# 13. CSS Flexbox: система компоновки элементов на веб-странице

Долгое время единственными надёжными инструментами CSS вёрстки были такие способы как **Float** (обтекание) и **позиционирование**.

С их помощью сложно или невозможно достичь следующих простых требований к макету:

* Вертикального выравнивания блока внутри родителя.
* Оформления всех детей контейнера так, чтобы они распределили между собой доступную ширину/высоту, независимо от того, сколько ширины/высоты доступно.
* Сделать все колонки в макете одинаковой высоты, даже если наполнение в них различно

CSS Flexbox — это технология для создания сложных гибких макетов за счёт правильного размещения элементов на странице.

**Основные преимущества flexbox**

1. Все блоки очень легко делаются “резиновым”, что уже следует из названия “flex”. Элементы могут сжиматься и растягиваться по заданным правилам, занимая нужное пространство.
2. Выравнивание по вертикали и горизонтали, базовой линии текста работает шикарно.
3. Расположение элементов в html не имеет решающего значения. Его можно поменять в CSS. Это особенно важно для некоторых аспектов responsive верстки.
4. Элементы могут автоматически выстраиваться в несколько строк/столбцов, занимая все предоставленное место.
5. Множество языков в мире используют написание справа налево rtl (right-to-left), в отличии от привычного нам ltr (left-to-right). Flexbox адаптирован для этого. В нем есть понятие начала и конца, а не права и лева. Т.е. в браузерах с локалью rtl все элементы будут автоматически расположены в реверсном порядке.
6. Синтаксис CSS правил очень прост и осваивается довольно быстро.

**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.

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.
* Создавать прижатый к низу страницы подвал сайта.

## 13.1. Основные понятия

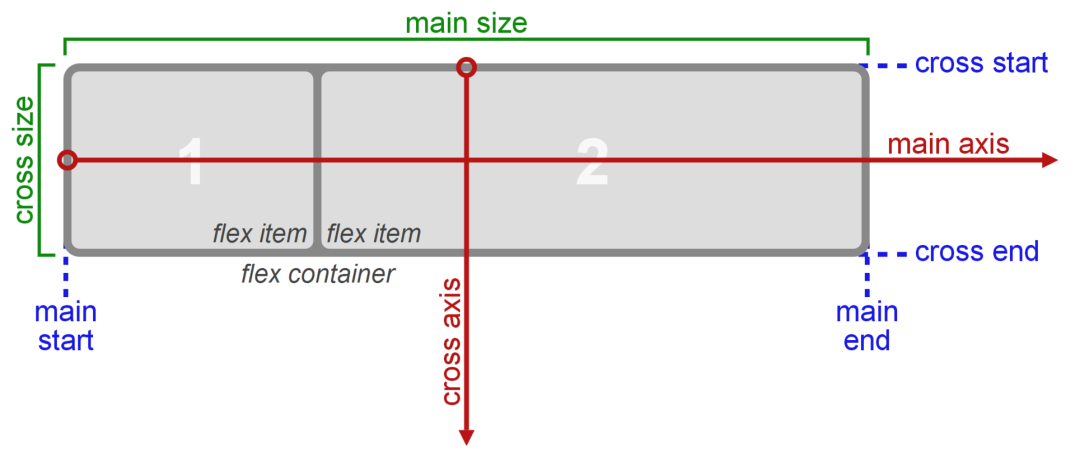


РИС. 1. МОДЕЛЬ FLEXBOX

**Главная ось (main axis)** — ось, вдоль которой выкладываются flex-элементы. Она простирается в основном измерении.

**Main start и main end** — линии, которые определяют начальную и конечную стороны flex-контейнера, относительно которых выкладываются flex-элементы (начиная с main start по направлению к main end).

**Основной размер (main size**) — ширина или высота flex-контейнера или flex-элементов, в зависимости от того, что из них находится в основном измерении, определяют основной размер flex-контейнера или flex-элемента.

**Поперечная ось (cross axis)** — ось, перпендикулярная главной оси. Она простирается в поперечном измерении.

**Cross start и cross end** — линии, которые определяют начальную и конечную стороны поперечной оси, относительно которых выкладываются flex-элементы.

**Поперечный размер (cross size)** — ширина или высота flex-контейнера или flex-элементов, в зависимости от того, что находится в поперечном измерении, являются их поперечным размером.

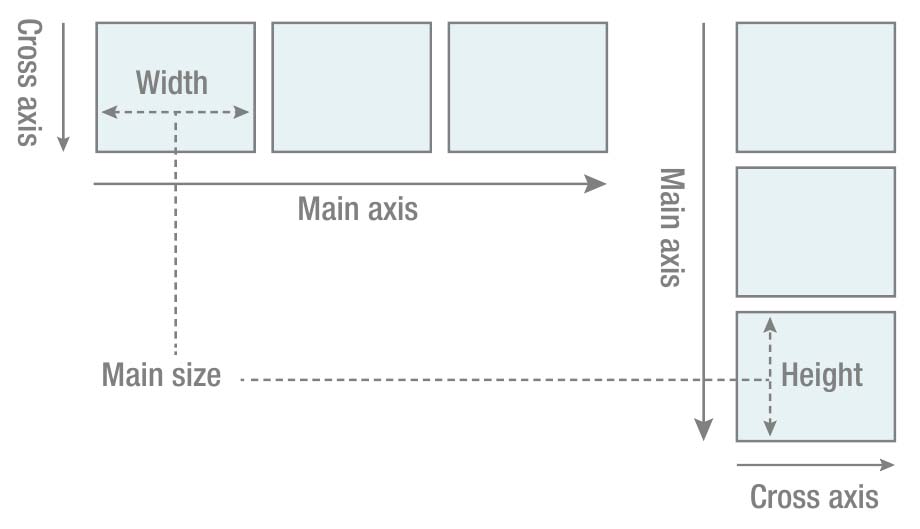


РИС. 2. РЕЖИМ СТРОКИ И КОЛОНКИ

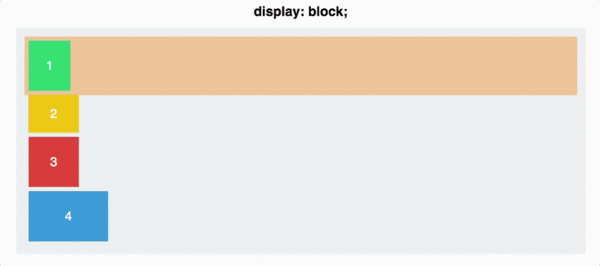
## 13.2. Flex-контейнер

Flexbox определяет набор CSS свойств для контейнера (***flex-контейнер***) и его дочерних элементов (***flex-блоков***). Первое, что нужно сделать – это указать контейнеру ***display:flex*** или ***display:inline-flex***.

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. Также, на flex-контейнер не оказывают влияние свойства column-\*, создающие колонки в тексте и псевдоэлементы ::first-line и ::first-letter.

Модель flexbox-разметки связана с определенным значением CSS-свойства display родительского html-элемента, содержащего внутри себя дочерние блоки.

Например, есть страница и на ней размещено 4 div разных размеров, которые находятся внутри серого div. У каждого div есть свойство display: block. Поэтому каждый блок занимает всю ширину строки.



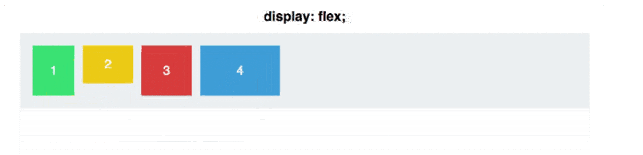
Чтобы начать работать с CSS Flexbox, нужно сделать контейнер *flex-контейнером*. Делается это так:

#container {

display: flex;

}

Так у блоков появилось свойство flex-контекст, которое в дальнейшем позволит ими управлять гораздо проще, чем с использованием стандартного CSS.



После установки данного значения 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.

