Видео недели:

1. [Таблицы](https://www.youtube.com/watch?v=C4TLhf2oy-U)
2. [Блочная модель](https://www.youtube.com/watch?v=2nvptxPBAjw)
3. [Float](https://www.youtube.com/watch?v=N7YNJ6ogNnc)
4. [Уникальное позиционирование](https://www.youtube.com/watch?v=9roqSHkhiv8)
5. [Flexbox](https://youtu.be/Cphwiqa_4Hg)

**Позиционирование**

Позиционированием называется положение элемента в системе координат. Различают несколько видов позиционирования: нормальное, абсолютное, фиксированное и относительное. В зависимости от типа, который устанавливается через свойство position, либо display, изменяются и системы координат, и способы управления позиционированием.

**Табличная верстка (устар.)**

Этот вид позиционирования является устаревшим, поэтому не нужно его использовать, но хорошо знать, что он такой тоже есть (был).

В чем суть этого способа - создается таблица [<table>](https://ru.w3docs.com/uchebnik-html/html-teg-table.html) и ее ячейки наполняются контентом, а размер линий-разделителей таблицы ставиться в 0, чтобы ее не было видно - при этом позиционирование элементов сохраняется.

Технически это довольно простой способ, но так делали лет 20 назад (когда особенно не задумывались о семантике, которую табличная верстка не поддерживает), а на сегодня придумали много интересных альтернатив.

**Блочная модель (используется по умолчанию)**

В соответствии с концепцией блочной модели, каждый элемент на странице представляет собой прямоугольный блок и может иметь ширину, высоту, поля, границы и отступы.

Элементы бывают блочными и строчными - блочные элементы занимают всю доступную ширину независимо от их содержимого и начинаются с новой строки. Строчные элементы занимают ширину, которая требуется для содержимого и выстраиваются на одной строке, друг за другом. Блочные элементы, как правило, используются для больших кусков контента, таких как заголовки и структурные элементы. Строчные элементы, как правило, применяются для маленьких частей содержимого без переноса (например, <span>, <a>).

Свойство **display**

Как именно отображаются элементы — как блочные или строчные или как-то ещё, определяется свойством display. Каждый элемент содержит значение свойства display по умолчанию, но как и с любым другим значением свойств, это значение может быть переписано. Есть немало значений для свойства display, но наиболее распространённые это block, inline, inline-block и none.

1. Мы можем изменить значение свойства display элемента, выбрав этот элемент в CSS и задав новое значение свойства display. Значение block сделает этот элемент блочным (по умолчанию обычно элементы итак блочные - и это свойство можно не задавать).
2. p {
3. display: block;
4. }
5. Значение inline сделает этот элемент строчным.
6. p {
7. display: inline;
8. }
9. Самое интересное - это значение inline-block. Его использование позволит элементу вести себя как блочный, включая все свойства блочной модели (которые мы вскоре рассмотрим). Но элемент будет отображаться на строке с другими элементами, а не будет начинаться с новой строки по умолчанию.
10. p {
11. display: inline-block;
12. }

Демонстрация inline-block :

<style>

p {

display: inline-block;

}

</style>

...

<p>Абзац один.</p>

<p>Абзац два.</p>

<p>Абзац три.</p>

1. В заключение, используя значение none мы полностью скрываем элемент и отображаем страницу, словно элемента не существует. Любые элементы, вложенные в такой элемент, также будут скрыты.
2. div {
3. display: none;
4. }

**Работа с блочной моделью**

Каждый элемент представляет собой прямоугольный блок и есть несколько свойств, которые устанавливают размер этого блока. Внутренность блока определяется шириной и высотой элемента, который может быть задан свойством display, содержимым элемента или свойствами width и height.

padding и затем border расширяют размеры блока наружу от ширины и высоты элемента. Наконец, любой указанный margin идёт за пределами рамки.

Каждая часть блочной модели соответствует свойству CSS: width, height, padding, border и margin.

Взглянем на эти свойства внутри некоторого кода:

div {

border: 6px solid #949599;

height: 100px;

margin: 20px;

padding: 20px;

width: 400px;

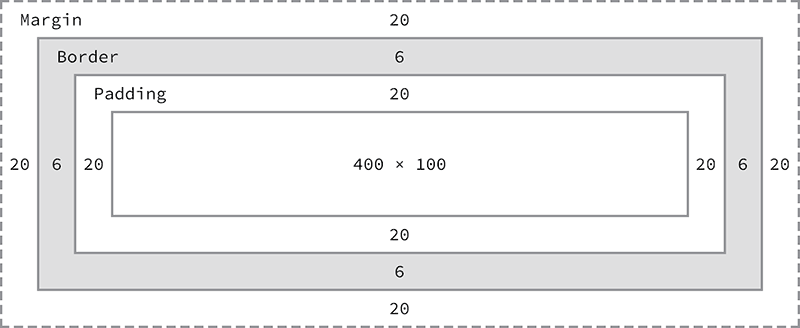
}

В соответствии с блочной моделью общая ширина элемента может быть рассчитана по следующей формуле:

*ширина = margin-right + border-right + padding-right + width + padding-left + border-left + margin-left*

Высота считается аналогично:

*высота =margin-top + border-top + padding-top + height + padding-bottom + border-bottom + margin-bottom*



Используя эти формулы, мы можем найти общую высоту и ширину блока из нашего примера.

Ширина: 492px = 20px + 6px + 20px + 400px + 20px + 6px + 20px

Высота: 192px = 20px + 6px + 20px + 100px + 20px + 6px + 20px

Блочная модель, без сомнения, одна из наиболее запутанных частей HTML и CSS. Мы устанавливаем значение свойства width как 400 пикселей, но фактическая ширина нашего элемента - 492 пикселя. Таким образом, для определения фактического размера блока мы должны принять во внимание поля, границы и отступы для всех четырёх сторон блока. Наша ширина включает не только значение свойства width, но и размер левого и правого поля, левую и правую границу, левые и правые отступы.

