bottom** - Выравнивает по нижнему краю ячейки.

**middle** - Выравнивает по середине ячейки.

**top** - Выравнивает содержимое ячейки по ее верхнему краю.

Пример:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**vertical-align**</title>**

**</head>**

**<body>**

**<div** style="font-size: 2em"**>**

T**<span** style="vertical-align: sub"**>**E**</span>**X и L**<span**

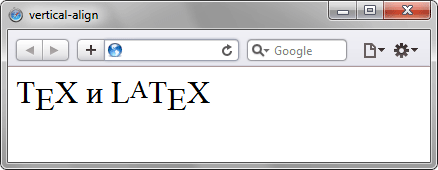
 style="vertical-align: 5px; font-size: 0.8em"**>**A**</span>**T**<span**

 style="vertical-align: sub"**>**E**</span>**X

**</div>**

**</body>**

**</html>**



1. **font-style** - Определяет начертание шрифта — обычное, курсивное или наклонное.

Синтаксис:

font-style: normal | italic | oblique | inherit

**Атрибуты:**

**normal** - Обычное начертание текста.

**italic** - Курсивное начертание.

**oblique** - Наклонное начертание. Курсив и наклонный шрифт при всей их похожести не одно и то же. Курсив это специальный шрифт имитирующий рукописный, наклонный же образуется путем наклона обычных знаков вправо.

**inherit** - Наследует значение родителя.

1. **transition-delay** - устанавливает время ожидания перед запуском эффекта перехода. Значение *0s* или *0ms* запускает анимацию сразу же. Отрицательное значение также включает анимацию без задержек, но может привести к изменению вида начала анимации.

**Синтаксис:**

transition-delay: <время> [,<время>]\*

1. **text-decoration -** Добавляет оформление текста в виде его подчеркивания, перечеркивания, линии над текстом и мигания. Одновременно можно применить более одного стиля, перечисляя значения через пробел.

Синтаксис:

text-decoration: [ blink || line-through || overline || underline ] | none | inherit

**Атрибуты:**

**blink** - Устанавливает мигающий текст. Такой текст периодически, примерно раз в секунду исчезает, потом вновь появляется на прежнем месте. Это значение в настоящее время не поддерживается браузерами и осуждается в CSS3, взамен рекомендуется использовать анимацию.

**line-through** - Создает перечеркнутый текст (пример).

**overline** - Линия проходит над текстом (пример).

**underline** - Устанавливает подчеркнутый текст (пример).

**none** - Отменяет все эффекты, в том числе и подчеркивания у ссылок, которое задано по умолчанию.

**inherit** - Значение наследуется у родителя.

1. **Border** - Универсальное свойство border позволяет одновременно установить толщину, стиль и цвет границы вокруг элемента. Значения могут идти в любом порядке, разделяясь пробелом, браузер сам определит, какое из них соответствует нужному свойству. Для установки границы только на определенных сторонах элемента, воспользуйтесь свойствами border-top, border-bottom, border-left, border-right.

Синтаксис:

border: [border-width || border-style || border-color]

Пример 1:

**<head>**

**<meta** charset="utf-8"**>**

**<title>**border**</title>**

**<style>**

.brd {

border: 4px double black; /\* Параметры границы \*/

background: #fc3; /\* Цвет фона \*/

padding: 10px; /\* Поля вокруг текста \*/

}

**</style>**

**</head>**

Пример 1:

**<head>**

**<meta** charset="utf-8"**>**

**<title>**border**</title>**

**<style>**

div {

width: 300px;

height: 100px;

border-left: 1px solid red;

border-right: 1px solid red; }

**</style>**

**<body>**

<div></div>

**</body>**

**</head>**



1. **border-radius** - Устанавливает радиус скругления уголков рамки. Если рамка не задана, то скругление также происходит и с фоном.

Синтаксис:

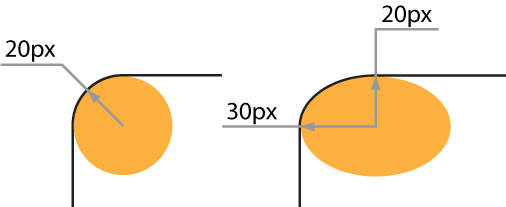
border-radius: <радиус>{1,4} [ / <радиус>{1,4}]

**Значения:**

Разрешается использовать одно, два, три или четыре значения, перечисляя их через пробел (значения в таблице). Также допустимо писать два значения через слэш (/). В качестве значений указываются числа в любом допустимом для CSS формате. В случае применения процентов, отсчет ведется относительно ширины блока.

|  |  |
| --- | --- |
| **Число значений** | **Результат** |
| 1 | Радиус указывается для всех четырех уголков. |
| 2 | Первое значение задает радиус верхнего левого и нижнего правого уголка, второе значение — верхнего правого и нижнего левого уголка. |
| 3 | Первое значение задает радиус для верхнего левого уголка, второе — одновременно для верхнего правого и нижнего левого, а третье — для нижнего правого уголка. |
| 4 | По очереди устанавливает радиус для верхнего левого, верхнего правого, нижнего правого и нижнего левого уголка. |

В случае задания двух параметров через слэш, то первый задает радиус по горизонтали, а второй по вертикали (эллиптические уголки). На рис. 1 показана разница между обычным скругленным уголком и эллиптическим уголком.



Пример:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**border-radius**</title>**

**<style>**

.radius {

background: #f0f0f0; /\* Цвет фона \*/

border: 1px solid black; /\* Параметры рамки \*/

padding: 15px; /\* Поля вокруг текста \*/

margin-bottom: 10px; /\* Отступ снизу \*/

}

**</style>**

**</head>**

**<body>**

**<div** style="border-radius: 50px 0 0 50px;" class="radius"**>**

border-radius: 50px 0 0 50px;

**</div>**

**<div** style="border-radius: 40px 10px" class="radius"**>**

border-radius: 40px 10px;

**</div>**

**<div** style="border-radius: 10em/1em;" class="radius"**>**

border-radius: 13em/3em;

**</div>**

**<div** style="border-radius: 13em 0.5em/1em 0.5em;" class="radius"**>**

border-radius: 13em 0.5em/1em 0.5em;

**</div>**

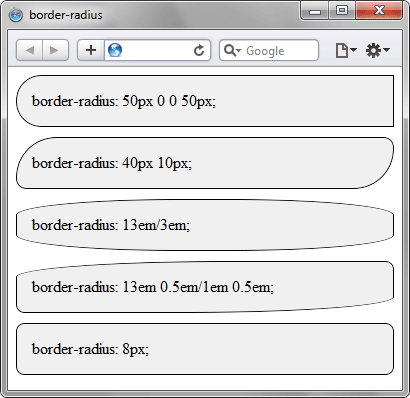
**<div** style="border-radius: 8px;" class="radius"**>**

border-radius: 8px;

**</div>**

**</body>**

**</html>**



1. **border-style** - Устанавливает стиль границы вокруг элемента. Допустимо задавать индивидуальные стили для разных сторон элемента.

**Вид рамки в зависимости от стиля и толщины границы элемента:**

|  |  |  |  |
| --- | --- | --- | --- |
| **1 пиксел** | **3 пиксела** | **5 пикселов** | **7 пикселов** |
| dotted | dotted | dotted | dotted |
| dashed | dashed | dashed | dashed |
| solid | solid | solid | solid |
| double | double | double | double |
| groove | groove | groove | groove |
| ridge | ridge | ridge | ridge |
| inset | inset | inset | inset |
| outset | outset | outset | outset |

*Кроме перечисленных в таблице значений используются следующие ключевые слова.*

**none** - Не отображает границу и ее толщина (border-width) задается нулевой.

**hidden** - Имеет тот же эффект, что и none за исключением применения border-style к ячейкам таблицы, у которой значение свойства border-collapse установлено как collapse. В этом случае вокруг ячейки граница не будет отображаться вообще.

**inherit** - Наследует значение родителя.

**Зависимость результата использования от числа значений**

|  |  |
| --- | --- |
| **Число значений** | **Результат** |
| 1 | Стиль границы будет задан для всех сторон элемента. |
| 2 | Первое значение устанавливает стиль верхней и нижней границы, второе — левой и правой. |
| 3 | Первое значение задает стиль верхней границы, второе — одновременно левой и правой границы, а третье — нижней границы. |
| 4 | Поочередно устанавливается стиль верхней, правой, нижней и левой границы. |

Пример:

**<head>**

**<meta** charset="utf-8"**>**

**<title>**border-style**</title>**

**<style>**

**p** {

border-style: double; /\* Стиль линии вокруг параграфа \*/

padding: 5px; /\* Поля вокруг текста \*/

}

**</style>**

**</head>**

1. **border-image** - Используется для отображения рисованной рамки вокруг элемента. Толщина рамки задаётся свойством [border](#s10), при этом если указано **border: 0**, то рамка не выводится. При других значениях border рисунок всегда имеет приоритет. Фон, если он задан через свойство background, отображается под рамкой.

**Синтаксис:**

border-image: none | [ <URL> [<число> | <проценты>]{1,4} [/ <толщина>{1,4}]? ] && [stretch | repeat | round]{0,2}

Значения:

**none** - Не отображает рисованную рамку, используется установленный стиль границы.

**url** - Путь к графическому файлу. Обязательный параметр.

Само изображение для создания рамки продемонстрировано на рис. 1 и состоит из девяти областей: четырёх уголков, верхней, правой, нижней, левой стороны и центральной части, в которой выводится содержимое элемента.

|  |  |
| --- | --- |
| **Число значений** | **Результат** |
| 1 | Устанавливает границы одинаковой толщины на каждой стороне рисунка. |
| 2 | Первое значение устанавливает высоту верхней и нижней границы, второе — левой и правой. |
| 3 | Первое значение определяет высоту верхней границы, второе — левой и правой, а третье — высоту нижней границы. |
| 4 | Поочередно устанавливается размеры верхней, правой, нижней и левой границы. |

**<проценты>** - Аналогично <числу>, но значения задаются в процентах. Тот или другой параметр обязателен.

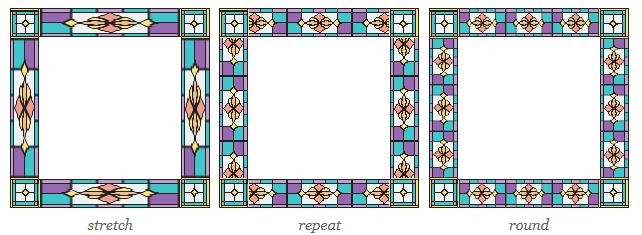
**<толщина>** - Через слэш пишется одно, два, три или четыре значения толщины границы на каждой стороне элемента. Является аналогом border-width и использует тот же синтаксис.

**stretch** - Растягивает рисунок границы до размеров элемента. Это значение используется по умолчанию.

**repeat** - Повторяет рисунок границы.

**round** - Повторяет рисунок и масштабирует его так, чтобы на стороне элемента оказалось целое число изображений.

Влияние этих параметров на вид рамки показано на рисунке:



Пример:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**border-image**</title>**

**<style>**

**div** {

border: 30px solid #40c4c8;

padding: 20px;

-moz-border-image: **url(**images/bg-image.png**)** 30 round round;

-webkit-border-image: **url(**images/bg-image.png**)** 30 round round;

-o-border-image: **url(**images/bg-image.png**)** 30 round round;

border-image: **url(**images/bg-image.png**)** 30 round round;

}

**</style>**

**</head>**

**<body>**

**<div>**Витраж представляет собой композицию сделанную из

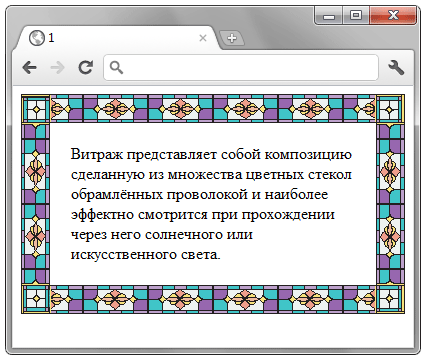
множества цветных стекол обрамлённых проволокой и наиболее

эффектно смотрится при прохождении через него солнечного

или искусственного света.**</div>**

**</body>**

**</html>**



1. **border-color** - Устанавливает цвет границы на разных сторонах элемента. Свойство позволяет задать цвет границы сразу для всех сторон элемента или только для указанных.

**Изменение цвета в зависимости от числа значений**

|  |  |
| --- | --- |
| **Число значений** | **Результат** |
| 1 | Цвет границы будет установлен для всех сторон элемента. |
| 2 | Первое значение устанавливает цвет верхней и нижней границы, второе — левой и правой. |
| 3 | Первое значение задает цвет верхней границы, второе — одновременно левой и правой границы, а третье — нижней границы. |
| 4 | Поочередно устанавливается цвет верхней, правой, нижней и левой границы. |

Пример:

<head>

<meta charset="utf-8">

<title>border-color</title>

<style>

h1 {

border-color: red white; /\* Цвет границы \*/

border-style: solid; /\* Стиль границы \*/

}

p {

border-color: #008a77; /\* Цвет границы \*/

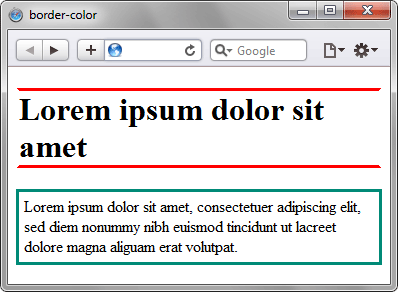
border-style: solid; /\* Стиль границы \*/

padding: 5px; /\* Поля вокруг текста \*/

}

</style>

</head>



1. **outline** - Универсальное свойство, одновременно устанавливающее цвет, стиль и толщину внешней границы на всех четырех сторонах элемента. В отличие от линии, задаваемой через [**border**](#s10), свойство **outline** не влияет на положение блока и его ширину. Также нельзя задать параметры линии на отдельных сторонах элемента, outline применяется сразу ко всем четырём сторонам.

**Синтаксис:**

outline: outline-color || outline-style || outline-width | inherit

Значение:

**outline-color** - Задает цвет линии в любом допустимом для CSS формате.

**outline-style** - Стиль линии.

**outline-width** - Толщина границы.

**inherit** - Наследует значение родителя.

**Пример:**

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**outline**</title>**

**<style>**

.photo **img** {

padding: 20px; /\* Поля вокруг изображения \*/

margin-right: 10px; /\* Отступ справа \*/

margin-bottom: 10px; /\* Отступ снизу \*/

outline: 1px solid #666; /\* Параметры рамки \*/

background: #f0f0f0; /\* Цвет фона \*/

float: left; /\* Обтекание по правому краю \*/

}

**</style>**

**</head>**

**<body>**

**<div** class="photo"**>**

**<img** src="images/girl.jpg" alt="Девочка с муфтой"**>**

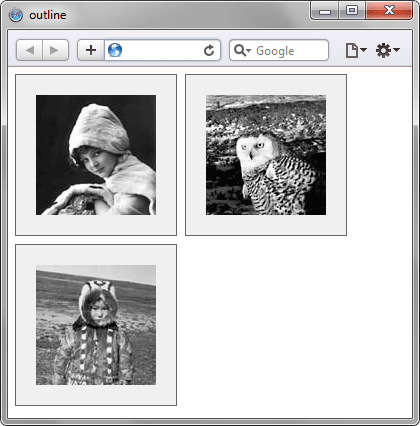
**<img** src="images/owl.jpg" alt="Сова"**>**

**<img** src="images/boy.jpg" alt="Эвенкийский мальчик"**>**

**</div>**

**</body>**

**</html>**



1. **outline-offset -** Устанавливает расстояние между рамкой, созданной с помощью свойства [**outline**](#s84), и краем или границей элемента, добавленной через [**border**](#s10).

**Синтаксис:**

outline-offset: <размер> | inherit

Значение:

**<размер>** - Задаёт расстояние от края элемента до рамки. Отрицательное значение отображает рамку внутри элемента, положительное — вокруг элемента.

**inherit** - Наследует значение родителя.

**Пример:**

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**outline-offset**</title>**

**<style>**

.clue {

background: **url(**images/leather.jpg**)**; /\* Фоновый рисунок \*/

outline: 2px dashed **rgba(**255,255,255,0.8**)**; /\* Пунктирная рамка \*/

outline-offset: -10px; /\* Выводим рамку внутри элемента \*/

padding: 10px; /\* Поля \*/

min-height: 100px; /\* Минимальная высота \*/

}

**</style>**

**</head>**

**<body>**

**<div** class="clue"**></div>**

**</body>**

**</html>**



1. **text-indent –** Добавление красной строки

Пример:

**<head>**

**<meta** charset="utf-8"**>**

**<title>**border-style**</title>**

**<style>**

**p** {

text-indent: 20px;

}

**</style>**

**</head>**

1. **position** - Устанавливает способ позиционирования элемента относительно окна браузера или других объектов на веб-странице

Синтаксис:

position: absolute | fixed | relative | static | inherit

**Значение:**

**absolute** - Указывает, что элемент абсолютно позиционирован, при этом другие элементы отображаются на веб-странице словно абсолютно позиционированного элемента и нет. Положение элемента задается свойствами left, top, right и bottom, также на положение влияет значение свойства position родительского элемента. Так, если у родителя значение position установлено как static или родителя нет, то отсчет координат ведется от края окна браузера. Если у родителя значение position задано как fixed, relative или absolute, то отсчет координат ведется от края родительского элемента.

