# Блочная модель документа

### Блочные боксы

Каждому HTML-элементу на странице соответствует прямоугольная область. Такие области называются боксами. Самые распространённые типы боксов — блочные и строчные. Блочный бокс — это крупная неразрывная прямоугольная область на странице. По умолчанию блочным боксом обладают крупные поточные элементы: заголовки, абзацы.

Блочные боксы имеют следующие особенности:

* До и после блочного бокса существует перенос строки.
* Для блочных боксов можно задавать ширину, высоту, внутренние и внешние отступы.
* Занимают всё доступное пространство по горизонтали.

Ещё один важный тег с блочным боксом — это тег <div>, который обозначает просто «блок» или «прямоугольный контейнер». Этот тег чаще всего используется для создания сеток.

### Строчные боксы

Строчные боксы — противоположность блочных. Строчные боксы располагаются друг за другом в одной строке, могут разрываться и находиться на нескольких строках. Особенности строчных боксов:

* До и после строчного бокса отсутствуют переносы строки.
* Ширина и высота строчного бокса зависит только от его содержания, задать размеры с помощью CSS **нельзя**.
* Можно задавать только горизонтальные отступы.

Эти теги предназначены для оформления текста на уровне небольших фраз и отдельных слов. А теги с блочным боксом, напротив, предназначены для разметки крупных блоков текста (заголовки, абзацы, списки) и создания сетки.

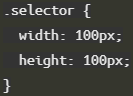
### Ширина и высота

Ширина и высота боксов задаются с помощью свойств width и height соответственно.

* По умолчанию блочные боксы занимают всю доступную **ширину**, которая равна ширине родительского контейнера или окна браузера.
* **Высота** по умолчанию блочных боксов зависит от их содержимого. Если задать блочному боксу ширину и высоту так, что содержимое не будет в него помещаться, то оно как бы «выпадет» из него.

*Строчные боксы не реагируют на задание ширины и высоты в CSS.*

Задание ширины:

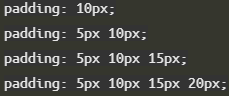
 Вернуть эти значения по умолчанию можно с помощью специального значения **auto.**

### Внутренние отступы, свойство padding

padding задаёт внутренние отступы блока — отступы от внешней границы блока до его содержимого. Задается и строчным и блочным.



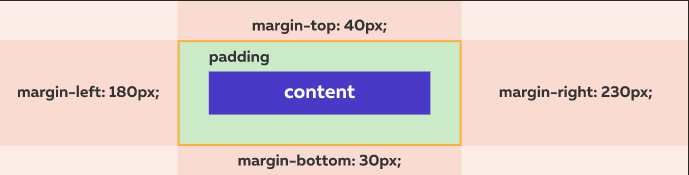
Сокращённое свойство padding:



* Все стороны
* Верх и низ, право и лево
* Верх, лево и право, низ
* Верхний, правый, нижний, левый.

### Внешние отступы, свойство margin

Свойство margin задаёт внешние отступы блока — отступы от внешней границы элемента до границ родительского элемента или до соседних элементов. Задается и строчным и блочным.



Сокращённое свойство margin работает аналогично свойству padding

### Рамки

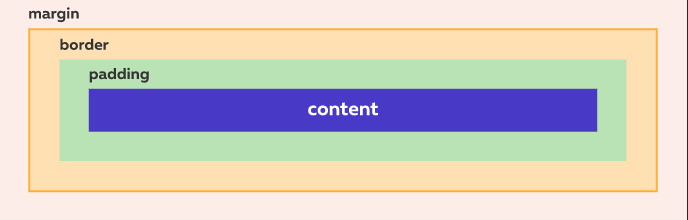
состоит из трёх компонентов:

* ширина рамки;
* стиль рамки;
* цвет.

Задавать рамку можно одним свойством border, а можно и с помощью отдельных свойств border-width, border-style, border-color. Рамку можно задавать элементам и со строчным, и с блочным боксом.

### Стандартная блочная модель

Область, занимаемая блочным боксом, складывается из его ширины и высоты содержания, внутренних и внешних отступов, ширины рамок.



## Тонкости блочной модели

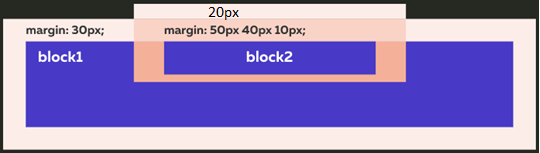
### «Схлопывание» внешних отступов

**Вертикальный** отступ между двумя соседними элементами равен максимальному отступу между ними. Если отступ одного элемента равен 50px, а второго 30px, то отступ между ними будет 50px.

**Горизонтальные** отступы между элементами просто складываются. Например, горизонтальный отступ между двумя элементами с отступами 30px будет равен 60px.

### «Выпадание» внешних отступов

Если внутри родительского блока расположить блок и задать ему отступ сверху, то внутренний блок прижмётся к верхнему краю родительского, а у родительского элемента появится отступ сверху. То есть верхний отступ внутреннего элемента «выпадает» из родительского элемента:



Если у родительского элемента тоже был задан внешний отступ, то выберется максимальный отступ между собственным и «выпавшим» (видно на картинке сверху).

Чтобы избавиться от эффекта выпадания, можно задать родительскому элементу внутренний отступ (паддинг) сверху или добавить рамку сверху:



*Внешние и внутренние отступы всегда складываются.*

### Как отцентровать элемент?

Чтобы отцентровать **блочный** элемент, нужно выполнить следующие действия:

* Задать элементу ширину, которая меньше ширины родительского контейнера.
* Задать для внешних отступов справа и слева значение auto.

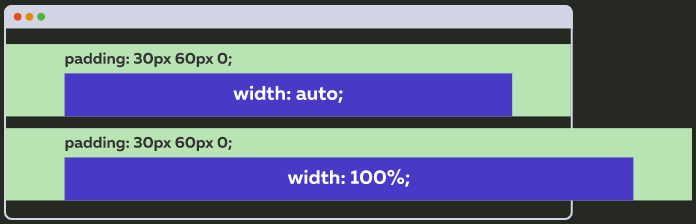
### Блочная модель и строчные боксы

Особенности поведения элементов со строчным боксом в блочной модели:

1. Не реагируют на CSS-свойства width и height.
2. Частично реагируют на margin, воспринимая только горизонтальные отступы.
3. Частично реагируют на padding, воспринимая только горизонтальные отступы.
4. При задании вертикальных padding визуально увеличиваются, но без увеличения занимаемого места (не отталкивают другие элементы).
5. Воспринимают рамки. Аналогично padding рамки сверху и снизу не увеличивают занимаемое элементом место.

### Ширина 100% и ширина по умолчанию

CSS-свойство width задаёт не общую ширину блока, **а только ширину содержания**. Общая ширина блока затем складывается из трёх компонентов: ширины содержания, внутренних отступов и ширины рамок слева и справа.



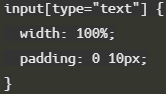
**Первый вариант**. Вариант по умолчанию, когда ширина не задаётся, соответствует значению width: auto;. В этом случае блок занимает всю ширину родительского блока. Если у блока есть внутренние отступы или рамки, то его ширина содержания автоматически уменьшается, а общая ширина остаётся равной ширине родителя.

**Второй вариант**. Когда ширина блока задана явно, например, width: 100%;. В этом случае ширина содержания блока равна ширине родительского блока. Если блоку добавить внутренние отступы и рамки, то его общая ширина становится больше ширины родителя.

*Помимо CSS* ***ширина полей ввода*** *может быть задана в значении атрибута size. Ширина width: auto для полей ввода рассчитывается из значения size по умолчанию и не растягивает поля на всю ширину контейнера.*

### Особенности применения width: 100%

Представьте, что у нас есть форма, в которой поля должны занимать всю ширину контейнера, но при этом иметь фиксированные внутренние отступы, чтобы текст не прилипал к краям. Форма может быть разной ширины. Для полей такой формы можно задать такие стили:

 Ширина 100% заставит поля быть такой же ширины, как родительский блок, а паддинг добавит отступы. Но, как вы уже догадались, за счёт отступов поле станет шире своего контейнера. И эта особенность создаёт проблему.