Давайте поближе посмотрим на все эти свойства width, height, padding, border и margin, которые формируют блочную модель.

* **width и height**
* **width**
* **height**

**Размеры строчно-блочных элементов**

**Пожалуйста, помните, что строчные элементы не принимают свойства width и height и любые их значения.** Блочные и строчно-блочные элементы, однако, принимают свойства width и height и соответствующие им значения.

* **margin и padding**
* **margin**
* **padding**
* **margin и padding для строчных элементов**

**Размеры блока**

До сих пор блочная модель была аддитивной (суммируемой). Если вы установили для width 400 пикселей, затем добавили 20 пикселей padding и border 10 пикселей на каждой стороне, фактическая полная ширина элемента становится 460 пикселей. Помните, мы должны сложить значения свойств width, padding и border вместе, чтобы получить реальную полную ширину элемента.

Блочная модель, однако, может быть изменена, чтобы вычисления происходили иначе. CSS3 ввёл свойство box-sizing, которое позволяет нам точно менять, как блочная модель работает и как вычисляются размеры элемента. Это свойство принимает три основных значения — content-box и border-box, каждое из которых оказывает несколько разное влияние на вычисление размера блока.

* **content-box**
* **border-box**

**Выбор размера блока**

Вообще говоря, лучшее значение box-sizing для применения — это border-box. Это значение намного упрощает нам математику. Если мы хотим, чтобы весь элемент был 400 пикселей в ширину, он останется 400 пикселей в ширину, независимо от того, какие значения padding или border мы к нему добавляем.

Кроме того, мы можем легко смешивать значения длины. Скажем, мы хотим, чтобы наш блок занимал 40% в ширину. Добавление padding 20 пикселей и border 10 пикселей для всех сторон элемента не сложно и мы по-прежнему можем гарантировать, что фактическая ширина нашего блока останется 40%, несмотря на использование пикселей в другом месте.

**Способы использования box-sizing**

1. ***box-sizing*** точечно. Используется стандартная блочная модель, а box-sizing \*\*включается для отдельных элементов.

.border-box-component {

box-sizing: border-box;

}

1. ***box-sizing*** по умолчанию через inherit \*\*\*\*\*- \*\*\*\*\*box-sizing \*\*включается для всех элементов с возможностью задавать исключения.

/\* это означает - все элементы наследуют свойство box-sizing от родителя \*/

\*, \* ::before, \* ::after {

box-sizing: inherit;

}

/\* а вот и родитель, где оно задано \*/

html {

box-sizing: border-box;

}

/\* а это класс-исключение, у которого одного не так \*/

.content-box-component {

box-sizing: content-box;

}

Единственный недостаток свойства box-sizing в том, что как часть спецификации CSS3 оно не поддерживается во всех браузерах, особенно не хватает поддержки в старых версиях. К счастью, это становится всё менее и менее актуальным, поскольку выходят новые версии. Скорее всего, мы можем безопасно использовать свойство box-sizing, но если вы заметили какие-то проблемы, стоит посмотреть с каким браузером они связаны.

Но как вы, наверное, уже поняли, блочная модель - отнюдь не самый простой и понятный способ позиционирования элементов, поэтому были придуманы еще варианты.

**Позиционирование через float**

Еще один из способов позиционирования элементов на странице — через свойство float. Это свойство довольно универсально и может применяться разными путями.

По существу, свойство float берёт элемент, убирает его из обычного потока страницы и позиционирует слева или справа от родительского элемента. Все остальные элементы на странице будут обтекать такой элемент. Например, абзацы будут обтекать изображение, если для элемента <img> установлено свойство float.

Когда свойство float применяется к нескольким элементам одновременно, это даёт возможность создать макет с обтекаемыми элементами расположенными рядом или напротив друг друга, как показано в многоколоночном макете.

Свойство float принимает несколько значений, два самых популярных — это left и right, они позволяют элементу располагаться слева или справа от своего родителя.

img {

float: left;

}

**Пример применения**

Допустим, мы хотим создать типовой макет страницы. Внутри элемента <body> HTML может выглядеть так:

<header>...</header>

<section>...</section>

<aside>...</aside>

<footer>...</footer>

Здесь элементы <section> и <aside> являются блочными, поэтому они укладываются один поверх другого по умолчанию. Однако мы хотим, чтобы эти элементы располагались бок о бок. Установив float для <section> как left, а для <aside> как right, мы можем позиционировать их как две колонки, расположенных напротив друг друга. Наш CSS должен выглядеть так:

section {

float: left;

}

aside {

float: right;

}

Тогда получится вот это:

Обычно обтекаемые элементы располагаются по краю родительского элемента, но если нет родителя, обтекаемый элемент будет располагаться по краю страницы.

Когда мы устанавливаем элемент обтекаемым, то убираем его из обычного потока HTML-документа. Это приводит к тому, что ширина этого элемента по умолчанию становится шириной его содержимого. Иногда, например, когда мы создаём колонки для многократно используемого макета, такое поведение нежелательно. Это можно исправить путём добавления свойства width с фиксированным значением для каждой колонки. Кроме того, чтобы обтекаемые элементы не соприкасались друг с другом, в результате чего содержимое одного элемента располагается рядом с другим, мы можем использовать свойство margin, чтобы установить пространство между элементами.

Ниже мы расширяем предыдущий блок кода, добавив margin и width для каждой колонки, чтобы лучше формировать наш желаемый результат.

section {

float: left;

margin: 0 1.5%;

width: 63%;

}

aside {

float: right;

margin: 0 1.5%;

width: 30%;

}

**Важно! float могут изменить значение display у элемента**

Для обтекаемого элемента также важно понимать, что элемент удаляется из обычного потока страницы и что у элемента может измениться значение display, заданное по умолчанию. Свойство float опирается на то, что у элемента значение display задано как block и может изменить значение display у элемента по умолчанию, если он ещё не отображается как блочный элемент.

