

# ОСНОВЫ веб-технологий

HTML & CSS

# CSS flexbox

**CSS flexbox** (*Flexible Box Layout Module*) — модуль макета гибкого контейнера — представляет собой способ компоновки элементов, в основе лежит идея оси.

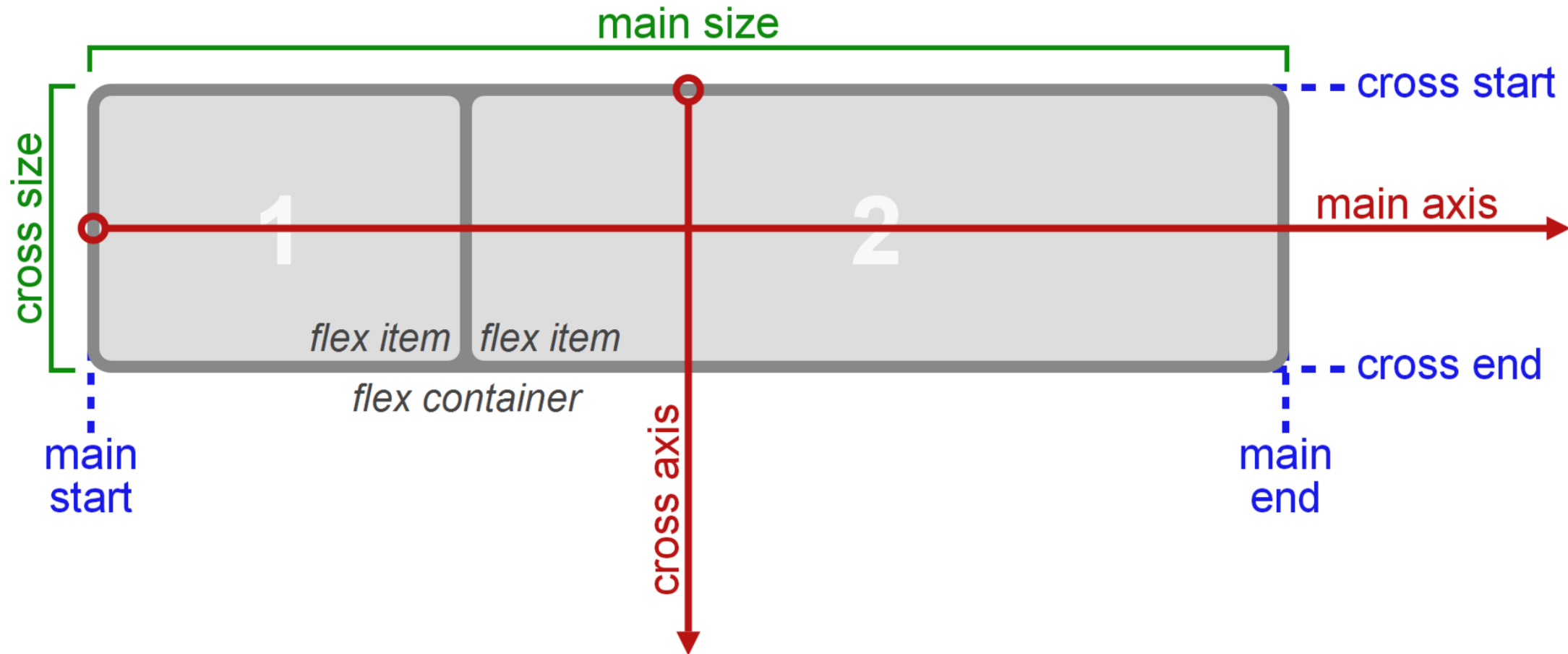
Flexbox состоит из **гибкого контейнера (flex container)** и **гибких элементов (flex items)**. Гибкие элементы могут выстраиваться в строку или столбик, а оставшееся свободное пространство распределяется между ними различными способами.

Flexbox используется для создания одномерных макетов, например, навигационной панели, так как flex-элементы можно размещать только по одной из осей.

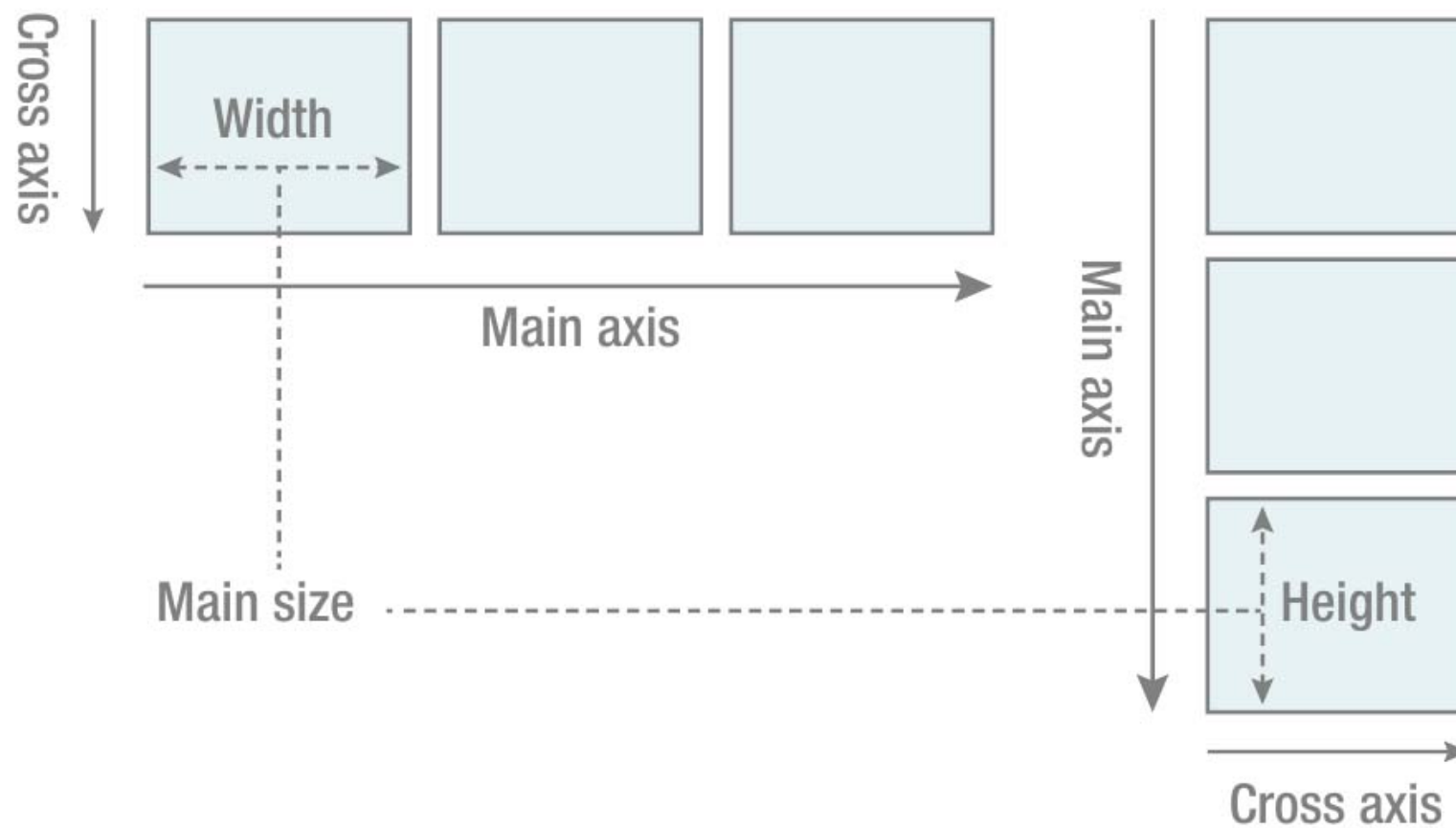
Модуль flexbox позволяет решать следующие задачи:

- Располагать элементы в одном из четырех направлений: слева направо, справа налево, сверху вниз или снизу вверх.
- Переопределять порядок отображения элементов.
- Автоматически определять размеры элементов таким образом, чтобы они вписывались в доступное пространство.
- Решать проблему с горизонтальным и вертикальным центрированием.
- Переносить элементы внутри контейнера, не допуская его переполнения.
- Создавать колонки одинаковой высоты.

# Flexbox Layout



# Flexbox Layout: режим строки и колонки



# Flex-контейнер

Flex-контейнер устанавливает новый гибкий контекст форматирования для его содержимого. Flex-контейнер не является блочным контейнером, поэтому для дочерних элементов не работают такие CSS-свойства, как `float`, `clear`, `vertical-align`.

Модель `flexbox`-разметки связана с определенным значением CSS-свойства `display` родительского `html`-элемента, содержащего внутри себя дочерние блоки. Для управления элементами с помощью этой модели нужно установить свойство `display` следующим образом:

```
.flex-container {  
/*генерирует flex-контейнер уровня блока*/  
  display: flex;  
}  
.flex-container {  
/*генерирует flex-контейнер уровня строки*/  
  display: inline-flex;  
}
```

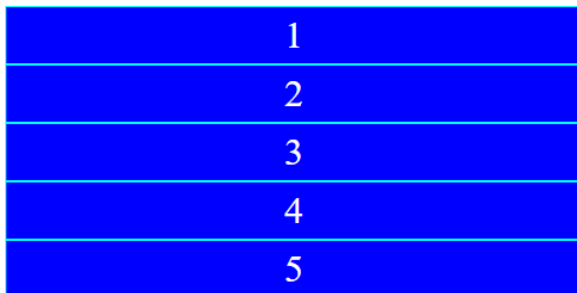
После установки данных значений свойства каждый дочерний элемент автоматически становится `flex`-элементом, выстраиваясь в один ряд (вдоль главной оси). При этом блочные и строчные дочерние элементы ведут себя одинаково, т.е. ширина блоков равна ширине их содержимого с учетом внутренних полей и рамок элемента.

Если родительский блок содержит текст или изображения без оберток, они становятся анонимными `flex`-элементами. Текст выравнивается по верхнему краю блока-контейнера, а высота изображения становится равной высоте блока, т.е. оно деформируется.

# Создание Flex контейнера

```
<div class="flex-container">
  <div class="flex-item">1</div>
  <div class="flex-item">2</div>
  <div class="flex-item">3</div>
  <div class="flex-item">4</div>
  <div class="flex-item">5</div>
</div>
```

```
.flex-item{
  background-color: blue;
  border: 1px solid cyan;
  color: white;
  padding: 5px;
  font-size: 2em;
  text-align: center;
}
.flex-container{
  display: flex;
}
```



`display: flex;`



# Направление главной оси: flex-direction

Свойство относится к flex-контейнеру. Управляет направлением главной оси, вдоль которой укладываются flex-элементы, в соответствии с текущим режимом записи.

- ❑ `flex-direction: row;` /\* Значение по умолчанию, слева направо (в rtl справа налево). Flex-элементы выкладываются в строку. Начало (main-start) и конец (main-end) направления главной оси соответствуют началу (inline-start) и концу (inline-end) оси строки (inline-axis).\*/
- ❑ `flex-direction: row-reverse;` /\* Направление справа налево (в rtl слева направо). Flex-элементы выкладываются в строку относительно правого края контейнера (в rtl — левого).\*/
- ❑ `flex-direction: column;` /\* Направление сверху вниз. Flex-элементы выкладываются в колонку.\*/
- ❑ `flex-direction: column-reverse;` /\* Колонка с элементами в обратном порядке, снизу вверх.\*/

# Выравнивание по главной оси: justify-content

Свойство выравнивает flex-элементы по главной оси flex-контейнера, распределяя свободное пространство, незанятое flex-элементами. Когда элемент преобразуется в flex-контейнер, flex-элементы по умолчанию сгруппированы вместе (если для них не заданы поля margin). Промежутки добавляются после расчета значений margin и flex-grow. Если какие-либо элементы имеют ненулевое значение flex-grow или margin: auto;, свойство не будет оказывать влияния.

