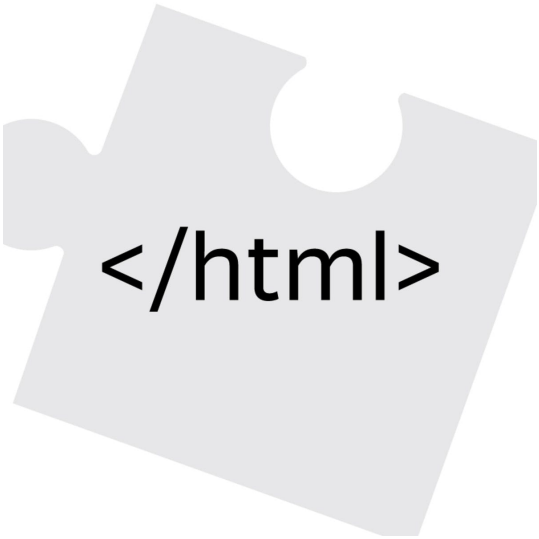
Модель отображения: раскладка страниц

Курс: Тонкости верстки

19 июля 2018 г.



Оглавление

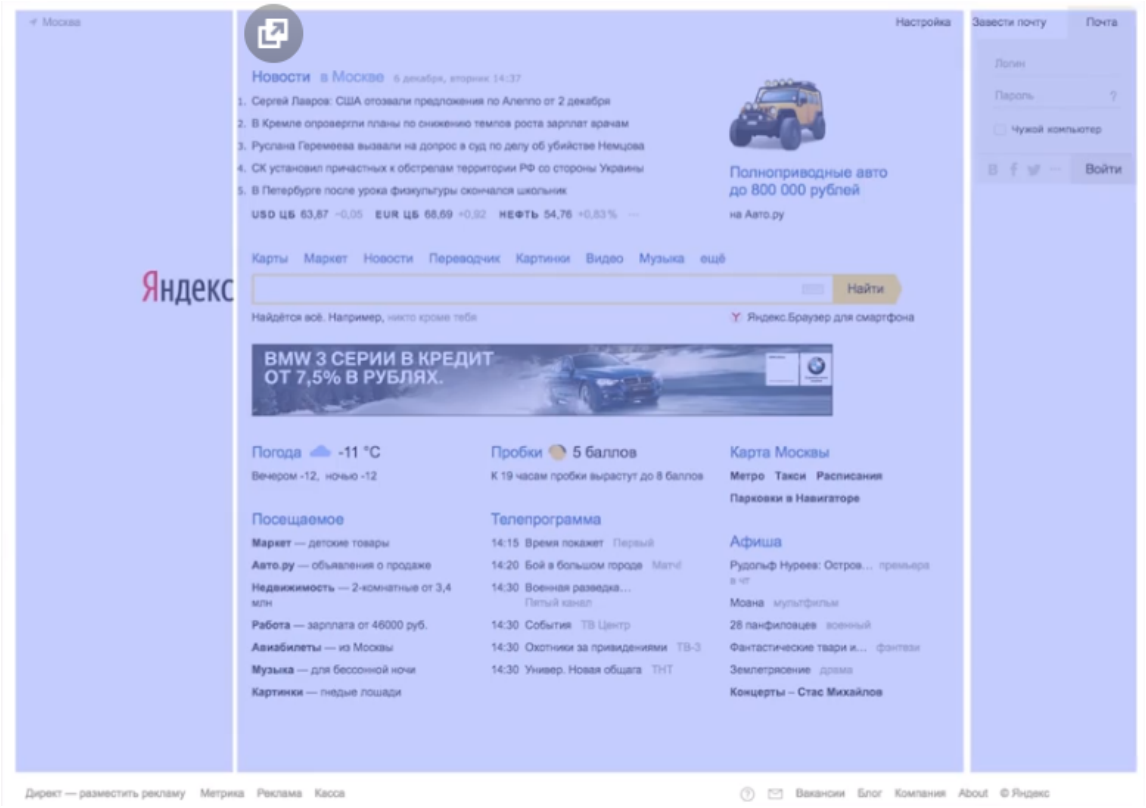
Глава 3

Раскладка страниц

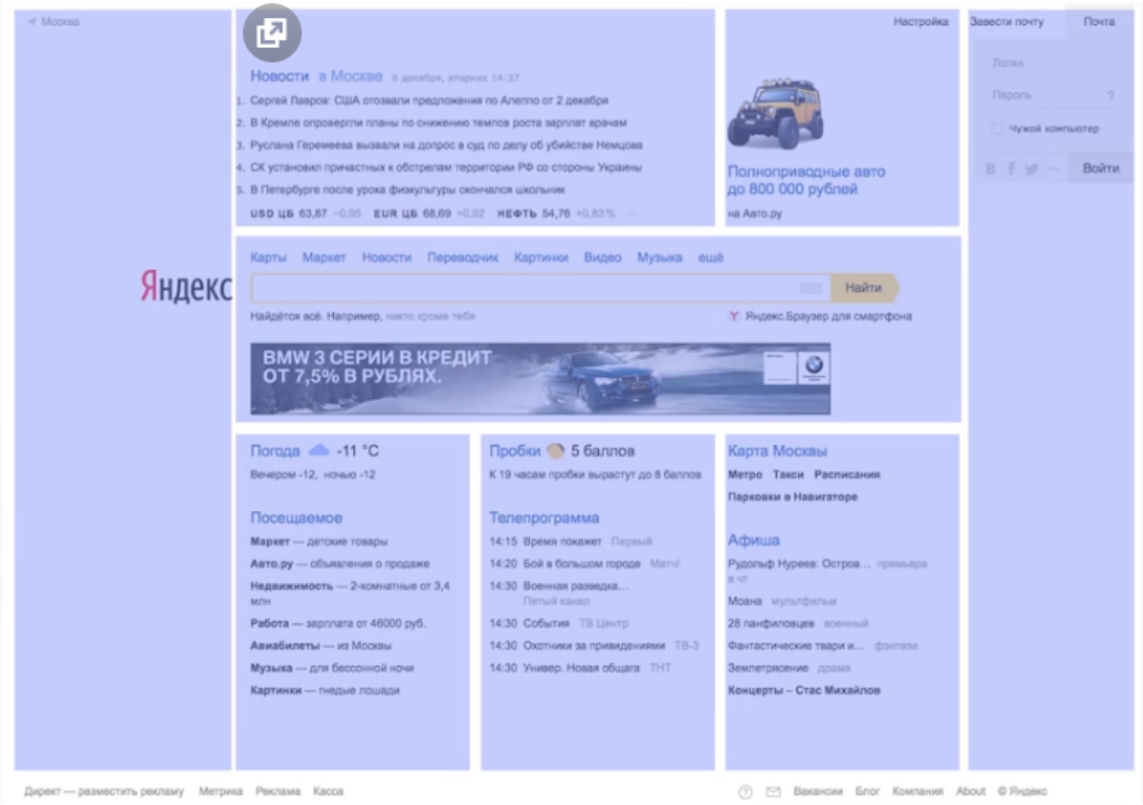
3.1. Разметка

3.1.1. Разметка таблицами

Рассмотрим главную страницу "Яндекса". Здесь я могу выделить три колонки: левая с логотипом, средняя с основым содержимым, и правая колонка. Раньше такую структуру реализовывали таблицами. На первый взгляд здесь три колонки и три ячейки.



Если вы посмотрите внимательнее, то нижняя часть — это тоже ячейки. И на самом деле ячеек снизу уже пять. Если посмотреть внимательно наверх, то там тоже не одна ячейка, а две. Правая, где у нас находится баннер, и левая, где у нас находятся новости.