Например, элемент, у которого display указан как inline, такой как строчный <span>, игнорирует любые свойства height или width. Однако, если для строчного элемента указать float, значение display изменится на block и тогда элемент уже может принимать свойства height или width.

Когда мы применяем float для элемента, то должны следить за тем, как это влияет на значение свойства display.

Для двух колонок вы можете установить float, для одной колонки как left, а для другой как right, но для нескольких колонок нам придётся изменить наш подход. Скажем, к примеру, мы хотели бы получить ряд из трёх колонок между нашими элементами <header> и <footer>. Если мы выбросим наш элемент <aside> и воспользуемся тремя элементами <section>, наш HTML может выглядеть следующим образом:

<header>...</header>

<section>...</section>

<section>...</section>

<section>...</section>

<footer>...</footer>

Чтобы расположить эти три элемента <section> в строку из трёх колонок, мы должны задать float для всех элементов <section> как left. Мы также должны настроить ширину <section> с учётом дополнительных колонок и расположить их один рядом с другим.

section {

float: left;

margin: 0 1.5%;

width: 30%;

}

Здесь у нас есть три колонки, все с равной шириной и значением margin, а также с float, заданным как left.

**Очистка и содержимое float**

Свойство float было первоначально разработано, чтобы позволить содержимому обтекать вокруг изображений. Для изображения может быть задано float и всё содержимое вокруг этого изображения, естественно, обтекает вокруг него. Хотя это прекрасно работает для картинок, свойство float на самом деле не было предназначено для использования в макете и с целью позиционирования и, таким образом, оно идёт с несколькими ловушками.

Одной из этих ловушек является то, что иногда надлежащие стили не отображаются на элементе, который расположен рядом с обтекаемым элементом или является его родителем. Когда элемент установлен обтекаемым, то он убирается из обычного потока страницы и, как результат, стили элементов вокруг этого обтекаемого элемента могут получить негативное влияние.

Часто значения свойств margin и padding интерпретируются некорректно, заставляя их сливаться с обтекаемым элементом. Другие свойства также могут быть затронуты.

Другая ошибка состоит в том, что иногда нежелательный контент начинает обтекать элемент с float. Удаление элемента из потока документа позволяет всем элементам вокруг обтекаемого элемента обходить его и задействовать любое доступное пространство вокруг обтекаемого элемента, что часто бывает нежелательно.

В нашем предыдущем примере с двумя колонками, после того как мы добавили float к элементам <section> и <aside>, но до того как установили свойство width к любому из них, содержимое внутри элемента <footer> располагалось между двумя обтекаемыми элементами выше него, заполняя всё доступное пространство. Следовательно, элемент <footer> находился бы в промежутке между элементами <section> и <aside>, занимая свободное место.

Согласитесь, выходит некрасиво:

Чтобы предотвратить содержимое от обматывания вокруг обтекаемых элементов, нам нужно сделать очистку float и вернуть страницу в её обычный поток. Мы рассмотрим как очистить float, а затем взглянем на их содержимое.

**Очистка float**

Очистка float происходит с помощью свойства clear, которое принимает несколько различных значений: наиболее часто используемые значения — left, right и both.

div {

clear: left;

}

Значение left очищает левые float, в то время как значение right очищает правые float. Значение both, однако, очистит левые и правые float и часто является наиболее идеальным вариантом.

Возвращаясь к нашему предыдущему примеру, если мы используем свойство clear со значением both для элемента <footer>, то можем очистить float. Важно, что clear применяется к элементу, указанному после обтекаемых элементов, а не раньше, чтобы вернуть страницу в её обычный поток.

footer {

clear: both;

}

Когда для элементов задано float важно следить за тем, как они влияют на поток страницы - и обязательно убедитесь, что поток страницы сбрасывается путём очистки или задается через правильное содержимое float. В противном случае, отслеживание float может вызвать немало головной боли, особенно на страницах, содержащих несколько строк, в каждой из которых несколько колонок.

Как вы, наверное, уже поняли, float - тоже не выход для беспроблемного позиционирования и он по сути являлся "костылем" в те времена, когда альтернатив толком не было 😅

**Уникальное позиционирование элементов**

Рано или поздно каждый пожелает точно позиционировать элемент, но float или строчно-блочные элементы не позволяют проделать такой трюк. Обтекаемые элементы, которые удаляют элемент из потока страницы, часто приводят к нежелательным результатам, поскольку окружающие элементы обтекают вокруг элемента с float. Строчно-блочные элементы, если только мы не создаём колонки, могут быть довольно неудобны, когда дело касается правильного положения. Для подобных ситуаций мы можем использовать свойство position в сочетании со свойствами смещения блока.

Свойство position определяет, как элемент позиционируется на странице и будет ли он отображаться в обычном потоке документа. Оно применяется в сочетании со свойствами смещения блока — top, right, bottom и left, которые точно определяют, где элемент будет расположен путём перемещения элемента в разных направлениях.

По умолчанию у каждого элемента значение position установлено как static, это означает, что элемент существует в обычном потоке документа и не принимает какие-либо свойства для его смещения. Значение static наиболее часто переписывается значением relative или absolute, которые мы рассмотрим дальше.

**Относительное позиционирование**

Значение relative для свойства position позволяет элементам отображаться в обычном потоке страницы, резервируя место для элемента как предполагалось и не позволяя другим элементам его обтекать. Однако, оно также позволяет модифицировать положение элемента с помощью свойств смещения. К примеру, рассмотрим следующие HTML и CSS:

**HTML**

<div>...</div>

<div class="offset">...</div>

<div>...</div>

**CSS**

div {

height: 100px;

width: 100px;

}

.offset {

left: 20px;

position: relative;

top: 20px;

}

Здесь для второго элемента <div> с классом offset задано значение position как relative, а также два свойства смещения — left и top. Это сохраняет исходное положение элемента и другим элементам не разрешено двигаться в эту область. Кроме того, свойства смещения перемещают элемент, выталкивая его на 20 пикселей от левого и на 20 пикселей от верхнего исходного местоположения.