- ❑ `justify-content: flex-start;` – значение по умолчанию. Flex-элементы выкладываются в направлении, идущем от начала flex-контейнера.
- ❑ `justify-content: flex-end;` – flex-элементы размещаются в конце flex-контейнера.
- ❑ `justify-content: center;` – flex-элементы выравниваются по центру flex-контейнера.
- ❑ `justify-content: space-between;` – flex-элементы равномерно распределяются по линии. Первый flex-элемент помещается вровень с краем начальной линии, последний flex-элемент — вровень с краем конечной линии, а остальные flex-элементы на линии распределяются так, чтобы расстояние между любыми двумя соседними элементами было одинаковым. Если оставшееся свободное пространство отрицательно или в строке присутствует только один flex-элемент, это значение идентично параметру `flex-start`.
- ❑ `justify-content: space-around;` – flex-элементы на линии распределяются так, чтобы расстояние между любыми двумя смежными flex-элементами было одинаковым, а расстояние между первым / последним flex-элементами и краями flex-контейнера составляло половину от расстояния между flex-элементами.



# Выравнивание по главной оси: justify-content

**justify-content: start;**



**justify-content: end;**



**justify-content: center;**



**justify-content: space-between;**



**justify-content: space-around;**



# Выравнивание по поперечной оси: `align-items` и `align-self`

Flex-элементы можно выравнивать по поперечной оси текущей строки flex-контейнера. `align-items` устанавливает выравнивание для всех элементов flex-контейнера, включая анонимные flex-элементы. `align-self` позволяет переопределить это выравнивание для отдельных flex-элементов. Если любое из поперечных `margin` flex-элемента имеет значение `auto`, `align-self` не имеет никакого влияния.

- ❑ `align-items: flex-start`; верхний край flex-элемента помещается вплотную с flex-линией (или на расстоянии, с учетом заданных полей `margin` и рамок `border` элемента), проходящей через начало поперечной оси.
- ❑ `align-items: flex-end`; нижний край flex-элемента помещается вплотную с flex-линией (или на расстоянии, с учетом заданных полей `margin` и рамок `border` элемента), проходящей через конец поперечной оси.
- ❑ `align-items: center`; поля flex-элемента центрируются по поперечной оси в пределах flex-линии.
- ❑ `align-items: stretch`; если поперечный размер flex-элемента вычисляется как `auto` и ни одно из поперечных значений `margin` не равно `auto`, элемент растягивается. Значение по умолчанию.

# Выравнивание по поперечной оси: align-items и align-self

**align-items: stretch;**



**align-items: flex-start;**



**align-items: flex-end;**



**align-items: center;**



# Управление многострочностью flex-контейнера: flex-wrap

Свойство определяет, будет ли flex-контейнер однострочным или многострочным, а также задает направление поперечной оси, определяющее направление укладки новых линий flex-контейнера.

**flex-wrap: nowrap;** /\* Значение по умолчанию. Flex-элементы не переносятся, а располагаются в одну линию слева направо (в rtl справа налево).\*/

**flex-wrap: wrap;** /\* Flex-элементы переносятся, располагаясь в несколько горизонтальных рядов (если не помещаются в один ряд) в направлении слева направо (в rtl справа налево).\*/

**flex-wrap: wrap-reverse;** /\* Flex-элементы переносятся на новые линии, располагаясь в обратном порядке слева-направо, при этом перенос происходит снизу вверх.\*/

# Краткая запись направления и многострочности: flex-flow

Свойство позволяет определить направления главной и поперечной осей, а также возможность переноса flex-элементов при необходимости на несколько строк. Представляет собой сокращённую запись свойств `flex-direction` и `flex-wrap`.

Значение по умолчанию `flex-flow: row nowrap;`

```
section {  
  display: flex;  
  flex-flow: row nowrap;  
}
```

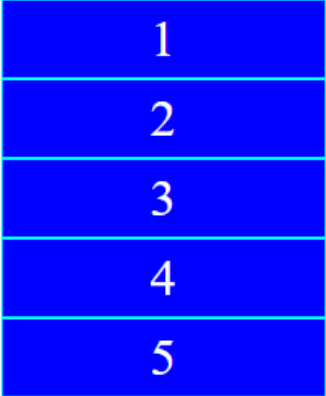
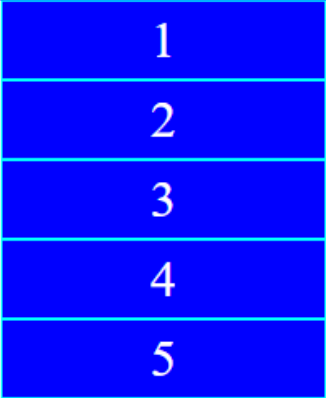
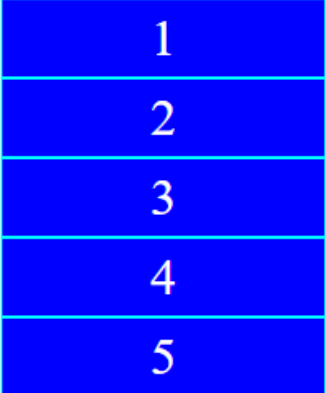

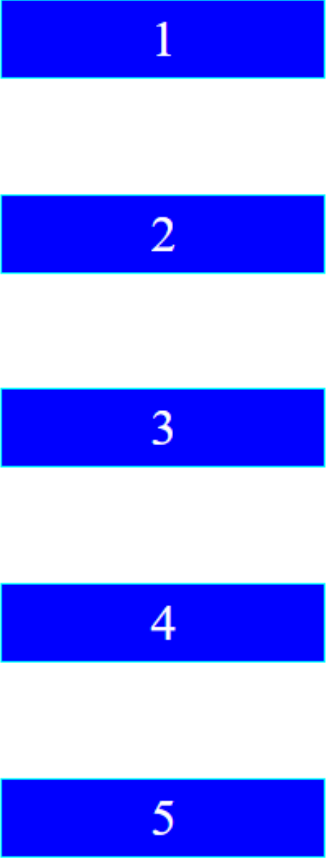
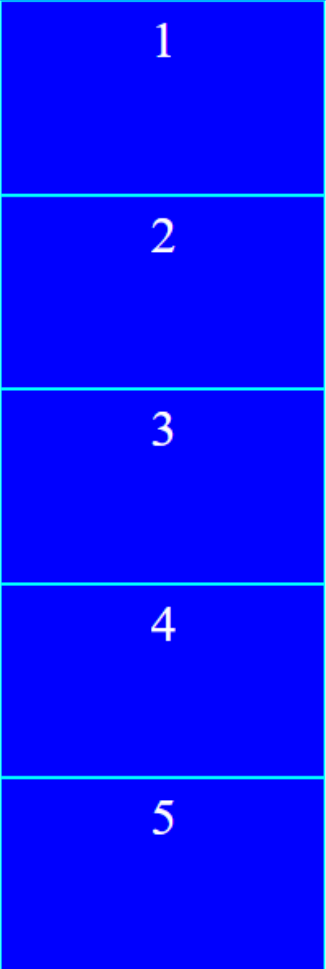
# Выравнивание строк flex-контейнера: align-content

