

Лекция №6 Верстка сайта

1. DIV-Верстка(Float)
2. Семантическая верстка HTML5
3. Верстка на Flex блоках
4. Bootstrap
5. CSS Grid
6. CSS Grid или Bootstrap?

1 DIV-Верстка (Float)

Колонки в веб-дизайн пришли из полиграфии, где они используются в качестве способа разбивки широкого текста на более узкие фрагменты, а также для разделения различной информации.

Наиболее распространенным вариантом является наличие на веб-странице двух колонок — одна из них, как правило, содержит навигацию, а во второй, более широкой колонке, размещается контент.

Для резиновых макетов имеет смысл установить три колонки, чтобы эффективно использовать полезную площадь веб-страницы. В любом случае выбор числа колонок зависит исключительно от объема информации на сайте и способе её организации.

А) Резиновый двухколоночный макет

Двухколоночный резиновый макет позволяет эффективно использовать площадь браузера, и вместе с тем достаточно удобен для самого широкого круга задач. Кроме того, такой макет не требует сложной работы над собой и его можно использовать на многих сайтах, комбинируя ширину колонок в пикселях и процентах. Примером такого макета служит сайт Хабрахабр <http://habrahabr.ru> (рис. 1), ширина колонок у него задана в процентах. Опять же, существует несколько подходов к формированию такого макета, но самый простой и удобный заключается в сочетании свойств **float** и **margin**.

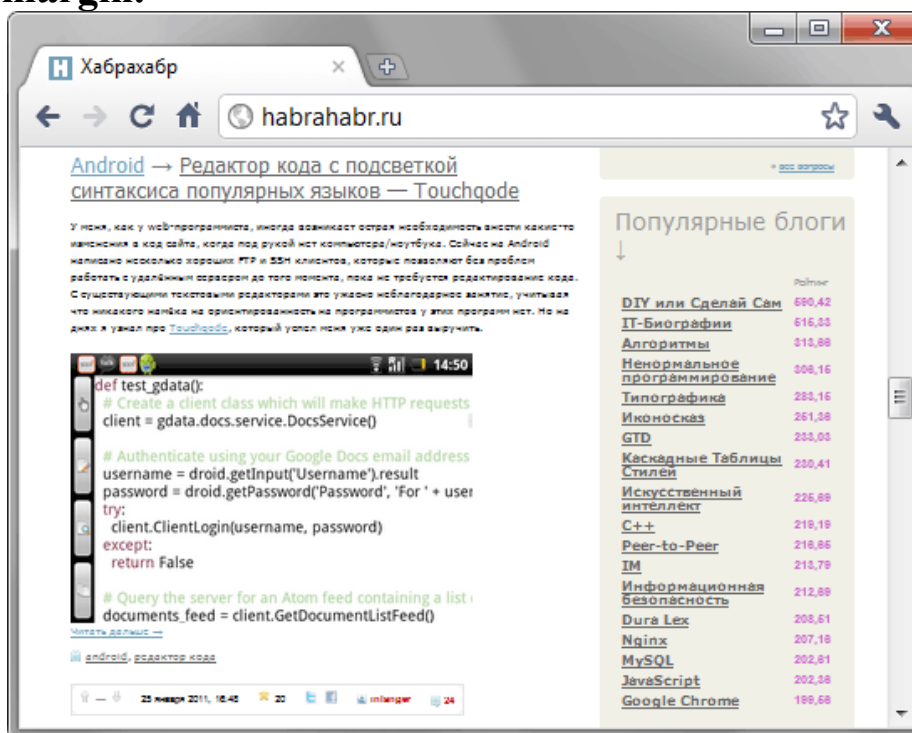


Рис. 1. Резиновый двухколоночный макет

Рассмотрим вариант, когда левая колонка имеет определенный размер, а ширина правой колонки устанавливается автоматически, исходя из ширины окна браузера. При этом ширина левой колонки может задаваться в пикселах или процентах.

В табл. 6.1 приведены основные стилевые свойства для формирования двух колонок.

Табл. 6.1. Левая колонка заданной ширины

Для левого слоя шириной 20%	
Левая колонка	Правая колонка
<code>float: left</code> <code>width: 20%</code>	<code>margin-left: 21%</code>
Для левого слоя шириной 200px	
<code>float: left</code> <code>width: 200px</code>	<code>margin-left: 210px</code>

Для левой колонки требуется всего два свойства: **float** — заставляет вторую колонку располагаться рядом по горизонтали с первой и **width**, которое устанавливает ширину колонки. Вторая колонка будет занимать всё оставшееся место, поэтому для нее указывать **width** не нужно.

Правая колонка характеризуется лишь одним свойством — **margin-left**, оно смещает левый край колонки на ширину левого слоя, плюс задает отступ между колонками.

Поэтому величина этого свойства в табл. 6.1 указана 21%, где 20% сама ширина колонки, а на один процент приходится расстояние между колонками. В случае задания ширины одной из колонок в пикселах, код останется прежним, но поменяются единицы измерения.

Из-за того, что мы имеем дело с резиновым макетом, который может занимать всю ширину окна браузера и уменьшаться до какого-то предела, имеет смысл сделать ограничения. Свойство **min-width** для слоя **layout** устанавливает минимальную ширину, если окно браузера будет меньше этого значения, появится горизонтальная полоса прокрутки. Свойство **max-width**, наоборот, задаёт максимальную ширину, которая не будет превышена.

Пример 6.1. Навигация слева

```
<head>
  <meta http-equiv="Content-Type"
content="text/html; charset=utf-8" />
  <title>Две колонки</title>
  <style type="text/css">
    .header, .sidebar, .content, .footer {
```

```

padding: 1 0px; /* Поля */
border: solid 1px #000; /* Параметры рамки */
background: #e3e8cc; /* Цвет фона */
}
.header { /* Верхняя часть с заголовком */
background: #e3e8cc; /* Цвет фона */
font-size: 24px; /* Размер шрифта */
}
.layout {
margin: 15px 0; /* Отступы сверху и снизу */
overflow: hidden; /* Отменяем действие float*/
min-width: 800px; /* Минимальная ширина */
max-width: 1200px; /* Максимальная ширина */
}
.sidebar { /* Навигация по сайту */
margin: 5px 0; /* Отступы сверху и снизу */
width: 100 px; /* Ширина меню */
float: left; /* Состыковка с другим слоем */
}
.sidebar ul {
list-style: none; /* Убираем маркеры */
padding: 0; /* Убираем отступы */
}
.content { /* Основное содержание страницы */
margin-left: 135px; /* Отступ слева */
}
</style>
</head>
<body>
<div class="header">Заголовок сайта</div>
<div class="layout">
<div class="sidebar">
<h2>Меню</h2>
<ul>
<li><a href="link1.html"> 1</a></li>
<li><a href="link2.html"> 2</a></li>
<li><a href="link3.html"> 3</a></li>
<li><a href="link4.html"> 4</a></li>
</ul>
</div>
<div class="content">
<h1>Название страницы</h1>
<p>Текст.</p>

```

```

    </div>
  </div>
  <div class="footer">Подвал</div>
</body>
</html>

```

Для формирования колонки заданной ширины справа, а не слева, код незначительно модифицируется.

В табл. 6.2 показаны стилевые параметры, которые требуются для этого случая.

Табл. 6.2. Правая колонка заданной ширины	
Для правого слоя шириной 20%	
Правая колонка	Левая колонка
float: right width: 20%	margin-right: 21%
Для правого слоя шириной 200px	
float: right width: 200px	margin-right: 210px

Расположение слоев в коде остается прежним, но значение свойства **float** меняется на **right**, а отступ на **margin-right**. Более никаких изменений не предполагается, поэтому пример 6.1 останется прежним и в нем только следует заменить стиль слоев **sidebar** и **content** на тот, что показан в примере 6.2.