Для относительно позиционированных элементов важно знать, что свойства смещения блока определяют, куда элемент будет перемещён, учитывая его исходное положение. Таким образом, свойство left со значением 20 пикселей фактически толкает элемент вправо на 20 пикселей. Свойство top со значением 20 пикселей затем будет толкать элемент вниз на 20 пикселей.

Когда мы позиционируем элемент с помощью свойств смещения, элемент перекрывает элемент под ним, а не сдвигает его вниз, как это делают свойства margin или padding.

**Абсолютное позиционирование**

Значение absolute для свойства position отличается от значения relative тем, что элемент с абсолютным позиционированием не появляется в обычном потоке документа, исходное пространство и положение абсолютно позиционируемого элемента не резервируется.

Абсолютное позиционирование делает две вещи:

1. **Элемент исчезает с того места, где он должен быть и позиционируется заново.** Остальные элементы, располагаются так, как будто этого элемента никогда не было.
2. **Координаты top/bottom/left/right для нового местоположения отсчитываются от ближайшего позиционированного родителя**, т.е. родителя с позиционированием, отличным от варианта по умолчанию. Если такого родителя нет – то относительно документа.

**HTML**

<section>

<div class="offset">...</div>

</section>

**CSS**

section {

position: relative;

}

div {

position: absolute;

right: 20px;

top: 20px;

}

В этом примере элемент <section> позиционируется относительно, но не включает каких-либо свойств смещения. Следовательно, его положение не меняется. Элемент <div> с классом offset включает значение position как absolute. Поскольку элемент <section> является ближайшим относительно позиционированным родительским элементом для <div>, то элемент <div> будет позиционироваться относительно элемента <section>.

Для относительно позиционированных элементов свойства смещения определяют, в каком направлении элемент будет перемещён относительного самого себя. Для абсолютно позиционируемых элементов свойства смещения определяют, в каком направлении элемент будет перемещён относительно его ближайшего относительно позиционированного родителя.

В результате свойств right и top, элемент <div> появится в 20 пикселях справа и 20 пикселях сверху внутри <section>.

Поскольку элемент <div> позиционируются абсолютно, он не располагается в обычном потоке страницы и будет перекрывать любые окружающие его элементы. Кроме того, исходное положение <div> не сохраняется и другие элементы могут занять это место.

**Фиксированное позиционирование**

Позиционирует объект точно так же, как absolute, но относительно window.

position: fixed;

Разница в нескольких словах:

**Когда страницу прокручивают, фиксированный элемент остаётся на своём месте и не прокручивается вместе со страницей.**

В следующем примере, при прокрутке документа, ссылка #top всегда остаётся на своём месте.

<style>

\*#top {

position: fixed;

right: 10px;

top: 10px;

background: #fee;

}\*

</style>

\*<a href="#" id="top">Наверх (остаётся при прокрутке)</a>\*

Фиксированное позиционирование.

<p>Текст страницы.. Прокрути меня...</p>

<p>Много строк..</p><p>Много строк..</p>

<p>Много строк..</p><p>Много строк..</p>

<p>Много строк..</p><p>Много строк..</p>

<p>Много строк..</p><p>Много строк..</p>

Как правило, большая часть позиционирования может происходить без применения свойств position и свойств смещения, но в некоторых случаях они могут оказаться чрезвычайно полезны. Но абсолютное позиционирование не подходит для 99% обычных ситуаций, когда у наших пользователей совершенно разные размеры экранов и устройств.

Что же тогда делать?! Мы рассмотрели уже 4 способа позиционирования... и все они не очень!

Решение есть! И это...

**Flexbox**

Flexbox предназначен для **создания гибких макетов**. С помощью этой технологии можно очень просто и гибко расставить элементы в контейнере, распределить доступное пространство между ними, и выровнять их тем или иным способом даже если они не имеют конкретных размеров.

Flexbox позволяет создать адаптивный дизайн намного проще, чем с использованием других видов позиционирования.

**Основы CSS Flexbox**

Создание CSS разметки с помощью Flexbox начинается с установки необходимому HTML элементу CSS-свойства display со значением flex или flex-inline.

После этого данный элемент становится **flex-контейнером**, а все его **непосредственные** дочерние элементы – **flex-элементами**. При этом когда мы говорим о flexbox то подразумеваем под этим только элемент с display:flex или display:flex-inline и все элементы **непосредственно** расположенные в нём. Таким образом в Flexbox имеется всего два типа элементов: flex-контейнер и flex-элемент.

<style>

.flex-container {

display: flex;

