Адаптивная верстка

Отзывчивый веб-дизайн является практикой создания веб-сайта, подходящего для работы на любом устройстве с любым размером экрана, независимо от того, насколько он большой или маленький, мобильный или настольный.

Отзывчивый веб-дизайн сосредоточен вокруг представления об интуитивном и приятном опыте для каждого.

Настольный компьютер и сотовый телефон пользователя, все они выигрывают от отзывчивых сайтов.

Термин отзывчивый веб-дизайн сам по себе придумал и в значительной степени разработал Итан Маркотт.

Многое из того, что рассмотрено в этом уроке было впервые сказано Итаном в сети и в его книге Отзывчивый веб-дизайн, которую стоит прочитать.

Responsive Design (RWD) — отзывчивый дизайн — проектирование сайта с определенными значениями свойств, например, гибкая сетка макета, которые позволяют одному макету работать на разных устройствах;

Adaptive Design (AWD) — адаптивный дизайн, или динамический показ — проектирование сайта с условиями, которые изменяются в зависимости от устройства, базируясь на нескольких макетах фиксированной ширины.

Отзывчивый веб-дизайн разбивается на три основных компонента, включая гибкие макеты, медиа-запросы и гибкий медиа-контент.

Первая часть, гибкие макеты — это практика построения макета сайта с гибкой сеткой, которая способна динамически уменьшать размер до любой ширины.

Гибкие сетки строятся с использованием относительных единиц длины, как правило, процентов или единиц em.

Эти относительные длины затем применяются, чтобы объявить основные значения свойств сетки, таких как width, margin или padding.

Относительно шрифта: em

1em – текущий размер шрифта. Можно брать любые пропорции от текущего шрифта: 2em, 0.5em и т.п.

Размеры в em – *относительные*, они определяются по текущему контексту.

```
<div style="font-size:1.5em">

  Студенты

  <div style="font-size:1.5em">Студенты ИС</div>

</div>
```

Так как значение в em высчитывается относительно *текущего шрифта*, то вложенная строка в 1.5 раза больше, чем первая.

Выходит, размеры, заданные в em, будут уменьшаться или увеличиваться вместе со шрифтом. С учётом того, что размер шрифта обычно определяется в родителе, и может быть изменён ровно в одном месте, это бывает очень удобно.

em

Когда единицы измерения em используются в дочерних элементах, которые не имеют определенного размера шрифта, они наследуют его от родителей, вплоть до корневого элемента документа.

```
.example {
    font-size: 20px;
}
```

```
.example {
    font-size: 20px;
    border-radius: .5em;
}
```

Значение border-radius равное .5em будет равно 10px (то есть 20 * 0,5). Аналогично:

```
.example {
    font-size: 20px;
    border-radius: .5em;
    padding: 2em;    }
```

В данном случае 1em этого элемента или его дочерних элементов (при отсутствии других определений `font-size`) будет равен 20px.

Если в CSS размер шрифта не определен, то em будет равна размеру шрифта, используемого по умолчанию в браузере. Чаще всего это значение составляет 16px.

Проценты %

Проценты %, как и em – относительные единицы.

Когда мы говорим «процент», то возникает вопрос – «Процент от чего?»

Как правило, процент будет от значения свойства родителя с тем же названием, но не всегда.

Это очень важная особенность процентов, про которую, увы, часто забывают.

Отличный источник информации по этой теме – стандарт, Visual formatting model details.

Примеры-исключения, в которых % берётся не так:

margin-left

При установке свойства margin-left в %, процент берётся от ширины родительского блока, а вовсе не от его margin-left.

line-height

При установке свойства line-height в %, процент берётся от текущего размера шрифта, а вовсе не от line-height родителя. Детали по line-height и размеру шрифта вы также можете найти в литературе.

width/height

Для width/height обычно процент от ширины/высоты родителя, но при position:fixed, процент берётся от ширины/высоты окна (а не родителя и не документа). Кроме того, иногда % требует соблюдения дополнительных условий, их можно дополнительно уточнить в литературе.

Единица `rem` задаёт размер относительно размера шрифта элемента `<html>`.

Относительно экрана: vw, vh, vmin, vmax

Во всех современных браузерах, исключая IE8-, поддерживаются новые единицы из черновика стандарта CSS Values and Units 3:

- vw – 1% ширины окна
- vh – 1% высоты окна
- vmin – наименьшее из (vw, vh), в IE9 обозначается vm
- vmax – наибольшее из (vw, vh)

Эти значения были созданы, в первую очередь, для поддержки мобильных устройств.

Их основное преимущество – в том, что любые размеры, которые в них заданы, автоматически масштабируются при изменении размеров окна.

Гибкие макеты не выступают за использование фиксированных единиц измерения, таких как пиксели или дюймы. Причина в том, что высота и ширина области просмотра непрерывно меняются от устройства к устройству. Макеты сайтов нужно адаптировать к этим изменениям и у фиксированных значений слишком много ограничений.

Формула, которая помогает определить пропорции гибкого макета с помощью относительных значений, основана на взятии целевой ширины элемента и делении её на ширину родительского элемента. Результатом является относительная ширина целевого элемента.

цель ÷ контекст = результат

Гибкая сетка

HTML

```
<div class="container">
  <section>...</section>
  <aside>...</aside>
</div>
```

CSS

```
.container {
  width: 538px;
}
section,
aside {
  margin: 10px;
}
section {
  float: left;
  width: 340px;
}
aside {
  float: right;
  width: 158px;
}
```

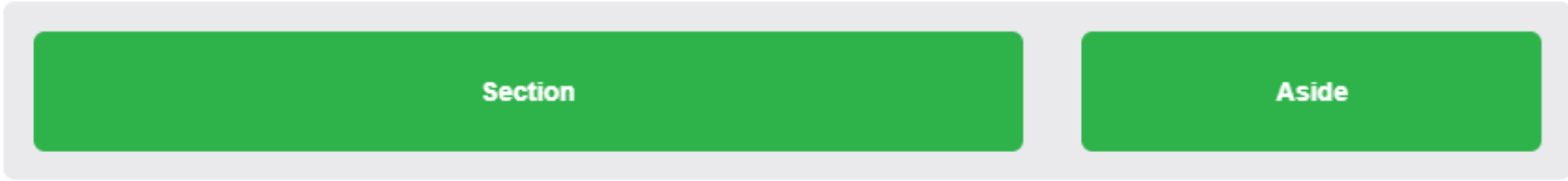
Используя формулу гибкой сетки можно взять все фиксированные единицы длины и превратить их в относительные единицы.

В этом примере используются процентами, но единицы em будут работать также хорошо.

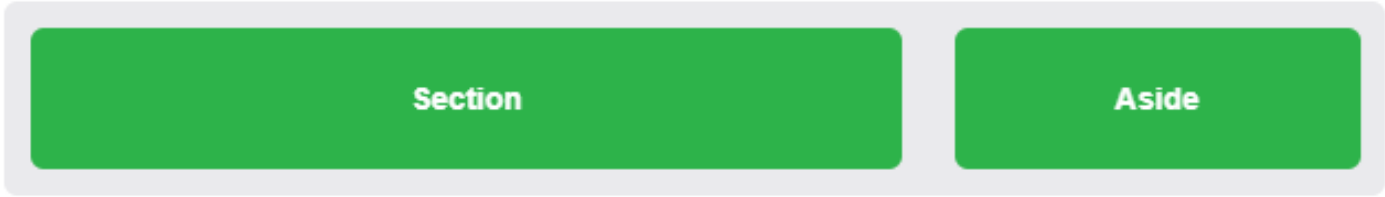
Независимо от того, насколько широким становится родительский container, margin и width для <section> и <aside> масштабируются пропорционально.

```
section,
aside {
  margin: 1.858736059%; /* 10px ÷ 538px = .018587361 */
}
section {
  float: left;
  width: 63.197026%;    /* 340px ÷ 538px = .63197026 */
}
aside {
  float: right;
  width: 29.3680297%;   /* 158px ÷ 538px = .293680297 */
}
```

100% ширина контейнера



75% ширина контейнера



50% ширина контейнера

Одного гибкого макета само по себе недостаточно.

Порой ширина браузера может быть настолько мала, что даже пропорциональное масштабирование макета будет создавать слишком узкие колонки для эффективного отображения контента.

В частности, когда макет становится слишком мал или слишком велик, текст может стать неразборчивым и компоновка может сломаться.

В этом случае могут быть использованы медиа-запросы, которые помогут оставить лучшее впечатление.

Гибкость текстового содержимого достигается путем вычисления размеров шрифта относительно размера шрифта в браузерах по умолчанию `16px`, например для фиксированного размера `font-size: 42px` относительный размер равен `42px / 16px = 2.625em`.

Проблема гибких изображений решается с помощью правила `img {width: 100%; max-width: 100%;}` для всех картинок на сайте.

Это правило гарантирует, что изображения никогда не будут шире, чем их контейнеры и никогда не превысят своих истинных размеров на больших экранах.

Медиа-запросы

HTML4 и CSS2 на данный момент поддерживают медиа-зависимые таблицы стилей адаптированные для разных *медиа-типов*.

Например, документ может использовать шрифт sans-serif при отображении на экране и шрифт serif при печати. ‘screen’ и ‘print’ являются двумя медиа-типами, которые были определены.

Медиа-запросы расширяют функциональность медиа типов, разрешая более точную маркировку таблиц стилей.

Медиа-запрос состоит из медиа-типа и нуля или больше выражений, которые проверяют условия конкретных *медиа-функций*.

Среди медиа-функций, которые могут использоваться в медиа-запросах, есть ‘width’, ‘height’, и ‘color’.

С помощью использования медиа-запросов, документы могут быть адаптированы на определенный диапазон выходных устройств без изменения самого контента.

Метатег viewport

Для управления разметкой в мобильных браузерах используется метатег `viewport`.

Мобильные браузеры отображают страницы в виртуальном окне просмотра, которое обычно шире, чем экран устройства.

С помощью метатега `viewport` можно контролировать размер окна просмотра и масштаб.

Страницы, адаптированные для просмотра на разных типах устройств должны содержать в разделе `<head>` метатег `viewport`.

```
<meta name="viewport" content="width=device-
width, initial-scale=1.0">
```

Свойство `width` определяет виртуальную ширину окна просмотра, значение `device-width` — физическую ширину устройства.

Другими словами, `width` отражает значение `document.documentElement.clientWidth`, a device-width — `screen.width`.

При первой загрузке страницы свойство `initial-scale` управляет начальным уровнем масштабирования, `initial-scale=1` означает, что 1 пиксель окна просмотра = 1 пиксель CSS.


htc
10:08 AM
device-width
width

Синтаксис

Все запросы начинаются с правила `@media`, после чего следует условие, в котором используются типы носителей, логические операторы и медиа-функции.

Типы носителей перечислены в таблице

Тип	Описание
all	Все типы. Это значение используется по умолчанию.
braille	Устройства, основанные на системе Брайля, которые предназначены для чтения слепыми людьми.
embossed	Принтеры, использующие для печати систему Брайля.
handheld	Смартфоны и аналогичные им аппараты.

print	Принтеры и другие печатающие устройства.
projection	Проекторы.
screen	Экран монитора.
speech	Речевые синтезаторы, а также программы для воспроизведения текста вслух. Сюда, например, можно отнести речевые браузеры.
tty	Устройства с фиксированным размером символов (телетайпы, терминалы, устройства с ограничениями дисплея).
tv	Телевизоры.

