

# Вёрстка на Flexbox в CSS. Полный справочник




Stas Bagretsov · [Follow](#)

13 min read · Feb 6, 2018



Share



🎉 Подписывайтесь на мой Twitter!  [@stassonmars](#) — теперь там ещё больше из мира фронтенда, да и вообще поговорим. 📢 Подписывайтесь, скоро будет много нового и ещё больше интересного ✨



**update 25.07.19** Долго и постепенно, на протяжении года находил и переводил статьи по больным темам при работе с CSS Flexbox и решил обновить статью, исправив и дополнив некоторые моменты, добавив подробные разъяснения на каждый момент. К тому же добавил подробные объяснения некоторых методик при использовании Flexbox



# Вёрстка на Flexbox в CSS

## Полный справочник

+ подробные описания сложных свойств

+ описания распространенных подходов

В этой статье вы узнаете буквально все свойства модуля Flexbox и даже больше — вы сможете посмотреть на примерах, как высчитываются и работают самые сложные из них, конечно же, параллельно увидев полное объяснение каждого. Эта статья является переводом самой последней версии статьи — [A Complete Guide to Flexbox](#). В интернете конечно есть другие переводы, но они либо для старых версий, либо в некоторых моментах, с неправильным переводом, а есть и частичные переводы, вырванные из контекста всего руководства.

В этой статье всё будет собрано в одно целое, а также, я перевел статьи отдельно для свойств, которые вызывают больше всего вопросов у других фронт-энд разработчиков. По сути в этой статье и её ответвлениях, собран полный справочник-руководство по работе с flexbox и его свойствами. Приятного чтения.

### Введение

Flexbox разметка в CSS даёт один из наиболее эффективных способов расстановки, выравнивания и распределения места между элементами внутри контейнера, даже если их размер неизвестен или динамичен (собственно, по этому его и называют flex, от слова flexible, что по-английски имеет двойственное значение — гибкий и уступчивый, что очень сочетается с моделью поведения flexbox).

Основной целью flexbox является предоставление возможности изменения своих элементов по ширине и высоте, для того, чтобы они максимально эффективно умещались в доступном месте родительского контейнера, в частности — это удобно в тех случаях, когда нужно соответствовать всем типам дисплеев устройств и размерам экранов. Flex контейнер расширяет вложенные элементы для того, чтобы заполнить доступное пространство или же урезает их, чтобы избежать переполнения.

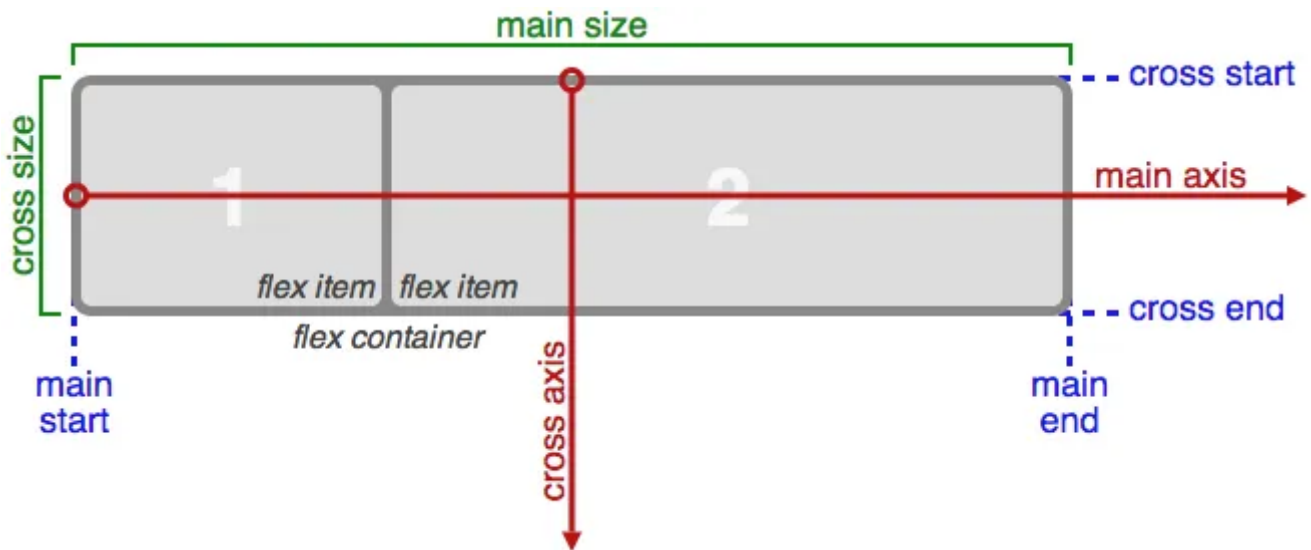
*Очень важный момент*, flexbox независим от направления, в отличие от обычных шаблонов, блочная модель основана на вертикальном размещении элементов, а инлайновая на горизонтальной. Пока это достаточно хорошо работало на простых страницах (и сайтах), эти недостатки в гибкости довольно сильно влияли на поддержку больших и сложных приложений, особенно когда дело касалось смены ориентирования, ресайзинга, растягивания или наоборот сокращения элементов.

Учтите, что Flexbox лучше всего подходит для компонентов в приложении и для не масштабных шаблонов. Существует Grid модель разметки, которая предназначена для работы с большими шаблонами и комплексными приложениями. Именно этот момент очень хорошо и доступно расписан в статьях — Решающая CSS битва: Grid против Flexbox и Когда нужно использовать Flexbox

## **Основы и терминология**

Так как flexbox является полноценным модулем, а не простым свойством, он включает в себя множество интересного, а в частности полный набор рабочих свойств. Некоторые из них созданы для использования на контейнере (родительский элемент, известный как flex-container), в то время как другие должны помогать исполнять свои роли дочерним элементам.

Если обычный шаблон основан как на блочных, так и на инлайновых элементах, формирующих поток, то flex-шаблон основан на нескольких потоках. Посмотрите на это изображение, наглядно показывающее основную идею flex шаблонизации.



Элементы в flexbox могут располагаться вдоль основной оси (от **main-start** до **main-end**) или же вдоль поперечной оси (от **cross-start** до **cross-end**).

*Main axis* — основная ось flex контейнера, вдоль которой располагаются flex элементы. Обратите внимание, что ей необязательно быть горизонтальной, все зависит от свойства `flex-direction`, о котором вы прочитаете позже.

