

Понятие боксовой модели, типы боксов.

Каждый элемент в CSS заключён в блок, и понимание поведения этих блоков — это ключ к умению задавать раскладку с помощью CSS, то есть выстраивать одни элементы относительно других элементов. В этом уроке мы надлежащим образом рассмотрим *блочную модель* CSS, чтобы вы могли создавать более сложные раскладки, понимая принцип ее работы и терминологию, которая к ней относится.

Блочные и строчные элементы

В CSS мы, говоря упрощённо, имеем два типа элементов — блочные и строчные. Эти характеристики относятся к поведению блоков в контексте потока страницы и относительно других блоков на странице.

Если элемент определён как блочный, то он будет вести себя следующим образом:

- Начнётся с новой строки.
- Будет расширяться вдоль строки таким образом, чтобы заполнить всё пространство, доступное в её контейнере. В большинстве случаев это означает, что блок станет такой же ширины, как и его контейнер, заполняя 100% доступного пространства.
- Будут применяться свойства [width](#) и [height](#).
- Внешние и внутренние отступы, рамка будут отодвигать от него другие элементы.

Если не изменить намеренно тип отображения на строчный, то такие элементы, как заголовки (например, `<h1>`) и `<p>`, все используют `block` как свой внешний тип отображения по умолчанию.

Если элемент имеет тип отображения `inline` (строчный), то:

- Он не будет начинаться с новой строки.
- Свойства [width](#) и [height](#) не будут применяться.

- Вертикальные внешние и внутренние отступы, рамки будут применяться, но не будут отодвигать другие строчные элементы.
- Горизонтальные внешние и внутренние отступы, рамки будут применяться и будут отодвигать другие строчные элементы.

Элемент `<a>`, используемый для ссылок, ``, `` и `` — всё это примеры по умолчанию строчных элементов.

Тип отображения, применяемый к элементу, определяется значениями свойства [display](#), такими как `block` и `inline`, и относится к внешнему значению `display`.

Экскурс: внутренний и внешний типы отображения

Здесь следует объяснить, что такое внутренние и внешние типы отображения. Как уже говорилось выше, каждый блок в CSS имеет *внешний* тип отображения, который определяет, блочный он или строчный.

Элементы также имеют *внутренний* тип отображения, который определяет расположение элементов внутри них. По умолчанию элементы внутри блока располагаются в [нормальном потоке](#): они ведут себя так же, как и любые другие блочные или строчные элементы (как описано выше).

Однако мы можем изменить внутренний тип отображения, используя такие значения `display` как `flex`. Если мы установим `display: flex;` для элемента, внешний тип отображения примет значение `block`, но внутренний тип изменится на `flex`. Любые прямые дочерние элементы этого блока станут *flex-объектами* и будут размещены в соответствии с правилами, изложенными в спецификации [Flexbox](#).

Когда вы перейдёте к более подробному изучению CSS вёрстки, вы встретите `flex` и другие внутренние значения, которые могут быть у ваших элементов, например [grid](#).

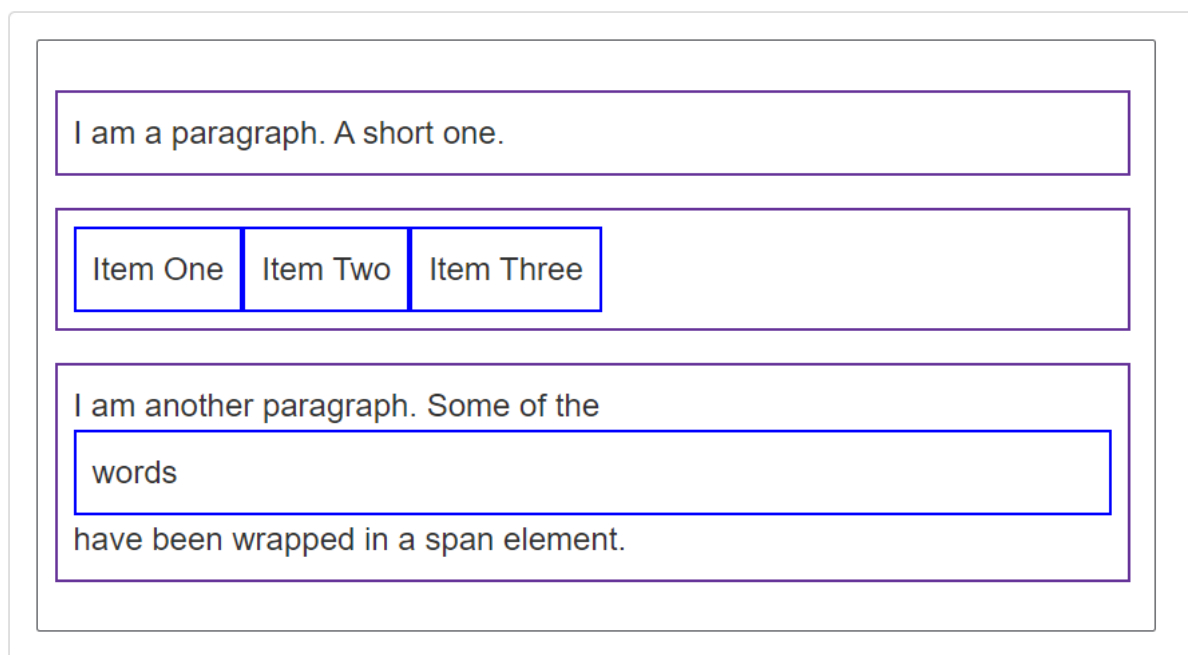
Тем не менее, блочное и строчное расположение — это поведение web-элементов по умолчанию. Как было сказано выше, это иногда называют *нормальным потоком* (*normal flow*), потому что при отсутствии какой-либо другой инструкции элементы имеют блочное или строчное расположение.

Примеры разных типов отображения

Давайте продолжим и рассмотрим некоторые примеры. Ниже мы имеем три разных элемента HTML с внешним типом отображения `block`. Первый — это абзац, который имеет обрамление, указанное в CSS. Браузер отображает его как блочный элемент, поэтому абзац начинается с новой строки и расширяется на всю доступную ему ширину.

Второй — это список, который сверстан с использованием `display: flex`. Это устанавливает flex-расположение для элементов внутри контейнера, однако сам список — блочный элемент и — как и абзац — расширяется на всю ширину контейнера и начинается с новой строки.

Ниже у нас есть абзац блочного типа, внутри которого есть два элемента ``. Эти элементы по умолчанию имеют тип `inline`, однако у одного из них задан класс `block`, для которого мы установили `display: block`.



```
p,  
ul {  
  border: 2px solid rebeccapurple;  
  padding: .5em;  
}
```

```
.block,
```

```
li {  
  border: 2px solid blue;  
  padding: .5em;  
}
```

```
ul {  
  display: flex;  
  list-style: none;  
}
```

```
.block {  
  display: block;  
}
```

```
<p>I am a paragraph. A short one.</p>
```

```
<ul>
```

```
  <li>Item One</li>
```

```
  <li>Item Two</li>
```

```
  <li>Item Three</li>
```

```
</ul>
```

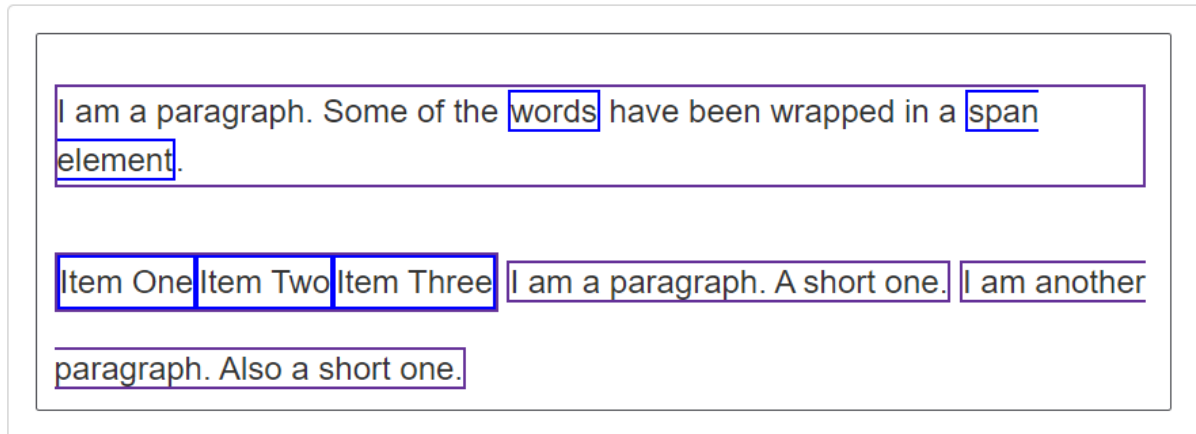
```
<p>I am another paragraph. Some of the <span class="block">words</span> have  
been wrapped in a <span>span element</span>.</p>
```

Мы можем видеть, как строчные элементы (`inline`) ведут себя в следующем примере. Элементы `` в первом абзаце строчные по умолчанию и поэтому не приводят к переносу строки.

У нас также есть элемент ``, для которого установлено `display: inline-flex`, что создаёт строчный элемент вокруг нескольких `flex`-объектов.

Наконец, у нас есть два абзаца, для которых установлено `display: inline`. И строчный `flex`-контейнер, и абзацы располагаются вместе на одной строке, а не начинаются каждый с новой строки, как они отображались бы, будучи блочными элементами.

В примере вы можете заменить `display: inline` на `display: block` или `display: inline-flex` на `display: flex` для переключения между этими двумя режимами отображения.



```
p,  
ul {  
  border: 2px solid rebeccapurple;  
}
```

```
span,  
li {  
  border: 2px solid blue;  
}
```

```
ul {  
  display: inline-flex;  
  list-style: none;  
  padding: 0;  
}
```

```
.inline {  
  display: inline;  
}
```

```
<p>  
  I am a paragraph. Some of the  
  <span>words</span> have been wrapped in a  
  <span>span element</span>.  
</p>
```

```
<ul>
  <li>Item One</li>
  <li>Item Two</li>
  <li>Item Three</li>
</ul>
<p class="inline">I am a paragraph. A short one.</p>
<p class="inline">I am another paragraph. Also a short one.</p>
```

Позже вы встретите такое понятие как flex-раскладка; главное, что нужно запомнить сейчас, это то, что изменение значения свойства `display` может изменить внешний тип отображения элемента на блочный или строчный, что меняет способ его отображения относительно других элементов в раскладке страницы.

В оставшейся части мы сосредоточимся на внешнем типе отображения.

Что такое блочная модель CSS?

Полностью блочная модель в CSS применяется к блочным элементам, строчные элементы используют не все свойства, определённые блочной моделью. Модель определяет, как разные части элемента — поля, рамки, отступы и содержимое — работают вместе, чтобы создать объект, который вы можете увидеть на странице. Дополнительная сложность заключается в том, что существуют стандартная и альтернативная блочные модели.

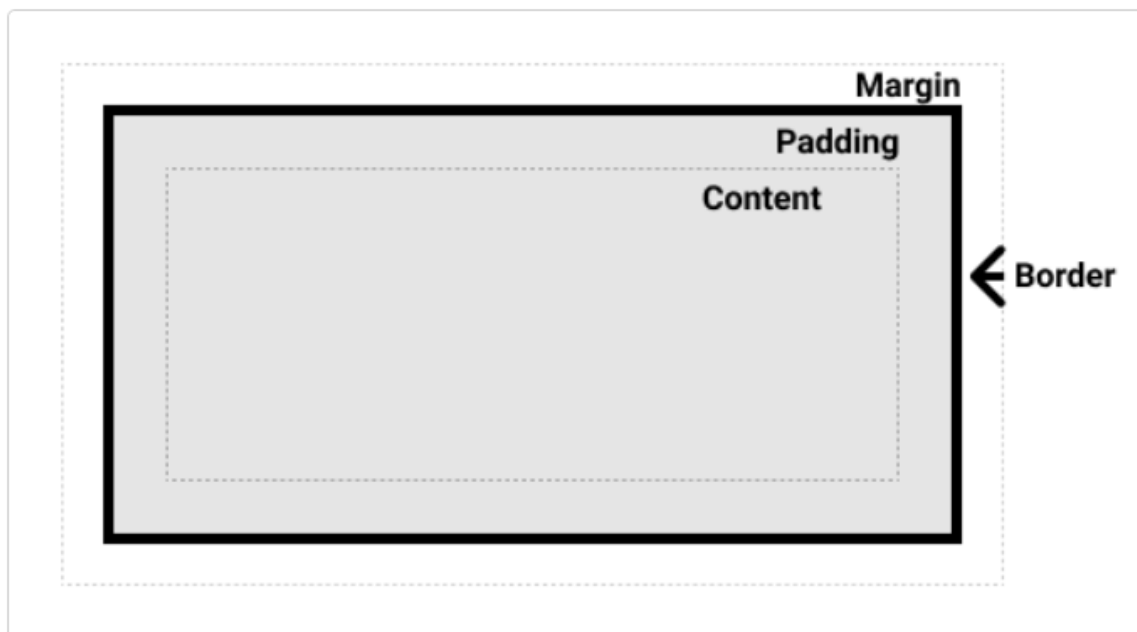
Составляющие элемента

Составляя блочный элемент в CSS мы имеем:

- Содержимое: область, где отображается ваш контент, размер которой можно изменить с помощью таких свойств, как [width](#) и [height](#).
- Внутренний отступ: отступы располагаются вокруг содержимого в виде пустого пространства; их размер контролируется с помощью [padding](#) и связанных свойств.
- Рамка: рамка оборачивает содержимое и внутренние отступы. Её размер и стиль можно контролировать с помощью [border](#) и связанных свойств.
- Внешний отступ: внешний слой, заключающий в себе содержимое, внутренний отступ и рамки, представляет собой пространство между

текущим и другими элементами. Его размер контролируется с помощью [margin](#) и связанных свойств.

Рисунок ниже показывает эти слои:



В стандартной блочной модели, если указать элементу атрибуты `width` и `height`, это определит ширину и высоту *содержимого*. Любые отступы и рамки затем добавляются к этой ширине и высоте для получения общего размера элемента. Это показано на изображении ниже.