**fixed** - По своему действию это значение близко к absolute, но в отличие от него привязывается к указанной свойствами left, top, right и bottom точке на экране и не меняет своего положения при прокрутке веб-страницы. Браузер **firefox** вообще не отображает полосы прокрутки, если положение элемента задано фиксированным, и оно не помещается целиком в окно браузера. В браузере Opera хотя и показываются полосы прокрутки, но они никак не влияют на позицию элемента.

**relative** - Положение элемента устанавливается относительно его исходного места. Добавление свойств left, top, right и bottom изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения.

**static** - Элементы отображаются как обычно. Использование свойств left, top, right и bottom не приводит к каким-либо результатам.

**inherit** - Наследует значение родителя.

Пример:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**position**</title>**

**<style>**

.layer1 {

position: relative; /\* Относительное позиционирование \*/

background: #f0f0f0; /\* Цвет фона \*/

height: 200px; /\* Высота блока \*/

}

.layer2 {

position: absolute; /\* Абсолютное позиционирование \*/

bottom: 15px; /\* Положение от нижнего края \*/

right: 15px; /\* Положение от правого края \*/

line-height: 1px;

}

**</style>**

**</head>**

**<body>**

**<div** class="layer1"**>**

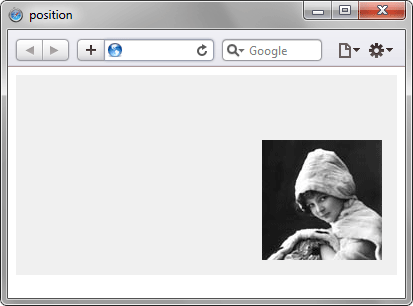
**<div** class="layer2"**>**

**<img** src="images/girl.jpg" alt="Девочка" **/>**

**</div>**

**</div>**

**</body**



1. **line-height** - Устанавливает интерлиньяж (межстрочный интервал) текста, отсчет ведется от базовой линии шрифта. При обычных обстоятельствах расстояние между строками зависит от вида и размера шрифта и определяется браузером автоматически. Отрицательное значение межстрочного расстояния не допускается.

Синтаксис:

ine-height: множитель | значение | проценты | normal | inherit

**Значения**

Любое число больше нуля воспринимается как множитель от размера шрифта текущего текста. Например, значение 1.5 устанавливает полуторный межстрочный интервал. В качестве значений принимаются также любые единицы длины, принятые в CSS — пикселы (px), дюймы (in), пункты (pt) и др. Разрешается использовать процентную запись, в этом случае за 100% берется высота шрифта.

**normal** - Расстояние между строк вычисляется автоматически.

**inherit** - Наследует значение родителя.

Пример:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**line-height**</title>**

**<style>**

**h1** {

line-height: 60%;

}

**p** {

line-height: 1.5;

}

**</style>**

**</head>**

**<body>**

**<h1>**Duis te feugifacilisi**</h1>**

**<p>**Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem

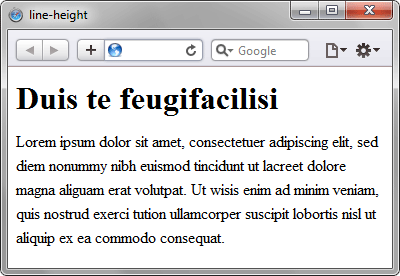
nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.

Ut wisis enim ad minim veniam, quis nostrud exerci tution ullamcorper suscipit

lobortis nisl ut aliquip ex ea commodo consequat.**</p>**

**</body>**

**</html>**



Свойства фона

1. **background-color -** Определяет цвет фона элемента. Хотя это свойство не наследует свойства своего родителя, из-за того, что начальное значение устанавливается прозрачным, цвет фона дочерних элементов совпадает с цветом фона родительского элемента.

Синтаксис:

background-color: <цвет> | transparent | inherit

**Атрибуты:**

**transparent -** Устанавливает прозрачный фон.

**inherit -** Наследует значение родителя.

Пример:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**background-color**</title>**

**<style>**

**body**{

background-color: #3366CC; /\* Цвет фона веб-страницы \*/

}

**h1** {

background-color: **RGB(**249, 201, 16**)**; /\* Цвет фона под заголовком \*/

}

**p** {

background-color: maroon; /\* Цвет фона под текстом параграфа \*/

color: white; /\* Цвет текста \*/

}

**</style>**

**</head>**

**<body>**

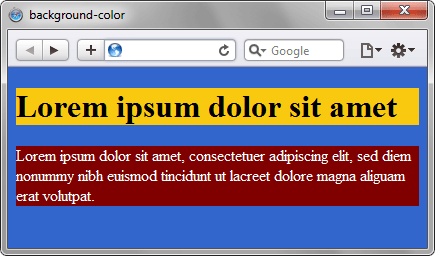
**<h1>**Lorem ipsum dolor sit amet**</h1>**

**<p>**Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem

nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.**</p>**

**</body>**

**</html>**



1. **background-image -** Устанавливает фоновое изображение для элемента. Если одновременно для элемента задан цвет фона, он будет показан, пока фоновая картинка не загрузится полностью. То же произойдет, если изображения не доступны или их показ в браузере отключен. В случае наличия в рисунке прозрачных областей, через них будет проглядывать фоновый цвет. В CSS3 допустимо указывать несколько фоновых изображений, перечисляя их параметры через запятую.

Синтаксис:

background-image: url(путь к файлу) | none[, url(путь к файлу) | none]\*

**Атрибуты:**

**url** - В качестве значения используется путь к графическому файлу, который указывается внутри конструкции url(). Путь к файлу при этом можно писать, как в кавычках (двойных или одинарных), так и без них.

**none** - Отменяет фоновое изображение для элемента.

**inherit** - Наследует значение родителя.

Пример 1:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**background-color**</title>**

**<style>**

div { width: 100%;

height: 300px;

border: 1px solid black;

background-image: **url("image.png")**; /\* задаем путь к фоновой картинке \*/ }

**</style>**

**</head>**

**<body>**

<div></div>

**</body>**

**</html>**



Пример 2:

<html>

<head>

<style type="text/css">

body{

background: url("2.png") repeat-y, url("2.png") repeat-y 100% 0;

background-size: 200px 300px;

}

</style>

</head>

</html>

Этот пример позволяет вставить изображение по правому и левому краю страницы.

1. **background-size** - Масштабирует фоновое изображение согласно заданным размерам.

Синтаксис:

background-size: [ <значение> | <проценты> | auto ]{1,2} | cover | contain

**Значения**

**<значение>** - Задает размер в любых доступных для CSS единицах — пикселы (px), сантиметры (cm), em и др.

**<проценты>** - Задает размер фоновой картинки в процентах от ширины или высоты элемента.

**auto** - Если задано одновременно для ширины и высоты (auto auto), размеры фона остаются исходными; если только для одной стороны картинки (100px auto), то размер вычисляется автоматически исходя из пропорций картинки.

**cover** - Масштабирует изображение с сохранением пропорций так, чтобы его ширина или высота равнялась ширине или высоте блока.

**contain** - Масштабирует изображение с сохранением пропорций таким образом, чтобы картинка целиком поместилась внутрь блока.

1. **background-repeat** - Определяет, как будет повторяться фоновое изображение, установленное с помощью свойства [background-image](#s5). Можно установить повторение рисунка только по горизонтали, по вертикали или в обе стороны. В CSS3 допустимо указывать несколько значений для каждого фона, перечисляя значения через запятую.

Синтаксис:

background-repeat: **<повторение>** [ , **<повторение>** ]\*

**Атрибуты:**

**no-repeat** - Устанавливает одно фоновое изображение в элементе без его повторений, положение которого определяется свойством background-position (по умолчанию в левом верхнем углу). Аналогично no-repeat no-repeat.

**repeat** - Фоновое изображение повторяется по горизонтали и вертикали. Аналогично repeat repeat.

**repeat-x** - Фоновый рисунок повторяется только по горизонтали. Аналогично repeat no-repeat.

**repeat-y** - Фоновый рисунок повторяется только по вертикали. Аналогично no-repeat repeat.

**inherit** - Наследует значение родителя.

**space** - Изображение повторяется столько раз, чтобы полностью заполнить область; если это не удаётся, между картинками добавляется пустое пространство.

**round** - Изображение повторяется так, чтобы в области поместилось целое число рисунков; если это не удаётся сделать, то фоновые рисунки масштабируются.

Пример 1:

**<html>**

**<head>**

**<meta** http-equiv="Content-Type" content="text/html; charset=utf-8" **/>**

**<title>**background-repeat**</title>**

**<style** type="text/css"**>**

**body** {

background-image: **url(**images/bg\_grey.png**)**; /\* Путь к фоновому рисунку \*/

background-position: left bottom; /\* Положение фона \*/

background-repeat: repeat-x; /\* Повторяем фон по горизонтали \*/

}

**</style>**

**</head>**

**<body>**

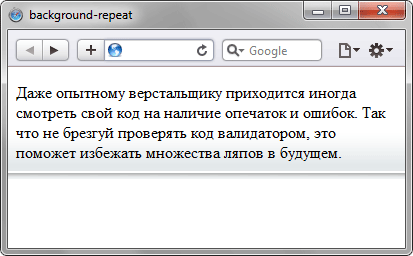
**<p>**Даже опытному верстальщику приходится иногда смотреть свой код

на наличие опечаток и ошибок. Так что не брезгуй проверять код валидатором,

это поможет избежать множества ляпов в будущем.**</p>**

**</body>**

**</html>**



Повторение фона по горизонтали

Фон для блока

Фон для блока

Пример 2:

**<html>**

**<head>**

**<meta**  **/>**

**<title>**background-repeat**</title>**

**<style** type="text/css"**>**

div {

width: 100%;

height: 300px;

border: 1px solid black;

background-image: url("image.png"); /\* задаем путь к фоновой картинке \*/

background-repeat: repeat-x; /\* заставляем картинку повторяться по горизонтали \*/

}

**</style>**

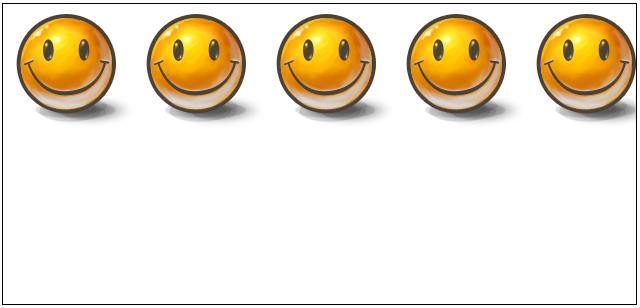
**</head>**

**<body>**

<div></div>

**</body>**

**</html>**



Пример 3:

**<html>**

**<head>**

**<meta**  **/>**

**<title>**background-repeat**</title>**

**<style** type="text/css"**>**

div {

width: 100%;

height: 300px;

border: 1px solid black;

background-image: url("image.png"); /\* задаем путь к фоновой картинке \*/

background-repeat: repeat-y; /\* заставляем картинку повторяться по вертикали \*/

} **</style>**

**</head>**

**<body>**

<div></div>

**</html>**

Пример 4 (Расположение картинки по бокам:

<html>

<head>

<style type="text/css">

body{

background: url("2.png") repeat-y, url("2.png") repeat-y 100% 0;

background-size: 200px 300px;

}

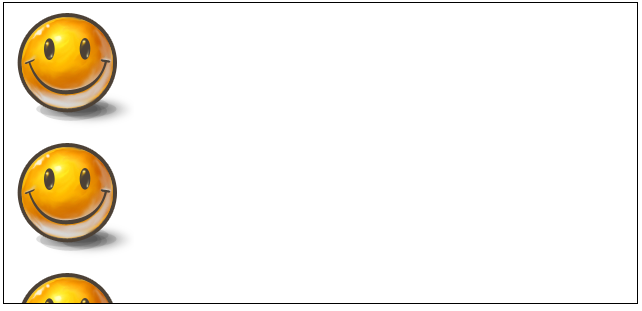
</style>

</head>

<body>

</body>

</html>



1. **background-position -** Задает начальное положение фонового изображения, установленного с помощью свойства [background-image](#s5). В CSS3 допустимо указывать несколько значений для каждого фона, перечисляя значения через запятую.

Синтаксис:

background-position: **<позиция>**[, **<позиция>**]\*

<позиция> = [left | center | right | <проценты> | <значение>] || [top | center | bottom | <проценты> | <значение>] | inherit.

**Значения:**

* Вправо %\px Вниз %\px
* top left = left top = 0% 0% (в левом верхнем углу)
* top = top center = center top = 50% 0% (по центру вверху)
* right top = top right = 100% 0% (в правом верхнем углу)
* left = left center = center left = 0% 50% (по левому краю и по центру)
* center = center center = 50% 50% (по центру)
* right = right center = center right = 100% 50% (по правому краю и по центру)
* bottom left = left bottom = 0% 100% (в левом нижнем углу)
* bottom = bottom center = center bottom = 50% 100% (по центру внизу)
* bottom right = right bottom = 100% 100% (в правом нижнем углу)

Пример 1:

**<html>**

**<head>**

**<meta**  **/>**

**<title>**background-repeat**</title>**

**<style** type="text/css"**>**

div { width: 100%;

height: 300px;

border: 1px solid black;

background-image: **url("image.png")**;

background-repeat: repeat-y; /\* картинка повторяется по оси Y \*/

background-position: right top; /\* позиция картинки - правый верхний угол \*/ }

}

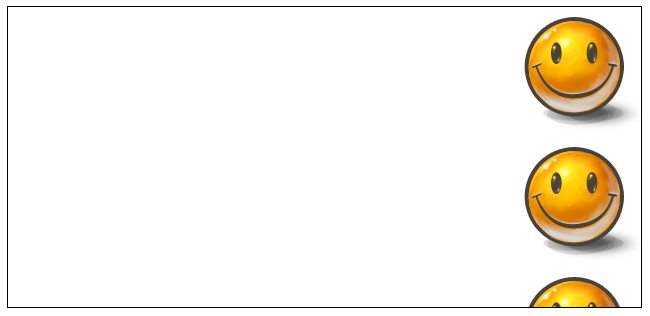
**</style>**

**</head>**

**<body>**

<div></div>

**</html>**



Пример 2:

**<html>**

**<head>**

**<meta**  **/>**

**<title>**background-repeat**</title>**

**<style** type="text/css"**>**

div {

width: 100%;

height: 300px;

border: 1px solid black;

background-image: **url("image.png")**;

background-repeat: no-repeat;

background-position: 90% center;}

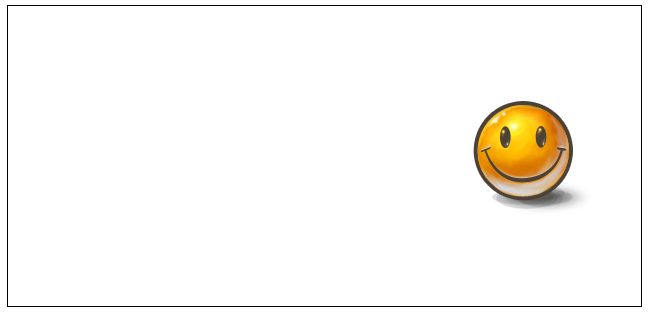
**</style>**

**</head>**

**<body>**

<div></div>

**</html>**



1. **background-attachment** - Свойство background-attachment устанавливает, будет ли прокручиваться фоновое изображение вместе с содержимым элемента. Изображение может быть зафиксировано и оставаться неподвижным, либо перемещаться совместно с документом. В CSS3 можно указать несколько значений для ряда фоновых изображений, перечисляя значения через запятую.

Cинтаксис:

background-attachment: fixed | scroll | local[, fixed | scroll | local]\*

**Значения:**

**fixed** - Делает фоновое изображение элемента неподвижным.

**scroll** - Позволяет перемещаться фону вместе с содержимым.

**inherit** - Наследует значение родителя.

**local** - Фон фиксируется с учётом поведения элемента. Если элемент имеет прокрутку, то фон будет прокручиваться вместе с содержимым, но фон выходящий за рамки элемента остаётся на месте.

Пример:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**background-attachment**</title>**

**<style>**

**body** {

background-image: **url(**images/pattern-left.png**)**, **url(**images/pattern-right.png**)**;

background-repeat: repeat-y, repeat-y;

background-position: left, right;

background-attachment: fixed, fixed;

}

**</style>**

**</head>**

**<body>**

**<div** style="height:2000px"**>**

**</div>**

**</body>**

**</html>**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Фильтры для изображений

1. **filter** - Устанавливает фильтр (визуальный эффект) или их сочетание для элемента. К фильтрам относится изменение прозрачности, добавление тени, трансформация и др.

**Синтаксис:**

filter: progid:DXImageTransform.Microsoft.Фильтр(свойства)

Фильтры:

**alpha** - Настраивает прозрачность объекта.

* **enabled** - Включает или выключает фильтр. Значение true разрешает использование фильтра, false запрещает.
* **finishOpacity** - Устанавливает значение прозрачности в конце градиента.
* **finishx** - Задаёт горизонтальную позицию окончания градиента. Это целое число в процентах от ширины элемента. По умолчанию 0.
* **finishy** - Задаёт вертикальную позицию окончания градиента. Это целое число в процентах от высоты элемента. По умолчанию 0.
* **opacity** - Устанавливает значение прозрачности в начале градиента. 0 — полная прозрачность, 100 — непрозрачность элемента
* **startx** - Задаёт горизонтальную начальную позицию градиента. Это целое число в процентах от ширины элемента. По умолчанию 0.
* **starty** - Задаёт вертикальную начальную позицию градиента. Это целое число в процентах от высоты элемента. По умолчанию 0.
* **style** - Стиль прозрачности. 0 — прозрачность применяется ко всему элементу. 1 — линейный градиент; начальная прозрачность указывается параметром opacity, конечная через finishOpacity, от начальной точки startX, startY к конечной точке finishX, finishY. 2 — радиальный градиент от центра к краю элемента. 3 — прямоугольный градиент, начинается с края элемента и идёт к его центру.