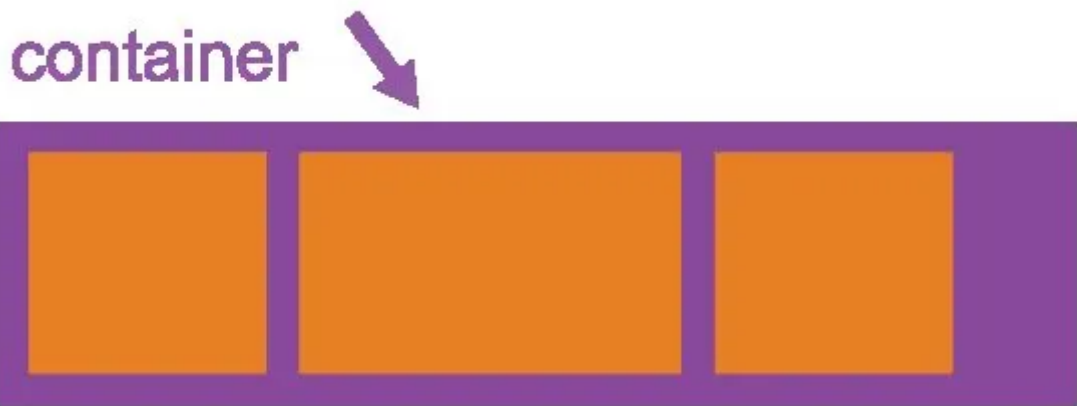
*Main-start | main-end* — Flex элементы расположены в рамках контейнера, начиная от **main-start** и заканчивая **main-end**.

*Main-size* — высота или ширина flex элемента, что зависит от того, в каком направлении идёт основная ось.

*Cross axis* — ось перпендикулярная главной оси, называется поперечной. Её направление зависит от направления главной оси.

*Cross-start | cross-end* — flex линии, заполненные элементами и расположенные в контейнере от **cross-start** до **cross-end**.

*Cross-size* — Ширина или высота flex-элемента, в зависимости от направления главной оси.



## Свойства родительских элементов (flex-container)

### #display

Это свойство создаёт сам flex контейнер, инлайновый или блочный, в зависимости от заданного значения. Также оно задает flex-контекст каждому прямому потомку.

```
.container {  
  display: flex; /* или inline-flex */  
}
```

Учтите, что CSS колонки не имеют эффекта на flex контейнер.

Но это ещё не всё. Есть свойство `inline-flex`, которое требует отдельного внимания и кейсов применения. В статье [Мистический inline-flex и что он делает](#) вы узнаете всё об этом интересном свойстве и даже немного больше.

### #flex-direction



Устанавливает направление главной оси и определяет направление flex элементов размещенных в flex контейнере. Flexbox это односторонняя

концепция представления шаблонизации. Поэтому flex элементы располагаются, в основном, вдоль горизонтальной или поперечной линии.

Когда вы немного поймете работу flexbox, то советую прочитать статью — [Плиточная раскладка на CSS Flexbox с помощью order и :nth-child\(\)](#). В ней вы узнаете о тонкостях работы с flex-direction и научитесь делать интересные фишки с CSS Flexbox.

```
.container {  
  flex-direction: row | row-reverse | column | column-reverse;  
}
```

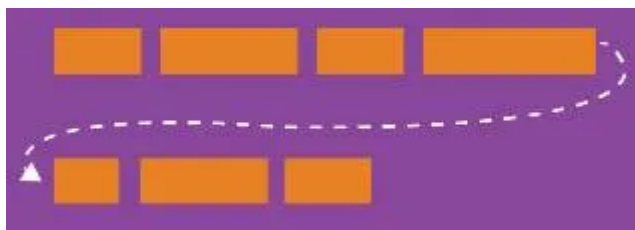
row (стандартное положение) — слева направо.

row-reverse — элементы располагаются справа налево.

column — тоже самое, что и row, только сверху вниз.

column-reverse — тоже самое, что и row-reverse, но снизу вверх.

### #flex-wrap



```
.container{  
  flex-wrap: nowrap | wrap | wrap-reverse;  
}
```

Изначально все flex-элементы будут пытаться уместиться в одну строку. Вы можете поменять это и дать возможность этим элементам переходить на другую строку, при необходимости.

nowrap — это значение по-дефолту, при котором все flex элементы будут

выстраиваться в одну линию.

`wrap` — flex элементы будут переноситься на несколько строк, от верха к низу.

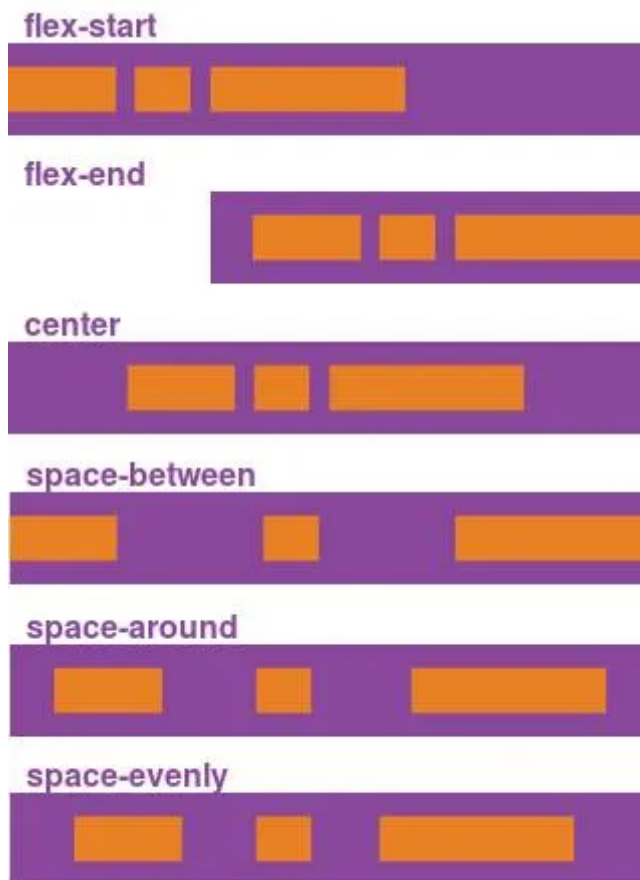
`wrap-reverse` — flex элементы будут переноситься на несколько строк снизу вверх.

### #flex-flow (применяется для родительского flex контейнера)

Это сокращение `flex-direction` и `flex-wrap` свойств, которые вместе определяют направление главной и поперечной оси. По-дефолту оно имеет значение `row nowrap`.

```
flex-flow: <'flex-direction'> || <'flex-wrap'>
```

### #justify-content



Свойство определяет выравнивание вдоль главной оси. Оно помогает распределить лишнее свободное пространство, когда, либо все flex элементы в линии имеют фиксированный размер, либо же нет, но уже достигли своего максимального размера. Оно также влияет на выравнивание элементов, когда те переполняют строку.

```
.container {  
  justify-content: flex-start | flex-end | center | space-between |  
space-around | space-evenly;  
}
```

`flex-start` — дефолтное состояние, при котором элементы расставляются от начала строки.