Пример 6.2. Стиль для добавления меню справа

```

.sidebar {
  width: 100px;
  float: right;
}
.content {
  margin-right: 135px;
}.sidebar {
  width: 100px;
  float: right;
}
.content {
  margin-right: 135px;
}

```

Б) Резиновый трёхколоночный макет

Резиновый макет с тремя колонками, пожалуй, самый гибкий и настраиваемый из существующих макетов. Сочетание процентов и пикселей для указания ширины колонок позволяет делать разные макеты, выбирая их под свои задачи.

На рис. 6.2 представлены варианты трёхколоночных макетов, для удобства они пронумерованы.

Здесь символ процентов (%) означает, что ширина колонки задана в процентах от ширины макета, px — ширина колонки указана в пикселах, а знак бесконечности (∞), что колонка занимает оставшееся пространство.

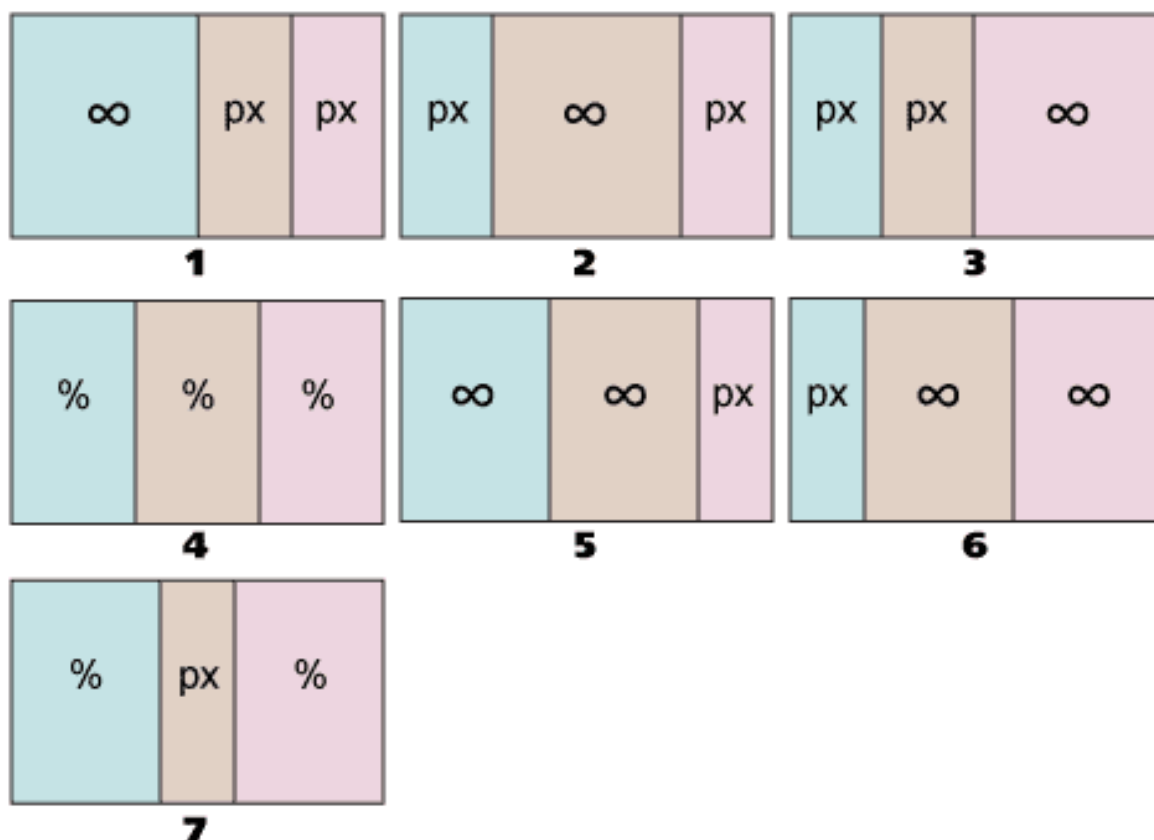
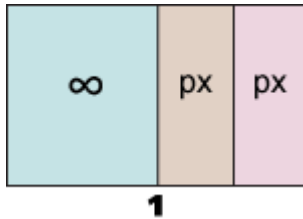


Рис. 6.2. Трёхколоночные макеты

Плавающие элементы - это универсальный метод построения разнообразных трёхколоночных макетов. В отличие от позиционирования он позволяет не заботиться о проблеме подвала, поскольку подвал всегда будет располагаться на своём месте под самой высокой колонкой.

Построение колонок происходит с помощью свойства float в сочетании со свойствами margin и width. В зависимости от выбранного макета меняется и порядок колонок в коде, вначале всегда следуют слои, к которым добавляется float.

Макет № 1. Резиновая первая колонка

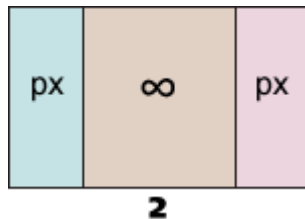


Ширина второй и третьей колонки указывается в пикселах, а их положение через свойство `float` со значением `right`. Для первой колонки также требуется задать свойство `margin-right` со значением равным суммарной ширине остальных колонок (пример 6.3).

Пример 6.3. Макет № 1—4.html

```
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Макет 1</title>
<style type="text/css">
  .header, .footer { background: #D5BAE4; }
  .layout {
    overflow: hidden; /* Отменяем обтекание */
  }
  .col1 {
    background: #C7E3E4; /* Цвет фона */
    margin-right: 300px; /*Сдвиг влево на ширину
колонок 2 и 3*/
  }
  .col2 {
    background: #E0D2C7; width: 200px;
    float: right; /* Обтекание слева */
  }
  .col3 {
    background: #ECD5DE; width: 100px;
    float: right; /* Обтекание слева */
  }
</style>
</head>
<body>
<div class="header">Шапка сайта</div>
<div class="layout">
  <div class="col3">Колонка 3</div>
  <div class="col2">Колонка 2</div>
  <div class="col1">Колонка 1</div>
</div>
<div class="footer">Подвал</div>
</body>
</html>
```

Макет № 2. Резиновая средняя колонка!!!

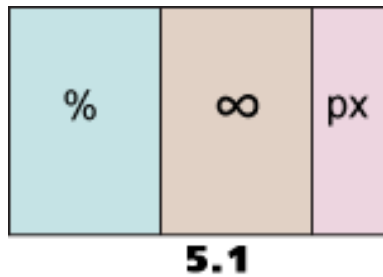


Ширина первой и третьей колонки задана в пикселах, а их положение — через свойство float со значением left для первой колонки и right для третьей. Средняя колонка, чтобы она сохраняла свой вид, содержит универсальное свойство margin, задающее отступ слева и справа

Пример Макет № 2

```
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title> Макет 2</title>
<style type="text/css">
.header, .footer { background: #D5BAE4; }
.layout { overflow: hidden; }
.col1 { background: #C7E3E4; float: left;
width: 200px; }
.col2 { background: #E0D2C7;
margin: 0 100px 0 200px; /* Отступ справа и
слева */ }
.col3 { background: #ECD5DE; width: 100px;
float: right; }
</style>
</head>
<body>
<div class="header">Шапка сайта</div>
<div class="layout">
<div class="col1">Колонка 1</div>
<div class="col3">Колонка 3</div>
<div class="col2">Колонка 2</div>
</div>
<div class="footer">Подвал</div>
</body>
</html>
```

Приведём стиль макета № 5 с первой колонкой, заданной в процентах.

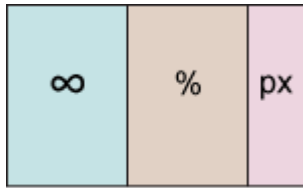


Здесь ширина первой колонки указывается в процентах, а её положение через свойство float со значением left, для третьей колонки ширина задана в пикселах, а значение свойства float как right. Для средней колонки остаётся только установить отступы слева и справа на ширину колонок.