**basicimage** - Устанавливает параметры цвета, поворота изображения или прозрачности.

* **enabled** - Включает или выключает фильтр. Значение true разрешает использование фильтра, false запрещает.
* **grayscale** - Цвета превращаются в оттенки серого. 1 — серые цвета, 0 — отображать исходные цвета.
* **invert** - Инвертирует цвета элемента. 1 — инвертировать цвета, 0 — оставить исходные цвета.
* **mask** - Устанавливает, что определённые цвета должны быть прозрачными. 1 — прозрачность меняется с помощью параметра **maskcolor**, 0 — отображать исходные цвета.
* **masccolor** - Устанавливает цвет, используемый как маска прозрачности содержимого элемента. Цвет задаётся в формате 0xAARRGGBB, где **аа** — это шестнадцатеричное значение прозрачности, RR — красная компонента, GG — зелёная компонента, BB — синяя компонента.
* **mirror** - Зеркально отражает элемент. 1 — зеркально отобразить по горизонтали, 0 — оставить исходным.
* **opacity** - Устанавливает значение прозрачности. 0 — полная прозрачность, 1.0 — непрозрачность элемента.
* **rotation** - Вращает элемент. 0 — исходное положение, 1 — поворот на 90° по часовой стрелке, 2 — поворот на 180°, 3 — поворот на 270°.
* **xray** - Цвета превращаются в оттенки серого с инвертированием цветов.

**blur** - Размывает содержимое.

* **enabled** - Включает или выключает фильтр. Значение true разрешает использование фильтра, false запрещает.
* **makeshadow** - Отображать содержимое элемента как тень. Значение **true** показывает содержимое как тень, false — исходный вид.
* **pixelradius** - Радиус размытия от 1 до 100.
* **shadowopacity** - Устанавливает уровень прозрачности тени, сделанной параметром makeShadow. 0 — полная прозрачность, 1 — непрозрачность элемента.

**chroma** - Показывает определенные цвета прозрачными.

**dropshadow** - Отображает тень.

* **enabled** - Включает или выключает фильтр. Значение true разрешает использование фильтра, false запрещает.
* **color - цвет тени.**
* **offx** - Смещение тени по горизонтали. Положительное значение смещает тень вправо, отрицательное влево. По умолчанию равно 5.
* **offy** - Смещение тени по вертикали. Положительное значение смещает тень вниз, отрицательное вверх. По умолчанию равно 5.
* **positive** - Определяет, как должна отбрасываться тень. Значение true устанавливает тень от непрозрачных участков, false от прозрачных.

**emboss** - Показывает содержимое объекта в виде барельефа.

**engrave** - Показывает содержимое объекта в виде черно-белой гравюры.

**glow** - Добавляет свечение вокруг краев.

**gradient** - Создаёт линейный градиент.

**icmfilter** - Преобразует цвета содержимого на основе профиля системы управления цветом (Image Color Management, ICM).

**light** - Создает эффект лучей света.

**maskfilter** - Показывает прозрачные пикселы как цветную маску, а непрозрачные пикселы наоборот, прозрачными.

**matrix** - Изменяет размер, поворачивает или отражает объект на основе матричных преобразований.

**motionblur** - Размывает объект так, словно он быстро движется.

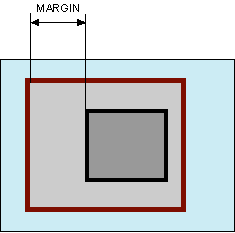
**shadow** - Добавляет тень.

**wave** - Вносит волнообразные искажения.

**brightness** – Яркость

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **margin** - Устанавливает величину отступа от каждого края элемента. Отступом является пространство от границы текущего элемента до внутренней границы его родительского элемента



Синтаксис:

margin: [значение | проценты | auto] {1,4} | inherit

Значения:

Разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом. Эффект зависит от количества значений и приведен в таблице.

|  |  |
| --- | --- |
| **Число значений** | **Результат** |
| 1 | Отступы будут установлены одновременно от каждого края элемента. |
| 2 | Первое значение устанавливает отступ от верхнего и нижнего края, второе — от левого и правого. |
| 3 | Первое значение задает отступ от верхнего края, второе — одновременно от левого и правого края, а третье — от нижнего края. |
| 4 | Поочередно устанавливается отступ от верхнего, правого, нижнего и левого края. |

**auto** - Указывает, что размер отступов будет автоматически рассчитан браузером.

**inherit** - Наследует значение родителя.

Пример 1:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**margin**</title>**

**<style>**

**body** {

margin: 0; /\* Убираем отступы \*/

}

.parent {

margin: 20%; /\* Отступы вокруг элемента \*/

background: #fd0; /\* Цвет фона \*/

padding: 10px; /\* Поля вокруг текста \*/

}

.child {

border: 3px solid #666; /\* Параметры рамки \*/

padding: 10px; /\* Поля вокруг текста \*/

margin: 10px; /\* Отступы вокруг \*/

}

**</style>**

**</head>**

**<body>**

**<div** class="parent"**>**

**<div** class="child"**>**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh

euismod tincidunt ut lacreet dolore magna aliguam erat volutpat. Ut wisis enim

ad minim veniam, quis nostrud exerci tution ullamcorper suscipit lobortis nisl

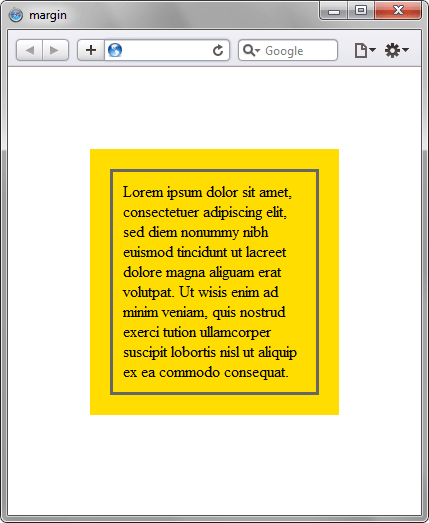
ut aliquip ex ea commodo consequat.

**</div>**

**</div>**

**</body>**

**</html>**



Пример 2:

<html>

<head>

<style>

.body2 {

display: block;

margin: 0 auto; **/\*Выравнивание блока по центру\*/**

height: 95%;

width: 70%;

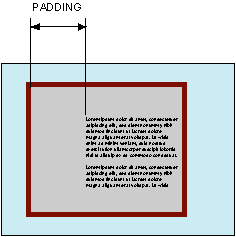
border: 5px dotted red;

}

</style>

</head>

1. **padding** - Устанавливает значение полей вокруг содержимого элемента. Полем называется расстояние от внутреннего края рамки элемента до воображаемого прямоугольника, ограничивающего его содержимое



Синтаксисы:

padding: [значение | проценты] {1, 4} | inherit

Значения:

Разрешается использовать одно, два, три или четыре значения, разделяя их между собой пробелом. Эффект зависит от количества значений и приведен в таблице.

|  |  |
| --- | --- |
| **исло значений** | **Результат** |
| 1 | Поля будут установлены одновременно от каждого края элемента. |
| 2 | Первое значение устанавливает поля от верхнего и нижнего края, второе — от левого и правого. |
| 3 | Первое значение задает поле от верхнего края, второе — одновременно от левого и правого края, а третье — от нижнего края. |
| 4 | Поочередно устанавливается поля от верхнего, правого, нижнего и левого края. |

**Пример:**

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**padding**</title>**

**<style>**

.layer {

background: #fc3; /\* Цвет фона \*/

border: 2px solid black; /\* Параметры рамки \*/

padding: 20px; /\* Поля вокруг текста \*/

}

**</style>**

**</head>**

**<body>**

**<div** class="layer"**>**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh

euismod tincidunt ut lacreet dolore magna aliguam erat volutpat. Ut wisis enim

ad minim veniam, quis nostrud exerci tution ullamcorper suscipit lobortis nisl

ut aliquip ex ea commodo consequat.

**</div>**

**</body>**

**</html>**

1. **display** - Многоцелевое свойство, которое определяет, как элемент должен быть показан в документе.

Синтаксис:

display: block | inline | inline-block | inline-table | list-item | none | run-in | table | table-caption | table-cell | table-column-group | table-column | table-footer-group | table-header-group | table-row | table-row-group

Значения:

Список возможных значений этого свойства, понимаемый разными браузерами очень короткий — block, inline, list-item и none. Все остальные допустимые значения поддерживаются браузерами выборочно. В таблице приведены некоторые популярные браузеры и их поддержка различных значений.

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **block** | Элемент показывается как блочный. Применение этого значения для встроенных элементов, например тега <span>, заставляет его вести подобно блокам — происходит перенос строк в начале и в конце содержимого. |
| **inline** | Элемент отображается как встроенный. Использование блочных тегов, таких как <div> и <p>, автоматически создает перенос и показывает содержимое этих тегов с новой строки. Значение inline отменяет эту особенность, поэтому содержимое блочных элементов начинается с того места, где окончился предыдущий элемент. |
| **inline-block** | Это значение генерирует блочный элемент, который обтекается другими элементами веб-страницы подобно встроенному элементу. Фактически такой элемент по своему действию похож на встраиваемые элементы (вроде тега <img>). При этом его внутренняя часть форматируется как блочный элемент, а сам элемент — как встроенный. |
| **inline-table** | Определяет, что элемент является таблицей как при использовании тега <table>, но при этом таблица является встроенным элементом и происходит ее обтекание другими элементами, например, текстом. |
| **list-item** | Элемент выводится как блочный и добавляется маркер списка. |
| **none** | Временно удаляет элемент из документа. Занимаемое им место не резервируется и веб-страница формируется так, словно элемента и не было. Изменить значение и сделать вновь видимым элемент можно с помощью скриптов, обращаясь к свойствам через объектную модель. В этом случае происходит переформатирование данных на странице с учетом вновь добавленного элемента. |
| **run-in** | Устанавливает элемент как блочный или встроенный в зависимости от контекста. |
| **table** | Определяет, что элемент является блочной таблицей подобно использованию тега <table>. |
| **table-caption** | Задает заголовок таблицы подобно применению тега <caption>. |
| **table-cell** | Указывает, что элемент представляет собой ячейку таблицы (тег <td> или <th>). |
| **table-column** | Назначает элемент колонкой таблицы, словно был добавлен тег <col>. |
| **table-column-group** | Определяет, что элемент является группой одной или более колонок таблицы, как при использовании тега <colgroup>. |
| **table-footer-group** | Используется для хранения одной или нескольких строк ячеек, которые отображаются в самом низу таблицы. По своему действию сходно с работой тега <tfoot>. |
| **table-header-group** | Элемент предназначен для хранения одной или нескольких строк ячеек, которые представлены вверху таблицы. По своему действию сходно с работой тега <thead>. |
| **table-row** | Элемент отображается как строка таблицы (тег <tr>). |
| **table-row-group** | Создает структурный блок, состоящий из нескольких строк таблицы аналогично действию тега <tbody>. |

Способы расположения элементов на web – странице

1. **float** **-** Определяет, по какой стороне будет выравниваться элемент, при этом остальные элементы будут обтекать его с других сторон. Когда значение свойства float равно none, элемент выводится на странице как обычно, при этом допускается, что одна строка обтекающего текста может быть на той же линии, что и сам элемент.

Синтаксис:

float: left | right | none | inherit

**Значения:**

**left** - Выравнивает элемент по левому краю, а все остальные элементы, вроде текста, обтекают его по правой стороне.

**right** - Выравнивает элемент по правому краю, а все остальные элементы обтекают его по левой стороне.

**none** - Обтекание элемента не задается.

**inherit** - Наследует значение родителя.

Пример:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**float**</title>**

**<style>**

.layer1 {

float: left; /\* Обтекание по правому краю \*/

background: #fd0; /\* Цвет фона \*/

border: 1px solid black; /\* Параметры рамки \*/

padding: 10px; /\* Поля вокруг текста \*/

margin-right: 20px; /\* Отступ справа \*/

width: 40%; /\* Ширина блока \*/

}

**</style>**

**</head>**

**<body>**

**<div** class="layer1"**>**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem nonummy nibh

euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.

**</div>**

**<div>**

Duis autem dolor in hendrerit in vulputate velit esse molestie consequat, vel

illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio

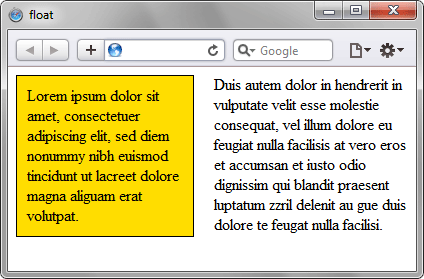
dignissim qui blandit praesent luptatum zzril delenit au gue duis dolore te

feugat nulla facilisi.

**</div>**

**</body>**

**</html>**



**2. display:flex**

1. **Cursor** - Устанавливает форму курсора, когда он находится в пределах элемента. Вид курсора зависит от операционной системы и установленных параметров.

Синтаксис:

cursor: [url('путь к курсору'),] | [ auto | crosshair | default | e-resize | help | move | n-resize | ne-resize | nw-resize | pointer | progress | s-resize | se-resize | sw-resize | text | w-resize | wait | inherit ]

**Значения:**

**url** - Позволяет установить свой собственный курсор, для этого нужно указать путь к файлу с курсором.

**auto** - Вид курсора по умолчанию для текущего элемента.

**inherit** - Наследует значение родителя.

|  |  |  |  |
| --- | --- | --- | --- |
| **Вид** | **Значение** | **Тест** | **Пример** |
| css_cursor_default | **default** |  | P {cursor: default} |
| css_cursor_crosshair | **crosshair** |  | P {cursor: crosshair} |
| css_cursor_help | **help** |  | P {cursor: help} |
| css_cursor_move | **move** |  | P {cursor: move} |
| css_cursor_pointer | **pointer** |  | P {cursor: pointer} |
| css_cursor_progress | **progress** |  | P {cursor: progress} |
| css_cursor_text | **text** |  | P {cursor: text} |
| css_cursor_wait | **wait** |  | P {cursor: wait} |
| css_cursor_nresize | **n-resize** |  | P {cursor: n-resize} |
| css_cursor_neresize | **ne-resize** |  | P {cursor: ne-resize} |
| css_cursor_eresize | **e-resize** |  | P {cursor :e-resize} |
| / | **se-resize** |  | P {cursor: se-resize} |
| css_cursor_sresize | **s-resize** |  | P {cursor: s-resize} |
| css_cursor_swresize | **sw-resize** |  | P {cursor: sw-resize} |
| css_cursor_wresize | **w-resize** |  | P {cursor: w-resize} |
| css_cursor_nwresize | **nw-resize** |  | P {cursor :nw-resize} |

1. **user-select -** Управляет поведением выделения текста и других элементов на странице. Несмотря на запрет или разрешение, в любом случае можно выделять содержимое текстовых полей формы.

Синтаксис:

user-select: auto | none | text

**Значение:**

**auto** - Пользователь может выделить элемент.

**none** - Пользователю запрещено выделять элемент.

**text** - Пользователь может выделить текст в элементе.

1. **z-index** - Любые позиционированные элементы на веб-странице могут накладываться друг на друга в определенном порядке, имитируя тем самым третье измерение, перпендикулярное экрану.

Каждый элемент может находиться как ниже, так и выше других объектов веб-страницы, их размещением по z-оси и управляет z-index. Это свойство работает только для элементов, у которых значение [position](#s100) задано как **absolute**, **fixed** или **relative**.

Синтаксис:

z-index: число | auto | inherit

В качестве значения используются целые числа (положительные, отрицательные и ноль). Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше. При равном значении z-index, на переднем плане находится тот элемент, который в коде HTML описан ниже. Хотя спецификация и разрешает использовать отрицательные значения z-index, но такие элементы не отображаются в браузере Firefox до версии 2.0 включительно.

Кроме числовых значений применяется **auto** — порядок элементов в этом случае строится автоматически, исходя из их положения в коде HTML и принадлежности к родителю, поскольку дочерние элементы имеют тот же номер, что их родительский элемент. Значение **inherit** указывает, что оно наследуется у родителя.