### Изменяем блочную модель, свойство box-sizing

Что бы решить проблему выше, нужно чтобы свойство width задавало не ширину содержания, а общую ширину. Такая возможность в CSS есть. Реализуется с помощью свойства box-sizing.

Это свойство имеет два значения:

* content-box — значение по умолчанию, соответствует стандартной блочной модели.
* border-box — изменяет режим расчёта ширины элемента: теперь ширина элемента включает и рамку, и внутренние отступы и, собственно, ширину содержания самого элемента.

При увеличении, например, паддингов, блок теперь не будет расширяться, **НО** только пока эти паддинги не начинают «щемить» основное содержание. Указывается элементу

### Управление типом бокса, свойство display

С его помощью, например, можно сделать боксы абзацев и заголовков строчными, а боксы спанов и стронгов — блочными элементами.

Свойство display: block обозначает блочный бокс элемента, а display: inline — строчный.

### Свойство display: inline-block

В HTML нет элементов с блочно-строчным боксом по умолчанию, но любой элемент можно переключить такой режим отображения, задав ему свойство display со значением inline-block.

Особенности элементов с блочно-строчным боксом:

* им можно задавать размеры, рамки и отступы, как и элементам с блочным боксом;
* их ширина по умолчанию зависит от содержания, а не растягивается на всю ширину контейнера;
* они не порождают принудительных переносов строк, поэтому могут располагаться на одной строке, пока помещаются в родительский контейнер;
* элементы в одной строке выравниваются **вертикально** подобно элементам со строчным боксом (как текст).

### Свойство display: table

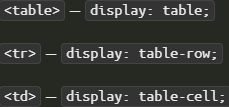
Особенности элементов с табличным боксом:

1. можно задавать ширину, высоту, рамки, отступы;
2. по умолчанию ширина зависит от содержания;
3. переносы строки до и после элемента.

Табличный тип бокса похож на блочный за исключением ширины по умолчанию.

### Свойство display: table-row

Соответствие тегов таблицы значениям display:



Строка таблицы — необычный элемент. Она является контейнером для ячеек и практически не имеет собственного отображения. Для неё можно только задавать цвет фона.

### Свойство display: table-cell

С помощью значения table-cell свойства display можно задать боксу элемента тип ячейка таблицы. Конечно, лучше чтобы элементы-ячейки находились внутри элементов-строк, которые находятся внутри элементов-таблиц.

Вы можете просто задать боксу элемента тип ячейка таблицы, не добавляя вокруг него дополнительных элементов-строк и таблиц. В этом случае браузер создаст дополнительные анонимные элементы строки и таблицы. Неудобство заключается в том, что вы не сможете ими управлять.

К таблицам, созданным с помощью CSS, можно применять те же свойства, что и к обычным таблицам. Например, задавать отступы между ячейками или режим схлопывания границ.

### Свойство display: none

Значение none свойства display используется очень часто. С его помощью можно скрыть элемент, как будто его и не было. Скрытый элемент не отображается и не занимает места на странице.

Это свойство применяется при создании выпадающих меню, динамических галерей, переключающихся вкладок и много где еще.

Есть ещё одно CSS-свойство, которое используется для сокрытия элементов. Это свойство visibility со значением hidden. Оно «прячет» элемент — он становится невидимым, но занимает место на странице.

# Сетки на дивах (на флотах и инлайн-блоках)

### Пробуем управлять потоком

Поток — это порядок отображения элементов на странице. По умолчанию блочные элементы отображаются как прямоугольные области, идущие друг за другом сверху вниз, а строчные элементы располагаются сверху вниз и слева направо и при необходимости переносятся на новую строку.

Потоком можно управлять и изменять привычное поведение элементов в потоке. Например, можно заставить блочные элементы двигаться не сверху вниз, а выстраиваться в несколько колонок.

А научившись управлять потоком, вы сможете строить сетки

### Погружение во флоаты

Изначально float было предназначено для того, чтобы включать обтекание элементов текстом. Можно сказать, что float — это свойство, включающее режим обтекания.

Свойство float имеет следующие значения:

* left — прижимает элемент к левому краю родителя, другие элементы обтекают его справа;
* right — прижимает элемент к правому краю родителя, другие элементы обтекают его слева;
* none — отключает режим обтекания и возвращает элементу нормальное поведение.

Зафлоатить элемент по центру нельзя.

### float и ширина

По умолчанию блочные элементы растягиваются на всю доступную ширину родителя.

**Если мы задаём элементу свойство float:left или float:right, то он прижимается к левому или правому краю, а также начинает ужиматься по ширине под своё содержимое.** **С той стороны, которая не прижата к краю родителя, появляется свободное место. Это место может быть занято другими элементами.**

Зафлоаченному элементу можно явно задавать размеры и отступы.

Есть тонкость, связанная со строчными элементами. Если зафлоатить строчный элемент, то он начинает вести себя как блочный, а именно: воспринимать размеры и отступы.

### float и выпадание из потока

Зафлоаченные элементы выпадают из потока, но лишь частично:

* Блочные элементы, которые идут в коде **после** зафлоаченного блока, перестают его замечать. Они подтягиваются вверх и занимают его место, как будто его и нет.
* Строчные же элементы, расположенные в коде после зафлоаченного блока, начинают обтекать его со свободной стороны.

Ещё раз: для блочных элементов флоатные не существуют, но текст внутри блоков флоатные обтекает.

Такое поведение флоатов даёт интересные эффекты:

1. Эффект прохождения сквозь блоки. Проявляется, когда зафлоаченный элемент выше чем несколько последующих за ним незафлоченных блоков.
2. Эффект выпадания из родителя или схлопывания родителя. Проявляется тогда, когда все дочерние блоки в родителе зафлоачены. В этом случае родитель схлопывается по высоте, как будто в нём нет содержимого, а блоки выпадают из него.

### Флоат рядом с флоатом

Флоатные элементы становятся невидимыми для блочных элементов и видимыми для текста. А как же они взаимодействуют друг с другом?

Если вкратце то: флоатные элементы видят друг друга.

Идущие друг за другом флоаты выстраиваются в ряд, пока им хватает свободного места. Если места не хватает, то они начинают переноситься на следующую строчку. Почти как текст.

### Когда флоатов много, а места мало

Следует отметить, что поведение нескольких флоатов, когда им не хватает места в одной строке, является очень странным.

Когда не влезающий флоат переносится на новую строку, возможно несколько вариантов и не все из них логичны. Например, флоат может «зацепиться» за один из предшествующих флоатов и встать ниже не в самом начале строки, а за предшествующим.

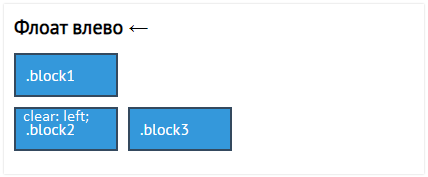
**Для того, чтобы флоатный блок мог обтекаться обычным, он должен в коде располагаться выше обычного.**

### Свойство clear

Свойство clear запрещает обтекание элемента другими элементами. Вот его значения:

1. left — запрещено обтекание слева;
2. right — запрещено обтекание справа;
3. both — запрещено обтекание с обеих сторон;
4. none — обтекание разрешено.

**Если после флоатного элемента расположен элемент с запрещённым обтеканием, то последний опускается под флоатный.**

****

**Свойство так же clear учит блочные элементы «видеть» зафлоаченные.**

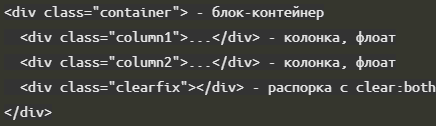
### Борьба с выпаданием флоатов: распорки

Эффект выпадания флоатов из родителя был большой проблемой при построении сеток. Ведь сетки на флоатах обычно делаются вот так:

1. Создаём блок-контейнер для колонок.
2. В контейнер добавляем флоатные блоки-колонки.
3. Рассчитываем ширины колонок так, чтобы им хватало места в родителе.

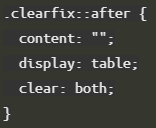
Такой подход работает неплохо. Но проблемы начинаются тогда, когда у контейнера есть фон, отличный от фона страницы. Когда колонки выпадали, родитель схлопывался и фон пропадал.