}

<style>

<!-- flex-контейнер -->

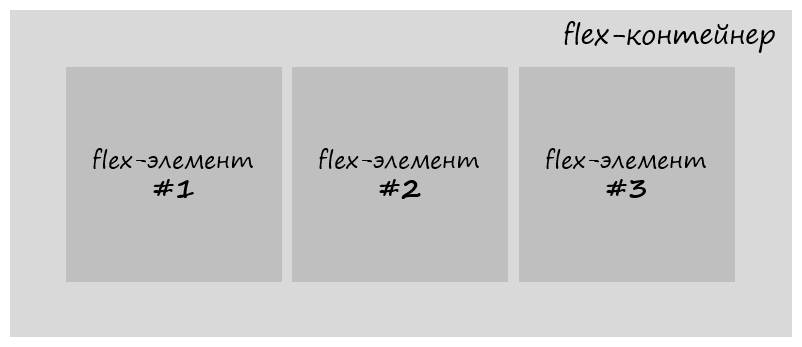
<div class="flex-container">

<div>flex-элемент #1</div>

<div>flex-элемент #2</div>

<div>flex-элемент #3</div>

</div>

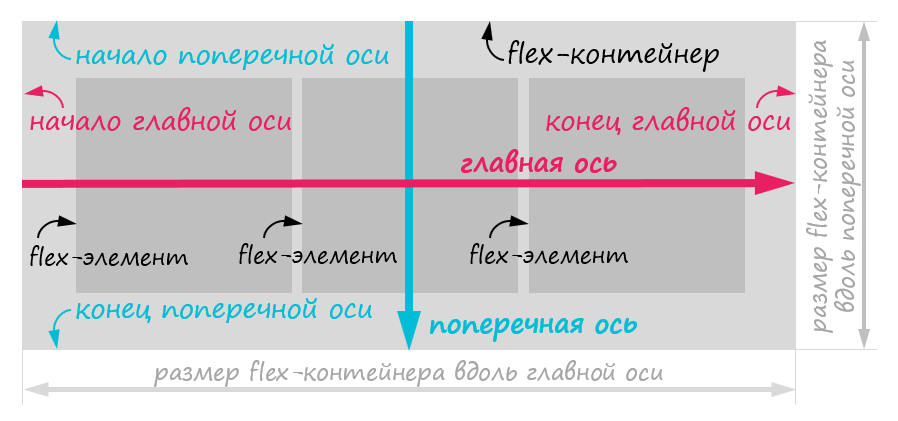


По умолчанию flex-элементы во flex-контейнере занимают всю его высоту.

Значение flex или inline-flex определяет то, как **flex-контейнер** будет представлен на странице. Если его необходимо отобразить в виде **блока**, то используйте значение flex. Если элемент необходимо представить как **строку**, то используйте значение inline-flex. В этом случае он будет занимать столько места странице, сколько необходимо для отображения его элементов.

**Устройство flex-контейнера. Направление осей**

На рисунке представлена схема устройства flex-контейнера:



Направление расположение flex-элементы в flex-контейнере определяется посредством осей.

В Flexbox имеются две оси. Первая ось называется **главной** (по умолчанию она направлена слева направо). Вторая - **поперечная** (по умолчанию направлена сверху вниз), **она всегда перпендикулярно главной**. Главная ось задаёт основное направление flex-элементов во flex-контейнере, а поперечная ось определяет их направление при переносе на новую линию.

По умолчанию элементы во flex-контейнере располагаются вдоль направления главной оси (т.е. слева направо) на одной линии.

Направление главной оси можно изменить, осуществляется это с помощью CSS-свойства flex-direction.

flex-direction: row;

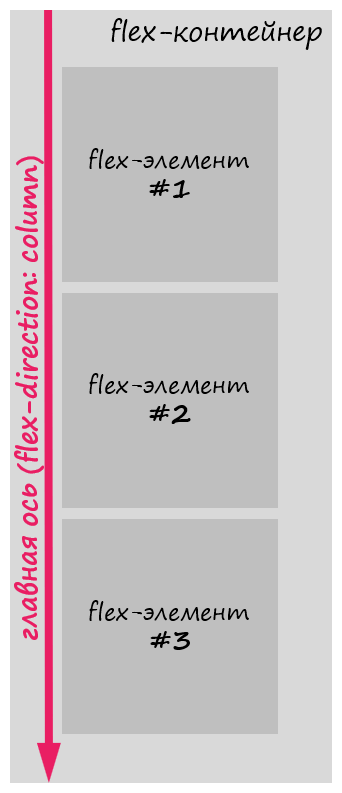
/\* row (слева направо) - по умолчанию

row-reverse (справа налево)

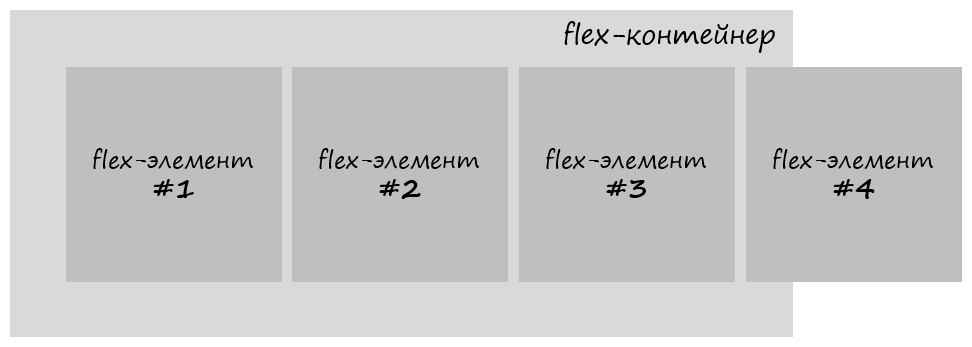
column (сверху вниз)

column-reverse (снизу вверх) \*/

С помощью этого свойства можно сделать так, чтобы flex-элементы располагались не рядами (rows), а колонками (columns). Осуществляется это с помощью значения column или column-reverse.



По умолчанию flex-элементы не переносятся на новую линию, даже когда им не хватает места в текущей линии. Они просто выходят за её пределы.



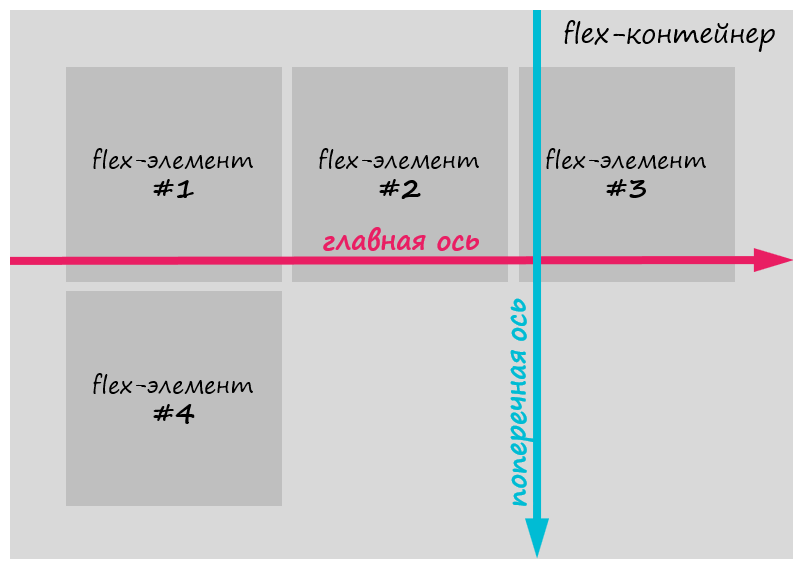
Но это можно изменить. Разрешить перенос flex-элементов на новые линии осуществляется с помощью установки flex-контейнеру CSS свойства flex-wrap со значением wrap или wrap-reverse.