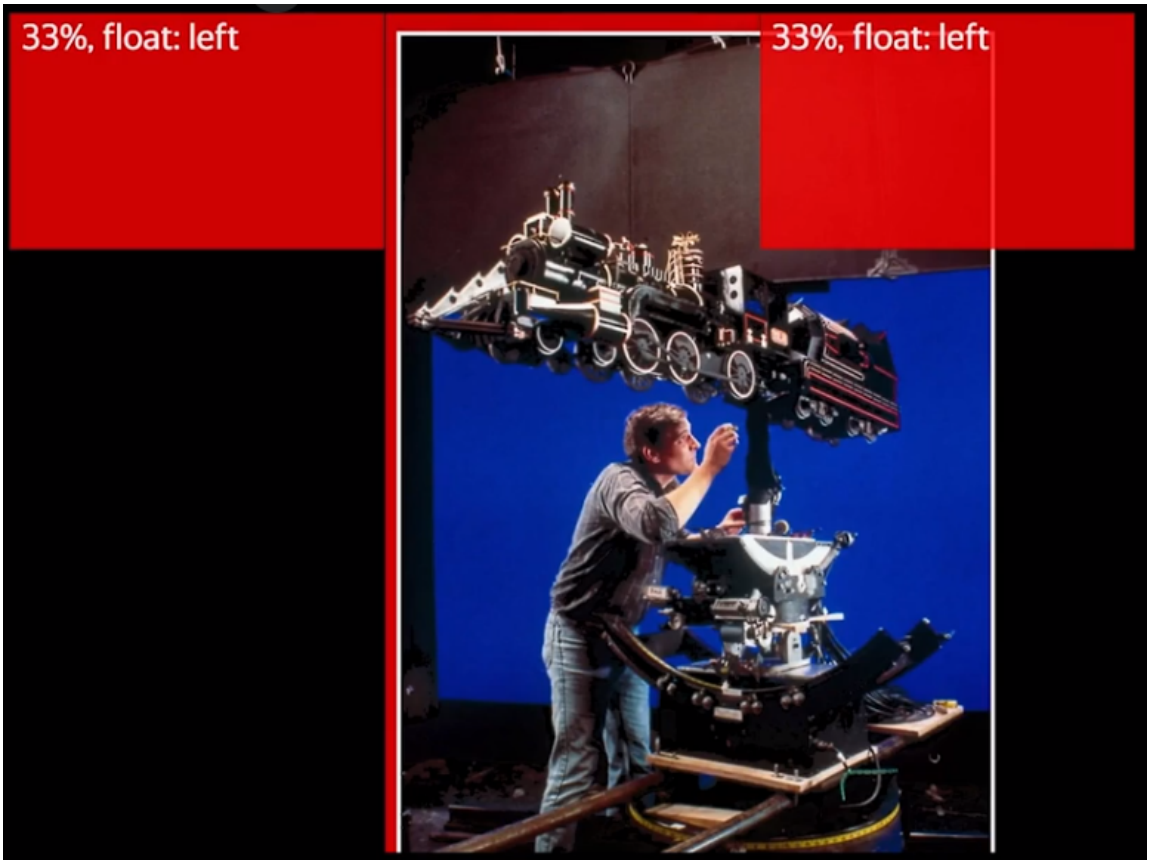
Сделать такую разметку с помощью таблиц несложно. Есть атрибут, который позволяет группировать ячейки `colspan`. Но есть еще и строки, которые нужно сгруппировать, в левом и правом столбцах. За это отвечает атрибут `rowspan`. Уже сейчас, глядя на эту разметку, можно понять, что ее очень неудобно поддерживать. Например, если мы захотим добавить еще одну ячейку в нижний уровень, то нам нужно будет менять атрибуты `colspan` и `rowspan` в некоторых местах. Можно также воспользоваться вложенными таблицами, но тогда мы будем терять согласованность между строками.