`flex-end` — состояние, в котором элементы размещены с конца строки.

`center` — элементы центрированы вдоль строки.

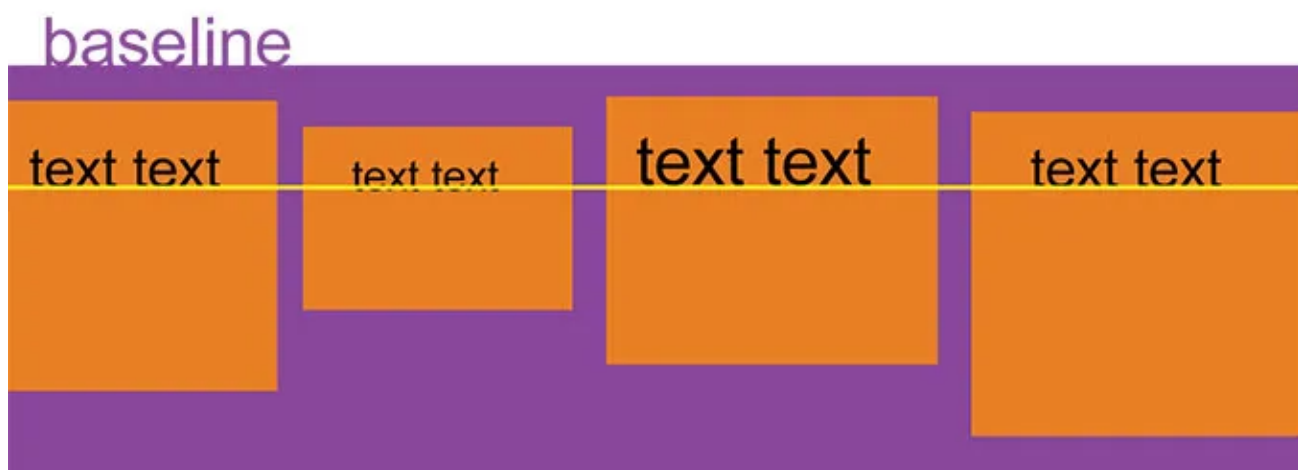
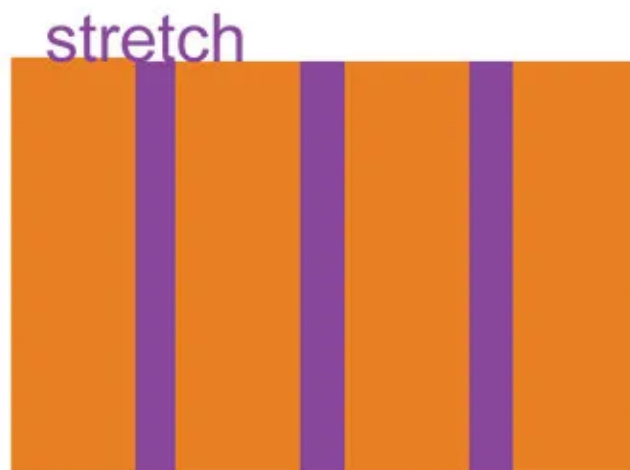
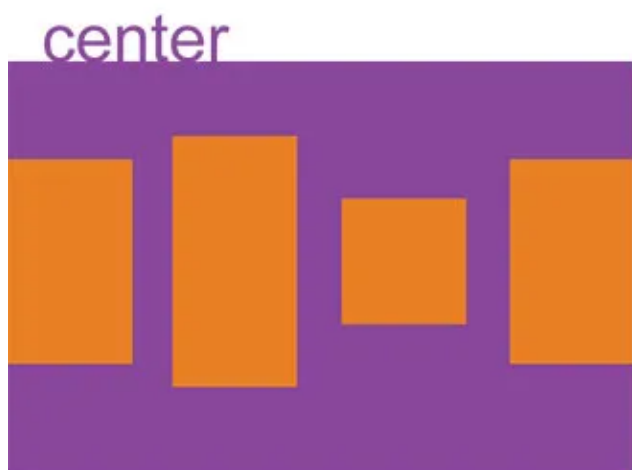
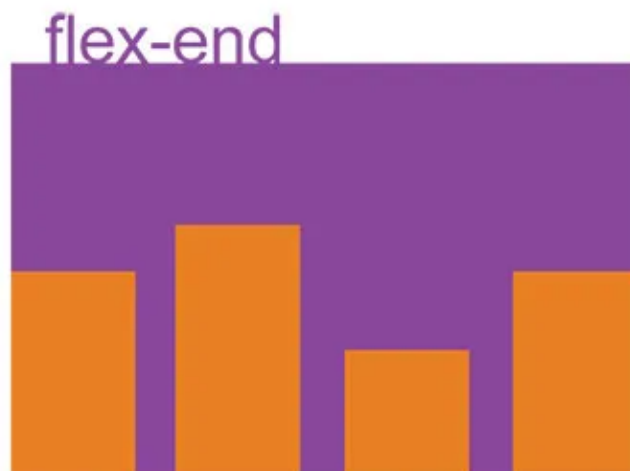
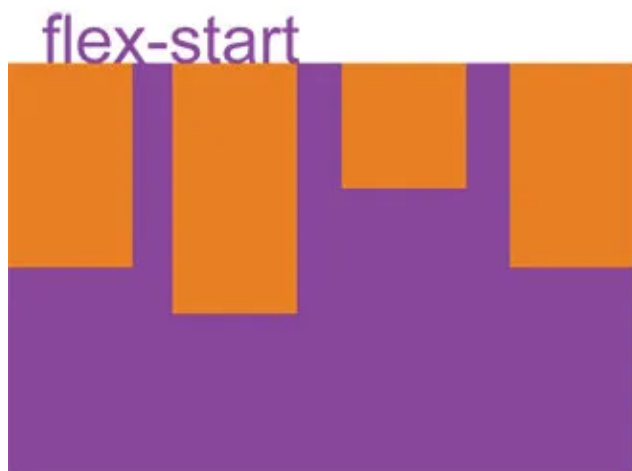
`space-between` — элементы равномерно распределены по строке, первый элемент находится в начале строки, последний в конце.

`space-around` — элементы равномерно распределены по строке с равным местом вокруг них. Учтите, что визуально пробелы не равномерны, так как все элементы имеют одинаковые пробелы с двух сторон. Первый элемент получит одну единицу свободного места от границы контейнера, но получит две единицы свободного места от следующего элемента, так как у него тоже есть одна единица свободного места с каждой из сторон.

`space-evenly` — элементы распределены таким образом, что свободное пространство между любыми двумя элементами равномерно, как и место до границы края контейнера.