flex-wrap: wrap;

/\* nowrap (только на одной линии - по умолчанию)

wrap (разрешить перенос flex-элементов на новые линии)

wrap-reverse (осуществлять перенос flex-элементов в обратном порядке) \*/



Значения wrap и wrap-reverse CSS-свойства flex-wrap определяют **направление поперечной оси**.

Свойства flex-direction и flex-wrap можно указать с помощью универсального CSS свойства flex-flow:

flex-flow: row nowrap; /\* 1 значение - flex-direction, 2 значение - flex-wrap \*/

**Выравнивание flex-элементов**

Во Flexbox выравнивание элементов внутри контейнера осуществляется по двум направлениям (осям).

**Выравнивание flex-элементов по направлению главной оси**

Выравнивание элементов вдоль основной оси осуществляется с помощью CSS свойства justify-content:

justify-content: flex-start;

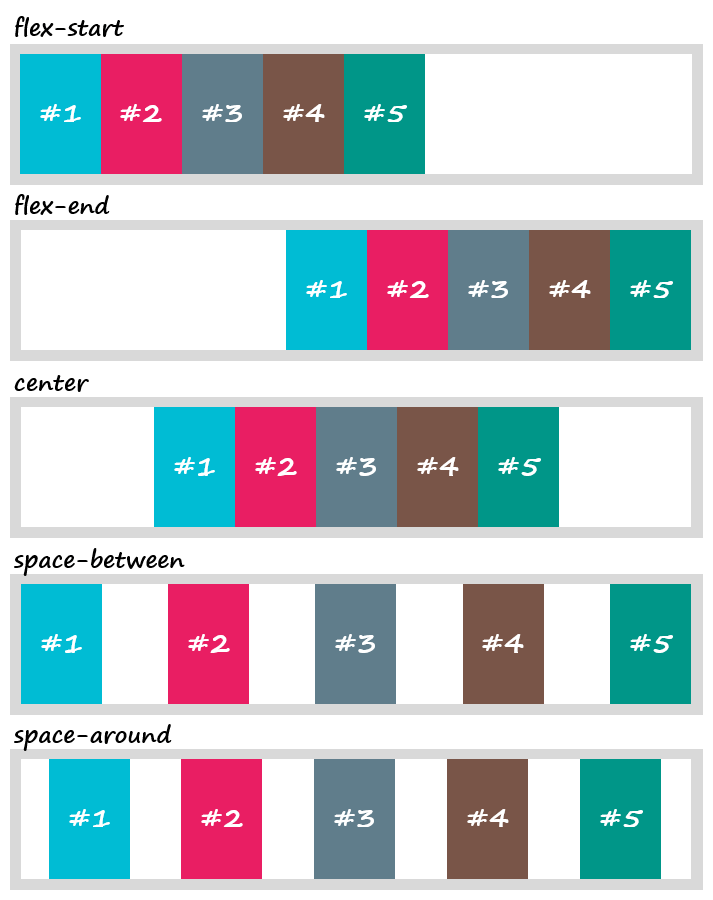
/\* flex-start (flex-элементы выравниваются относительно начала оси) – по умолчанию

flex-end (flex-элементы выравниваются относительно конца оси)

center (по центру flex-контейнера)

space-between (равномерно, т.е. с одинаковым расстоянием между flex-элементами)

space-around (равномерно, но с добавлением половины пространства перед первым flex-элементом и после последнего) \*/



**Выравнивание flex-элементов вдоль поперечной оси**

Выравнивание flex-элементов во flex-контейнере по направлению поперечной оси осуществляется с помощью CSS-свойства align-items:

align-items: stretch;

/\* stretch (растягиваются по всей длине линии вдоль направления поперечной оси) – по умолчанию

flex-start (распологаются относительно начала поперечной оси)

flex-end (относительно конца поперечной оси)

baseline (относительно базовой линии)

center (по центру) \*/

align-items: stretch;

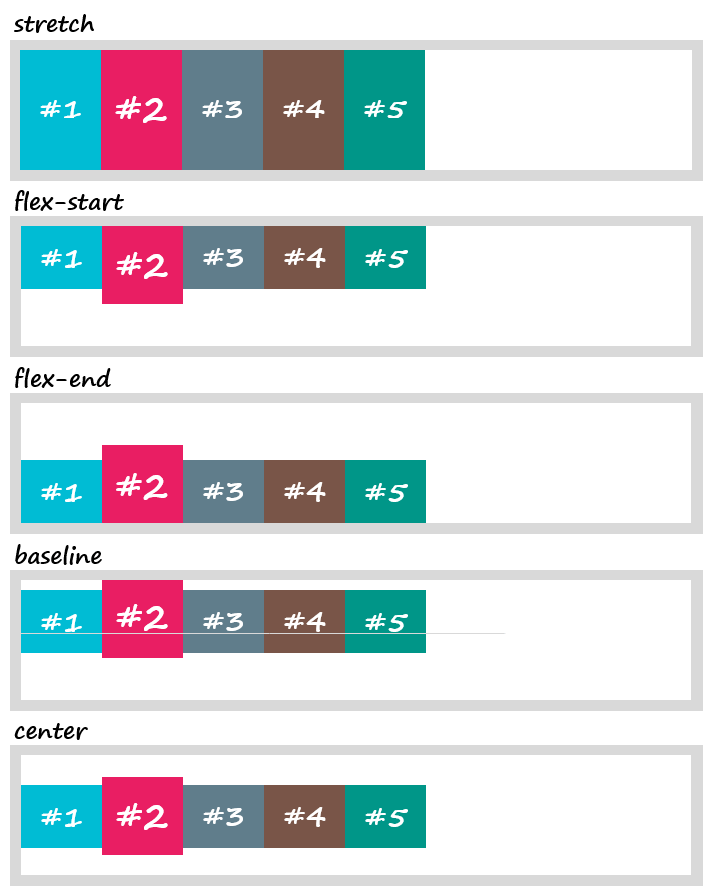
/\* stretch (растягиваются по всей длине линии вдоль направления поперечной оси) – по умолчанию