Предположим, что в элементе есть следующий CSS определяющий `width`, `height`, `margin`, `border`, и `padding`:

```
.box {  
  
width: 350px;  
  
height: 150px;
```

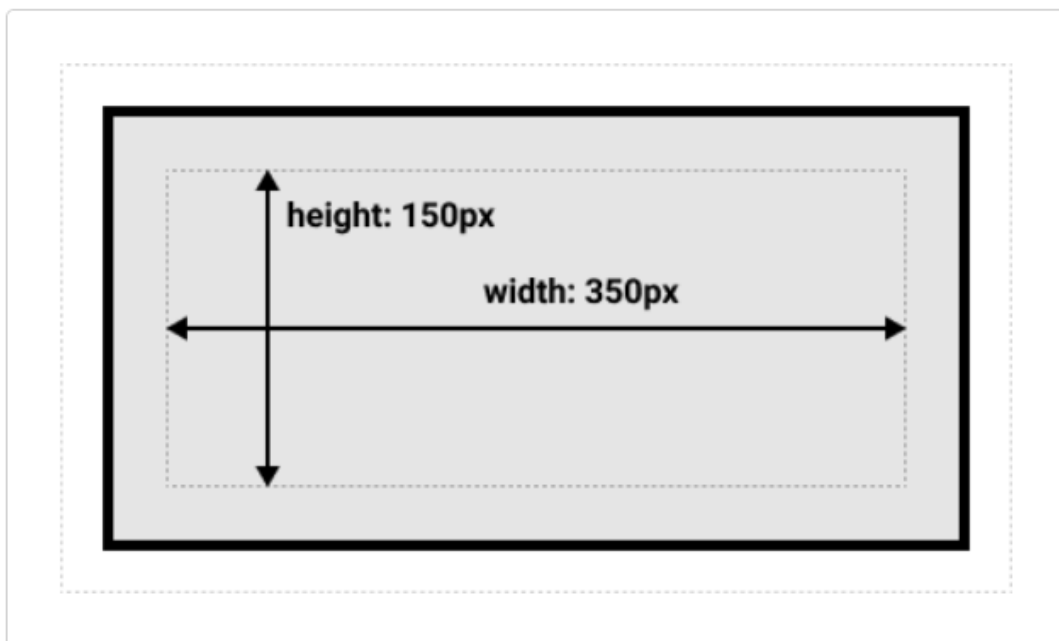
```
margin: 10px;
```

```
padding: 25px;
```

```
border: 5px solid black;
```

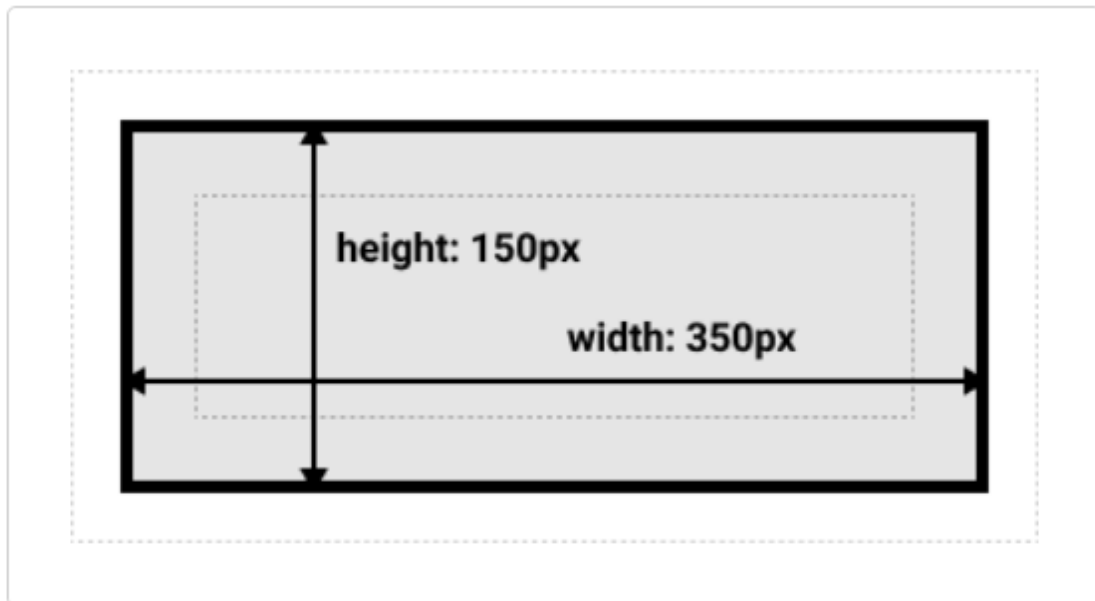
```
}
```

Пространство, занимаемое нашим объектом с использованием стандартной блочной модели, на самом деле будет равно 410px в ширину ($350 + 25 + 25 + 5 + 5$) и 210px в высоту ($150 + 25 + 25 + 5 + 5$), поскольку отступы и рамки добавляются к размерам поля содержимого.



Вы можете подумать, что довольно неудобно добавлять рамки и отступы, чтобы получить реальный размер элемента, и окажетесь правы! По этой причине, спустя некоторое время после стандартной блочной модели, в CSS была введена альтернативная блочная модель. При использовании альтернативной модели любая ширина — это ширина видимой части элемента на странице, поэтому ширина области содержимого будет равна общей ширине минус ширина рамки и внутреннего отступа. Тот же CSS, который

использовался выше, даст следующий результат (ширина = 350px, высота = 150px).



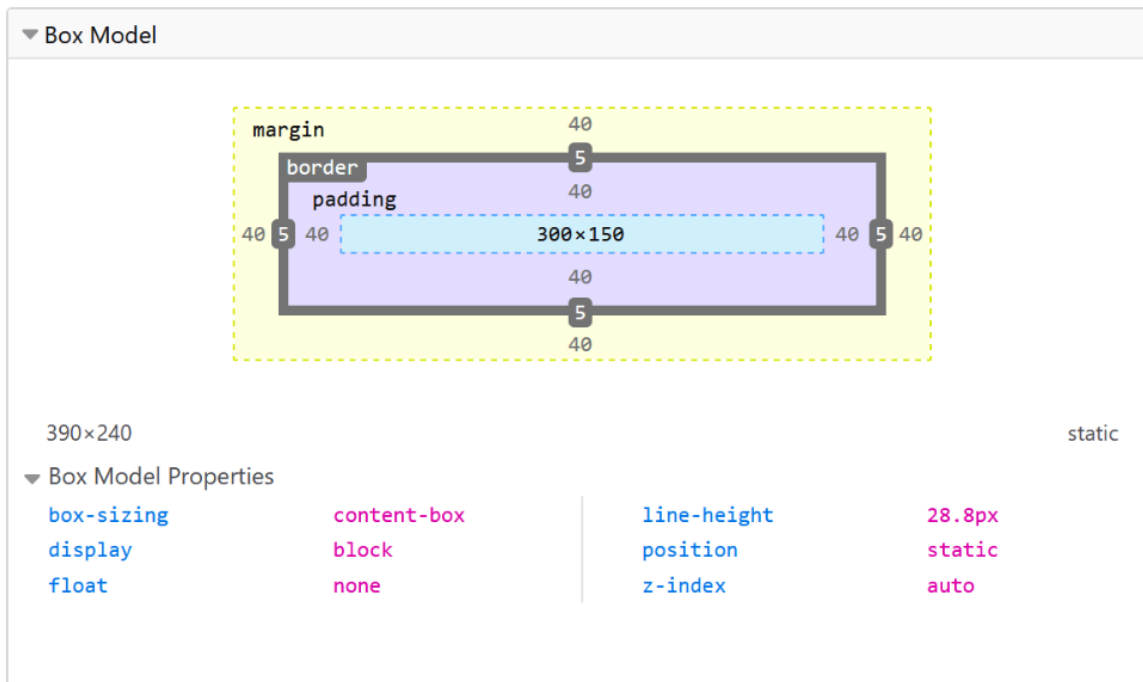
По умолчанию браузеры используют стандартную блочную модель. Если вы хотите использовать альтернативную блочную модель для элемента, установите для него свойство `box-sizing: border-box`. С помощью этого вы говорите браузеру о том, что рамка элемента определяется любыми размерами, которые вы устанавливаете.

```
.box {  
  
    box-sizing: border-box;  
  
}
```

Если вы хотите, чтобы все ваши элементы использовали альтернативную блочную модель, что является распространённым выбором среди разработчиков, установите свойство `box-sizing` для элемента `<html>`, затем задайте всем элементам наследование этого значения (`inherit`), как показано в примере ниже. Если вы хотите понять ход мыслей, стоящий за этим решением, читайте статью [the CSS Tricks article on box-sizing](#).

```
html {  
  
    box-sizing: border-box;  
  
}  
  
*,  
  
*::before,  
  
*::after {  
  
    box-sizing: inherit;  
  
}
```

[Инструменты разработчика](#) вашего браузера могут значительно облегчить понимание блочной модели. Если вы проверите элемент в инструментах разработчика Firefox, вы можете увидеть его размер, а также внешний и внутренний отступы и рамку. Проверка элемента таким способом — отличный способ выяснить, действительно ли размер вашего блока такой, какой вы думаете!



Внешние, внутренние отступы и рамки

Вы уже видели свойства [margin](#), [padding](#) и [border](#) в работе в приведённом выше примере. Используемые в этом примере свойства — сокращённые и позволяют нам устанавливать все четыре стороны блока одновременно. У них также есть эквивалентные полные свойства, которые позволяют индивидуально управлять разными сторонами блока.

Давайте рассмотрим эти свойства более подробно.

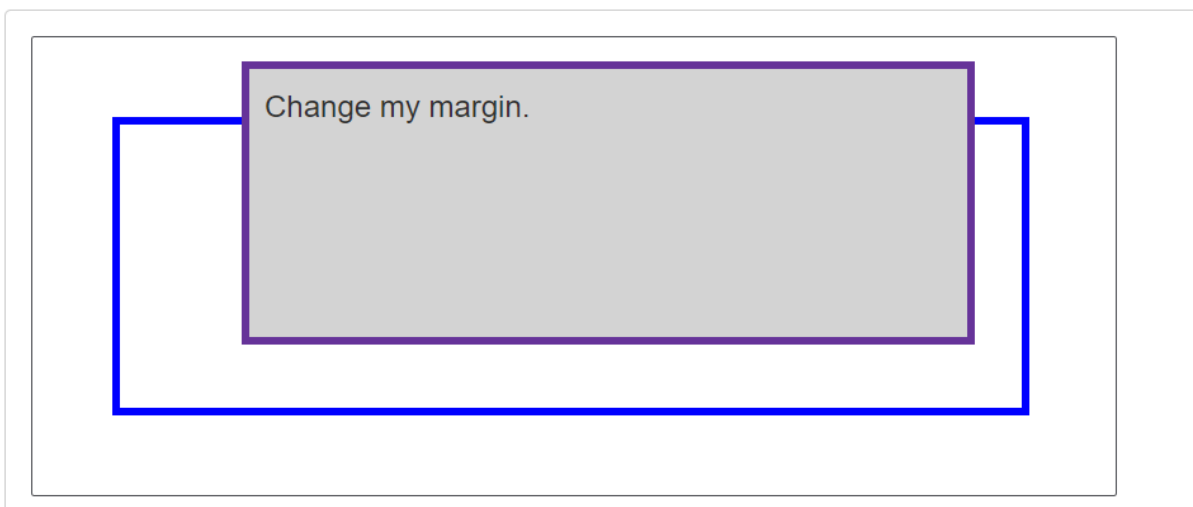
Внешний отступ (margin)

Внешний отступ — это невидимое пространство вокруг вашего элемента. Оно отталкивает другие элементы от него. Внешний отступ может быть как положительным, так и отрицательным. Негативное значение может привести к перекрытию некоторых элементов страницы. Независимо от того, используете ли вы стандартную или альтернативную блочную модель, внешний отступ всегда добавляется после расчёта размера видимого блока.

Мы можем контролировать все поля элемента сразу, используя свойство [margin](#), или каждую сторону индивидуально, используя эквивалентные полные свойства:

- [margin-top](#)
- [margin-right](#)
- [margin-bottom](#)
- [margin-left](#)

В примере ниже, попробуйте изменить значение `margin` чтобы увидеть как блок смещается, создавая или удаляя пространство (если вводить отрицательные значения `margin`) между этим элементом и элементом его содержащим.



```
.box {
```

```
margin-top: -40px;
```

```
margin-right: 30px;
```

```
margin-bottom: 40px;
```

```
margin-left: 4em;
```

```
}
```

```
<div class="container">
```

```
<div class="box">Change my margin.</div>
```

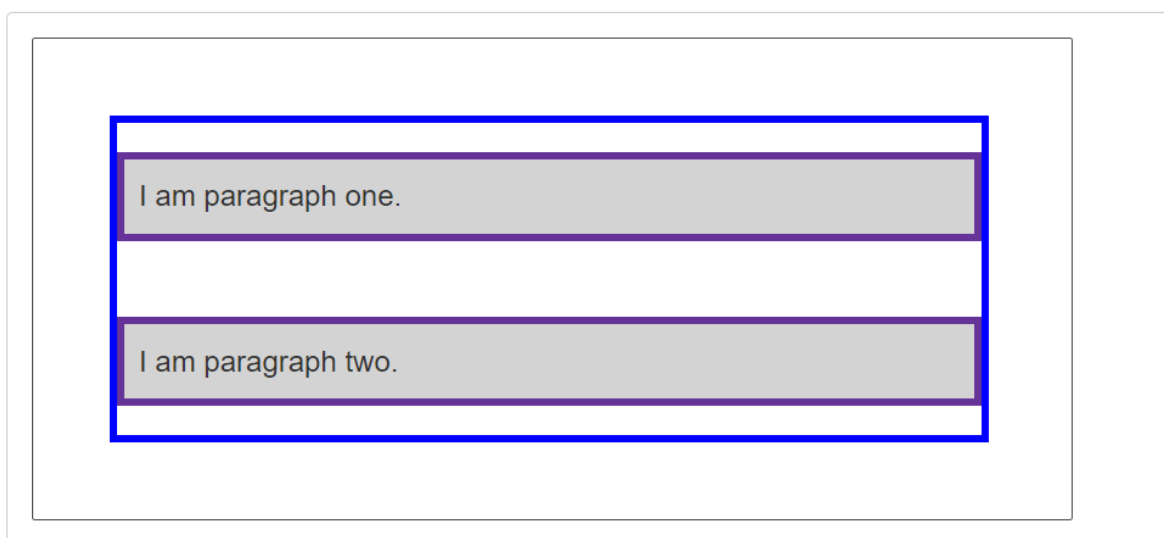
```
</div>
```

Схлопывание внешних отступов

Ключевой момент, который нужно понимать в отношении внешних отступов (margin), это концепция схлопывания. Если у вас есть два элемента, внешние отступы которых соприкасаются, и оба значения margin положительные, то эти значения будут объединены в одно, равное большему из двух значений. А если одно или оба значения отрицательны, то сумма отрицательных значений будет вычтена из общей суммы.

В примере ниже есть два абзаца. Первому абзацу задан margin-bottom 50 пикселей. У второго абзаца margin-top 30 пикселей. Отступы схлопываются так, что в результате margin между двумя блоками составляет 50 пикселей, а не сумму отдельных значений margin.

Вы можете проверить это, установив второму абзацу margin-top равный 0. Видимое расстояние между двумя абзацами не изменится — отступ остаётся равен 50 пикселям, заданным в margin-bottom первого абзаца. Если вы установите значение -10px, то увидите, что margin становится 40px — происходит вычитание из положительного значения 50px у первого абзаца.



```
.one {
```

```
margin-bottom: 50px;  
  
}
```

```
.two {  
  
margin-top: 30px;  
  
}
```

```
<div class="container">  
  
  <p class="one">I am paragraph one.</p>  
  
  <p class="two">I am paragraph two.</p>  
  
</div>
```

Существует ряд правил, которые определяют, когда внешние отступы схлопываются, а когда нет. Для получения подробной информации см. [margin collapsing](#). Главное, что нужно сейчас помнить, — это то, что схлопывание отступов существует. Если вы создаёте пространство с внешними отступами и не получаете ожидаемого результата, вероятно, именно это и происходит.

Рамка

Рамка располагается между `margin` и `padding` блочного элемента. Если вы используете стандартную блочную модель, размер рамки прибавляется к значениям `width` и `height` элемента. Если вы используете альтернативную блочную модель, то размер рамки уменьшает поле контента своего блока, так как значения рамки входят в заданные ему `width` и `height`.

Для стилизации рамок существует большое количество различных свойств: четыре рамки, и каждая из них имеет свой стиль, ширину и цвет, которыми мы можем манипулировать.

Вы можете установить ширину, стиль или цвет всех четырёх рамок сразу, используя свойство [border](#).

