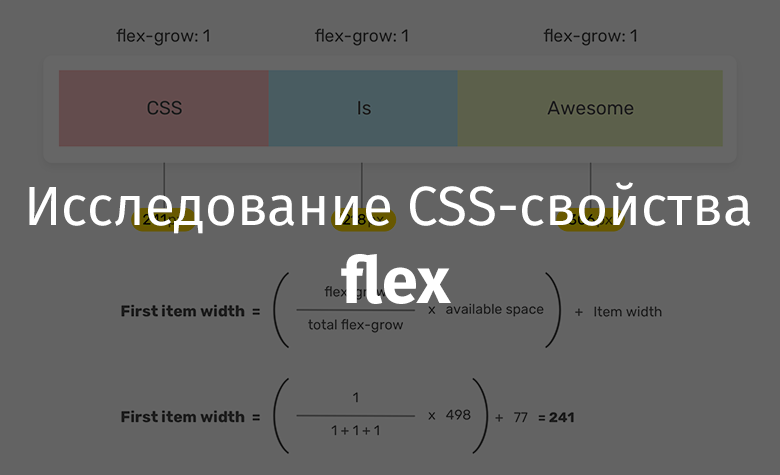
Исследование CSS-свойства flex

[Автор оригинала: Ahmad Shadeed](https://ishadeed.com/article/css-flex-property/)

* [Блог компании RUVDS.com](https://habr.com/ru/company/ruvds/),
* [Разработка веб-сайтов](https://habr.com/ru/hub/webdev/),
* [CSS](https://habr.com/ru/hub/css/)
* Перевод

Вам когда-нибудь было интересно узнать о том, как работает сокращённое CSS-свойство flex? Оно позволяет задавать значения свойств flex-grow, flex-shrink и flex-basis. Я обратил внимание на то, что данное свойство чаще всего используют в виде flex: 1, что позволяет flex-элементу растягиваться, занимая доступное пространство.  
  
[](https://habr.com/ru/company/ruvds/blog/515298/)  
  
В этом материале я хочу поговорить о сокращённом свойстве flex и о тех свойствах, значения которых устанавливают с его помощью. Я собираюсь рассказать о том, когда и почему можно пользоваться этими свойствами, приведу практические примеры.

Свойство flex-grow

CSS-свойство flex-grow используется для настройки так называемого «коэффициента роста» элементов (flex grow factor), что позволяет этим элементам растягиваться, занимая доступное пространство. В качестве значений этого свойства можно использовать только целые числа. Рассмотрим пример.  
  
Вот HTML-разметка:

<div class="wrapper">

  <div class="item item-1"></div>

  <div class="item item-2"></div>

  <div class="item item-3"></div>

</div>

Вот стили:

.wrapper {

    display: flex;

    flex-wrap: wrap;

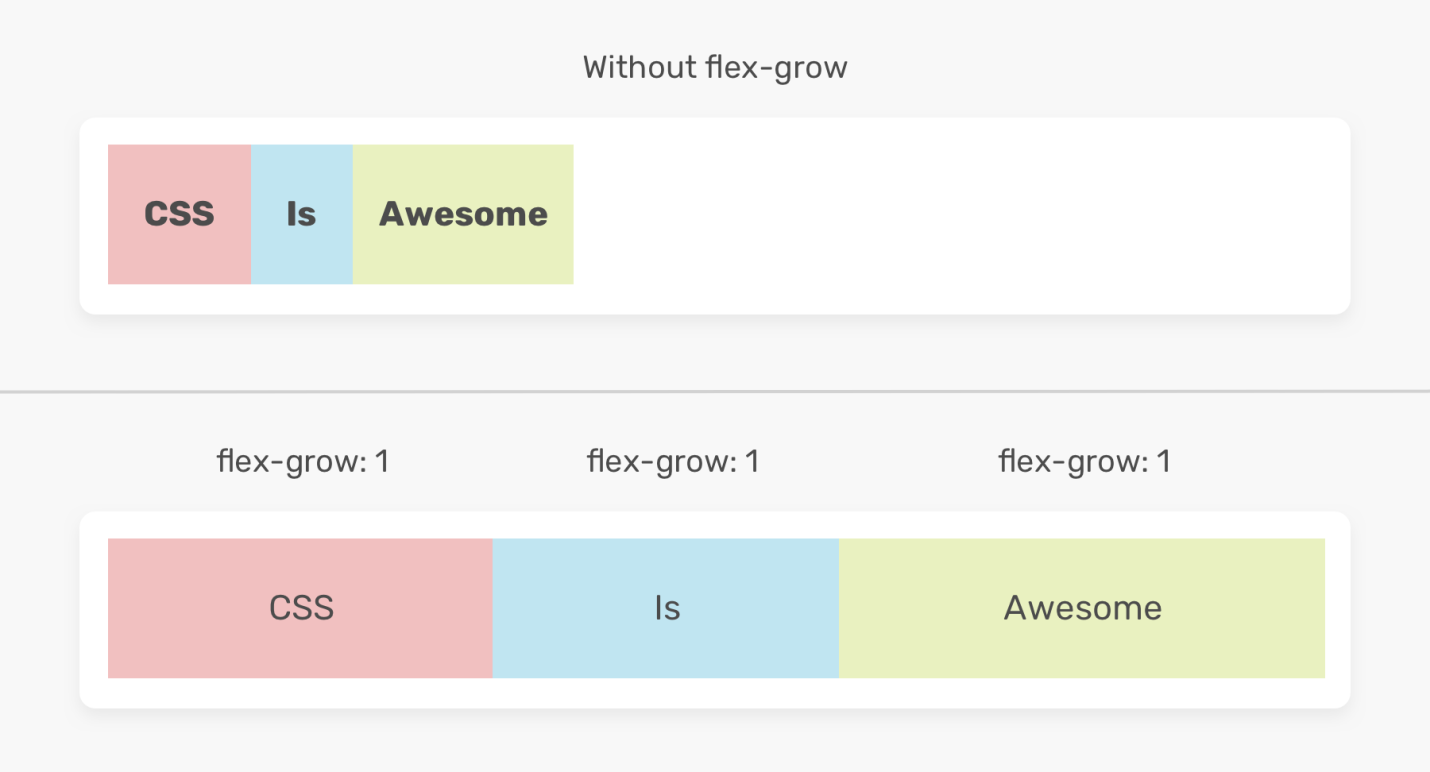
}

.item {

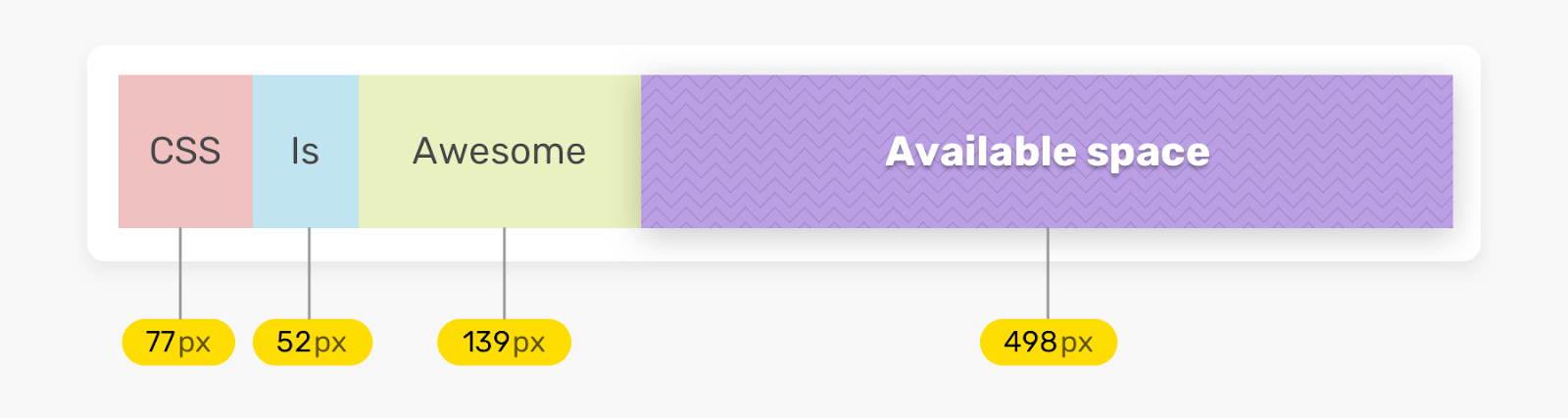
    flex-grow: 1;