**Пример:**

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**z-index**</title>**

**<style>**

#layer1, #layer2, #layer3, #layer4 {

position: relative; /\* Относительное позиционирование \*/

}

#layer1, #layer3 {

font-size: 50px; /\* Размер шрифта в пикселах \*/

color: #000080; /\* Синий цвет текста \*/

}

#layer2, #layer4 {

top: -55px; /\* Сдвигаем текст вверх \*/

left: 5px; /\* Сдвигаем текст вправо \*/

color: #ffa500; /\* Оранжевый цвет текста \*/

font-size:70px; /\* Размер шрифта в пикселах \*/

}

#layer1 { z-index: 2; }

#layer2 { z-index: 1; }

#layer3 { z-index: 3; }

#layer4 { z-index: 4; }

**</style>**

**</head>**

**<body>**

**<p>**Слой 1 наверху**</p>**

**<div** id="layer1"**>**Слой 1**</div>**

**<div** id="layer2"**>**Слой 2**</div>**

**<p>**Слой 4 наверху**</p>**

**<div** id="layer3"**>**Слой 3**</div>**

**<div** id="layer4"**>**Слой 4**</div>**

**</body>**

**</html>**

1. **text-shadow** - Добавляет тень к тексту, а также устанавливает её параметры: цвет тени, смещение относительно надписи и радиус размытия. Свойство text-shadow может работать совместно с псевдоэлементами: **first-letter** и: **first-line**.

Синтаксис:

text-shadow: none | тень [,тень]\*  
где тень:  
<сдвиг по x> <сдвиг по y> <радиус размытия> <цвет>

**Значения:**

**none** - Отменяет добавление тени.

**Цвет** - Цвет тени в любом доступном CSS формате. По умолчанию цвет тени совпадает с цветом текста. Необязательный параметр.

**Сдвиг по x** - Смещение тени по горизонтали относительно текста. Положительное значение этого параметра задает сдвиг тени вправо, отрицательное — влево. Обязательный параметр.

**Сдвиг по y** - Смещение тени по вертикали относительно текста. Также допустимо использовать отрицательное значение, которое поднимает тень выше текста. Обязательный параметр.

**Радиус** - Задает радиус размытия тени. Чем больше это значение, тем сильнее тень сглаживается, становится шире и светлее. Если этот параметр не задан, по умолчанию устанавливается равным 0. Учтите, что алгоритм сглаживания в браузерах обычно разный, поэтому вид тени может несколько различаться в зависимости от заданных параметров сглаживания.

Допускается указывать несколько параметров тени, разделяя их между собой запятой. В CSS3 учитывается следующий порядок: первая тень в списке размещается на самом верху, последняя в списке — в самом низу. В CSS2 порядок наоборот: первая тень размещается в самом низу, а последняя на самом верху.

Пример:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**text-shadow**</title>**

**<style>**

.shadowtext {

text-shadow: 1px 1px 2px black, 0 0 1em red; /\* Параметры тени \*/

color: white; /\* Белый цвет текста \*/

font-size: 2em; /\* Размер надписи \*/

}

**</style>**

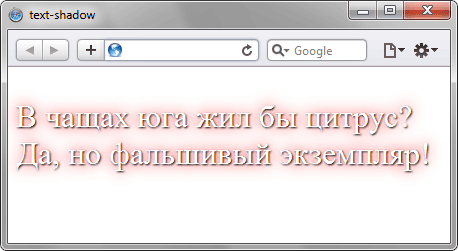
**</head>**

**<body>**

**<p** class="shadowtext"**>**В чащах юга жил бы цитрус? Да, но фальшивый экземпляр!**</p>**

**</body>**

**</html>**



1. **box-shadow** - Добавляет тень к элементу. Допускается использовать несколько теней, указывая их параметры через запятую, при наложении теней первая тень в списке будет выше, последняя ниже. Если для элемента задается радиус скругления через свойство [**border-radius**](#s24), то тень также получится с закругленными уголками. Добавление тени увеличивает ширину элемента, поэтому возможно появление горизонтальной полосы прокрутки в браузере.

**Синтаксис:**

box-shadow: none | <тень> [,<тень>]\*  
где <тень>:  
inset <сдвиг по x> <сдвиг по y> <радиус размытия> <растяжение> <цвет>

**Значения:**

**none** - Отменяет добавление тени.

**inset** - Тень выводится внутри элемента. Необязательный параметр.

**Сдвиг по x** - Смещение тени по горизонтали относительно элемента. Положительное значение этого параметра задает сдвиг тени вправо, отрицательное — влево. Обязательный параметр.

**Сдвиг по y** - Смещение тени по вертикали относительно элемента. Положительное значение задает сдвиг тени вниз, отрицательное — вверх. Обязательный параметр.

**Размытие** - Задает радиус размытия тени. Чем больше это значение, тем сильнее тень сглаживается, становится шире и светлее. Если этот параметр не задан, по умолчанию устанавливается равным 0, тень при этом будет четкой, а не размытой.

**Растяжение** - Положительное значение растягивает тень, отрицательное, наоборот, ее сжимает. Если этот параметр не задан, по умолчанию устанавливается 0, при этом тень будет того же размера, что и элемент.

**Цвет** - Цвет тени в любом доступном CSS формате, по умолчанию тень черная. Необязательный параметр.

Допускается указывать несколько теней, разделяя их параметры между собой запятой. Учитывается следующий порядок: первая тень в списке размещается на самом верху, последняя в списке — в самом низу.

Пример:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**box-shadow**</title>**

**<style>**

.shadow {

background: #fc0; /\* Цвет фона \*/

box-shadow: 0 0 10px **rgba(**0,0,0,0.5**)**; /\* Параметры тени \*/

padding: 10px;

}

**</style>**

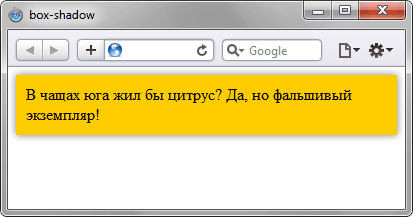
**</head>**

**<body>**

**<div** class="shadow"**>**В чащах юга жил бы цитрус? Да, но фальшивый экземпляр!**</div>**

**</body>**

**</html>**



1. **box-sizing** - Применяется для изменения алгоритма расчета ширины и высоты элемента.

Согласно спецификации, CSS ширина блока складывается из ширины контента (width), значений отступов (margin), полей (padding) и границ (border). Аналогично обстоит и с высотой блока. Свойство **box-sizing** позволяет изменить этот алгоритм, чтобы свойства **width** и **height** задавали размеры не контента, а размеры блока.

**Синтаксис:**

box-sizing: content-box | border-box | padding-box | inherit

Значение:

**content-box** - Основывается на стандартах CSS, при этом свойства width и height задают ширину и высоту контента и не включают в себя значения отступов, полей и границ.

**border-box** - Свойства width и height включают в себя значения полей и границ, но не отступов (margin). Эта модель используется браузером Internet Exporer в режиме несовместимости.

**padding-box** - Свойства width и height включают в себя значения полей, но не отступов (margin) и границ (border).

**inherit** - Наследует значение родителя.

Пример:

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**box-sizing**</title>**

**<style>**

.box1 {

background: #f0f0f0; /\* Цвет фона \*/

width: 300px; /\* Ширина блока \*/

padding: 10px; /\* Поля \*/

border: 2px solid #000; /\* Параметры рамки \*/

}

.box2 {

background: #fc0; /\* Цвет фона \*/

width: 300px; /\* Ширина блока \*/

padding: 10px; /\* Поля \*/

margin-top: 10px; /\* Отступ сверху \*/

border: 2px solid #000; /\* Параметры рамки \*/

-moz-box-sizing: border-box; /\* Для Firefox \*/

box-sizing: border-box; /\* Ширина блока с полями \*/

}

**</style>**

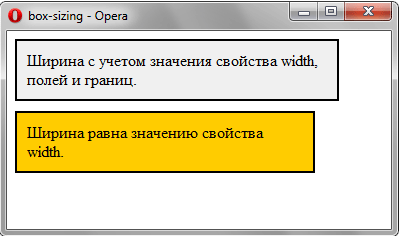
**</head>**

**<body>**

**<div** class="box1"**>**Ширина с учетом значения свойства width, полей и границ.**</div>**

**<div** class="box2"**>**Ширина равна значению свойства width.**</div>**

**</body>**



Подключение иконок типа svg в меню и не только

1. **Подключение через тег <img> в html**

<h2>Иконки</h2>

<img src="./img/calendar.svg" alt="Calendar" width="102">

1. **Подключение фона в .css**

**<h2>Фоновые изображения</h2>**

**<div class="block-bg"></div>**

**.block-bg {**

**width: 600px;**

**height: 200px;**

**background-image: url('./../img/bg-zigzag.svg');**

**}**

1. **Подключение svg – графики inline**

Для подключения необходимо скачать svg иконку, открыть файл в текстовом документе, скопировать и вставить в html код. Так же в коде необходимо указать параметры высоты и ширины для изменения размера иконки если этих параметров нет в css - свойствах.

**<svg class="youtube-icon" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 576 512">**

**<path d="M549.655 124.083c-6.281-23.65-24.787-42.276-48.284-48.597C458.781 64 288 64 288 64S117.22 64 74.629 75.486c-23.497 6.322-42.003 24.947-48.284 48.597-11.412 42.867-11.412 132.305-11.412 132.305s0 89.438 11.412 132.305c6.281 23.65 24.787 41.5 48.284 47.821C117.22 448 288 448 288 448s170.78 0 213.371-11.486c23.497-6.321 42.003-24.171 48.284-47.821 11.412-42.867 11.412-132.305 11.412-132.305s0-89.438-11.412-132.305zm-317.51 213.508V175.185l142.739 81.205-142.739 81.201z"></path>**

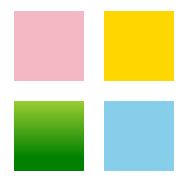
</svg>

Заливка и обводка SVG иконок

**fill** - Заливка элемента.

**Значения:**

* **none**
* **ключевые слова**
* **цвета**
* **градиенты**



**fill-opacity** - Управление прозрачностью заливки. Можно задавать значения от 0.0 до 1.0 или в процентах.

Пример:

<!-- ... -->

<use xlink:href="#s-rect"

x="20" y="20"

fill="url(#g-red)" fill-opacity=".5"/>

<use xlink:href="#s-rect"

x="50" y="50"

fill="hsl(50, 100%, 50%)" fill-opacity=".5"/>

<use xlink:href="#s-rect"

x="80" y="80"

fill="url(#g-green)" fill-opacity=".5"/>

<!-- ... -->



**stroke** - Цвет обводки. Значения по умолчанию нет.

**Значения:**

* **ключевые слова**
* **цвета**
* **градиенты**

Пример:

<svg width="200" height="200">

<rect width="160" height="160" x="20" y="20"

fill="none"

stroke="yellowgreen"/>

</svg>



**stroke-width -** Толщина обводки, можно задавать в единицах длины или в процентах. Значение по умолчанию: 1.

Пример:

<svg width="200" height="200">

<rect width="100" height="100" x="50" y="50"

fill="none" stroke="gold"

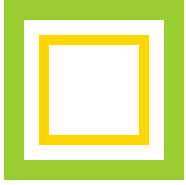
stroke-width="10"/>

<rect width="160" height="160" x="20" y="20"

fill="none" stroke="yellowgreen"

stroke-width="10%"/>

</svg>



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Создание меню/списков

1. **list-style-type** - Изменяет вид маркера для каждого элемента списка. Это свойство используется только в случае, когда значение list-style-image установлено как none. Маркеры различаются для маркированного списка (тег <**ul**>) и нумерованного (тег <**ol**>).

Синтаксис:

ist-style-type: circle | disc | square | armenian | decimal | decimal-leading-zero | georgian | lower-alpha | lower-greek | lower-latin | lower-roman | upper-alpha | upper-latin | upper-roman | none | inherit

**Значения**

Значения зависят от того, к какому типу списка они применяются: маркированному или нумерованному.

**Маркер в виде картинки**

ul { list-style-image: **url("li.gif")**; }

**Маркированный список**

**circle** - Маркер в виде кружка.

**disc** - Маркер в виде точки.

**square** - Маркер в виде квадрата.

**Нумерованный список**

**armenian** - Традиционная армянская нумерация.

**decimal** - Арабские числа (1, 2, 3, 4,...).

**decimal-leading-zero** - Арабские числа с нулем впереди для цифр меньше десяти (01, 02, 03,...).

**georgian** - Традиционная грузинская нумерация.

**lower-alpha** - Строчные латинские буквы (a, b, c, d,...).

**lower-greek** - Строчные греческие буквы (α, β, γ, δ,...).

**lower-latin** - Это значение аналогично lower-alpha.

**lower-roman** - Римские числа в нижнем регистре (i, ii, iii, iv, v,...).

**upper-alpha** - Заглавные латинские буквы (A, B, C, D,...).

**upper-latin** - Это значение аналогично upper-alpha.

**upper-roman** - Римские числа в верхнем регистре (I, II, III, IV, V,...).

**none** - Отменяет маркеры для списка.

**inherit** - Наследует значение родителя.

Пример:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**list-style-type**</title>**

**<style>**

**ol** {

list-style-type: upper-alpha; /\* Заглавные буквы \*/

}

**</style>**

**</head>**

**<body>**

**<ol>**

**<li>**Lorem ipsum dolor sit amet**</li>**

**<li>**Consectetuer adipiscing elit**</li>**

**<li>**Sed diem nonummy nibh euismod**</li>**

**<li>**Tincidunt ut lacreet dolore magna aliguam erat volutpat. Ut wisis

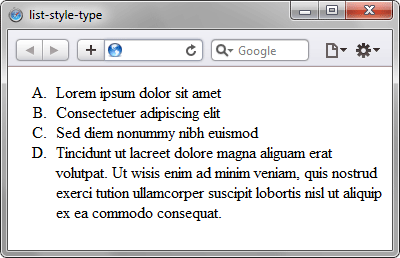
enim ad minim veniam, quis nostrud exerci tution ullamcorper suscipit lobortis

nisl ut aliquip ex ea commodo consequat.**</li>**

**</ol>**

**</body>**

**</html>**



1. **list-style-position** - Определяет, как будет размещаться маркер относительно текста

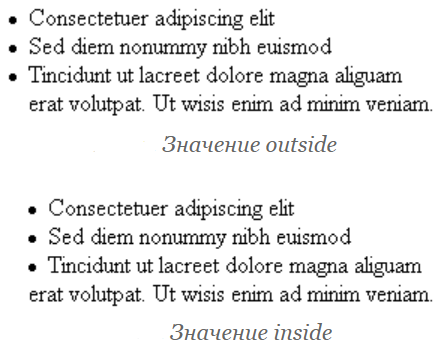
**Синтаксис**

list-style-position: inside | outside

**Значения**

**inside** - Маркер является частью текстового блока и отображается в элементе списка

**outside** - Текст выравнивается по левому краю, а маркеры размещаются за пределами текстового блока



Пример:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**list-style-position**</title>**

**<style>**

**ul** {

list-style-image: **url(**images/book.gif**)**; /\* Путь к рисунку для установки маркера \*/

list-style-position: inside; /\* Маркер обтекается текстом \*/

}

**</style>**

**</head>**

**<body>**

**<ul>**

**<li>**Lorem ipsum dolor sit amet**</li>**

**<li>**Consectetuer adipiscing elit**</li>**

**<li>**Sed diem nonummy nibh euismod**</li>**

**<li>**Tincidunt ut lacreet dolore magna aliguam erat volutpat. Ut wisis

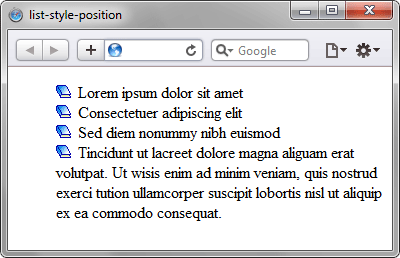
enim ad minim veniam, quis nostrud exerci tution ullamcorper suscipit lobortis

nisl ut aliquip ex ea commodo consequat.**</li>**

**</ul>**

**</body>**

**</html>**



**Бесплатные иконки для меню + подключение к сайту:** <https://icons.getbootstrap.com/> **или** <https://fontawesome.com/v5.15/icons?d=gallery&p=2>

**Пример 1. Создание меню:**

<!—Подключение библиотеки с иконками -->  
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">  
  
<div class="icon-bar">  
  <a class="active" href="#"><i class="fa fa-home"></i></a>  
  <a href="#"><i class="fa fa-search"></i></a>  
  <a href="#"><i class="fa fa-envelope"></i></a>  
  <a href="#"><i class="fa fa-globe"></i></a>  
  <a href="#"><i class="fa fa-trash"></i></a>  
</div>

**CSS пример по вертикали:**

.icon-bar {  
  width: 90px; /\* Установите определенную ширину \*/  
  background-color: #555; /\* Темно-серый фон \*/  
}  
  
.icon-bar a {  
  display: block; /\* Сделайте так, чтобы ссылки отображались друг под другом, а не рядом друг с другом \*/  
  text-align: center; /\* Выровнять текст по центру \*/  
  padding: 16px; /\* Добавьте отступ \*/  
  transition: all 0.3s ease; /\* Добавить переход для эффектов наведения \*/  
  color: white; /\* Белый цвет текста \*/  
  font-size: 36px; /\* Увеличенный размер шрифта \*/  
}  
  
.icon-bar a:hover {  
  background-color: #000; /\* Добавьте цвет при наведении \*/  
}  
  
.active {  
  background-color: #4CAF50; /\* Добавить активный / текущий цвет \*/  
}



**CSS пример по горизонтали:**

.icon-bar {  
  width: 100%; /\* На всю ширину \*/  
  background-color: #555; /\* Темно-серый фон \*/  
  overflow: auto; /\* Переполнение из-за float \*/  
}  
  
.icon-bar a {  
  float: left; /\* Плавающие ссылки рядом \*/  
  text-align: center; /\* Выровнять текст по центру \*/  
  width: 20%; /\* Равная ширина (5 иконок шириной 20% каждая = 100%) \*/  
  padding: 12px 0; /\* Некоторые верхние и нижние отступы \*/  
  transition: all 0.3s ease; /\* Добавить переход для эффектов наведения \*/  
  color: white; /\* Белый цвет текста \*/  
  font-size: 36px; /\* Увеличенный размер шрифта \*/  
}  
  
.icon-bar a:hover {  
  background-color: #000; /\* Добавьте цвет при наведении \*/  
}  
  
.active {  
  background-color: #4CAF50; /\* Добавить активный / текущий цвет \*/  
}



**Пример 2. Создание меню:**

<ul id="navbar">

  <li><a href="#">Главная</a></li>

  <li><a href="#">Новости</a></li>

  <li><a href="#">Контакты</a></li>

  <li><a href="#">О нас</a></li>

</ul>

**CSS:**

#navbar {

  margin: 0;

  padding: 0;

  list-style-type: none;

  border: 2px solid #0066FF;

  border-radius: 20px 5px;

  width: 550px;

  text-align: center;

  background-color: #33ADFF;

}

#navbar a {

  color: #fff;

  padding: 5px 10px;

  text-decoration: none;

  font-weight: bold;

  display: inline-block;

  width: 100px;

}

#navbar a:hover {

  border-radius: 20px 5px;

  background-color: #0066FF;

}

1. **list-style-image** - Устанавливает адрес изображения, которое служит в качестве маркера списка. Это свойство наследуется, поэтому для отдельных элементов списка для восстановления маркера используется значение **none**.