Чтобы установить индивидуальные свойства для каждой из четырёх сторон, вы можете использовать:

- [border-top \(en-US\)](#)
- [border-right \(en-US\)](#)
- [border-bottom](#)
- [border-left \(en-US\)](#)

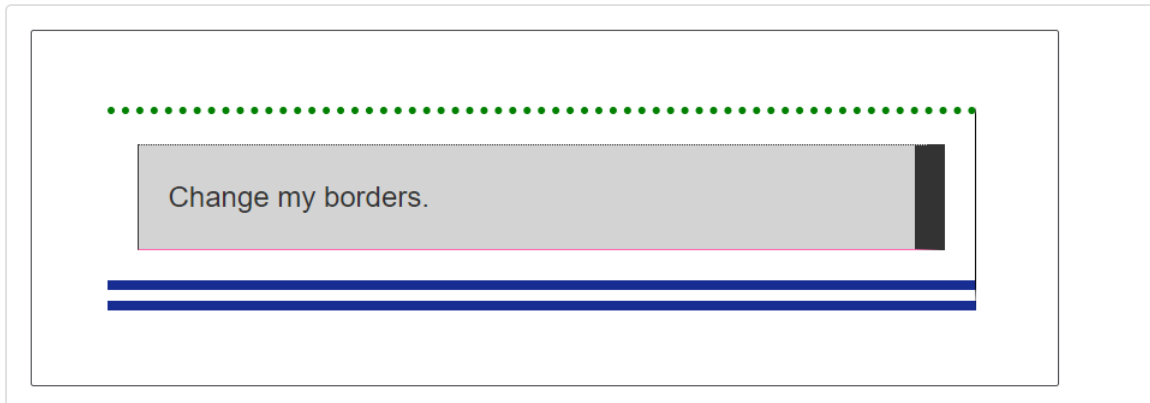
Для установки ширины, стиля или цвета всех рамок используйте:

- [border-width](#)
- [border-style \(en-US\)](#)
- [border-color \(en-US\)](#)

Чтобы установить ширину, стиль или цвет для каждой рамки индивидуально, вы можете использовать следующие свойства:

- [border-top-width \(en-US\)](#)
- [border-top-style \(en-US\)](#)
- [border-top-color \(en-US\)](#)
- [border-right-width \(en-US\)](#)
- [border-right-style \(en-US\)](#)
- [border-right-color \(en-US\)](#)
- [border-bottom-width \(en-US\)](#)
- [border-bottom-style \(en-US\)](#)
- [border-bottom-color \(en-US\)](#)
- [border-left-width \(en-US\)](#)
- [border-left-style \(en-US\)](#)
- [border-left-color \(en-US\)](#)

В примере ниже мы использовали различные сокращённые и полные способы создания рамок. Поиграйте с различными свойствами, чтобы проверить, как вы поняли принципы их работы. Информацию о различных стилях, которые вы можете использовать, можно найти на страницах MDN о свойствах рамок.



```
.container {  
  
  border-top: 5px dotted green;  
  
  border-right: 1px solid black;  
  
  border-bottom: 20px double rgb(23,45,145);  
  
}
```

```
.box {  
  
  border: 1px solid #333333;  
  
  border-top-style: dotted;  
  
  border-right-width: 20px;
```



```
border-bottom-color: hotpink;

}

<div class="container">

  <div class="box">Change my borders.</div>

</div>
```

Внутренний отступ (padding)

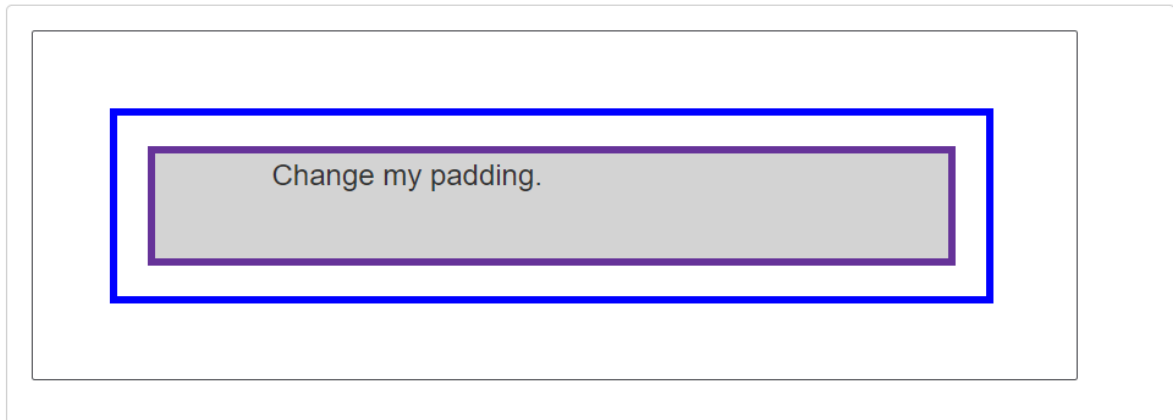
Внутренний отступ расположен между рамкой и областью контента блока. В отличие от внешних отступов (margin), вы не можете использовать отрицательные значения для padding: они должны быть положительными или равными 0. Любой применённый к вашим элементам фон будет отображаться под областью padding, поэтому внутренний отступ обычно используется, чтобы отодвинуть контент от рамок.

Вы можете контролировать значение padding для всех сторон элемента, используя свойство [padding](#), или для каждой стороны индивидуально, используя следующие полные свойства:

- [padding-top \(en-US\)](#)
- [padding-right](#)
- [padding-bottom \(en-US\)](#)
- [padding-left](#)

Если вы измените значения padding для класса `.box` в примере ниже, то увидите, что это изменяет положение текста внутри элемента.

Вы также можете изменить padding для класса `.container`, который задаёт отступ между контейнером и блоком. Внутренний отступ может быть изменён для любого элемента и создаст пространство между его рамкой и содержимым.



```
.box {
```

```
padding-top: 0;
```

```
padding-right: 30px;
```

```
padding-bottom: 40px;
```

```
padding-left: 4em;
```

```
}
```

```
.container {
```

```
padding: 20px;
```

```
}
```

```
<div class="container">
```

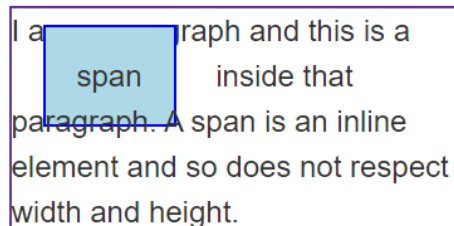
```
<div class="box">Change my padding.</div>
```

```
</div>
```

Блочная модель и строчные элементы

Всё, сказанное ранее, полностью применимо к блочным элементам. Некоторые из свойств могут быть также применены и к строчным (inline) элементам, например к ``.

В приведённом ниже примере у нас есть `` внутри абзаца, и мы применили к нему `width`, `height`, `margin`, `border` и `padding`. Вы можете видеть, что ширина и высота игнорируются. Вертикальные внешние и внутренние отступы и рамки применены, но они не изменяют положение других элементов относительно нашего строчного элемента, и поэтому отступы и рамка перекрывают другие слова в абзаце. Горизонтальные внешние и внутренние отступы и рамки применены и заставляют другие элементы отодвинуться от нашего.



I a graph and this is a
span inside that
paragraph. A span is an inline
element and so does not respect
width and height.

```
span {
```

```
margin: 20px;
```

```
padding: 20px;
```

```
width: 80px;
```

```
height: 50px;
```

```
background-color: lightblue;
```

```
border: 2px solid blue;  
  
}
```

```
<p>
```

I am a paragraph and this is a `span` inside that paragraph. A span is an inline element and so does not respect width and height.

```
</p>
```

Использование `display: inline-block`

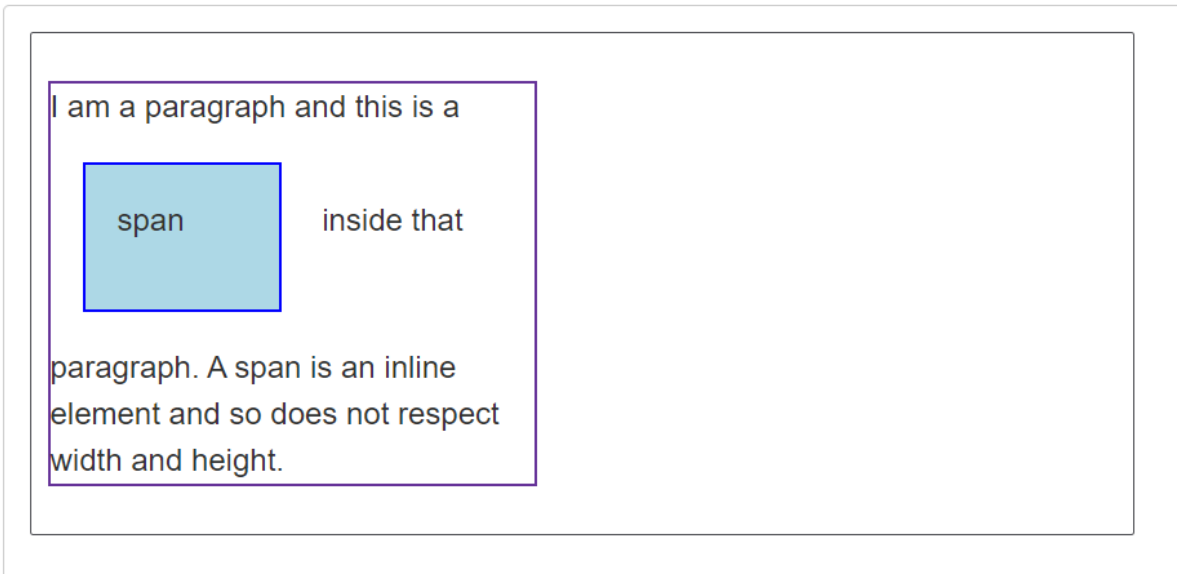
Существует особое значение `display`, которое представляет собой золотую середину между `inline` и `block`. Это полезно в ситуациях, когда вы не хотите, чтобы элемент переносился на новую строку, но нужно, чтобы он применял `width` и `height` и избегал перекрытия, показанного выше.

Элемент с `display: inline-block` применяет ряд свойств блочного элемента, о которых мы уже знаем:

- Применяются свойства `width` и `height`.
- Использование `padding`, `margin` и `border` приведёт к тому, что другие элементы будут отодвинуты от нашего элемента.

Он не перенесётся на новую строку и станет больше, чем его содержимое, только если вы явно зададите свойства `width` и `height`.

В следующем примере мы добавили `display: inline-block` к нашему элементу ``. Попробуйте изменить значение свойства на `display: block` или полностью удалить строку, чтобы увидеть разницу.



```
span {
```

```
margin: 20px;
```

```
padding: 20px;
```

```
width: 80px;
```

```
height: 50px;
```

```
background-color: lightblue;
```

```
border: 2px solid blue;
```

```
display: inline-block;
```

```
}
```

```
<p>
```

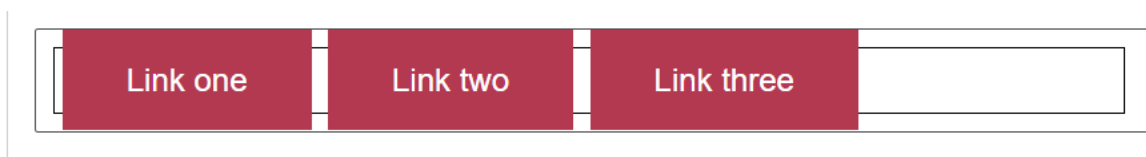
I am a paragraph and this is a `span` inside that paragraph. A span is an inline element and so does not respect width and height.

</p>

Это может быть полезно, когда вы хотите создать ссылку с большой областью попадания, добавив `padding`. `<a>` — это строчный элемент, такой же как ``; вы можете использовать `display: inline-block`, чтобы разрешить применение отступов, что упростит пользователю переход по ссылке.

Довольно часто это можно увидеть в панелях навигации. Приведённая ниже навигация отображается в виде строки с использованием `flexbox`, и мы добавили отступы к элементу `<a>`, потому что хотим, чтобы `background-color` изменялся при наведении курсора на `<a>`. Отступы перекрывают рамку элемента ``. Это происходит потому, что `<a>` — строчный элемент.

Добавьте в правило `display: inline-block` с помощью селектора `.links-list a`, и вы увидите, как он решает эту проблему, заставляя другие элементы соблюдать отступы.



```
.links-list a {  
  
    background-color: rgb(179,57,81);  
  
    color: #fff;  
  
    text-decoration: none;  
  
    padding: 1em 2em;  
  
}
```

```
.links-list a:hover {  
  
    background-color: rgb(66, 28, 40);  
  
    color: #fff;  
  
}  
  
<nav>  
  
    <ul class="links-list">  
  
        <li><a href="">Link one</a></li>  
  
        <li><a href="">Link two</a></li>  
  
        <li><a href="">Link three</a></li>  
  
    </ul>  
  
</nav>
```

Анализ сетки на макете, определение крупной сетки страницы

Если дизайнер проектирует макет «на глазок», то устраивает на странице хаос из элементов. Но есть другой путь: воспользоваться модульной сеткой.

Что такое модульные сетки

Модульная сетка — это система организации объектов в макете, основанная на колонках, рядах и отступах между ними. По направляющим сетки дизайнер выравнивает все элементы сайта: текстовые блоки, картинки, кнопки, фактоиды.

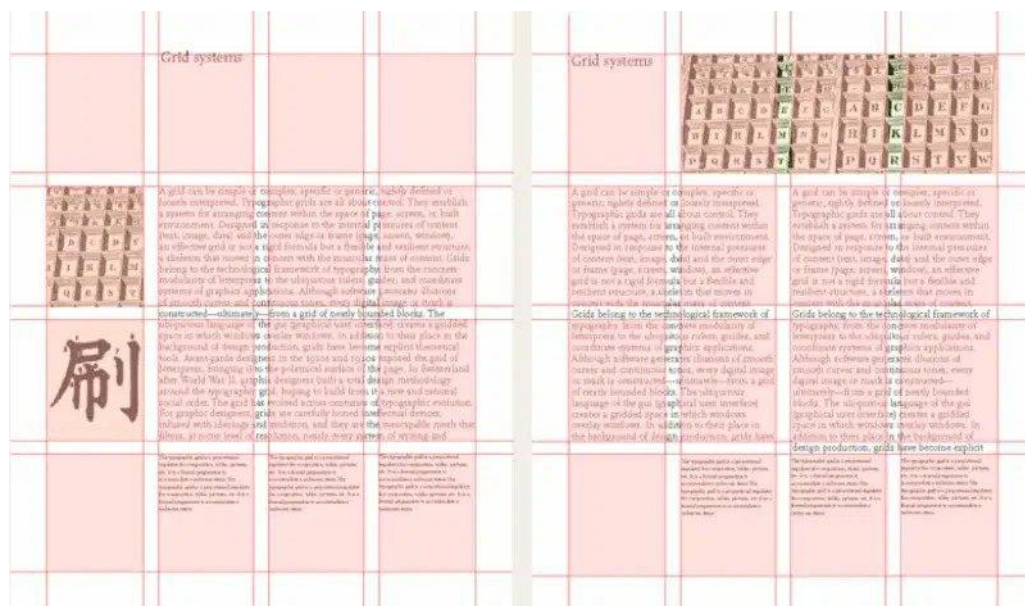
Если **лендинг создан** без использования модульной сетки, то при чтении глаз спотыкается о края невыровненных элементов и кажется, что на странице бардак.

Модульная сетка в дизайне сайта упорядочивает расположение элементов на странице, но у неё есть ещё несколько функций:

- **Определять единый стиль.** С помощью сетки дизайнер устанавливает правила выравнивания и добавления новых элементов в макет.
- **Ускорять вёрстку макета.** С сеткой веб-дизайнер тратит меньше усилий на размещение новых элементов на лендинге — достаточно выбрать направляющую и привязать элемент к ней.
- **Делать макет эстетичным.** Если элементы лендинга пропорциональны и структурированы, пользователям приятнее их воспринимать.
- **Помогать пользователю легче считывать информацию.** Сетка создаёт визуальный порядок, в котором проще ориентироваться. Например, если все заголовки выровнены по направляющей в левой части экрана, то при поиске пользователь будет «сканировать» взглядом только эту область, а не всю страницу.
- **Упрощать разработку макета в вёрстке.** **Фронтенд-разработчикам** проще верстать сайты, созданные дизайнерами по модульной сетке.