Пример Макет 5.1

```
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
  <title>Макет 5.1</title>
  <style type="text/css">
    .header, .footer { background: #D5BAE4; }
    .layout { overflow: hidden; }
    .col1 { background: #C7E3E4; width: 40%; float:
left; }
    .col2 { background: #E0D2C7; margin: 0 200px 0
40%; }
    .col3 { background: #ECD5DE; width: 200px;
float: right; }
  </style>
</head>
<body>
  <div class="header">Шапка сайта</div>
  <div class="layout">
    <div class="col1">Колонка 1</div>
    <div class="col3">Колонка 3</div>
    <div class="col2">Колонка 2</div>
  </div>
  <div class="footer">Подвал</div>
</body>
</html>
```


Ширина первой колонки вычисляемая



5.2

Это наиболее хитрый макет, поскольку для первой колонки её ширина напрямую не указывается. Но чтобы ограничить контент необходимо для свойства `margin-right` указать значение, совмещающее проценты и пиксели. В CSS2, как уже говорилось, подобное задать нельзя, поэтому добавим внутрь слоя `col1` ещё один слой с именем `wrap` и добавим отступ каждому из этих слоёв. Одному в процентах, другому в пикселах

Пример Макет 5.2

```
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Макет 5.2</title>
<style type="text/css">
.header, .footer { background: #D5BAE4; }
.layout { overflow: hidden; }
.col1 { margin-right: 40%; }
.col1 .wrap { margin-right: 200px; background:
#C7E3E4; }
.col2 { background: #E0D2C7; width: 40%; float:
right; }
.col3 { background: #ECD5DE; width: 200px;
float: right; }
</style>
</head>
<body>
<div class="header">Шапка сайта</div>
<div class="layout">
<div class="col3">Колонка 3</div>
<div class="col2">Колонка 2</div>
<div class="col1">
<div class="wrap">
Колонка 1
</div> </div>
</div>
<div class="footer">Подвал</div>
</body></html>
```

Д) Эластичные шаблоны

Большинство дизайнов ориентировано на использование фиксированных значений при верстке: ширина и высота блоков, размер шрифта. Это позволяет сверстанному шаблону «не разваливаться» при изменении масштабов просмотра и сохранять свойство кроссбраузерности.

Однако в этом есть один большой минус – при большом разрешении экрана маленькие фиксированные блоки теряются на большой площади и остаются незамеченными. Тут даже не спасет «резиновая» верстка, т.к. сайт будет выглядеть еще более нечитабельным, например на 19" мониторах при разрешении больше 1280 по ширине. На ноутбуке 17" с разрешением 1400x800 просматривать «резиновый» сайт очень неудобно. И желание оставаться на таком сайте отпадает быстро. Существует ли способ, позволяющий управлять масштабами не только текста, но и всего сайта?

Первое, что сразу приходит в голову – это управление масштабированием. К счастью, данную функцию поддерживает большинство современных браузеров. Они позволяют управлять лишь масштабами текста.

Наша задача – сделать эластичным не только текст, но и содержащие его блоки. Особенность верстки эластичного шаблона в том, что все величины должны быть указаны не в пикселях (px) и не в процентах, а в em.

Почему неэффективно использование значений в процентах? Потому что, использование десятых и сотых долей в значении воспринимается не всеми браузерами. В то время как для em величин можно указывать 3 знака после запятой и каждая цифра будет учитываться.

Применение em позволяет делать любые элементы эластичными.

Стоит отметить, что создание эластичного шаблона потребует много расчетов, а именно – перевода привычных нам пикселей в em.

По умолчанию 1 em равен величине шрифта, который мы укажем у элемента body. Если величина шрифта не задана, то большинство браузеров автоматически устанавливают размер 16px, тогда и 1em = 16px.

Отсюда:

$0.5em = 8px$

$10em = 160px$ и т.д.

$0,625em (62,5\%) = 10px$. Это удобная точка отсчета.

Прописываем правила:

```
html{ font-size:100%; } /*необходимо для IE*/
body{ background:#eee;
font-size: 0.625em;
font-family:Arial;
text-align:center}
```

Теперь 1em будет равен 10px.

Пример "эластичной" верстки:

Нам нужно прописать правила для элементов H1, P, IMG и блока wrap. Для последнего установим ширину в 600px, предварительно переведя в em: $600\text{px}/10 = 60\text{em}$.

```
#wrap{
width: 60em;
margin: 1.5em auto; /* 15px/10 =1.5em*/
border: 0.1em solid #ccc; /* 1px/10 =0.1em*/
background: #fff;
text-align: left;
}
```

Для заголовка назначим размер шрифта, эквивалентный 16px, а для абзацев 12px

```
h1{
font-size: 1.6em; /* 16px/10 =1.6em*/
margin: 0.833em; /* 10px/16 =0.833em*/
font-weight: 300;
}
p{
font-size: 1.2em;
line-height: 1.333em; /* 16px/12 =1.33em*/
margin-bottom: 1.25em; /* 15px/12 =1.25em*/
}
```

Не забудем также, что габариты картинок теперь тоже надо указывать в em. Но это не проблема, когда формула преобразований так проста.

Присвоим элементу img следующие правила:

```
img {
width: 8.333em; /* 100px/12 =8.33em 12 - у блока p*/
height: 8.333em;
margin:0 0.833em 0.833em 0; /* 10px/12 =0.833em*/
}
```

2 Семантическая верстка HTML5

А) Новые семантические элементы

В HTML5 происходит чёткое отделение визуального отображения элементов страницы от их семантического значения.

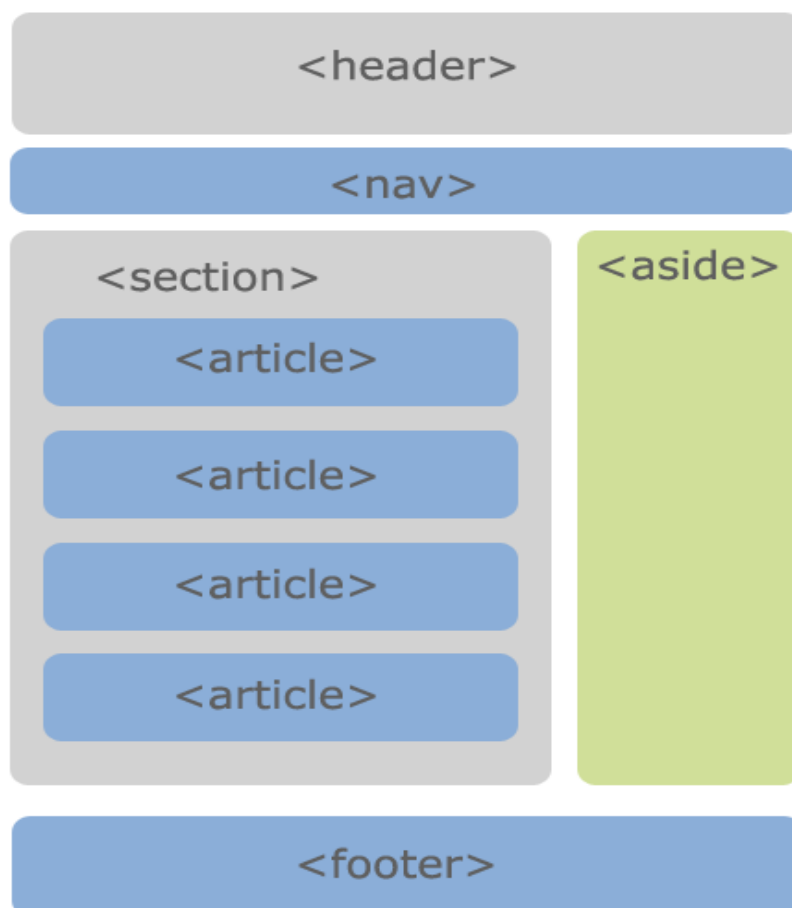
Поскольку учёт семантики занимает в html5 немаловажное место, это отразилось как на появлении новых тегов, так и изменении значения тегов старых.