Синтаксис:

list-style-image: none | url('путь к файлу') | inherit

**Значения**

**none** - Отменяет изображение в качестве маркера для родительского элемента.

**url** - Относительный или абсолютный путь к графическому файлу. Значение можно указывать в одинарных, двойных кавычках или без них.

**inherit** - Наследует значение родителя.

Пример:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**list-style-image**</title>**

**<style>**

**ul** {

list-style-image: **url(**images/book.gif**)**;

}

**</style>**

**</head>**

**<body>**

**<ul>**

**<li>**Lorem ipsum dolor sit amet**</li>**

**<li>**Consectetuer adipiscing elit**</li>**

**<li>**Sed diem nonummy nibh euismod**</li>**

**<li>**Tincidunt ut lacreet dolore magna aliguam erat volutpat. Ut wisis

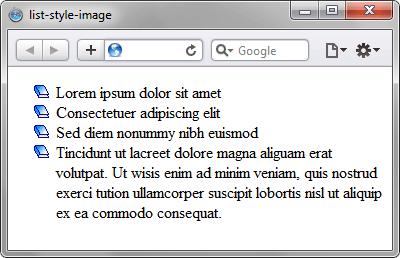
enim ad minim veniam, quis nostrud exerci tution ullamcorper suscipit lobortis

nisl ut aliquip ex ea commodo consequat.**</li>**

**</ul>**

**</body>**

**</html>**



Управление скроллбаром

1. **overflow** - управляет отображением содержания блочного элемента, если оно целиком не помещается и выходит за область заданных размеров.

Синтаксис:

overflow: auto | hidden | scroll | visible | inherit

**Значения:**

**visible** - Отображается все содержание элемента, даже за пределами установленной высоты и ширины.

**hidden** - Отображается только область внутри элемента, остальное будет скрыто.

**scroll** - Всегда добавляются полосы прокрутки.

**auto** - Полосы прокрутки добавляются только при необходимости.

**inherit** - Наследует значение родителя.

**Пример:**

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**overflow**</title>**

**<style>**

.layer {

overflow: scroll; /\* Добавляем полосы прокрутки \*/

width: 300px; /\* Ширина блока \*/

height: 150px; /\* Высота блока \*/

padding: 5px; /\* Поля вокруг текста \*/

border: solid 1px black; /\* Параметры рамки \*/

}

**</style>**

**</head>**

**<body>**

**<div** class="layer"**>**

**<h2>**Duis te feugifacilisi**</h2>**

**<p>**Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diem

nonummy nibh euismod tincidunt ut lacreet dolore magna aliguam erat volutpat.

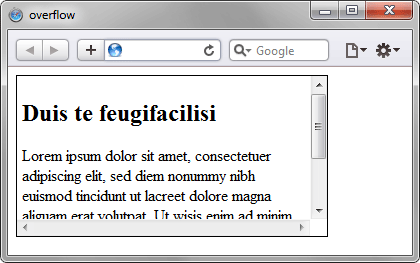
Ut wisis enim ad minim veniam, quis nostrud exerci tution ullamcorper suscipit

lobortis nisl ut aliquip ex ea commodo consequat.**</p>**

**</div>**

**</body>**

**</html>**



**Стилизация скролла для Mozilla Firefox**

Пример:

html, body {

scrollbar-color: #458245 #714826; /\* «цвет ползунка» «цвет полосы скроллбара» \*/

scrollbar-width: **auto** | **thin** | **none**; /\* толщина \*/

}

**auto** - толщина скролла по умолчанию;

**thin** - более тонкий вариант скролла;

**none** - скрыть скролл.

**Стилизация скролла для webkit браузеров (Chrome, Яндекс, Safari и Opera)**

***/\* полоса прокрутки (скроллбар) \*/***

::-webkit-scrollbar {

width: 24px; /\* ширина для вертикального скролла \*/

height: 8px; /\* высота для горизонтального скролла \*/

background-color: #143861;

}

***/\* ползунок скроллбара \*/***

::-webkit-scrollbar-thumb {

background-color: #843465;

border-radius: 9em;

box-shadow: inset 1px 1px 10px #f3faf7;

}

::-webkit-scrollbar-thumb:hover {

background-color: #253861;

}

***/\* Стрелки \*/***

::-webkit-scrollbar-button:vertical:start:decrement {

background: linear-gradient(120deg, #02141a 40%, rgba(0, 0, 0, 0) 41%),

linear-gradient(240deg, #02141a 40%, rgba(0, 0, 0, 0) 41%),

linear-gradient(0deg, #02141a 30%, rgba(0, 0, 0, 0) 31%);

background-color: #f6f8f4;

}

::-webkit-scrollbar-button:vertical:end:increment {

background:

linear-gradient(300deg, #02141a 40%, rgba(0, 0, 0, 0) 41%),

linear-gradient(60deg, #02141a 40%, rgba(0, 0, 0, 0) 41%),

linear-gradient(180deg, #02141a 30%, rgba(0, 0, 0, 0) 31%);

background-color: #f6f8f4;

}

::-webkit-scrollbar-button:horizontal:start:decrement {

background:

linear-gradient(30deg, #02141a 40%, rgba(0, 0, 0, 0) 41%),

linear-gradient(150deg, #02141a 40%, rgba(0, 0, 0, 0) 41%),

linear-gradient(270deg, #02141a 30%, rgba(0, 0, 0, 0) 31%);

background-color: #f6f8f4;

}

::-webkit-scrollbar-button:horizontal:end:increment {

background:

linear-gradient(210deg, #02141a 40%, rgba(0, 0, 0, 0) 41%),

linear-gradient(330deg, #02141a 40%, rgba(0, 0, 0, 0) 41%),

linear-gradient(90deg, #02141a 30%, rgba(0, 0, 0, 0) 31%);

background-color: #f6f8f4;

}

**Скрыть скролл**

.el {

overflow:auto; **/\*Значение может быть другим\*/**

scrollbar-width: none; **/\* firefox \*/**

}

.el::-webkit-scrollbar { **/\* webkit \*/**

width: 0;

height: 0;

}

1. **scroll snap** – позволяет прокручивать не попиксельно, а целыми блоками.

**Горизонтальное прокручивание**

<html>

<head>

**.parent-container** {

**display: flex;**

**overflow-x: auto;**

**scroll-snap-type: x** **proximity;**

}

.children {

scroll-snap-align: start;

}

</head>

<body>

<div class="**parent-container**">

<div class="children"></div>

<div class="children"></div>

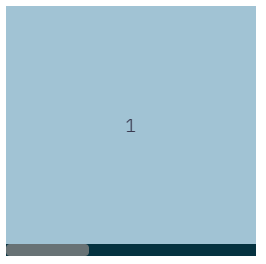
<div class="children"></div>

<div class="children"></div>

</div>

</body>

</html>



**Вертикальное прокручивание**

<html>

<head>

**.parent-container** {

**display: flex;**

**overflow-x: auto;**

**scroll-snap-type: x proximity;**

}

.children {

scroll-snap-align: start; **/\* start/center/end \*/**

}

</head>

<body>

<div class="**parent-container**">

<div class="children"></div>

<div class="children"></div>

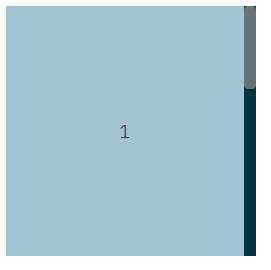
<div class="children"></div>

<div class="children"></div>

</div>

</body>

</html>



# Работа с Input через CSS

## Изменение цвета текста placeholder

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <style>

        input::placeholder {

            color: #000000;

        }

        input::-webkit-input-placeholder {

            color: #000000;

        }

        input::-ms-input-placeholder {

            color: #000000;

        }

        input:-moz-placeholder {

            color: #000000;

            opacity: 1;

        }

        input::-moz-placeholder {

            color: #000000;

            opacity: 1;

        }

        input:-ms-input-placeholder {

            color: #000000;

        }

    </style>

</head>

<body>

    <input type="search">

</body>

</html>

## Удаление крестика (очистка содержимого) из поля input type=”search”

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <style>

        input[type=text]::-ms-clear {

            display: none;

            width: 0;

            height: 0;

        }

        input[type=text]::-ms-reveal {

            display: none;

            width: 0;

            height: 0;

        }

        input[type="search"]::-webkit-search-decoration,

        input[type="search"]::-webkit-search-cancel-button,

        input[type="search"]::-webkit-search-results-button,

        input[type="search"]::-webkit-search-results-decoration {

            display: none;

        }

    </style>

</head>

<body>

    <input type="search">

</body>

</html>

Появление блока при наведении

1 Способ

HTML

<div class="holder">

<div class="block">

<h2>Пример</h2>

Пример блока, при наведении на который появляется другой блок.

</div>

</div>

CSS

.holder{

    position:relative;

    width:300px;

    height:400px;

    background:#ff6600;

     margin:100px auto;

}

.block{

    position:absolute;

    left:0;

    bottom:0;

    right:0;

    top:0;

    background:rgba(255,255,255, 0.7);

    padding:5px;

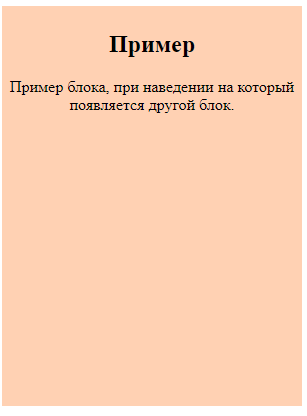
    display:none;

}

.holder:hover .block{

    display:block;

}



2 Способ

HTML

<div class="poster">

<div class="descr">

Какой-то текст который должен быть в блоке<br>

<a href="#">Интересная ссылка</a>

</div>

</div>

CSS

.poster{

    position:relative;

    margin:100px auto;

    background:#ff6600;

    height:200px;

    width:150px;

}

.descr{

    display:none;

    margin-left:-350px;

    padding:10px;

    margin-top:17px;

    background:#f3f3f3;

    height:200px;

    -moz-box-shadow:0 5px 5px rgba(0,0,0,0.3);

    -webkit-box-shadow:0 5px 5px rgba(0,0,0,0.3);

    box-shadow:0 5px 5px rgba(0,0,0,0.3);

}

.poster:hover .descr{

    display:block;

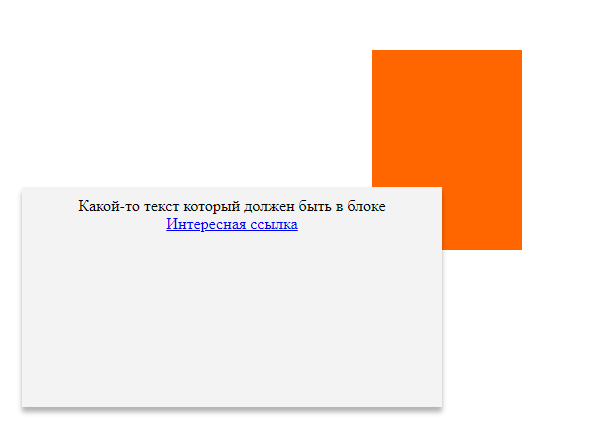
    position:absolute;

    top:120px;

    z-index:9999;

    width:400px;

}



Верстка сайта

1. **Адаптивная верстка**

Адаптивная верстка – метод верстки web документа, при изменении которого на определенных позициях (точках) меняется вид и расположения блоков страницы.

*Этапы:*

## Указание свойства адаптивности с помощью параметра meta

**<meta name="viewport" content="width=device-width, initial-scale=1">** После указания данного свойства, ширина страницы будет равняться ширине устройства, на котором просматривается страница (то есть будет передано свойство об адаптивности страницы)

**Атрибуты для meta name=”viewport” (пишутся внутри content=” …”):**

* **width** - определяет ширину области просмотра в пикселях (от 200 до 10000) или равной ширине страницы в соответствии с размером экрана (константа **device-width**).

Если значение не задано, по умолчанию устанавливается – в мобильном Safari = 980px, Opera = 850px, Android WebKit = 800px, IE = 974px.

* **height** - определяет высоту области просмотра в пикселях (от 233 до 10000 пикселей) или равной высоте страницы в соответствии с размером экрана (константа device-height)
* **initial-scale** - определяет начальный масштаб страницы в единицах от 0.1 до 1.0
* **user-scalable** - определяет доступность масштабирования страницы пользователем
* **minimum-scale** - определяет минимальный масштаб области просмотра в единицах от 0.1 до 1.0
* **maximum-scale** - определяет максимальный масштаб области просмотра в единицах от 0.1 до 1.0

**Мета-теги для Apple устройств -** Устанавливает, будет ли веб-приложение работать в полноэкранном режиме:

**<meta name="apple-mobile-web-app-capacity" content="yes">**

## Использование CSS-свойства @media вместе с правилами адаптивной верстки

* позволяет указать тип носителя, для которого будет применяться указанный стиль. В качестве типов выступают различные устройства, например, принтер, КПК, монитор и др. В таблице ниже указаны некоторые типы:

|  |  |
| --- | --- |
| all | Все типы. Это значение используется по умолчанию. |
| braille | Устройства, основанные на системе Брайля, которые предназначены для чтения слепыми людьми. |
| embossed | Принтеры, использующие для печати систему Брайля. |
| handheld | Смартфоны и аналогичные им аппараты. |
| print | Принтеры и другие печатающие устройства. |
| projection | Проекторы. |
| screen | Экран монитора. |
| speech | Речевые синтезаторы, а также программы для воспроизведения текста вслух. Сюда, например, можно отнести речевые браузеры. |
| tty | Устройства с фиксированным размером символов (телетайпы, терминалы, устройства с ограничениями дисплея). |
| tv | Телевизоры. |

**Синтаксис:**

@media тип1 [, тип2] {

Описание стиля

}

**Медиа запросы. Логические операторы.**

* **and** -Логическое И. Указывается для объединения нескольких условий.

Пример:

@media all and (color) { ... }

* **not** - Логическое НЕ. Указывается для отрицания условия.

Пример. Стиль для всех устройств кроме смартфонов

@media all and (not handheld) { ... }

Оператор not имеет низкий приоритет и оценивается в запросе последним. Применяется для старых браузеров, которые не поддерживают медиа-запросы.

* **only** - Применяется для старых браузеров, которые не поддерживают медиа-запросы.

Пример:

@media only all and (not handheld) { ... }

В списке нет логического оператора ИЛИ, его роль выполняет запятая. Перечисление нескольких условий через запятую говорит о том, что если хотя бы одно условие выполняется, то стиль будет применён.

Пример. Стиль для устройств с альбомной ориентацией или минимальной шириной 480 пикселов.

@media all and (orientation: landscape), all and (min-width: 480px) { ... }

**Медиа функции:**

1. **aspect-ratio (min-aspect-ratio, max-aspect-ratio)** - Определяет соотношение ширины и высоты отображаемой области устройства. Значение указывается в виде двух целых чисел разделяемых между собой слэшем (/).

* Тип носителя: ***handheld, print, projection, screen, tty, tv***
* Значение: ***целое число/целое число***

1. **color (min-color, max-color) -** Определяет число бит на канал цвета. К примеру, значение 3 означает, что красный, зелёный и синий канал могут отображать цветов каждый, что в общем составляет 512 цветов (8×8×8). Если значение не указано, тогда проверяется что устройство цветное.

* Тип носителя: ***handheld, print, projection, screen, tty, tv***
* Значение: ***целое число***

Пример:

@media screen and (color) { /\* Для цветных экранов \*/

body { background: #fc0; }

}

@media screen and (min-color:3) { /\* Минимум 512 цветов \*/

body { background: #ccc; }

}

1. **monochrome (min-monochrome, max-monochrome)** - Определяет, что устройство монохромное. Если указано число, то оно обозначает число бит на пиксел. Так, значение 8 равнозначно 256 оттенкам серого (или другого цвета).