Свойство `align-content` устанавливает тип выравнивания строк флекс-элементов по вертикали внутри флекс-контейнера, позволяя управлять свободным пространством.

Свойство `align-content` работает только в случае, если разрешен перенос строк и указано направление `flex-flow: row/row-reverse/column/column-reverse wrap/wrap-reverse`; и высота flex-контейнера.

- ❑ `align-content: flex-start`; строки укладываются по направлению к началу flex-контейнера. Край первой строки помещается вплотную к краю flex-контейнера, каждая последующая — вплотную к предыдущей строке.
- ❑ `align-content: flex-end`; строки укладываются по направлению к концу flex-контейнера. Край последней строки помещается вплотную к краю flex-контейнера, каждая предыдущая — вплотную к последующей строке.
- ❑ `align-content: center`; строки укладываются по направлению к центру flex-контейнера. Строки расположены вплотную друг к другу и выровнены по центру flex-контейнера с равным расстоянием между начальным краем содержимого flex-контейнера и первой строкой и между конечным краем содержимого flex-контейнера и последней строкой.
- ❑ `align-content: space-between`; строки равномерно распределены в flex-контейнере. Если оставшееся свободное пространство отрицательно или в flex-контейнере имеется только одна flex-линия, это значение идентично `flex-start`. В противном случае край первой строки помещается вплотную к начальному краю содержимого flex-контейнера, край последней строки — вплотную к конечному краю содержимого flex-контейнера. Остальные строки распределены так, чтобы расстояние между любыми двумя соседними строками было одинаковым.
- ❑ `align-content: space-around`; строки равномерно распределены в flex-контейнере с половинным пробелом на обоих концах. Если оставшееся свободное пространство отрицательно, это значение идентично `center`. В противном случае строки распределяются таким образом, чтобы расстояние между любыми двумя соседними строками было одинаковым, а расстояние между первой / последней строками и краями содержимого flex-контейнера составляло половину от расстояния между строками.
- ❑ `align-content: stretch`; значение по умолчанию. Строки флекс-элементов равномерно растягиваются, заполняя все доступное пространство.

# Выравнивание строк flex-контейнера: align-content

<code>align-content: center;</code>	<code>align-content: flex-start;</code>	<code>align-content: flex-end;</code>	<code>align-content: space-between;</code>	<code>align-content: space-around;</code>	<code>align-content: stretch;</code>
					

# Гибкое изменение размеров flex элементов

## Коэффициент роста: `flex-grow`

Свойство определяет коэффициент роста одного flex-элемента относительно других flex-элементов в flex-контейнере при распределении положительного свободного пространства. Свободное пространство — разница между размером flex-контейнера и размером всех его flex-элементов вместе. Если все элементы имеют одинаковый коэффициент `flex-grow`, то все они получают одинаковую долю свободного пространства, в противном случае оно распределяется в соответствии с соотношением, определённым различными коэффициентами `flex-grow`.

## Коэффициент сжатия: `flex-shrink`

Свойство указывает коэффициент сжатия flex-элемента относительно других flex-элементов при распределении отрицательного свободного пространства. Умножается на базовый размер `flex-basis`. Отрицательное пространство распределяется пропорционально тому, насколько элемент может сжаться, поэтому, например, маленький flex-элемент не уменьшится до нуля, пока не будет заметно уменьшен flex-элемент большего размера.

## Базовый размер: `flex-basis`

Свойство устанавливает начальный основной размер flex-элемента до распределения свободного пространства в соответствии с коэффициентами гибкости. Для всех значений, кроме `auto` и `content`, базовый гибкий размер определяется так же, как `width` в горизонтальных режимах записи. Процентные значения определяются относительно размера flex-контейнера, а если размер не задан, используемым значением для `flex-basis` являются размеры его содержимого.



# Гибкое изменение размеров flex элементов

Определяющим аспектом гибкого макета является возможность «сгибать» flex-элементы, изменяя их ширину/высоту (в зависимости от того, какой размер находится на главной оси), чтобы заполнить доступное пространство в основном измерении. Это делается с помощью свойства `flex`.

Свойство `flex` является сокращённой записью свойств `flex-grow`, `flex-shrink` и `flex-basis`.

Значение по умолчанию: `flex: 0 1 auto;`. Можно указывать как одно, так и все три значения свойств.

Flex-элемент будет полностью «негибок», если его значения `flex-grow` и `flex-shrink` равны нулю, и «гибкий» в противном случае.