Логические операторы, применяемые в медиа-запросах

and - Логическое И. Указывается для объединения нескольких условий.

Пример. Стиль для всех цветных устройств

```
@media all and (color) { ... }
```

not - Логическое НЕ. Указывается для отрицания условия.

Пример. Стиль для всех устройств кроме смартфонов

```
@media all and (not handheld) { ... }
```

Оператор `not` имеет низкий приоритет и оценивается в запросе последним, поэтому выражение `@media not all and (color) { ... }` следует понимать как `@media not (all and (color)) { ... }` а не `@media (not all) and (color) { ... }`

only - Применяется для старых браузеров, которые не поддерживают медиа-запросы.

Пример. Стиль для новых браузеров

```
@media only all and (not handheld) { ... }
```

В списке нет логического оператора ИЛИ, его роль выполняет запятая.

Перечисление нескольких условий через запятую говорит о том, что если хотя бы одно условие выполняется, то стиль будет применён.

Пример. Стиль для устройств с альбомной ориентацией или минимальной шириной 480 пикселов.

```
@media all and (orientation: landscape), all and (min-width: 480px) { ... }
```

Также при использовании операторов следует указывать скобки, чтобы менять приоритет операций.

Медиа-функции

Медиа-функции задают технические характеристики устройства, на котором отображается документ.

Стиль выполняется в том случае, если запрос возвращает истину, иными словами, указанные условия выполняются.

Большинство функций содержат приставку min- и max-, которая соответствуют минимальному и максимальному значению.

Так, `max-width: 400px` означает, что ширина окна браузера меньше 400 пикселей, а `min-width: 1000px`, наоборот, сообщает, что ширина окна больше 1000 пикселов.

aspect-ratio (min-aspect-ratio, max-aspect-ratio)

Тип носителя: handheld, print, projection, screen, tty, tv.
Значение: целое число/целое число

Определяет соотношение ширины и высоты отображаемой области устройства. Значение указывается в виде двух целых чисел разделяемых между собой слэшем (/).

color (min-color, max-color)

Тип носителя: handheld, print, projection, screen, tty, tv. Значение: целое число

Определяет число бит на канал цвета. К примеру, значение 3 означает, что красный, зелёный и синий канал могут отображать 2^3 цветов каждый, что в общем составляет 512 цветов (8×8×8). Если значение не указано, тогда проверяется что устройство цветное. В примере показана такая проверка.

Пример. Стиль для цветных устройств

```
@media screen and (color) { /* Для цветных экранов */
  body { background: #fc0; }
}
@media screen and (min-color:3) { /* Минимум 512 цветов */
  body { background: #ccc; }
}
```

color-index (min-color-index, max-color-index)

Тип носителя: handheld, print, projection, screen, tty, tv. Значение: целое число

Определяет количество цветов, которое поддерживает устройство. В примере показан стиль для экранов отображающих не меньше 256 цветов.

Пример. Цветной дисплей

```
@media all and (min-color-index: 256) {
                    ...
}
```

device-aspect-ratio (min-device-aspect-ratio, max-device-aspect-ratio)

Тип носителя: handheld, print, projection, screen, tty, tv. Значение:

целое число/целое число

Определяет соотношение сторон экрана устройства.

Значение указывается в виде двух целых чисел разделяемых между собо
слэшем (/).

В примере показано, как установить стиль для экранов с соотношение
сторон 16:9 и более.

Пример. «Киношное» соотношение

```
@media screen and (min-device-aspect-ratio: 16/9)
                                ...
}
```

device-height (min-device-height, max-device-height)

Тип носителя: все кроме speech. Значение: размер

Определяет всю доступную высоту экрана устройства или печатной страницы.

device-width (min-device-width, max-device-width).

Тип носителя: все кроме speech. Значение: размер

Определяет всю доступную ширину экрана устройства или печатной страницы. В примере в зависимости от разрешения монитора устанавливается ширина слоя. Так, для значения 1280 пикселов ширина макета задаётся как 1100px.

```
    @media screen and (min-device-width:
1600px) {
     div {width: 1500px;}
    }
    @media screen and (device-width: 1280px)
{     div {width: 1100px;}
    }
    @media screen and (device-width: 1024px)
{     div {width: 980px;}
    }
```

grid

Тип носителя: all. Значение: нет

Определяет, что это устройство с фиксированным размером символов. Размеры букв на таком устройстве занимают одинаковую ширину и высоту и выстраиваются по заданной сетке. К подобным устройствам можно отнести терминалы, а также телефоны, которые поддерживают только один шрифт.

Если вам требуется форматировать текст, не указывайте его размер в пикселах, для подобных устройств используется единица em.

```
  @media screen and (grid) and (max-width:
15em) {
     body { font-size: 2em; }
   }
```

height (min-height, max-height)

Тип носителя: все кроме speech. Значение: размер

Высота отображаемой области.

width (min-width, max-width)

Тип носителя: все кроме speech. Значение: размер

Описывает ширину отображаемой области. Это может быть окно браузера или печатная страница. В данном примере при уменьшении окна до 600 пикселей и меньше меняется цвет фона веб-страницы.

```
body { background: #f0f0f0; }
 @media screen and (max-width: 600px) {
   body { background: #fc0; }
 }
```

monochrome (min-monochrome, max-monochrome).

Тип носителя: handheld, print, projection, screen, tty, tv. Значение: целое число

Определяет, что устройство монохромное. Если указано число, то оно обозначает число бит на пиксел. Так, значение 8 равнозначно 256 оттенкам серого (или другого цвета). В примере показан стиль для монохромного и цветного принтера.

Пример. Стиль для принтера

```
@media print and (monochrome) {
 body { font-family: Times, 'Times New Roman',
serif; }

h1, h2, p { color: black; }
}

@media print and (color) {
 body { font-family: Arial, Verdana, sans-serif;
}

h1, h2, p { color: #556b2f; }
}
```

orientation

Тип носителя: handheld, print, projection, screen, tty, tv. Значение: landscape | portrait

Определяет, что устройство находится в альбомном режиме (ширина больше высоты) или портретном (ширина меньше высоты).

В примере устанавливается разная фоновая картинка в случае альбомной (landscape) или портретной ориентации (portrait).

Пример. Использование ориентации устройства

```
@media screen and (orientation: landscape) {
  #logo { background: url(logo1.png) no-
repeat; }
}
@media screen and (orientation: portrait) {
  #logo { background: url(logo2.png) no-
repeat; }
}
```

resolution (min-resolution, max-resolution)

Тип носителя: handheld, print, projection, screen, tv. Значение: разрешение в dpi (точек на дюйм) или dpcm (точек на сантиметр)

Определяет разрешение устройства, например, принтера. В примере стиль будет работать для принтера с минимальным разрешением 300 точек на дюйм.

Пример. Разрешение принтера

```
@media print and (min-resolution: 300dpi) {
                    ...
}
```

scan

Тип носителя: tv. Значение: interlace | progressive

Определяет тип развертки телевизора — чересстрочная (interlace) или прогрессивная (progressive). При чересстрочной развёртке телевизор вначале показывает нечётные строки кадра, затем чётные, что позволяет сократить передаваемые данные. В прогрессивной развёртке кадр передаётся и показывается целиком.

Позиционирование
float
flex

Позиционирование элементов

- Свойство position, в сочетании со свойствами `left`, `top`, `right`, `bottom`, `display`, `clear` и ряда других позволяет управлять положением элементов на странице и порядком их вывода.
- Свойство `position` может принимать такие значения:

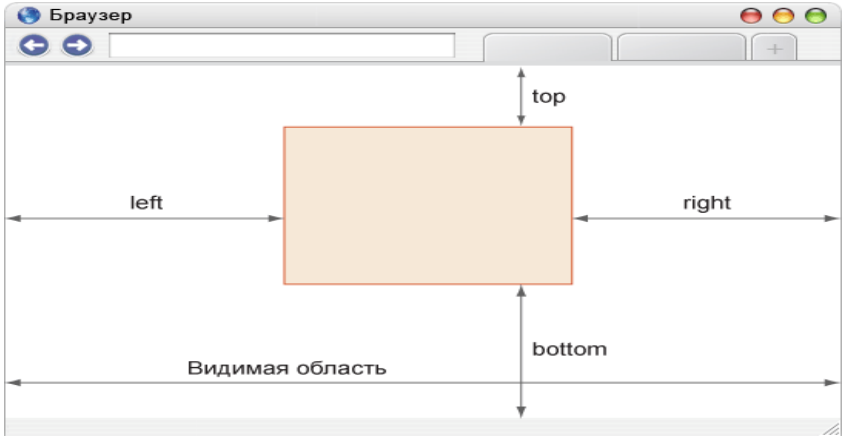
`static` — нормальное положение

`relative` — относительное позиционирование

`absolute` — абсолютное позиционирование

`fixed` — фиксированное положение

Абсолютное позиционирование



- Слой не меняет своё исходное положение, если у него нет св-в `right`, `left`, `top`, `bottom`.
- Ширина слоя, если она не задана явно, равна ширине контента плюс значения полей, границ и отступов.

- Свойства `left` и `top` имеют более высокий приоритет по сравнению с `right` и `bottom`.
- Если `left` задать отрицательное значение, то слой уйдёт за левый край браузера, полосы прокрутки при этом не возникнет. То же относится и к свойству `top`, только слой уйдёт за верхний край.
- Если `left` задать значение больше ширины видимой области или указать `right` с отрицательным значением, появится горизонтальная полоса прокрутки. Тоже для `top`, только полоса прокрутки вертикальная.

- Одновременно указанные свойства `left` и `right` формируют ширину слоя, если `width` не указано. Аналогично произойдёт и с высотой слоя (`top`, `bottom` и `height`).
- Элемент с абсолютным позиционированием перемещается вместе с документом при его прокрутке.

Фиксированное положение

- Слой привязывается к указанной свойствами `left`, `top`, `right` и `bottom` точке на экране и не меняет своего положения при прокрутке веб-страницы.
- При выходе фиксированного слоя за пределы видимой области справа или снизу от неё, не возникает полос прокрутки.

Относительное позиционирование

- Положение элемента устанавливается относительно его исходного места.
- Положительное значение `left` определяет сдвиг вправо от левой границы элемента, отрицательное — сдвиг влево.
- Положительное значение `top` задаёт сдвиг элемента вниз , отрицательное — сдвиг вверх.

- При положительном значении `right` сдвигает элемент влево от его правого края, при отрицательном — сдвигает вправо.
- При положительном значении `top` элемент опускается вниз, при отрицательном поднимается вверх.

CSS слои

- Свойство `z-index` определяет уровень стека HTML-элемента.
- Элемент с бóльшим номером перекрывает элемент с меньшим номером.
- Свойство `z-index` будет работать только с тем элементом, у которого свойство position установлено в одно из трех значений: `absolute`, `fixed` или `relative`.
- Использование JavaScript: CSS-свойство «`z-index`», записывается как «`zIndex`».

Свойство "float"

Свойство `float` в CSS занимает особенное место.