3.1.2. Разметка `div`'ами

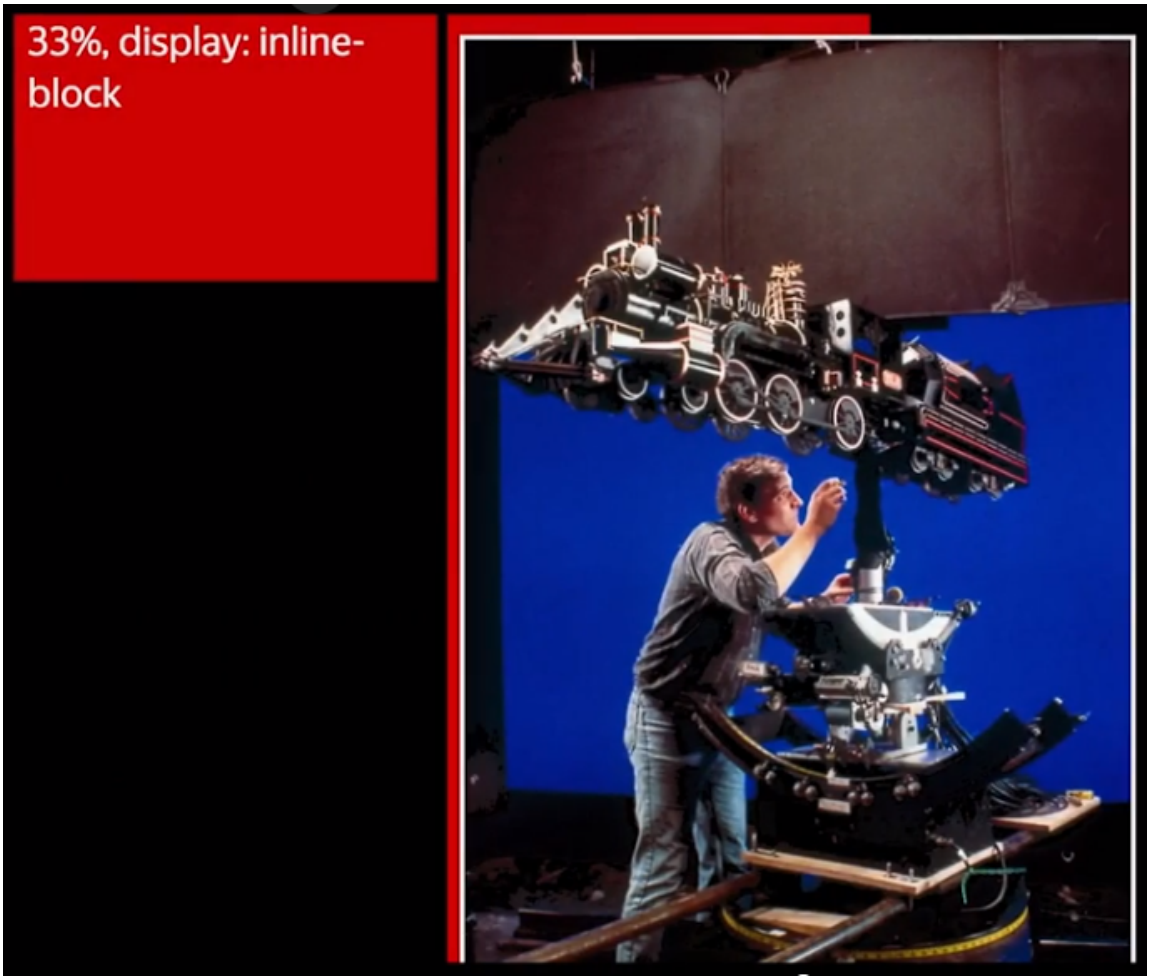
Сейчас все поменялось на `div`, и расставлять элементы стали с помощью CSS. В чем же тут проблема? Чтобы ее увидеть, давайте рассмотрим несколько реализаций разметки, и как они ведут себя в зависимости от содержимого.
Рассмотрим, например, трехколоночную разметку, где колонки имеют одинаковую ширину. Эталонная реализация такой разметки с помощью таблиц. Так

как если мы поместим в среднюю колонку картинку, которая по ширине превышает размеры колонки, то таблица поведет себя максимально корректно. Крайние колонки немножко подожмутся по ширине, и средняя колонка вместит картинку полностью.

Если точно такую же разметку реализовать, например, с помощью плавающих блоков, то правая колонка наползает на картинку, и часть контента мы просто не увидим.



Другой пример — реализация трехколонника с помощью `inline`-блоков. Здесь мы вообще не видим правой колонки, она перенеслась на другую строчку.



3.1.3. Основные проблемы разных разметок

Какие же основные проблемы раскладок с помощью различных CSS-свойств?

- плавающая разметка: нужно применять хаки типа `clearfix`, не всегда разметка ведет себя корректно в зависимости от контента;
- абсолютное позиционирование: смежные элементы не накладываются друг на друга, но они выводятся из потока и не могут быть связаны друг с другом;
- таблицы или свойство `display: table`: очень много лишней разметки;

- `inline`-блоки: нестабильное поведение разметки, нужно что-то будет делать с пробелами, которые могут появиться между блоками.

3.2. Спецификация `flexbox`

`flexbox` — это спецификация, которая описывает набор правил, позволяющих контролировать размер, порядок и выравнивание элементов по нескольким осям, а также распределение свободного места между элементами и многое другое.

`flexbox` — это первая система для раскладки, которая не является хаком.

Основные преимущества `flexbox`:

- элементы могут сжиматься и растягиваться по заданным правилам, занимая нужное пространство;
- во `flexbox` реализовано выравнивание по вертикали и горизонтали «из коробки»;
- расположение элементов в html теперь не имеет решающего значения. Вы можете написать их в одном порядке, а отрендерить в другом;
- элементы могут автоматически выстраиваться в несколько строк или столбцов, занимая все предоставленное место;
- локализация для языков с другим написанием;
- просто и понятно;
- поддержка у `flexbox` действительно большая. Фактически все последние версии всех современных браузеров уже поддерживают `flexbox`.