W3C рекомендует использовать сокращённую запись, так как она правильно сбрасывает любые неуказанные компоненты, чтобы подстроиться под типичное использование.

```
article:nth-of-type(1){
  flex: 1 0 200px;
}
article:nth-of-type(2){
  flex: 2 1 200px;
}
article:nth-of-type(3){
  flex: 3 1 200px;
}
```

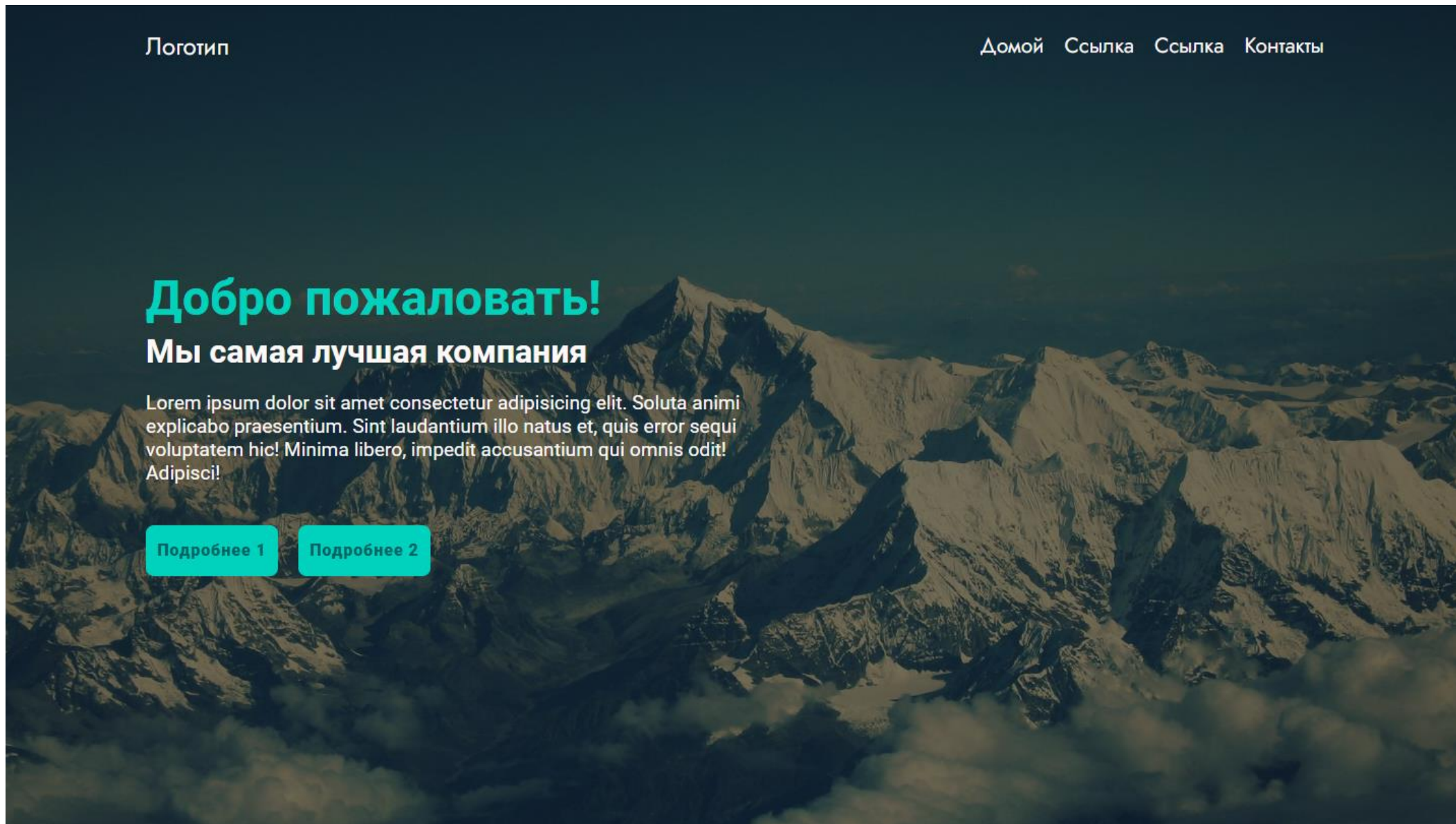
# Порядок отображения flex-элементов: order

Свойство определяет порядок, в котором flex-элементы отображаются и располагаются внутри flex-контейнера. Применяется к flex-элементам.

Первоначально все flex-элементы имеют `order: 0`; При указании значения от `-1` для элемента он перемещается в начало строки, значение `1` — в конец. Если несколько flex-элементов имеют одинаковое значение `order`, они будут отображаться в соответствии с исходным порядком.

```
article:nth-of-type(2){  
    order: -1;  
}
```

# Пример использования Flex box



# Пример использования Flex box

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>000 Ромашка</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header class="header">
    <a href="#" class="logo">Логотип</a>
    <nav class="navbar">
      <a href="#" class="active">Домой</a>
      <a href="#">Ссылка</a>
      <a href="#">Ссылка</a>
      <a href="#">Контакты</a>
    </nav>
  </header>
  <main class="home">
    <div class="home-content">
      <h1>Добро пожаловать!</h1>
      <h2>Мы самая лучшая компания</h2>
      <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Soluta animi explicabo praesentium. Sint laudantium illo natus et, quis
error sequi voluptatem hic! Minima libero, impedit accusantium qui omnis odit! Adipisci!</p>
      <div class="home-btn-group">
        <a href="#" class="btn">Подробнее 1</a>
        <a href="#" class="btn">Подробнее 2</a>
      </div>
    </div>
  </main>
</body>
</html>
```

# Пример использования Flex box

```
:root{
  --home-bg-color: rgb(0, 100, 100);
  --home-main-color: rgb(255, 250, 250);
  --home-add-color: rgb(0, 210, 190);
}

* {
  margin: 0;
  padding: 0;
  font-family: 'Roboto', sans-serif;
  box-sizing: border-box;
}

body{
  background-color: var(--home-bg-color);
  color: var(--home-main-color);
  background: linear-gradient(rgba(0, 0, 0, 0.6), rgba(0, 0, 0, 0.5)), url(bg.jpg);
  background-size: cover;
  background-position: center;
}

.header{
  position: absolute;
  width: 100%;
  min-height: 80px;
  padding: 0 10%;
  background-color: transparent;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.logo{
  font-size: 1.5rem;
  color: var(--home-main-color);
  text-decoration: none;
  font-family: 'Jost', sans-serif;
  transition: 0.3s;
}
```

```
.navbar{
  display: flex;
  flex-wrap: nowrap;
  background-color: transparent;
  justify-content: space-between;
  align-items: center;
  height: 100%;
}