До его появления расположить два блока один слева от другого можно было лишь при помощи таблиц.

Синтаксис:

```
float: left | right | none | inherit;
```

Всего есть 4 значения для свойства `float`.

`Left` и `right` используются для соответствующих направлений.

`None` (по умолчанию) - обеспечивает, что элемент не будет "плавать".

И `inherit`, которое говорит, что поведение должно быть такое же, как и у родительского элемента.

При применении этого свойства происходит следующее:

1. Элемент позиционируется как обычно, а затем вынимается из документа потока и сдвигается влево (для `left`) или вправо (для `right`) до того как коснётся либо границы родителя, либо другого элемента с `float`.

2. Если пространства по горизонтали не хватает для того, чтобы вместить элемент, то он сдвигается вниз до тех пор, пока не начнёт помещаться.

3. Другие непозиционированные блочные элементы без `float` ведут себя так, как будто элемента с `float` нет, так как он убран из потока.

4. Строки (inline-элементы), напротив, «знают» о float и обтекают элемент по сторонам.

1. Элемент при наличии `float` получает `display:block`.
2. То есть, указав элементу, у которого display:inline свойство `float: left/right`, мы автоматически сделаем его блочным. В частности, для него будут работать `width/height`.
3. Исключением являются некоторые редкие display наподобие `inline-table` и `run-in`.
4. Ширина `float`-блока определяется по содержимому.

5. Вертикальные отступы `margin` элементов с `float` не сливаются с отступами соседей, в отличие от обычных блочных элементов.

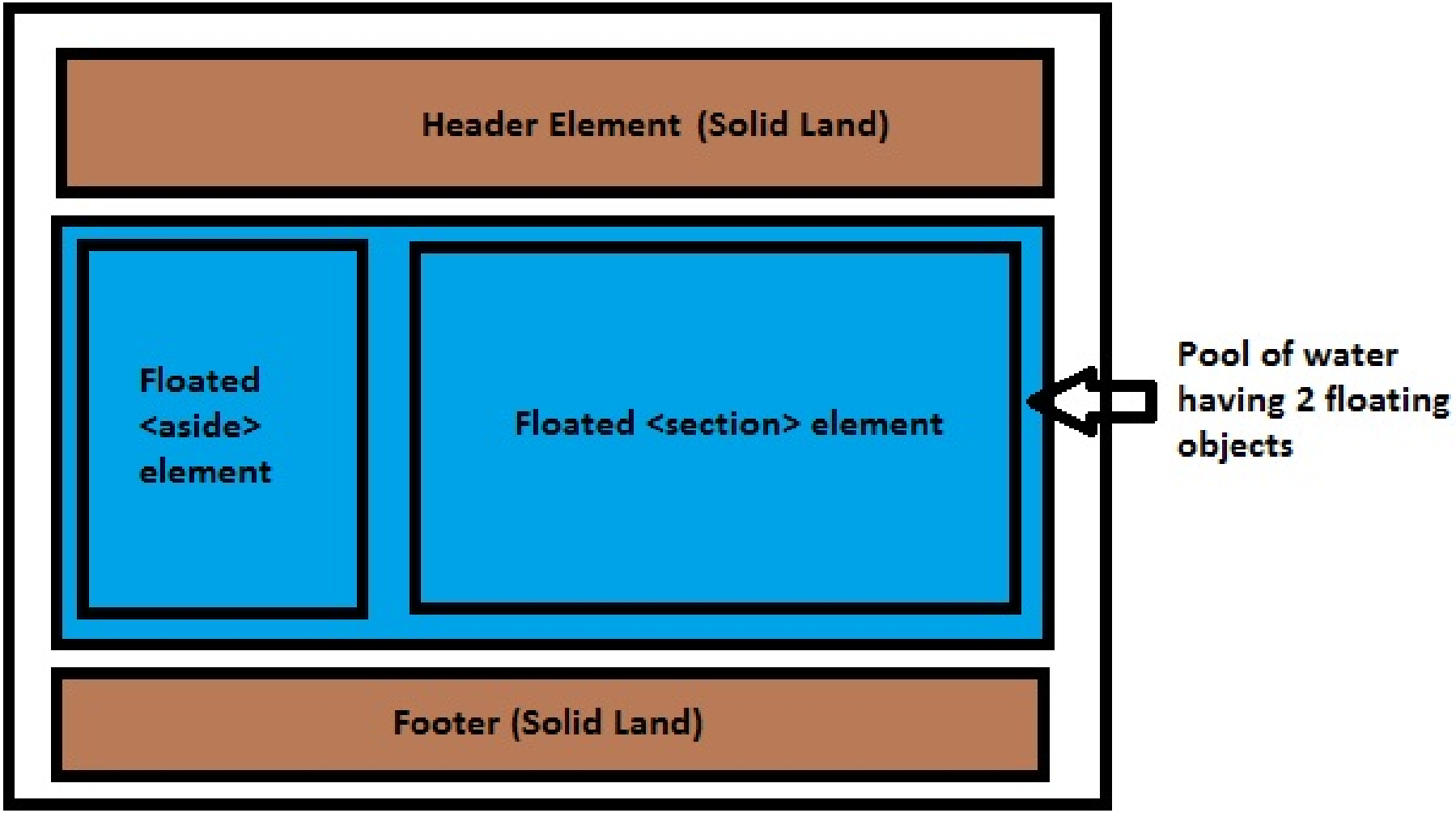
Для чего можно использовать float?

Текст с картинками

Одно из первых применений `float`, для которого это свойство когда-то было придумано – это вёрстка текста с картинками, отжатыми влево или вправо.

Каждая картинка, у которой есть `float`, обрабатывается в точности по алгоритму, указанному выше.

Помимо обтекания текстом изображений, `float` может использоваться для создания макета всего сайта


Header Element (Solid Land)
Floated <aside> element
Floated <section> element
Pool of water having 2 floating objects
Footer (Solid Land)

```
<header>
    Header
</header>
<aside>
    Aside (Floated Left)
</aside>
<section>
    Content (Floated Left)
</section>
```

```
<!-- Clearing Floating Elements-->
<div class="clear"></div>
<footer>
    Footer
</footer>
```

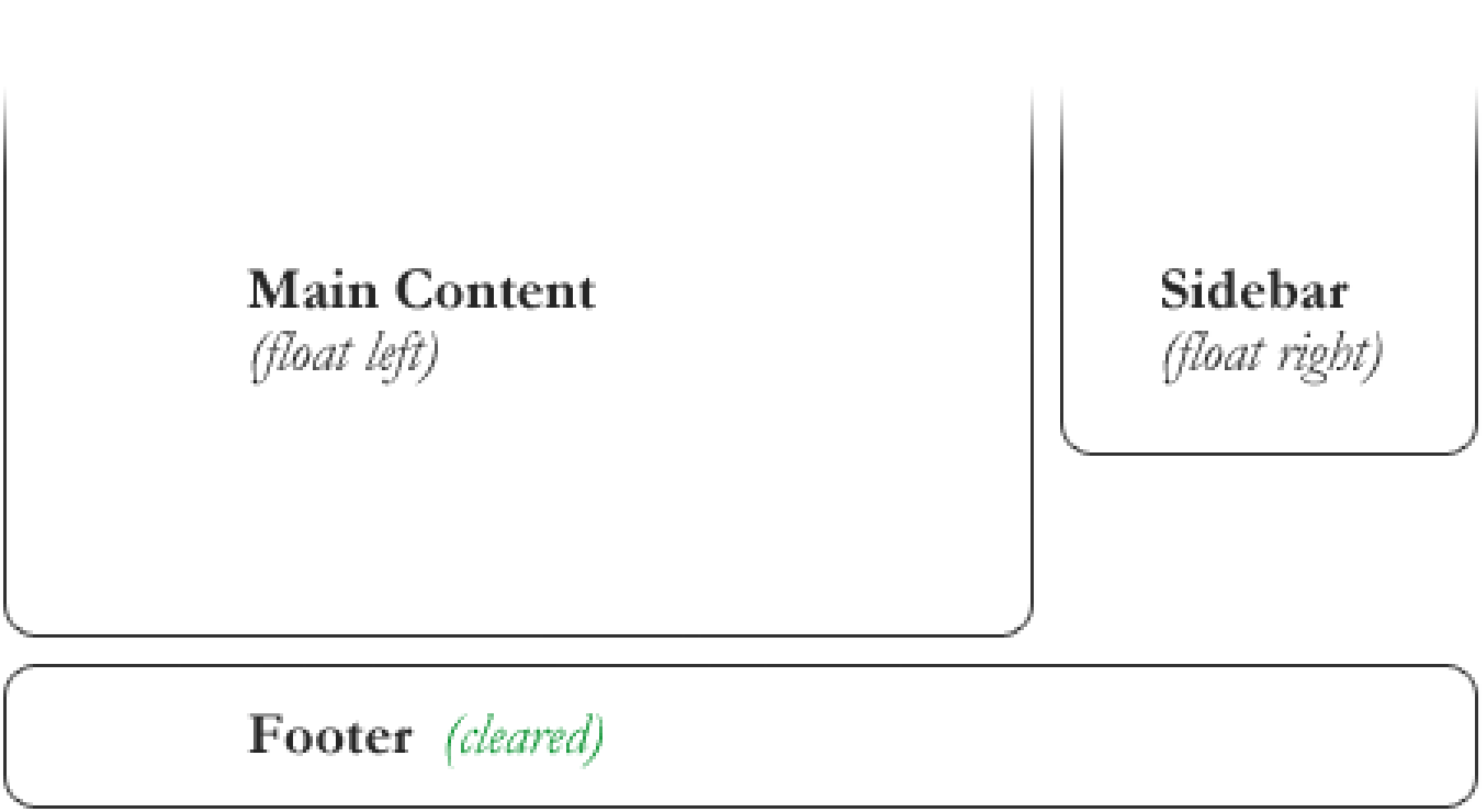
```
header, footer {
  border: 5px solid #000;
  height: 100px;
}
aside {
  float: left;
  width: 30%;
  border: 5px solid #000;
  height: 300px; }
```

```
section {
float: left;
width: 70%; border: 5px solid #000;
height: 300px; }
.clear {
clear: both;
}
```

Отмена свойства float

В примере, сайд-бар прижат к правому краю (`float: right;`), а его высота меньше, чем область основного контента. Поэтому `footer` будет поднят выше, поскольку для него хватает высоты и этого требует поведение `float`.

Чтобы исправить ситуацию, ему необходимо установить свойство `clear`, которое гарантирует, что элемент выведется ниже `float`-элементов.


Main Content
(float left)
Sidebar
(float right)
Footer (cleared)

Свойство `clear` может принимать четыре значения.

Both, наиболее используемое, применяется для отмены float каждого из направлений.

`Left` и `Right` - используются для отмены `float` одного из направлений.

`None` - по умолчанию, обычно не используется, за исключением случаев, когда необходимо отменить значение `clear`.

Значение `inherit` было бы пятым значением, но оно странным образом не поддерживается в Internet Explorer.