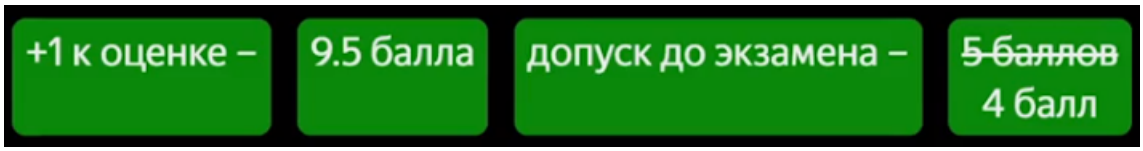
Чтобы начать пользоваться `flexbox`, нужно написать всего одно свойство: `display: flex`.
Допустим, у нас есть абстрактная система допуска до экзамена, в которой 9,5 баллов дает +1 к оценке на экзамене, а 5 баллов дает допуск до экзамена. Такая верстка может выглядеть как-то так. Здесь я добавил дополнительных украшений и скруглений уголков, чтобы выглядело красиво.

```
<div> +1 к оценке -
 ↪  </div>
<div> 9.5 баллов </div>
<div> допуск до экзамена
 ↪  - </div>
<div> 5 балоов </div>
```

Что будет, если мы обертке всех этих элементов зададим свойство flex? Все начнет выглядеть вот так вот.

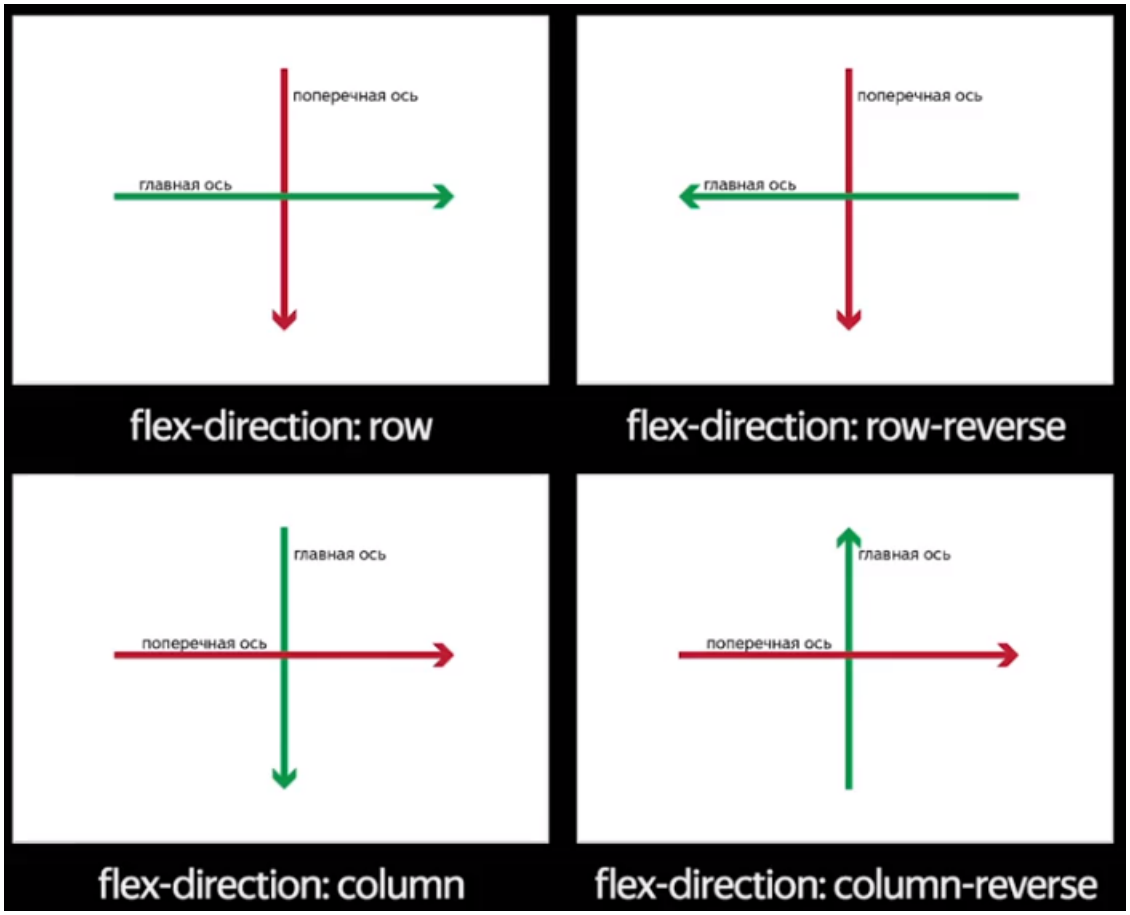
```
.wrapper {
display: flex;
}
```

Например, в четвертом элементе мы хотим изменить значение, и теперь у нас для того, чтобы допуститься до экзамена, нужно не 5 баллов, а 4. Что произошло: все элементы автоматически подстроили свою высоту под высоту последнего элемента. Это первое замечательное свойство `flex`-элементов. Высота подстраивается под максимальную высоту.



3.2.1. Оси и работа с ними

Основу спецификации `flexbox` составляют оси. В любой `flex`-разметке этих осей две: главная ось и поперечная ось, при этом поперечная ось всегда перпендикулярна главной. Главная ось задает направление выстраивания элементов, а поперечная ось задает направление роста элементов.
Важно понимать, что эти направления и оси можно менять. За это отвечает свойство `flex-direction`. По умолчанию `flex-direction` равно `row`, но мы можем задать `row-reverse`, и тогда элементы будут выстраиваться не слева направо, а справа налево. Мы также можем вообще поменять эти оси местами и сказать, что у нас теперь элементы выстраиваются сверху вниз, а растут слева направо. Ну и, наконец, мы можем и это поведение развернуть.



Давайте рассмотрим на конкретных примерах. Разных значениях свойства `flex-direction`. Как вы видите, положение элементов не соответствует их положению в `html`. Это один из способов поменять порядок элементов.