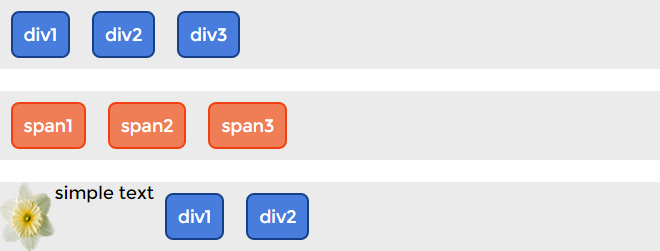


РИС. 3. ВЫРАВНИВАНИЕ ЭЛЕМЕНТОВ В МОДЕЛИ FLEXBOX

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.

## 13.3. Flex-элементы

Flex-элементы — блоки, представляющие *содержимое* flex-контейнера в потоке. Flex-контейнер устанавливает новый контекст форматирования для своего содержимого, который обуславливает следующие особенности:

* Для flex-элементов блокируется их значение свойства display. Значение display: inline-block; и display: table-cell; вычисляется в display: block;.
* Пустое пространство между элементами исчезает: оно не становится своим собственным flex-элементом, даже если межэлементный текст обернут в анонимный flex-элемент. Для содержимого анонимного flex-элемента невозможно задать собственные стили, но оно будет наследовать их (например, параметры шрифта) от flex-контейнера.
* Абсолютно позиционированный flex-элемент не участвует в компоновке гибкого макета.
* Поля margin соседних flex-элементов не схлопываются.
* Процентные значения margin и padding вычисляются от внутреннего размера содержащего их блока.
* margin: auto; расширяются, поглощая дополнительное пространство в соответствующем измерении. Их можно использовать для выравнивания или раздвигания смежных flex-элементов.
* Автоматический минимальный размер flex-элементов по умолчанию является минимальным размером его содержимого, то есть min-width: auto;. Для контейнеров с прокруткой автоматический минимальный размер обычно равен нулю.

## 13.4. Порядок отображения flex-элементов и ориентация

Содержимое flex-контейнера можно разложить в любом направлении и в любом порядке (переупорядочение flex-элементов внутри контейнера влияет только на визуальный рендеринг).

### 13.4.1. Направление главной оси: *flex-direction*

По умолчанию все предметы располагаются вдоль главной оси — слева направо. Именно поэтому блоки в предыдущем примере выстроились в линию, когда мы применили display: flex. А вот flex-direction позволяет вращать главную ось.

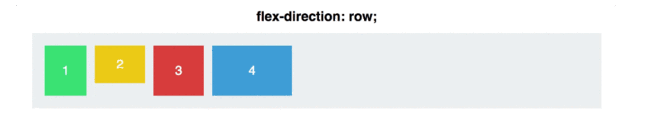
#container {

display: flex;

flex-direction: column;

}





Обратите внимание, что flex-direction: column не выравнивает блоки по оси, перпендикулярной главной. Главная ось сама меняет своё расположение, и теперь направлена сверху вниз.

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

|  |  |
| --- | --- |
| **flex-direction** | |
| Значения: |  |
| 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). |
| row-reverse | Направление справа налево (в rtl слева направо). Flex-элементы выкладываются в строку относительно правого края контейнера (в rtl — левого). |
| column | Направление сверху вниз. Flex-элементы выкладываются в колонку. |
| column-reverse | Колонка с элементами в обратном порядке, снизу вверх. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

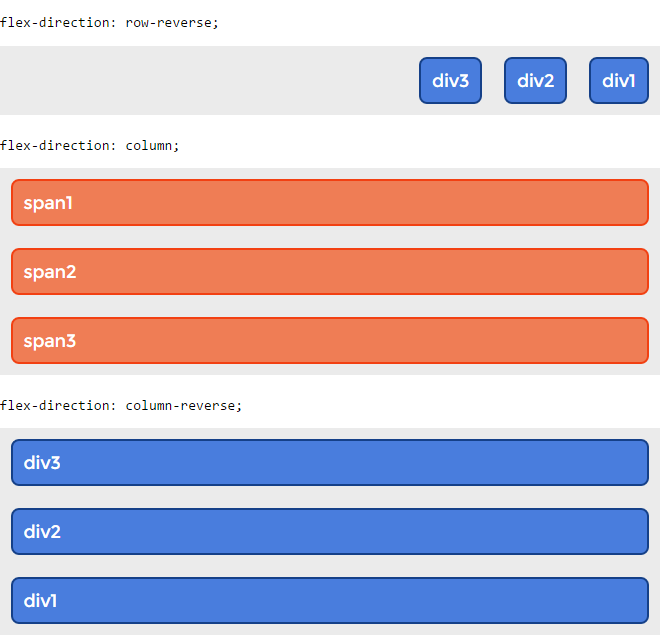


РИС. 4. СВОЙСТВО FLEX-DIRECTION ДЛЯ LEFT-TO-RIGHT ЯЗЫКОВ

**Синтаксис** CSS

.flex-container {

display: -webkit-flex; *//https://itchief.ru/html-and-css/vendor-prefixes*

-webkit-flex-direction: row-reverse;

display: flex;

flex-direction: row-reverse;

}

## 13.5. Выравнивание во Flexbox

В **Flexbox** есть свойства, которые начинаются с align- или с justify-. Важно помнить следующее:

* justify- *выполняет выравнивание по главной оси* в том же направлении, что и flex-direction;
* align- выполняет выравнивание *по оси поперёк* направлению flex-direction.

### 13.5.1. Выравнивание по главной оси: *justify-content*

Свойство ***justify-content*** 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. Свойство не наследуется.

Обратите внимание, что свойство justify-content производит видимый эффект только **при наличии свободного места**. Поэтому если у вас набор элементов занимает всё пространство на главной оси, justify-content ничего не даст.

Justify-content может принимать 5 значений:

1. flex-start
2. flex-end
3. center
4. space-between
5. space-around

|  |  |
| --- | --- |
| **justify-content** | |
| Значения: |  |
| flex-start | Значение по умолчанию. Flex-элементы выкладываются в направлении, идущем от начальной линии flex-контейнера. |
| flex-end | Flex-элементы выкладываются в направлении, идущем от конечной линии flex-контейнера. |
| center | Flex-элементы выравниваются по центру flex-контейнера. |
| 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. |
| 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. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

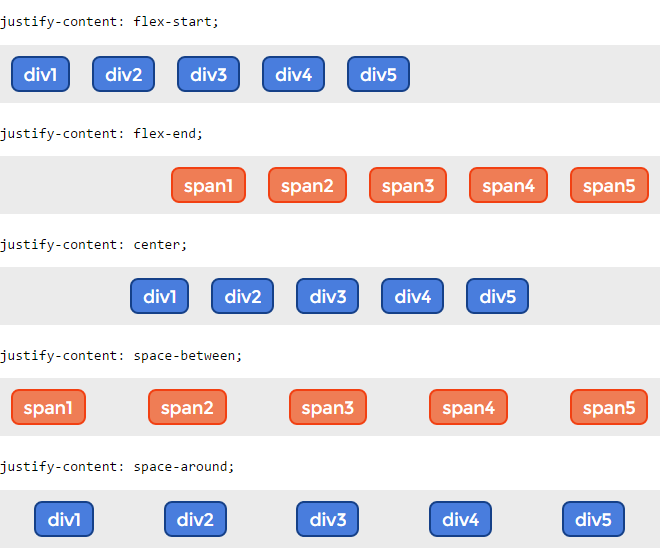


РИС. 9. ВЫРАВНИВАНИЕ ЭЛЕМЕНТОВ И РАСПРЕДЕЛЕНИЕ СВОБОДНОГО ПРОСТРАНСТВА С ПОМОЩЬЮ СВОЙСТВА JUSTIFY-CONTENT

**Синтаксис** CSS

#container {

display: flex;

flex-direction: row;

justify-content: flex-start;

}

Space-between задаёт одинаковое расстояние между блоками, но не между контейнером и блоками. Space-around также задаёт одинаковое расстояние между блоками, но теперь расстояние между контейнером и блоками равно половине расстояния между блоками.