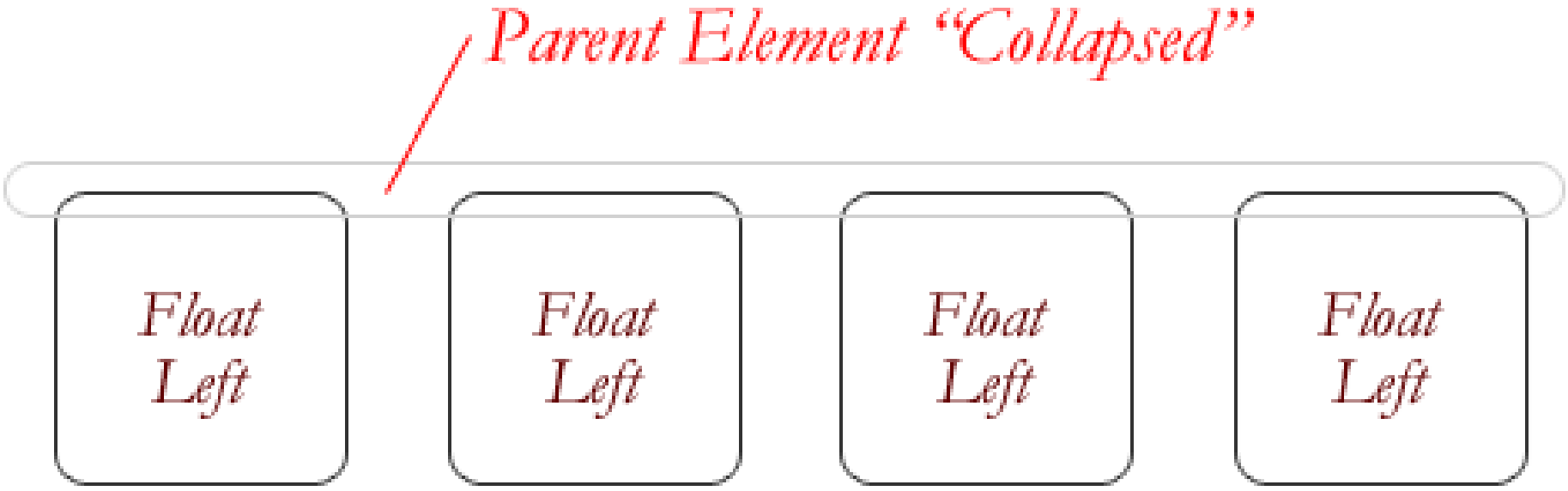
Отмена только левого или правого `float`, встречается гораздо реже, но, безусловно, имеет свои цели.

Большой коллапс

Ещё одна удивительная вещь при работе со свойством `float` - это то, что его использование может влиять на родительский элемент.

Если такой элемент содержит только `float`-элементы, то он буквально схлопывается, то есть его высота равна нулю.

Это не всегда заметно, если у родительского элемента не установлен какой-либо видимый фон.



Для того чтобы изменить такое поведение, необходимо добавить элемент отменяющий `float` после `float`-элементов, но до закрытия родительского элемента.

Метод пустого div-а. Используется, в буквальном смысле, пустой div. `<div style="clear: both;"></div>`.

Иногда на его месте может использоваться элемент `<br />` или какой-нибудь другой, но divис пользуется чаще всего, потому что по умолчанию у него нет никакого стиля, нет особого назначения и вряд ли к нему применён общий стиль через CSS.

- Метод `overflow`. Основан на том, что родительскому элементу необходимо установить свойство `overflow`. Если значение этого свойства установлено в `auto` или `hidden`, то родительский элемент увеличится, чтобы вместить в себя все `float`-элементы. Этот метод выглядит более семантически правильным, поскольку не требует дополнительных элементов.

Однако, если вы соберётесь использовать ещё один `div`, только для того чтобы использовать этот подход (имеется в виду родительский `div`), то это будет то же самое, что предыдущий пример с добавлением пустого `div`-а.

Кроме того, помните, что свойство `overflow` предназначено для других целей.

Будьте аккуратны и не допустите, что часть контента у вас пропадёт, либо появятся никому не нужные скролл-бары.

Метод простой очистки. Этот метод использует замечательный псевдо селектор CSS - `:after`. Гораздо лучше чем использование overflow для родительского элемента.

Вы просто устанавливаете ему дополнительный класс, объявленный, например, так:

```
.clearfix:after {
    content     : ".";
    visibility  : hidden;
    display     : block;
    height      : 0;
    clear       : both;
}
```

Этот способ добавляет незаметное для глаз содержимое и отменяет `float`.

FLEX BOX

CSS flexbox *(Flexible Box Layout Module)* — модуль макета гибкого контейнера — представляет собой способ компоновки элементов. В основе flexbox лежит идея оси.

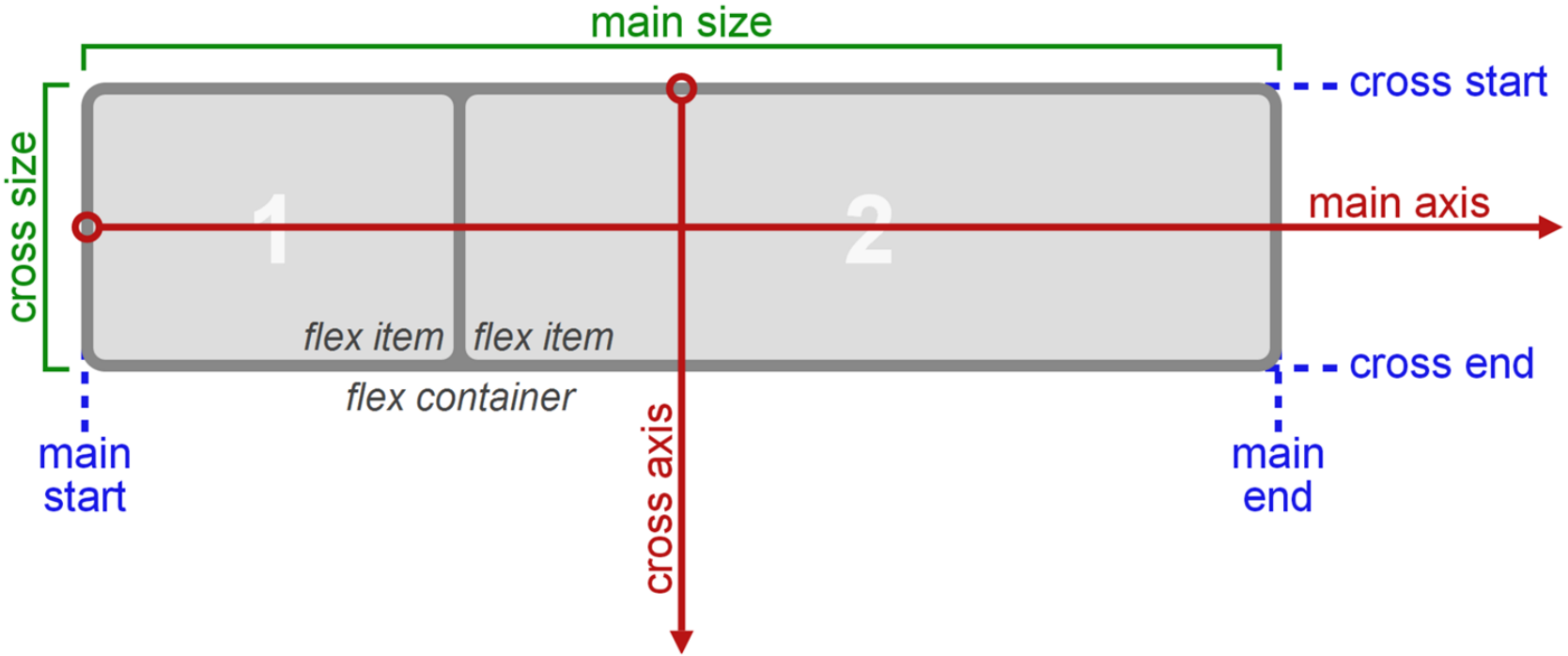
Flexbox состоит из **flex-контейнера** — родительского контейнера и **flex-элементов** — дочерних блоков.

Дочерние элементы могут выстраиваться в строку или столбик, а оставшееся свободное пространство распределяется между ними различными способами.

Модуль flexbox позволяет решать следующие задачи:

- Располагать элементы в одном из четырех направлений: слева направо, справа налево, сверху вниз или снизу вверх;
- Переопределять порядок отображения элементов;
- Автоматически определять размеры элементов таким образом, чтобы они вписывались в доступное пространство;
- Решать проблему с горизонтальным и вертикальным центрированием;
- Переносить элементы внутри контейнера, не допуская их переполнения;
- Создавать колонки одинаковой высоты;
- Создавать прижатый к низу страницы подвал сайта.

Flexbox решает специфические задачи — создание одномерных макетов, например, размещение элементов навигации, так как flex-элементы можно позиционировать только вдоль главной или поперечной оси.



Свойства flex-контейнера

Flex-контейнер устанавливает новый гибкий контекст форматирования для его содержимого. Flex-контейнер не является блочным контейнером, поэтому для внутренних блоков не работают такие CSS-свойства, как `float`, `clear`, `vertical-align`.

Также, на flex-контейнер не оказывают влияние свойства `column-`, создающие колонки в тексте и псевдоэлементы `::first-line` и `::first-letter`.

Свойство display

Модель flexbox-разметки связана с определенным значением CSS-свойства `display` родительского html-элемента, содержащего внутри себя дочерние блоки.

Для управления элементами с помощью этой модели нужно установить свойство `display` следующим образом:

```
.flex-container {
display: flex; /*отображает контейнер как блочный элемент*/
}
.flex-container {
 display: inline-flex; /*отображает контейнер как строчный элемент*/
}
```

После установки данных значений свойства каждый дочерний элемент автоматически становится flex-элементом, выстраиваясь в ряд (вдоль главной оси) колонками одинаковой высоты, равной высоте блока-контейнера.

При этом блочные и строчные дочерние элементы ведут себя одинаково, т.е. ширина блоков равна ширине их содержимого с учетом внутренних полей и рамок элемента.

div1 div2 div3

span1 span2 span3

 simple text div1 div2

Если родительский блок содержит текст или изображения без оберток, они становятся анонимными flex-элементами.
Текст выравнивается по верхнему краю блока-контейнера, а высота изображения становится равной высоте блока, т.е. оно деформируется.

Выравнивание элементов по горизонтали justify-content

Свойство выравнивает flex-элементы по ширине flex-контейнера, распределяя оставшееся свободное пространство, незанятое flex-элементами.

Для выравнивания элементов по вертикали используется свойство `align-content`.

Свойство не наследуется.

`flex-start` - Значение по умолчанию. Flex-элементы позиционируются от начала flex-контейнера.

`flex-end` - Flex-элементы позиционируются относительно правой границы flex-контейнера.

`center` - Flex-элементы выравниваются по центру flex-контейнера.

`space-between` - Flex-элементы выравниваются по главной оси, свободное место между ними распределяется следующим образом: первый блок располагается в начале flex-контейнера, последний блок – в конце, все остальные блоки равномерно распределены в оставшемся пространстве, а свободное пространство равномерно распределяется между элементами.