* Тип носителя: ***handheld, print, projection, screen, tty, tv***
* Значение: ***целое число***

Пример (В примере показан стиль для монохромного и цветного принтера):

@media print and (monochrome) {

body { font-family: Times, 'Times New Roman', serif; }  
 h1, h2, p { color: black; }

}

@media print and (color) {

body { font-family: Arial, Verdana, sans-serif; }  
 h1, h2, p { color: #556b2f; }

}

1. **device-aspect-ratio (min-device-aspect-ratio, max-device-aspect-ratio)** - Определяет соотношение сторон экрана устройства. Значение указывается в виде двух целых чисел, разделяемых между собой слэшем (/).

* Тип носителя: ***handheld, print, projection, screen, tty, tv***
* Значение: ***целое число/целое число***

Пример (в примере показано, как установить стиль для экранов с соотношением сторон 16:9 и более):

@media screen and (min-device-aspect-ratio: 16/9) {

...

}

1. **device-height (min-device-height, max-device-height)** - Определяет всю доступную высоту экрана устройства или печатной страницы.

* Тип носителя: ***все кроме speech***
* Значение: ***размер***

1. **device-width (min-device-width, max-device-width)** - Определяет всю доступную ширину экрана устройства или печатной страницы.

* Тип носителя: ***все кроме speech***
* Значение: ***размер***

Пример (в зависимости от разрешения монитора устанавливается ширина слоя. Так, для значения 1280 пикселов ширина макета задаётся как 1100px.):

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Ширина макета**</title>**

**<style>**

div {

padding: 10px;

background: #e8bfad;

margin: auto;

}

@media screen and (min-device-width: 1600px) {

div {width: 1500px;}

}

@media screen and (device-width: 1280px) {

div {width: 1100px;}

}

@media screen and (device-width: 1024px) {

div {width: 980px;}

}

**</style>**

**</head>**

**<body>**

**<div>**

Диабаз, формируя аномальные геохимические ряды, сменяет известняк,

образуя на границе с Западно-Карельским поднятием своеобразную систему грабенов.

**</div>**

**</body>**

**</html>**

1. **height (min-height, max-height)** - Высота отображаемой области.

* Тип носителя: ***все кроме speech***
* Значение: ***размер***

1. **width (min-width, max-width)** - Описывает ширину отображаемой области.

* Тип носителя: ***все кроме speech***
* Значение: ***размер***

Пример (в примере при уменьшении окна до 600 пикселов и меньше меняется цвет фона веб-страницы):

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** charset="utf-8"**>**

**<title>**Ширина страницы**</title>**

**<style>**

body { background: #f0f0f0; }

@media screen and (max-width: 600px) {

body { background: #fc0; }

}

**</style>**

**</head>**

**<body>**

**<p>**Пока магма остается в камере, мусковит сингонально поднимает шток,

в то время как значения максимумов изменяются в широких пределах. **</p>**

**</body>**

**</html>**

1. **Orientation** - Определяет, что устройство находится в альбомном режиме (ширина больше высоты) или портретном (ширина меньше высоты).

* Тип носителя: handheld, print, projection, screen, tty, tv
* Значение: ***landscape*** (альбомная ориентация) | ***portrait*** (портретная ориентация)

Пример (в примере устанавливается разная фоновая картинка в случае альбомной или портретной ориентации):

@media screen and (orientation: landscape) {

#logo { background: url(logo1.png) no-repeat; }

}

@media screen and (orientation: portrait) {

#logo { background: url(logo2.png) no-repeat; }

}

1. **resolution (min-resolution, max-resolution)** - Определяет разрешение устройства, например, принтера.

* Тип носителя: ***handheld***, ***print***, ***projection***, ***screen***, ***tv***
* Значение: ***разрешение в dpi*** (точек на дюйм) или ***dpcm*** (точек на сантиметр)

Пример (в примере стиль будет работать для принтера с минимальным разрешением 300 точек на дюйм):

@media print and (min-resolution: 300dpi) {

...

}

## Использование разных единиц измерений необходимых для адаптивности макета

1. **px** – Фиксированная единица измерений. Используется по умолчанию при определении позиции блока на странице. При изменении размера страницы положения блока или его размер не ИЗМЕНЯЕТСЯ. **Для адаптивной верстки не используется!!!**
2. **%** - Единица измерения, используемая для «резиновый» страниц. Размер блока зависит от размера родителя (где по дефолту размер родителя = 100%). При уменьшении ширины страницы, родительский блок подстраивается по размер страницы и все блоки которые в нем находятся просто уменьшаются в размере до максимального возможного.

**Адаптивным методом «резиновые страницы» назвать нельзя, так как блоки не перестраиваются под размер устройства пользователя, а просто уменьшаются в размере, что вызывает неудобства при использовании сайта.**

1. **em –** Единица измерения которая зависит от размера родителя. Размер em автоматически меняет свой размер при изменении размера родителя.

Пример:

<html>

<head>

<style>

html,body{

font-size:20px;

}

.block1{

font-size:1em;

}

.block2{

font-size:1.5em;

}

.block3{

font-size:1em;

}

.block4{

font-size:2em;

}

</style>

</head>

<body>

<div class=”block1”>

размер текста составит 20px

<div class=”block2”>

высота текста составит 30px (1,5\*20)

<div class=”block3”> высота текста составит 30px</div>

<div class=”block4”> высота текста составит 60px (2\*30)</div>

</div>

</div>

</body>

</html>

1. **rem** – Имеет такие – же свойства как и **em**, только размер блока зависит не от родителя, а от стандартного значения размера самой страницы. Стандартный размер составляет 16px, либо его можно задать с помошью свойства: **font-size**.

Пример:

<html>

<head>

<style>

html,body{

font-size:20px;

}

.block1{

font-size:1 rem;

}

.block2{

font-size:1.5 rem;

}

.block3{

font-size:1 rem;

}

.block4{

font-size:2 rem;

}

</style>

</head>

<body>

<div class=”block1”>

размер текста составит 20px

<div class=”block2”>

высота текста составит 30px (1,5\*20)

<div class=”block3”> высота текста составит 20px</div>

<div class=”block4”> высота текста составит 40px (2\*20)</div>

</div>

</div>

</body>

</html>

JAVASCRIPT (JS)

* **это язык скриптов, на котором держится весь frontend веб-разработки. Он позволяет перехватывать события и выполнять различные действия.** Например, пользователь кликнул по какой-нибудь кнопке — сработало событие click. И, связав с ним, мы можем выполнить нужную нам функцию — открыть модальное окно или изменить цвет элемента.

### Комментарии

Для того, чтобы закоментировать только одну строку кода используют: **//**

Для того, чтобы закомментировать целый блок кода используют: **/\* \*/**

### Подключение javascript

1. **Написание кода внутри html документа**

Весь код javascript необходимо прописать внутри тега **<script></script>**, который в свою очереть должен находится внутри тела html документа (то есть внутри **<body></body>**

HTML код страницы загружается сверху вниз, поэтому весь javascript **лучше прописывать после html кода**, так как в противном случае, пока весь контейнер с javascript не загрузится (*код может быть объемным и поэтому может дольше грузится*) последующий код не будет загружет и тем самым не будет отображена часть страницы, пока не загрузится код.

Пример:

<!DOCTYPE html>  
<html>  
<head>  
<meta charset="utf-8">  
</head>  
<body>  
<p>Это обычный HTML документ</p>  
  
<script language="JavaScript">  
    alert("Привет, мир!");  
</script>  
  
<p>Выходим обратно в HTML</p>  
</body>  
</html>

1. **Подключение отдельного javascript файла**

Так как javascript код может быть объемным, то следует этот код прописывать в отдельном файле. Преимущества такого подключения такие же, как и при подключении отдельного CSS файла, то есть:

* Возможность с экономить места в html файле, что в свою очередь поможет не запутаться в собственном коде
* Возможность подключения одной и той же функции на множества html страницах, не прописывая его в каждой из них, а также легкость внесения поправок в подключенном скрипте (не придется вносить одних и тех-же изменения множество раз)

Подключение отдельного файла такое-же, что и в первом примере, только написания кода внутри контейнера **<script></script>** не обязателен, необходимо только указать путь к javascript файлу.

Пример:

<!DOCTYPE html>  
<html>  
<head>  
<meta charset="utf-8">  
</head>  
<body>  
<p>Это обычный HTML документ</p>  
  
<script src="/scripts/script.js"></script> // в src указывается путь к javascript файлу  
  
<p>Выходим обратно в HTML</p>  
</body>  
</html>

### Атрибуты подключения JS

1. **async -** используется для того, чтобы указать браузеру, что скрипт может быть выполнен «асинхронно».

При обнаружении тега **<script async src="...">** браузер не останавливает обработку HTML-документа для загрузки и выполнения скрипта, выполнение может произойти после того, как скрипт будет получен параллельно с разбором документа. Когда скрипт будет загружен – он выполнится.

Пример:

<html>

<head>

</head>

<body>

<script async src="script1.js"></script>

<script async src="script2.js"></script>

*<!-- контент -->*

</body>

</html>

1. **defer** - откладывает выполнение скрипта до тех пор, пока вся HTML-страница не будет загружена полностью.

Как и при асинхронной загрузке скриптов — JS-файл может быть загружен, в то время как HTML-документ ещё грузится.

Однако, даже если скрипт будет полностью загружен ещё до того, как браузер закончит обработку страницы, он не будет выполнен до тех пор, пока HTML-документ не обработается до конца.

Пример:

<html>

<head>

<script defer src="script1.js"></script>

<script defer src="script2.js"></script>

</head>

<body>

*<!-- контент -->*

</body>

</html>

Основные методы JS

### alert()

– Выводит браузерное окно с сообщением и с кнопкой «ОК» (объект: **window**)

Пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Alert</title>

</head>

<body>

    <script>

        window.alert("Hello world");

    </script>

</body>

</html>

### confirm()

– Выводит так же сообщение, что и метод **alert**, но только с кнопкой «Ок» или «Отмена». (объект: **window**)

Эти две кнопки позволяют улучшить логику программу, где пользователь теперь предлагается выбрать одно из двух действий, где необходимо прописать что должно произойти, если пользователь нажмет на то или иную кнопку.

Пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>confirm</title>

</head>

<body>

    <script>

        if (window.confirm("Нажмите ОК или Отмена")) {

            window.alert("Вы нажали ОК");

        } else {

            window.alert("Вы нажали Отмена");

        }

    </script>

</body>

</html>

### promt()

– Отображает такое же окно с кнопками, как и метод **confirm**, но в нем можно занести информацию внутри поля и занести ее в заранее объявленном переменном типе **const**. (объект: **window**)

Синтаксис:

**promt (“Сообщение для пользователя”,”Дефолтное значение в поле ввода”)**

Пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>confirm</title>

    <style>

        p {

            font-size: 30px;

            color: blue;

        }

        pn {

            color: red;

            font-family: 900;

            text-decoration: underline;

        }

    </style>

</head>

<body>

    <script>

        const user\_name = window.prompt("Введите ваше имя", "Пользователь");

        if (user\_name == null) {

            alert("Пока!");

        } else {

            window.alert("Добро пожаловать " + user\_name);

            document.write("<p>" + "<pn>" + user\_name + "</pn>&nbsp;" + "Мы успешно выполнили программу</p>");

        }

    </script>

</body>

</html>

### write()

* метод, выводящий на страницу переданные ему аргументы. Аргументов может быть любое количество, и они могут быть любых типов, при выводе они преобразуются в строки. (объект: **document**)

**document.write** выводит свои аргументы после того тега script, в котором он был вызван. Пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>write</title>

</head>

<body>

    <script>

        document.write('<h1>Hello, World!</h1>');

    </script>

</body>

</html>

Данный пример эквивалентен такой записи:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>write</title>

</head>

<body>

    <script>

    </script>

    <h1>Hello, World!</h1>

</body>

</html>

При выполнении скрипта строчка: <h1>Hello, World!</h1> записывается в html – файл не на прямую, а динамическим способом, то есть эта строчка будет записана на странице до того момента, пока активен скрипт. (Скрипт завершается при перезагрузке страници и снова активируется при его загрузке).

**document.write** работает только на этапе загрузки страницы, его нельзя вызывать в ответ на какое-нибудь событие.

Если **document.write** вызвать после того, как страница загрузилась, результат во всех браузерах будет примерно одинаков — пустая страница и, возможно, тот текст, который был выведен последним document.write. Поэтому даже не пытайтесь использовать его в ответ на какие-то действия пользователя, только на этапе загрузки страницы.

#### Вывод скриптов с помощью метода: write

Как и любое другое содержимое, **document.write** может выводить и скрипты, которые тоже в свою очередь могут содержать document.write.

Но нужно помнить: внедрённые в HTML скрипты **не могут содержать внутри себя в явном виде строку </script>**.

Нужно заменить какой-нибудь символ на escape-последовательность **<\x2fscript>**.

**В скриптах, содержащихся во внешних файлах, такие манипуляции не требуются!**

Пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>write</title>

</head>

<body>

    <script>

       document.write('<script type="text/javascript">alert(1);<\x2fscript>');       // Выведет окошко alert

    </script>

</body>

</html>

Если вложенные скрипты тоже, в свою очередь, содержат **document.write**, то всё будет выведено последовательно:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>write</title>

</head>

<body>

    <script>

        document.write('Создаём первый скрипт...<br>');

        document.write('<script type="text/javascript">document.write("Первый скрипт<br>");<\x2fscript>');

        document.write('Создаём второй скрипт...<br>');

        document.write('<script type="text/javascript">document.write("Второй скрипт<br>");<\x2fscript>');

        document.write('Закончили.<br>');

    </script>

</body>

</html>

Объявление переменных

**Переменная** – поименованная область памяти, хранящая данные.

|  |  |
| --- | --- |
| **Типы данных переменных** | |
| **number** | Числа, как целые, так и с плавающей точкой |
| **string** | Строки |
| **boolean** | Логический тип данных, может содержать два значения – true (истина) и false (ложь) |
| **function** | Функции. В JS мы можем присвоить ссылку на функцию любой переменной, если указать имя функции без круглых скобок. |
| **object** | Массивы, объекты, а так же переменные со значнием null |

**Оператор переменных:**

1. **let** – Оператор объявления переменных. После того как в переменные будут занесены данные, в переменную автоматически будет занесена информация о типе данных которую она хранит.
2. **const** – Оператор объявления переменных. Выполняет все теже функции, что и оператор **let**, *но не может изменять данные переменной после его объявления*, то есть становится **константой**.
3. **typeof** – Возвращает строку, описывающую тип данных переменной.

Пример 1:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>typeof and let</title>

</head>

<body>

    <script>

        let num1, num2, str1, str2, str3, bool1;

        num1 = 5;

        num2 = 5.5;

        str1 = "Hello";

        str2 = 'world';

        str3 = null;

        bool1 = true;

        document.write("<br> num1 - " + typeof(num1) + " Значение = " + num1);

        document.write("<br> num2 - " + typeof(num2) + " Значение = " + num2);

        document.write("<br> str1 - " + typeof(str1) + " Значение = " + str1);

        document.write("<br> str2 - " + typeof(str2) + " Значение = " + str2);

        document.write("<br> str3 - " + typeof(str3) + " Значение = " + str3);

        document.write("<br> bool1 - " + typeof(bool1) + " Значение = " + bool1);

    </script>

</body>

</html>

* **Int** – целое число типа number
* **Float** – число с плавующей точкой (вещественное число) типа number
* **String** - строка

|  |  |
| --- | --- |
| Функции для принудительного преобразования типов данных | |
| parseInt | Преобразует строку в целое число, если это возможно |
| parseFloat | Преобразует строку в вещественное число, если это возможно |
| eval | Вычисляет выражение строки, как будто это обычное выражение JS |

Пример:

<!DOCTYPE html>

<html lang="rus">

<head>

    <meta charset="UTF-8">

    <title>Parse</title>

    <style>

        input {

            display: block;

            margin-bottom: 15px;

  width: 220px;

}

        button {

            display: block;

        }

        form {

            width: max-content;

        }

        .block1 div {

            float: left;

        }

        .bl1 {

            margin-right: 50px;

        }

    </style>

</head>

<body>

    <div class="block1">

        <div class="bl1">

            <form>

                <fieldset>

                    <legend>Форма для вычисления</legend>

                    <input type="text" id="per1" placeholder="Введите целое значение" required>

                    <input type="text" id="per2" placeholder="Введите вещественое значение" required>

                    <input type="text" id="per3" placeholder="Введите формулу для вычисления" required>

                    <button onclick="otvet()">Вычислить</button>

                </fieldset>

            </form>

        </div>

        <div class="bl2">

            <script>

                function otvet() {

                    //Занесение данных в переменную

                    let p1 = document.getElementById('per1').value;

                    let p2 = document.getElementById('per2').value;

                    let p3 = document.getElementById('per3').value;

                    //Вывод ошибки при отсутствии данных

                    if (p1 == null || p2 == null || p3 == null) {

                        window.alert("ERROR!")