flex-start (распологаются относительно начала поперечной оси)

flex-end (относительно конца поперечной оси)

baseline (относительно базовой линии)

center (по центру) \*/



**Выравнивание линий flex-контейнера**

CSS Flexbox позволяет выравнивать не только сами flex-элементы, но и линии на которых они расположены.

align-content: stretch

/\* stretch (растягиваются по всей длине поперечной оси) – по умолчанию

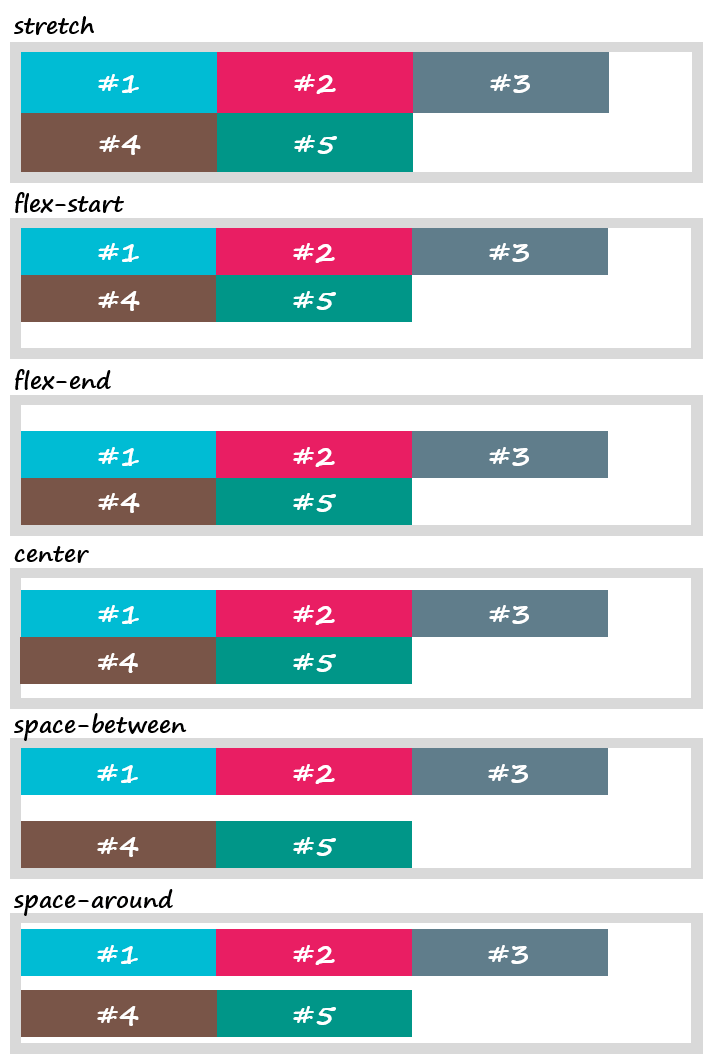
flex-start (относительно начала поперечной оси)

flex-end (относительно конца поперечной оси)

center (по центру)

space-between (равномерно, т.е. с одинаковым расстоянием между линиями)

space-around (равномерно, но с добавлением половины пространства перед первой линией и после последней) \*/



Свойство align-content имеет смысл использовать только тогда, когда flex-элементы во flex-контейнере располагаются на нескольких линиях. Чтобы это произошло, необходимо, во-первых, чтобы ширина всех flex-элементов была больше ширины flex-контейнера, а во-вторых flex-контейнер должен иметь в качестве CSS-свойства flex-wrap значение wrap или wrap-reverse.

**CSS-свойство align-self**

Свойство align-self в отличие от предыдущих (justify-content, align-items и align-content) предназначено для flex-элементов. Оно позволяет изменить **выравнивание flex-элемента** вдоль направления поперечной оси. Свойство align-self может принимать такие же значения как align-items.

align-self: stretch; /\* на весь размер контейнера \*/

/\*

Еще варианты:

auto (по умолчанию) - наследует родительское значение

flex-start - c начала

flex-end - с конца

baseline - по условной средней линии

center - по центру

\*/

**Изменение порядка следования flex-элементов**

По умолчанию flex-элементы отображаются во flex-контейнере в том порядке, в котором они расположены в HTML коде. Для изменения порядка следования одних flex-элементов относительно других в CSS Flexbox можно использовать свойство order. Данное CSS свойство выстраивает flex-элементы во flex-контейнере в порядке возрастания их номеров.

order: 0;

/\*

0 (по умолчанию)

либо любое целое положительное или отрицательное число

\*/

**Управление шириной flex-элемента**

Во Flexbox есть несколько CSS свойств, определяющих то, какая ширина может быть у flex-элемента.

**CSS-свойство flex-basis**

Данное свойство предназначено для **установления начальной ширины flex-элементу**. Задавать значение ширины можно посредством различных единиц измерения, таких как px, %, em и др. По умолчанию данное свойство имеет значение auto (в этом случае ширина элемента будет рассчитываться автоматически на основании его содержимого).

**Конечная ширина flex-элемента будет определяться** в зависимости от значений CSS-свойств flex-grow и flex-shrink, которые установлены не только для этого элемента, но и для других flex-элементов этого flex-контейнера.