}

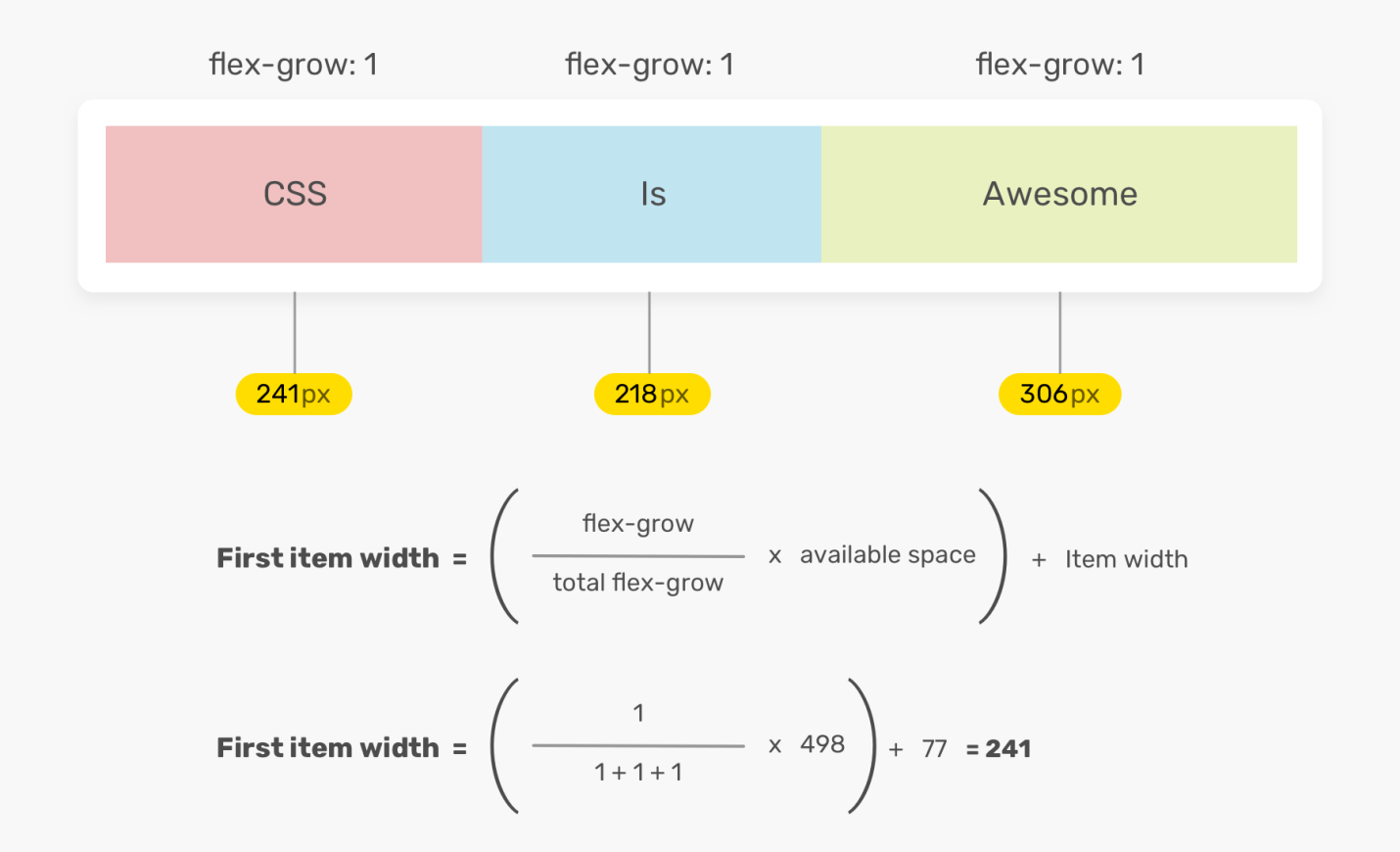
Свойство flex-grow может воздействовать на ширину или на высоту элемента, что зависит от значения свойства flex-direction. Рассматривая следующие примеры, учитывайте то, что используемое в них свойство flex-direction установлено в значение, задаваемое по умолчанию (row). Если это будет не так — я об этом скажу.  
  
Обратите внимание на то, что без использования flex-grow ширина flex-элементов будет установлена в значение, применяемое по умолчанию, то есть — она будет равна их исходной ширине. А если используется flex-grow: 1, то доступное свободное пространство распределяется между элементами.



*Вверху свойство flex-grow не применяется. Внизу применяется flex-grow: 1*  
  
Возможно, тут у вас возникнет вопрос о том, как именно между flex-элементами распределяется свободное пространство. Это — хороший вопрос. Скоро я на него отвечу.  
  
На следующем рисунке показано то, как элементы выглядят без использования свойства flex-grow. Другими словами, здесь показаны элементы, имеющие свой обычный размер.



*Свойство flex-grow не используется*  
  
Для того чтобы понять то, как вычисляется ширина flex-элементов, взгляните на формулы, показанные ниже. Я нашёл эти формулы в [материале](https://www.samanthaming.com/flexbox30/22-flex-grow-calculation/) Саманты Минг (за что её благодарю!).  
  
Давайте посчитаем ширину первого элемента — того, в котором находится текст CSS.



*Нахождение ширины элемента*  
  
Итак, тут используется такая формула:

Ширина элемента = ((flex-grow элемента / сумма значений flex-grow) \* доступное пространство) + исходная ширина элемента

Разберём эту формулу:

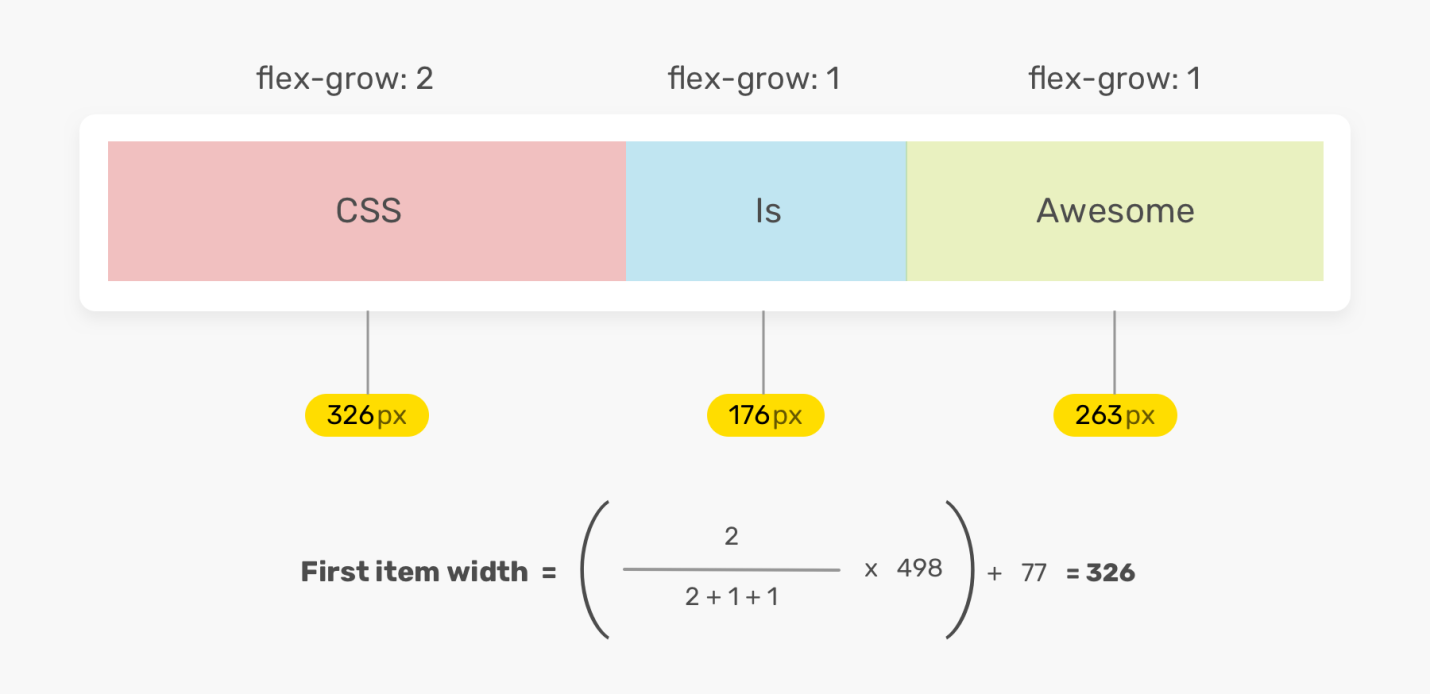
* flex-grow элемента — это коэффициент роста, заданный для элемента.
* сумма значений flex-grow — это сумма значений коэффициентов роста всех элементов.
* доступное пространство — это свободное пространство контейнера, которое имеется в нём до применения механизмов flex-grow.
* исходная ширина элемента — это, как несложно догадаться, ширина элемента до её изменения с помощью flex-grow.