### 13.5.2. Выравнивание по поперечной оси: *align-items* и *align-self*

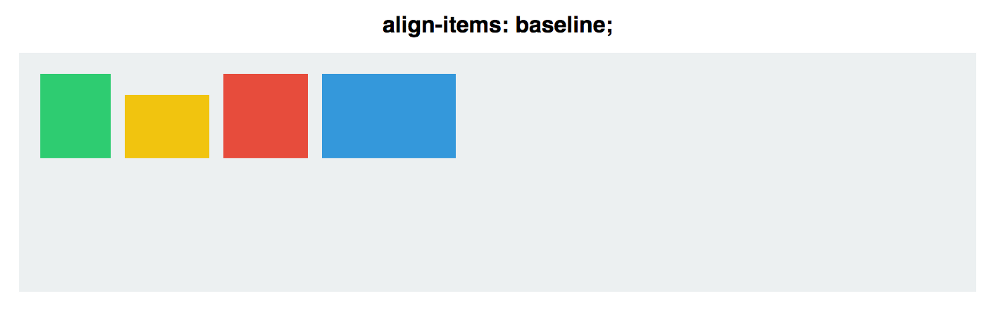
Если justify-content работает с *главной осью*, то align-items работает с осью, *перпендикулярной главной оси*, т.е. flex-элементы можно выравнивать по поперечной оси текущей строки flex-контейнера. align-items устанавливает выравнивание для всех элементов flex-контейнера, включая анонимные flex-элементы. 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.

**13.5.2.1. Align-items**

Свойство **align-items** выравнивает flex-элементы, в том числе и анонимные flex-элементы по *поперечной оси*. Вернёмся к flex-direction: row и пройдёмся по командам align-items

|  |  |
| --- | --- |
| **align-items** | |
| Значения: |  |
| 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. |
| 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. |
| center | Поля flex-элемента центрируется по поперечной оси в пределах flex-линии. |
| baseline | Базовые линии всех flex-элементов, участвующих в выравнивании, совпадают. |
| stretch | Если поперечный размер flex-элемента вычисляется как auto и ни одно из поперечных значений margin не равно auto, элемент растягивается. Значение по умолчанию. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Стоит заметить, что для align-items: stretch высота блоков должна быть равна auto. Для align-items: baseline теги параграфа убирать не нужно, иначе получится так:



Чтобы получше разобраться в том, как работают оси, объединим justify-content с align-items и посмотрим, как работает выравнивание по центру для двух свойств flex-direction:

|  |  |
| --- | --- |
|  |  |

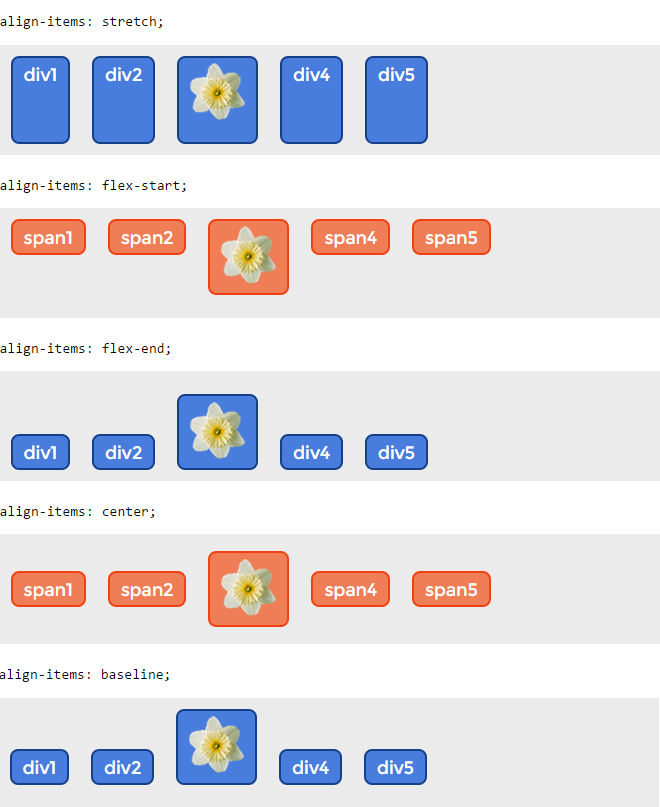


РИС. 10. ВЫРАВНИВАНИЕ ЭЛЕМЕНТОВ В КОНТЕЙНЕРЕ ПО ВЕРТИКАЛИ

**Синтаксис** CSS

.flex-container {

display: flex;

align-items: flex-start;}

**13.5.2.2. Align-self**

Свойство **align-self** отвечает за выравнивание *отдельно* взятого *flex-элемента* по высоте flex-контейнера. Переопределяет выравнивание, заданное align-items.

Позволяет выравнивать элементы по отдельности:

#container {

align-items: flex-start;

}

.square#one {

align-self: center;

}

// Only this square will be centered.

|  |  |
| --- | --- |
| **align-self** | |
| Значения: |  |
| auto | Значение по умолчанию. Flex-элемент использует выравнивание, указанное в свойстве align-items flex-контейнера. |
| 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. |
| 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. |
| center | Поля flex-элемента центрируется по поперечной оси в пределах flex-линии. |
| baseline | Flex-элемент выравнивается по базовой линии. |
| stretch | Если поперечный размер flex-элемента вычисляется как auto и ни одно из поперечных значений margin не равно auto, элемент растягивается. Значение по умолчанию. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

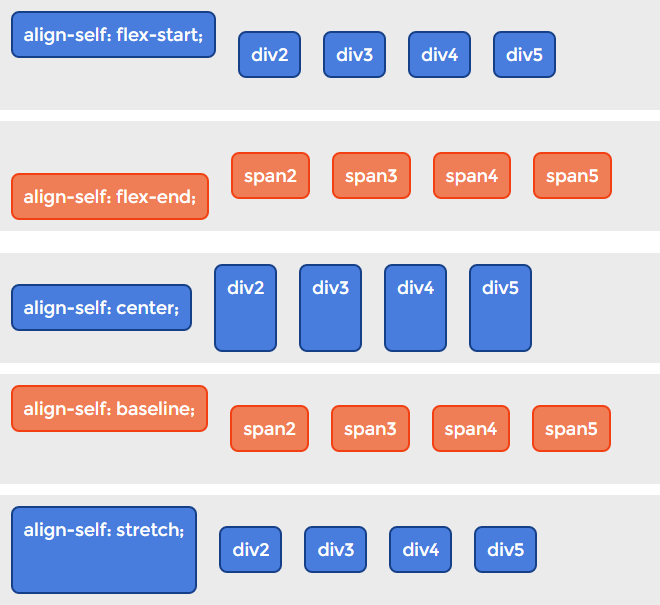


РИС. 11. ВЫРАВНИВАНИЕ ОТДЕЛЬНЫХ FLEX-ЭЛЕМЕНТОВ

**Синтаксис** CSS

.flex-container {

display: flex;

}

.flex-item {

align-self: center;

}

Для двух блоков применим align-self, а для остальных — align-items: center и flex-direction: row.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

### 13.5.3. Выравнивание строк flex-контейнера: *align-content*

Если в контейнере несколько осей и указано flex-wrap: wrap, то можно использовать align-content для выравнивания строк на поперечной оси. Но требуется дополнительное пространство.