Следующее свойство задает выравнивание вдоль главной оси. Оно называется `justify-content`. Значение по умолчанию — `flex-start`, то есть элементы выстраиваются к началу главной оси.

Какие же еще значения есть у `justify-content`? `flex-end`, `center` — все элементы выстраиваются посередине. Но есть два замечательных свойства — это `space-between`, чем-то поведение этого свойства похоже на значение `justify` свойства `text-align`, за исключением того, что нам не нужно вставлять абстрактный последний элемент, который будет задавать перенос:

и `space-around`, когда у нас пространство есть не только между средними элементами, но и от края содержащего элемента.

Следующее свойство — `align-items`, оно, в противовес `justify-content`, задает выравнивание вдоль поперечной оси. По умолчанию значение — `stretch`, то есть блоки растянуты и занимают все доступное пространство вдоль поперечной оси. Именно поведение значения свойства `align-items: stretch` мы наблюдали, когда увеличивали высоту крайнего элемента.

Давайте же рассмотрим другие значение свойства `align-items`. `flex-start` — в таком случае у нас элементы поджимаются по контенту.

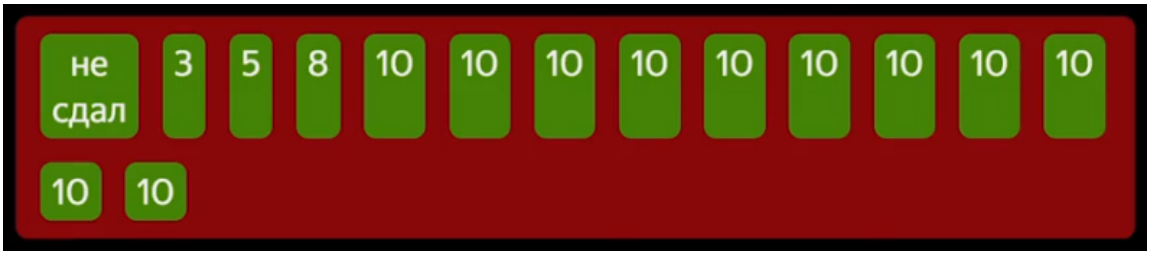
`flex-end` — в данном случае их содержимое не только поджимается, но еще и выравнивается к концу поперечной оси. Также есть `center`

и `baseline`. `baseline` — это самое интересное значение, здесь мы, например, можем задать `padding`, и тогда у нас все элементы все равно будут выстроены по базовой линии.

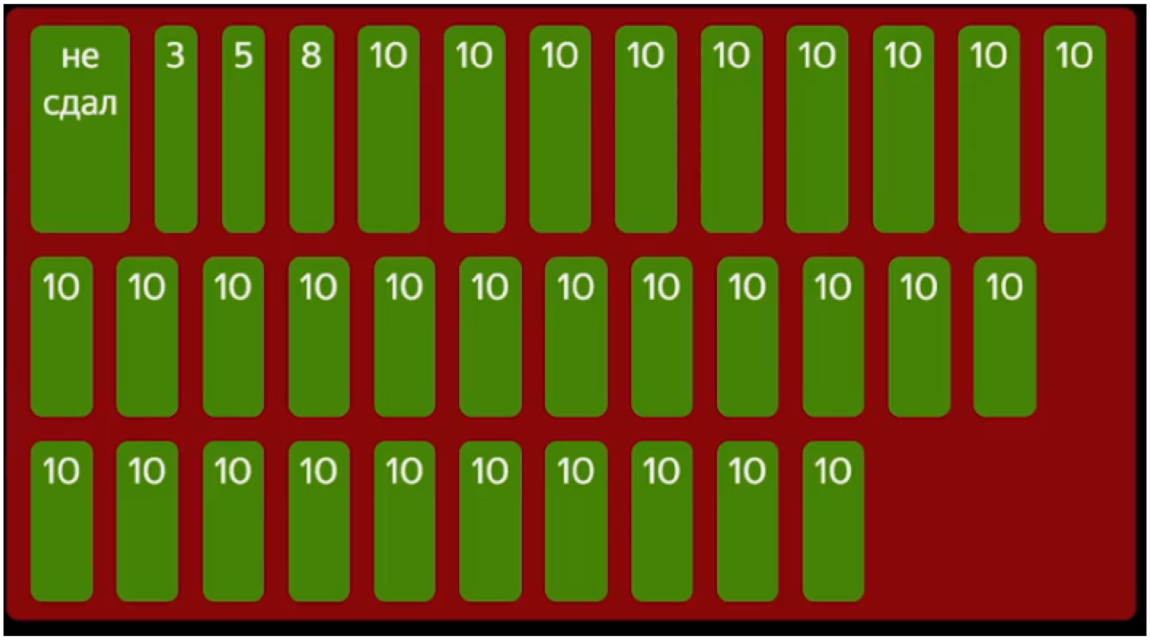
3.2.2. Многострочность

Следующее свойство `flexbox`-элементов — это многострочность. По умолчанию, если ширина элементов внутри родителя будет больше, чем ширина родителя, то вы увидите вот такое поведение, то есть элементы будут продолжаться и за пределами родителя.

Так происходит потому, что свойство `flex-wrap`, которое отвечает за режим форсирования выстраивания блоков в одну линию, по умолчанию имеет значение `nowrap`. Но мы можем это поведение поменять и сфорсировать перестраивание блоков. Например, мы можем сказать `flex-wrap: wrap`, и тогда, если элементы не влезают на строчку, они переносятся на следующую строчку.



У `flex-wrap` есть и значение `wrap-reverse`, когда элементы переносятся на предыдущую строчку.
Следующее свойство — `flex-flow`. Это комбинация значений свойств `flex-direction` и `flex-wrap`, то есть группирующее свойство.
Следующее свойство — `align-content`, определяет то, каким образом образовавшиеся при многострочности ряды блоков будут выровнены по вертикали и как поделят между собой все пространство `flex`-контейнера. `align-content` не работает, если у `flex`-контейнера не задана высота. По умолчанию значение `stretch` — ряды блоков растянуты и занимают все имеющееся пространство.