Из всех новых компонентов спецификации быстрее всего приживаются именно семантические элементы, например, **Header, Footer, Section и Aside**. Многие из них основаны на типичных шаблонах использования, которые были выявлены в процессе сбора сведений редактором Йеном Хиксоном (Ian Hickson). Это наглядно видно на примере **Header и Footer**, которые отражают общепринятые шаблоны `id="header"` и `id="footer"`, применяемые во всей глобальной сети.

Новые элементы HTML5 начинаем рассматривать с крупных структурных единиц. Отныне в вёрстке есть место новым тегам:

<header> <footer> <nav> <article> <section> <aside>

Это крупные с точки зрения вёрстки элементы, условно соответствующие давно привычному нам тегу **DIV**.



Новые элементы:

- **section** представляет часть документа или раздел
- **article** представляет независимую часть содержания для включения в документ статей
- **aside** представляет часть содержания, которая только частично связана с остальной страницей
- **header** представляет заголовок **section**
- **footer** - нижний колонтитул, может содержать информацию об авторе, авторском праве и так далее
- **nav** представляет раздел документа, предназначенный для навигации
- **dialog** может использоваться для выделения диалогов:

Блок **nav** используется для создания блоков навигации. Как правило данный блок содержит группу ссылок, и аналогично **footer** и **header** также может быть использован несколько раз на странице. Например, один блок **nav** может служить для организации основного меню страницы, другой использоваться для второстепенных ссылок. Замечательной особенностью тега **nav** является способность карманных устройств обнаруживать такие блоки. Это ещё один аргумент в сторону использования **nav**.

Элементы **article** и **section** имеют некоторые общие черты и служат для организации смысловых блоков страницы. И **article**, и **section**, как правило, предполагают наличие своего заголовка в виде одного из тегов H1, H2, H3 и т.д., о которых мы ещё поговорим ниже.

```
<article>
<H1>Стихотворение о зафлуженном сайте</H1>
.....
</article>
```

Если у такого блока заголовок по смыслу не предусматривается, то вполне логично использовать для организации такого блока обычный тег **div**.

Несмотря на общие черты и свойства, у тегов **article** и **section** имеются различия, связанные с их предназначением. Предполагается, что **article** будет использоваться для организации некоего целостного и связанного по смыслу блока. Это не обязательно должна быть статья, но ключевые особенности — связанность и целостность. Это может быть фрагмент текста, который целиком может быть перенесён в другие издания. Тег **section** больше служит для отделения сайта на разные блоки, например, по тематике. К примеру, на нашем сайте есть как статьи о тушканчиках, так и о гиппопотамах. Логично поделить с помощью **section** сайт на разделы, а внутри каждого из таких разделов размещать статьи, используя для этого **article**. Если говорить в общем, точного определения по выбору **article** или **section** нет, скорее есть масса практических

рекомендаций. В любом случае чем дальше будете изучать html5, тем больше ваш опыт будет помогать в выборе того или иного тега.

Новый тег **aside** служит, в основном, для организации блоков, имеющих косвенное отношение к основному содержанию страницы, либо не имеющих к нему никакого отношения вообще. В качестве примера использования **aside** можно привести сайдбар на сайте. Как правило, содержимое в сайдбарах не сильно связано с темой основной части страницы, часто там располагаются сторонние баннеры и т.д. Ещё один пример применения **aside** – организация с их помощью различных врезок в статьи.

Новые блоки (header, footer, nav, aside, article, section) не только допускают вложенность друг в друга, но в некоторых случаях такое применение является, пожалуй, очень даже верным. Предположим, что у нас имеется блог, в котором авторы размещают свои статьи. Кроме того, каждая статья со временем обрастает некоторым количеством комментариев. Как сверстать такую страницу? Вполне разумным будет поместить саму статью внутри тега **article**, а каждый из комментариев к ней оформить как отдельный **article**, вложенный в **article** статьи. Почему бы и нет? Более того, обычно в заголовке каждой статьи принято указывать информацию с автором, датой выхода, наименованием рубрики, числом комментариев, количеством просмотров и т.п. Для этих целей вполне можно использовать вложенный блок **header**.

А что же наш старый знакомый элемент **div**, неужели для него в html5 теперь нет места? Вовсе нет, блоки **div** по-прежнему используются. У **div** практически полностью отсутствует врождённая семантика. Блоки с использованием тегов header, footer, aside, nav, article и section хотя и могут выглядеть визуально аналогично **div**, но несут изначально заложенную в них семантику. У **div** есть неоспоримое преимущество: он отображается верно даже устаревшими браузерами.

Новые структурные теги (элементы) описываются точно так же, как это делалось и для блоков **div**. А именно, с помощью селектора тега в CSS.

Приведём примеры:

```
header {
    width: 100%;
    margin: 0px;
    background: #000088;
}
article {
    padding: 10px 20px 10px 40px;
}
footer {
    width: 100%;
```

```
margin: 0px;
background: #444;
color: #ddd;
}
```

С использованием новых тегов пустой шаблон HTML5 может выглядеть так:

```
<!DOCTYPE HTML>
<html lang="ru">
<head>
  <meta charset="utf-8" />
  <title></title>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
<header>Заголовок страницы</header>
<nav>Меню навигации</nav>
<aside>Боковая колонка SideBar</aside>
<article>
  Контент - основное содержимое страницы.
</article>
<footer>Подвал сайта</footer>
</body>
</html>
```

3 Верстка на Flex блоках

Технология Flexbox поддерживается уже почти всеми используемые на сегодняшний момент браузерами (с использованием префиксов: IE10+, Edge12+, Firefox 2+, Chrome 4+, Safari 3.1+, Opera 12.1+, iOS Safari 3.2, Opera mini, Android 2.1+, Blackberry 7+).

А) Основы Flexbox (сетка)

В основу Flexbox положена сетка. Она состоит всего из 2 элементов. Первый элемент – это **flex-контейнер** и второй- **flex-элементы**.

Создание flex-контейнера осуществляется посредством добавления к необходимому HTML элементу CSS-свойства display со значением **flex** или **flex-inline**.

После этого **все непосредственные дочерние элементы flex-контейнера** (дети) автоматически становятся flex-элементами.

Значение flex или flex-inline определяет то, как **flex-контейнер** будет представлен на странице. Если его необходимо представить как **блок**, то используйте значение **flex**. Если элемент необходимо представить

как **строчный**, то используйте значение **flex-inline**. В этом случае он будет занимать столько места странице, сколько необходимо для отображения его элементов.

Flex-элементы по умолчанию занимают всю высоту flex-контейнера.

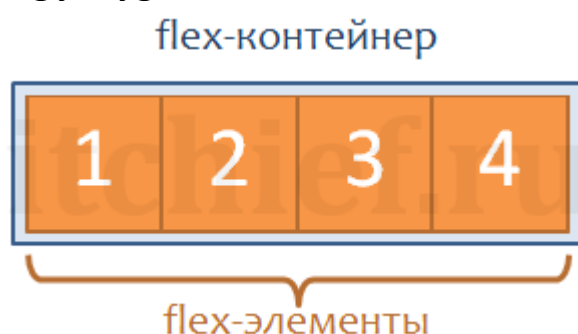
HTML разметка:

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
```

CSS-стили:

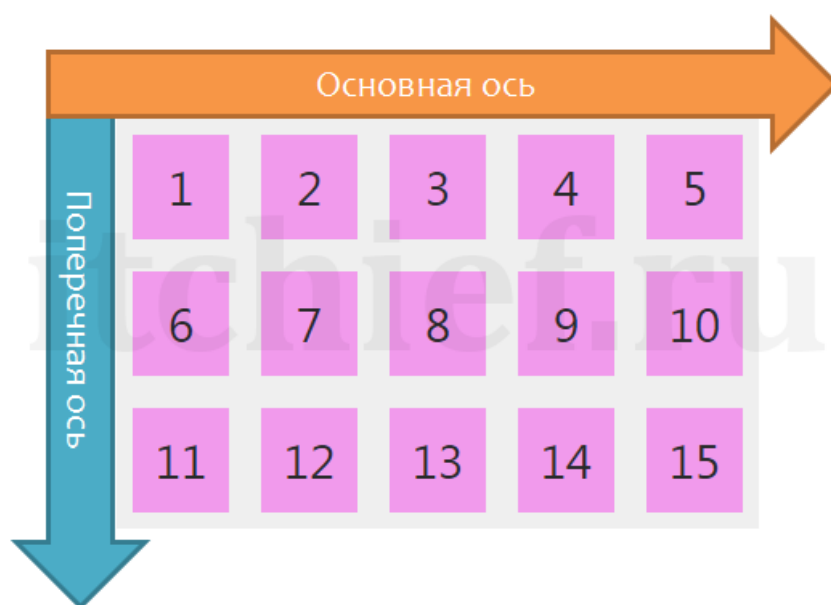
```
.flex-container {
  display: flex; /* flex || inline-flex */
}
```

Структура flexbox сетки



Б) Направление выстраивания flex-элементов

Указание направления выстраивания flex-элементов внутри flex-контейнера осуществляется посредством **осей**.



Во flexbox выделяют **2 оси**. Первая ось называется **основной** или **главной** (по умолчанию направлена вправо). Вторая – **поперечная** (по умолчанию направлена вниз).

Элементы во flex-контейнере располагаются в одну линию (по умолчанию) даже тогда, когда им не хватает места. Выстраиваются flex-элементы в flex-контейнере по направлению основной оси.



В CSS Flexbox разрешить перенос Flex-элементов на новые линии (если им не хватает места в текущей линии) осуществляется с помощью установки flex-контейнеру CSS свойства **Flex-Wrap** со значением **Wrap** или **Wrap-Reverse**.

```
flex-wrap: wrap;
/*
```

nowrap (в одну линию - по умолчанию)

wrap (разрешить перенос flex-элементов на новые линии)

wrap-reverse (осуществлять перенос flex-элементов в обратном порядке)

```
*/
```

Установка направления главной оси Flexbox осуществляется с помощью CSS-свойства **Flex-Direction**.

```
flex-direction: row;
/*
```

row (вправо) - по умолчанию

row-reverse (налево)

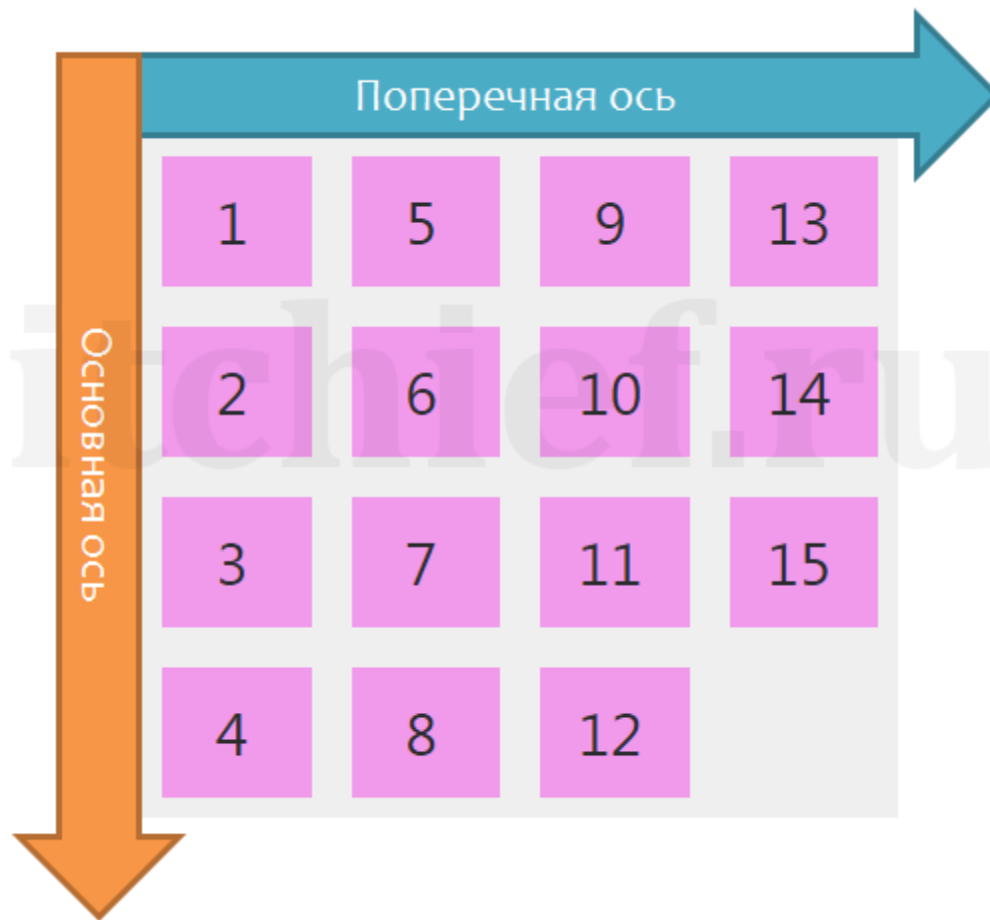
column (вниз)

column-reverse (вверх)

```
*/
```

С помощью этого свойства можно сделать так, чтобы flex-элементы располагались не горизонтально (строками), а вертикально (колонками).

CSS Flexbox - Расположение элементов при установке **flex-direction** значения **column**, а **flex-wrap** **wrap**.



Свойства **flex-direction** и **flex-wrap** можно указать с помощью универсального CSS свойства **flex-flow**:

```
flex-flow: row nowrap;  
1 значение - flex-direction,  
2 значение - flex-wrap
```

В) Выравнивание flex-элементов

Во Flexbox выравнивание элементов внутри контейнера осуществляется по двум направлениям (осям).

Выравнивание элементов вдоль основной оси осуществляется с помощью CSS свойства **justify-content**:

```
justify-content: flex-start;  
flex-start (flex-элементы выравниваются относительно  
начала оси) - по умолчанию  
flex-end (flex-элементы выравниваются относительно  
конца оси)  
center (по центру flex-контейнера)  
space-between (равномерно, т.е. с одинаковым  
расстоянием между flex-элементами)  
space-around (равномерно, но с добавлением половины  
пространства перед первым flex-элементом и после  
последнего)
```

Варианты выравнивания flex-элементов вдоль главной оси

flex-start



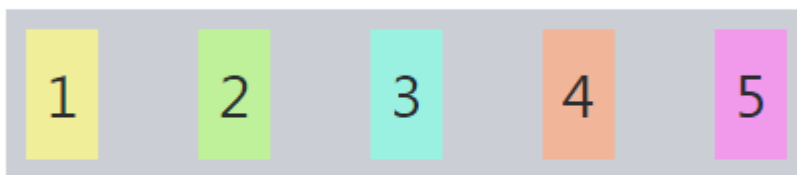
flex-end



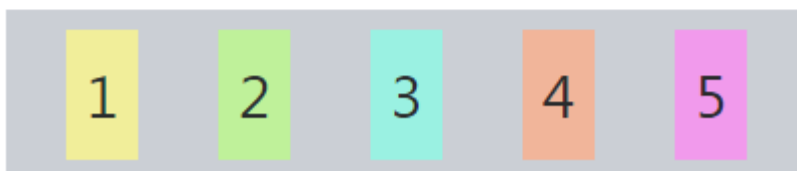
center



space-between



space-around



Выравнивание flex-элементов во flex-контейнере по направлению поперечной оси осуществляется с помощью CSS-свойства **align-items**:

```
align-items: stretch;
/*
stretch (растягиваются по всей длине поперечной оси)
– по умолчанию
flex-start (относительно начала поперечной оси)
flex-end (относительно конца поперечной оси)
baseline (относительно базовой линии)
center (по центру)
*/
```

Варианты выравнивания flex- элементов вдоль поперечной оси

stretch



flex-start



flex-end



baseline



center



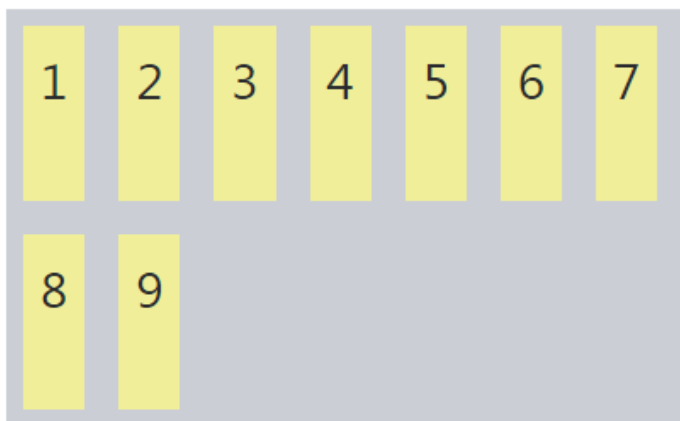
Выравнивание линий flex-контейнера

CSS Flexbox позволяет выравнивать не только сами flex-элементы, но и линии, на которых они расположены.

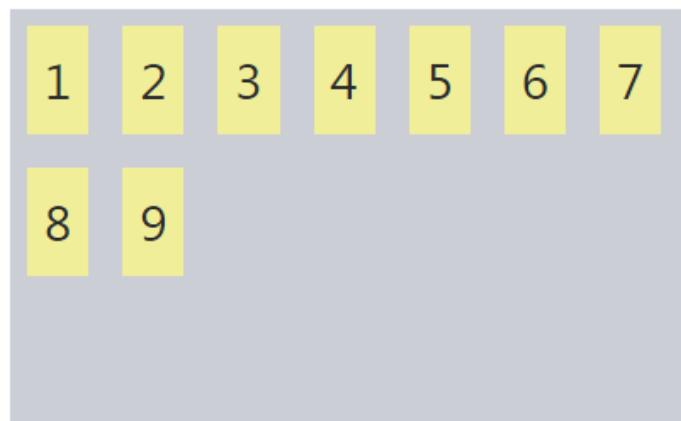
```
align-content: stretch
/* stretch (растягиваются по всей длине
поперечной оси) – по умолчанию
flex-start (относительно начала поперечной оси)
flex-end (относительно конца поперечной оси)
center (по центру)
space-between (равномерно, т.е. с одинаковым
расстоянием между линиями)
space-around (равномерно, но с добавлением
половины пространства перед первой линией и после
последней)
*/
```

Варианты выравнивания линий flex контейнера

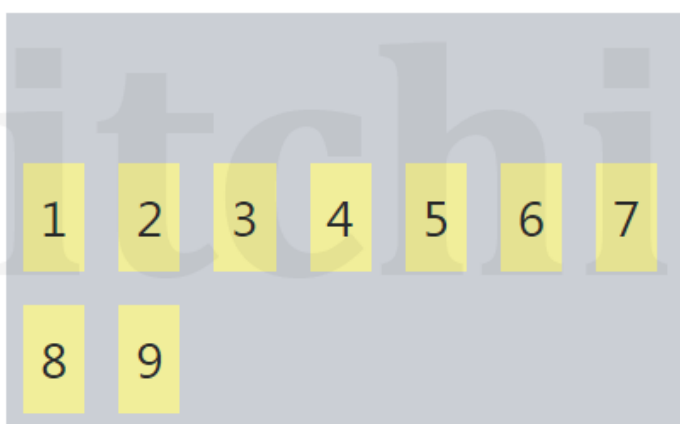
stretch



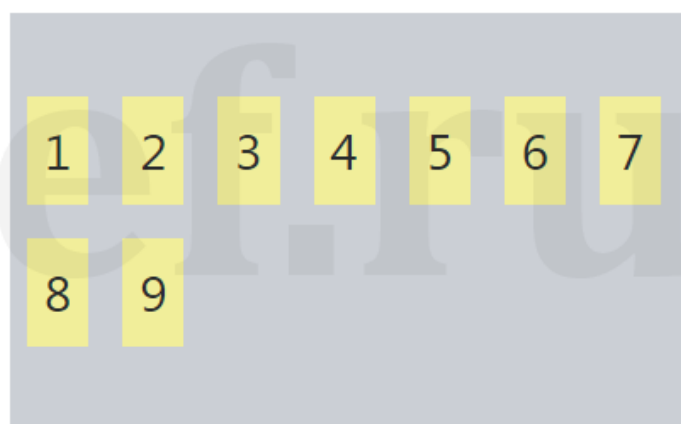
flex-start



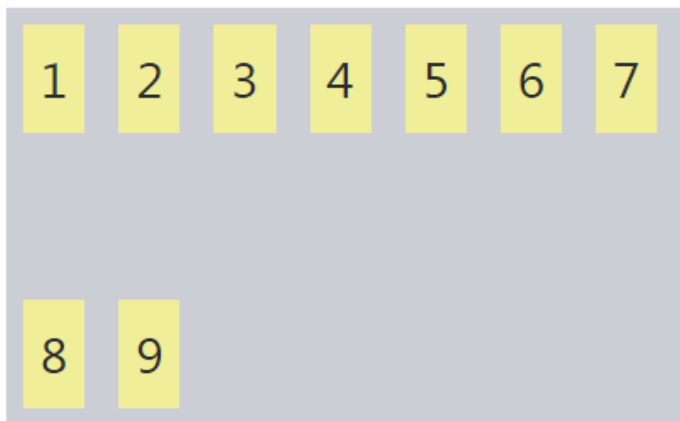
flex-end



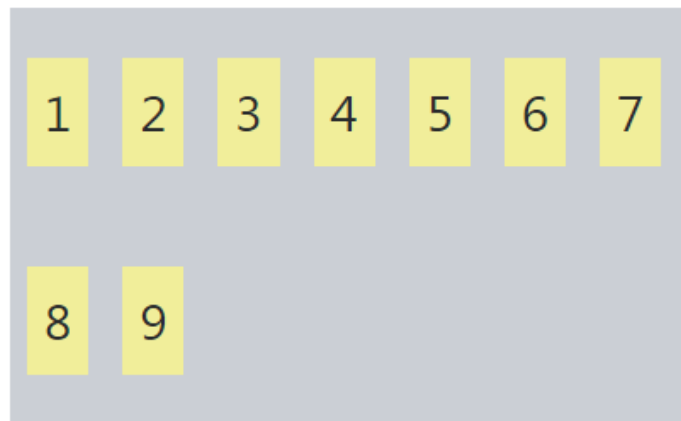
center



space-between



space-around



Свойство **align-content** имеет смысл использовать только тогда, когда flex-элементы во flex-контейнере располагаются на нескольких линиях. Чтобы это произошло, необходимо, во-первых, чтобы ширина всех flex-элементов была больше ширины flex-контейнера, а во-вторых flex-контейнер должен иметь в качестве CSS-свойства **flex-wrap** значение **wrap** или **wrap-reverse**.

Д) CSS-свойство align-self

Свойство **align-self** в отличие от предыдущих (**justify-content**, **align-items** и **align-content**) предназначено для flex-элементов. Оно позволяет изменить **выравнивание flex-элемента** вдоль направления поперечной оси. Свойство **align-self** может принимать такие же значения как **align-items**.