В результате подстановки в эту формулу реальных значений получается следующее:

Ширина элемента = ( (1 / 3) \* 498) + 77 = 241

Разные значения flex-grow, задаваемые для разных элементов

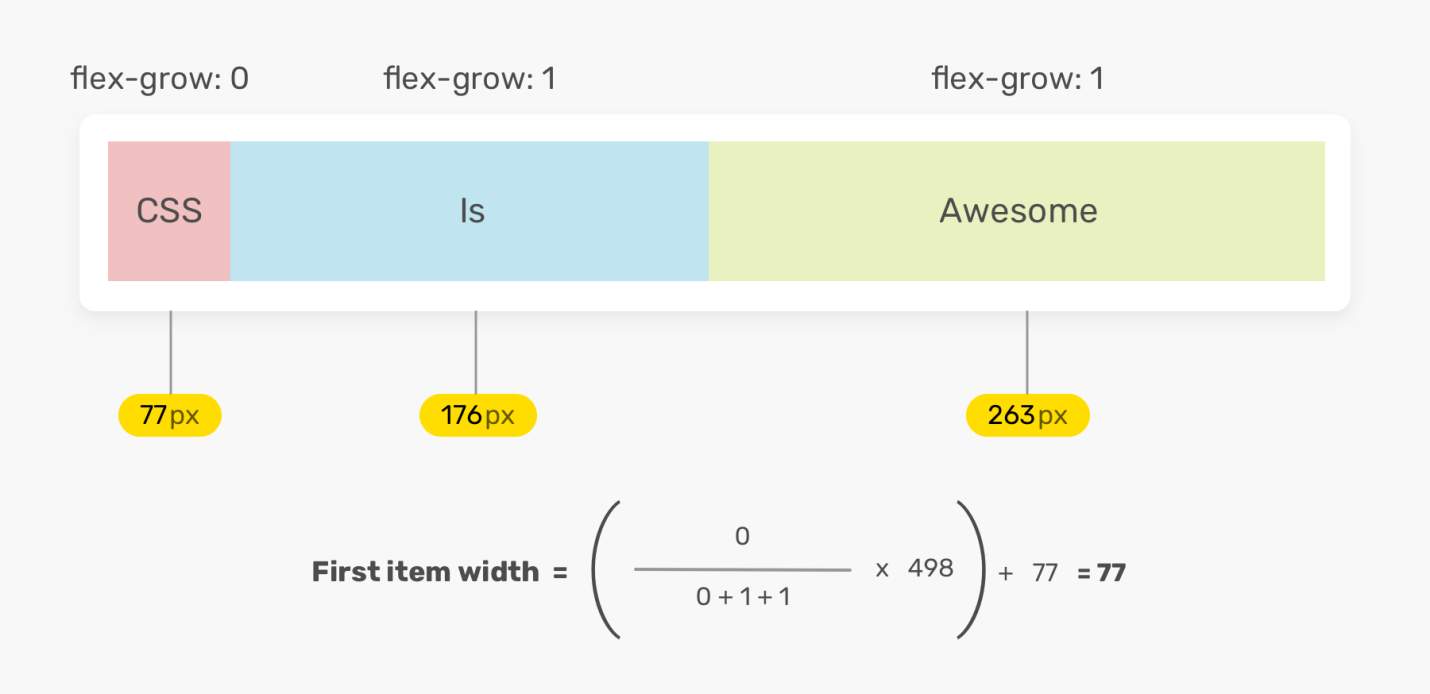
В предыдущем примере для всех flex-элементов использовалось одно и то же значение flex-grow. Попробуем теперь назначить первому элементу свойство flex-grow: 2. Как будет вычисляться ширина элементов теперь? Рассматривая следующую формулу, помните о том, что ширина свободного пространства в нашем примере равняется 498px.



*Нахождение ширины элемента в ситуации, когда разным элементам заданы разные значения flex-grow*  
  
Как видите, тут используется тот же механизм расчёта ширины элементов, который был рассмотрен выше. Главное отличие заключается в том, что значение flex-grow для первого элемента задано как 2, что находит отражение в формуле расчёта ширины элементов.

Можно ли использовать 0 в качестве значения flex-grow?

Конечно можно! Так как свойство flex-grow принимает целочисленные значения, в него можно записать и 0. Это будет означать, что мы не хотим, чтобы flex-элемент менял бы размеры, занимая некоторую часть свободного пространства контейнера.



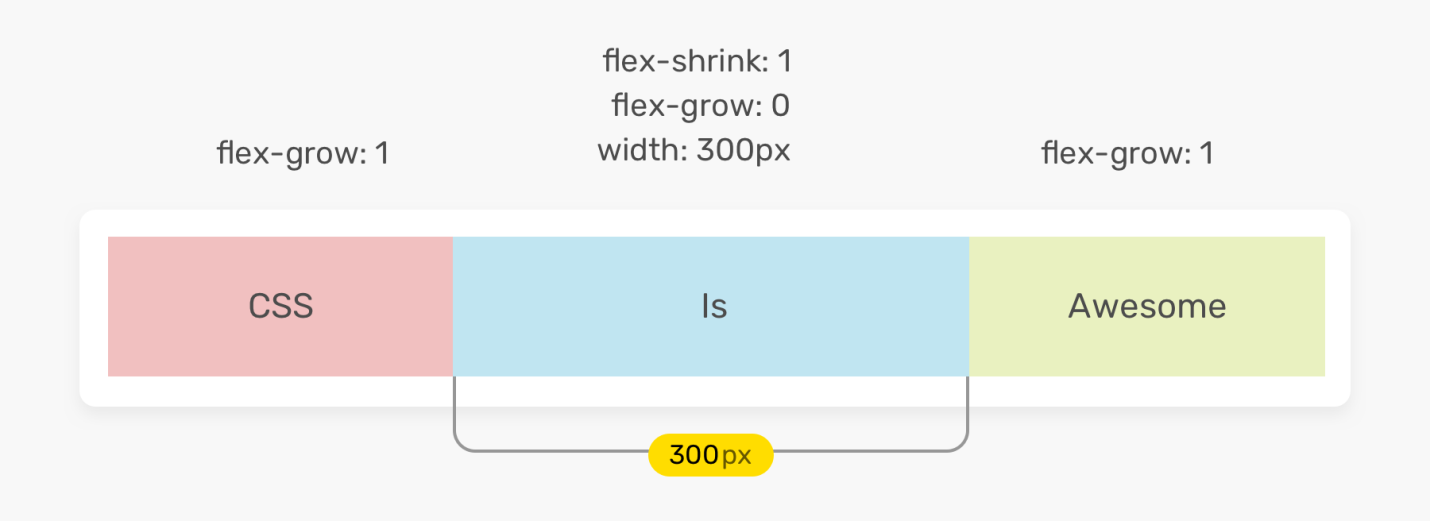
*Последствия установки свойства flex-grow в значение 0*  
  
Как видите, элемент, которому назначено свойство flex-grow: 0, не меняет ширину. Этот приём может быть полезен в тех случаях, когда нужно, чтобы некий flex-элемент сохранял бы свою исходную ширину.

Использование flex-grow не приводит к тому, что элементы становятся одинаковыми

Существует распространённое заблуждение, в соответствии с которым использование flex-grow позволяет сделать так, чтобы соответствующие элементы имели бы одинаковую ширину. Это — ошибка. Смысл использования flex-grow заключается в распределении доступного пространства между элементами. Как вы могли видеть, анализируя вышеприведённые формулы, итоговая ширина flex-элементов вычисляется на основе их исходной ширины (то есть той, которую они имели до применения flex-grow).  
  