Необходимо было найти способ борьбы с выпаданием. Тут и пригодилось свойство clear:both. Стали применять такую структуру:



То есть начали добавлять после зафлоаченных колонок пустой элемент-распорку со свойством clear:both (незафлоченный). Этот элемент видел колонки, не давал им пройти через себя, а заодно и растягивал родительский блок по высоте.

### Борьба с выпаданием флоатов: псевдораспорки



класс clearfix добавляется к контейнеру, внутри которого лежат флоатные колонки.

### Погружение в inline-block

Блочно-строчные элементы ведут себя двояко. Снаружи они выглядят как обычные строчные, но внутри они ведут себя как блочные.

От строчных им достались следующие черты:

* по ширине они ужимаются под своё содержимое;
* могут располагаться в одну строку;
* реагируют на вертикальное выравнивание, vertical-align;
* реагируют на горизонтальное выравнивание, text-align, заданное у родителя.

От блочных:

* им можно задавать размеры с помощью width и height;
* а также внешние и внутренние отступы и рамки, которые работают во всех направлениях и увеличивают размер элемента.

При переносе на следующую строку блочно-строчные ведут себя намного логичней. Высота строк у них рассчитывается по максимальному элементу с учётом отступов. Выстраиваются в ровную строку друг под другом, если не хватает места.

### inline-block и пробелы в коде

Блочно-строчные ведут себя как текст, поэтому если в коде есть пробел между элементами, то он отображается и на странице. Этот пробел увеличивает отступы между товарами, не давая им поместиться в одну строку.

Бороться с пробелом после блочно-строчных можно несколькими способами:

* удалять пробелы в коде;
* обнулять размер шрифта;
* играться с маргинами после блочно-строчного.

# Флексбокс, часть 1

### Свойство display: flex, flex-элемент

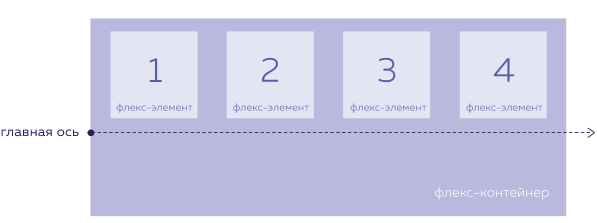
Как включить флексбокс? Очень просто: нужно задать элементу свойство display: flex;. После этого происходит два события:

1. Элемент с display: flex; превращается во «флекс-контейнер» и внутри него начинает происходить вся магия гибкой раскладки.
2. Непосредственные потомки этого элемента превращаются во «флекс-элементы» и начинают играть по новым правилам.

Блоки растянутся на всю высоту контейнера.

### Свойство flex-direction, главная ось

Вместо направлений «лево» и «право» во флексбоксе используется понятие «главная ось». Поток флекс-элементов «течёт» вдоль главной оси от её начала к её концу.



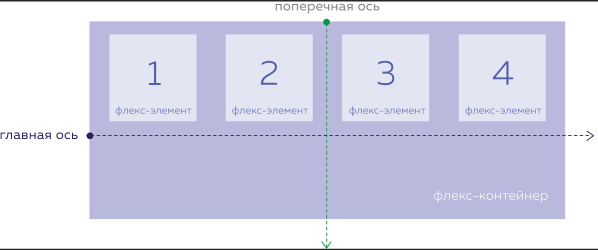
По умолчанию главная ось направлена слева направо, но её можно разворачивать во всех направлениях с помощью свойства flex-direction, которое **задаётся для флекс-контейнера**. Значения свойства:

* row — значение по умолчанию, главная ось направлена слева направо.
* column — главная ось направлена сверху вниз.
* row-reverse — главная ось направлена справа налево.
* column-reverse — главная ось направлена снизу вверх.

Флекс-элементы всегда располагаются вдоль главной оси

### Поперечная ось flexbox

Вместо направлений «верх» и «низ» во флексбоксе используется понятие «поперечная ось». Вдоль этой оси работают «вертикальные» выравнивания.



Поперечная ось всегда перпендикулярна главной оси и поворачивается вместе с ней:

* Если главная ось направлена горизонтально, то поперечная ось смотрит вниз.
* Если главная ось направлена вертикально, то поперечная ось смотрит направо.

### Свойство justify-content, выравнивание вдоль главной оси

Аналог в тексте: text-align **(задается контейнеру)**

Вместо «горизонтального» выравнивания во флексбоксе используется свойство для распределения элементов вдоль главной оси — **justify-content**. Это свойство задаётся для флекс-контейнера.

По умолчанию **flex-start**. При этом значении элементы располагаются у начала главной оси. Чтобы располагались по центру главной оси, нужно задать для justify-content значение **center**.

А значение **flex-end** расположит элементы в конце главной оси.

### Свойство justify-content: space-between | space-around

У флексбокса есть значения justify-content, которые равномерно распределяют флекс-элементы вдоль главной оси:

* space-between — расстояния между соседними элементами одинаковые, между элементами и краями флекс-контейнера отступов нет.
* space-around — расстояния между соседними элементами одинаковые, между элементами и краями флекс-контейнера есть отступ, равный половине расстояния между соседними элементами.

**Небольшой итог:**

Свойство justify-content управляет распределением элементов вдоль главной оси и имеет пять значений:

* Значение по умолчанию: flex-start
* flex-end
* center
* space-between
* space-around

### Свойство align-items, выравнивание вдоль поперечной оси

**Это свойство тоже задаётся для флекс-контейнера**, в отличии от текстового аналога vertical-align.

По умолчанию является *stretch*. *Именно благодаря этому значению флекс-элементы растягиваются на всю «высоту» флекс-контейнера*. Если флекс-элементам задана высота, то растягиваться они не будут.

Чтобы выровнять по центру: **align-items: center**

### Свойство align-items: flex-start | flex-end

Чтобы расположить флекс-элементы в начале или в конце поперечной оси, нужно использовать значения flex-start и flex-end для свойства align-items.

Эти значения аналогичны значениям свойства justify-content, отличается только ось

### Свойство align-items: baseline

Если задать его контейнеру, то флекс-элементы будут выравниваться по базовой линии текста в них. Эта воображаемая линия проходит по нижней части букв.

Если выровнять флекс-элементы по базовой линии, то они выстроятся так, чтобы текст в них был как бы на «одной строке».

**Небольшой итог:**

* значение по умолчанию stretch,
* flex-start,
* flex-end,
* center,
* baseline.

### Свойство align-self, эгоистичное выравнивание

Полный аналог vertical-align у текста

Распределение элементов по главной оси задаётся для всего флекс-контейнера и на все флекс-элементы действует одинаково. Задать какому-то элементу отличное от других распределение по главной оси нельзя. И это вполне логично, ведь тогда элементы будут «сталкиваться» друг с другом.

C поперечной осью всё проще. Можно сказать, что у каждого элемента она своя, и можно задавать им разное поперечное выравнивание. Для этого используется свойство align-self, **которое задаётся для самих флекс-элементов**, а не для флекс-контейнера.

**У свойства align-self те же самые значения, что и у align-items.**

### Свойство align-self: baseline

Аналогично align-items для отдельного флекс-элемента можно задать и выравнивание по базовой линии с помощью align-self со значением baseline.

### Свойство flex-wrap, перенос flex-элементов

Что будет, если флекс-элементов в контейнере станет больше, чем может уместиться в один ряд?

* Они будут сжиматься до минимально возможной ширины.
* Даже если им задать ширину явно, механизм флексбокса может её уменьшить.
* Если они перестанут помещаться в контейнер и после уменьшения, то они выйдут за его пределы, но продолжат располагаться в один ряд.

Это можно изменить свойством флекс-контейнера **flex-wrap**. По умолчанию имеет значение **nowrap**.

Значение **wrap** разрешает перенос флекс-элементов на новую строку, если они не помещаются в контейнер.

### Свойство flex-wrap: wrap-reverse, обратный перенос

Если перенос флекс-элементов разрешён, то ряды элементов располагаются вдоль поперечной оси. Первый ряд располагается в начале поперечной оси, а последний в конце. Но так работает только значение wrap.