Свойство ***align-content*** выравнивает строки в flex-контейнере при наличии дополнительного пространства на *поперечной оси*, аналогично выравниванию отдельных элементов на главной оси с помощью свойства justify-content. Свойство не влияет на однострочный flex-контейнер.

|  |  |
| --- | --- |
| **align-content** | |
| Значения: |  |
| 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. |
| 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. |
| 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. |
| 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. |
| space-around | Строки равномерно распределены в flex-контейнере с половинным пробелом на обоих концах. Если оставшееся свободное пространство отрицательно, это значение идентично центcenter. 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. |
| stretch | Значение по умолчанию. Строки flex-элементов равномерно растягиваются, заполняя все доступное пространство. Если оставшееся свободное пространство отрицательно, это значение идентично flex-start. В противном случае свободное пространство будет разделено поровну между всеми строками, увеличивая их поперечный размер. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

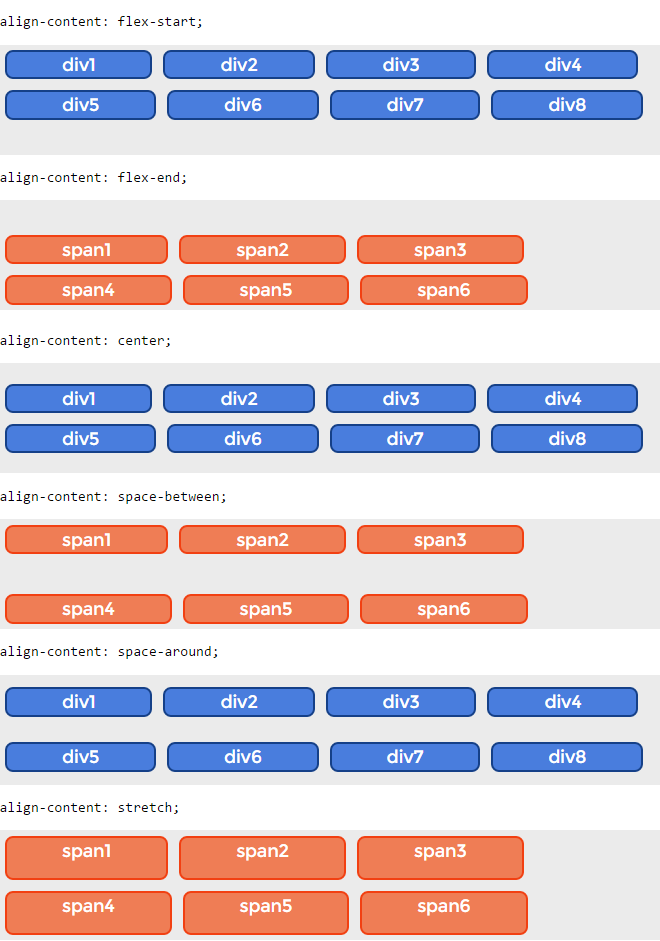


РИС. 12. МНОГОСТРОЧНОЕ ВЫРАВНИВАНИЕ FLEX-ЭЛЕМЕНТОВ

**Синтаксис**

.flex-container {

display: flex;

/\*flex-flow: row wrap;\*/

flex-wrap: wrap;

align-content: flex-end;

height: 100px;

}

### 13.5.4. 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.

По умолчанию flex-элементы укладываются в одну строку, вдоль главной оси. При переполнении они будут выходить за пределы ограничивающей рамки flex-контейнера.

|  |  |
| --- | --- |
| **flex-wrap** | |
| Значения: |  |
| nowrap | Значение по умолчанию. Flex-элементы не переносятся, а располагаются в одну линию слева направо (в rtl справа налево). |
| wrap | Flex-элементы переносятся, располагаясь в несколько горизонтальных рядов (если не помещаются в один ряд) в направлении слева направо (в rtl справа налево). |
| wrap-reverse | Flex-элементы переносятся на новые линии, располагаясь в обратном порядке слева-направо, при этом перенос происходит снизу вверх. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

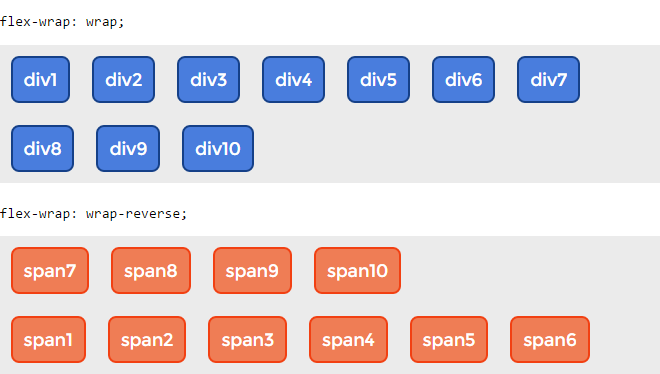


РИС. 5. УПРАВЛЕНИЕ МНОГОСТРОЧНОСТЬЮ С ПОМОЩЬЮ СВОЙСТВА FLEX-WRAP ДЛЯ LTR-ЯЗЫКОВ

**Синтаксис** CSS

.flex-container {

display: flex;

flex-wrap: wrap;}

### 13.5.5. Краткая запись направления и многострочности: *flex-flow*

Свойство *flex-flow* 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;.

|  |  |
| --- | --- |
| **flex-flow** | |
| Значения: |  |
| направление | Указывает направление главной оси. Значение по умолчанию row. |
| многострочность | Задаёт многострочность поперечной оси. Значение по умолчанию nowrap. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис** CSS

.flex-container {

display: -webkit-flex;

-webkit-flex-flow: row wrap;

display: flex;

flex-flow: row wrap;}

### 13.5.6. Порядок отображения flex-элементов: *order*

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

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

|  |  |
| --- | --- |
| **order** | |
| Значения: |  |
| число | Свойство задается целым числом, отвечающим за порядок отображения flex-элементов. Значение по умолчанию 0. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис** CSS

.flex-container {

display: flex;

}

.flex-item {

order: 1;}



РИС. 6. ПОРЯДОК ОТОБРАЖЕНИЯ FLEX-ЭЛЕМЕНТОВ

## 13.6. Гибкость 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. Flex-контейнер распределяет свободное пространство между своими дочерними элементами (пропорционально их коэффициенту flex-grow) для заполнения контейнера или сжимает их (пропорционально их коэффициенту flex-shrink), чтобы предотвратить переполнение.

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

### 13.6.1. Задание гибких размеров одним свойством: *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.

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

|  |  |
| --- | --- |
| **flex** | |
| Значения: | **Описание** |
| [flex-grow](https://msiter.ru/references/css-reference/flex-grow) коэффициент растяжения | Коэффициент увеличения *ширины* flex-элемента относительно других flex-элементов. |
| [flex-shrink](https://msiter.ru/references/css-reference/flex-shrink) коэффициент сужения | Коэффициент уменьшения *ширины* flex-элемента относительно других flex-элементов. |
| [flex-basis](https://msiter.ru/references/css-reference/flex-basis) базовая ширина | Базовая *ширина* flex-элемента. |
| auto | Эквивалентно flex: 1 1 auto;. |
| none | Эквивалентно flex: 0 0 auto;. |
| initial | Устанавливает значение свойства в значение по умолчанию. Эквивалентно flex: 0 1 auto;. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис** CSS

.flex-container {

display: flex;

}

.flex-item {

flex: 3 1 100px;}

### 13.6.2. Коэффициент роста: *flex-grow*

Свойство *flex-grow* определяет коэффициент роста одного flex-элемента относительно других flex-элементов в flex-контейнере при распределении положительного свободного пространства. Если сумма значений flex-grow flex-элементов в строке меньше 1, они занимают менее 100% свободного пространства.

Задает коэффициент увеличения элемента при наличии свободного места в контейнере. По умолчанию flex-grow: 0 т.е. никакой из элементов не должен увеличиваться и заполнять свободное место в контейнере.

По умолчанию **flex-grow: 0**

|  |  |
| --- | --- |
| **flex-grow** | |
| Значения: |  |
| число | Положительное целое или дробное число, устанавливающее коэффициент роста flex-элемента. Значение по умолчанию 0. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

Примеры:

* Если всем элементам указать flex-grow:1, то все они растянуться одинаково и заполнять все свободное место в контейнере.
* Если одному из элементов указать flex-grow:1, то он заполнит все свободное место в контейнере и выравнивания через justify-content работать уже не будут: свободного места нет выравнивать нечего...
* При flex-grow:1. Если один из них имеет flex-grow:2, то он будет в 2 раза больше, чем все остальные
* Если все flex-блоки внутри flex-контейнера имеют flex-grow:3, то они будут одинакового размера
* При flex-grow:3. Если один из них имеет flex-grow:12, то он будет в 4 раза больше, чем все остальные

Как это работает? Допустим, что контейнер имеет ширину 500px и содержит два элемента, каждый из которых имеет базовую ширину 100px. Значит, в контейнере остается 300 свободных пикселей. Теперь, если первому элементу укажем flex-grow:2;, а второму flex-grow: 1;, то блоки займут всю доступную ширину контейнера и ширина первого блока будет 300px, а второго 200px. Объясняется это тем, что доступные 300px свободного места в контейнере распределились между элементами в соотношении 2:1, +200px первому и +100px второму.