Если вам нужно сделать так, чтобы все элементы из некоего набора имели бы одинаковую ширину, то знайте, что сделать это можно, воспользовавшись свойством flex-basis. Мы поговорим об этом ниже.

Свойство flex-shrink

Свойство flex-shrink позволяет задать так называемый «коэффициент сжатия» элемента (flex shrink factor). Если размер всех flex-элементов больше, чем размер их контейнера, размер элементов будет уменьшен в соответствии с назначенными им значениями свойства flex-shrink.



*Центральному элементу назначено свойство flex-shrink: 1*  
  
Вот стили к этому примеру:

.item-2 {

    width: 300px;

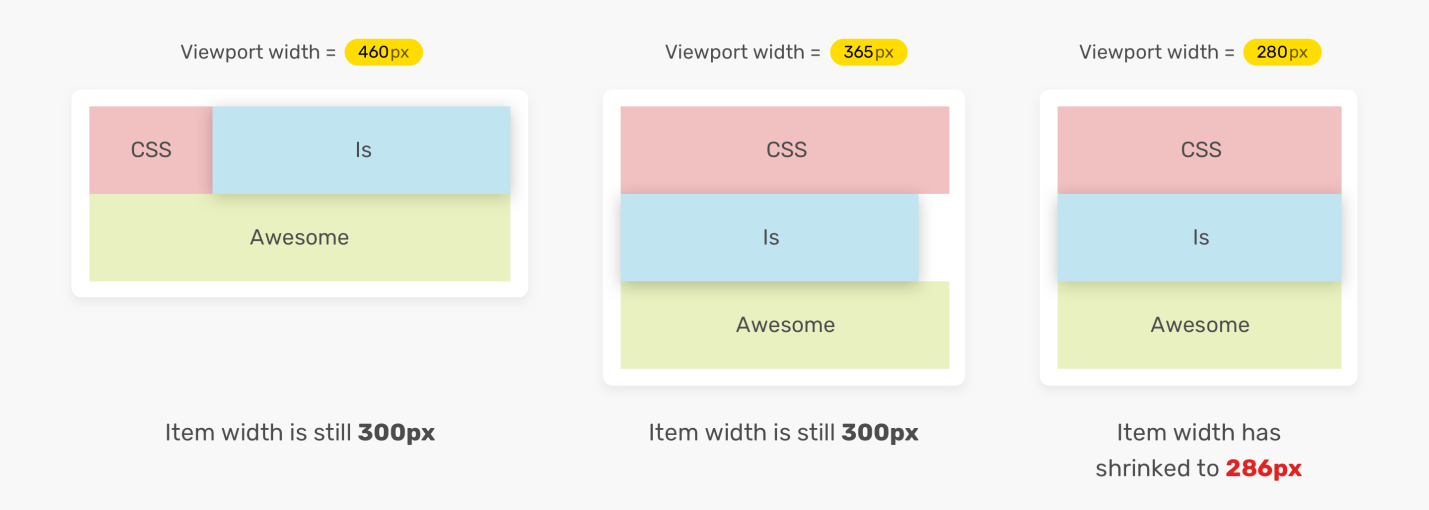
    flex-shrink: 1;

}

Браузер сделает ширину элемента item-2 равной 300px при выполнении следующих условий:

* Общая ширина всех элементов меньше ширины контейнера.
* Ширина области просмотра страницы равна или больше ширины элемента.

Вот как элементы из предыдущего примера ведут себя при их выводе в областях просмотра разной ширины.



*Ширина элемента с текстом Is не уменьшается до тех пор, пока на экране достаточно места для его вывода*  
  
Как видите, элемент не сжимается, сохраняя ширину в 300px до тех пор, пока ему хватает места.

Свойство flex-basis

Свойство flex-basis задаёт базовый размер flex-элемента, который он имеет до распределения свободного пространства в соответствии с коэффициентами гибкости, задаваемыми свойствами flex-grow и flex-shrink. По умолчанию, для всех значений, кроме auto и content, значение flex-basis равняется ширине элемента, а при использовании свойства flex-direction: column — его высоте.  
  
Свойство flex-basis принимает те же значения, которые могут принимать свойства width и height. Значением, задаваемым по умолчанию, является auto, которое устанавливается на основе значения content. Значение content задаётся автоматически, на основе размера содержимого flex-элемента.  
  
Применим к элементу item-1 из нашего примера следующие стили:

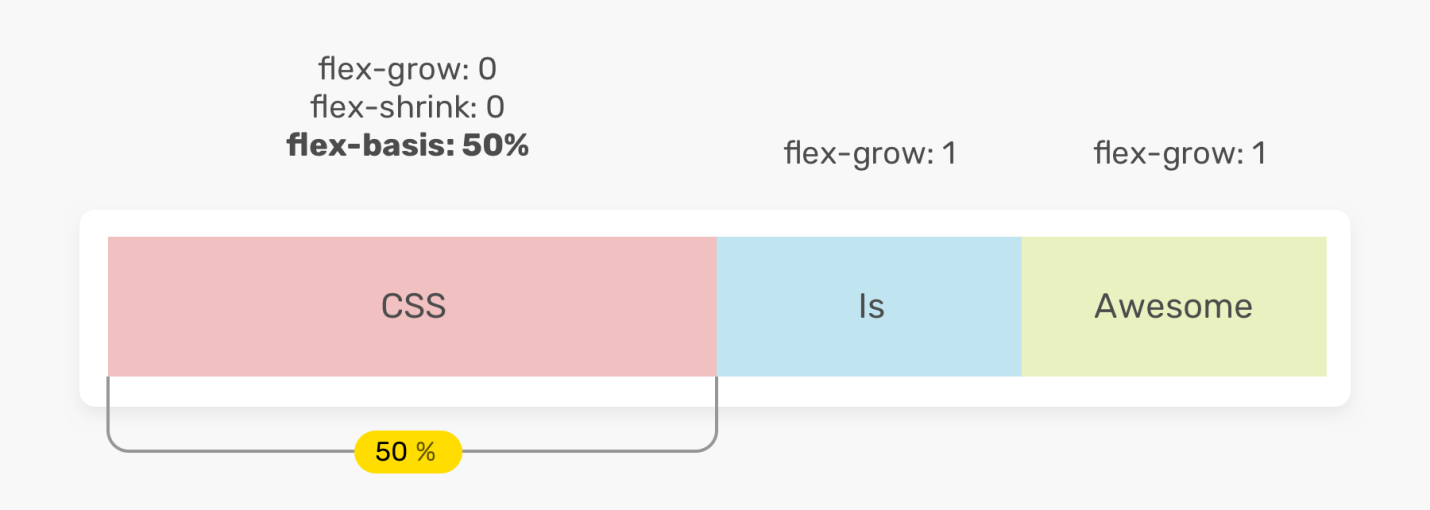
.item-1 {

    flex-grow: 0;

    flex-shrink: 0;

    flex-basis: 50%;

}



*Использование свойства flex-basis: 50%*  
  
Здесь первому элементу назначено свойство flex-basis: 50%. При этом тут важно сбросить в 0 свойство flex-grow, что позволит сделать так, чтобы размер элемента не превысил бы 50%.  
  
Что произойдёт, если вместо этого записать в свойство flex-basis значение 100%? Это приведёт к тому, что элемент займёт 100% ширины родительского элемента, а другие элементы будут перенесены на новую строку.  
  