## **#align-items**





Это свойство определяет стандартное поведение того, как flex элементы будут располагаться вдоль поперечной оси на заданной строке. Это своеобразная версия `justify-content`, но только для поперечной оси, которая перпендикулярна главной.

```
.container {  
  align-items: stretch | flex-start | flex-end | center | baseline;
```

}

`flex-start` — всё размещается с начала поперечной оси

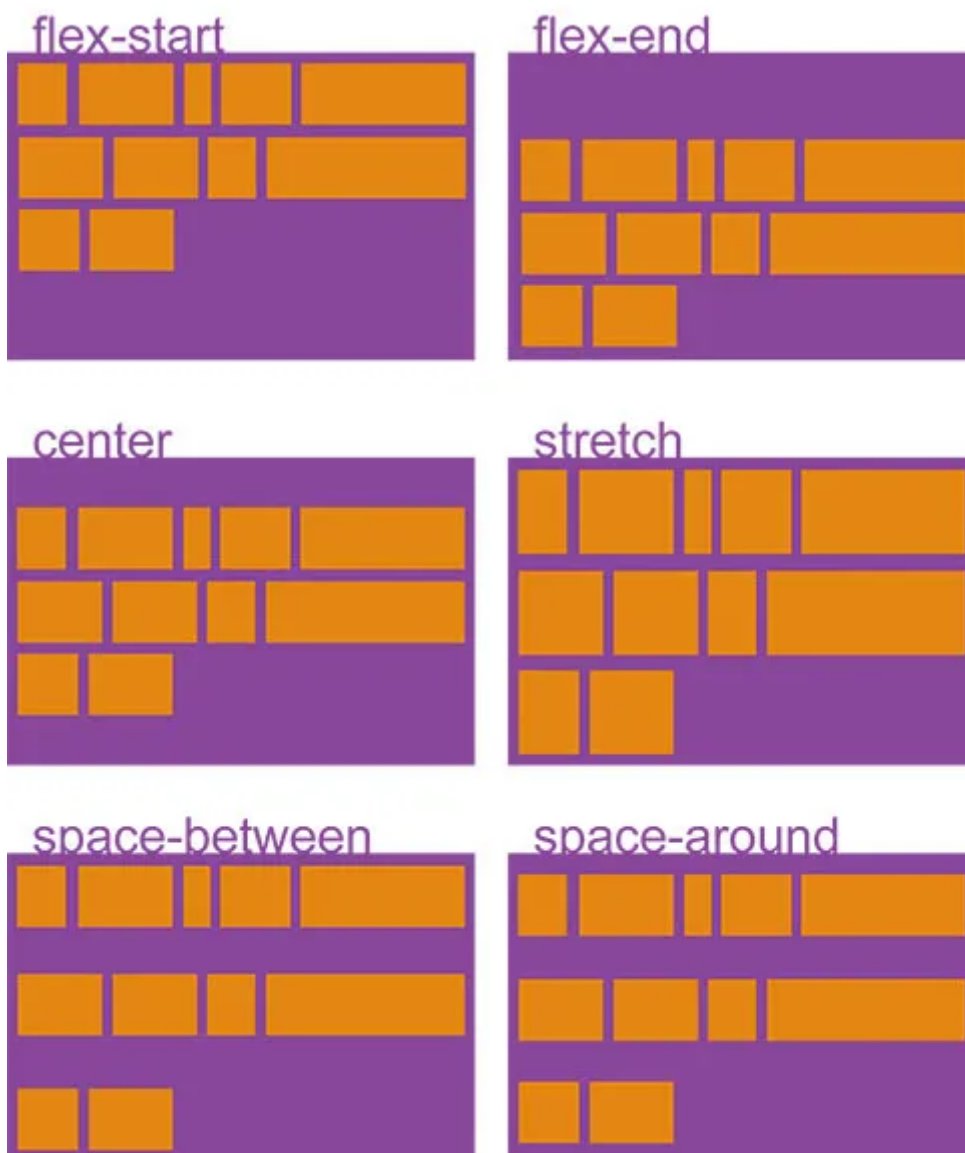
`flex-end` — все элементы размещаются с конца поперечной оси

`center` — элементы центрируются по поперечной оси

`baseline` — элементы выравниваются по базовой линии

`stretch` — это дефолтное состояние, при котором элементы заполняют контейнер, с учетом `min-width` и `max-width`.

### **#align-content**



Это свойство выравнивает и распределяет строки контейнера, когда есть свободное пространство в поперечной оси, подобно тому как `justify-content`, оно выравнивает элементы по главной оси.

Учтите, что это свойство не приносит эффекта, когда есть только одна строка flex элементов.

```
.container {  
  align-content: flex-start | flex-end | center | space-between |  
space-around | stretch;  
}
```

`flex-start` — строки расположены от начала контейнера.

`flex-end` — строки расположены от конца контейнера.

`center` — строки расположены от центра контейнера.

`space-between` — строки равномерно распределены, первая строка находится в начале контейнера, тогда как последняя находится в конце.

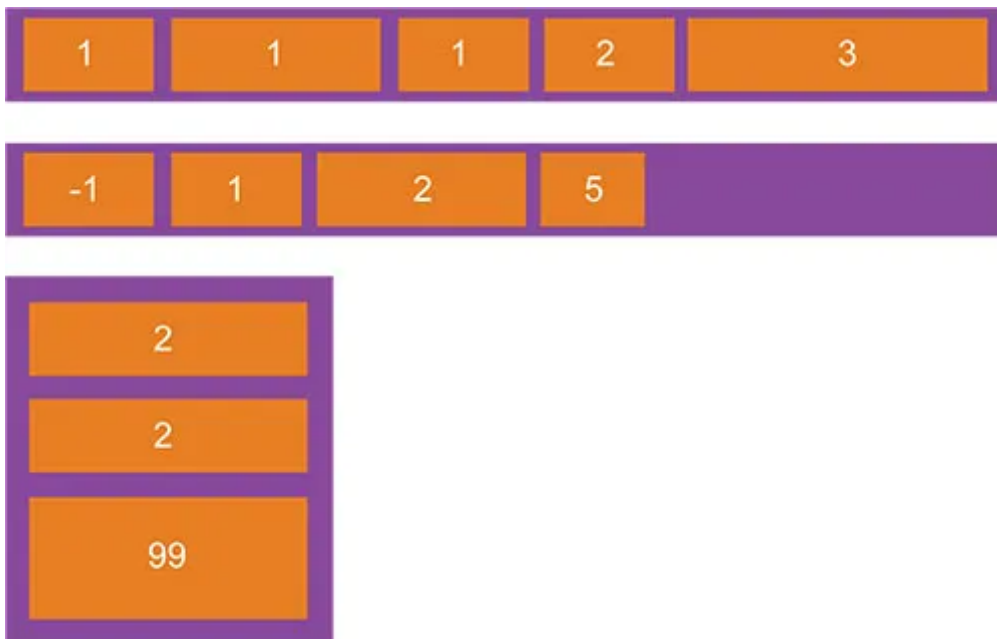
`space-around` — строки равномерно распределены с равным местом вокруг каждой строки.