Заметка: в значении можно указывать дробные числа, например: flex-grow:0.5

Задавать значения надо отдельно для каждого элемента. Если задать одно значение – это для всех элементов.

**flex-grow: 0 0 0 0**



**flex-grow:  1 0 0 0**



**flex-grow: 1 2 0 0**



**flex-grow: 1 1 1 1**



**flex-grow: 1 1 3 1**



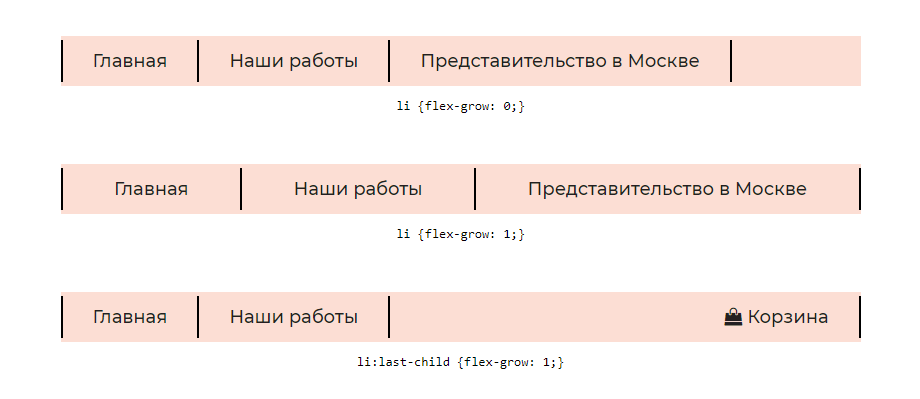
[](https://codepen.io/html5book/pen/KEqOJB)

РИС. 7. УПРАВЛЕНИЕ СВОБОДНЫМ ПРОСТРАНСТВОМ В ПАНЕЛИ НАВИГАЦИИ С ПОМОЩЬЮ FLEX-GROW

**Синтаксис** CSS

.flex-container {

display: flex;

}

.flex-item {

flex-grow: 3;}

### 13.6.3. Коэффициент сжатия: *flex-shrink*

Задает коэффициент уменьшения элемента. Свойство противоположное *flex-grow* и определяет как элемент должен сжиматься, если в контейнере не остается свободного места. Т.е. свойство начинает работать, когда сумма размеров всех элементов больше чем размер контейнера.

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. По умолчанию **flex-shrink:1**

Допустим, что контейнер имеет ширину 600px и содержит два элемента, каждый из которых имеет ширину 300px - flex-basis:300px;. Т.е. два элемента полностью заполняют контейнер. Первому элементу укажем flex-shrink: 2;, а второму flex-shrink: 1;. Теперь уменьшим ширину контейнера на 300px, т.е. элементы должны сжаться на 300px чтобы находится внутри контейнера. Сжиматься они будут в соотношении 2:1, т.е. первый блок сожмется на 200px, а второй на 100px и новые размеры элементов станут 100px и 200px.

Заметка: в значении можно указывать дробные числа, например: 0.5 - flex-shrink:0.5

|  |  |
| --- | --- |
| **flex-shrink** | |
| Значения: |  |
| число | Положительное целое или дробное число, устанавливающее коэффициент уменьшения flex-элемента. Значение по умолчанию 1. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**flex-shrink: 1 1 1 1**



**flex-shrink: 0 1 1 1**



**flex-shrink:  2 1 1 1**



**flex-shrink: 3 2 1 1**



**flex-shrink: 4 1 1 1**



Еще пример

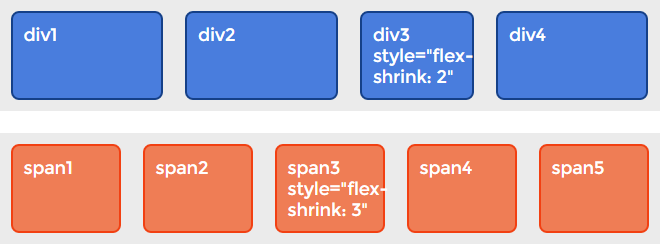


РИС. 8. СУЖЕНИЕ FLEX-ЭЛЕМЕНТОВ В СТРОКЕ

**Синтаксис** CSS

.flex-container {

display: flex;

}

.flex-item {

flex-shrink: 3;

}

### 13.6.4. Базовый размер: *flex-basis*

Отвечает за изначальный размер элементов до того, как они будут изменены другими свойствами CSS Flexbox. 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.

Значение можно указать в px, em, rem, %, vw, vh и т.д. Итоговая ширина будет зависеть от базовой ширины и значений flex-grow, flex-shrink и контента внутри блока. При auto элемент получает базовую ширину относительно контента внутри него.

По умолчанию: auto

|  |  |
| --- | --- |
| **flex-basis** | |
| Значения: |  |
| auto | Значение по умолчанию. Элемент получает базовый размер, соответствующий размеру его содержимого (если он не задан явно). |
| content | Определяет базовый размер в зависимости от содержимого flex-элемента. |
| длина | Базовый размер определяется так же, как для ширины и высоты. Задается в единицах длины. |
| initial | Устанавливает значение свойства в значение по умолчанию. |
| inherit | Наследует значение свойства от родительского элемента. |

**Синтаксис** CSS

.flex-container {

display: flex;

}

.flex-item {

flex-basis: 100px;}

flex-basis: auto auto auto auto



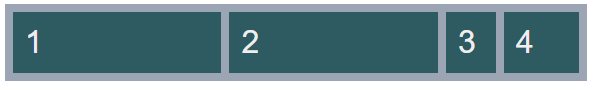
flex-basis:  50% auto auto auto



flex-basis:  50% 50% auto auto



flex-basis:  50% 50% 50px 100px



<https://medium.com/@stasonmars/%D1%80%D0%B0%D0%B7%D0%BD%D0%B8%D1%86%D0%B0-%D0%BC%D0%B5%D0%B6%D0%B4%D1%83-width-%D0%B8-flex-basis-f34e658ce6a2>

**Пример**

**<https://tuhub.ru/posts/flexbox-complete-guide>**

**flex: {grow shrink basis}**

Короткая запись трех свойств: flex-grow flex-shrink flex-basis.

По умолчанию: **flex: 0 1 auto**

Однако можно указать и одно, и два значения:

flex: none; /\* 0 0 auto \*/

/\* число \*/

flex: 2; /\* flex-grow (flex-basis переходит в 0) \*/

/\* не число \*/

flex: 10em; /\* flex-basis: 10em, есть единицы измерения только в basis \*/

flex: 30px; /\* flex-basis: 30px \*/

flex: auto; /\* flex-basis: auto \*/

flex: content; /\* flex-basis: content \*/

flex: 1 30px; /\* flex-grow и flex-basis \*/

flex: 2 2; /\* flex-grow и flex-shrink (flex-basis переходит в 0) \*/

flex: 2 2 10%; /\* flex-grow и flex-shrink и flex-basis \*/

**Чем отличается flex-basis от width?**

Ниже важные различия между flex-basis и width / height:

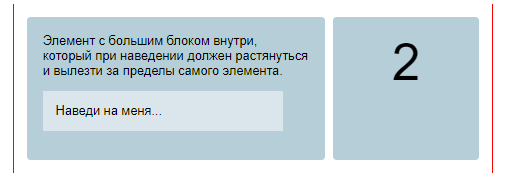
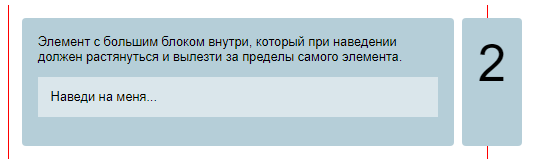
* flex-basis работает только для главной оси. Это значит что при flex-direction:row flex-basis контролирует ширину (width), а при flex-direction:column контролирует высоту (height**). [Смотрите пример](https://jsfiddle.net/kamaev/bgaeL4uw/" \t "_blank)**.
* flex-basis применяется только к flex элементам. А значит если отключить flex у контейнера это свойство не будет иметь эффекта.
* Абсолютные элементы контейнера не участвуют во flex конструкции... А значит, flex-basis не влияет на элементы flex контейнера, если они абсолютны position:absolute. Им нужно будет указать width / height.
* При использовании свойства flex 3 значения (flex-grow/flex-shrink/flex-basis) можно скомбинировать и записать коротко, а для width grow или shrink нужно писать отдельно. Например: flex:0 0 50% == width:50%; flex-shrink:0;. Иногда это просто неудобно.