.navbar a{
  color: var(--home-main-color);
  text-decoration: none;
  font-size: 1.3rem;
  font-family: 'Jost', sans-serif;
  transition: 0.3s;
  padding: 0 10px;
}

.navbar a:hover, .logo:hover{
  color: var(--home-add-color);
}

.home{
  height: 100vh;
  padding: 0 10%;
  display: flex;
  align-items: center;
}

.home-content{
  max-width: 600px;
}

.home-content h1{
  font-size: 3rem;
  font-weight: 700;
  color: var(--home-add-color);
  margin-bottom: 5px;
}
```

```
.home-content h2{
  font-size: 2rem;
  font-weight: 700;
  margin-bottom: 20px;
}

.home-content p{
  font-size: 1.2rem;
  margin-bottom: 40px;
}

.home-btn-group{
  height: 50px;
  width: 280px;
  display: flex;
  justify-content: space-between;
}

.home-btn-group .btn{
  text-decoration: none;
  background-color: var(--home-add-color);
  color: var(--home-bg-color);
  border: 2px solid var(--home-add-color);
  border-radius: 8px;
  font-size: 1rem;
  font-weight: 900;
  letter-spacing: 1px;
  width: 130px;
  height: 100%;
  display: inline-flex;
  align-items: center;
  justify-content: center;
  transition: 0.3s;
}

.home-btn-group .btn:hover{
  background-color: transparent;
  color: var(--home-add-color);
}
```

# Основные виды верстки сайта

## **Адаптивная**

Основные параметры сайта подстраиваются под характеристики пользовательского устройства. Ресурс в результате должен удобно читаться практически с любого гаджета или компьютера.

В процессе адаптивной верстки сайта HTML код пишется под несколько вариантов разрешений экрана. Как минимум, страницы адаптируют под ПК, ноутбуки, планшеты и смартфоны. Используемые при этом брейк-поинты (точки слома) сообщают браузеру места кода, где сайт необходимо трансформировать для обеспечения максимальной читаемости.

## **Мобильная**

Сайт создается на выделенном поддомене и адаптируется только под мобильные устройства. Соответственно, ресурс станет удобен для чтения лишь пользователям смартфонов и планшетов. Мобильная верстка сайта по сравнению с адаптивной менее затратна. Выбирать этот вариант целесообразно, когда использование ресурса с компьютера или ноутбука изначально не предусматривается.

## **Отзывчивая (Responsive)**

Во многом напоминает адаптивную верстку, но в то же время принципиально отличается от нее. Отличие заключается в действиях на брейк-поинтах. Если в адаптивном варианте браузер резко меняет сверстаный макет, то в отзывчивом происходит плавное изменение верстки между точками слома с максимальным сохранением стабильного вида страницы. Фактически данный подход сочетает в себе признаки адаптивности и «резиновости».

## **Резиновая**

Ширина блока задается в процентах от ширины экрана пользователя. Благодаря этому «резиновые» сайты одинаково хорошо читаются с любых устройств. Тем не менее, адаптивная верстка, равно как и отзывчивая, постепенно вытесняет резиновую.

## **Фиксированная**

В данном случае макет разрабатывается с фиксированными параметрами (например, с точным указанием размеров в пикселях). Такой подход устарел и применяется редко.



# Адаптивный и отзывчивый дизайн

- Отзывчивый (Responsive) дизайн (RWD) создается с гибкой сеткой или другими подходящими, но статическими свойствами, позволяющими одному шаблону быть актуальным для нескольких гаджетов.
- Адаптивный (Adaptive) дизайн (AWD) проектируется с условиями, изменяющимися, исходя из устройства серфинга, в его базе есть уже ряд макетов, он динамический и ориентирован на диагональ экранов.

Во случае адаптивного дизайна во главу угла ставится функциональность – особенности устройств учитываются в обязательном порядке. От размеров дисплея напрямую зависит характер расположения блоков. При разработке отзывчивого дизайна ведущую роль играют медиазапросы и относительная сетка, заданная в процентах.

Есть разница и в работе серверных скриптов: при AWD они сначала определяют, с чего заходит человек, а потом загружают оптимизированную под устройство версию страницы.

Что лучше использовать? Зависит от структуры web-ресурса: если она «резиновая», отзывчивость будет весьма удобной, вот только от нее может снизиться скорость загрузки (при наличии большого количества визуальных элементов). Адаптивность сайта – это мощный инструмент во всех остальных случаях: он развязывает дизайнерам руки, дает возможность разделить пользователей на категории и предложить каждой ориентированные на нее решения. Проще говоря, владельцы смартфонов будут видеть один макет, созданный именно для их удобства, планшетов – другой, стационарных ПК – третий.



# Медиазапросы

Одним из самых важных инструментов при создании адаптивной вёрстки является использование медиазапросов. Медиазапросы — специальные условные конструкции, которые позволяют применять стили только для определённых устройств.

Медиазапросы могут записываются как в css-файле, так и при подключении css-файла

## Использование внутри css-файла:

```
@media (условия) {  
    /* CSS-код, который будет выполнен для данного условия */  
}
```

В качестве условия могут выступать различные значения и константы.



# Медиазапросы: ориентация экрана

Для определения ориентации экрана используется ключ `orientation`, значением которого может выступать одно из двух значений:

- ❑ `landscape` — условие выполнится для устройств с горизонтальной ориентацией экрана. Горизонтальная — ориентация, при которой ширина viewport больше его высоты.
- ❑ `portrait` — условие выполнится для устройств с вертикальной ориентацией экрана. Вертикальная — ориентация, при которой высота viewport больше его ширины.