Вот соответствующие стили:

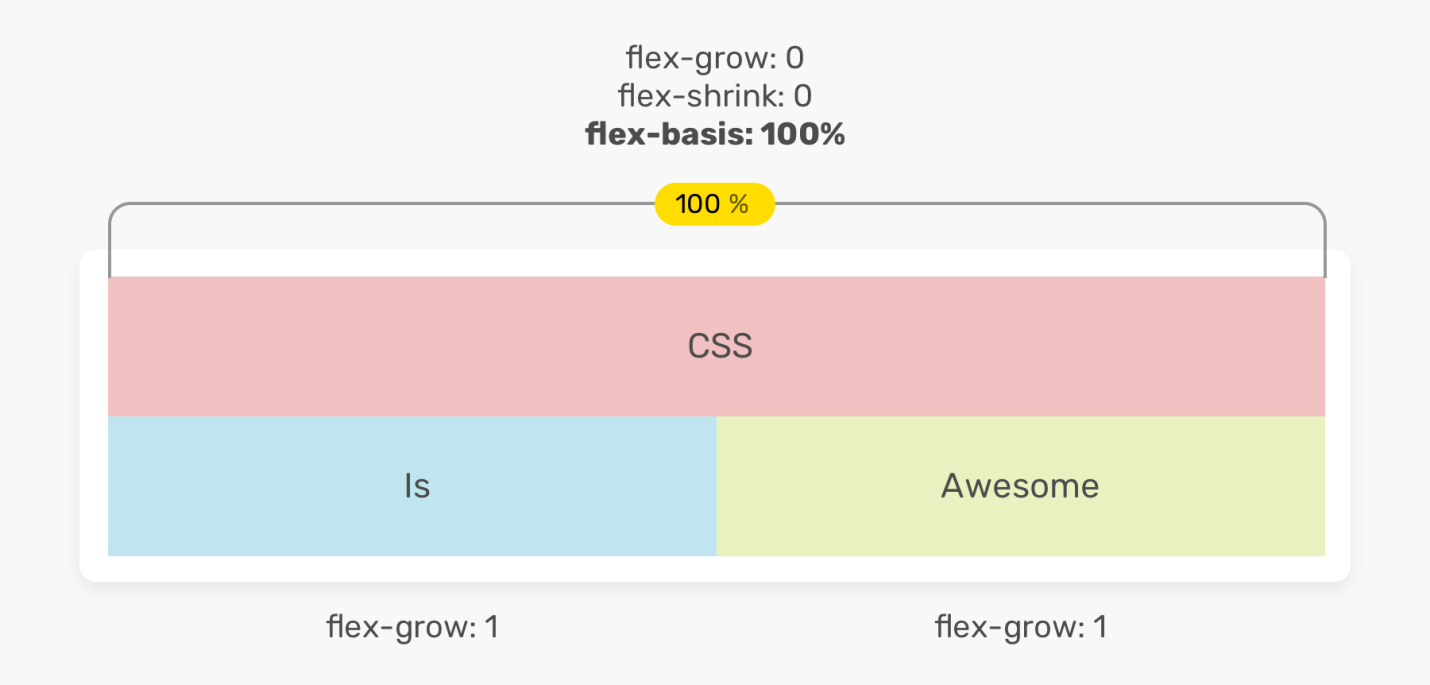
.item-1 {

    flex-grow: 0;

    flex-shrink: 0;

    flex-basis: 100%;

}



*Результат использования свойства flex-basis: 100%;*

Сокращённое свойство flex

Свойство flex позволяет, в сокращённом формате, задавать значения свойств flex-grow, flex-shrink и flex-basis. Значением по умолчанию, которое принимает это свойство, является auto. Оно соответствует flex: 0 1 auto. Это означает, что оно позволяет flex-элементам увеличиваться в размерах, основываясь на размерах их содержимого.  
  
В данном контексте мне хотелось бы обратить ваше внимание на одну важную деталь. Речь идёт о flex-элементах с абсолютными и относительными размерами. И, кстати, это не имеет отношения к [CSS-позиционированию](https://ishadeed.com/article/learn-css-positioning/). Это относится к модели flexbox.

Flex-элементы с относительными размерами

Вот CSS-код:

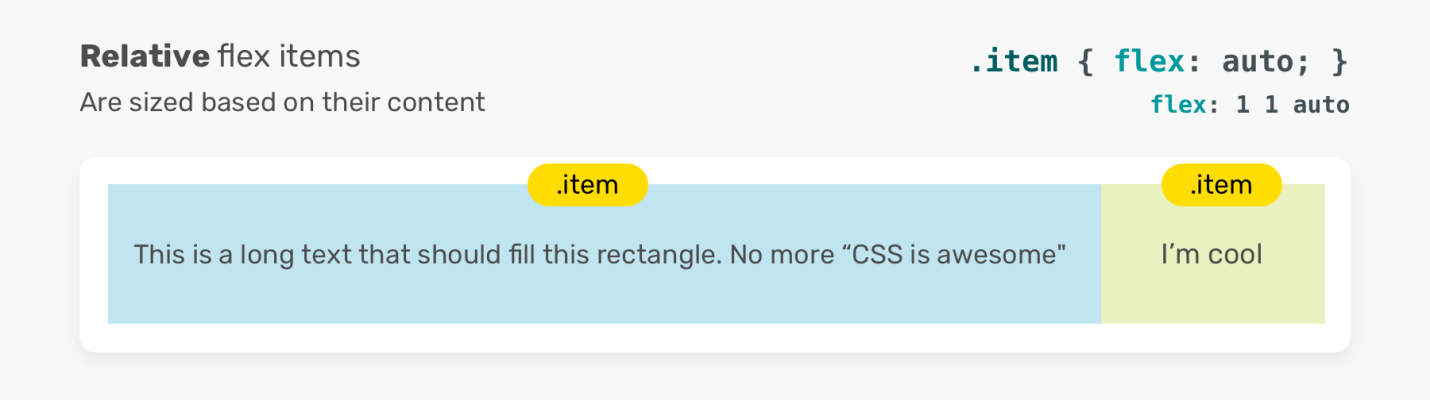
.item {

*/\* Значение, используемое по умолчанию, эквивалентно flex: 1 1 auto \*/*

    flex: auto;

}

Здесь размер flex-элементов основан на их содержимом. В результате элементы, в которых больше содержимого, будут больше.



*Элемент, в котором больше содержимого, будет больше*

Flex-элементы с абсолютными размерами

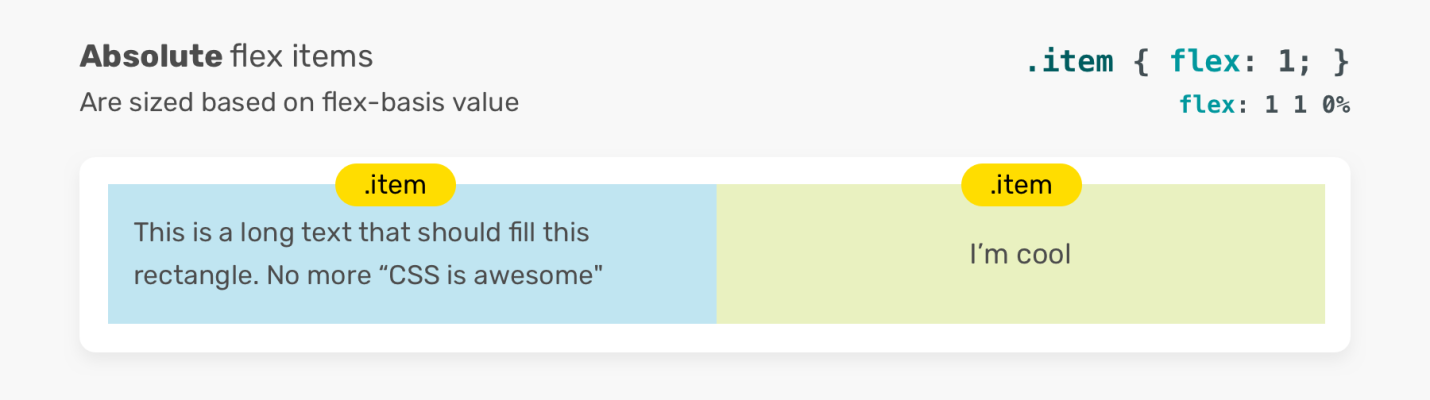
Если же, в отличие от предыдущего примера, свойство flex-basis будет установлено в значение 0, это приведёт к тому, что все flex-элементы увеличатся до одного и того же размера.  
  
Вот стиль:

.item {

*/\* Аналогично flex: 1 1 0% \*/*

    flex: 1;

}



*Элементы имеют одинаковый размер*

Особенности свойства flex, которые мне нравятся

Само название свойства flex намекает на то, что оно умеет гибко работать с передаваемыми ему значениями. Это наделяет его некоторыми особенностями, которые кажутся мне привлекательными. Рассмотрим несколько примеров.

▍Одно значение без единиц измерения

Вот CSS-код:

.item {

    flex: 1;

}

Здесь в свойство flex записывается единственное значение без единиц измерения. Это значение будет воспринято системой как значение свойства flex-grow. Его эквивалент будет выглядеть как flex: 1 1 0.

▍Два значения без единиц измерения

Взглянем на следующий стиль:

.item {

    flex: 1 1;

}

Тут мы в явном виде задаём, соответственно, значения flex-grow и flex-shrink. При этом flex-basis будет сброшено в значение, применяемое по умолчанию.