`stretch` — стандартное состояние, при котором строки растягиваются на вся оставшееся место.

После того, как вы немного поймете то, как работают эти свойства направления потоков, очень советую прочитать статью — [Вертикальное и горизонтальное центрирование всего и вся в CSS Flexbox](#). Тут вы поймете как делать центрирование с этими двумя свойствами, а ещё увидите рабочие кейсы с CSS Flexbox, которые вы будете часто видеть при вёрстке.

## Свойства дочерних элементов

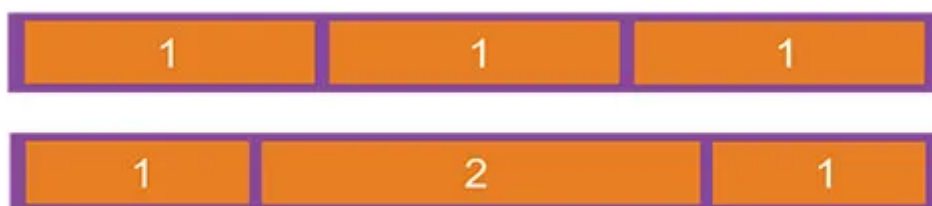
### #Order



По-дефолту, флекс элементы располагаются в исходном порядке 1, 2, 3 и т.д. Однако, свойство `order` контролирует порядок в котором элементы могут располагаться.

```
.item {  
  order: <любое целое число>; /* дефолтное 0 */  
}
```

### #flex-grow



Это свойство определяет способность флекс элемента при необходимости становится больше. Оно принимает безразмерное значение, которое служит пропорцией или долей. Оно указывает какое количество свободного места внутри контейнера элемент должен взять.

Если все элементы в контейнере имеют `flex-grow` со значением 1, то это означает то, что оставшееся место в нём распределено в равной мере среди потомков. Также, помните, что оно не принимает отрицательных значений.

Это свойство само по себе довольно сложное, поэтому я перевел отдельную статью по ней, где подробно описывается как работает flex-grow в CSS.

```
.item {  
  flex-grow: <число>; /* дефолтное 0 */  
}
```

### #flex-shrink

Это свойство определяет способность flex элемента сокращаться при необходимости. Оно также не принимает отрицательных значений. Вообще тоже очень интересное свойство, с которым у многих бывают проблемы при реализации.

```
.item {  
  flex-shrink: <число>; /* дефолтное 1 */  
}
```

Это свойство ещё сложнее, чем flex-grow и требует отдельного внимания. В этой статье вы узнаете о тонкостях его работы, как оно высчитывается и как работает с дробными значениями — Как работает flex-shrink в CSS. Подробное руководство.

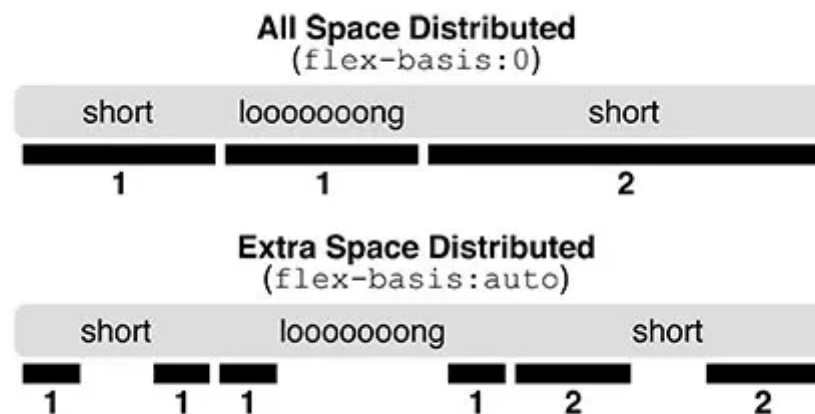
### #flex-basis

Это свойство определяет стандартный размер элемента, перед тем как оставшееся место будет распределено. Это может быть длина ( 20%, 5rem и тп) или ключевое значение. Ключевое значение auto означает «эй, посмотрите на мои свойства ширины и высоты» (этим раньше занималось свойство main-size,

пока не устарело и не было удалено). Значение `content` означает, что размер основывается на контенте элемента — это свойство пока что не очень хорошо поддерживается, поэтому его очень тяжело протестировать и довольно сложно понять, что делают его собратья, такие как `max-content`, `min-content` и `fit-content`.

```
.item {  
  flex-basis: <длина> | auto; /* дефолтное auto */  
}
```

Если выставить значение на `0`, то дополнительное место вокруг контента не будет учтено. Если выставить на `auto`, то дополнительное свободное место будет распространяться, основываясь на его `flex-grow` значении. В общем, посмотрите график ниже.



Очень советую и строго рекомендую к прочтению статью — [Разница между width и flex-basis](#). В ней вы узнаете важную разницу между `width` и `flex-basis`, а также поймете как что работает при разных условиях выставления размеров.

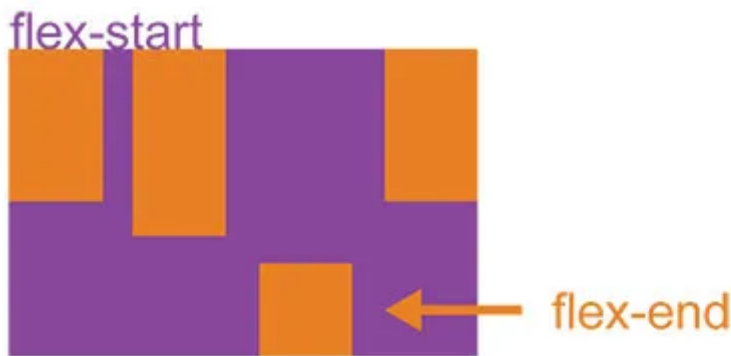
## #flex

Это сокращение для `flex-grow`, `flex-shrink` и `flex-basis` — все вместе взятые. Второй и третий параметры опциональны, то есть `flex-shrink` и `flex-basis`. По-дефолту оно имеет значение `0 1 auto`.

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

Рекомендуется использовать сокращенное свойство, вместо набора индивидуальных свойств. Оно выставляет другие значения должным образом, вернее грамотно.