```
@media (orientation: landscape) {  
    /* При горизонтальной ориентации фоновым цветом сайта будет белый */  
    body {  
        background: #FFF;  
    }  
}
```

# Медиазапросы: разрешение экрана

При использовании медиазапросов мы также можем исходить из ширины или высоты `viewport`. Для этого используются знакомые нам по обычным CSS-правилам условия `width`, `min-width`, `max-width` для ширины и `height`, `min-height`, `max-height` для высоты.

С помощью таких условий создаются `breakpoint` — контрольные точки. Это границы значений, по которым видоизменяется наш макет. Такие точки остановки позволяют иметь правила для мониторов, планшетов, телефонов и иных смарт-устройств.

CSS является каскадной таблицей, поэтому порядок стилей необходимо контролировать. Существует два подхода — `Desktop First` и `Mobile First`.

**Desktop First.** Вначале пишутся стили для больших мониторов, а в последствии, используя медиазапросы, дописываются стили для всё более маленьких значений `viewport`. Его характерная черта в медиазапросах — использование конструкции `max-width` в качестве условия.

**Mobile First.** Его особенностью является то, что вначале пишутся стили под мобильные устройства, а затем, используя медиазапросы, пишутся стили для больших размеров `viewport`. Если в `Desktop First` основной конструкцией являлось использование `max-width`, то в `Mobile First` используется `min-width`.

# Медиазапросы: разрешение экрана

```
<style>
  /* Здесь будут все стили для устройств с
  viewport больше 1400 пикселей. */

  @media (max-width: 1400px) {
    /* Стили для устройств, у которых ширина
    viewport меньше или равно 1400 пикселей, но
    больше 990 пикселей. Эти стили будут
    использованы для планшетов и ноутбуков с низким
    разрешением */
  }

  @media (max-width: 990px) {
    /* Стили для устройств, у которых ширина
    viewport меньше или равно 990 пикселей, но
    больше 770 пикселей. Эти стили подойдут для
    некоторых мобильных устройств и планшетов */
  }

  @media (max-width: 770px) {
    /* Стили для устройств, у которых ширина
    viewport меньше или равно 770 пикселей. Это
    множество мобильных устройств */
  }
</style>
```

```
<style>
  /* Здесь будут все стили для мобильных
  устройств с viewport меньше 770 пикселей. */

  @media (min-width: 770px) {
    /* Стили для устройств, у которых ширина
    viewport больше или равно 770 пикселей*/
  }

  @media (min-width: 990px) {
    /* Стили для устройств, у которых ширина
    viewport больше или равно 990 пикселей, но
    меньше 1400 пикселей */
  }

  @media (min-width: 1400px) {
    /* Стили для устройств, у которых ширина
    viewport больше или равно 1400 пикселей*/
  }
</style>
```

# Медиазапросы: логические операторы

Условия внутри медиазапросов можно комбинировать. Для этого существует три логических оператора: И, ИЛИ, НЕ.

**Логическое «И».** Означает, что несколько условий должны быть выполнены для того, чтобы CSS-стили применились к элементу. Для использования логического «И» используется ключевое слово `and`.

```
@media (orientation: portrait) and (min-width: 600px) { }
```

**Логическое «ИЛИ».** Свойства применятся в том случае, если хотя бы одно из условий будет выполнено. Условия для этого отделяются запятыми.

```
@media (orientation: portrait), (min-width: 600px) { }
```

**Логическое «НЕ».** Свойства применятся в том случае, если условие не выполняется. Используется ключевое слово `not`. Реальное использование этого оператора не велико, в виду сложности и не интуитивности происходящего.

# Медиазапросы: типы устройств

В `@media` можно указывать определённые типы устройств:

- ❑ `all` — для всех;
- ❑ `print` — для принтеров и в режиме предварительного просмотра страницы перед печатью;
- ❑ `screen` — для устройств с экранами;
- ❑ `speech` — для программ чтения с экрана.

```
@media screen { ... }
```

```
@media screen, print { ... }
```

# Использование медиазапросов при подключении CSS

Медиазапросы возможно писать не только внутри CSS-файла, но и использовать их в HTML при подключении файла стилей. В этом случае медиазапрос указывается в атрибуте `media`.

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Подключение CSS файлов</title>
  <!-- Общие стили для проекта -->
  <link rel="stylesheet" href="style.css">
  <!-- Стили для экранов с viewport не менее 750px -->
  <link rel="stylesheet" media="screen and (min-width: 750px)" href="style750px.css">
</head>
```

Кроме `<link>`, их также можно использовать в `@import`:

```
@import url(mobile.css) screen and (max-width: 991.98px);
@import url(desktop.css) screen and (min-width: 992px);
```

# Встраиваемое содержимое

Встраиваемое содержимое — содержимое, которое импортируется в документ из других источников, другого языка разметки или пространства имён. Некоторые элементы могут иметь резервный контент, который будет задействован, если внешний ресурс не может быть использован.

Вы можете встраивать видео, аудио, pdf-документы и т.п. как из внутренних, так и внешних источников — приложений или сайтов.

# Элемент <iframe>

Элемент <iframe> используется для встраивания другого HTML-документа в текущий, при этом он полностью изолирован от JavaScript и CSS родительского элемента.

```
<iframe src="path" height="500px" width="500px"></iframe>
```

Ограничение отображения страниц сайта во фреймах сторонних доменов позволяет защитить ресурс от Clickjacking'a и существенно снижает риск проведения Cross-site scripting атак.



# Повышение безопасности iframe

- Используйте только при необходимости
- Используйте HTTPS
- Используйте атрибут `sandbox`, который устанавливает дополнительные ограничения для любого содержимого, размещенного в `<iframe>`

Разрешенные значения `sandbox` :

`allow-forms` — разрешает содержимому фрейма отправлять формы. Если это ключевое слово не используется, отправка формы блокируется.

`allow-modals` — разрешает ресурсу открывать модальные окна (например, с помощью `alert`).

`allow-orientation-lock` — разрешает ресурсу блокировать ориентацию экрана.

`allow-pointer-lock` — разрешает ресурсу блокировать курсор мыши.

`allow-popups` — разрешает всплывающие окна.

`allow-popups-to-escape-sandbox` — разрешает всплывающим окнам не наследовать режим песочницы (например, чтобы в всплывающем окне, в котором есть JavaScript, он был разрешен без `allow-scripts` у фрейма).

`allow-same-origin` — разрешает загрузку содержимого фрейма (воспринимает это как контент из того же источника, что и родительский документ). Блокирует всплывающие окна, поэтому может использоваться для безопасного открытия контента.

`allow-scripts` — разрешает ресурсу запускать сценарии (но не создавать всплывающие окна).