Если для flex-wrap задать значение wrap-reverse, то элементы будут переноситься, а ряды будут располагаться в обратном порядке: первый в конце поперечной оси, а последний в начале.

### Свойство align-content, выравнивание *строк* flex-контейнера

Похоже на justify-content, только управляет выравниванием **рядов** флекс-элементов **вдоль поперечной оси**. У этих свойств почти одинаковые значения:

* flex-start,
* flex-end,
* center,
* space-between,
* space-around,
* и stretch, которое есть у align-content, но нет у justify-content, и является значением по умолчанию.

Свойство align-content «перекрывает» заданное значение align-items, которое управляет выравниванием флекс-элементов вдоль поперечной оси. Это происходит и в случае, когда есть только один ряд флекс-элементов, и когда рядов несколько.

**align-content следует использовать только когда строк больше одной**

### Свойство align-content: stretch и align-items

В случае одновременного задания align-items и align-content свойство align-items не отключается полностью, а может влиять на отображение флекс-элементов в рядах.

**Это происходит, когда мы используем для align-content значение по умолчанию — stretch.** *Оно растягивает* ***ряды*** *флекс-элементов на всю свободную высоту контейнера*, при этом оставшееся свободное место между ними делится поровну. И только тогда внутри строк работает align-items

Отображение строк при align-content: stretch зависит от значения align-items:

* Если у align-items задано значение stretch, то элементы в строках растягиваются на всю высоту своей строки.
* Если значение отлично от stretch, то элементы в строках ужимаются под своё содержимое и выравниваются **в строках** в зависимости от значения align-items.

### Свойство align-content: не-stretch и align-items

Значение align-items влияет на отображение рядов во флекс-контейнере, если у align-content задано значение stretch. **Есть ли похожее влияние на остальные значения align-content? Нет.**

### Align-contnet – резюме

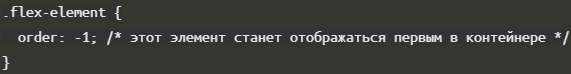
Свойство align-content — «гибридное». Мы переводим его как «выравнивание», но оно больше похоже на «распределение», justify-content, от которого оно позаимствовало два значения space-between и space-around.

Близость с «распределением» подчёркивает и отсутствие значения baseline — всё-таки свойство работает с рядами, а не с отдельными элементами.

От «выравниваний» же, align-items и align-self, это свойство получило значение по умолчанию stretch и возможность «растягивать» ряды по высоте.

### Свойство order, порядковый номер flex-элемента

Порядковый номер задаётся целым числом, положительным или отрицательным.



По умолчанию порядковый номер флекс-элементов равен 0, а сортировка элементов производится по возрастанию номера.

### Идеальное центрирование, margin: auto у flex-контейнера

Как отцентровать элемент по вертикали и горизонтали так, чтобы центровка сохранялась при изменении размеров элемента или контейнера?

Ответ прост: задать контейнеру раскладку флексбокса, а дочернему флекс-элементу margin: auto.

Стоит обратить внимание на интересный момент. Если центруемых флекс-элементов в контейнере будет несколько, то отступы между ними будут равномерными. То есть будет происходить распределение элементов внутри флекс-контейнера чем-то похожее на justify-content: space-around.

Тоже самое можно получить при использовании justify-content и align-items

# Флексбокс, часть 2

### Флекс-элементы и блочная модель

Ширина, высота, внутренние отступы и рамки для флекс-контейнеров и флекс-элементов работают как обычно: общий размер элементов складывается из этих компонентов. Это поведение также можно менять с помощью свойства box-sizing.

Есть и несколько важных отличий:

* флекс-элементы, в отличие от блочных элементов, не растягиваются на всю ширину контейнера по умолчанию;
* на флекс-элементы не действует свойство float.

### Особенности свойства margin

Оно таит много сюрпризов:

* внешние отступы не схлопываются, ни по горизонтали, ни по вертикали;
* внешние отступы не выпадают, ни из флекс-контейнера, ни из флекс-элементов;
* значение auto получило премию журнала Форбс в номинации «Самое влиятельное значение CSS-свойства внутри флекс-контейнера».

Всё дело в механизме распределения свободного места. Если внутри флекс-контейнера есть свободное место, то оно перераспределяется так:

* находятся элементы, у которых есть внешние отступы со значением auto;
* всё свободное место в соответствующих направлениях отдаётся таким отступам (то есть задаётся определённый размер отступа в пикселях);
* если элементов с автоматическими отступами на одном направлении несколько, то свободное место между ними перераспределяется поровну;

только после этого запускаются механизмы выравнивания.

Поэтому margin: auto; влияет на положение флекс-элементов на обеих осях, а также «ломает» механизмы выравнивания, ведь выравнивание происходит, когда есть свободное место. Но если всё свободное место «перетекло» в автоматический отступ, то и выравнивать нечего.

### Выравнивание (вертикальное) и внешние отступы

Автоматический margin влияет и на выравнивание флекс-элементов вдоль поперечной оси.

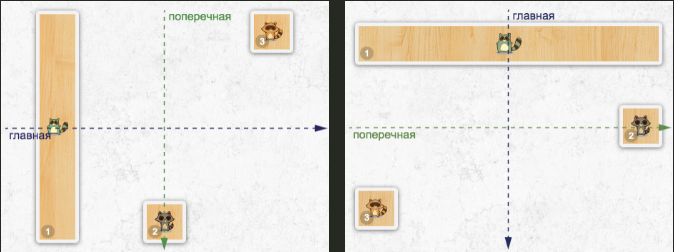
Если у флекс-элемента отступ сверху или снизу автоматический, то на него не влияют, ни align-items, ни align-self. Такой элемент прижмётся либо к верху контейнера, либо к низу.

А если задать автоматические отступы с противоположных сторон, то элемент разместится по центру флекс-контейнера, так как свободное место «впитается» отступами поровну.

**Margin вертикальный не работает когда строк несколько и когда align-content не stretch (вместе два условия)**

### Направление главной оси и внешние отступы

Будет результат таким, как на картинке снизу, если повернуть главную ось?



**Нет, не будет!** Дело в том, что «старые нефлексовые» свойства, такие как отступы или размеры, ничего не знают про направление осей. Они «мыслят по-старому», понятиями «верх» и «низ», «право» и «лево».

### Базовый размер флекс-элемента, flex-basis

На примере отступов видно, что «старые» свойства внутри флекс-контейнера ведут себя достаточно глупо. Ширина и высота тоже не умеют реагировать на поворот главной оси. Поэтому ввели понятия *главный размер* или *main size* и *поперечный размер* или *cross size.*

Если главная ось направлена горизонтально, то главный размер — это ширина, свойство width, а поперечный размер — это высота, свойство height. Если главная ось направлена вертикально, то всё наоборот.

Но есть свойство, которое знает про главную ось - это **flex-basis**. Оно задаёт базовый размер флекс-элемента или размер вдоль главной оси.

Если flex-basis не задан или его значение равно auto, то базовый размер берётся из width или height. Свойство flex-basis «сильнее» свойств width и height, и если у флекс-элемента заданы все три свойства, то flex-basis переопределит либо ширину, либо высоту в зависимости от направления главной оси.

### Коэффициент растягивания элементов, flex-grow

базовый размер (flex-basis) — это не просто размер элемента вдоль главной оси, это еще *начальный* или *исходный* размер вдоль главной оси.

Если внутри флекс-контейнера по главной оси остаётся свободное место, то мы можем попросить флекс-элемент, чтобы он увеличился и занял это место. Это делается с помощью свойства flex-grow которое можно назвать «коэффициентом флекс-жадности» флекс-элемента.

Значение по умолчанию — 0, может принимать положительные значения.

Если значение flex-grow равно нулю, то флекс-элемент «не претендует» на оставшееся свободное место во флекс-контейнере. Получается, что базовый размер — это исходный размер флекс-элементов до применения flex-grow.

### Пропорциональное растягивание элементов

Если сразу у нескольких флекс-элементов значение flex-grow больше нуля, то они будут делить свободное место между собой.

Свободное место будет добавляться флекс-элементам пропорционально значениям их «коэффициента жадности». Например, если во флекс-контейнере есть два элемента:

 то первому элементу достанется одна часть свободного места, а второму две части, всего частей будет три.