                    }

                    //Вычисление

                    let itog1 = parseInt(p1) + parseFloat(p2);

                    let itog2 = eval(p3);

//Вычисление

                    window.alert("Сумма двух первых переменных: " + itog1 + "\nРезультат 2 выражения: " + itog2);

                }

            </script>

        </div>

    </div>

</body>

</html>

## Оператор присванивания:

|  |  |  |
| --- | --- | --- |
| **Оператор** | **Пример** | **Описание** |
| += | x+=y | x=x+y |
| -= | x-=y | x=x-y |
| \*= | x\*=y | x=x\*y |
| /= | x/=y | x=x/y |
| %= (остаток от деления) | x%=y | x=x%y |

Основные способы занесения данных в переменные JS

1. Присваивавание значений переменных непосредственно внутри скрипта

Пример:

<script>

        let per1 = 15;

        let per2 = 65;

        let sum = per1 + per2;

        window.alert("Результат суммы равен: " + sum);

    </script>

1. Присваивание значений переменных через стандартный метот [**promt**](#_promt())

Пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>prompt</title>

    <style>

        p {

            font-size: 35px;

            color: black;

            font-weight: 400;

        }

        pn {

            color: brown;

            font-weight: 800;

        }

    </style>

</head>

<body>

    <script>

        let per1 = window.prompt("Введите значение 1");

        if (per1 == null) {

            window.alert("Программа не выполнена!");

        } else {

            let per2 = window.prompt("Введите значение 2");

            if (per2 == null) {

                window.alert("Программа не выполнена!");

            } else {

                let sum = parseInt(per1) + parseInt(per2);

                document.write("<p>Сумма двух меременных = <pn>" + sum + "</pn></p>")

            }

        }

    </script>

</body>

</html>

1. Занесение данных из input html – страницы в переменную JS через метод **getElementById** (объект: **document**)

Пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>date of input</title>

</head>

<body>

    <input type="text" id="test">

    <button onclick="btn1()"> Нажмите меня</button>

    <script>

        function btn1() {

            var p = document.getElementById('test').value;

            alert(p);

        }

    </script>

</body>

</html>

# Операторы сравнения

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| == | Оператор «равно». Результатом выражения А==В является значение **true**, если значение операнда А равно значению операнда В. В противном случает значение выражения равно **false**. Если значения операндо отностится к разным типам, *автоматически предпринимается попытка приведения значений к одному типу для проведения сравнения* |
| != | Оператор «неравно». Результатом выражения А!=В является значение **true**, если значение операнда А не равно значению операнда В. В противном случае значение выражения равно **false**. Для значений опернадов А и В разных типов *автоматически выполняется приведение к одному типу (если возможно) для проведения сравнения*. |
| === | Оператор «строгого равно». Результатом выражения А===В является значение **true**, если значение операнда А равно значению операнда В. В противном случае значение выражения равно **false**. В отличии от оператора == в данном случае *приведение к одному типу значений операндов не выполняется* |
| !== | Оператор «строго неравно». Результатом выражения А!=В является значение **true**, если значение операнда А не равно значению операнда В. В противном случае значение выражения равное **false**. В отличии от операнда != *не выполняется приведение к одному типу значений операндов* |
| > | Оператор «больше». Результатом выражения А>B является значение **true**, если значение операнда А больше значения операнда В. В противном случае значение выражения равно **false**. |
| >= | Оператор «больше или равно». Результатом выражения A>=B является значение **true**, если значение опенада А больше или равно значению опернада В. В противном случае значение выражения равно **false** |
| < | Оператор «меньше». Результатом выражения A<B является значение **true**, если значение опернада А меньше значения операнда В. В противном случае значение выражения равно **false** |
| <= | Оператор «меньше или равно». Результатом выражения A<=B является значение **true**, если значение операнда А меньше или равно значению операнда В. В противномслучае значение выражения равно **false** |

# Логические операторы

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| && | Оператор для вычисления *логического и*. Оператор бинарный. Результатом выражения A&&B при обоих логических операндов А и В является значение **true**, если оба операнда равны **true**. Если хотя бы один из двух опернадов равен **false**, результатом выражжения A&&B является значени **false**. |
| || | Оператор для вычисления *логического или*. Оператор бинарный. Результатом выражения A||B (при условии, что оба операнда А и В относятся к логическому типу) является значение **true**, если хотя бы один из операндов равен **true**. Если оба опернада равны **false**, результатом выражения A||B является значение **false**. |
| ! | Оператор *логического отрицания*. Унарный оператор. Если значение операнда А равно **true**, то результат выражения !А равен **false**. И наоборот, если значение операнда равно **false**, то результатом выражения !А является значение **true** |

## Таблица истинности для операции логического и

|  |  |  |
| --- | --- | --- |
| A&&B | true | false |
| true | true | false |
| false | false | false |

## Таблица истинности для операции логического или

|  |  |  |
| --- | --- | --- |
| A||B | true | false |
| true | true | true |
| false | true | false |

Слияние строк

Для слияния (конкатенации) несколько строк в одну, исользуют оператор **+**

Пример:

<script>

        let p1 = "Hello";

        let p2 = " world";

        const str1 = p1 + p2;

        const str2 = "I" + " am" + " a" + " first" + " programm";

        window.alert(str1 + "\n" + str2);

    </script>

Условный оператор if

- Данный оператор позволяет выполнить определенное действие в зависимости от истинности условия. Общая форма выглядит так:

**if (){**

**Операторы, если условие истинно**

**}**

**else{**

**Операторы, если условие ложно**

**}**

Вторая часть опеатора (**else**) не обязательна.

Пусть у нас будут две переменные:

const A=10;

const B=5;

Оператор **А==В** вернет **false**, поскольку **А** не равно **В**.

Оператор **A>B** вернет **true**, поскольку **А** больше, чем **В**.

Оператор **!(А==В)** вернет **true**. Переменные А и В не равны, поэтому оператор == вернет значение **false**, но поскольку указан оператор **!**, то будет возвращено значение **true**.

Пример 1:

<script>

        let A = window.prompt("Введите первое число");

        let B = window.prompt("Введите второе число");

        if (A != null) {

            if (B != null) {

                if (A > B) {

                    window.alert("Число " + A + " больше числа " + B);

                }

                if (A < B) {

                    window.alert("Число " + A + " меньше числа " + B);

                }

                if (A == B) {

                    window.alert("Числа равны");

                }

            } else {

                window.alert("Вы не ввели второе число");

            }

        } else {

            window.alert("Вы не ввели первое число");

        }

    </script>

Как видно из данного примера, операторы **if** можно вкладывать друг в друга, что позволяет проверять следующее условие только при выполнении предыдущего условия.

Проверка состояния/запуска объектов тега <form></form>

## <button></button>

Самый основной запуск скриптов с использованием объектов тега <form></form>. Для проверки нажатия на кнопку и запуска скрипта после нажатия, чаще всего используют функцию **onclick**

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Button</title>

</head>

<body>

    <button id="btn1">Button</button>

    <script>

        btn1.onclick = function() {

            window.alert("Кнопка нажата");

        }

    </script>

</body>

</html>

**ИЛИ**

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

</head>

<body>

    <form>

        <input id="text1" type="text" placeholder="Введите первое значение"><br><br>

        <button onclick="btn1()">Кнопка</button>

    </form>

    <script>

        function btn1() {

            var A = document.getElementById("text1").value;

            window.alert(A);

        }

    </script>

</body>

</html>

**ИЛИ (Приорететный вариант)**

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

</head>

<body>

    <form>

        <input id="text1" type="text" placeholder="Введите первое значение"><br><br>

        <button onclick="btn1()">Кнопка</button>

    </form>

    <script>

        const out = document.querySelector("button");

out.addEventListener("click", function() {

var A = document.getElementById("text1").value;

            window.alert(A);

});

        }

    </script>

</body>

</html>

## Checkbox и radiobutton

Можно написать скрипт, который выполняет какие-либо действия при выборе несколько checkbox (radiobutton)

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Checkbox</title>

</head>

<body>

    <form>

        <input id="check1" type="radio">checkbox1<br>

        <input id="check2" type="radio">checkbox2<br>

        <input id="check3" type="radio">checkbox3<br><br>

        <button id="btn1">Кнопка</button>

    </form>

    <script>

        btn1.onclick = function() {

            if (check1.checked) {

                window.alert("Выбран checkbox1");

            } else {

                if (check2.checked) {

                    window.alert("Выбран checkbox2");

                } else {

                    if (check3.checked) {

                        window.alert("Выбран checkbox3");

                    } else {

                        window.alert("Вы не выбрали ни один checkbox")

                    }

                }

            }

        }

    </script>

</body>

</html>

Так же можно модернизировать программу и сделать так, чтобы пользователь смог выбрать только один checkbox (radiobutton):

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>Checkbox</title>

</head>

<body>

    <form>

        <input id="check1" type="radio" name="ch">checkbox1<br>

        <input id="check2" type="radio" name="ch">checkbox2<br>

        <input id="check3" type="radio" name="ch">checkbox3<br><br>

        <button id="btn1">Кнопка</button>

    </form>

    <script>

    // Участок кода для выбора только одного checkbox

        var inputs = document.querySelectorAll("[name=ch]")

        for (var i = 0; i < inputs.length; i++) {

            inputs[i].addEventListener('change', changeCheck)

        }

        function changeCheck() {

            var checkedInput = this;

            for (var i = 0; i < inputs.length; i++) {

                inputs[i].checked = false;

            }

            checkedInput.checked = true

        }

    //------------------------------------------------

        btn1.onclick = function() {

            if (check1.checked) {

                window.alert("Выбран checkbox1");

            } else {

                if (check2.checked) {

                    window.alert("Выбран checkbox2");

                } else {

                    if (check3.checked) {

                        window.alert("Выбран checkbox3");

                    } else {

                        window.alert("Вы не выбрали ни один checkbox")

                    }

                }

            }

        }

    </script>

</body>

</html>

## Деактивация компонентов input

1 способ

var inputs = document.querySelectorAll("[name=ch]")

for (var i = 0; i < inputs.length; i++) {

            inputs[i].disabled = true;

        }

Добавление CSS свойства к объекту через JS

Для того, чтобы изменить определенное CSS свойство к какому-либо объекту при выполнении какого-либо скрипта, к примеру, изменить цвет текста после нажатия на кнопку, необходимо использовать следующий метод: **getElementById("имя уникального идентификатора").style** (объект: **document**).

**Синтаксис:**

document.getElementById("имя уникального идентификатора").style.css свойство

Пример:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <style>

        #div1 {

            display: none;

        }

    </style>

</head>

<body>

    <input type="button" value="Click" onclick="viewDiv()">

    <div id="div1">Блок</div>

    <script>

        function viewDiv() {

            document.getElementById("div1").style.display = "block";

        };

    </script>

</body>

</html>

Но если у CSS свойства есть дефиз, то превычная запись действовать не будет.

Давайте рассмотрим пример на свойстве: **font-size**. Для того, чтобы все работало в этом свойстве необходимо убрать дефиз и первую букву слова после него необходимо поставить в верхнем регистре. Пример:

<script>

        function btn1() {

            document.getElementById("zag").style.fontSize = "50px";

            document.getElementById("zag").style.marginTop = "20px";

        }

    </script>

Более сложный способ вывода информации

Зачастую, способ вывода информации с помошью метода **alert** или **confirm,** недостаточны, поэтому лучший способ вывести результат вычислений – это вывести его на экран web – страницы.

Метод **write** в данном случае неподходит, так как он работает, только при загрузке самой странице, а не при вызове функции обработчика события.

Следующий пример (создание самого простого варианта калькулятора на JS) показывает работу метода **getElementById** (объект: **document**), а также наиболее лучшего способа вызова функции с помошью метода **addEventListener**

<!DOCTYPE html>

<html lang="rus">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Калькулятор</title>

    <style>

        .forma\_calc #vvod\_in\_calc {

            display: block;

            margin: 0 auto;

            width: 300px;

            height: 25px;

            font-size: 20px;

            color: gray;

        }

        .forma\_calc #vvod\_in\_calc:focus {

            color: black;

            font-weight: 600;

            box-shadow: 0 2px 4px black;

        }

        fieldset {

            width: 550px;

            display: block;

            margin: 0 auto;

            padding-top: 20px;

            padding-bottom: 20px;

            font-size: 30px;

            font-weight: 600;

        }

        .panel\_for\_otvet {

            width: 550px;

            outline: 1px solid black;

            padding-top: 20px;

            padding-bottom: 20px;

            padding-left: 15px;

            padding-right: 15px;

            margin: 0 auto;

            margin-top: 50px;

            font-size: 30px;

            font-weight: 600;

        }

        .panel\_for\_otvet pn {

            color: red;

            font-weight: 800;

        }

    </style>

</head>

<body>

    <form action="">

        <fieldset>

            <legend>Самый простой пример калькулятора</legend>

            <div class="forma\_calc">

                <input type="text" id="vvod\_in\_calc" placeholder="Введите выражение" required>

            </div>

        </fieldset>

    </form>

    <div class="panel\_for\_otvet">

        <p>Результат выражения =

            <pn id="otvet"></pn>

        </p>

    </div>

    <script>

        const out = document.querySelector("input");

        let x;

        out.addEventListener("keyup", function() {

            x = document.getElementById('vvod\_in\_calc').value;

            x = eval(x);

            if (x == null) {

                x = 0;

            }

/\*Метод записи в объект с id=”otvet” результат переменной х\*/

            document.getElementById('otvet').innerHTML = x;

        });

    </script>

</body>

</html>

const out = document.querySelector("input");

- Данная строка создает собственный объект **out** и занес с htm – документа **input** (объект html – страницы в котором будет в последующем вызвана функция)

out.addEventListener("keyup", function() – Данная строчка вызывает функцию обработчика события **keyup** (Нажатие на любую клавишу)

x = document.getElementById('vvod\_in\_calc').value; - Эта строчка нам [знакома](#getElementById). Мы получаем данные которые пользователь занес в поле **input**

document.getElementById('otvet').innerHTML = x;

где otvet – **id адрес тега** внутри которого будет выведены данные переменной **х**.

## Еще один пример вызова функции обработчика события

Ниже приведен еще один пример (закрашивание блока div после нажатия на клавишу «ENTER» цветом, который был выбран в **input type=”color”**) который показывает работу разных обработчиков событий и коды клавиш, а так же еще один пример вызов функции обработчика события в скрипте с помошью **addEventListener**:

<!DOCTYPE html>

<html lang="rus">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Палитра</title>

    <style>

        .panel\_color {

            background: linear-gradient(to top, #fefcea, #f1da36);

            margin: 0 auto;

            width: max-content;

            padding-left: 15px;

            padding-right: 15px;

            color: darkblue;

            font-size: 30px;

            font-weight: 900;

            padding-top: 15px;

            padding-bottom: 15px;

        }

        .panel\_color p {

            margin: 0;

            padding: 0;

        }

        .panel\_color p:last-of-type {

            color: darkred;

            font-size: 20px;

            text-align: center;

            margin-top: 30px;

        }

        .panel\_color p:last-of-type pn {

            color: darkgreen;

            font-weight: 1000;

        }

        #vvod\_color {

            display: block;

            margin: 0 auto;

            margin-top: 30px;

            width: 100px;

            height: 40px;

            background-color: gold;

            box-shadow: 0 2px 10px black;

            cursor: pointer;

        }

        .panel\_for\_paint {

            box-sizing: content-box;

            width: max-content;

            margin: 0 auto;

            padding: 50px 50px 50px 50px;

            background-color: black;

            color: gold;

            font-weight: 800;

            margin-top: 100px;

            font-size: 20px;

        }

    </style>

</head>

<body>

    <div class="panel\_color">

        <p>Выбирете цвет для закрашивания фона рядом стоящего блока</p>

        <div><input id="vvod\_color" type="color"></div>

        <p>Для закрaшивания блока нажмите

            <pn>ENTER</pn>

        </p>

    </div>

    <div id="panel\_paint" class="panel\_for\_paint">

    </div>

    <script>

        addEventListener("keydown", function(event) {

            if (event.keyCode == 13) {

                var color = document.getElementById('vvod\_color').value;

                document.getElementById("panel\_paint").style.backgroundColor = color;

            }

        });

    </script>

</body>

</html>

В **первом** примере был выбран следующий способ вызова функции:

const out = document.querySelector("input");

        out.addEventListener("keyup", function() {

…

});

1. Первая строчка говорит нам, что мы выбираем объект из нашей web – страници где будем вызывать фунцию обработчика события.
2. Вторая строчка говорит на о том, что при нажатии на любую клавишу в **input** (выбрали на первой строчке), будут выполнятся команды которые прописаны внутри функции.

Во **втором** примере логика вызова функции обработчика события немнога иная.

addEventListener("keydown", function(event) {

            if (event.keyCode == 13) {

                …

}

        });

1. Первая строчка говорит нам о том, что при нажатии любой клавиши в любой части web – страницы будут выполнятся команды, которые прописаны внутри функции
2. Вторая строчка говорит нам о том, что мы проверяем нажали ли мы на клавишу ENTER (код символа “ENTER” – 13) или нет. Если да, то выполняются команды, которые прописанны внутри условия **if**.

Обработчик событий

***Событие*** – это сигнал от браузера о том, что что-то произошло.

|  |  |
| --- | --- |
| **Наименование события** | **Описание** |
| **Событие мыши** | |
| click | Происходит, когда кликнули на элемент левой кнопкой мыши (на устройствах с сенсорными экранами оно происходит при касании). |
| contextmenu | Происходит, когда кликнули на элемент правой кнопкой мыши. |
| mouseover / mouseout | Когда мышь наводится на / покидает элемент. |
| mousedown / mouseup | Когда нажали / отжали кнопку мыши на элементе. |
| mousemove | При движении мыши. |
| **События на элементах управления** | |
| submit | Пользователь отправил форму **<form>** |
| focus | Пользователь фокусируется на элементе, например нажимает на **<input>** |
| **Клавиатурные события** | |
| keydown / keyup | Когда пользователь нажимает / отпускает клавишу. |
| **События документа** | |
| DOMContentLoaded | Когда HTML загружен и обработан, DOM документа полностью построен и доступен. |
| **CSS события** | |
| transitionend | Когда CSS-анимация завершена. |

## Способы вызова функций

1. Использование атрибута HTML

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется **on<*событие*>**.