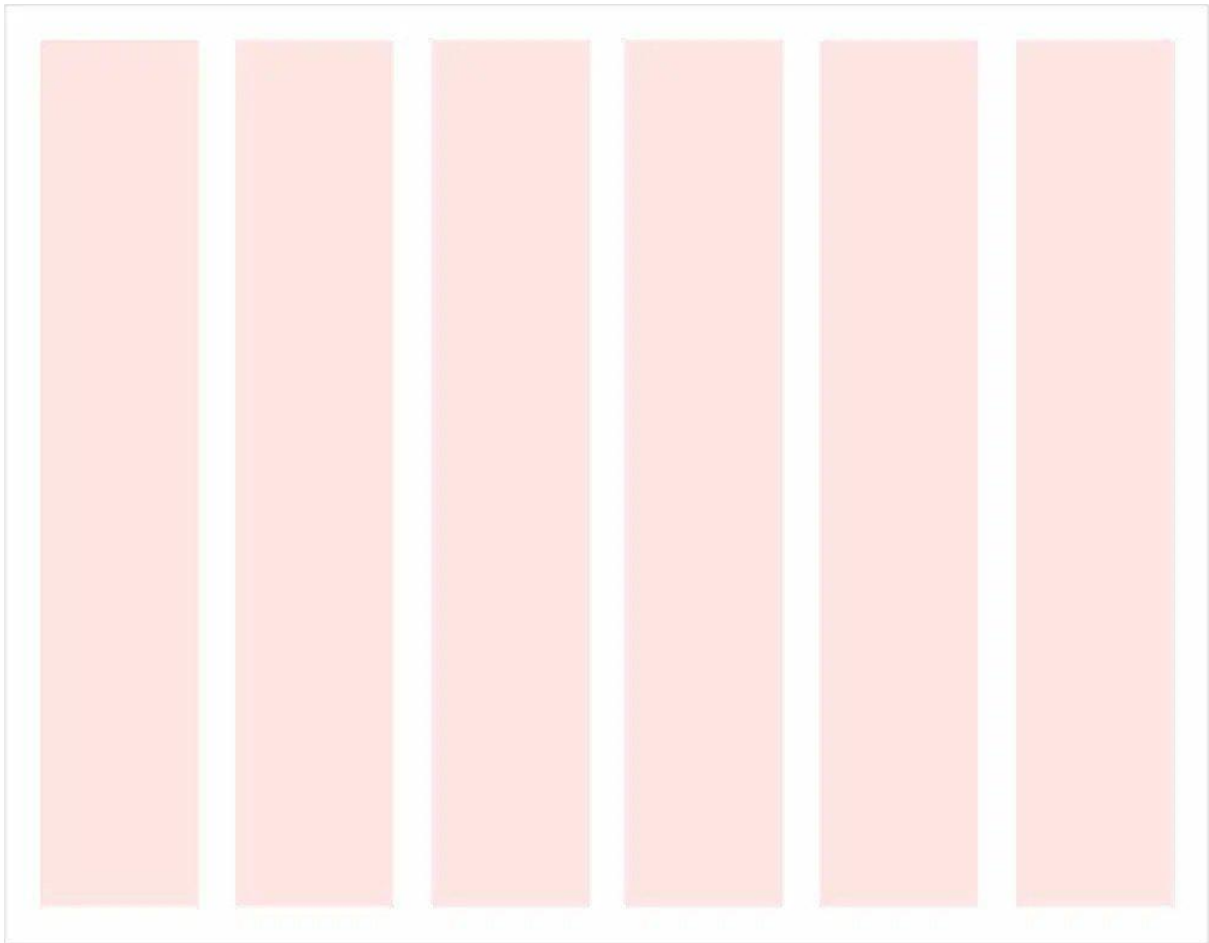
Виды модульных сеток

Есть два вида сеток: модульная и колоночная. Модульная используется в графическом дизайне и полиграфии, то есть там, где пространство ограничено форматом листа. Модульная сетка состоит из вертикальных и горизонтальных направляющих.



Второй вид сетки — колоночная. Это та же модульная сетка, но адаптированная для страниц в интернете: горизонтальных направляющих нет, потому что нет ограничений по формату.

Ширина веб-страницы конечна, а высота зависит от количества контента и может быть любой. Страница растягивается на много тысяч пикселей, если этого требует замысел дизайнера.



Спецификация Flexible Box Layout: оси флексов, их направление и расположение вдоль них флекс-элементов; растяжение, сужение и базовый размер флексов.

Flexbox (или просто flex) — это способ позиционирования элементов в [CSS](#). С помощью этой функции можно быстро и легко описывать, как будет располагаться тот или иной блок на веб-странице. Элементы выстраиваются по заданной оси и автоматически распределяются согласно настройкам.

В верстке есть такое понятие, как сетка, или лэйаут (layout). Это то, как блоки — крупные и мелкие — расположены на странице. Списки, наборы ссылок, виджеты и статьи — все это распределено по веб-странице не хаотично, а по определенной структуре. Эта структура и есть сетка. Flex — один из способов ее задать. Flexbox появился в CSS 3.0. До него для

построения сеток использовали таблицы и `float`'ы — команды для смещения элементов в сторону. Это было неудобно, трудоемко и сильно ограничивало возможности верстки. Благодаря Flexbox верстать стало удобнее и легче, появилась возможность реализовать более интересные дизайнерские идеи.

Flexbox расшифровывается как **Flexible Box** — гибкий ящик или блок. Гибкость — основная идея этого способа. Читается как «флексбокс» или просто «флекс».

Кто пользуется Flexbox

- **Верстальщики** — позиционирование элементов с помощью языка стилей CSS относится к верстке, и в некоторых компаниях этим занимаются отдельные сотрудники.
- **Фронтенд-разработчики** — другой подход в коммерческой разработке подразумевает, что версткой занимаются фронтендеры и они же «оживляют» интерфейсы с помощью JavaScript.
- **Фуллстак-разработчики** — им тоже следует понимать, как работает верстка, хотя сталкиваются они с ней реже, чем предыдущие два специалиста.
- **Другие сотрудники** — эпизодические задачи по верстке могут появляться у специалистов техподдержки, контент-менеджеров и прочих специалистов. Но они сталкиваются с такой необходимостью редко.

Flexbox — один из двух современных методов построения сеток страницы. Сетка — своеобразный каркас, который описывает, как расположены элементы. Благодаря ей веб-страницы выглядят понятно и аккуратно, а

блоки не «плывут» вразнобой. Флексы используют для построения больших и малых сеток.

- Большие, основные сетки — это те, которые описывают положение крупных блоков. Например, это заголовки, текстовые блоки, боковые блоки со ссылками и виджетами и так далее.
- Малые, микросетки — это сетки внутри крупных блоков. Они описывают, как в них располагаются более маленькие элементы: ссылки, тексты, картинки и так далее.

Еще флексбокс применяется для других задач: выравнивания элементов, расположения их в списке, мелких дизайнерских исправлений и так далее.

Как работает Flexbox

Принцип работы флексбокса — выстраивание элементов внутри какого-то крупного блока по прямой оси. Распределение и поведение этих элементов описывает верстальщик.

Создание контейнера. Крупный блок, внутри которого расположены элементы, называется флекс-контейнером. Чтобы превратить блок во флекс-контейнер, нужно задать для него CSS-свойство:

```
display: flex;
```

Эту команду можно прочесть как «способ отображения: флексбокс». После этого все, что находится внутри контейнера, будет подчиняться правилам флекса — превратится во флекс-элементы. Но это касается только прямых потомков — блоков непосредственно в контейнере. Если

внутри них тоже вложены какие-то элементы, на них правила не распространяются.

Например, есть список из карточек, внутри каждой из них — текст и картинка. Если список превратить во флекс-контейнер, сами карточки — элементы списка — станут флекс-элементами и автоматически распределятся по оси. Но текст и картинка внутри каждой карточки не станут флекс-элементами — для этого пришлось бы превращать во флекс-контейнер саму карточку.

Распределение по оси. Когда блок превращается во флекс-контейнер, его содержимое автоматически выстроится вдоль прямой оси. По умолчанию она горизонтальная и «смотрит» слева направо, но это поведение можно изменить с помощью CSS.

Ось можно представить как линию графика, а элементы — как точки на ней. Это упрощает понимание.

Можно изменять расположение элементов на оси, но привязка к ней все равно останется. Это ключевая особенность флексбокса.

Отсутствие строгих параметров. Еще одно важное свойство флекса — чтобы определить положение элементов, не обязательно строго задавать их координаты. Флексбокс автоматически заполняет все возможное пространство внутри флекс-контейнера: для этого он может сжимать или расширять элементы, изменять отступы между ними. Главное — чтобы соблюдались правила, которые задал человек.

Например, можно указать, чтобы крайние элементы в ряду прижимались к краям контейнера, а элементы между ними распределялись равномерно. Это свойство `justify-content: space-between`. В таком случае, каким бы ни был размер экрана пользователя, элементы будут расположены одинаково. Просто на маленьком экране пространство между ними будет меньше, а на крупном — больше. А если экран будет совсем маленьким, элементы сожмутся или перенесутся на другую строку в зависимости от того, какое поведение задал верстальщик.

Это удобно для создания «резиновой» верстки — такой, которая хорошо смотрится на экранах разных размеров. Не нужно прописывать все до пикселя для каждого возможного разрешения: многое делается автоматически. Это и есть гибкость — способность адаптироваться к разным условиям.

Изменение параметров. Для каждого параметра — особенностей отображения и поведения элементов, изменения оси и так далее — есть свое CSS-свойство. Нужно применить его к требуемому флекс-контейнеру или флекс-элементу — прописать это свойство с нужными значениями.

CSS дает возможность изменять свойства в зависимости от условий. Стилиевой язык работает по каскаду: то, что написано ниже в коде, «перекрывает» написанное раньше. Поэтому в нижней части файла стилей можно задать другое поведение, скажем, для крупных экранов — тогда, если расширение экрана пользователя окажется большим, элементы будут вести себя иначе.

Можно, например, изменить направление оси, выравнивание элементов по ней и другие особенности. Можно и вовсе удалить флекс-контейнер:

применить свойство `display` с другим значением, не `flex`. Тогда все правила, связанные с флексом, автоматически перестанут работать.

Какие возможности дает Flexbox

Мы уже говорили, что поведением элементов на оси можно гибко управлять. Это причина, по которой флекс так удобен: достаточно задать набор правил, и флекс-элементы будут автоматически выстраиваться в соответствии с ним. Вот что можно сделать.

Изменять ось. С помощью команды `flex-direction` можно изменить направление оси. У команды есть четыре значения:

- `row` — ряд, горизонтальная линия слева направо;
- `reverse-row` — ряд справа налево;
- `column` — колонка, вертикальная линия сверху вниз;
- `reverse-column` — колонка снизу вверх.

Чаще всего пользуются значениями `row` и `column`. `Row` хорошо подходит для горизонтальных меню, списков и так далее — элементов, которые расположены «в строку». А `column` — для наборов элементов, расположенных вертикально, «в столбик».

Выстраивать элементы по оси. Когда нужное направление оси указано, элементы автоматически выстроятся вдоль него. По умолчанию они прижаты к левому или верхнему краю оси в зависимости от ее направления — горизонтального или вертикального. Но способ построения можно изменить с помощью свойства `justify-content`, выровнять контент.

У него есть такие значения:

- `flex-start` или просто `start` — выравнивать по «начальной точке» оси, то есть по левому краю для `row` или по верхнему для `column`;
- `flex-end` или `end` — выстраивать по «конечной точке», правой или нижней;
- `center` — концентрировать элементы в центре;
- `space-between` — прижимать крайние элементы к левому и правому краям, а элементы между ними распределять равномерно;
- `space-around` — то же самое, но с отступами от левого и правого края для крайних элементов. Каждый отступ от края в два раза меньше, чем отступы между элементами;
- `space-evenly` — то же самое, но отступы от краев такие же, как между элементами.

На практике чаще всего используют значения `start`, `center` и `space-between`. Последнее удобно, например, для создания горизонтальных меню — они должны тянуться по всей ширине страницы. Но тут есть нюанс, о котором мы поговорим позже.

Выравнивать элементы. Можно выстроить элементы не только вдоль оси, но и «поперек» нее: для `row` — по вертикали, для `column` — по горизонтали. Поведение описывается свойством `align-items`.

Его значения похожи на `justify-content`, но все же отличаются:

- `stretch` — элементы растягиваются поперек оси и заполняют все свободное пространство по обе стороны от нее;

- `start` — элементы сохраняют свой размер и выравниваются по верхнему или левому краю;
- `center` — элементы группируются по центру. Можно сказать, что их центральная часть «закреплена» на оси;
- `end` — элементы выравниваются по нижнему или правому краю;
- `baseline` — выравнивание происходит по «базовой линии» контента.

Помните, что в этом случае речь идет не об основном, а о вспомогательном направлении. То есть, если ось идет горизонтально (`row`), то свойство `align-items` будет распределять элементы в ряду по вертикали, и наоборот — для `column`.

Изменять поведение контента и элемента. Кроме `align-items`, также есть похожие свойства, но управляющие немного другими сущностями:

- `order` определяет порядок расположения — где будет находиться конкретный элемент;
- `align-content` похоже на `align-items`, работает только для многострочных контейнеров — там, где элементы переносятся на другую строку. Это свойство определяет расположение строк относительно контейнера. Значения у него такие же, как у `justify-content`;
- `align-self` — это `align-items` для отдельного элемента. Свойство задается не контейнеру, а одному из элементов внутри него. Оно «переписывает» значение `align-items` для конкретного элемента. Например, все остальные могут быть растянуты по `stretch`, а один, измененный с помощью `align-self`, — располагаться по центру (`center`).

Напрашивается идея с `justify-self`, но во флексбоксе такое свойство не работает — расположение каждого из элементов на оси слишком зависит от других. Зато оно есть в других способах распределения.

Управлять переносом элементов. Что, если элементов станет слишком много и они не поместятся на оси? В таком случае есть два варианта поведения:

- элементы автоматически сожмутся, чтобы все уместились в одном ряду или колонке;
- элементы не будут сжиматься, а излишки перенесутся на другую строку / в другую колонку.

Этим поведением управляет свойство `flex-wrap`. По умолчанию его значение — `nowrap`, то есть не переносить, а сжимать. Если указать значение `wrap`, элементы будут переноситься на другую строку, а `wrap-reverse` — будут переноситься, но в противоположном направлении. То есть новая строка возникнет не снизу, а сверху.

Какое поведение выбрать — зависит от ситуации. Иногда элементы просто не могут достаточно сжаться, и тогда при `nowrap` они «сломаются» и «выпадут» за пределы сетки. В таком случае лучше выбирать `wrap`. А иногда разработчик точно знает, что добавлять новые элементы в это место не будут, — тогда подойдет и `nowrap`.

Управлять сжатием и расширением. Выше мы сказали, что элементы внутри флекс-контейнера могут изменять размер: сжиматься и расширяться в зависимости от параметров. Это удобно, но иногда бывает

нужно изменить сжатие и расширение по умолчанию. Тут тоже есть специальные свойства:

- `flex-grow` — управляет расширением отдельного элемента. Значение задается как целое положительное число, по умолчанию оно 1 для каждого элемента. Если увеличить это значение, элемент начнет «требовать» себе больше места на оси, чем другие;
- `flex-shrink` — управляет сжатием отдельного элемента. По умолчанию значение для всех элементов — 1. Если поставить его как 0, элемент вообще не будет сжиматься, а если увеличить до 2, 3 и так далее — будет сжиматься сильнее других;
- `flex-basis` — устанавливает «базовый» размер элемента, относительно которого он будет расширяться и сжиматься.

Эти свойства нужны, когда важно сделать какой-то элемент больше или меньше других.

Для каких элементов обычно используют Flexbox

Чаще всего флексы применяют там, где нужна «одномерная» сетка, то есть надо управлять расположением элементов только в одном направлении. Обычно это микросетки, мелкие элементы внутри какого-то большого блока. Для разметки крупных блоков на странице целиком флекс используют редко — это зачастую не очень удобно.

Кроме флексов, есть еще два способа располагать элементы. Первый — обычный блочный метод, когда блоки просто следуют друг за другом по вертикали: отступы между ними прописываются вручную, ничего не сжимается и не расширяется. Второй способ — грид-контейнеры, еще

одна возможность CSS 3.0. С их помощью можно создавать двумерные сетки и по-разному располагать элементы внутри: например, вверху один широкий блок, а внизу два или три маленьких.

Обычно для каждого блока верстальщик сам определяет, флексом он будет описываться или гридом. Ни один из этих способов не лучше — они просто предназначены для разных случаев.

Преимущества Flexbox

Гибкость. Возможность гибко управлять положением элементов — одно из главных преимуществ флекса. Благодаря ей он намного удобнее предыдущих способов построения сеток и позволяет не задумываться о размерах элементов вплоть до пикселя. С помощью Flexbox можно настроить автоматическое выравнивание и распределение по оси, а не задавать все эти параметры вручную.