Если установить значения 50 и 100 – то ничего не измениться, так как важно не само значение коэффициента, а его соотношение с коэффициентами остальных элементов.

### Расчёт итогового размера с flex-grow (проверка уже подобранных коэфов)

1. Свободное место = Ширина контейнера - Сумма базовых размеров элементов
2. Доля свободного места = Свободное место / Сумма flex-grow всех элементов
3. Итоговый размер = Базовый размер + (Доля свободного места \* flex-grow)

*Но если задать флекс-элементам нулевой базовый размер, свободное место будет занимать всю ширину флекс-контейнера, и коэффициенты жадности будут другими.*

Использовать flex-basis: 0 и flex-grow для точного управления относительными размерами не стоит. Лучше использовать базовый размер в процентах.

**Тонкость.** На размер оставшегося свободного места влияет не только flex-basis, но и рамки, и отступы (border-box в помощь). Если flex-basis явно задано нулевое значение, то min-width на размер свободного места влиять не будет, так как ограничения размеров к флекс-элементам применяются уже после перераспределения свободного места.

Логика рассчета коэфов:



Оба элемента имеют flex-basis: 50px. Первый элемент должен увеличиться на **50px**, чтобы стать шириной в 100px, а второй элемент, чтобы увеличиться до 200px должен «съесть» остальные **150px.** По итогу: 50px от 200px это 1/4, а 150 от 200 – это 3/4 . Значит, первому элементу зададим коэф – 1, а второму - 3

### Коэффициент сжатия элементов, flex-shrink

Если сумма базовых размеров флекс-элементов больше, чем размер флекс-контейнера, то возникает *отрицательное пространство*.

Флекс-элементы умеют распределять отрицательное пространство между собой и сжиматься.

За уменьшение флекс-элементов отвечает свойство flex-shrink, которое можно назвать «коэффициентом сжатия». Свойство flex-shrink принимает неотрицательные числовые значения, его значение по умолчанию — 1.

Если значение flex-shrink больше нуля, то флекс-элемент будет уменьшаться, «впитывая» часть отрицательного пространства, если оно существует.

Если значение flex-shrink равно нулю, то флекс-элемент уменьшаться не будет.

Флекс-элементы стараются быть максимально «гибкими» и не выпадать из своего контейнера, поэтому у flex-shrink значение по умолчанию равно 1. Но если задавать нулевые значения для коэффициента сжатия, то выпадения элементов добиться можно.

### Пропорциональное сжатие элементов

Свойство flex-shrink очень похоже на свойство flex-grow. Оно задаёт пропорции, в которых флекс-элементы «впитывают» отрицательное пространство. Чем больше значение коэффициента сжатия у элемента, тем больше отрицательного пространства он «впитает» и тем сильнее сожмётся.

### Расчёт итогового размера с flex-shrink (проверка подобранных коэфов)

1. Рассчитываем отрицательное пространство (ОП) во флекс-контейнере:

**ОП = Ширина контейнера - Сумма базовых размеров элементов**

1. Находим сумму произведений базовых размеров (СПБР) элементов на их коэффициенты сжатия:

**СПБР = (Базовый размер1 \* flex-shrink1) + (Базовый размер2 \* flex-shrink2) + … + (Базовый размерn \* flex-shrinkn)**

1. Для каждого элемента считаем «нормированный коэффициент сжатия» (НКС), для чего произведение базового размера элемента на его коэффициент сжатия делим на СПБР:

**НКС = (Базовый размер \* flex-shrink) / СПБР**

1. Базовый размер элемента уменьшаем на часть ОП пропорциональную НКС элемента. ОП для расчёта берём по модулю, то есть отбрасывая минус:

**Итоговый размер = Базовый размер - (НКС \* ОП)**

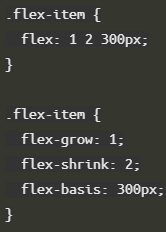
**Чтобы подобрать коэфы под размеры можно составить уравнение.**

*Есть несколько тонкостей, касающихся сжатия флекс-элементов:*

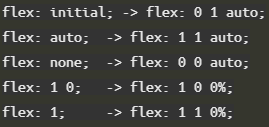
* *элементы сжимаются в пределах своих базовых размеров, внутренние отступы и рамки не сжимаются (border-box поможет);*
* *«ограничительные» свойства, такие как min-width, применяются к элементам после этапа перераспределения свободного места или отрицательного пространства.*

***И эти тонкости могут приводить к неожиданным эффектам, когда элементы выпадают из флекс-контейнера.***

### Сокращённое свойство flex

С помощью сокращённого свойства flex можно одновременно задать коэффициенты растягивания, сжатия и базовый размер флекс-элемента.

Ещё у свойства flex есть особые значения: initial, auto, none. Также второй и третий компоненты необязательны. Ниже показаны различные значения свойства и их расшифровки.



### Многострочный флекс-контейнер и flex-shrink

В таком контейнере свойство flex-shrink будет работать как обычно, но необходимость в нём будет возникать намного реже. Ведь при нехватке места в строке флекс-элементы будут переноситься на новую строку.

Но если появятся флекс-элементы, базовый размер которых больше размера флекс-контейнера, то такие элементы будут сжиматься и занимать целую строку. Наверное, это единственный случай, когда flex-shrink делает что-то полезное в многострочном контейнере.

### Многострочный флекс-контейнер и flex-grow

свойство flex-grow в многострочном флекс-контейнере срабатывает намного чаще и пользы приносит намного больше.

В каждой строке такого контейнера может быть свободное место и механизм перераспределения этого места работает построчно.

### flex-basis: 100% и flex-wrap

Если задать базовый размер флекс-элемента 100% и при этом включить перенос элементов на новую строку, то элементы расположатся столбцом, хотя главная ось контейнера будет по-прежнему направлена слева направо.

Это логично, потому что каждый элемент будет занимать всю ширину строки

### Поля ввода с динамической шириной

Требования:

1. *На одной строке с полем могут находиться другие элементы: кнопки, ссылки, подписи.*
2. *Размер дополнительных элементов не определён, он зависит от их содержимого.*
3. *При этом поле должно растягиваться на всё оставшееся в родительском контейнере место.*
4. *И изменять ширину при изменении размеров контейнера.*

* Превратим контейнер поля ввода во флекс-контейнер, все элементы внутри него превратятся во флекс-элементы, базовый размер которых будет зависеть от их содержания — flex-basis: auto;. И останется только задать ненулевой коэффициент растягивания полям ввода.
* Проблемы могут появиться в слишком узких контейнерах: по умолчанию поля ввода не будут сжиматься после определённой ширины, что приведёт к выпаданию текста из остальных элементов.
* Чтобы справиться с этими проблемами, надо задать всем элементам кроме полей ввода нулевой коэффициент сжатия, а самим полям ввода явно прописать минимальную ширину, чтобы они могли сжаться сильнее до этой ширины

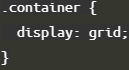
# Знакомство с гридами

### Эксперименты наглядно

<https://htmlacademy.ru/courses/269/run/14> - можно поиграться с рядами и столбцами.

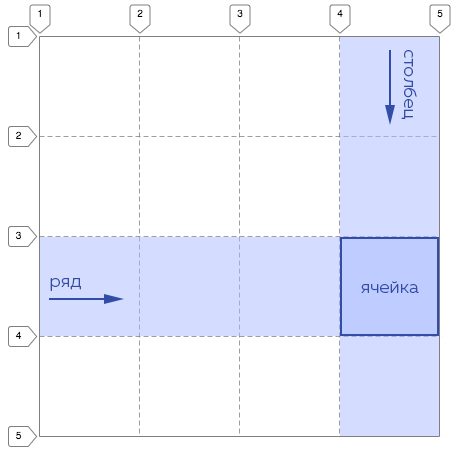
### Координаты грид-элементов: grid-row-start и grid-column-start

Чтобы сделать блок грид-контейнером, нужно задать ему соответствующее значение свойства display:



Дочерние элементы в грид-контейнере становятся грид-элементами и располагаются согласно раскладке грида.

В гриде элементы располагаются по двумерной сетке. То есть грид состоит из рядов и столбцов, располагающихся между линий, которые нумеруются по порядку:



А чтобы расположить элемент внутри грида, нужно задать ему координаты. То есть заставить элемент говорить: «Я — элемент грида, начинаюсь с такой-то линии столбцов и с такой-то линии рядов.»