▍Одно значение, представляющее собой некий размер

Поговорим о том, как будет «расшифрован» следующий стиль:

.item {

    flex: 100px;

*/\* flex: 1 1 100px \*/*

}

Значение 100px будет рассматриваться системой как значение flex-basis. А во flex-grow и flex-shrink будет, по умолчанию, записано 1.

▍Использование значения 0 без указания единиц измерения

Предположим, нужно установить flex-basis в значение 0, воспользовавшись свойством flex. Мы, для решения этой задачи, решим поступить так:

.item {

    flex: 0;

}

Делать так не рекомендуется, так как это способно запутать и тех, кто будет читать подобный код, и браузер. Как понять — к чему именно относится этот 0? К свойству flex-grow, flex-shrink или flex-basis? В общем — путаница получается. Вот что говорится об этом в [спецификации CSS](https://drafts.csswg.org/css-flexbox-1/#flex-property):  
  
*Ноль без указания единиц измерения, перед которым нет двух значений коэффициентов гибкости, должен интерпретироваться как значение коэффициента гибкости. Для того чтобы избежать неправильной интерпретации или ошибочного объявления свойства, авторы должны, задавая нулевое значение flex-basis, использовать единицы измерения, или ставить перед этим значением два значения, относящиеся к коэффициентам гибкости.*  
  
То есть, чтобы избежать неоднозначностей, вышеприведённый код стоит переписать так:

.item {

    flex: 0%;

*/\* flex: 1 1 0% \*/*

}

Здесь используется конструкция 0%, но можно, например, воспользоваться и конструкцией 0px.

Пользуйтесь сокращённым свойством flex

Когда вам надо задать свойства flex-grow, flex-shrink и flex-basis, лучше всего использовать для этого сокращённое свойство flex.  
  
Заглянем в [спецификацию CSS](https://drafts.csswg.org/css-flexbox-1/#flex-components):  
  
*Авторам рекомендуется управлять гибкими элементами, используя сокращённое свойство flex, а не применяя напрямую полные имена свойств, так как при применении сокращённого свойства производится корректный сброс неуказанных значений к состоянию, соответствующему распространённым способам их использования.*

Сценарии использования и практические примеры

▍Аватары пользователей



*Аватар, при настройке которого использовано свойство flex*  
  
Модель flexbox часто используется для оформления компонентов страниц, имеющих отношение к пользователям. Например, это касается аватара пользователя и соответствующей подписи, которые должны находиться на одной строке.  
  
Вот разметка:

<div class="user">

    <img class="user\_\_avatar" src="shadeed.jpg" alt="" />

    <div>

        <h3>Ahmad Shadeed</h3>

        <p>Author of Debugging CSS</p>

    </div>

</div>

Вот стиль:

.user {

    display: flex;

    flex-wrap: wrap;

    align-items: center;

}

.user\_\_avatar {

    flex: 0 0 70px;

    width: 70px;

    height: 70px;

}

Обратите внимание на то, что, настраивая элемент user\_\_avatar, я применил свойство flex: 0 0 70px. Это важно, так как без этой настройки в некоторых устаревших браузерах изображение может вывестись неправильно. Более того, приоритет свойства flex выше приоритета свойства width (если речь идёт о ситуации, в которой применяется flex-direction: row) и свойства height (в ситуации, когда применяется flex-direction: column).  
  
Если изменить размер аватара, воздействуя только на свойство flex, браузер проигнорирует то, что задано в свойстве width. Вот соответствующий стиль:

.user\_\_avatar {

*/\* Ширина будет 100px, а не 70px \*/*

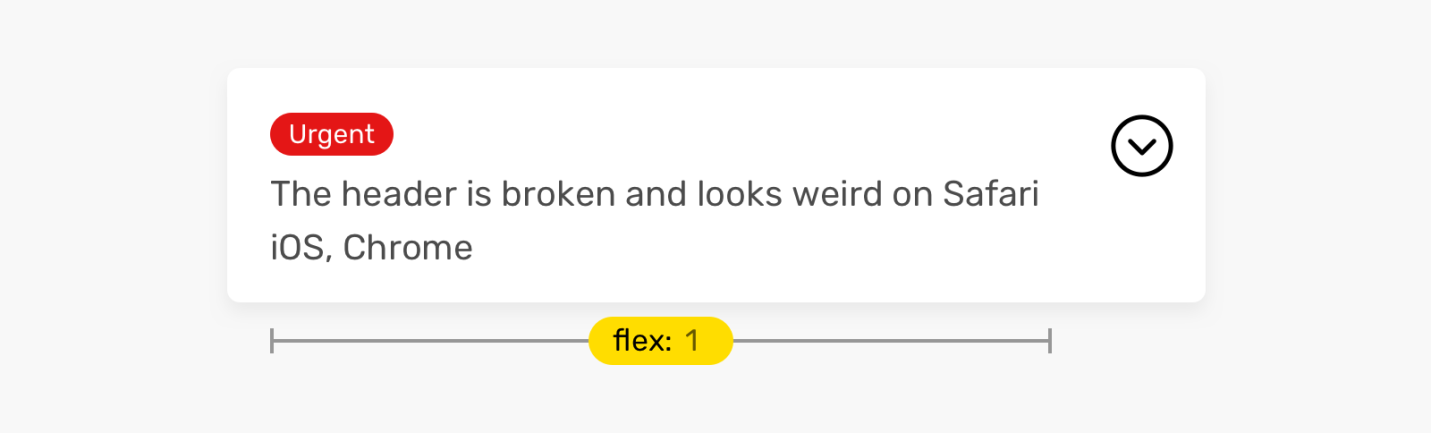
    flex: 0 0 100px;

    width: 70px;

    height: 70px;

}

▍Заголовок элемента



*Заголовок элемента*  
  
Предположим, нам надо оформить заголовок некоего элемента. Этот заголовок должен занять всё доступное ему пространство. Решить эту задачу можно с помощью свойства flex: 1:

.page-header {

    display: flex;

    flex-wrap: wrap;

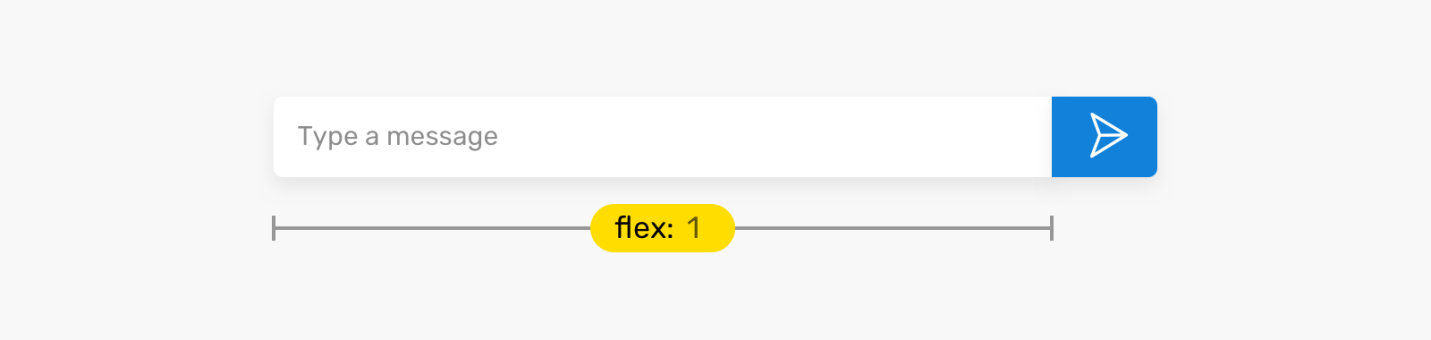
}

.page-header\_\_title {

    flex: 1;

}

▍Поле ввода для подготовки сообщения



*Поле ввода*  
  
Подобные поля часто встречаются в приложениях для обмена сообщениями, вроде тех, что есть в Facebook и Twitter. Такое поле должно занимать всё доступное ему пространство. При этом кнопка для отправки сообщения должна иметь фиксированную ширину. Вот как это выглядит в CSS:

form {

    display: flex;

    flex-wrap: wrap;