По возможности все же отдавайте предпочтение flex-basis. Используйте width только когда не подходит flex-basis.

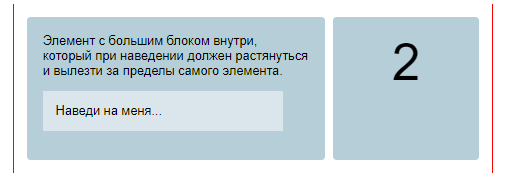
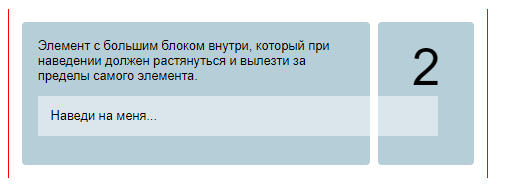
**Отличие flex-basis от width - баг или фича?**

Контент внутри flex элемента распирает его и не может выйти за его пределы. Однако если установить ширину через width или max-width, а не flex-basis, то элемент внутри flex контейнера сумеет выйти за пределы этого контейнера (иногда нужно именно такое поведение). Пример:

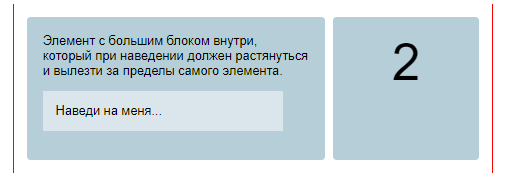
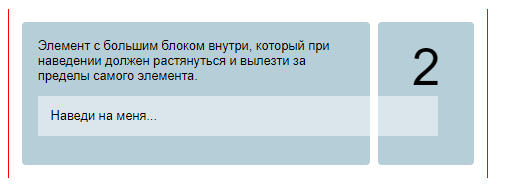
Ширина элементов установлена через: **flex-basis**

Ширина элементов установлена через:  **width**

Ширина элементов установлена через:  **max-width**

**Примеры Flex верстки**

**#1 Простой пример с выравниванием по вертикали и горизонтали**. Начнем с самого простого примера - выравнивание по вертикали и горизонтали одновременно и при любой высоте блока, даже резиновой.

<div class="parent">

<span class="child">Текст по середине</span>

</div>

<style>

.parent { display: flex; }

.child { margin: auto; }

</style>

Или так, без блока внутри:

<div class="center-text">

Текст по середине

</div>

<style>

.center-text {

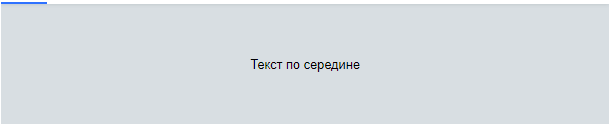
display: flex;

justify-content: center;

align-items: center;

}

</style>



**#2 Адаптивное меню на flex**

Сделаем меню в самом верху страницы. На широком экране оно должно быть справа. На среднем выравниваться по середине. А на маленьком каждый элемент должен быть на новой строке.

<div class="nav">

<a href="#">Главная</a>

<a href="#">О нас</a>

<a href="#">Продукты</a>

<a href="#">Контакты</a>

</div>

<style>

.nav {

display: flex;

justify-content: flex-end; /\* разместим справа \*/

background: #6e9cc3;

}

.nav a {

color:#fff; padding:15px 10px; text-decoration:none; text-align:center;

}

.nav a:hover { background:#5c8db7; }

/\* меньше 800px \*/

@media all and (max-width: 800px) {

.nav { justify-content: space-around; }

.nav a{ flex-grow:1; /\* растягиваться на всю ширину \*/ }

}

/\* меньше 500px \*/

@media all and (max-width: 500px) {

.nav { flex-direction: column; }

}

</style>

#### 

**#3 Адаптивные 3 колонки**

Этот пример показывает как быстро и удобно сделать 3 колонки, которые при сужении будут превращаться в 2 и затем в 1.

Обратите внимание, что сделать это можно без использования media правил, все на flex.

<div class="flex">

<div class="item">1</div>

<div class="item">2</div>

<div class="item">3</div>

<div class="item">4</div>

<div class="item">5</div>

<div class="item">6</div>

</div>

<style>

.flex{

display: flex;

flex-wrap: wrap;

max-width: 700px; /\* макс ширина \*/

margin: 0 auto; /\* выровняем по центру \*/

}

.item{

flex:1 1 calc(33.33% - 30px); /\* отнимем margin и скажем растягиваться \*/

margin:5px;

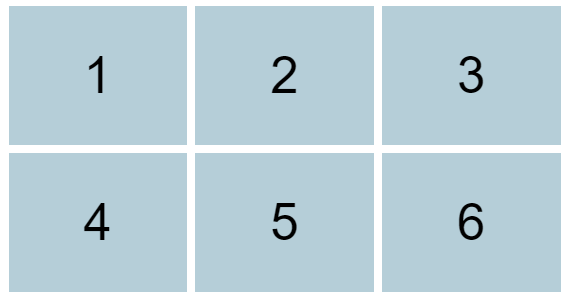
box-sizing:border-box; /\* чтобы внутренний отступ не влиял когда там будет текст... \*/

min-width:170px; /\* мин. ширина блока, чтобы переносились на другой ряд \*/

padding:50px 20px; font-size:400%; text-align:center; background:#b5ced8; /\* для красоты \*/

}

</style>



**#4 Адаптивные блоки на flex**

Допустим нам нужно вывести 3 блока, один большой и два маленьких. При этом нужно чтобы блоки подстраивались под маленькие экраны. Делаем:

<div class="flex">

<div class="mainitem">1</div>

<div class="sideitem">

<div class="item">2</div>

<div class="item">3</div>

</div>

</div>

<style>

.flex{

display: flex;

max-width: 700px; /\* макс ширина блока \*/

margin: 0 auto; /\* выравниваем блок по центру \*/

font-size:400%;

}

.mainitem, .item{

flex-grow:1; /\* растягивание элементов \*/

background:#b5ced8;

display: flex; justify-content: center; align-items: center; /\* цифры по центру \*/

}

.mainitem{

min-height:300px; /\* высота основного блока \*/

}

.sideitem{

/\* flex:0 0 150px; \*/

flex-basis:150px; /\* ширина 150 \*/

flex-shrink:0; /\* Убираем уменьшение = 150 \*/

display: flex;

flex-direction: column;

}

.item{

flex-basis:50%;

min-height:150px;

margin-left:10px;

}

.item:first-child{ margin-bottom:10px; }

/\* меньше 600px \*/

@media screen and (max-width: 600px) {

.flex{ flex-wrap: wrap; } /\* для переноса \*/

.sideitem{ flex-direction:row; flex-basis:100%; }

.item{ margin-left:0; margin-top:10px; }

.item:first-child{ margin-bottom:0; margin-right:10px; }

}

/\* меньше 450px \*/

@media screen and (max-width: 450px) {

.sideitem{ flex-wrap: wrap; } /\* разрешаем перенос \*/

.item{ flex-basis:100%; } /\* на всю ширину \*/

.item:first-child{ margin-right:0; }

}

</style>



**#5 Галерея на flex и transition**

Этот пример показывает как быстро можно сделать симпатичный аккордеон с картинками на flex. Обратите внимание на свойство transition для flex.