Когда мы задаём координату начала грид-элементу по рядам или столбцам, он *автоматически продлевается до следующей линии рядов или столбцов* и таким образом образуется грид-ячейка.

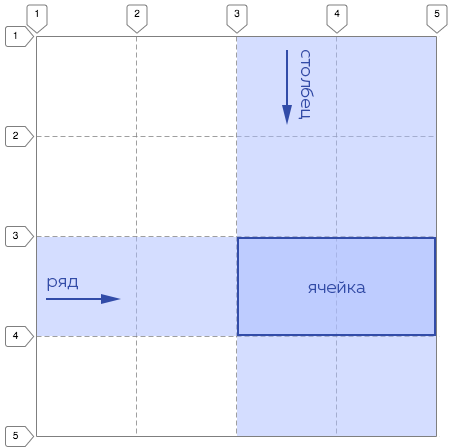
На примере выше ячейка начинается на 4 линии столбцов и 3 линии рядов. В коде делается это с помощью свойств grid-column-start и grid-row-start:



### Координаты грид-элементов: grid-column-start и grid-column-end

Теперь мы знаем, что будет, если указать начало грид-ячейки по ряду и по столбцу — её размер автоматически станет равен одному «делению» грида.

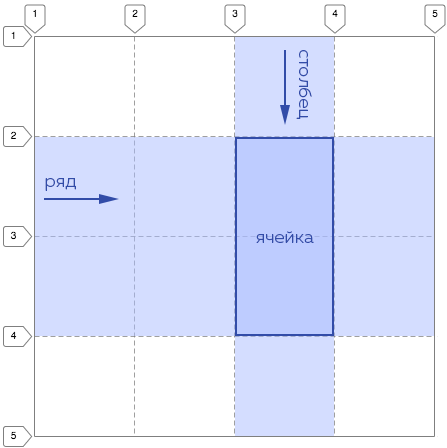
Если нужно продлить грид-ячейку на несколько рядов или столбцов, то нужно указать помимо той линии, где ячейка начинается, ещё ту линию, где она заканчивается:





### Координаты грид-элементов: grid-row-start и grid-row-end

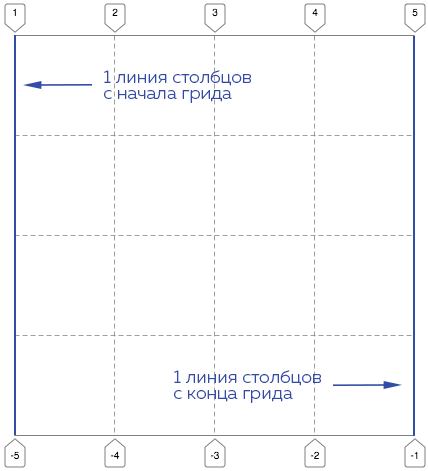
**grid-row-end** – позволяет растянуть ячейку и по рядам тоже:



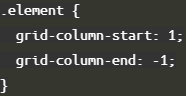


### Отрицательные значения grid-column-start и grid-column-end

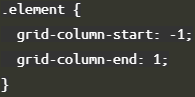
Интересной особенностью грид-ячейки является возможность начинать отсчёт её координат как от начала (первой линии ряда или столбца), так и от конца грида (последней линии ряда или столбца). Вот как это выглядит схематично:



То есть, чтобы сказать ячейке: «Начинайся на первой линии столбцов от начала и заканчивайся на первой линии столбцов с конца», нужно написать такой код:



А для того чтобы сказать ячейке: «Начинайся на первой линии столбцов с конца грида и заканчивайся на первой линии с начала», нужно написать:



**Аналогично и для rows!**

### Наложение элементов грида

В случае пересечения координат двух элементов грида они просто располагаются один поверх другого в порядке следования в разметке, как будто это стопка листов. Напоминает абсолютное позиционирование, не правда ли?

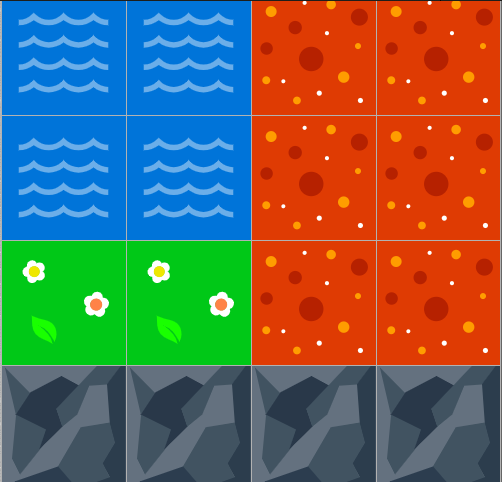
### Наложение элементов грида и свойство z-index

Как и на абсолютно спозиционированные элементы, на пересекающиеся грид-элементы действует старое-доброе свойство z-index.

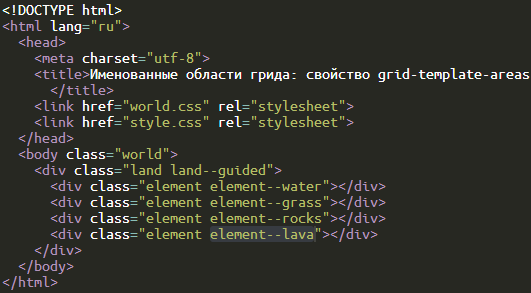
Чем выше z-index, тем выше элемент в «стопке». Если у двух элементов одинаковый z-index, то выше будет располагаться тот, который идёт следующим по разметке.

### Именованные области грида: свойство grid-template-areas & grid-area

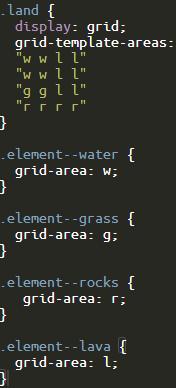
Легче показать:



HTML



CSS



В свойстве ***grid-template-areas*** мы указываем расположение наших блоков.

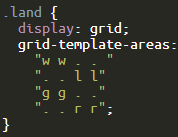
В свойствах блоков ***grid-area***мы указываем каким грид-элементом является текущий div в свойстве grid-template-areas.

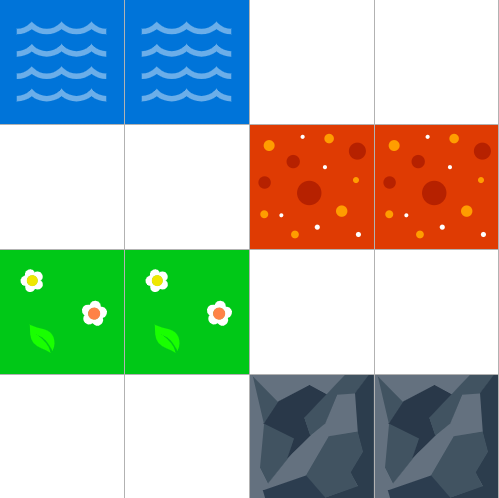
Ограничения:

* Грид должен быть прямоугольной формы, а количество объявленных столбцов в каждой строке должно быть одинаковым.
* Одному элементу в HTML может соответствовать только одна грид-область
* Названия областей должны начинаться с буквы и могут включать цифры и дефис. Например, "lava lava2 lava-3"

### Свойство grid-template-areas и пустые области грида

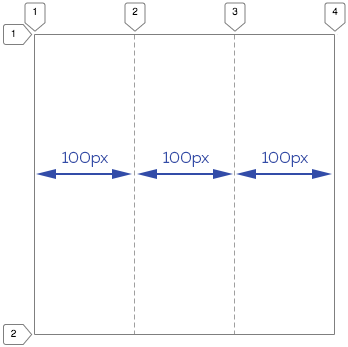
Свойство ***grid-template-areas*** позволяет некоторые области помечать как пустые. Для этого вместо буквенного именования области используется символ точки «.»:





### Автоматические координаты элементов в гриде: столбцы

Свойство ***grid-template-columns***перечисляет количество и ширину будущих столбцов грида. К примеру, запись *grid-template-columns: 100px 100px 100px*; говорит гриду: «*Ты будешь иметь три столбца, каждый шириной 100px.»:*

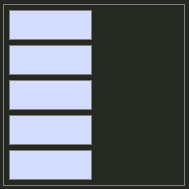
**

**Если не задавать параметры для рядов**, то их количество становится переменным. Порядок действий такой:

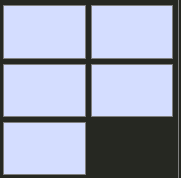
1. Все грид-элементы выстраиваются в один ряд согласно объявленной раскладке столбцов.
2. Если грид-элементов больше, чем столбцов, то не вписавшиеся в один ряд элементы переносятся на следующий ряд и заполняют его.
3. Новые ряды формируются до тех пор, пока все грид-элементы не будут вписаны в раскладку по столбцам. Если последний ряд заполняется элементами не полностью, то оставшееся место в гриде остаётся пустым.