### #align-self



Это свойство позволяет стандартному выравниванию (или если точно `align-items`) элементов, быть перезаписанному для определенного flex элемента. Чтобы увидеть все значения `align-self`, посмотрите их в `align-items`.

```
.item {
  align-self: auto | flex-start | flex-end | center | baseline |
stretch;
}
```

Примите во внимание, что `float`, `clear` и `vertical-align` не имеют эффекта на flex элементы.

### Примеры

Давайте начнем с очень очень простого примера, решающего *ежедневную* проблему — идеальное центрирование.

```
.parent {
  display: flex;
  height: 300px; /* Неважно */
}
```

```
.child {  
  width: 100px;  /* Неважно */  
  height: 100px; /* Неважно */  
  margin: auto;  /* Магия! */  
}
```

Суть в том, что `margin` с параметром `auto` в flex контейнере поглощает свободное пространство. Таким образом, выставив вертикальный отступ на `auto`, вы заставите элементы идеально сцентрироваться по двум осям.

Поняли этот момент? А теперь остановитесь и обязательно почитайте статью — [Всё о магии отступов в CSS Flexbox](#). В ней вы детально узнаете о том, как работать с отступами и что с ними делать. Бонусом прочитаете очень интересные примеры использования, которые вам пригодятся в работе.

Теперь давайте возьмем другие свойства. Рассмотрим список из шести элементов, с фиксированными размерами *для эстетического вида*, но они могут автоматически менять размер. Мы бы хотели, чтобы они были равномерно и хорошо распределены по горизонтальной оси, таким образом, чтобы при изменении размера окна браузера, все было гладко. И это без медиа запросов.

```
.flex-container {  
  /* Сначала зададим контейнеру flex контекст */  
  display: flex;  
  
  /* Далее определим направление главной оси и можно ли элементам  
  переноситься на другую строку  
  * Помните, что это одно и тоже:  
  * flex-direction: row;  
  * flex-wrap: wrap;  
  */  
  flex-flow: row wrap;  
  
  /* Далее мы определим каким образом будет распределено оставшееся  
  место */  
}
```



```
    justify-content: space-around;
}
```

Сделано! Все остальное это проблемы стилей. Ниже вы сможете видеть работу этого примера. Вообще пройдите на сам сайт, чтобы по изменять размеры окон и понаблюдать то, что происходит.

HTML

CSS

Result

EDIT ON

1

2

3

4

5

6

Resources

1× 0.5× 0.25×

Rerun

Давайте еще что-нибудь попробуем. Представьте, что у вас есть выровненная по правому борту навигация, находящаяся на самом верху вашего сайта, но вы хотите, чтобы она центрировалась на среднеразмерных экранах и была одной колонкой на совсем маленьких устройствах. Это сделать довольно просто.

```
/* Большие экраны */
.navigation {
  display: flex;
  flex-flow: row wrap;
  /* Выставляет элементы к концу строки по главной оси */
```

```
    justify-content: flex-end;
}

/* Средние экраны */
@media all and (max-width: 800px) {
    .navigation {
        /* На экранах средних размеров мы их равномерно центрируем,
        распределяя между ними пустое место */
        justify-content: space-around;
    }
}

/* Маленькие экраны */
@media all and (max-width: 500px) {
    .navigation {
        /* На малых экранах мы выставляем элементы в колонку */
        flex-direction: column;
    }
}
```

HTML

CSS

Result

EDIT ON

Run Pen

Resources1x0.5x0.25xRerun

Давайте попробуем что-нибудь ещё интереснее, пока играемся с flex'ами! Как насчет ориентированного на мобильные устройства трех-колоночного

шаблона с шапкой и подвалом на всю ширину экрана? И всё это вне зависимости от порядка элементов. Интересно?

```
.wrapper {  
  display: flex;  
  flex-flow: row wrap;  
}
```

```
/* Мы указываем всем элементам ширину 100% */  
.wrapper > * {  
  flex: 1 100%;  
}
```

**/\* Мы полагаемся на исходный порядок для мобильного подхода к делу  
\* В этом случае он такой:**

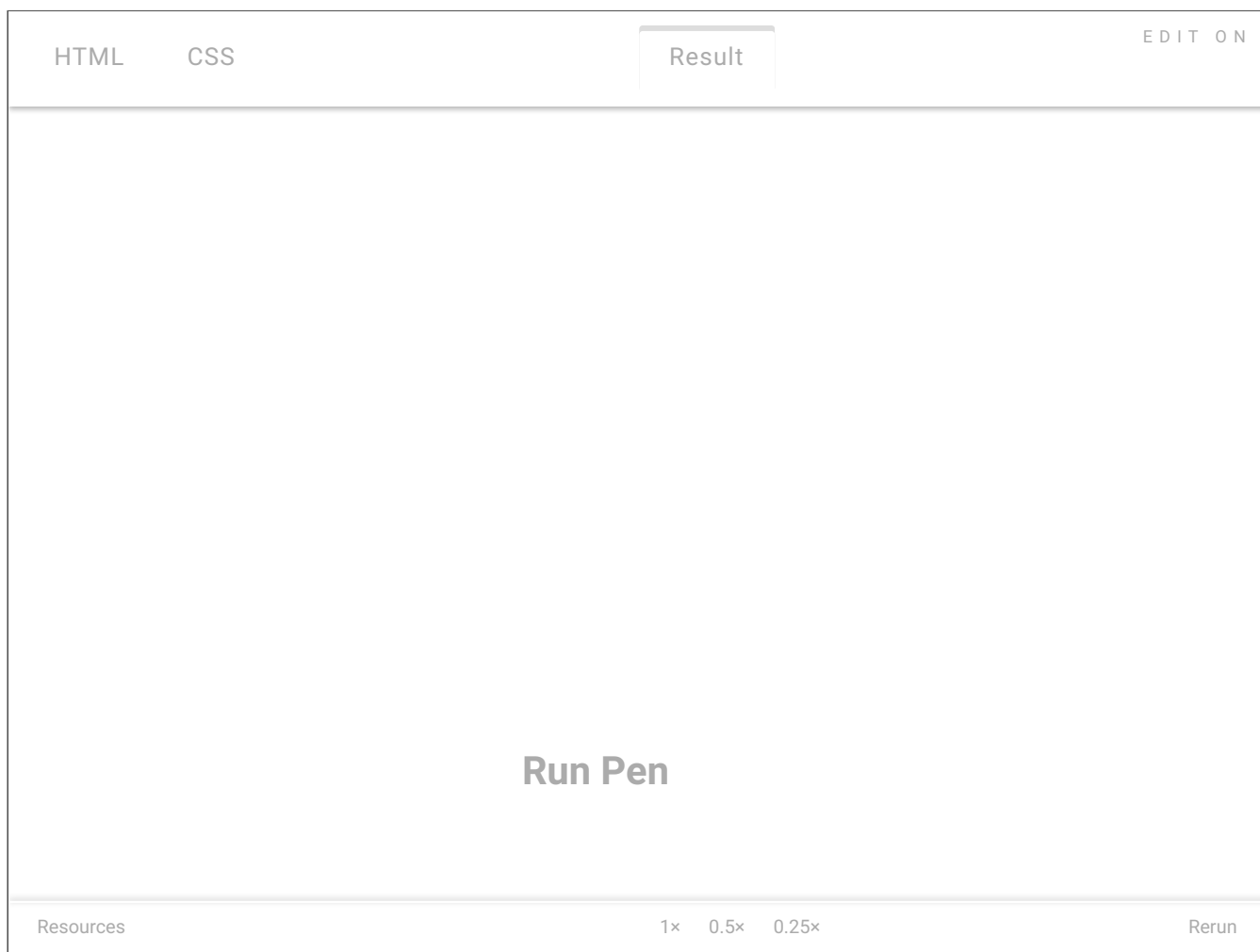
```
* 1. header  
* 2. article  
* 3. aside 1  
* 4. aside 2  
* 5. footer  
*/
```