Адаптивность. Flex отлично подходит для создания «резиновой» верстки, когда размеры элементов меняются в зависимости от параметров экрана. Такая верстка — хорошая практика по сравнению со статической, потому что лучше выглядит на разных устройствах. У нее ниже риск «развалиться» и стать нефункциональной.

Удобное выравнивание. Выравнивание во флексбоксе — мощный и очень удобный инструмент. С его помощью можно в пару строк реализовать вещи, которые без флекса отняли бы куда больше усилий. Поэтому иногда флексбокс используют даже не по назначению — например, для точного центрирования контента внутри блока.

Хорошая поддержка в браузерах. Flexbox появился довольно давно, и сейчас его поддерживают все современные браузеры. Поддержка отсутствует только в очень старых версиях интернет-обозревателей, которыми сейчас мало кто пользуется. В большинстве компаний считается, что поддерживать настолько старые браузеры уже не стоит.

Адаптация под разные языки. Выше мы рассказывали про значения `reverse-row`, `reverse-column` и `reverse-wrap` и говорили, что они используются редко. Но это справедливо только для латиницы и кириллицы. Во многих языках практикуется чтение справа налево — эти свойства отлично подходят для адаптации веб-страниц под такие языки. Особенно удобно, если страница существует на разных языках с разными способами чтения: например, на английском и фарси.

Недостатки Flexbox

Риск «выпадения». Флекс не универсален. Большинство его минусов связано с позиционированием элементов. Из-за того, что оно автоматическое, возможны ситуации, когда что-то ломается или выпадает. Если такое случится, верстка станет непригодной к использованию.

Пример — тот же `nowrap`. Если перенос отключен, элементы автоматически сжимаются, и до определенного предела это выглядит красиво и удобно. Но если они слишком маленькие, пользователю становится неудобно. А если их много, а сжать уже не получается, невлезшие элементы «вывалятся» за пределы контейнера. Они могут даже уйти за пределы экрана, и тогда ими нельзя будет пользоваться.

Таких ситуаций можно избегать — заранее предусматривать в верстке защиту от переполнения. Например, сразу прописывать возможность переноса и его параметры.

Риск «расползания верстки». Есть и еще одна ситуация — явление, когда при определенных условиях элементы начинают располагаться некрасиво. Выше мы рассказывали, что свойство `space-between` хорошо подходит для «растягивания» элементов в горизонтальном меню и что тут есть деталь. Это один из примеров такого поведения.

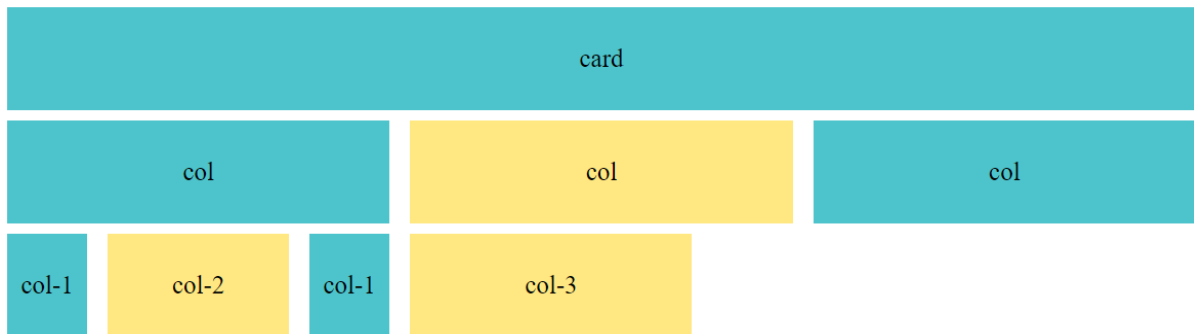
Горизонтальное меню со `space-between` смотрится отлично, пока его пунктов не становится много. Включается защита от переполнения, и лишние пункты переносятся на другую строку. И тут `space-between` играет плохую службу: если лишних пунктов два, они «приклеиваются» к краям контейнера, а по центру остается пустая область. А изменить расположение элементов только по одной строке нельзя.

Поэтому стоит заранее думать о возможных изменениях в верстке и предусматривать такие моменты, чтобы флекс-контейнеры нормально смотрелись при любых разрешениях и условиях.

Сложности с двумерными сетками. С помощью флексбокса сложно создать функциональную и красивую двумерную сетку — такую, в которой элементы расположены не вдоль линии, а как бы на плоскости. Но для этого есть грид, поэтому в таких ситуациях предпочтительнее использовать его.

Создание мелкой сетки компонентов на флексах.

Итогом этого раздела, станет создание вот такой сетки на flexbox:



Подготовка проекта

Давайте для начала создадим стандартные стили для нашего проекта:

```
* {
```

```
  box-sizing: border-box;
```

```
}
```

```
body {
```

```
  margin: 0;
```

```
  background-color: #fff;
```

```
}
```

```
.container {  
  
margin: 0 auto;  
  
max-width: 1200px;  
  
padding: 0 20px;  
  
}
```

Свойство `box-sizing: border-box`; изменяет алгоритм расчета размеров элемента.

Для `body` убираем стандартный отступ и делаем ему фон.

Также создаем `.container`, которому мы зададим максимальную ширину, добавим ему отступы по краям, чтобы контент не прилипал к краям экрана и пишем `margin: 0 auto`; для того, чтобы отцентровать наш контейнер по центру

Создание карточки

Также создадим карточку (какой-то блок, который вы будите в дальнейшем верстать внутри элементов вашей сетки):

```
.card-1 {  
background-color: #4dc4cc;  
}  
.card-2 {
```



```
background-color: #ffe882;
}
[class*="card-"] {
display: flex;
align-items: center;
justify-content: center;
margin-top: 10px;
height: 100px;
}
```

Тут все достаточно просто, мы создаем стили для карточки, она имеет высоту (чего не стоит делать в реальном проекте) и позиционирование всех дочерних элементов по центру, а также 2 разных фона, чтобы отличать их.

Создание сетки

Теперь давайте приступим к написанию самой сетки.

Создадим `.col` это будет класс одной колонки, которая будет поровну делить свободное место с другими колонками:

```
.col {
padding: 0 10px;
flex-grow: 1;
}
```

Свойство `padding: 0 10px;` задает отступы между колонками. Так, как колонки будут друг друга касаться, то в данной ситуации отступ между колонками будет не по 10 пикселей, а по 20.

Теперь создадим класс `.grid`, который будет иметь следующий вид:

```
.grid {  
display: flex;  
flex-wrap: wrap;  
}
```

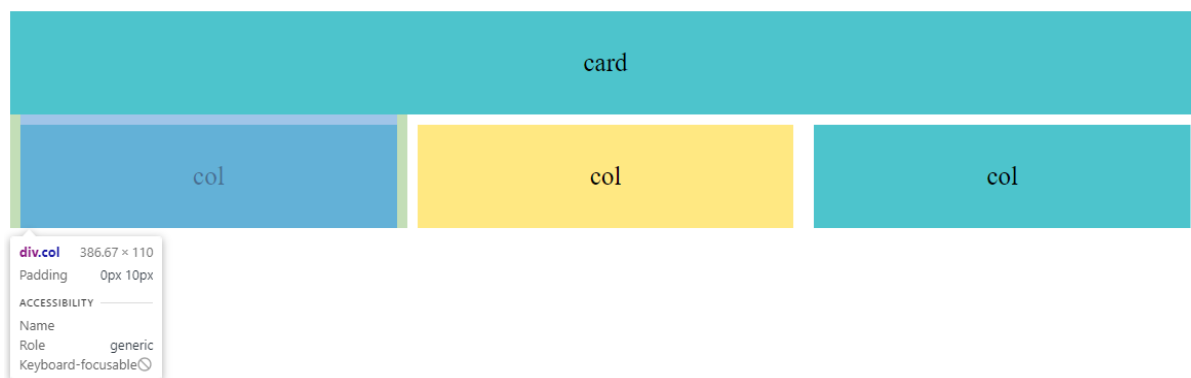
`.grid` является flex контейнером, в котором все элементы будут идти слева направо. С помощью свойства `flex-wrap: wrap;` мы указываем, что элементы нашего контейнера могут переноситься, если не будут влезать в нашу flex строку.

Теперь давайте посмотрим на нашу сетку, сделав вот такую вертску:

Теперь давайте посмотрим на нашу сетку, сделав вот такую вертску:

```
<div class="container">  
<div class="card-1">card</div>  
</div>  
<div class="container">  
<div class="grid">  
<div class="col"><div class="card-1">col</div></div>  
<div class="col"><div class="card-2">col</div></div>  
<div class="col"><div class="card-1">col</div></div>  
</div>  
</div>
```

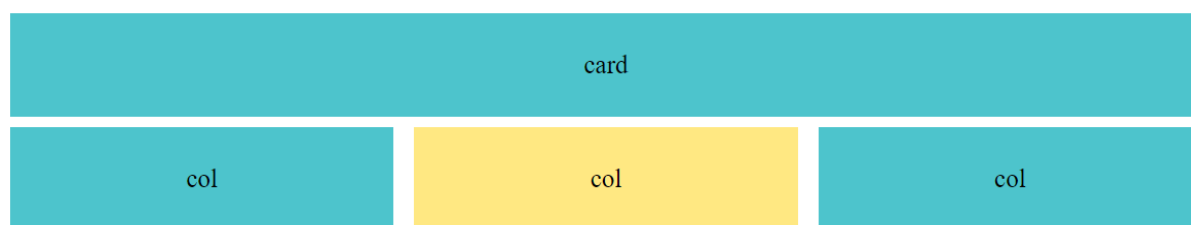
В результате мы увидим вот такой результат:



В этой верстке первый контейнер является контрольным, чтобы проверить, что наша сетка занимает весь контейнер без остатка. Как мы видим, сейчас у нас есть ошибка, так как наш класс col имеет отступ с обеих сторон он еще и отталкивается от стенок нашего контейнера на те самые 10 пикселей, которые мы указали в padding. Для того, чтобы это исправить давайте просто укажем отрицательный margin для нашего класса grid:

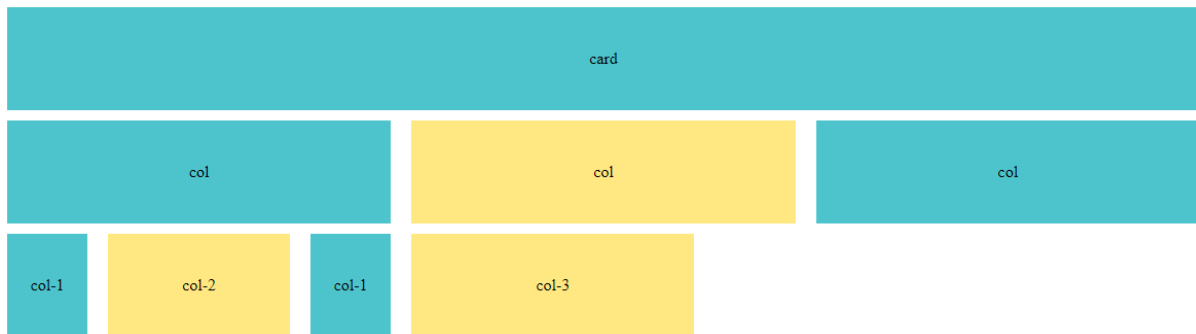
```
.grid {  
display: flex;  
flex-wrap: wrap;  
margin: 0 -10px;  
}
```

Теперь давайте проверим, что в браузере все корректно отображается:



Создание сложных сеток

Теперь давайте поговорим о более сложных сетках. Как правило бывают ситуации, когда блоки в дизайне выглядят так:



Во-первых нам надо знать, что в сетке всегда есть 12 колонок. То есть наш блок может занимать от 1 до 12 колонок в ширину. У нас это будут классы, которые будут называться col-2, где 2 - это количество колонок, которое будет занимать наш блок.

Теперь давайте напишем классы для наших блоков:

```
.col-1 {  
padding: 0 10px;  
width: calc(100% / 12);  
}  
.col-2 {  
padding: 0 10px;  
width: calc(100% / 12 * 2);  
}  
.col-3 {  
padding: 0 10px;  
width: calc(100% / 12 * 3);
```

```
}  
.col-4 {  
padding: 0 10px;  
width: calc(100% / 12 * 4);  
}  
.col-5 {  
padding: 0 10px;  
width: calc(100% / 12 * 5);  
}  
.col-6 {  
padding: 0 10px;  
width: calc(100% / 12 * 6);  
}
```

Для того, чтобы указать ширину блока мы будем высчитывать ширину 1 нашей колонки, после чего умножать на число колонок, которое должен занимать блок, для этого воспользуемся функцией `calc` из `css`, в дальнейшем хорошо было бы написать это сразу в процентах, чтобы стили применялись быстрее. Также по аналогии дописываем остальные классы до 12.

Теперь давайте создадим более сложную верстку и посмотрим на результат:

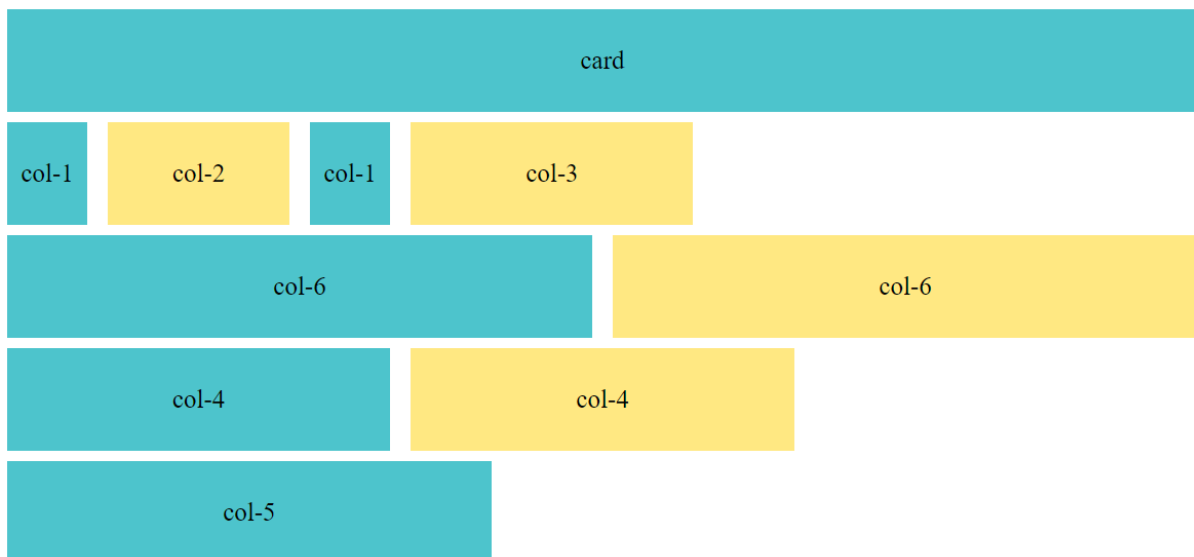
```
<div class="container">  
<div class="card-1">card</div>  
</div>  
<div class="container">  
<div class="grid">
```

```

<div class="col-1"><div class="card-1">col-1</div></div>
<div class="col-2"><div class="card-2">col-2</div></div>
<div class="col-1"><div class="card-1">col-1</div></div>
<div class="col-3"><div class="card-2">col-3</div></div>
<div class="col-6"><div class="card-1">col-6</div></div>
<div class="col-6"><div class="card-2">col-6</div></div>
<div class="col-4"><div class="card-1">col-4</div></div>
<div class="col-4"><div class="card-2">col-4</div></div>
<div class="col-5"><div class="card-1">col-5</div></div>
</div>
</div>