Мы можем задать `flex-start`, `flex-end` или `center` — поведение, уже знакомое вам, поэтому не будем останавливаться на этом подробнее. Также мы можем задать `space-between` и `space-around`, также поведение очень похоже на `justify-content`.

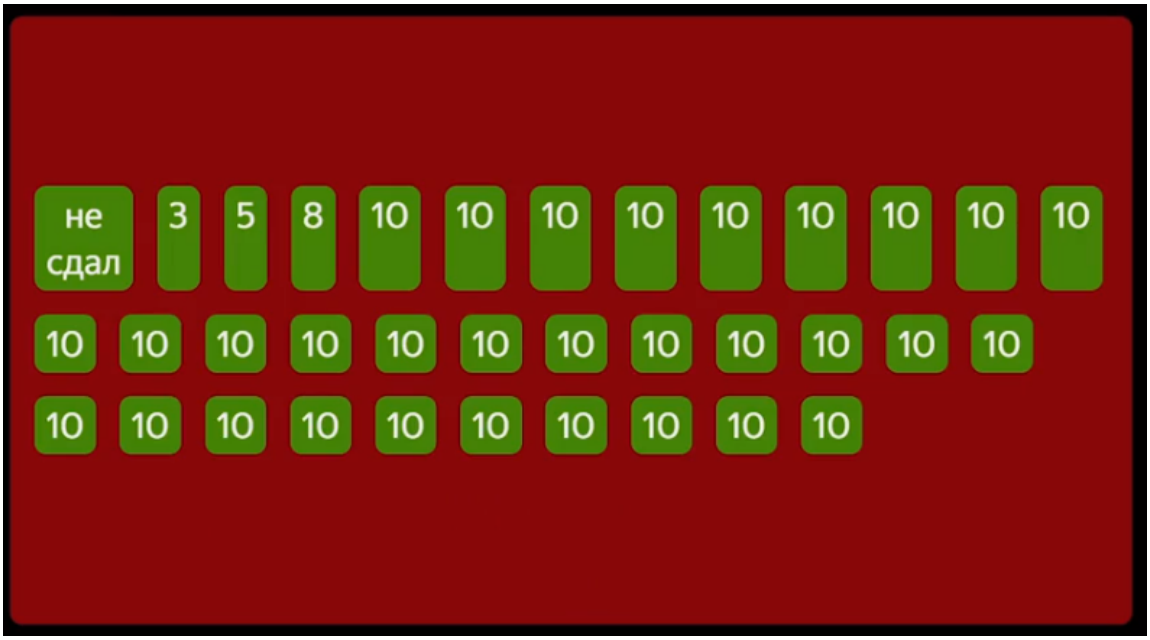
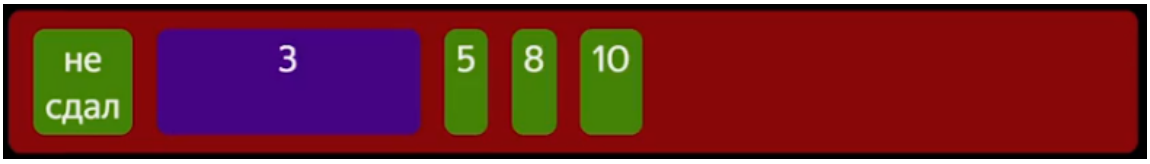


Рис. 3.1: `align-content: center`

3.2.3. Изменение свойств дочерних элементов

Самое замечательное в спецификации `flex`: мы можем менять поведение для конкретных элементов `flex`-контейнера. Например, мы можем какому-то конкретному элементу задать выравнивание только для него. За это отвечает свойство `align-self` с уже знакомыми свойствами `flex-start`, `center` и так далее.

Следующее свойство — `flex-basis`. Оно задает изначальный размер элемента по главной оси, до того, как к нему будут применены различные другие преобразования. Например, мы можем сказать flex-basis: 200px. В данном случае этот элемент будет иметь действительно ширину 200 пикселей, потому что пространство позволяет ему иметь эту ширину.

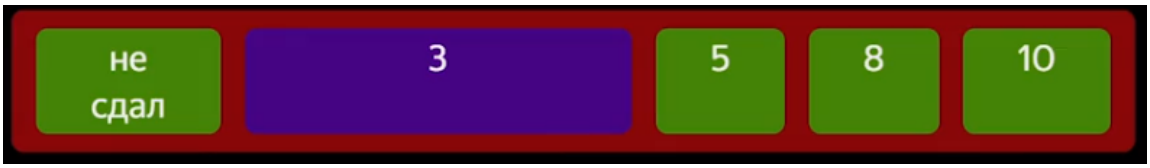


Мы можем задать `flex-basis: 600px`, а можем задать `1000px`. Но 600 от 1000

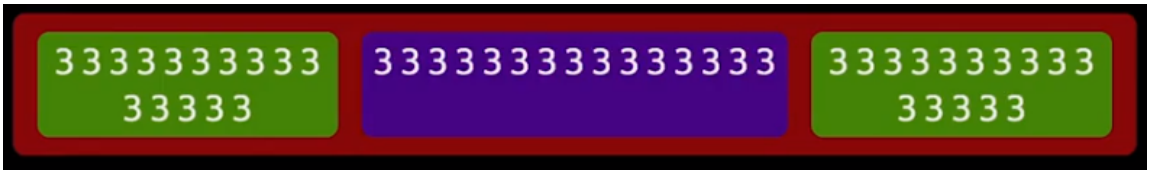
ничем не отличается, потому что дальше у нас были применены другие преобразования, которые, например, говорят, что все элементы должны быть растянуты вдоль поперечной оси. Ну и, соответственно, у нас ширина не применилась.