}

input {

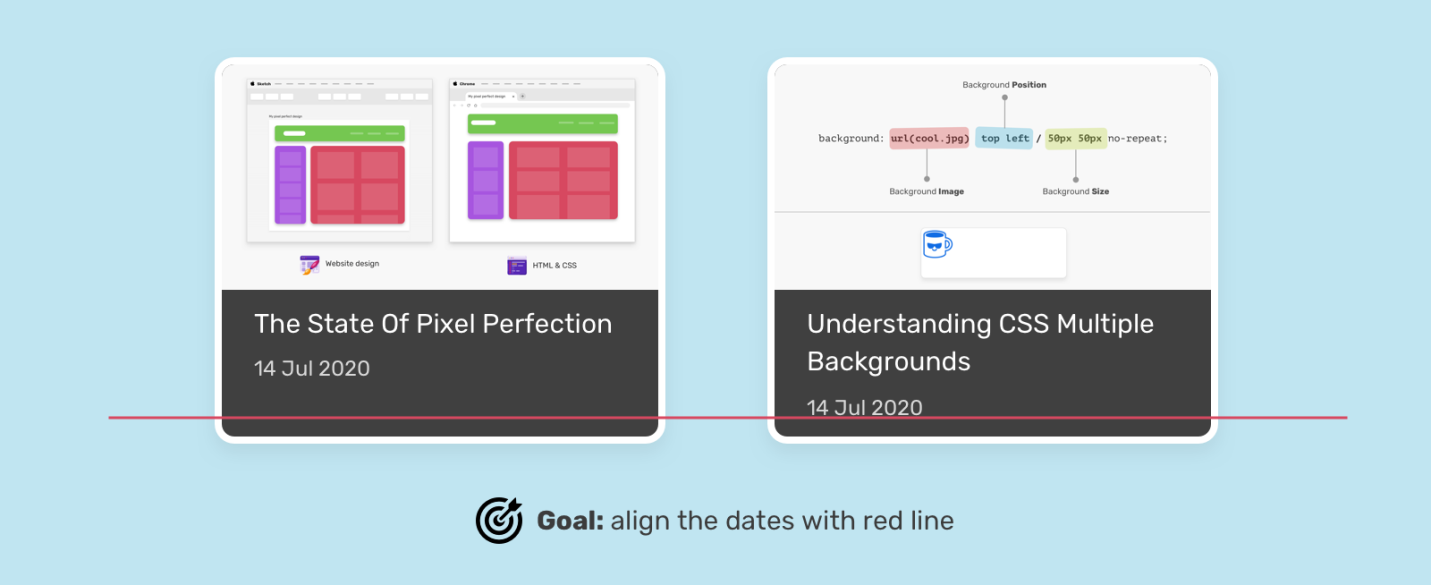
    flex: 1;

*/\* Другие стили \*/*

}

Этот пример являет собой удачную демонстрацию использования свойства flex-grow.  
  
Надо отметить, что некоторым способам использования свойства flex не уделяется достаточно внимания в публикациях, посвящённых этому свойству. Давайте это исправим.

▍Выравнивание нижних элементов, находящихся на разных карточках



*Цель — выровнять обе даты по красной линии*  
  
Предположим, что перед нами — макет, сформированный средствами CSS Grid, в котором имеются две колонки. Проблема тут в том, что даты, выводимые в конце текстового содержимого карточек, не выровнены. А нам нужно, чтобы они находились бы на одной линии, показанной на предыдущем рисунке.  
  
Для решения этой задачи можно воспользоваться моделью flexbox.  
  
Вот разметка:

<div class="card">

    <img src="thumb.jpg" alt="">

    <h3 class="card\_\_title">Title short</h3>

    <time class="card\_\_date"></time>

</div>

Достичь нашей цели можно, воспользовавшись свойством flex-direction: column. Элемент card\_\_title, заголовок, можно стилизовать, применив свойство flex-grow и сделав так, чтобы он занимал бы всё доступное ему пространство. В результате окажется, что строка с датой попадёт в нижнюю часть карточки. Так произойдёт даже в том случае, если сам заголовок окажется достаточно коротким.  
  
Вот стили:

.card {

    display: flex;

    flex-direction: column;

}

*/\* Первое решение \*/*

.card\_\_title {

    flex-grow: 1;

}

Решить эту задачу можно и не прибегая к свойству flex-grow. А именно, речь идёт о применении модели flexbox и механизма автоматической настройки внешних отступов с использованием ключевого слова auto. Я, кстати, написал об этом [подробную статью](https://habr.com/ru/company/ruvds/blog/494716/).

*/\* Второе решение \*/*

.card\_\_date {

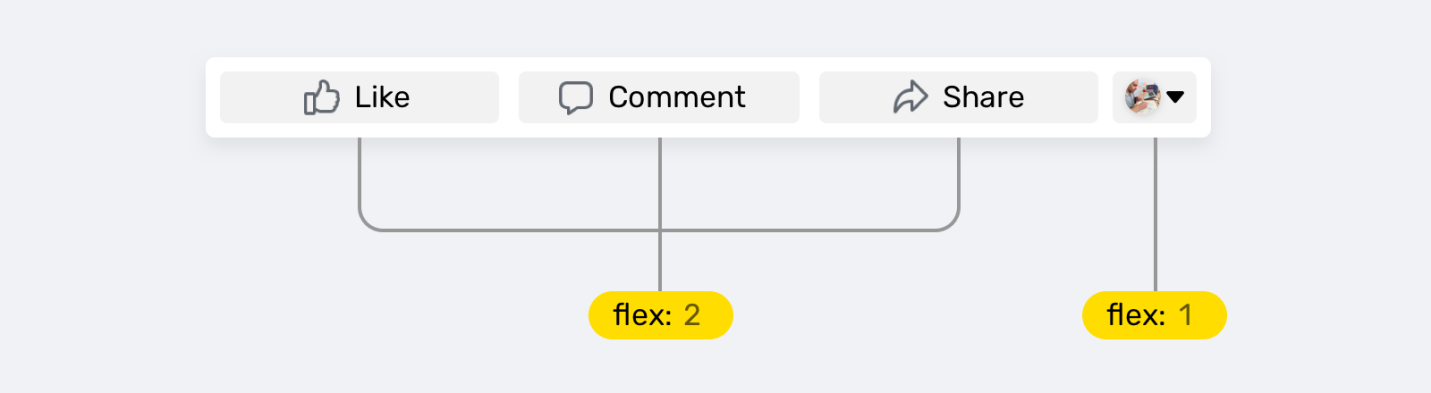
    margin-top: auto;

}

Использование разных значений для коэффициентов гибкости

В этом разделе я хочу рассказать об использовании свойств flex-grow и flex-shrink, которым назначают значения, отличные от 1. Сейчас я приведу несколько практических примеров использования подобной схемы настройки этих свойств.

▍Панели управления



*Набор элементов управления*  
  
На подготовку этого примера меня вдохновил дизайн Facebook (значки я тоже взял оттуда). На панели управления (элемент actions в следующем коде) имеется четыре элемента. Последний (элемент actions\_\_item.user) имеет ширину, которая меньше, чем ширина других элементов. Сформировать подобную панель управления можно так:

.actions {

    display: flex;

    flex-wrap: wrap;

}

.actions\_\_item {

    flex: 2;