```

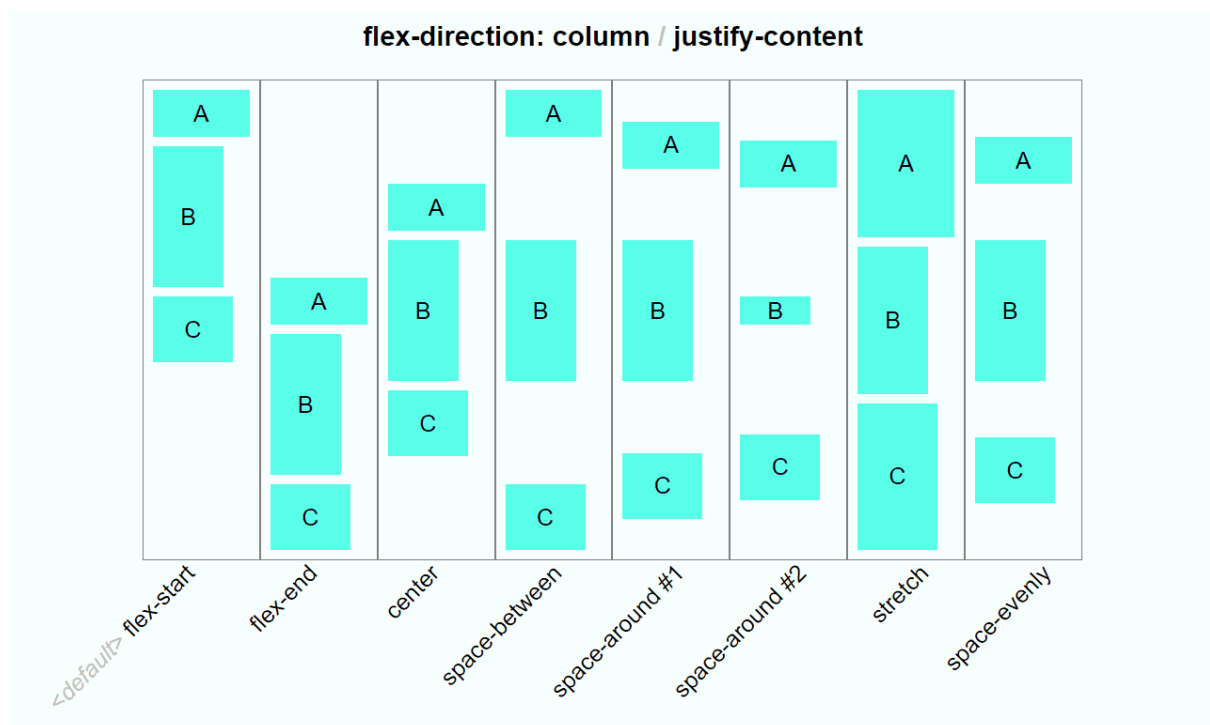
Также посмотрим на результат:



Как мы видим, то у нас вывелась наша сетка, давайте посмотрим на принцип ее работы на основе первой строки сетки. У нас есть `.col-1 + .col-2 + .col-1 + .col-3 = 7` колонок. Из-за чего блок с шириной в 6 колонок

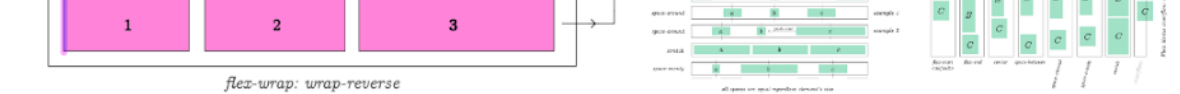
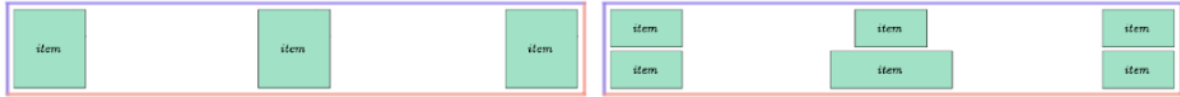
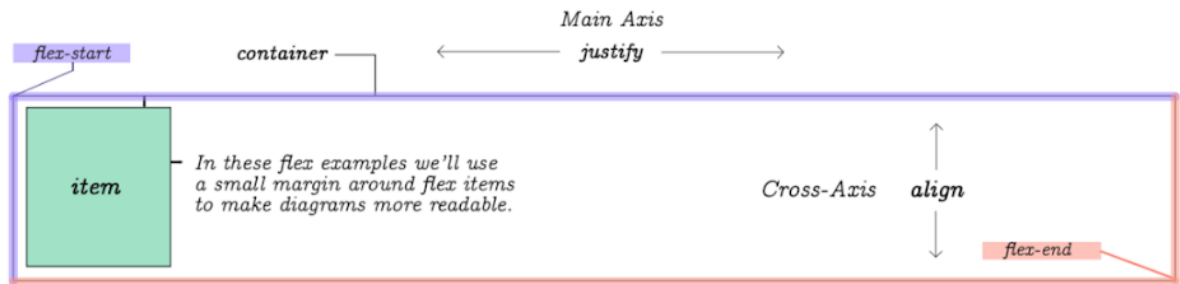
переносится на новую строку, так как $7 + 6 = 13$, а мы помним, что наша сетка содержит только 12 колонок. Остальное вы можете проверить по аналогии.

Тонкости и ограничения флексов: порядок флекс-элементов, многострочный флекс-контейнер и расположение строк внутри него, расчет размеров флекс-элементов.



Как и CSS Grid, Flex Box довольно сложен, потому что состоит из двух составляющих: контейнера и элементов внутри него.

Поэтому прилагаются диаграммы свойств Flex



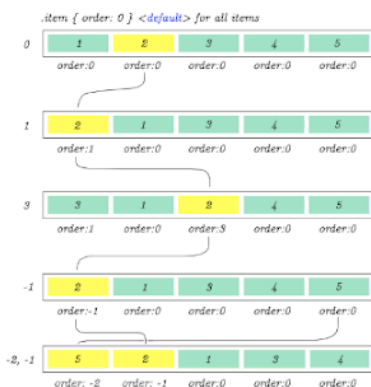
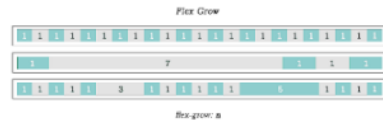
flex-shrink: 7 (shrink by 7 times more than the rest of items)

`flex-basis: auto;`

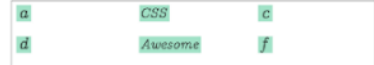
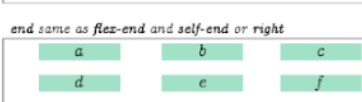
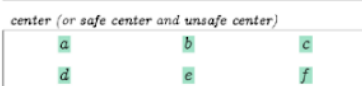
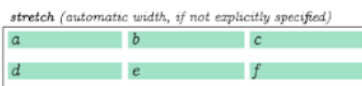
`a CSS Is Awesome b c`

`flex-basis: 50px;`

`a CSS Is Awesome b c`



normal / auto <default> also same as start and flex-start or self-start or left



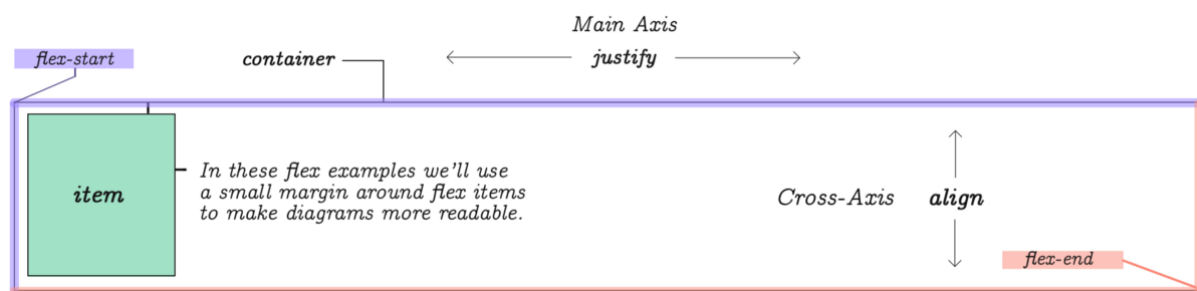
Давайте рассмотрим каждую диаграмму отдельно. К концу этого руководства по Flex вы узнаете обо всех его возможностях.

CSS Flex или Flex Box

Flex — это набор правил для автоматического растягивания нескольких столбцов и строк внутри родительского контейнера.

display:flex

В отличие от многих других свойств CSS, в Flex есть основной контейнер и вложенные в него элементы. Некоторые свойства CSS-Flex относятся только к контейнеру. А другие можно применить только к элементам внутри него.



Вы можете думать о flex-элементе как о родительском контейнере со свойством `display: flex`. Элемент, помещенный в контейнер, называется `item`. Каждый контейнер имеет границы начала(`flex-start`) и конца гибкости(`flex-end`), как показано на этой диаграмме.

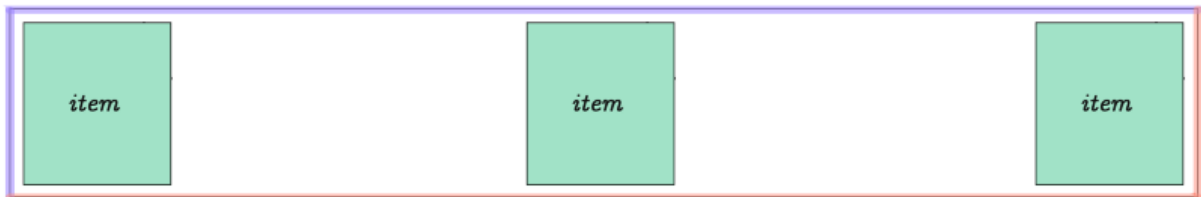
Горизонтальная (main) и вертикальная (cross) оси

Хотя список элементов представлен линейно, необходимо обращать внимание на строки и столбцы. По этой причине Flex включает в себя координатные оси. Горизонтальная ось называется `main-axis`, а вертикальная — `cross-axis`.

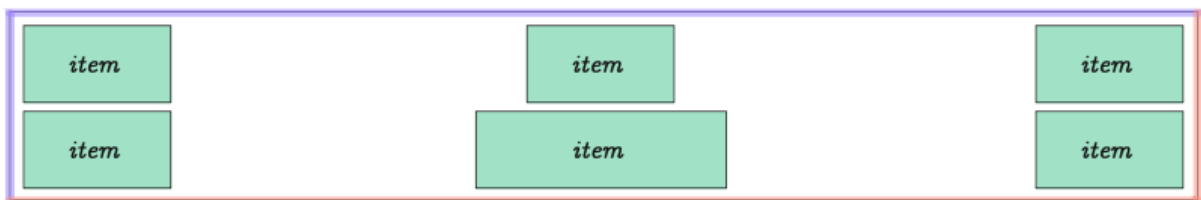
Чтобы управлять шириной содержимого контейнера и промежутками между элементами, которые растягиваются вдоль `main-axis`, необходимо использовать

Justify-content. Для управления вертикальными изменениями элементов необходимо использовать align-items.

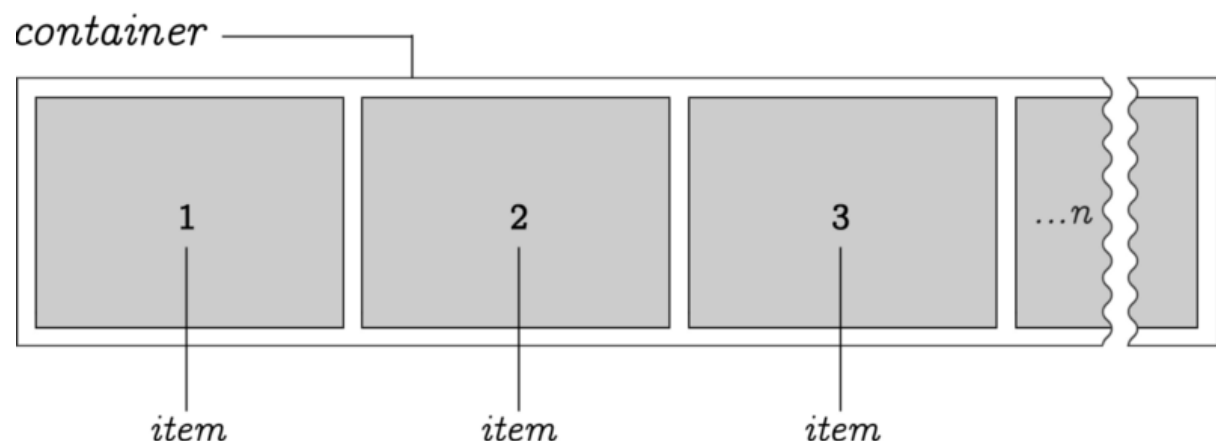
Если у вас есть 3 столбца и 6 элементов, Flex автоматически создаст вторую строку для размещения оставшихся элементов. Если у вас в списке более 6 элементов, будут созданы дополнительные строки.



По умолчанию, элементы Flex равномерно распределяются внутри контейнера по горизонтальной оси. Мы рассмотрим различные свойства и значения.



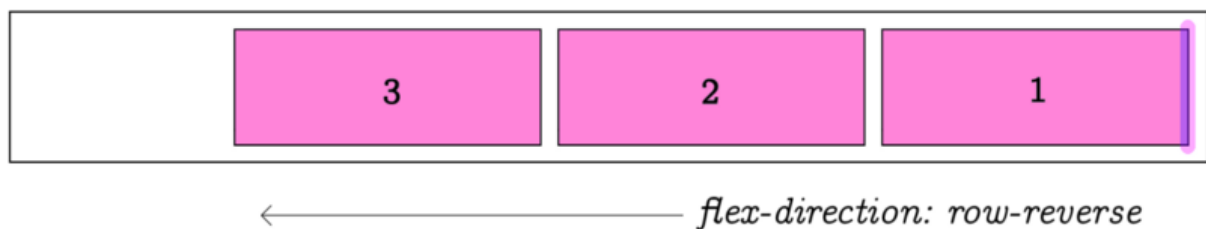
Распределение строк и столбцов внутри родительского элемента определяется свойствами CSS Flex flex-direction, flex-wrap и некоторыми другими, которые будут продемонстрированы дальше.



:У нас есть произвольное n-количество элементов, расположенных в контейнере. По умолчанию элементы растягиваются слева направо. Однако направление можно изменить.

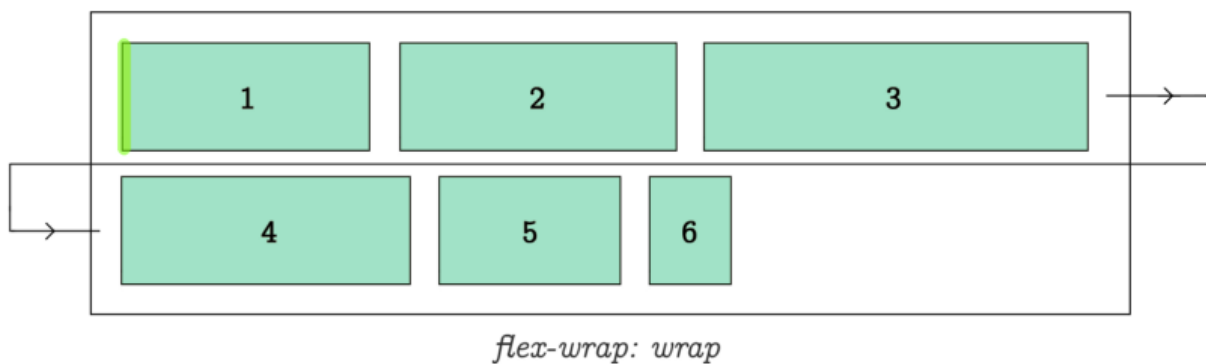
Direction

Можно задать направление движения элементов (по умолчанию слева направо).



`flex-direction: row-reverse` изменяет направление движения списка элементов. По умолчанию стоит значение `row`, что означает движение слева направо.

Wrap



`flex-wrap: wrap` определяет перенос элементов на другую строку, когда в родительском контейнере заканчивается место.