Еще два свойства, которые введены в спецификации `flexbox`, отвечают за "жадность" и "бедность". Соответствующие свойства называются `flex-grow` и `flex-shrink`. `flex-grow` определяет то, насколько отдельный блок может быть больше соседних блоков. Например, в данном случае мы можем сказать фиолетовому блоку `flex-grow: 2`. Если мы всем остальным блокам не зададим значения свойства `flexbox`, то этот элемент займет все доступное пространство.
Если же мы всем элементам зададим `flex-grow: 2`, то здесь уже какие-то дополнительные вычисления применяются к блокам.

Или, например, мы можем задать всем элементам `flex-grow: 2`, а фиолетовому блоку — `flex-grow: 4`, и тогда он всегда будет стараться быть в два раза больше, чем все элементы.



В обратную сторону работает свойство `flex-shrink`. Оно определяет, насколько `flexbox` будет уменьшаться относительно соседних элементов внутри flex-контейнера. В данном случае мы можем задать среднему элементу `flex-shrink: 0`, и тогда оно будет работать следующим образом.



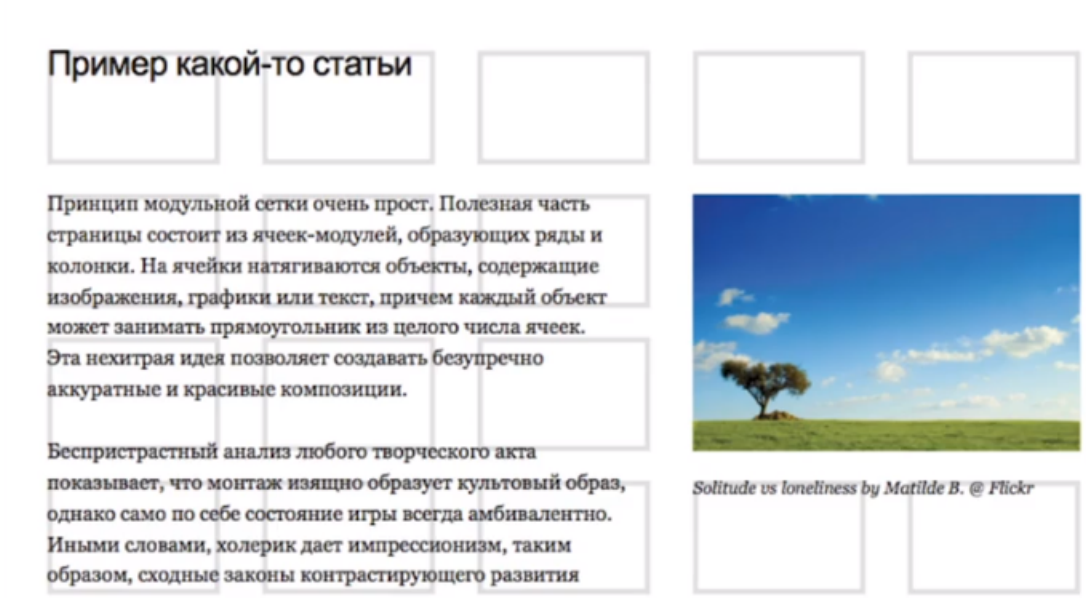
Последнее свойство, которое есть в спецификации `flexbox`, позволяет определить порядок конкретного элемента. За это отвечает свойство `order`. По умолчанию оно равно 0. Этот порядок для любого элемента можно менять. Например, вы можете сказать второму элементу `order: 1`. И тогда у вас сначала выведутся все блоки с `order: 0`, а потом со свойством `order: 1`. И так далее, вы можете менять и ставить элемент в любое место вашей разметки.

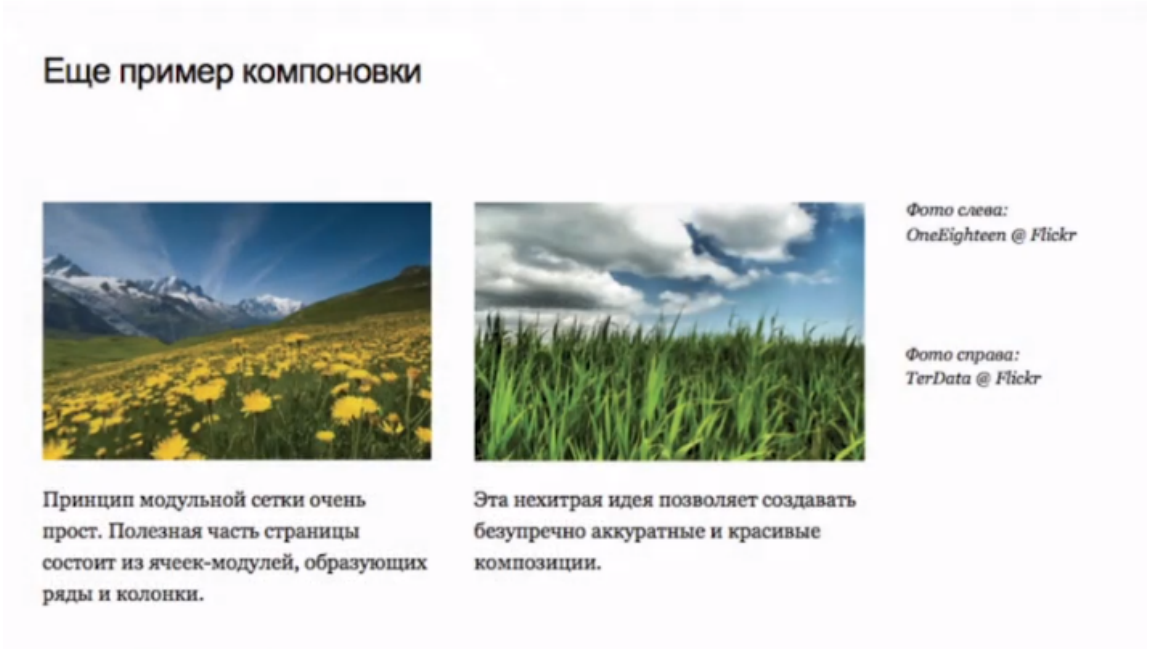
3.3. Спецификация `grid`

Недостатки `flexbox`:

- вертикально расположенные элементы никак не связаны друг с другом, например, мы не можем выровнять их границы;
- порядок элементов требует дополнительной договоренностей, если не обозначить их сразу же, то верстка станет неконтролируемой и сложной в поддержании.