<div class="flex">

<div class="item img1"></div>

<div class="item img2"></div>

<div class="item img3"></div>

<div class="item img4"></div>

<div class="item img5"></div>

</div>

<style>

.flex {

display: flex;

overflow:hidden; /\*скроем тень\*/

}

.item {

height:300px;

flex:20%; /\* = flex-basis:20%; \*/

transition: flex 300ms ease;

box-shadow: 0 0 19px 3px #000; margin: 1px; /\* красивости тени \*/

}

.item:hover {

flex: 0.1 0.1 400px;

background-size: 100% 100%

}

.img1 { background: url('http://lorempixel.com/400/300/cats/1') 0 0 no-repeat; }

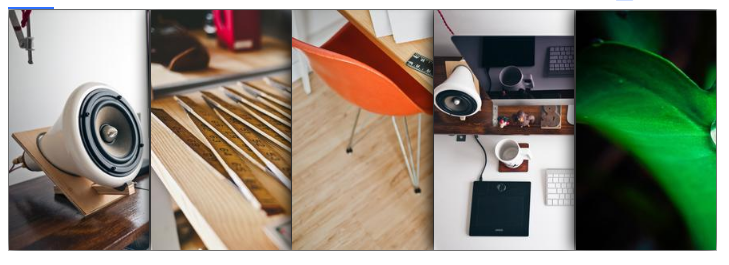
.img2 { background: url('http://lorempixel.com/400/300/cats/2') 0 0 no-repeat; }

.img3 { background: url('http://lorempixel.com/400/300/cats/3') 0 0 no-repeat; }

.img4 { background: url('http://lorempixel.com/400/300/cats/4') 0 0 no-repeat; }

.img5 { background: url('http://lorempixel.com/400/300/cats/5') 0 0 no-repeat; }

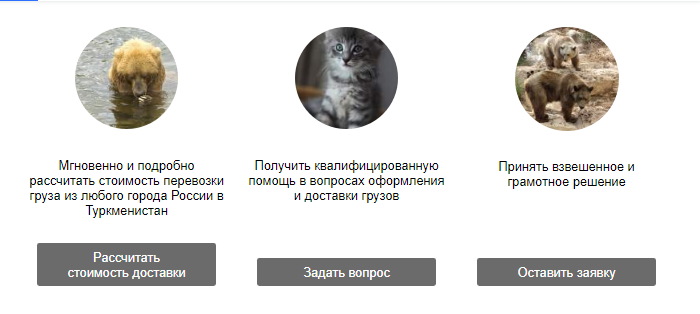
</style>



**#6 Флекс во флекс (просто пример)**

Задача сделать гибкий блок. Так чтобы начало текста в каждом блоке находилось на одной линии по горизонтали. Т.е. при сужении ширины, блоки растут в высоту. Нужно, чтобы картинка была вверху, кнопка всегда внизу, а текст по середине начинался по одной горизонтальной линии...

Для решения этой задачи, сами блоки растягиваются флексом и им установлена максимально возможная ширина. Каждый внутренний блок также является флекс конструкцией, с повернутой осью flex-direction:column; и элемент в середине (где находится текст) растягивается flex-grow:1; чтобы заполнить всё свободное пространство, так достигается результат - текст начинался с одной линии...



# 14.Валидация CSS

Валидация (проверка) кода CSS на соответствие стандартам имеет определенные преимущества и ничем не отличается от валидации html. Документ с CSS кодом будет считаться валидным в том случае, если он прошел соответствующую проверку и не содержит ошибок.

Если используются таблицы стилей преимущественно во внешнем файле, это на много облегчает проверку кода, так как придется проверять всего лишь один файл, а не каждую страницу по отдельности. Если допущены ошибки при написании кода таблицы стилей, можно так же воспользоваться **валидатором**, для того, чтобы не искать ошибки вручную.

### Как пользоваться валидатором

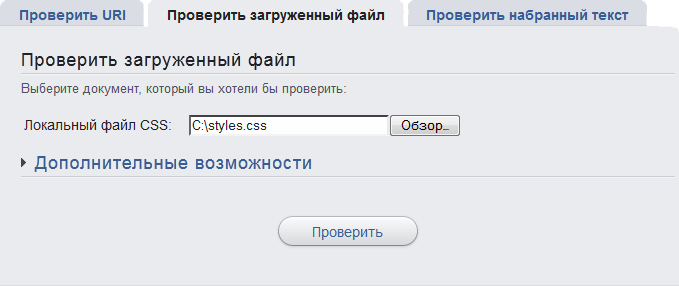
Самый простой способ проверки CSS кода предоставляет основной интерфейс валидатора. На его странице вы найдете три способа проверки:

* Проверка по URL

Достаточно ввести URL веб-страницы, которую надо проверить. После проверки кода валидатором можно увидеть одну из двух надписей - либо о том, что все прошло успешно и вас поздравят с валидным кодом, либо о том, что в вашем css коде были обнаружены ошибки, которые следуют исправить.

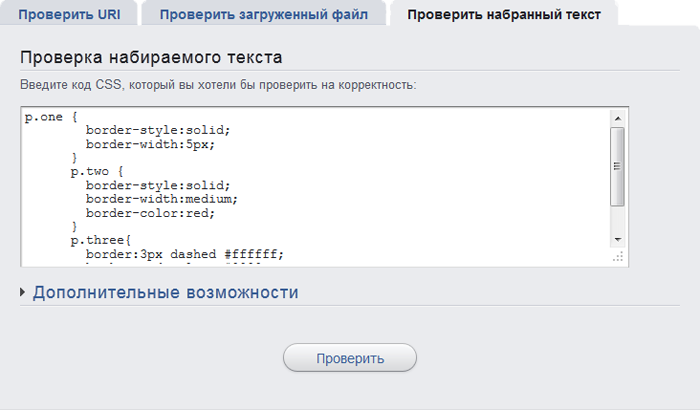
* Проверка загруженного файла

Этот способ позволяет загрузить проверяемый документ на сервер. Чтобы это сделать, надо нажать кнопку «Обзор» и выбрать файл, который нужно проверить. Сервер автоматически распознает тип загружаемого файла.



* Проверка набранного кода

Этот способ идеален для проверки части CSS-файла. Необходимо только скопировать код в текстовое поле.



## Теги <video> и <audio>

Хотя веб-браузеры имеют встроенную поддержку изображений с самых первых дней, встраивание аудио или видео на веб-страницу всегда требовало дополнительных подключаемых модулей (отдельных программных компонентов), которые добавляют больше возможностей браузеру, но не являются его частью (например Flash, QuickTime, RealPlayer и другие).

Плагин также означает, что приложение, которое воспроизводит аудио и видео содержимое, не является частью браузера. Подключаемый модуль заблокирован в виртуальной среде со своей программой, изолированной от браузера и от другого содержимого страницы. Кроме того, подключаемые модули являются необязательными, так что нельзя быть уверенным, что посетители сайта имеют нужное программное обеспечение для просмотра содержимого.

Стандарт HTML5 вводит новые элементы <audio> и <video>, которые позволяют веб-разработчикам встраивать звук и видео на веб-страницу, не требуя собственных плагинов. Конечно, из этого следует, что сам браузер должен иметь возможность воспроизведения таких файлов, на данный момент только последние версии браузеров поддерживают эти элементы.

### Медиа кодеки и форматы

Цифровые аудио и видео данные обрабатываются через кодек, формулу, которая преобразует и сжимает звук или видео в поток битов для передачи через Интернет (термин "кодек" представляет сокращенное сочетание двух слов "код" и "декодирование"). Когда данные доходят до конечного пользователя, то он должен иметь тот же кодек для декодирования закодированного сигнала и преобразования его обратно в звук или видео.