Flow

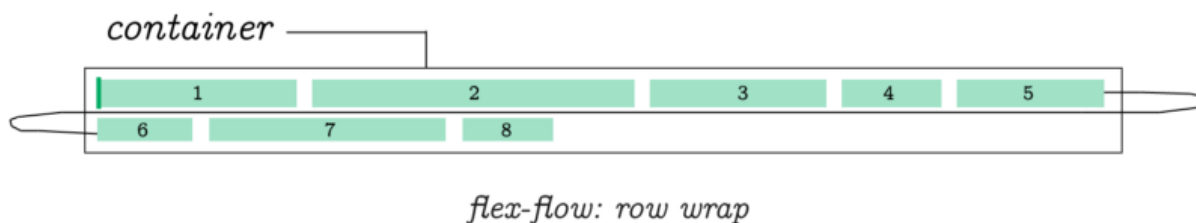


flex-flow: flex-direction<value> flex-wrap<value>

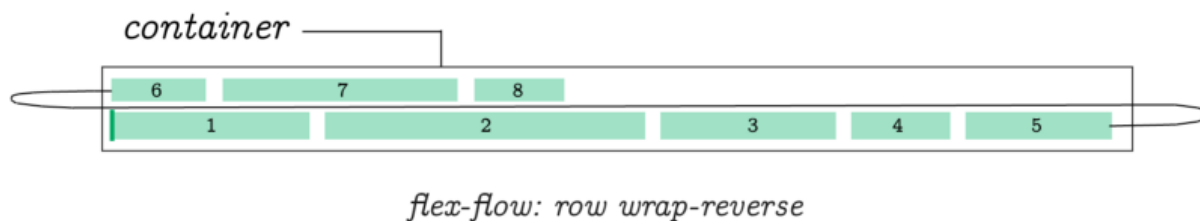
flex-flow включает в себя flex-direction и flex-wrap, что позволяет определять их с помощью одного свойства.

Примеры:

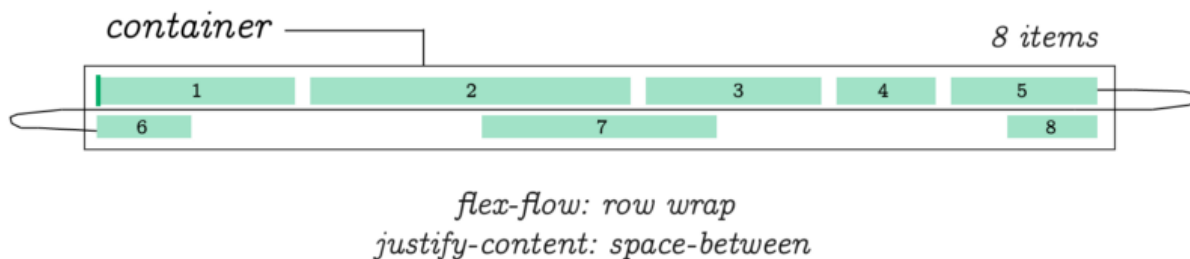
- flex-flow: row wrap определяет значения flex-direction как row и flex-wrap как wrap.



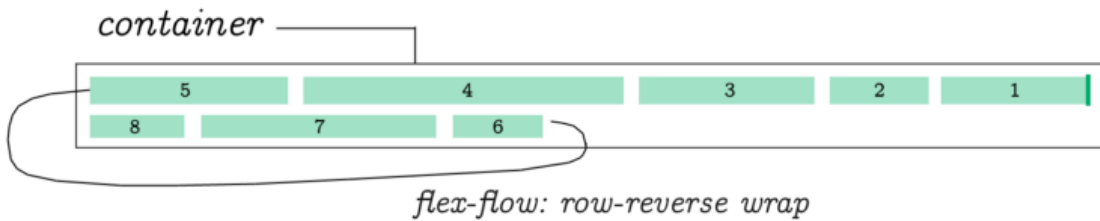
- flex-flow:row wrap-reverse (перенос элементов вверх)



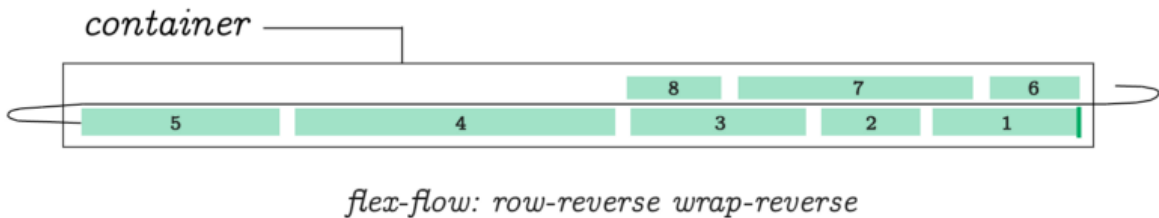
- flex-flow:row wrap (стандартный перенос элементов); justify-content: space-between (пробел между элементами);



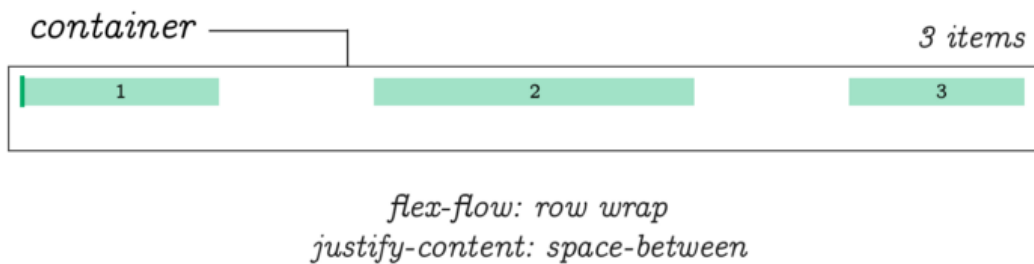
- flex-flow: row-reverse wrap (направление движения справа налево со стандартным переносом сверху вниз)



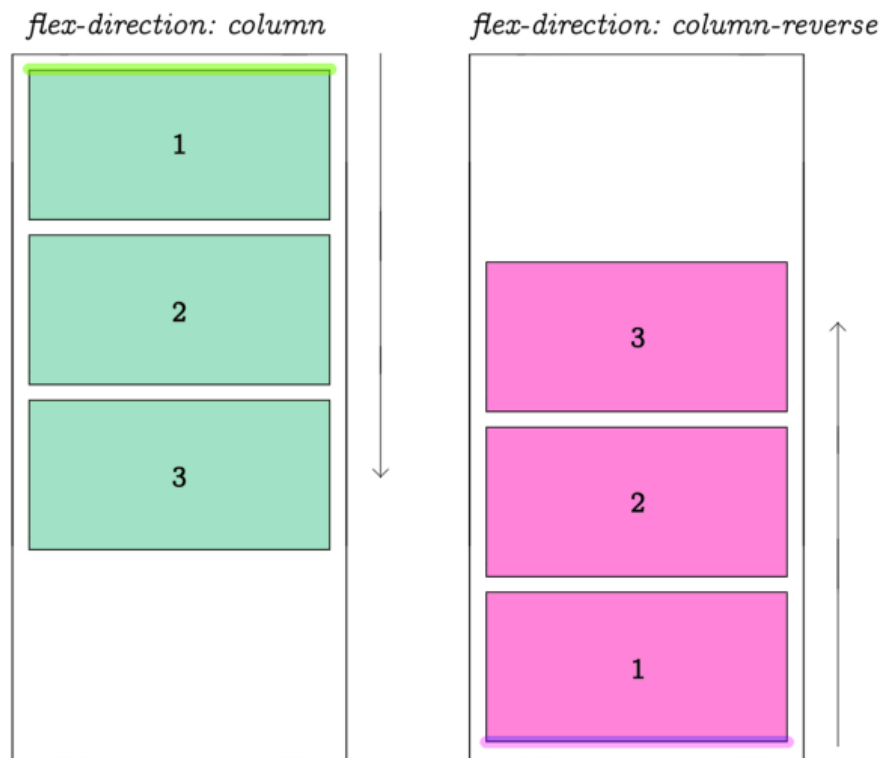
- flex-flow: row-reverse wrap-reverse (направление движения справа налево и обратный перенос элементов);



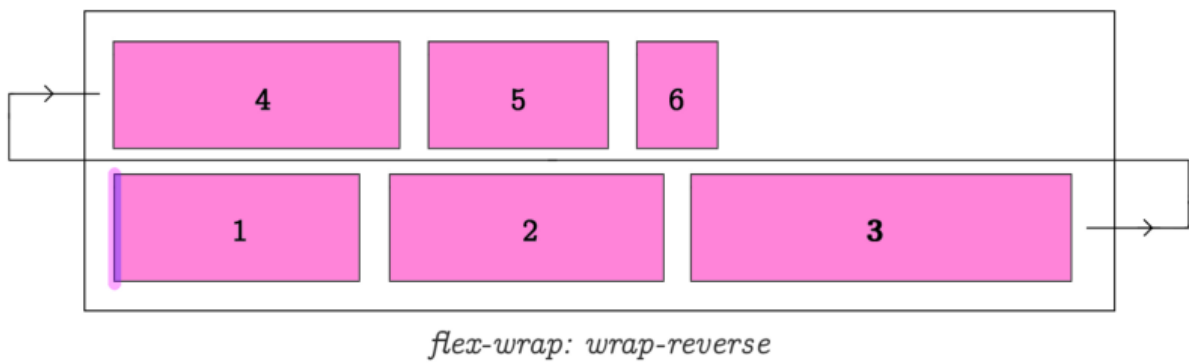
- flex-flow: row wrap; justify-content: space-between; (стандартный перенос и направление; расстояние между элементами)



Направление можно изменить, чтобы сделать вертикальную ось главной.



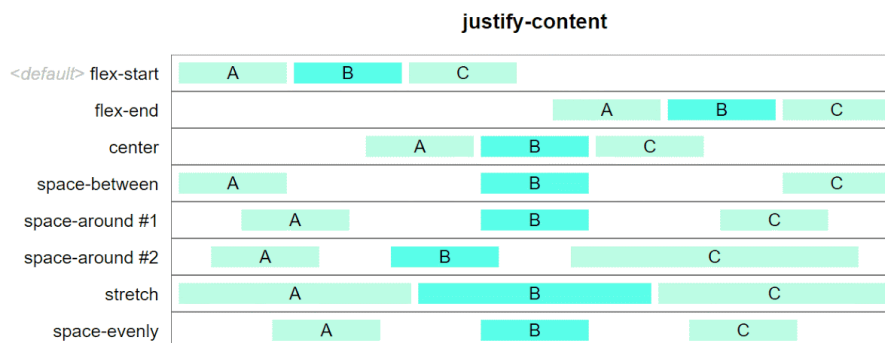
Когда мы меняем `flex-direction` на `column`, свойство `flex-flow` ведет себя точно так же, как и в предыдущих примерах. За исключением `wrap-reverse`, когда элементы переносятся снизу вверх.



`justify-content`



Анимация для прояснения того, как работает пример выше

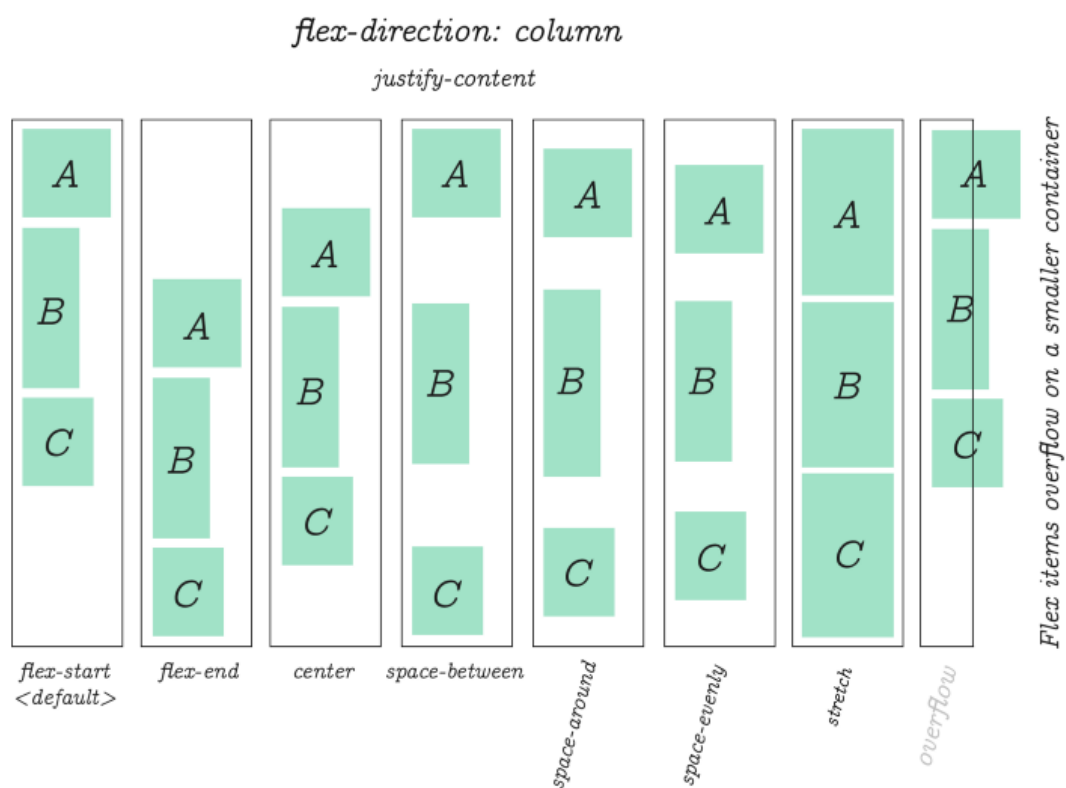


Анимированные возможности justify-content.

Надеюсь, эта CSS-анимация поможет лучше понять работу justify content.

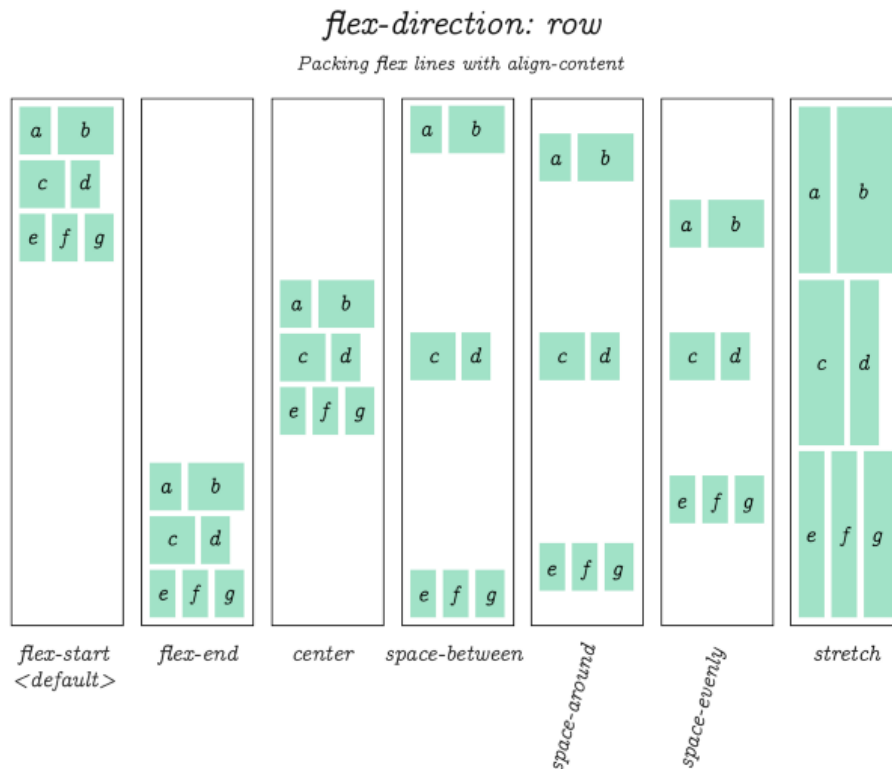
Свойства следующие: flex-direction: row; justify-content: flex-start | flex-end | center | space-between | space-around | stretch | space-evenly. В этом примере мы используем только 3 элемента в строке.

Нет никаких ограничений на количество элементов, которые можно использовать в Flex. Эти диаграммы демонстрируют только поведение элементов, когда одно из перечисленных значений применяется к свойству justify-content.



То же свойство `justify-content` используется для выравнивания элементов, когда `flex-direction` is `column`.