Еще один недостаток незаментен сразу, так как он относится к области дизайна, а именно — к модульной системе верстки. Рассмотрим ее на примере. Для того, чтобы ее проиллюстрировть, достаточно этих двух картинок с наложенными на них сетками:





Текст и картинки выстраиваются по колонкам и столбцам.

Модульная система верстки — система верстки, при которой основой композиции полос и разворотов становится модульная сетка с определенным шагом (модулем), одинаковым или разным по горизонтали и вертикали.

В CSS есть спецификация, которая позволяет реализовать эту систему, она называется CSS Grid Layout. Эта спецификация реализована и работает во всех браузерах.
При `grid`-разметке содержимое контейнера позиционируется и выравнивается относительно сетки.

Сетка — набор пересекающихся вертикальных и горизонтальных линий, которые делят контейнер на зоны, внутри которых располагаются компоненты верстки.

3.3.1. Строки и столбцы

Для задания строк и столбцов в Grid Layout используются свойства `grid-template-columns` и `grid-template-rows`.
В данном случае мы задаем 3 строки и 5 столбцов с фиксированными размерами:

```
.layout {
display: grid;
grid-template-columns: 100px 10px 100px 10px 100px;
grid-template-rows: 20px 10px 30px;
}
```

Главное преимущество `grid`-сетки — в задании сетки, подстраивающейся под контент.
Пример. Здесь мы используем два новых значения: знакомое нам `auto`, автоматически выстраивающее ширину под контент, и fr, иначе flex factor. Элемент со значением 2fr всегда будет в два раза больше, чем элемент со значением 1fr.

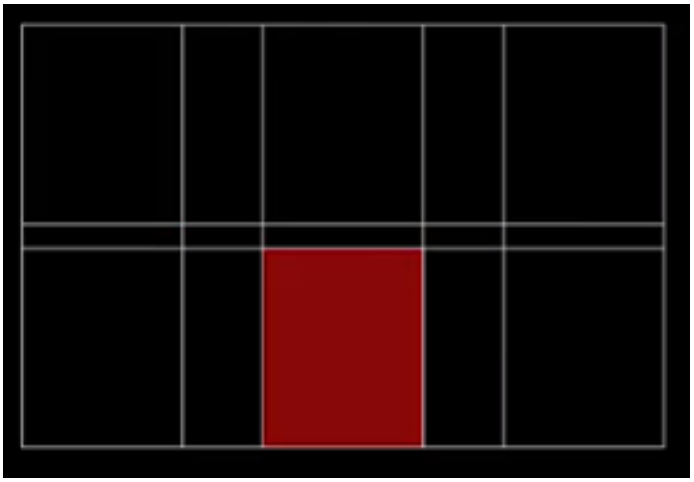
```
.layout {
display: grid;
grid-template-columns: 100px 1fr 100px 2fr 100px;
grid-template-rows: auto 10px auto;
}
```

Для того, чтобы задать `grid`-разметку, нам нужно обертке задать свойство `display: grid`. Чтобы указать положение элемента, воспользуемся свойствами `grid-column-start, grid-column-end, grid-row-start, grid-row-end`.

```
<div class="wrapper">
<div class="a">a</div>
<div class="b">b</div>
<div class="c">c</div>
<div class="d">d</div>
<div class="e">e</div>
<div class="f">f</div>
</div>

.wrapper {
display: grid;
}

.a {
grid-column-start: 1;
grid-column-end: 2;

grid-row-start: 1;
grid-row-end: 2;
}
```

Значения данных свойств — это не соответствующие колонки, а левые или верхние границы соответствующих элементов.
Также мы можем поместить элемент в середину. В данном случае он начинается у левой границы третьей колонки и заканчивается у левой границы четвертой колонки, аналогично со строками.

```css
.b {
grid-column-start: 3;
grid-column-end: 4;

grid-row-start: 3;
grid-row-end: 4;
}
```



Элемент может быть растянут вдоль строк или столбцов. Например, элемент здесь заканчивается у левой границы шестой колонки. Ее не существует, но можно представить, что она там есть.

```css
.b {
grid-column-start: 3;
grid-column-end: 6;

grid-row-start: 3;
grid-row-end: 4;
}
```

Каждый раз прописывать все четыре свойства достаточно долго, поэтому есть группирующее свойство только для колонок/рядов: `grid-column/grid-row` соответственно, и в целом: `grid-area`. Важно запомнить, в каком порядке указываются границы. `grid-area: grid-row-start`, `grid-column-start`, `grid-row-end`, `grid-column-end`.

```
.b {
grid-column: 3 / 6;

grid-row: 3 / 4;
}
```

```
.b {
grid-area: 3 / 3 / 4 / 6;
}
```

3.3.2. Шаблоны

В спецификации `grid` есть еще один способ задавать положение: с помощью шаблонов. Чтобы это сделать, нужно шаблон определить. Он определяется с помощью свойства `grid-template-areas`. Строка в этом свойстве группируется с помощью двойных кавычек. Первая строка полностью соответствует одной ячейке)`colspan = 5`). Далее во второй строке три столбца, в третьей - два.

```
.layout {
display: grid;
grid-template-columns: 100px 1fr 100px 2fr 100px;
grid-template-rows: auto 10px auto;

grid-template-areas: "h h h h h"
"a b b b c"
"d d d e e";
}
```

Чтобы спозиционировать элемент в соответствующую ячейку, дальше мы пользуемся свойством `grid-area` и указываем название области. Если мы указываем область `h`, то элемент займет всю первую строку, как задано в шаблоне:

```
.a {
grid-area: h;
}
```