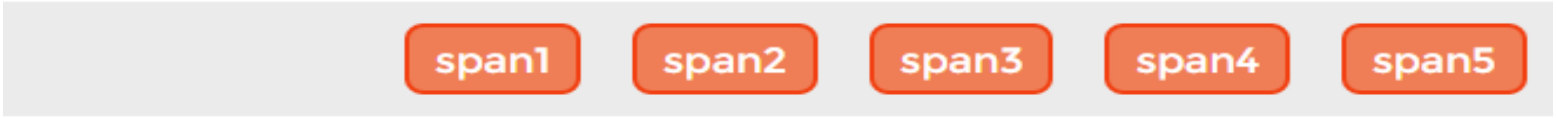
`space-around` - Flex-элементы выравниваются по главной оси, а свободное место делится поровну, добавляя отступы справа и слева.

`initial` - Устанавливает значение свойства в значение по умолчанию.

`inherit` - Наследует значение свойства от родительского элемента.

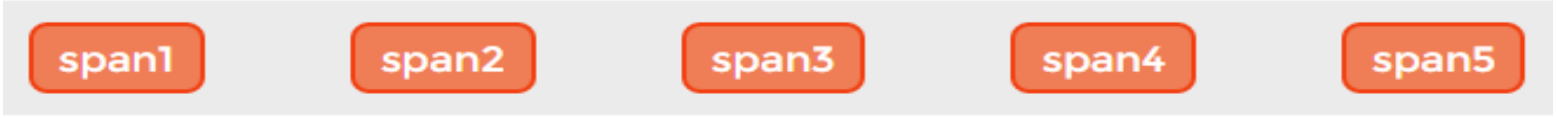
```
justify-content: flex-start;
```


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




```
justify-content: center;
```


```
justify-content: space-between;
```




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

Выравнивание элементов по вертикали align-items

Свойство выравнивает flex-элементы, в том числе и анонимные flex-элементы по перпендикулярной оси (по высоте).

Не наследуется.

`stretch` – Значение по умолчанию. Flex-элементы растягиваются, занимая все пространство по высоте.

`flex-start` – Flex-элементы выравниваются по левому краю flex-контейнера относительно верхнего края блока-контейнера.

`flex-end` – Flex-элементы выравниваются по левому краю flex-контейнера относительно нижнего края блока-контейнера.

`center` – Flex-элементы выравниваются по центру flex-контейнера.

`baseline` – Flex-элементы выравниваются по базовой линии.

```
align-items: stretch;
```


```
align-items: flex-start;
```


```
align-items: flex-end;
```

```
align-items: center;
```


```
align-items: baseline;
```

Направление главной оси flex-direction

Свойство определяет, каким образом flex-элементы укладываются во flex-контейнере, задавая направление главной оси flex-контейнера.

Они могут располагаться в двух главных направлениях — горизонтально, как строки или вертикально, как колонки.

Главная ось по умолчанию идет слева направо.

Поперечная - сверху вниз.

Свойство не наследуется.

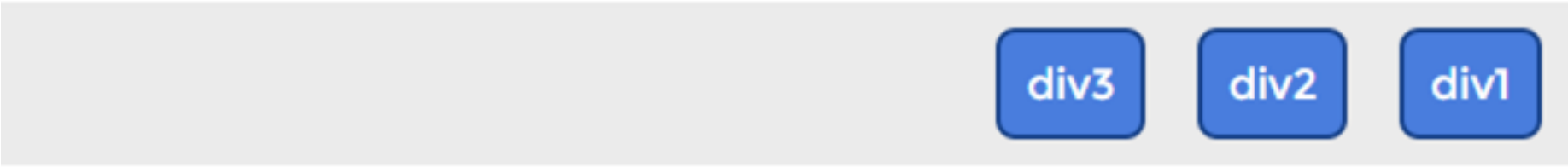
`row` – Значение по умолчанию, слева направо (в rtl справа налево). Flex-элементы выкладываются в строку. Начало (main-start) и конец (main-end) направления главной оси соответствуют началу (inline-start) и концу (inline-end) инлайн оси (inline-axis).

`row-reverse` - Направление справа налево (в rtl слева направо). Flex-элементы выкладываются в строку относительно правого края контейнера (в rtl — левого).

`column` – Направление сверху вниз. Flex-элементы выкладываются в колонку.

`column-reverse` – Колонка с элементами в обратном порядке, снизу вверх.

```
flex-direction: row-reverse;
```




```
flex-direction: column;
```

```
flex-direction: column-reverse;
```

Многострочность элементов flex-wrap

Свойство управляет тем, как flex-контейнер будет выкладывать flex-элементы — в одну строку или в несколько, и направлением, в котором будут укладываться новые строки.

По умолчанию flex-элементы укладываются в одну строку.

При переполнении контейнера их содержимое будет выходить за границы flex-элементов. Не

`nowrap` – Значение по умолчанию. Flex-элементы не переносятся, а располагаются в одну линию слева направо (в rtl справа налево).

`wrap` – Flex-элементы переносятся, располагаясь в несколько горизонтальных рядов (если не помещаются в один ряд) в направлении слева направо (в rtl справа налево).

`wrap-reverse` – Flex-элементы переносятся, располагаясь в обратном порядке слева-направо, при этом перенос происходит снизу вверх.

```
flex-wrap: wrap;
```


```
flex-wrap: wrap-reverse;
```

Краткая запись направления и многострочности flex-flow

Свойство предоставляет возможность в одном свойстве задать направление главной оси и многострочность поперечной оси, т.е. сокращённая запись свойств `flex-direction` и `flex-wrap`.
Значение по умолчанию `flex-flow: row nowrap;`.
Не наследуется.

Многострочное выравнивание align-content

Свойство выравнивает строки flex-элементов по вертикали во flex-контейнере, позволяя управлять свободным пространством.

Свойство работает только в случае, если разрешен перенос строк и указано направление flex-flow: row/row-reverse/column/column-reverse wrap/wrap-reverse; и высота flex-контейнера.

Не наследуется.

`stretch` – Значение по умолчанию. Строки flex-элементов равномерно растягиваются, заполняя все доступное пространство.

`flex-start` – Строки flex-элементов выравниваются по левому краю flex-контейнера относительно верхнего края блока-контейнера.

`flex-end` – Строки flex-элементов выравниваются по левому краю flex-контейнера относительно нижнего края блока-контейнера.

`center` – Строки flex-элементов выравниваются по высоте по середине flex-контейнера относительно его левого края.

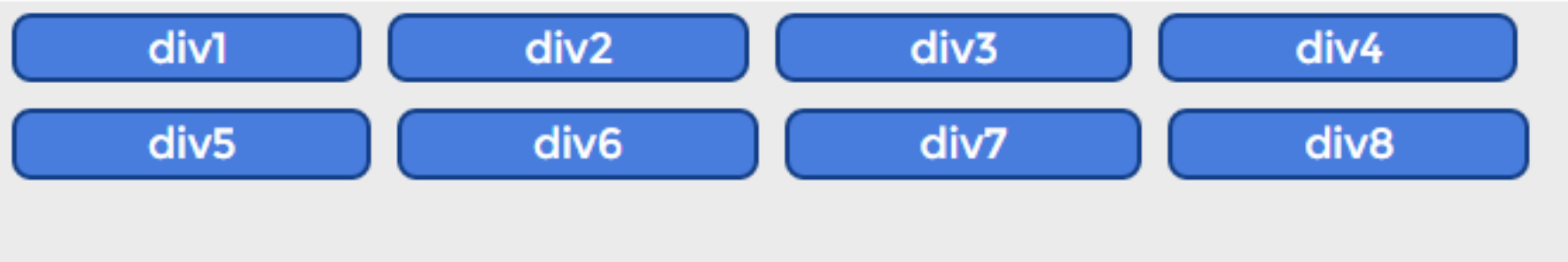
`space-between` – Строки flex-элементов выравниваются по высоте по середине flex-контейнера относительно его левого края.

Свободное пространство распределяется между ними.

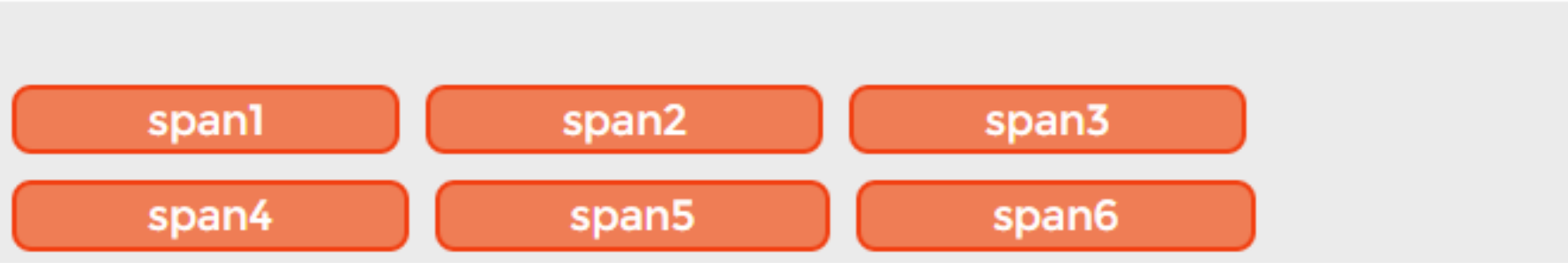
Первый ряд flex-элементов прижимается к началу flex-контейнера, последний ряд — к нижнему краю.

`space-around` - Строки flex-элементов равномерно распределяются по высоте, свободное пространство добавляется сверху и снизу строки.

```
align-content: flex-start;
```

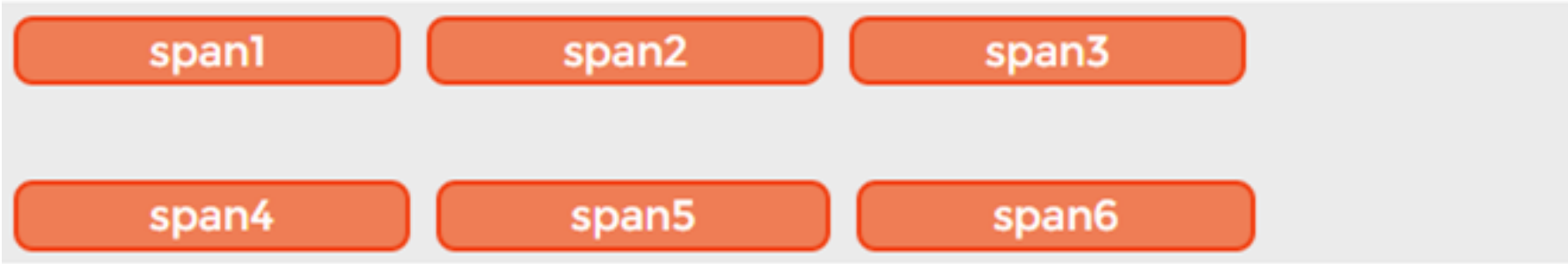



```
align-content: flex-end;
```

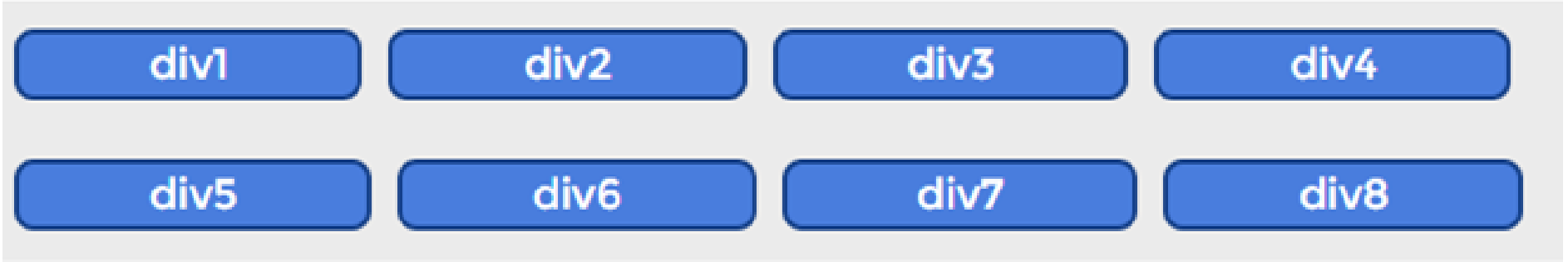


```
align-content: center;
```


```
align-content: space-between;
```



```
align-content: space-around;
```




```
align-content: stretch;
```