Некоторые медиа кодеки запатентованы, то есть они принадлежат только одной компании и не относятся к открытым стандартам, и, как правило, владельцы патентов обычно взимают лицензионные сборы за использование их алгоритмов. Разработчики браузеров, такие как Apple, Google и Microsoft, имеют большой капитал и готовы лицензировать запатентованные кодеки для своих браузеров. Другие производители браузеров, такие как Mozilla и Opera вместо этого выбирают открытые стандарты кодеков и не используют запатентованные их виды. Даже не смотря на то, что последние версии всех браузеров поддерживают встроенные медиа-файлы HTML5, они еще не пришли к соглашению какие кодеки лучше использовать.

После того как медиа данные закодированы, они должны быть инкапсулированы и упакованы для доставки в одном из нескольких форматов. Эти форматы являются контейнерами для медиа-файлов, которые передаются между сервером и клиентом. Для проигрывания встроенных медиа-файлов браузер должен сначала прочитать формат контейнера, а затем расшифровать закодированные данные внутри него. Так же, как браузеры поддерживают разные кодеки, они также поддерживают различные форматы контейнеров для встроенных медиа-файлов.

Встроенные медиа-файлы должны быть переданы с надлежащим типом контента для каждого формата, так и клиент, и сервер смогут распознать, как обрабатывать эти файлы. Веб-сервер обрабатывает медиа типы автоматически, по крайней мере, для наиболее распространенных форматов. Для некоторых новых форматов может понадобиться дополнительная настройка сервера, обычно это заключается в добавлении нового типа контента в файл конфигурации.

### Тег <audio>

**Элемент** <**audio**> внедряет звуковой файл на веб-страницу. Это заменяемый элемент, но он не относится к пустым элементам, поэтому он может содержать и собственный контент и другие элементы. В своей простейшей форме элемент <audio> должен иметь только атрибут **src**, указывающий путь к звуковому файлу:

|  |
| --- |
| <audio src="audio/file.mp3"></audio> |

Однако он редко используется в такой простой форме. По умолчанию элемент <audio> не имеет элементов управления и поэтому полностью невидимый. Он будет располагаться в HTML-документе и может быть доступен в браузере или JavaScript, но простое внедрение аудио файла не принесет особой пользы. С помощью логического атрибута **controls** можно добавить к аудио-файлу его собственные элементы управления, которые будут отображаться на веб-странице в месте расположения элемента <audio>:

|  |
| --- |
| <audio src="audio/file.mp3" controls></audio> |

Эти элементы управления предоставляются браузером и могут визуально отличаться в зависимости от браузера, но предоставляемые функции остаются неизменными:

* кнопка play/pause
* полоса прокрутки
* отображение времени
* громкость

Элемент <audio> может содержать один или несколько **элементов** <**source**>, каждый из которых будет ссылаться на свой аудио-файл, что будет полезно из-за отсутствия единого формата, поддерживаемого всеми браузерами. В этом случае браузер будет проигрывать первый медиа-файл, который он поддерживает, игнорируя все остальные:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>Название документа</title>  </head>  <body>  <audio controls="controls">  <source src="song.ogg" type="audio/ogg">  <source src="song.mp3" type="audio/mpeg">  </audio>  </body>  </html> |

**Аудио форматы и поддержка браузерами**

В настоящее время, существует 3 поддерживаемых аудио формата для элемента <audio>: MP3, Wav, Ogg

|  |  |  |  |
| --- | --- | --- | --- |
| **Браузер** | **MP3** | **Wav** | **Ogg** |
| Internet Explorer 9 или выше | да | нет | нет |
| Firefox 4.0 или выше | нет | нет | да |
| Google Chrome 6 или выше | да | да | да |
| Apple Safari 5 или выше | да | да | нет |
| Opera 10.6 или выше | нет | да | да |

### Тег <video>

**Элемент** <**video**> встраивает цифровое видео на веб-страницу. Как и в случае с <audio>, элемент <video> может иметь атрибут **src**, указывающий URL-адрес видеофайла:

|  |
| --- |
| <video src="video/file.mp4" controls></video> |

Атрибут **controls** добавляет элементы управления, с помощью которых пользователь сможет взаимодействовать с плеером. Атрибут **autoplay** указывает браузеру, что видео должно начать воспроизведение сразу, как только страница будет загружена, если он не указан, то в плеере будет отображен первый кадр исходного видео. Но чаще всего он не несет никакой полезной информации о том, что будет представлено с помощью видео-контента. Вместо этого можно предоставить любое изображение, указав к нему путь в атрибуте **poster**:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>Название документа</title>  </head>  <body>  <video width="320" height="240" controls="controls" poster="logo.png">  <source src="movie.mp4" type="video/mp4">  <source src="movie.ogg" type="video/ogg">  Ваш браузер не поддерживает тег video.  </video>  </body>  </html> |

Элемент <video> также может содержать один или несколько элементов <**source**>, которые будут указывать на файлы разных форматов:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8">  <title>Название документа</title>  </head>  <body>  <video width="320" height="240" controls="controls">  <source src="movie.mp4" type="video/mp4">  <source src="movie.ogg" type="video/ogg">  </video>  </body>  </html> |

**Видео форматы и поддержка браузерами**

В настоящее время, существует 3 поддерживаемых видео формата для элемента <video>: MP4, WebM и Ogg.

|  |  |  |  |
| --- | --- | --- | --- |
| **Браузер** | **MP4** | **WebM** | **Ogg** |
| Internet Explorer 9 или выше | да | нет | нет |
| Firefox 4.0 или выше | нет | да | да |
| Google Chrome 6 или выше | да | да | да |
| Apple Safari 5 или выше | да | нет | нет |
| Opera 10.6 или выше | нет | да | да |

## Валидация HTML-документа

Различные браузеры по-разному обрабатывают погрешности в HTML. Другими слова-ми, если в коде появляются ошибки, неясно, как веб-страница будет отображаться в различных браузерах. И только если в коде нет ошибок, большинство браузеров пока-жут страницу одинаково. Если начать дизайн с помощью CSS, но при этом HTML не отвечает определенным требованиям, то различия могут стать еще больше.

### Что такое валидация?

Валидация представляет собой проверку документа специальной программой - валидатором на соответствие установленным веб-стандартам и обнаружение имеющихся погрешностей. Данные стандарты названы спецификацией (разрабатывается Консорциумом Всемирной паутины, или сокращенно W3C).

Работа **валидатора** заключается в следующем: сначала определяется тип документа (указывается с помощью <!**DOCTYPE**>), а затем на предмет корректности и отсутствие ошибок проверяется HTML-код. При этом так же проверяется правильность использования названия тегов, их вложенности.

### Как пользоваться валидатором?

Интерфейс валидатора предоставляет три способа для проверки вашего HTML.

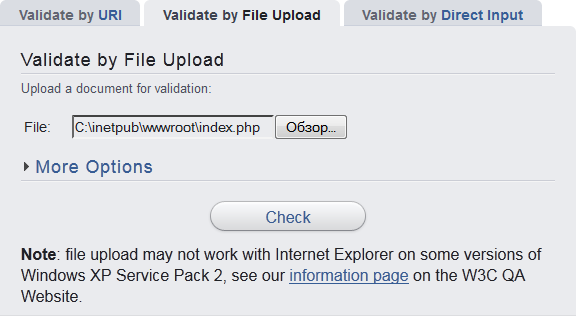
https://validator.w3.org/#validate\_by\_upload

**Проверка по URL адресу**

Если страница находится в сети, то можно просто напечатать или скопировать нужный URL-адрес в форму, нажать кнопку **Check**, и сервисная программа сама найдет ваш код и проверит его.

**Проверка загружаемого файла**

Можно нажать кнопку "**Обзор**" и выбрать файл на компьютере. После этого нажать кнопку **Check**, и браузер загрузит файл с вашего компьютера на удаленный сервер, а сервисная программа проверит его.



**Проверка набранного кода**

Можно скопировать код страницы и вставить в форму. Затем нажать кнопку **Check**, и сервисная программа проверит код HTML.