**/\* Средние экраны \*/**

```
@media all and (min-width: 600px) {  
  /* Мы указываем двум сайдбарам разделить строку */  
  .aside { flex: 1 auto; }  
}
```

**/\* Large screens \*/**

```
@media all and (min-width: 800px) {  
  /* Мы меняем порядок первого сайдбара и main* И указываем главному  
  элементу взять в два раза больше ширины чем другие два сайдбара.*/  
  .main { flex: 2 0px; }  
  .aside-1 { order: 1; }  
  .main { order: 2; }  
  .aside-2 { order: 3; }  
  .footer { order: 4; }  
}
```



Если вы используете Bootstrap 4, который уже перешел на CSS Flexbox, то рекомендую к прочтению статью — [CSS Flexbox в Bootstrap 4](#). В ней вы узнаете о том, как упростить себе разработку на Bootstrap с помощью CSS Flexbox и уже встроенных классов, функционал которых описан в этой статье выше.

### Префиксы Flexbox

Flexbox нужны вендорные префиксы для поддержки большинства браузеров. Они не только включают добавленные сначала свойства с вендорным префиксом, тут есть аж целые **специальные** свойства и названия значений. Так получилось, потому что спецификация flexbox изменялась, создавая «старые», «промежуточные» и «новые» версии.

Возможно, лучшим решением является написание кода на последней версии синтаксиса и перезапуск вашего CSS через автопрефиксер, который очень хорошо справляется с добавлением фолбэков.

Более того, есть Sass примеси, чтобы помочь с некоторыми префиксерами, которые также дают вам представление о том, что вам нужно сделать.

```
@mixin flexbox() {
  display: -webkit-box;
  display: -moz-box;
  display: -ms-flexbox;
  display: -webkit-flex;
  display: flex;
}

@mixin flex($values) {
  -webkit-box-flex: $values;
  -moz-box-flex: $values;
  -webkit-flex: $values;
  -ms-flex: $values;
  flex: $values;
}

@mixin order($val) {
  -webkit-box-ordinal-group: $val;
  -moz-box-ordinal-group: $val;
  -ms-flex-order: $val;
  -webkit-order: $val;
  order: $val;
}

.wrapper {
  @include flexbox();
}

.item {
  @include flex(1 200px);
  @include order(2);
}
```

## Баги

Flexbox определенно не без багов. Лучшая их коллекция находится тут. Это open source пространство, где вы сможете отслеживать их новые появления. Так что, я думаю, что очень важно указать на них.

## Поддержка браузерами

Лично я считаю, что сейчас уже нет смысла разбирать детально эту тему. Достаточно зайти на **CanIUse** и самому убедиться, что уже 98.68%

поддерживают эту технологию.

% of all users			?
95.72%	+	2.96%	= 98.68%
95.56%	+	1.94%	= 97.5%

 1.95K

 10



Follow



Open in app ↗

Sign up

Sign in

 Medium



More from Stas Bagretsov

4	5	6
7	8	9
10	11	12



Stas Bagretsov

## Вёрстка на Grid в CSS. Полное руководство и справочник

👉 Подписывайтесь на мой Twitter! [@stassonmars](#)— теперь там ещё больше из мира фронтенда, да и вообще поговорим. 📢 Подписывайтесь, скоро...

30 min read · Mar 11, 2018



5K



38



# JS

Как работает `reduce()` в JavaScript, когда его нужно применять и какие крутые вещи он делает



Stas Bagretsov

## Как работает `reduce()` в JavaScript, когда его нужно применять и какие крутые вещи можно с ним...

В этой статье вы узнаете про метод `reduce()` и какие клевые штуки можно вытворять с его помощью, выйдя за рамки его общеизвестного...