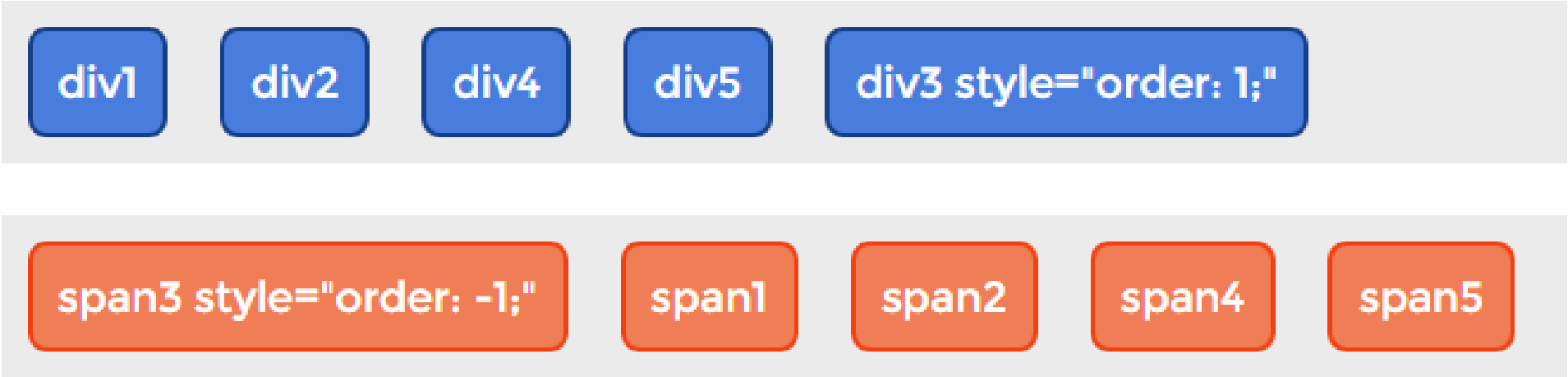
Свойства flex-элементов

Порядок отображения элементов order

Свойство определяет порядок, в котором flex-элементы отображаются внутри flex-контейнера.

По умолчанию для всех flex-элементов задан порядок `order: 0;` и они следуют друг за другом по мере добавления во flex-контейнер. Самый первый flex-элемент по умолчанию расположен слева.

Чтобы поставить любой flex-элемент в начало строки, ему нужно назначить `order: -1;`, в конец строки — `order: 1;`.
Свойство не наследуется.


div1
div2
div4
div5
div3 style="order: 1;"
span3 style="order: -1;"
span1
span2
span4
span5

Базовая ширина элемента flex-basis

Свойство позволяет задать базовую ширину flex-элемента, относительно которой будет происходить растяжение `flex-grow` или сужение `flex-shrink` элемента.

Не наследуется.

`auto` – Значение по умолчанию. Элемент получает базовую ширину, соответствующую ширине контента внутри него, если она не задана явно.

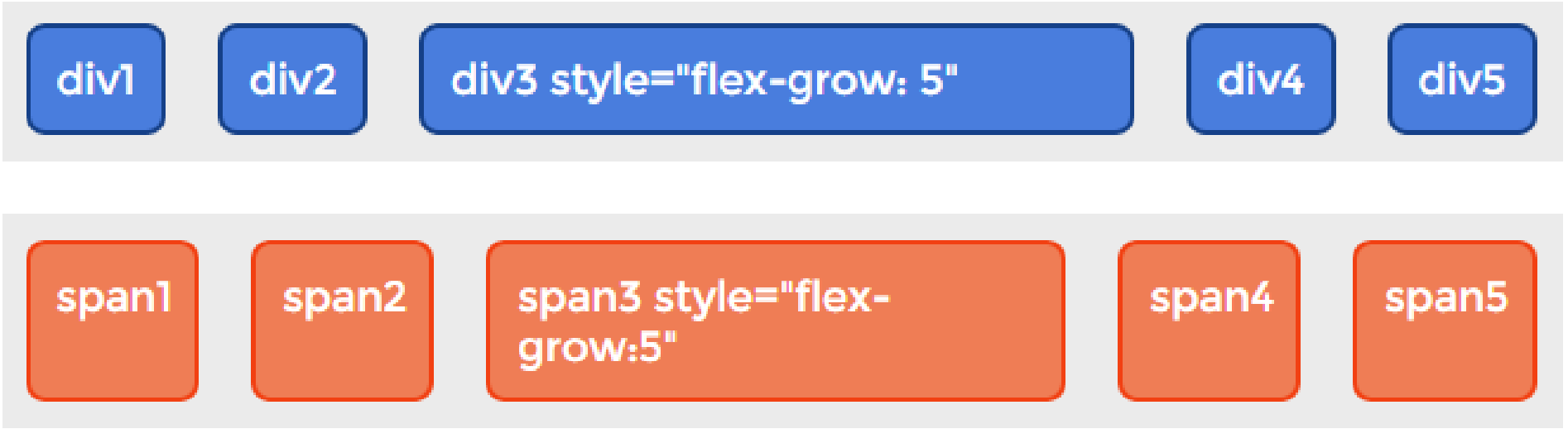
число - Ширина элемента задается в `px`, `%`, `em` и других единицах измерения.

Растяжение элементов flex-grow

Свойство определяет коэффициент увеличения ширины flex-элемента относительно других flex-элементов.
Свойство не наследуется.

число - Положительное целое или дробное число, устанавливающее коэффициент увеличения flex-элемента.

Значение по умолчанию `0`.


div1
div2
div3 style="flex-grow: 5"
div4
div5
span1
span2
span3 style="flex-grow:5"
span4
span5

Сужение элементов flex-shrink

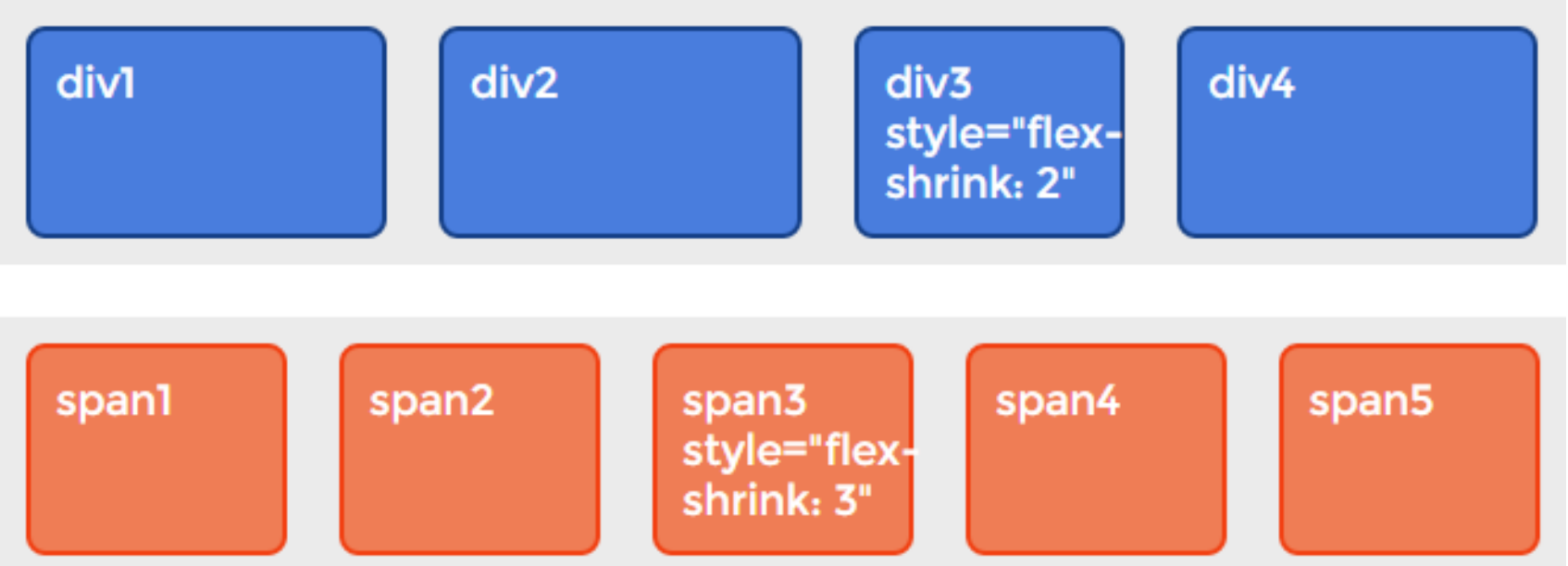
Свойство указывает коэффициент уменьшения ширины flex-элемента относительно других flex-элементов.

Работает только если для элемента задана ширина с помощью свойства `flex-basis` или `width`.

Свойство не наследуется.

число - Положительное целое или дробное число, устанавливающее коэффициент уменьшения flex-элемента.

Значение по умолчанию `1`.



Задание базовой ширины и трансформации элемента одним свойством flex

Свойство представляет собой сокращённую запись свойств `flex-grow`, `flex-shrink` и `flex-basis`.
Значение по умолчанию: `flex: 0 1 auto;`.
Можно указывать как одно, так и все три значения свойств.
Свойство не наследуется.

Выравнивание отдельных элементов align-self

Свойство отвечает за выравнивание отдельно взятого flex-элемента по высоте flex-контейнера.

Переопределяет выравнивание, заданное `align-items`.

Не наследуется.

`auto` - Значение по умолчанию. Flex-элемент использует выравнивание, указанное в свойстве `align-items` flex-контейнера.