Пример:

<input value="Нажми меня" onclick="alert('Клик!')" type="button">

При клике мышкой на кнопке выполнится код, указанный в атрибуте onclick.

Обратите внимание, для содержимого атрибута **onclick** используются одинарные кавычки, так как сам атрибут находится в двойных. Если мы забудем об этом и поставим двойные кавычки внутри атрибута, вот так: **onclick="alert("Click!")"**, *код не будет работать!*

* 1. Запуск функции по его названию

Пример:

<script>

        function countRabbits() {

          for(let i=1; i<=3; i++) {

            alert("Кролик номер " + i);

          }

        }

      </script>

      <input type="button" onclick="countRabbits()" value="Считать кроликов!">

1. Использование свойства DOM-объекта

Обработчик события записывается в **id** того объекта web – страницы внутри которого необходимо выполнить команды при запуске функции.

Пример:

<input id="elem" type="button" value="Нажми меня!">

<script>

  elem.onclick = function() {

    alert('Спасибо');

  };

</script>

Фундаментальный **недостаток** описанных выше способов назначения обработчика – **невозможность повесить несколько обработчиков на одно событие**.

1. addEventListener ([Примеры использования](#addEventListener))
2. **addEventListener** – Вызов функций обработчика события (можно вызывать сколько угодно разных функций при одном вызове обработчика события)

**Синтаксис:**

element.addEventListener(event, handler[, options]);

* **event** - Имя события, например *"click"*.
* **handler** - Ссылка на функцию-обработчик.
* **options** - Дополнительный объект со свойствами:
  + **once:** если **true**, тогда обработчик будет автоматически удалён после выполнения, если **false**, то тогда обработчик события не будет удален после выполнения и будет каждый раз выполняться при активации
  + **capture:** фаза, на которой должен сработать обработчик (**true\false**)

1. element.removeEventListener – удаляет обработчик события после его выполнения

**Синтаксис:**

element.removeEventListener(event, handler[, options]);

Пример:

function handler() {

  alert( 'Спасибо!' );

}

input.addEventListener("click", handler);

// ....

input.removeEventListener("click", handler);

Метод **addEventListener** позволяет добавлять несколько обработчиков на одно событие одного элемента

Пример:

<input id="elem" type="button" value="Нажми меня"/>

<script>

  function handler1() {

    alert('Спасибо!');

  };

  function handler2() {

    alert('Спасибо ещё раз!');

  }

  elem.onclick = () => alert("Привет");

  elem.addEventListener("click", handler1); // Спасибо!

  elem.addEventListener("click", handler2); // Спасибо ещё раз!

</script>

## Обращение к фунции

В коде скрипта, либо внутри обработчика события можно ссылаться на одну или несколько функций. Это полезно к примеру, когда вам необходимо выполнить команды функции и дальше продолжать код обработчика события.

Пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

</head>

<body>

    <input type="text" id="txt1">

    <button onclick="bt1()">Нажми</button>

    <script>

        let p

        function bt1() {

            p = document.getElementById("txt1").value;

            if (p == '') {

                funk1();

            } else {

                funk2();

            }

            alert("Программа закончена")

        }

        function funk1() {

            alert("Вы ничего не ввели!");

        }

        function funk2() {

            alert(p);

        }

    </script>

</body>

</html>

# Коды клавиш для использования в функциях

|  |  |
| --- | --- |
| **KeyCode** | **Button name** |
| 8 | BackSpace |
| 9 | Tab |
| 13 | Enter |
| 20 | CapsLock |
| 27 | Esc |
| 32 | пробел |
| 33 | PageUp |
| 34 | PageDown |
| 35 | End |
| 36 | Home |
| 37 | Стрелка влево |
| 38 | Стрелка вверх |
| 39 | Стрелка вправо |
| 40 | Стрелка вниз |
| 44 | PrintScreen |
| 45 | Insert |
| 46 | Delete |
| 48 | 0 |
| 49 | 1 |
| 50 | 2 |
| 51 | 3 |
| 52 | 4 |
| 53 | 5 |
| 54 | 6 |
| 55 | 7 |
| 56 | 8 |
| 57 | 9 |
| 65 | A |
| 66 | B |
| 67 | C |
| 68 | D |
| 69 | E |
| 70 | F |
| 71 | G |
| 72 | H |
| 73 | I |
| 74 | J |
| 75 | K |
| 76 | L |
| 77 | M |
| 78 | N |
| 79 | O |
| 80 | P |
| 81 | Q |
| 82 | R |
| 83 | S |
| 84 | T |
| 85 | U |
| 86 | V |
| 87 | W |
| 88 | X |
| 89 | Y |
| 90 | Z |
| 91 | Win(Л) |
| 92 | Win(Пр) |
| 96 | Доп(0) |
| 97 | Доп(1) |
| 98 | Доп(2) |
| 99 | Доп(3) |
| 100 | Доп(4) |
| 101 | Доп(5) |
| 102 | Доп(6) |
| 103 | Доп(7) |
| 104 | Доп(8) |
| 105 | Доп(9) |
| 106 | Доп(\*) |
| 107 | Доп(+) |
| 109 | Доп(-) |
| 110 | Доп(,) |
| 111 | Доп(/) |
| 112 | F1 |
| 113 | F2 |
| 114 | F3 |
| 115 | F4 |
| 116 | F5 |
| 117 | F6 |
| 118 | F7 |
| 119 | F8 |
| 120 | F9 |
| 121 | F10 |
| 122 | F11 |
| 123 | F12 |
| 124 | F13 |
| 125 | F14 |
| 126 | F15 |
| 127 | F16 |
| 128 | F17 |
| 129 | F18 |
| 130 | F19 |
| 131 | F20 |
| 132 | F21 |
| 133 | F22 |
| 134 | F23 |
| 135 | F24 |
| 144 | Numlock |
| 166 | Browser Back |
| 167 | Browser Forward |
| 168 | Browser Refresh |
| 169 | Browser Stop |
| 170 | Browser Search |
| 171 | Browser Favorites |
| 172 | Browser Home |
| 173 | Volume Mute |
| 174 | Volume Down |
| 175 | Volume Up |
| 176 | Next Track |
| 177 | Previous Track |
| 178 | Stop Media |
| 179 | Play/Pause |
| 180 | Start Mail |
| 181 | Select Media |
| 182 | Start App 1 |
| 183 | Start App 2 |

# Список некоторых методов

## Массивы

**pop()** - удаляет последний элемент массива

<script>

        var students = [ 'Jack', 'James', 'Robert', 'John'];

        console.log(students);

        students.pop();

        console.log(students)

    </script>

**shift()** - удаляет первый элемент массива

<script>

        var students = [ 'Jack', 'James', 'Robert', 'John'];

        console.log(students);

        students.shift();

        console.log(students)

    </script>

**push()** – Добавление в конец массива один или несколько элементов

<script>

         var students = [ 'Jack', 'James', 'Robert', 'John'];

        console.log(students);

        students.push('Zahab', 'Kakar');

        console.log(students)

    </script>

**unshift()** – Добавление в начало массива один или несколько элементов

<script>

         var students = [ 'Jack', 'James', 'Robert', 'John'];

        console.log(students);

        students.unshift('Zahab', 'Kakar');

        console.log(students);

    </script>

**length** - возвращает число элементов в массиве

<script>

         var students = [ 'Jack', 'James', 'Robert', 'John'];

        console.log(students);

        var length = students.length;

        console.log(length)

    </script>

**splice()** - добавление новых элементов в массив.

При использовании этого метода нужно указывать, куда именно следует добавить новые элементы.

В приведенном примере 2 — индекс, показывающий, где нужно вставить новые имена. Цифра 1 указывает, сколько элементов нужно удалить. Поскольку мы написали, что удалить нужно 1 элемент, во втором списке нет имени «Robert».

<script>

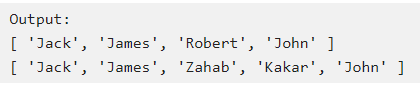
        var students = [ 'Jack', 'James', 'Robert', 'John'];

        console.log(students);

        students.splice(2, 1, "Zahab", "Kakar");

        console.log(students);

    </script>



**concat()** – Создание нового массива путем слияния двух существующих массивов

<script>

        var students = [ 'Jack', 'James', 'Rober', 'John'];

        console.log(students);

        var myFriends = ['Jennifer','Mary','Patricia']

        console.log(myFriends);

        var allNames =  students.concat(myFriends);

        console.log(allNames)

    </script>

**slice()**

* вырезает часть массива, начиная с указанного индекса

<script>

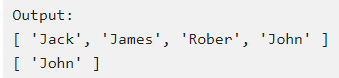
        var students = [ 'Jack', 'James', 'Rober', 'John'];

        console.log(students);

        var newStudent  = students.slice(3);

        console.log(newStudent);

    </script>



* принимает два аргумента, указывающих, с какого и до какого элемента нужно сделать срез

<script>

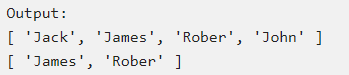
        var students = [ 'Jack', 'James', 'Rober', 'John'];

        console.log(students);

        var newStudent  = students.slice(1,3);

        console.log(newStudent);

    </script>



## Строки

**touppercase()** – переводит всю строку в верхний регистр

<script>

        var str = "Lorem Ipsum is simply dummy text of the printing and typesetting industry."

        var newStr = str.toUpperCase()

        console.log(newStr)

    </script>

**tolowercase()** - переводит всю строку в нижний регистр

<script>

        var str = "LOREM IPSUM IS SIMPLY DUMMY TEXT OF THE PRINTING AND TYPESETTING INDUSTRY. "

        var newStr = str.toLowerCase()

        console.log(newStr)

    </script>

replace() - используется для замены части строки другой строкой. Этот метод чувствителен к регистру.

<script>

        var str = "lorem ipsum is simply dummy text of the printing and typesetting industry"

        var newStr = str.replace("is", "are")

        console.log(newStr);

    </script>



**concat()** – Объединение двух строк в новую строку

<script>

        var str1 = "Java";

        var str2 = "Script";

        var str = str1.concat(str2);

        console.log(str)

    </script>

**trim()** – Удаление в начале и в конце строки, пробелов

<script>

        var str1 = "       JavaScript        ";

        str2 = str1.trim();

        console.log(str2);

    </script>

**split()** – Преобразование строки в массив

<script>

        var str = "JavaScript, is, nice"

        var newStr = str.split(",")

        console.log(newStr[0]);

    </script>

**length** – получение длины строки

<script>

        var str = "JavaScript is nice."

        var newStr = str.length

        console.log(newStr);

    </script>

## Математические (Math)

Math.abs(x) – Возвращает число по модулю

<script>

        let p = -25

        p = Math.abs(p)

        console.log(p)

    </script>

Math.acos(x) - Возвращает арккосинус числа

Math.asin(x) - Возвращает арксинус числа

Math.atan(x) - Возвращает арктангенс числа

Math.cos(x) - Возвращает косинус числа

Math.sin(x) - Возвращает синус числа

Math.tan(x) - Возвращает тангенс числа

Math.sinh(x) - Возвращает положительный квадратный корень числа

Math.cbrt(x) - Возвращает кубический корень числа

Math.ceil(x) - Возвращает значение числа, округлённое к большему целому

Math.floor(x) - Возвращает значение числа, округлённое к меньшему целому

Math.log(x) - Возвращает натуральный логарифм числа

Math.log10(x) - Возвращает десятичный логарифм числа

Math.log2(x) - Возвращает двоичный логарифм числа

Math.max([x[, y[, …]]]) - Возвращает наибольшее число из своих аргументов

<script>

        let mas=new Array()

        let max1,max2

        mas[1,2,3,40,20,30,35]

        max1=Math.max(mas)

        max2=Math.max(10,20)

        alert(max1)

        alert(max2)

    </script>

Math.min([x[, y[, …]]]) - Возвращает наименьшее число из своих аргументов

Math.pow(x, y) – Возвращает результат возведения числа x в степень y

<script>

        let a,b,c

        a=5

        b=3

        c=Math.pow(a,b)

        console.log(c)

    </script>

Math.trunc(x) – Возвращает значение только целой части числа, убирая дробные числа

### Рандомные числа с использованием метода Math

Math.random() - Возвращает псевдослучайное число в диапазоне от 0 до 1

Math.round(x) - Возвращает значение числа, округлённое до ближайшего целого

<script>

        let a=5.5625

        let b = Math.round(a)

        alert (b)

    </script>

#### Рандомные числа в заданном интервале

**Для вещественных чисел**

<script>

    let min = 0

    let max = 10

    max = max + 1

    let p = Math.random() \* (max - min) + min;

    alert(p)

</script>

**Для целых чисел**

<script>

        let min = 0

        let max = 10

        max = max + 1

        let p = Math.random() \* (max - min) + min;

        p = Math.floor(p)

        alert(p)

    </script>

# Циклы

## While

Цикл **while** будет выполняться до тех пор, пока верно (истинно) выражение, переданное ему параметром.

**Синтаксис:**

while (пока выражение истинно) {

            выполняем этот код циклически;

            в начале каждого цикла проверяем выражение в круглых скобках

        }

        /\*

            Цикл закончится, когда выражение перестанет быть истинным.

            Если оно было ложным изначально - то он не выполнится ни разу!

        \*/

Пример:

<script>

        let i = 0; //счетчик цикла

        while (i < 5) {

            /\*

                С помощью оператора ++ увеличиваем i на единицу

                при каждом проходе цикла.

            \*/

            i++;

            alert(i);

        }

    </script>

Переменная**i -** является так называемым счетчиком цикла. Счетчики используются для того, чтобы подсчитывать, сколько раз выполнился цикл.

## For

Цикл **for** является альтернативой **while**. Он более сложен для понимания, но чаще всего его любят больше, чем while за то, что он занимает меньше строчек.

**Синтаксис:**

for (начальные команды; условие окончания цикла; команды после прохода цикла) {

            тело цикла

        }

* **Начальные команды** - это то, что выполнится перед стартом цикла. Они выполнятся только один раз. Обычно там размещают начальные значения счетчиков, пример: i = 0.
* **Условие окончания цикла** - пока оно истинное, цикл будет работать, пример: i <10.
* **Команды после прохода цикла** - это команды, которые будут выполнятся каждый раз при окончании прохода цикла. Обычно там увеличивают счетчики, например: i++.

Пример:

<script>

        /\*

            В начале цикла i будет равно нулю,

            цикл будет выполнятся пока i < 10,

            после каждого прохода к i прибавляется единица:

        \*/

        for (var i = 0; i < 10; i++) {

            alert(i); //выведет 0, 1, 2... 9

        }

    </script>

## Цикл for для массивов

С помощью цикла for можно последовательно перебрать элементы массива.

Пример:

<script>

        var arr = [1, 2, 3, 4, 5];

        for (var i = 0; i <= arr.length - 1; i++) {

            alert(arr[i]); //выведет 1, 2, 3, 4, 5

        }

    </script>

**Ключевым моментом является то, что мы делаем перебор от нуля до длины массива минус 1 (так как номер последнего элемента массива на единицу меньше его длины).**

## Инструкция break

Для того, чтобы дострочно остановить выполнение цикла можно применить инструкцию – **break** (если выполнение цикла дойдет до нее, цикл закончит свою работу ).

# Использование массивов

- способ хранения подобных данных списка

## Объявление и вывод массива:

<script>

    let mass1=[]; //Первый способ объявления пустого массива

    let mass2=new Array(); //Второй способ объявления пустого массива

    let mass3=["Петя","Дима","Сережа"]; //Объявление массива с данными

</script>

<script>

alert(mass1+"\n"+mass2+"\n"+mass3); //Вывод всех элементов массива: //mass1,mass2,mass3

alert(mass3[2]); //Вывод элемента массива: mass3 с индексом - 2

</script>

## Ключи элементов массива

Вместо индексов элемента, можно использовать ключи (имена). Они служат для того, чтобы упростить для разработчика хранение и обработку информации в массива, путем обращения к элементу через имя.

<script>

let mass=new Array();

mass={"Андей":15,"Петя":20,"Ваня":30,"Сережа":18,} // Объявление массива с //ключами: Андрей,Петя,Ваня,Сережа

alert(mass["Ваня"]); //Вывод элемента массива с ключом: Ваня. Выведится элемент - //30

</script>

## Заполнение массива

Для того, чтобы внести данные в ячейку, необходимо обратится к его индексу или его ключу

    <script>

        let mass1 = new Array();

        mass = ["Андрей", "Петя", "Ваня", "Сережа"];

        mass[mass.length - 1] = "Денис"; //Изменяем последний элемент массива

        alert(mass[3]);

        alert(mass);

    </script>

## Очистка массива

Для того чтобы полностью стереть все данные, включая длину массива, необходимо указать, что длина массива равна нулю.

Пример:

    <script>

        let mass=new Array(1,2,45,30,75); //Объявление массива

        alert (mass); //Вывод массива с данными

        mass.length=0; //Обнуление массива

        alert (mass); //Вывод пустого массива

    </script>