Пример:

Возьмём грид с пятью элементами, которому задан один столбец. По раскладке получается, что в первый ряд грида может поместиться всего один грид-элемент. Остальные элементы будут переноситься на новые ряды, и таким образом сформируют пять рядов, то есть выстроятся в столбец:



Если же мы увеличим количество столбцов до двух, то в один ряд будут помещаться уже по два элемента. То есть всего элементы сформируют три ряда, в первом и втором ряду будет по два элемента, а в третьем — один:



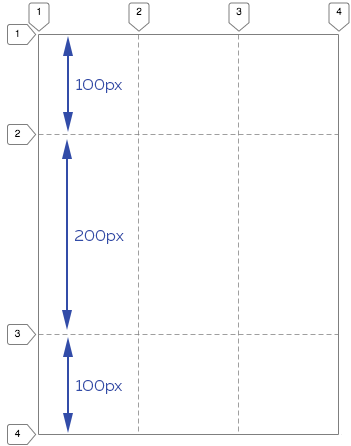
Что касается высоты рядов, то она распределится равномерно, чтобы ряды заполнили собой всё пространство грида.

### Автоматические координаты элементов в гриде: столбцы и ряды

Аналогично работает и свойство ***grid-template-rows***, только оно сообщает гриду сколько рядов он будет содержать и какой они будут высоты.

Запись *grid-template-rows:* 100px 200px 100px; говорит гриду:

"Ты будешь иметь три ряда, первый высотой 100px, второй высотой 200px и третий — 100px.»:



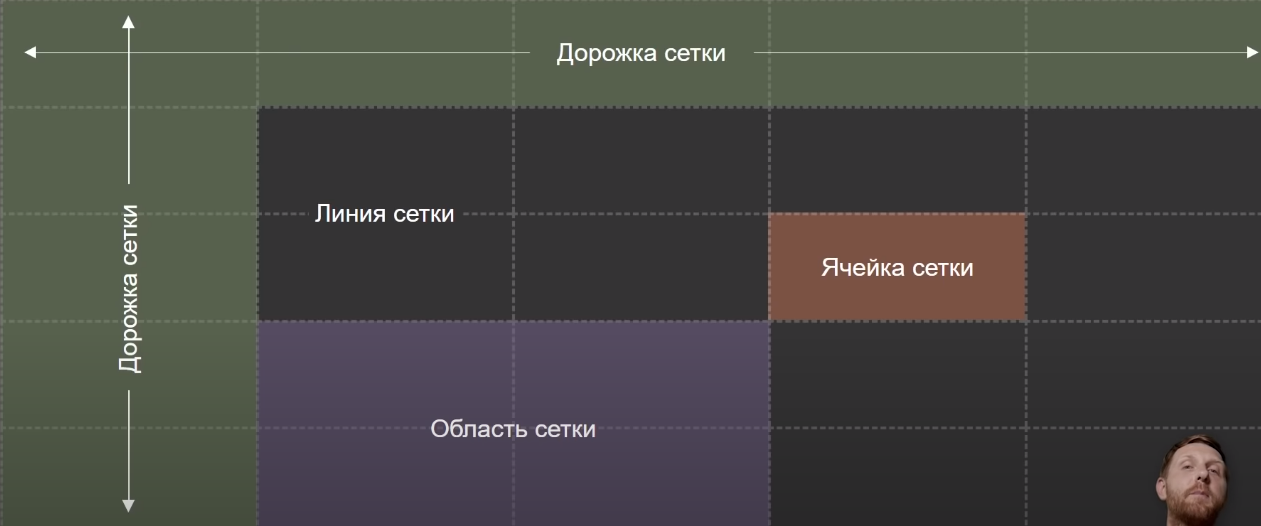
Если при этом столбцам не заданы явные параметры, то по умолчанию, будет существовать всего один столбец, элементы в котором растянутся на всю его ширину.

В случае, если в гриде есть ряды, которым *не задана явная высота* в **grid-template-rows**, оставшееся свободное пространство по высоте внутри контейнера распределится среди них равномерно.

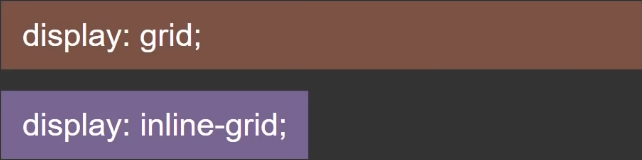
**Грид по умолчанию будет иметь всего один столбец во всю ширину контейнера, а каждый его элемент будет занимать целый ряд.**

# Знакомство с гридами, часть 2. Подробнее про грид-контейнер

### Визуализация

****

* *display: grid* – грид контейнер займет всю ширину родителя
* *display: inline-grid –* грид контейнер ужмется по содержимому



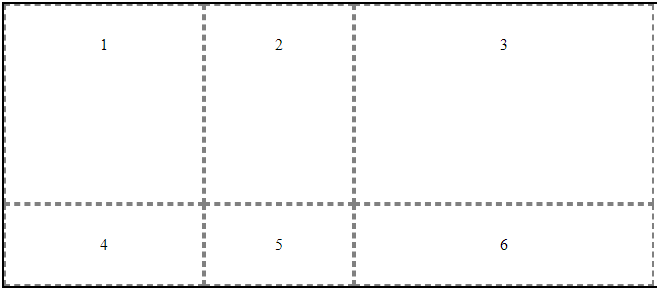
### Параметры сетки

Когда мы задаем такие параметры сетки:



Мы указываем, что явно будет только один столбец в 200px и явно высота первой строки будет 200px. Все, что не влезет в эту сетку – будет перенесено на следующую строку, но высота в 200px больше цепляться к элементам в неявной сетке не будет. А ширина колонки останется 200px.

Пример:



Для этой сетки заданы стили:



То есть в свойствах есть только определение высоты первой строки в 200px. Все, что не влезло в первую строку – не зависит от этой величины.

* Если у серых дивов не указаны размеры – они будут занимать всю ячейку полносью!

### Расположение элементов в сетке по умолчанию

Элементы, которые находятся внутри грид-контейнера будут по порядку помещены в ячейки сетки друг за другом и по умолчанию занимают ровно одну ячейку грида, если не заданы размеры меньше размера ячейки, либо не указаны свойства *grid-column-start, grid-column-end (grid-row-start, grid-row-end),* которые управляют положением элемента в гриде.

### Фракции грида

В этой записи указано, что все три колонки будут одинаковых размеров:



А эта, что вторая будет в два раза больше первой и третьей:



Для рядов это тоже применимо.