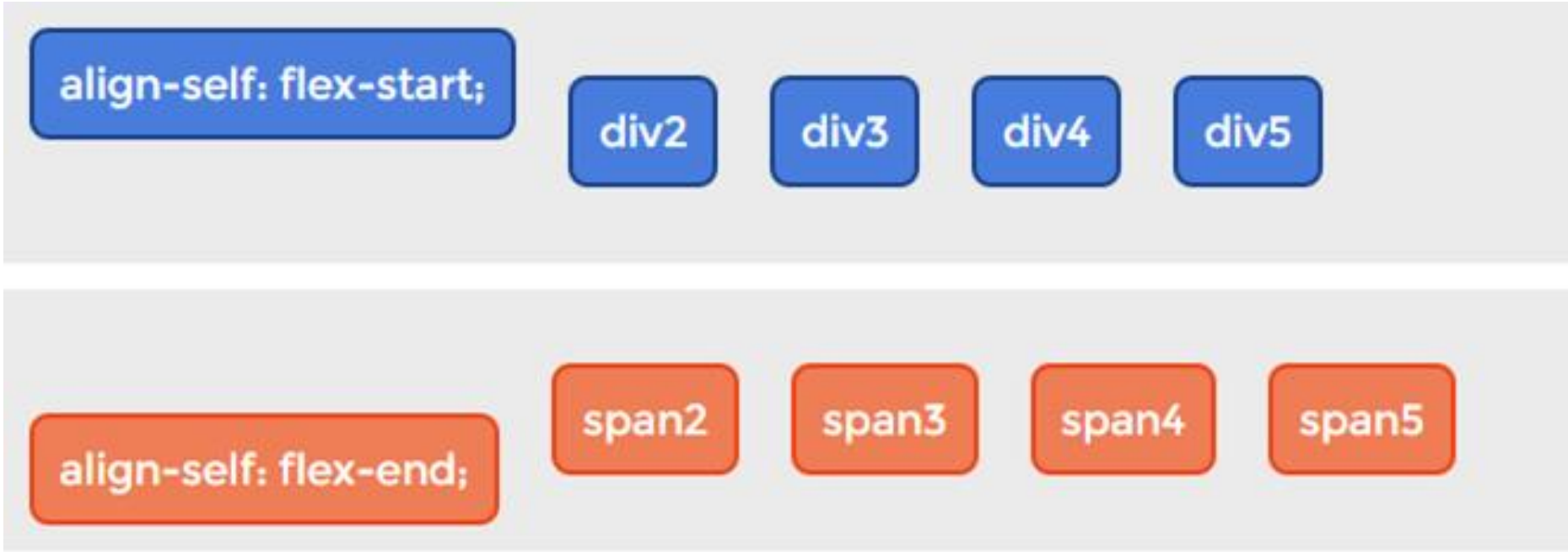
`flex-start` - Flex-элемент выравнивается по верхнему краю flex-контейнера, относительно левой границы.

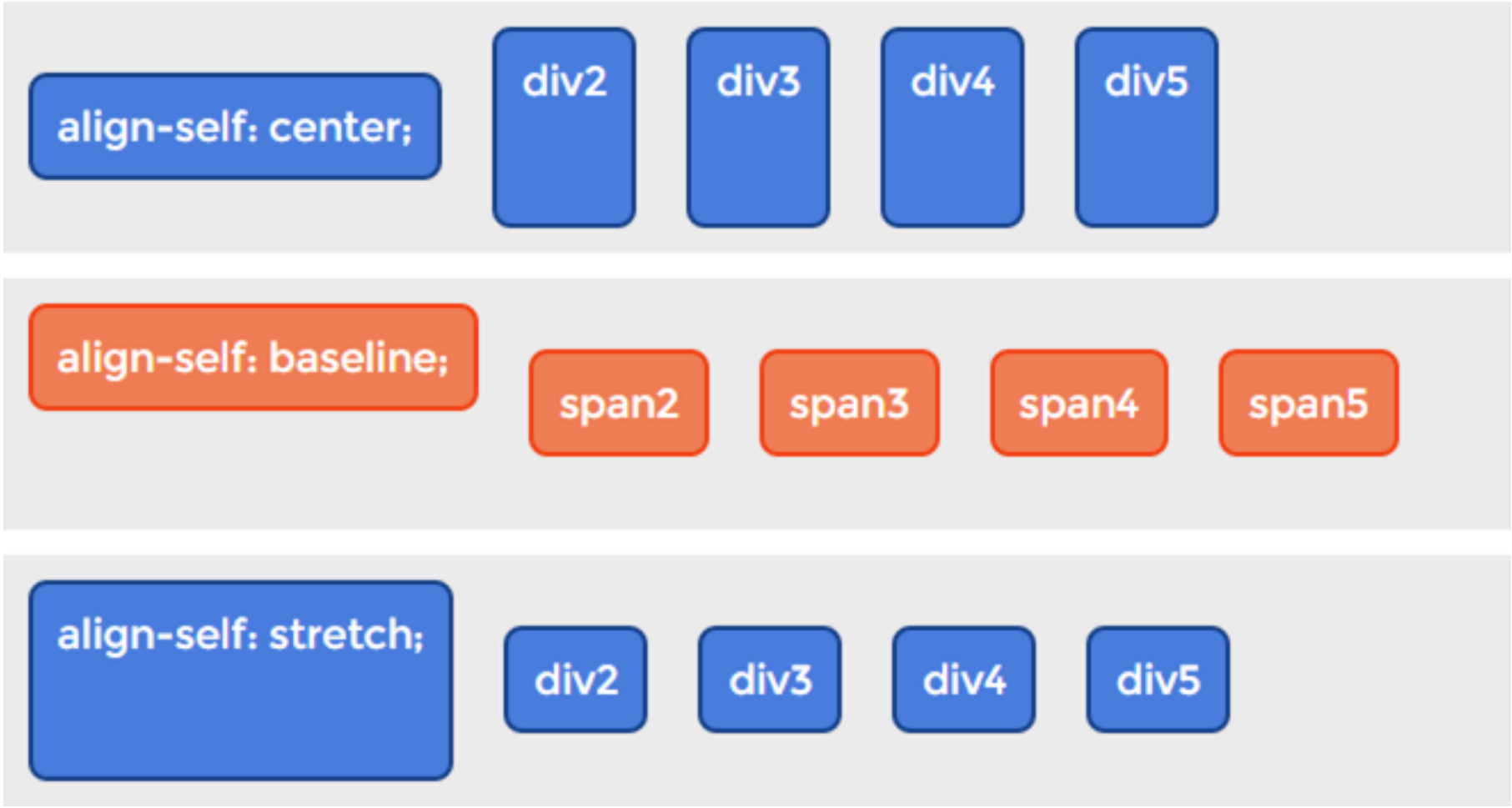
`flex-end` - Flex-элемент выравнивается по нижнему краю flex-контейнера, относительно левой границы.

`center` – Flex-элемент выравнивается по высоте по середине flex-контейнера, относительно левой границы.

`baseline` – Flex-элемент выравнивается по базовой линии flex-контейнера, относительно левой границы.

`stretch` – Flex-элемент растягивается на всю высоту flex-контейнера, при этом учитываются поля и отступы.


align-self: flex-start;
div2
div3
div4
div5
align-self: flex-end;
span2
span3
span4
span5


align-self: center;
div2
div3
div4
div5
align-self: baseline;
span2
span3
span4
span5
align-self: stretch;
div2
div3
div4
div5

Вёрстка на Grid в CSS.

История

Grid модуль в CSS был разработан CSS Working Group для того, чтобы предоставить наилучший способ создания шаблонов в CSS.

Он попал в Candidate Recommendation в феврале 2017 года, а основные бразуеры начали его поддержку в марте 2017 года.

Основная концепция

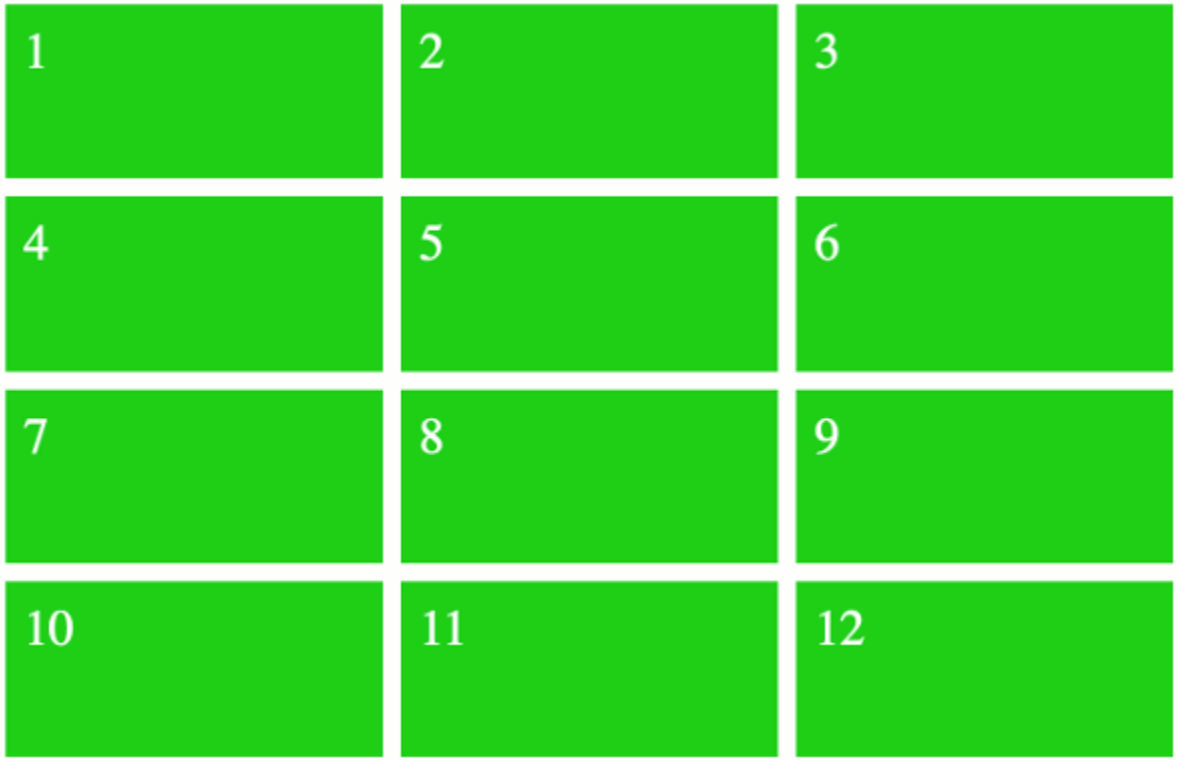
Grid шаблон работает по системе сеток.

Grid это набор пересекающихся горизонтальных и вертикальных линий, которые создают размеры и позиционируют систему координат для контента в grid-контейнере.

Чтобы создать Grid, вам просто нужно выставить элементу display: grid. Это автоматически сделает всех прямых потомков этого элемента — grid элементами.

Обычно первым шагом является определение того, сколько колонок и рядов есть в grid. Но даже это опционально.