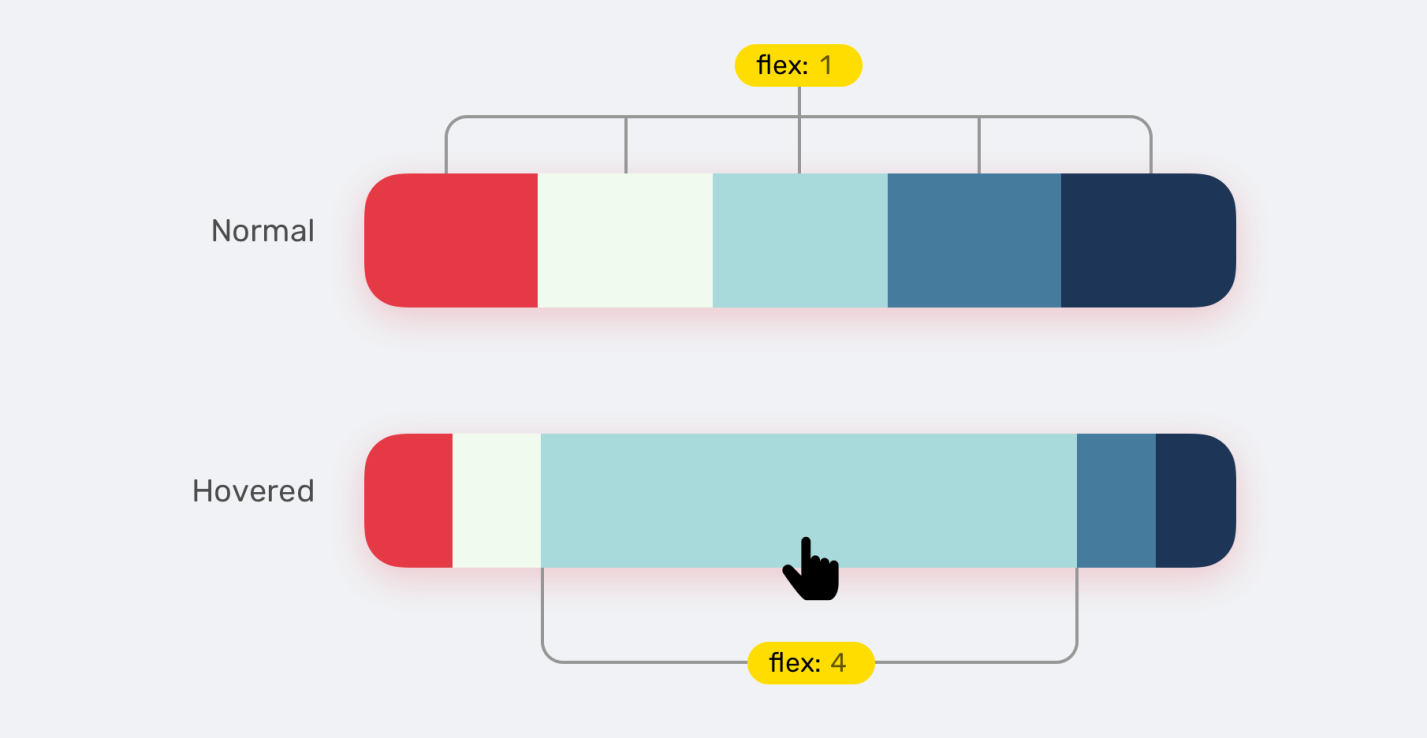
}

.actions\_\_item.user {

    flex: 1;

}

▍Анимация, в ходе которой элемент растягивается



*Элемент растягивается при наведении на него указателя мыши*  
  
Flex-элементы можно анимировать при наведении на них указателя мыши. Этот приём может оказаться крайне полезным. Вот соответствующий CSS-код:

.palette {

    display: flex;

    flex-wrap: wrap;

}

.palette\_\_item {

    flex: 1;

    transition: flex 0.3s ease-out;

}

.palette\_\_item:hover {

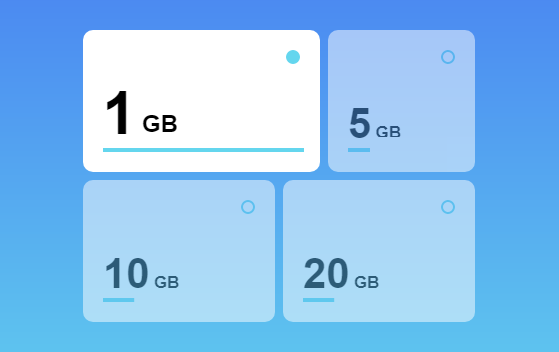
    flex: 4;

}

[Здесь](https://ishadeed.com/assets/flex-css/expand-animation.mp4) можно найти видео, демонстрирующее всё это в действии.

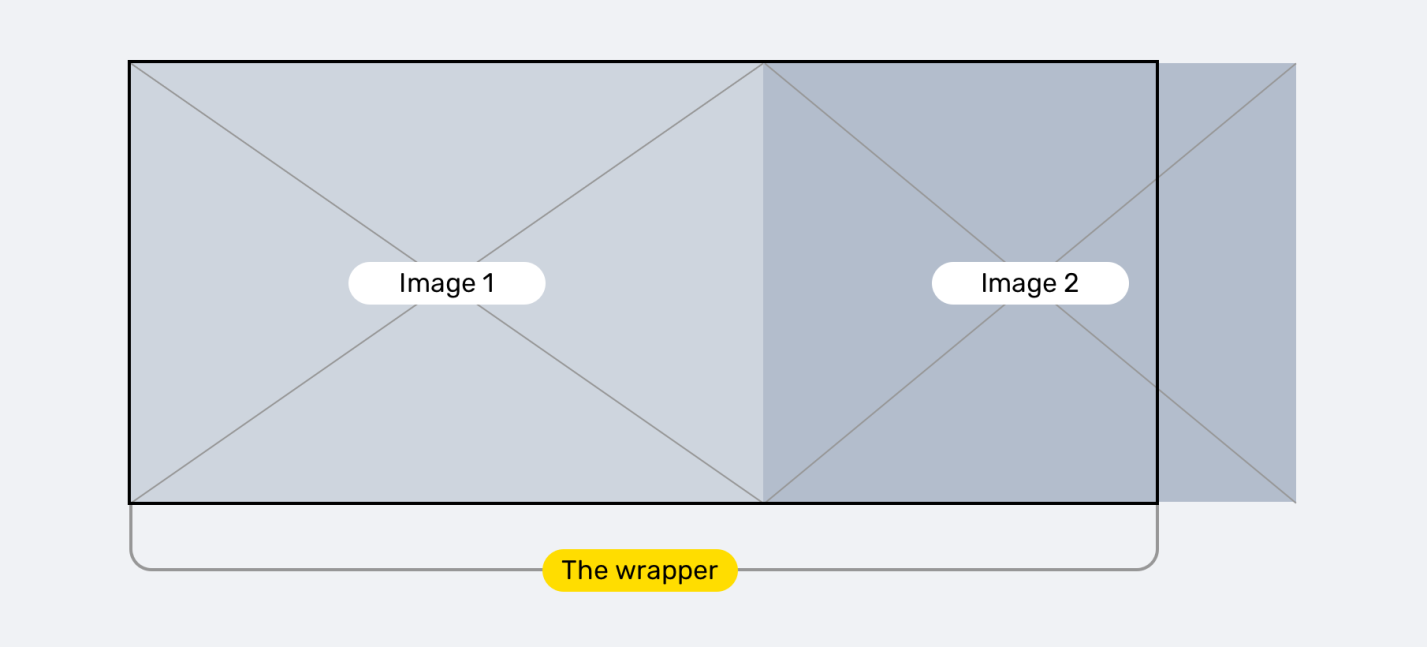
▍Увеличение размера активной карточки

Мне очень нравится [этот](https://codepen.io/shshaw/pen/EbjvbQ) проект на CodePen, в котором реализован механизм увеличения размеров карточки с описанием тарифного плана, по которой щёлкнул пользователь. Размер карточки меняется благодаря установке её свойства flex-grow в значение 4.



*Увеличение размера активной карточки*

▍Что делать, если содержимое контейнера больше самого контейнера?



*Элементы не помещаются в контейнер*  
  
Некоторое время тому назад мне пришло письмо от одного из моих читателей, который рассказал мне о возникшей у него проблеме и попросил помочь с её решением. Суть этой проблемы представлена на предыдущем рисунке. Здесь есть пара изображений, которые должны находиться в пределах элемента-контейнера, но размер этих изображений больше размеров контейнера.  
  
Попробуем такой стиль:

.wrapper {

    display: flex;

}

.wrapper img {

    flex: 1;

}

Но даже при использовании свойства flex: 1 изображения в контейнер не помещаются. Обратимся к [спецификации CSS](https://www.w3.org/TR/css-flexbox-1/):  
  
*Flex-элементы, по умолчанию, не уменьшаются в размерах до величин, меньших, чем размеры их содержимого (длина самого длинного слова или элемента с фиксированным размером). Для того чтобы это изменить, нужно воспользоваться свойствами min-width или min-height.*  
  
В нашем случае проблема заключается в том, что изображения слишком велики и модель flexbox не уменьшает их до таких размеров, чтобы они поместились бы в контейнер. Исправить это можно так:

.wrapper img {

    flex: 1;

    min-width: 0;

}

Подробности смотрите в [этой](https://ishadeed.com/article/min-max-css/#setting-min-width-to-zero-with-flexbox) моей статье.

Итоги

Здесь я рассказал вам о свойстве flex и о том, как им пользоваться. Надеюсь, вы узнали из этого материала что-то новое и полезное.