8 min read · Oct 21, 2019



1.1K



5



# JS Понимаем замыкания в JavaScript. Раз и навсегда

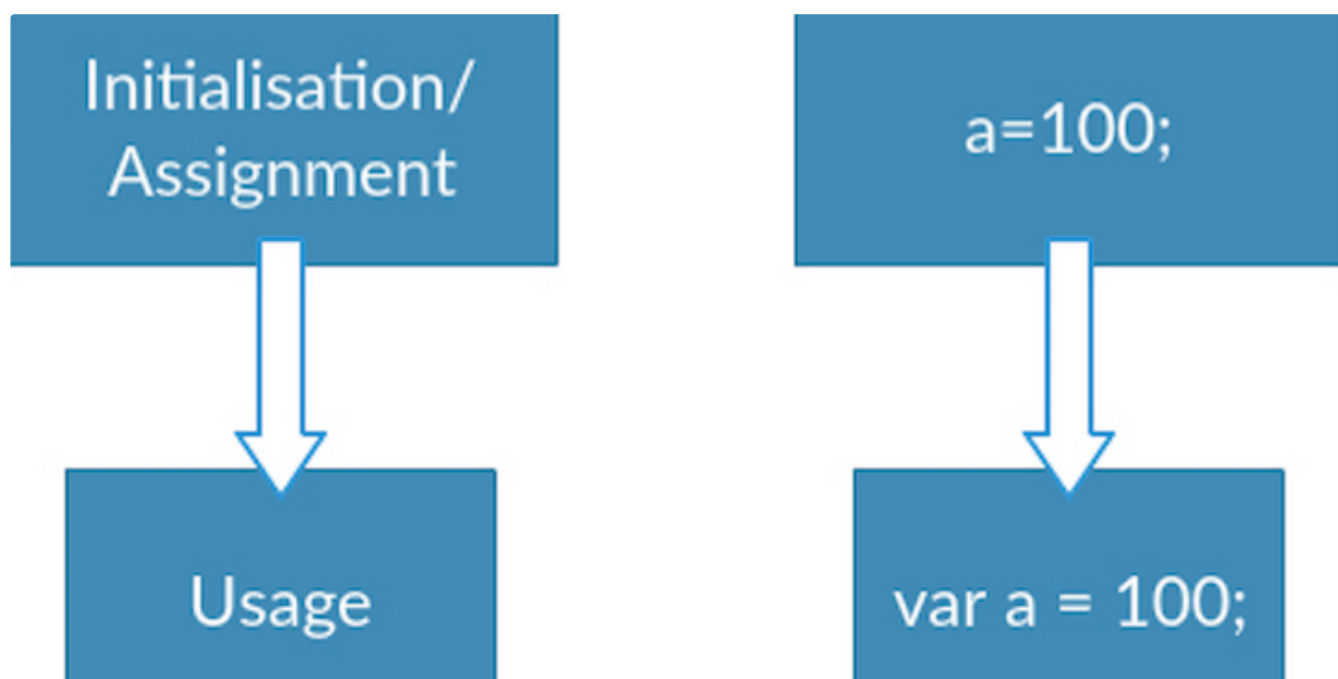
 Stas Bagretsov

## Понимаем замыкания в JavaScript. Раз и навсегда

На самом деле замыкания могут показаться запутанными с первого взгляда, но по факту в них нет ничего сложного.

10 min read · Mar 12, 2019

 1.91K  8



 Stas Bagretsov

## Разбираемся с “поднятием” (hoisting) в JavaScript



## Это перевод статьи Understanding Hoisting in JavaScript

9 min read · Feb 23, 2018



2.7K

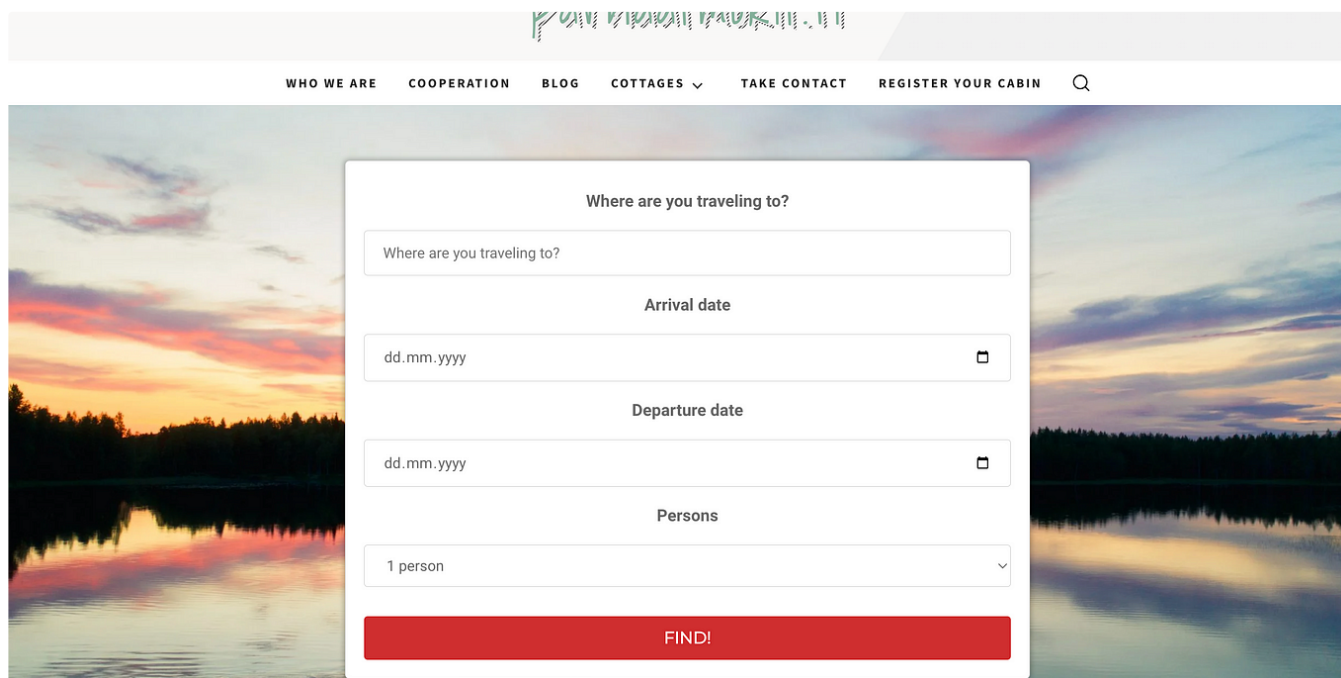


12



See all from Stas Bagretsov

## Recommended from Medium



Artturi Jalli

## I Built an App in 6 Hours that Makes \$1,500/Mo

Copy my strategy!



· 3 min read · Jan 23, 2024



5.8K



86



# { JSON } is slow?

```
{  
  "name": "JSON is slow!",  
  "blog": true,  
  "writtenAt": 1695884403,  
  "topics": ["JSON", "Javascript"]  
}
```



## Alternatives?



Vaishnav Manoj in DataX Journal

## JSON is incredibly slow: Here's What's Faster!

Unlocking the Need for Speed: Optimizing JSON Performance for Lightning-Fast Apps and Finding Alternatives to it!

16 min read · Sep 28, 2023



13.1K



154



### Lists

#### Staff Picks

580 stories · 742 saves

#### Stories to Help You Level-Up at Work

19 stories · 473 saves

#### Self-Improvement 101

20 stories · 1334 saves

#### Productivity 101

20 stories · 1220 saves

 James Presbitero Jr. in Practice in Public


## These Words Make it Obvious That Your Text is Written By AI

These 7 words are painfully obvious. They make me cringe. They will make your reader cringe.

5 min read · Dec 31, 2023

 37K  963



 Unbecoming

## 10 Seconds That Ended My 20 Year Marriage

It's August in Northern Virginia, hot and humid. I still haven't showered from my morning trail run. I'm wearing my stay-at-home mom...

★ · 4 min read · Feb 16, 2022

 75K    1058



 Gowtham Oleti

## Apps I Use And Why You Should Too.

Let's skip past the usual suspects like YouTube, WhatsApp and Instagram. I want to share with you some less familiar apps that have become...

10 min read · Nov 14, 2023

 17.2K    299



 Julien Etienne

## Stop Using localStorage

Bid farewell to localStorage! Embrace IndexedDB for speed, type-safe storage, and non-blocking data transactions. #IndexedDB #WebDev

7 min read · Dec 22, 2023



3.8K



56



---

See more recommendations