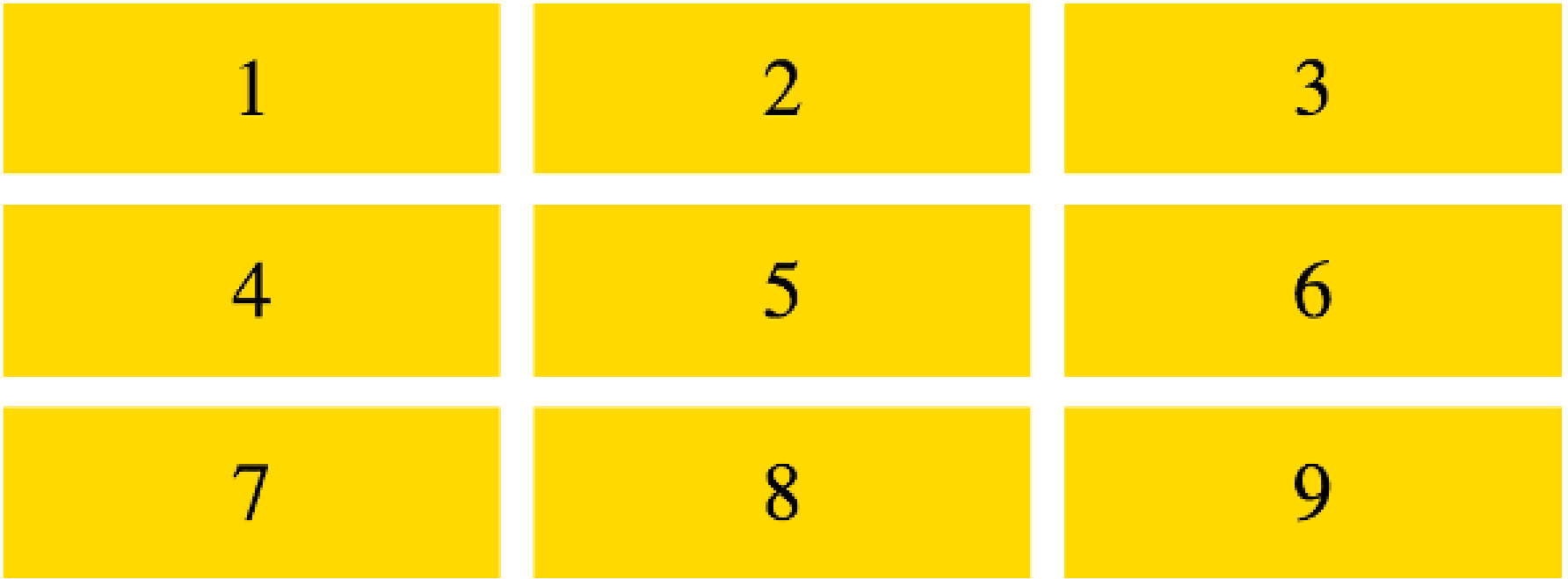
Этот грид - 12 grid элементов. Каждый из этих элементов зеленый и между ними есть небольшое расстояние.



Все эти `grid` элементы одного размера, но они могли бы быть любого размера, какого необходимо.

Некоторые могли бы охватывать несколько столбцов и рядов, другие могли бы оставаться размеров с одну ячейку.

Создаем Grid



```
1. <div id="grid">
2.  <div>1</div>
3.  <div>2</div>
4.  <div>3</div>
5. ...
6.  <div>7</div>
7.  <div>8</div>
8.  <div>9</div>
9. </div>
```

Внешний `<div>` это контейнер гридов.

Соотвественно, все элементы вложенные в него будут являться грид элементами.

Но по-факту, это не будет полноценными гридом, пока мы не применим кое-какой CSS для него. (см. класс grid)

```
<style>
#grid {
  display: grid;
  grid-template-rows: 1fr 1fr 1fr;
  grid-template-columns: 1fr 1fr 1fr;
  grid-gap: 2vw;
  }
#grid > div {
  font-size: 5vw;
  padding: .5em;
  background: gold;
  text-align: center;
}
</style>
```

`display: grid -` Превращает элемент в grid контейнер. Это все, что нужно для того, что создать грид. Теперь есть грид-контейнер и грид-элементы. Значения гридов создают блочный контейнер. Также можно использовать display: inline-grid, что создать строчный грид-контейнер. Или использовать display: subgrid, чтобы создать подсетку, это значение используется на самих grid элементах.

`grid-template-rows: 1fr 1fr 1fr -` Выстраивает ряды в гриде. Каждое значение представляет размер ряда. В этом случае все значения равны 1fr.

Но конечно же для этого можно было бы использовать разные значения, такие как `100px`, `7em`, `30%` и так далее. Вы также можете назначать имена строкам вместе с их размерами.

`grid-template-columns: 1fr 1fr 1fr -` Тоже самое, что и выше, только определяет колонки в гридах.

`grid-gap: 2vw -` Выставляет разрыв. То есть пробелы между грид элементами. Тут исп. vw единицу длины, которая относительна ширине viewport, но также мы можем использовать `10px`, `1em` и т. д.

`Grid-gap` свойство это сокращение для grid-row-gap и

Функция repeat()

Можно использовать функцию repeat() для повторяющихся объявлений значения размера элемента.

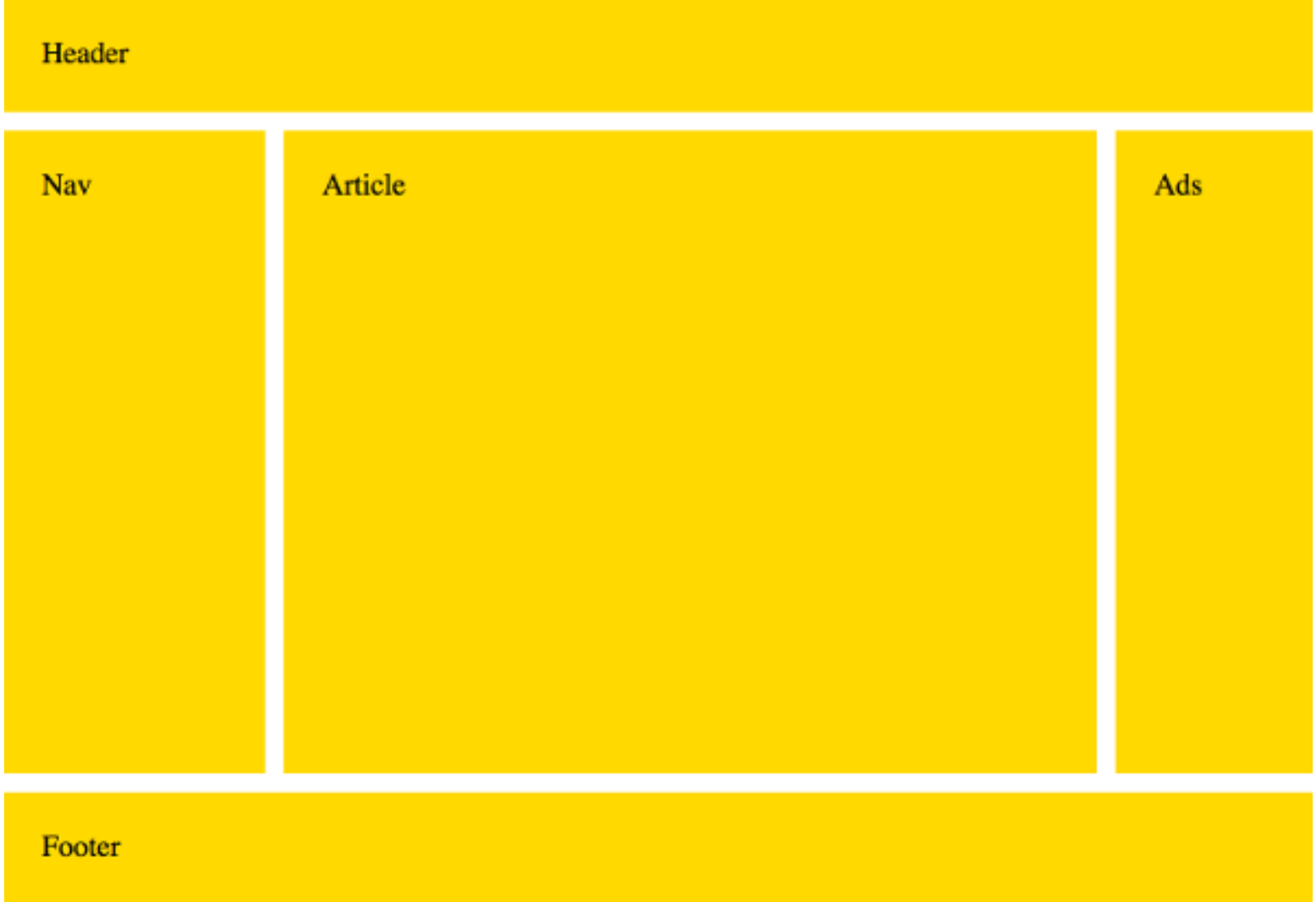
Для примера, вместо того, чтобы делать это: `grid-template-rows: 1fr 1fr 1fr 1fr 1fr;`

Слдедующим образом:

```
grid-template-rows: repeat(5, 1fr);
```

Создание шаблона сайта с CSS Grid

Гриды включают в себя интуитивный «ASCII-графический» синтаксис, в котором вы можете виртуально «видеть» шаблон в коде, по-этому становится очень легко создавать и изменять ваш шаблон. Даже значительные изменения могут быть сделаны за несколько секунд. Этот интуитивный синтаксис также помогает с адаптивным веб-дизайном. Создание разных шаблонов для разных устройств становится довольно тривиальным делом при использовании гридов.


Header
Nav
Article
Ads
Footer

HTML разметка выглядит таким образом:

```
<body>

<header id=”pageHeader”>Header</header>

<article id=”mainArticle”>Article</article>

<nav id=”mainNav”>Nav</nav>

<div id=”siteAds”>Ads</div>

<footer id=”pageFooter”>Footer</footer>

</body>
```

```
body {
display: grid;
grid-template-areas:
“header header header”
“nav article ads”
“footer footer footer”;
grid-template-rows: 60px 1fr 60px;
grid-template-columns: 20% 1fr 15%;
grid-gap: 10px;
height: 100vh;
margin: 0;   }
```

```
header, footer, article, nav, div {
padding: 20px; background: gold;   }
#pageHeader {    grid-area: header;     }
#pageFooter  {     grid-area: footer;      }
#mainArticle   {    grid-area: article;      }
#mainNav       {     grid-area: nav;       }
#siteAds         {     grid-area: ads;       }
```

А теперь давайте посмотрим на ASCII-графику, о которой мы говорили прежде.

```
grid-template-areas:
“header header header”
“nav article ads”
“footer footer footer”;
```

Этот кусок определяет наш шаблон. Просто смотря на код, мы можем видеть, что это 3x3 грид (три ряда и три колонки).

Таким образом у нас получается пять грид областей на девяти грид ячейках, так как некоторые грид-области занимают несколько ячеек.

Хедер занимает весь первый ряд в три ячейки, а футер занимает весь нижний ряд, также забирая три ячейки. Навигационная, контентная и рекламная секции, все вместе делят место во втором ряду, где каждому из этих элементов достается по одной ячейке.

Теперь необходимо назначить каждую из этих грид-областей каждому элементу

Свойство `grid-area` это сокращение свойства, которое позволяет вам размещать грид-элементы в гриде.

В нашем случае, мы просто отсылаемся к названиям, которые мы предварительно указали в grid-template-areas.

Остаток кода относится к размерам, пробелам и высотам, в общем скорее к декоративной области.

Следующий код выдает размеры строкам и колонкам:

```
grid-template-rows: 60px 1fr 60px;
grid-template-columns: 20% 1fr 15%;
```

Первая и третья строки — обе в `60px` высотой, а вторая строка забирает все оставшееся место.

Первый столбец равен `20%`, а третий `15%`. Второй же забирает все оставшееся место.