`allow-top-navigation` — разрешает открывать ссылку фрейма в родительском документе.

`allow-top-navigation-by-user-activation` — разрешает открывать ссылку фрейма, нажатую пользователем, в родительском документе.

`allow-downloads` — позволяет выполнять загрузки пользователя.

Синтаксис: `sandbox="allow-forms allow-modals"`

# Пример: youtube и карты

## Youtube

Сначала перейдите на <https://www.youtube.com/> и найдите понравившееся вам видео.

Под видео вы найдёте кнопку «Поделиться» - нажмите, чтобы отобразить параметры совместного доступа.

Выберите кнопку «Встроить», и вам будет предоставлен код `<iframe>` - скопируйте его.

## Google map

Откройте <https://www.google.ru/maps/>

Откройте меню/ссылка/код

Выберите встраиваемое содержимое

Скопируйте код

## Yandex map

Откройте карту <https://yandex.ru/maps>

Выберите поделиться в дополнительном меню

Скопируйте текст для виджета с картой

# Элемент <picture>

Элемент `picture` — ещё один инструмент, который предоставляет дополнительные возможности управления выбором изображений. В отличие от `srcset` и `sizes`, элемент `picture` подразумевает ручной выбор изображения в зависимости от условий.

Внутри `picture` через дочерние элементы `source` можно указывать изображения и медиаусловия, при которых они будут выбраны:

```
<picture>
  <source srcset="/img/300x350.png" media="(max-width: 350px)" >
  <source srcset="/img/800x630.png" media="(max-width: 800px)" >
  <source srcset="/img/2000x1000.png" media="(max-width: 1000px)" >
  <source srcset="/img/3000x1500.png" media="(min-width: 1001px)" >
  
</picture>
```

Элемент `img` обязателен — через него задаются размеры, стили и прочие атрибуты изображения. Если браузер не поддерживает `picture` или `source`, подставляется изображение из `img`.

Всё это напоминает `srcset` и `sizes`, но есть принципиальное отличие. В `srcset` и `sizes` мы указываем условия и целевой размер изображения. Подбором наиболее подходящего изображения из доступных занимается браузер. В `source` мы явно указываем ресурс, который должен быть показан при выполнении условия.

# Элемент <picture>

Можно одновременно менять изображение с помощью медиаусловий и дать браузеру возможность выбирать из вариантов изображения. Для этого нужно добавить варианты и описать критерии подстановки с помощью `sizes` в `source`:

```
<picture>
  <source srcset="/img/300x350.png" media="(max-width: 350px)" />
  <source srcset="/img/800x630.png" media="(max-width: 800px)" />
  <source srcset="/img/2000x1000.png 2x,
                /img/1000x500.png 1x"
        sizes="1000px"
        media="" />
  
</picture>
```

Если задача переключения изображений не стоит, то от `picture` будет не очень много пользы. В таких случаях лучше использовать `srcset` и `sizes`.

# Элемент <audio>

Элемент <audio> используется для внедрения звукового контента в веб-страницы. В общем виде HTML-разметка имеет следующий вид:

```
<audio src="name.ogg" controls></audio>
```

Атрибут controls добавляет отображение браузерами интерфейса управления аудио плеера — кнопки воспроизведения, паузы, громкости.

В настоящий момент не существует аудио формата, который бы работал во всех браузерах, поэтому для обеспечения доступности контента максимально широкой аудитории рекомендуется включать несколько источников звука, представленных с использованием атрибута src элемента <source>. Одновременно можно добавить резервный контент для браузеров, которые не поддерживают элемент <audio>.

```
<audio controls>
  <source src="name.ogg" type="audio/ogg">
  <source src="name.mp3" type="audio/mpeg">
  <a href="name.mp3">Скачать name.mp3</a>
</audio>
```

# Элемент <audio>. Атрибуты

- `controls` указывает браузеру, что нужно отобразить базовые элементы управления воспроизведением (начинать и останавливать воспроизведение, переходить в другое место записи, регулировать громкость).
- `loop` циклическое воспроизведение аудио файла.
- `muted` выключает звук при воспроизведении аудио файла.
- `preload` атрибут, отвечающий за предварительную загрузку аудио контента. Не является обязательным, некоторые браузеры игнорируют его. Возможные значения:
  - `auto` — браузер загружает аудио файл полностью, чтобы он был доступен, когда пользователь начнет его воспроизведение.
  - `metadata` — браузер загружает первую небольшую часть аудио файла, чтобы определить его основные характеристики.
  - `none` — отсутствие автоматической загрузки аудио файла.

Стандарт `html` поддерживает только аудиоформаты `mp3`, `wav` и `ogg`.

# Элемент <video>

HTML-видео — новый стандарт для размещения мультимедийных файлов в сети с оригинальным программным интерфейсом без привлечения подключаемых модулей. С помощью элемента <video> появилась возможность добавлять видеосодержимое на веб-страницы, а также стилизовать внешний вид видеоплеера при помощи CSS-стилей.

В простом варианте HTML-разметка для размещения видеофайла на странице имеет следующий вид:

```
<video src="video.ogv" controls></video>
```

Атрибут controls отвечает за появление элементов управления видеоплеером. Вы можете добавить изображение с помощью атрибута poster, которое браузер будет использовать, пока загружается видео или пока пользователь не нажмет на кнопку воспроизведения, а также задать высоту и ширину видео.

Как и в случае с аудиофайлами, рекомендуется перечислять в <source> все форматы, начиная с более предпочтительного. Также нужно указывать MIME-тип для каждого видеофайла.

```
<video controls width="400" height="300">  
  <source src="video.mp4" type="video/mp4">  
  <source src="video.webm" type="video/webm">  
  <source src="video.ogv" type="video/ogg">  
</video>
```

# Элемент <video>

На данный момент браузеры поддерживают три основных видео формата:

Формат	Видеокодк	Аудиокодк
.mp4	H.264	AAC
.ogg/.ogv	Theora	Vorbis
.webm	VP8	Vorbis

```
<video controls width="710" height="538" poster="/media/poster.png" preload="none">  
  <source src="/media/video.mp4" type="video/mp4">  
</video>
```