Этот подход позволяет не проводить расчеты при указании отступов между колонками грида

### Ограничение по минимальной/максимальной ширине

Тут вторая колонка будет стремится занять все место, а когда сжимаешь – она будет уменьшаться, но не станет уже 150px:



Эта запись говорит, если есть место, то расширься до 200px, если нет, то сожмись, но максимум до 150px:



### Расширение исходя из содержимого ячейки

Эта запись позволяет расширять ширину первой колонки по содержимому:

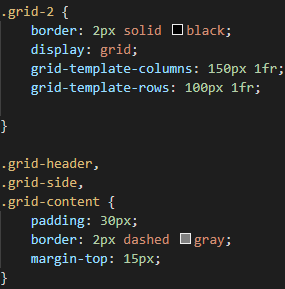


Эта запись делает тоже самое, но с ограничением расширения до 400px:

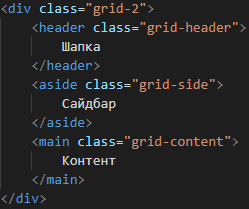


### Именованные области, пример

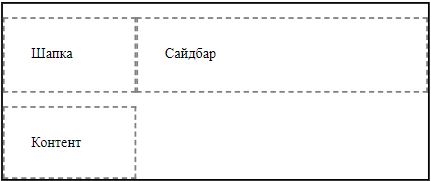
CSS:



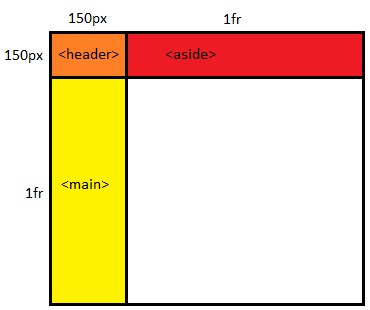
HTML:



Структура получилась следущей:

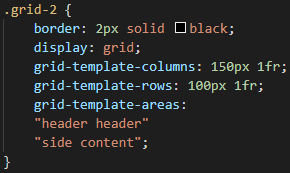


Почему так? Потому каждый грид элемент попал друг за другом в свою ячейку:

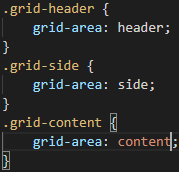


**Решение:**

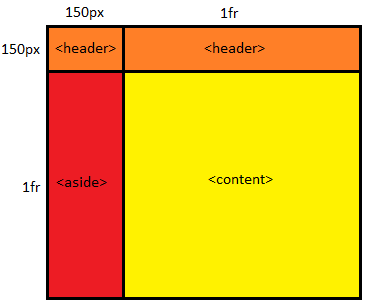
Мы можем указать, какие ячейки должны занимать грид-элементы в сетке:

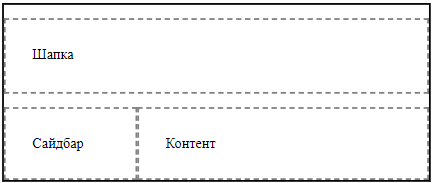


Но также нужно связать наши элементы с этими псевдонимами:



Теперь элементы поместились в ячейки грида так, как нам нужно:

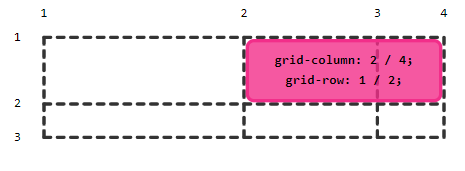




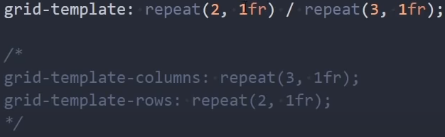
### **Сокращения**

grid-column, grid-row:

* *grid-column*: сокращенная запись для свойств *grid-column-start и grid-column-end.*
* *grid-row:* сокращенная запись для свойств *grid-row-start и grid-row-end.*
* *grid-area:* сокращенная запись для всех 4 свойств сразу..

****

**grid-template-columns, grid-template-rows:**

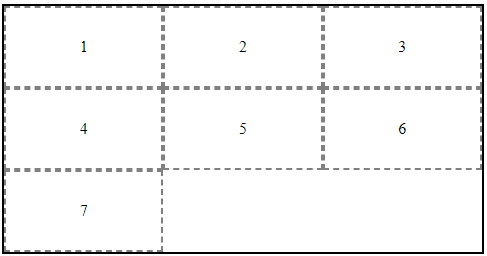
****

### Неявная сетка

Объявлено три столбца и два ряда одинаковых размеров:



Согласно такой сетке, явной будет сетка 3x2, только 6 элементов попадут в явную сетку, остальные попадут в неявную (7 элемент):



Чтобы задать значения высоты неявного ряда есть свойство:



Для столбцов есть *grid-auto-columns*, но оно работает только когда явно столбцы вообще не заданы, потому что элементы переносятся на новую строку, а не новый столбец, если не влезают в явную сетку.

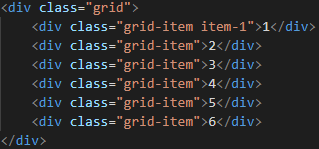
# Знакомство с гридами, часть 3. Подробнее про грид-элементы

### Интересные особенности грид-элементов

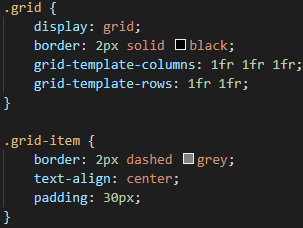
* Если, например, span имеет родителя с display: grid – то он становится блочным элементом.
* margin’ы у грид-элементов не схлопываются во всех направлениях
* Вычисление width и height грид-элементов происходят в зависимости от свойств элемента сетки align-items.
* С помощью свойства order можно управлять порядком вывода грид-элементов

### Указание линий грид-элементу, пример

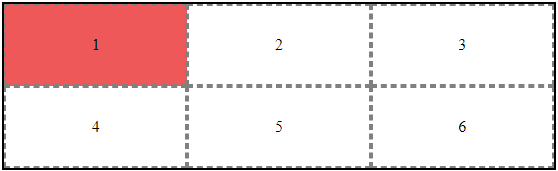
HTML:



CSS:



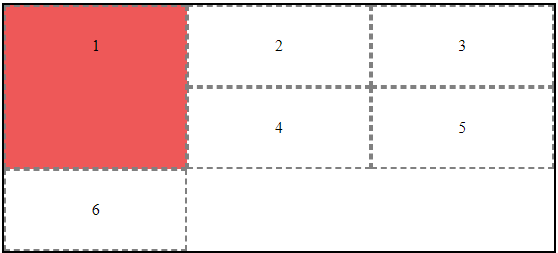
Overview:



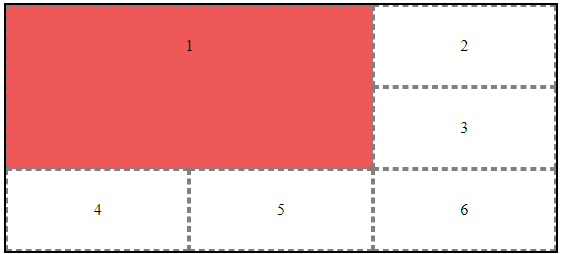
Теперь зададим, что первый элемент будет занимать две строки в высоту:



Следовательно, шестой (последний) элемент будет вытеснен на новую строку:



А если еще указать, что он будет занимать две колонки в ширину, то и вовсе получится, что он займет 4 ячейки грида и вытеснит уже три последних элемента на новую строку:



Таким образом блоки вообще можно двигать в разные места сетки

### Выравнивание элементов сетки

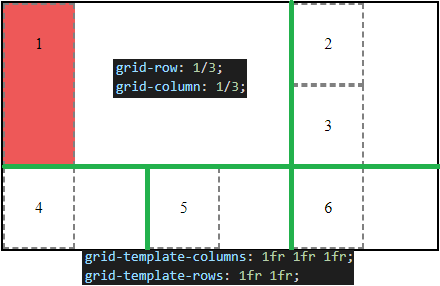
По умолчанию элемент грида заполняет всю ячейку, если содержимого для этого недостаточно (и ему не заданы размеры).

Это обуславливается свойствами *justify-items* и *align-items,* которые stretch по умолчанию. Задаются контейнеру!

* **justify-items:**

1. *start*; - элементы прижимаются к левой стороне своих ячеек и ужимаются по содержимому.

Причем, если элемент занимал несколько ячеек ранее, то в эти ячейки все равно никто не зайдет, хоть даже и размер элемента может не позволять их занять:



1. *end;* - тоже самое, только элементы прижмутся к правой стороне.

* **align-items:**

Те же свойства start и end, только работают вертикально (прижимает к верху или к низу)

* Так же можно **отдельно выравнивать грид-элементы** с помощью свойств *align-self* и *justify-self*
* Чтобы отцентровать элемент посередине ячейки можно использовать *margin: auto*

### Расстояние между рядами и колонками сетки