**CSS-свойство flex-grow**

Это свойство определяет, может ли **начальная ширина flex-элемента увеличиваться (расти)**. Увеличение ширины flex-элемента осуществляется за счёт **свободного пространства линии**. В качестве значения CSS-свойства flex-grow указывается **целое число**. Именно это значение и определяет (если оно больше или равно 1) какую часть свободного пространства flex-элемент заберёт себе.

**CSS-свойство flex-shrink**

Данное свойство определяет, может ли ширина flex-элемента уменьшиться. Уменьшение ширины flex-элемента будет осуществляться только в том случае, если **ширины линии будет не достаточно для отображения всех flex-элементов**, расположенных в ней. Необходимая ширина рассчитывается на **основании начальной ширины**, которую имеет каждый flex-элемент в ней.

**CSS-свойство flex**

Для удобной установки flex-grow, flex-shrink и flex-basis можно использовать CSS свойство flex.

Значение по умолчанию:

flex: 0 1 auto;

/\*

0 - flex-grow (1 значение)

1 - flex-shrink (2 значение)

auto - flex-basis (3 значение)

\*/

**Особенности flexbox**

* Размеры флекс-элементов рассчитываются как в обычной блочной модели: *рамки + отступы + размеры содержания*
* Работает box-sizing
* Флекс-элементы по умолчанию ужимаются под содержимое
* Не работает *float*
* Внешние отступы не схлопываются и не выпадают
* Всегда явно задавайте размер колонок: width \*\*или flex-basis
* Расстояния между колонок можно задавать:
  + с помощью jusify-content, если отступы одинаковые
  + с помощью margin, если отступы разные
* flex-grow не подходит, если нужно добиться точного соответствия макету
* Не стоит использовать flexbox для верстки больших блоков текста

**Выводы**

Как вы, наверное, уже поняли - именно flexbox-позиционирование является самым простым и удобным вариантом в современном мире и именно оно является официально рекомендованным вариантом от World Wide Web Consortium (W3C) (это те же самые ребята, которые придумали правила для наших валидаторов и для всемирной паутины вообще).

Есть ли у него недостатки? Да, он не поддерживается старыми версиями браузеров, но такие сейчас занимают не более 2-3% рынка, поэтому это не страшно.

P.S. Тут можно поэкспериментировать с флексами: <https://the-echoplex.net/flexyboxes/>

Шпаргалка по флексбоксам <http://fls.guru/flexbox.html>

**Домашнее задание на эту неделю:**

**Тест 💎**

1. Почему в большинстве ситуаций стоит использовать flexbox-позиционирование?
2. Самостоятельно изучите способ позиционирования через display: table и ответьте на вопрос, для каких ситуаций оно лучше всего подходит?
3. Какими способами можно сделать горизонтальное выравнивание по центру? Минимум 3 варианта, можно больше
4. Какие есть оси во флекс-верстке и как задается их направление?
5. Разберитесь, как работает свойство margin: auto во флекс-верстке, приведите пример использования
6. В чем преимущества box-sizing?
7. Чем отличается flex-grow от flex-shrink?
8. Как можно добиться следующего позиционирования элементов:
9. Какой в итоге будет размер у элемента (можно округлить)?
10. Самостоятельно разберитесь, зачем нужно свойство order?
11. Каким кодом можно сделать такую таблицу?
12. Изучите материалы и найдите, для каких ситуаций подходит position: fixed?
13. Каким способом лучше всего верстать большие блоки текста?
14. Как расчитывается размер flex-контейнера?

**Практическое задание 💎**

1. Пройти игру <https://flexboxfroggy.com/> (подтверждение - скриншот 23 или 24 уровня)
2. Сверстать страничку любым способом, чтобы получить такой результат и прислать ссылку на github:
3. Полностью доделать страницу из прошлой недели с помощью новых знаний о верстке и Flexbox (если хочется, то можно поменять макет) и прислать ссылку на github

Если вы хотите оставить свой репозиторий скрытым, то давайте доступ для аккаунта **itgirlschool**

**Ссылка на свою страничку из википедии 💎**

ОТВЕТЫ:

1. FLEXBOX- позиционирование позволяет создавать гибкие макеты, сделать адаптивный дизайн становится намного проще, чем с использованием других методов позиционирования.
2. CSS-свойство display: table и другие, делают вывод группы элементов подобно таблице <table>, но с ограничением – объединения ячеек colspan и rowspan не поддерживаются. Желательно использовать только по назначению, а не для верстки.
3. Align-items: center;

Align-self:center;

Justifu-content:center;

1. Существует 2 оси выравнивания: главная и поперечная.

Для задания направления используют flex-direction: row, column, row-reverse, column-reverse.

1. При использовании margin:auto браузер выбирает подходящее значение для использования. Например, в некоторых случаях это значение может быть использовано для центрирования элемента.

margin: auto; /\* верх и низ: отступ 0 \*/

/\* блок отцентрирован горизонтально \*/

1. box-sizing позволяет нам точно понимать, как блочная модель работает и как вычисляются размеры элемента. лучшее значение box-sizing для применения — это border-box. Это значение намного упрощает нам математику. Если мы хотим, чтобы весь элемент был 400 пикселей в ширину, он останется 400 пикселей в ширину, независимо от того, какие значения padding или border мы к нему добавляем. Кроме того, мы можем легко смешивать значения длины.
2. Flex-grow определяет, может ли **начальная ширина flex-элемента увеличиваться (расти)**.

Flex-shrink определяет, может ли ширина flex-элемента уменьшиться.

1. Justify-content: space-around
2. 148,523\*17,807
3. Order используют для изменения порядка следования элементов относительно других.
4. Ответ в моем репозитории Github
5. Если мы используем position:fixed, когда страницу прокручивают, фиксированный элемент остаётся на своём месте и не прокручивается вместе со страницей.
6. Блочная модель, т.к. флекс нежелателен, а табличная верстка устарела
7. Размеры флекс-элементов рассчитываются как в обычной блочной модели: рамки + отступы + размеры содержания