Packing Flex Lines (согласно спецификации Flex)

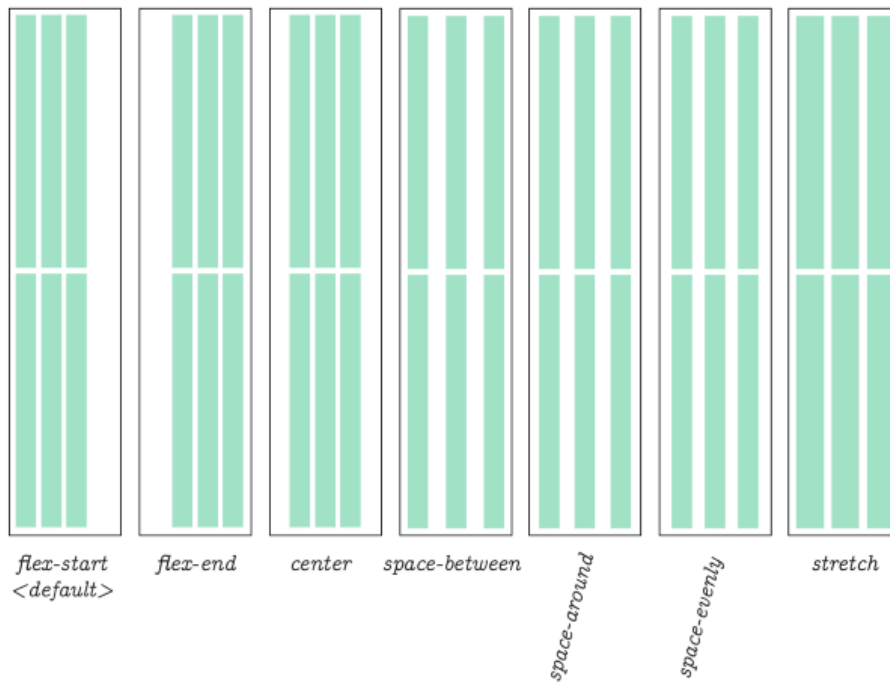


Не уверен, реализовано ли это в каком-либо браузере, но когда-то это было частью спецификации CSS-flex и, вероятно, стоит упомянуть об этом для полноты картины.

В спецификации Flex это называется «*Packing Flex Lines*». По сути, это работает так же, как в примерах выше. Однако стоит обратить внимание, что интервал расположен между целыми группами элементов. Это полезно, когда вы хотите создать зазоры между группами.

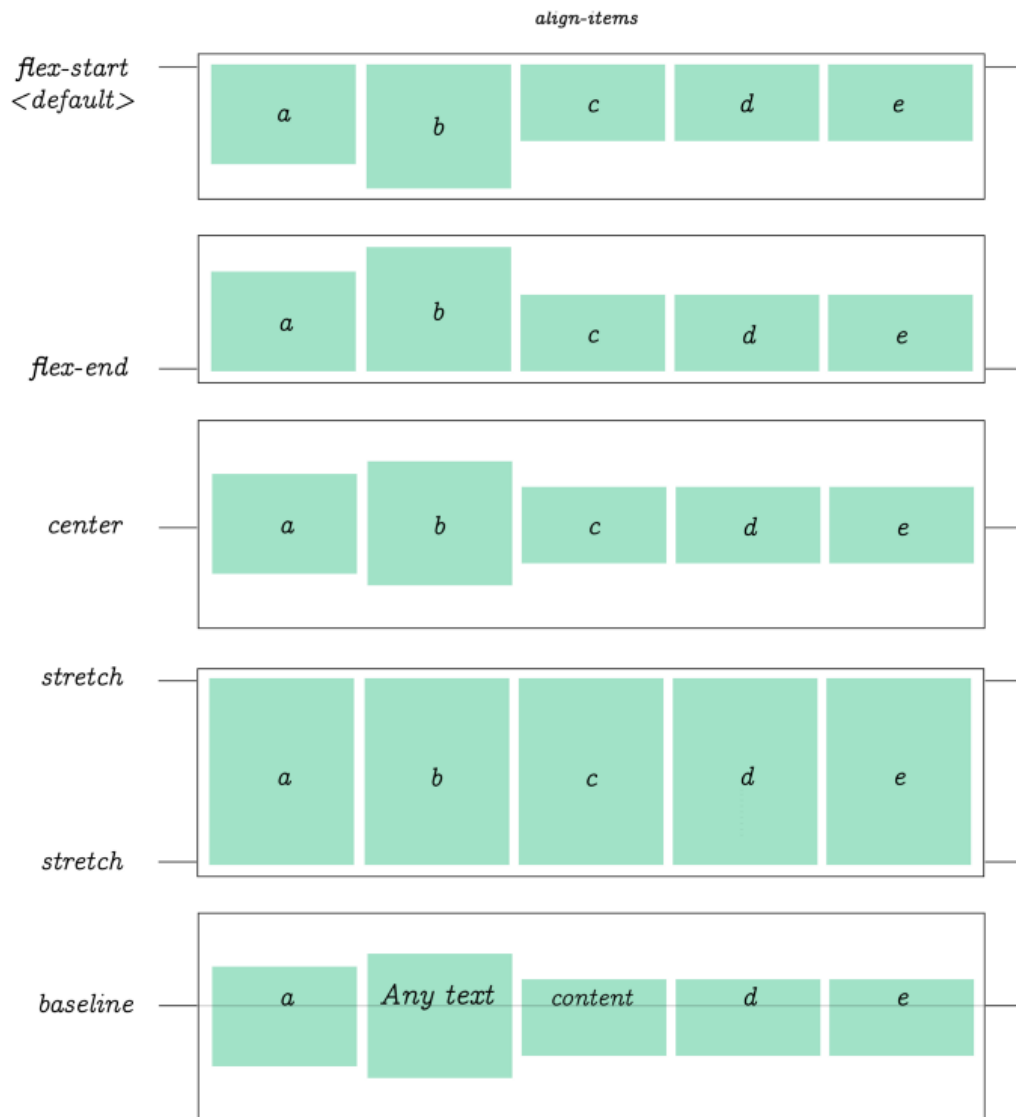
flex-direction: column

Packing flex lines with align-content



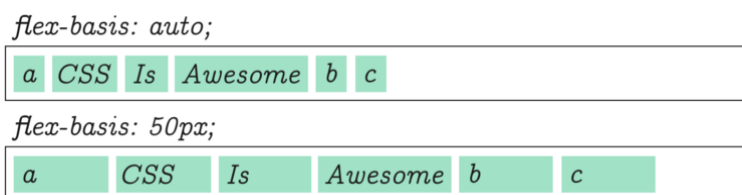
Packing Flex Lines, но теперь с `flex-direction: column`

`align-items`



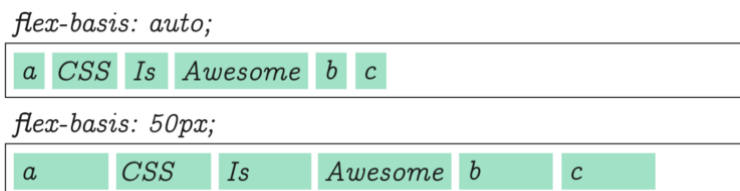
align-items контролирует выравнивание элементов по горизонтали относительно родительского контейнера.

flex-basis



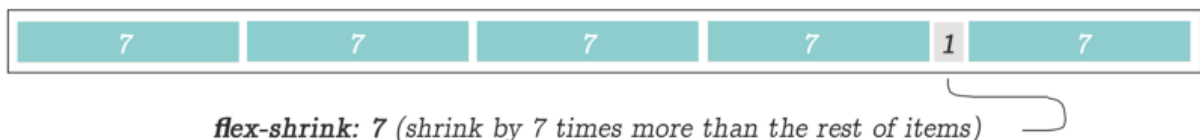
`flex-basis` работает аналогично другому свойству CSS: `min-width`. Оно увеличивает размер элемента в зависимости от содержимого. Если свойство не задействуется, то используется значение по умолчанию.

`flex-grow`



`flex-grow` применяется к конкретному элементу и масштабирует его относительно суммы размеров всех других элементов в той же строке, которые автоматически корректируются в соответствии с заданным значением свойства. В примере значение `flex-grow` для элементов было установлено на 1, 7 и (3 и 5) в последней строке.

`flex-shrink`



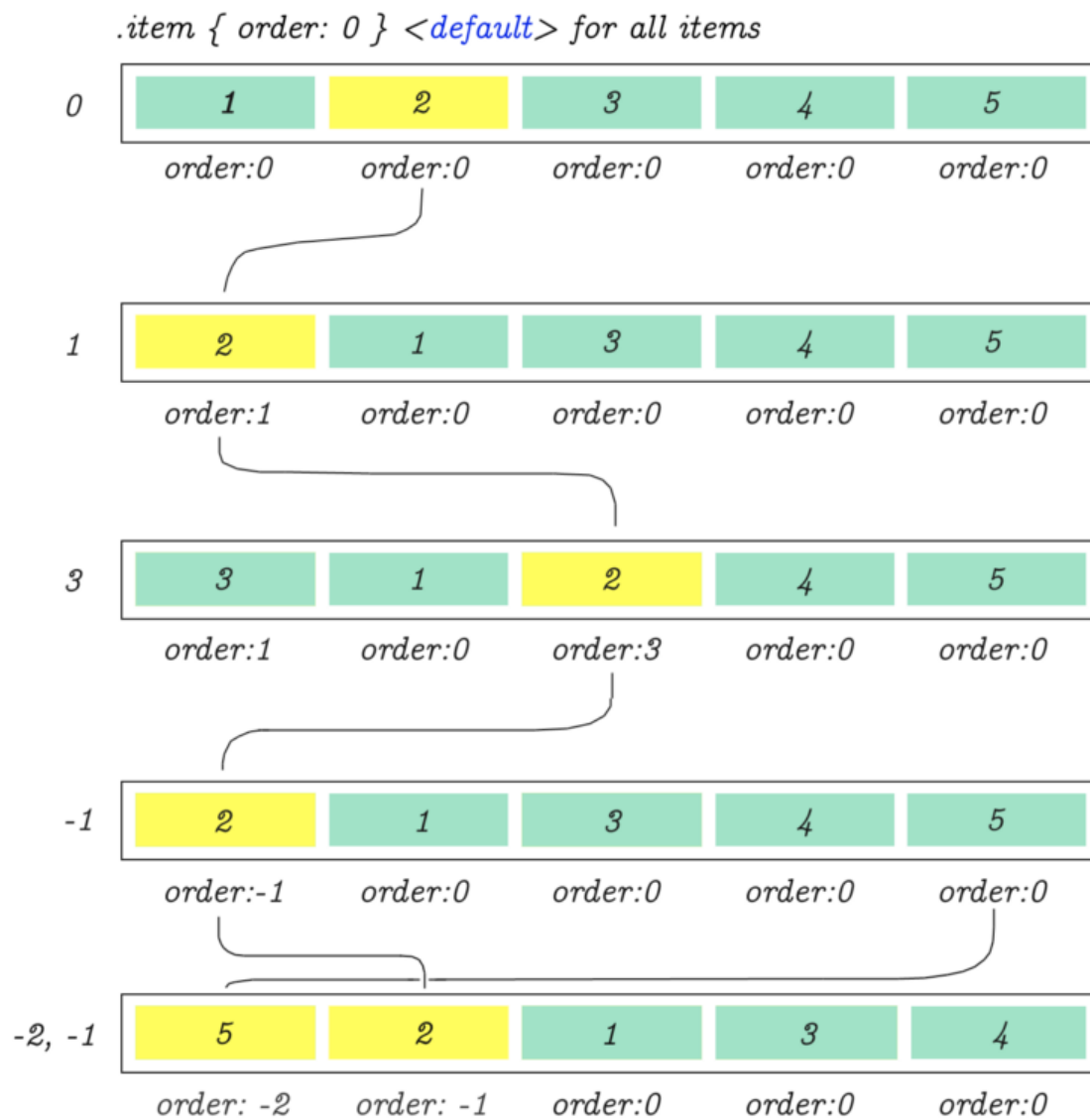
`flex-shrink` — это противоположность `flex-grow`. В примере значение `flex-shrink` равно 7. При таком значении размер элемента равен 1/7 размера окружающих его элементов (размер которых автоматически скорректирован).

```
.item { flex: none | [ <flex-grow> <flex-shrink> || <flex-basis> ] }
```

При работе с отдельными элементами можно использовать только одно свойство из трёх: `flex-grow` , `flex-shrink` или `flex-base`.

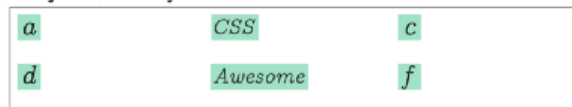
`order`

Используя свойство `order` , можно изменить естественный порядок элементов.

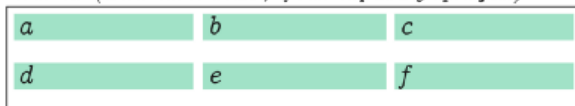


`justify-items`

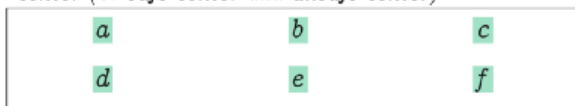
normal / auto <default> also same as start and flex-start or self-start or left



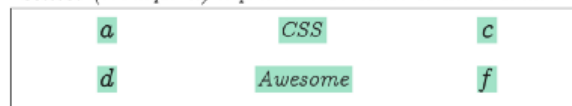
stretch (automatic width, if not explicitly specified)



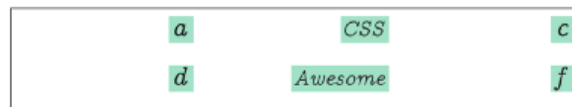
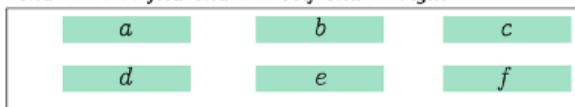
center (or safe center and unsafe center)



center (example 2) expands based on content's width



end same as flex-end and self-end or right



И последнее, что нужно для тех, кто хочет использовать CSS Grid вместе с Flex Box... justify-items в CSS Grid похожи на justify-content в Flex. Свойства, описанные на приведенной выше диаграмме, не будут работать в Flex, но в значительной степени эквивалентны сетке для выравнивания содержимого ячеек.

При использовании Flex нужно учитывать, что:

- Flex-элементы по умолчанию ужимаются под свое содержимое; Это может пригодиться, когда есть блоки, размер которых изначально неизвестен и зависит от количества контента.
- Внешние отступы flex-элементов не схлопываются и не выпадают, в отличие от блочной модели.
Схлопывание и выпадение отступов полезно, в основном, при верстке текстовых страниц, поэтому во избежание путаницы в отступах при создании сеток отсутствие такого поведения приходится очень кстати.
- Flex-элементы умеют перераспределять свободное пространство вокруг себя, таким образом меняя свои размеры;

Больше не нужно вручную задавать ширину в процентах флекс-элементам, если необходимо равномерно заполнить все пространство флекс-контейнера. Браузер сам рассчитает размеры и распределит элементы по флекс-контейнеру.

- Внутри одного флекс-контейнера можно менять порядок флекс-элементов, хотя в DOM-дереве порядок остается неизменным. Бывают макеты страниц, в которых порядок следования элементов отличается на мобильной и десктопной версиях. Flex справится с задачей.
- Флекс-элементу можно задать не только горизонтальные автоматические отступы, но и вертикальные; к тому же есть специальные свойства, с помощью которых очень просто выравнивать элементы внутри флекс-контейнера по горизонтали или вертикали.
- Флекс-элементы могут переноситься на следующую строку, если указано соответствующее свойство.
Эта особенность полезна в каталогах и различных списках, где изначально количество всех элементов неизвестно. Если их станет слишком много и предусмотрен перенос, то страница в этом случае не развалится.

Стоит отметить, что Flexbox поддерживается в Internet Explorer 10-11, хоть и с некоторыми оговорками.

Из недостатков можно отметить то, что Flex не хватает при верстке писем. Не во всех почтовых клиентах он работает корректно.