```
align-items: stretch;
/*
  auto (по умолчанию) || stretch || flex-start ||
flex-end || baseline || center
*/
```

Пример:

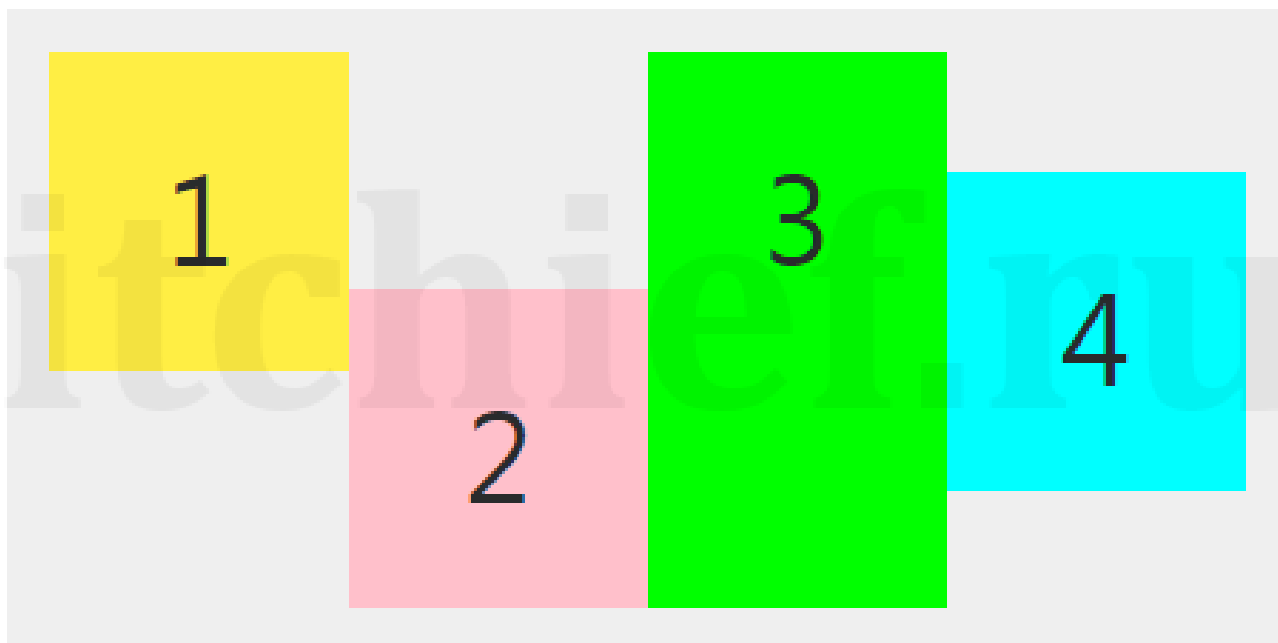
```
<div class="flex-container">
  <div class="flex-container_element-1">      1 </div>
  <div class="flex-container_element-2">      2 </div>
  <div class="flex-container_element-3">      3 </div>
  <div class="flex-container_element-4">      4 </div>
</div>
```

CSS:

```
.flex-container {
  display: flex;
  width: 300px;
  height: 150px;
  align-items: center;
  padding: 10px;
  background-color: #efefef;
}
.flex-container_element-1,
.flex-container_element-2,
.flex-container_element-3,
.flex-container_element-4 {
  flex-basis: 70px;
  text-align: center;
  padding: 15px;
  font-size: 30px;
}
.flex-container_element-1 {
  align-self: flex-start;
  background: #fe4;
}
.flex-container_element-2 {
  align-self: flex-end;
  background: pink;
}
```

```
.flex-container_element-3 {
  align-self: stretch;
  background: lime;
}
.flex-container_element-4 {
  align-self: auto;
  background: cyan;
}
```

Как работает CSS свойство **align- Self**



Е) Изменение порядка следования flex-элементов

По умолчанию flex-элементы отображаются во flex-контейнере в том порядке, в котором они расположены в HTML коде. Для изменения порядка следования одних flex-элементов относительно других в CSS Flexbox можно использовать свойство **order**. Данное CSS свойство выстраивает flex-элементы во flex-контейнере в порядке возрастания их номеров.

```
order: 0;
/*
  0 (по умолчанию)
  целое положительное или отрицательное число
*/
```

Например:

```
<div class="flex-container">
  <div class="flex-container_element-1">...</div>
  <div class="flex-container_element-2">...</div>
  <div class="flex-container_element-3">...</div>
  <div class="flex-container_element-4">...</div>
</div>
```

CSS:

```
.flex-container {  
  display: flex;  
/* переместим 2 flex-элемент в конец */  
.flex-container_element-2 {  order: 2;}  
/* передвинем 3 элемент до 2 */  
.flex-container_element-3 {  order: 1;}  
/* расположим 4 flex-элемент до 1 */  
.flex-container_element-4 {  order: -1;}
```

Как работает CSS свойство **order**:



Ж) Управление шириной flex-элемента

Во Flexbox есть несколько CSS свойств, определяющих то, какая ширина может быть у flex-элемента.

CSS-свойство flex-basis

Данное свойство предназначено для **установления начальной ширины flex-элементу**. Задавать значение ширины можно посредством различных единиц измерения, таких как px, %, em и др. По умолчанию данное свойство имеет значение **auto** (в этом случае ширина элемента будет рассчитываться автоматически на основании его содержимого).

Конечная ширина flex-элемента будет определяться в зависимости от значений CSS-свойств **flex-grow** и **flex-shrink**, которые установлены не только для этого элемента, но и для других flex-элементов этого flex-контейнера.

CSS-свойство flex-grow

Это свойство определяет, может ли **начальная ширина flex-элемента увеличиваться (расти)**. Увеличение ширины flex-элемента осуществляется за счёт **свободного пространства линии**. В качестве значения CSS-свойства **flex-grow** указывается **целое число**. Именно это значение и определяет (если оно больше или равно 1) какую часть свободного пространства flex-элемент заберёт себе.

Например:

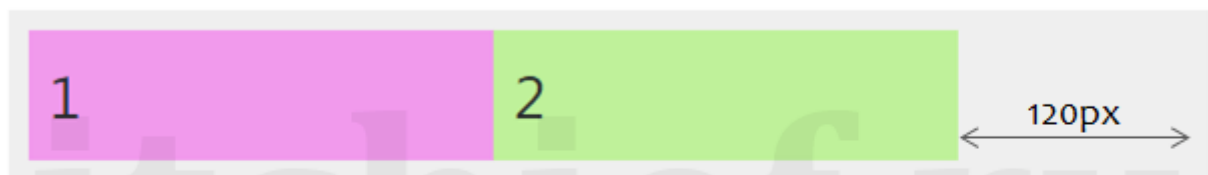
```
<div class="flex-container">  
  <div class="flex-container_element-1">...</div>  
  <div class="flex-container_element-2">...</div>  
</div>
```

CSS:


```
.flex-container {  
  display: flex;  
  width: 600px;}  
.flex-container_element-1 {  
  flex-basis: 40%;  
  flex-grow: 1;}  
.flex-container_element-2 {  
  flex-basis: 40%;  
  flex-grow: 4;}
```

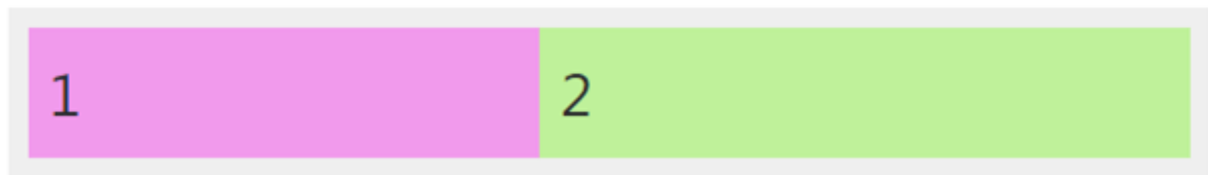
Как работает CSS свойство flex grow:

до установки flex-grow



после установки flex-grow

(для первого flex-элемента значения 1, для второго – 4)



В этом примере, если flex-элементы расположены на одной линии и в ней есть свободное пространство ($600 \times (1 - 0,8) = 120\text{px}$):

- к ширине элемента `.flex-container_element-1` добавится $1/5$ часть этого пространства ($120 \times 1/5 = 24\text{px}$);
- к ширине элемента `.flex-container_element-2` добавится $4/5$ части этого пространства ($120 \times 4/5 = 96\text{px}$).

Другими словами, CSS свойство `flex-grow` позволяет не просто указать, что ширина flex-элемента может вырасти, но и задать, насколько эта величина может вырасти по отношению к другим элементам.

По умолчанию CSS свойство `flex-grow` имеет значение 0. Это означает, что flex-элемент не может расти (увеличивать свою ширину).

CSS-свойство flex-shrink

Данное свойство определяет, может ли ширина flex-элемента уменьшиться. Уменьшение ширины flex-элемента будет осуществляться только в том случае, если ширины линии будет не достаточно для отображения всех flex-элементов, расположенных в ней.

Необходимая ширина рассчитывается на основании начальной ширины, который имеет каждый flex-элемент в ней.

Например:

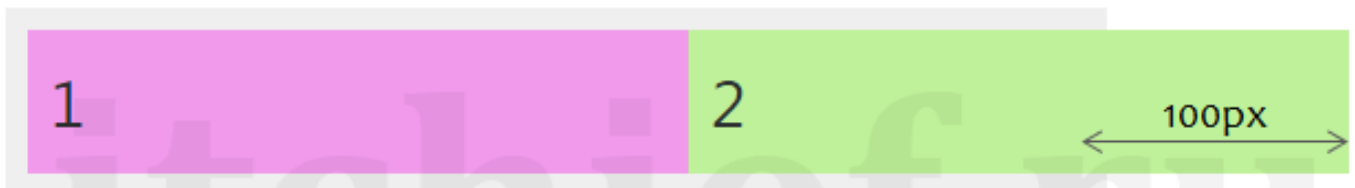
```
<div class="flex-container">
  <div class="flex-container_element-1">...</div>
  <div class="flex-container_element-2">...</div>
</div>
```

CSS:

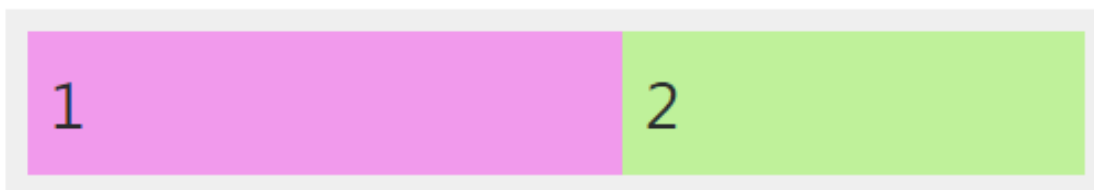
```
.flex-container {
  display: flex;
  width: 500px;}
.flex-container_element-1 {
  flex-basis: 300px;
  flex-shrink: 1;}
.flex-container_element-2 {
  flex-basis: 300px;
  flex-shrink: 3;}
```

Как работает CSS свойство flex- shrink:

flex-элементам **запрещено уменьшаться** (значение flex-shrink для первого и второго элемента равно 0)



flex-элементам **разрешено уменьшаться** (значение flex-shrink для первого flex-элемента равно 1, для второго – 3)



Ширина flex-контейнера 500px. Для отображения flex-элементов необходимо 600px. В итоге не хватает 100px. В этом примере уменьшаться могут 2 flex-элемента (`.flex-container_element-1` и `.flex-container_element-2`). Ширина первого flex-элемента `.flex-container_element-1` в данном случае составит $300 - 1/4 * 100 = 275$ px. Ширина второго flex-элемента `.flex-container_element-2` в данном случае составит $300 - 3/4 * 100 = 225$ px.

Значение по умолчанию:

```
flex-shrink: 1;
```

Если вам необходимо запретить уменьшение ширины flex-элементу, то в качестве значения свойства `flex-shrink` необходимо указать число 0.

CSS-свойство flex:

Для удобной установки `flex-grow`, `flex-shrink` и `flex-basis` можно использовать CSS свойство `flex`.

Значение по умолчанию:

```
flex: 0 1 auto;
/*
  0 - flex-grow (1 значение)
  1 - flex-shrink (2 значение)
  auto - flex-basis (3 значение)
*/
```

II) Верстка макета страницы на CSS Flexbox

В этом разделе создадим простой адаптивный макет на Flexbox.

Структура макета будет состоять из 3 секций:

- **header** (для вывода заголовка и основного меню);
- **main** (для отображения основной части);
- **footer** (для футера).

Основную часть (main) в свою очередь разделим ещё на 2 раздела (их позиционирование будем осуществлять с помощью CSS Flexbox). На больших экранах ($\geq 992\text{px}$) эти разделы выстроим горизонтально, а на остальных - вертикально ($< 992\text{px}$).

```
<header class="container">
  [Шапка страницы...]
</header>
<main class="container">
  <div class="row-flex">
    <article class="main_article col-flex">
      [Основная часть...]
    </article>
    <aside class="main_aside col-flex">
      [Дополнительная часть...]
    </aside>
  </div>
</main>
<footer class="container">
  [Футер...]
</footer>
```

CSS:

```
/* контейнер (устанавливает ширину блока в
зависимости от ширины viewport) */
.container {
  position: relative;
  margin-left: auto;
  margin-right: auto;
  padding-right: 15px;
  padding-left: 15px;}
@media (min-width: 576px) {
```

```
.container {
    width: 540px;
    max-width: 100%; }}
@media (min-width: 768px) {
    .container {
        width: 720px;
        max-width: 100%; }}
@media (min-width: 992px) {
    .container {
        width: 960px;
        max-width: 100%; }}
@media (min-width: 1200px) {
    .container {
        width: 1140px;
        max-width: 100%; }}
/* flex-контейнер */
.row-flex {
    display: -webkit-box;
    display: -webkit-flex;
    display: -ms-flexbox;
    display: flex;
    -webkit-flex-wrap: wrap;
        -ms-flex-wrap: wrap;
            flex-wrap: wrap;
    margin-right: -15px;
    margin-left: -15px;}
/* CSS настройки flex-элементов */
.col-flex {
    position: relative;
    width: 100%;
    min-height: 1px;
    padding-right: 15px;
    padding-left: 15px;}
/* ширина блоков article и aside по умолчанию */
.main_article, .main_aside {
    -webkit-box-flex: 0;
    -webkit-flex: 0 0 100%;
        -ms-flex: 0 0 100%;
            flex: 0 0 100%;
    max-width: 100%;
}
/* ширина блоков article и aside для больших экранов */
```

```

@media (min-width: 992px) {
  /* 2/3 от ширины контейнера */
  .main_article {
    -webkit-box-flex: 0;
    -webkit-flex: 0 0 66.666667%;
    -ms-flex: 0 0 66.666667%;
    flex: 0 0 66.666667%;
    max-width: 66.666667%;
  }
  /* 1/3 от ширины контейнера */
  .main_aside {
    -webkit-box-flex: 0;
    -webkit-flex: 0 0 33.333333%;
    -ms-flex: 0 0 33.333333%;
    flex: 0 0 33.333333%;
    max-width: 33.333333%;
  }
}

```

Для поддержки макета большинством браузеров добавим в CSS необходимые префиксы и **max-width**.

Для «превращения» блока во flex-контейнер будем использовать класс **row-flex**. Установку ширины каждому flex-элементу (**main_article** и **main_aside**) внутри flex-контейнера будем осуществлять с помощью CSS-свойства **flex**.

Макет веб-страницы на Flexbox



Единственные доводы в сторону float:

1. старые браузеры (IE9-).
2. поддержка и понимание работы существующего кода (на данный момент существует действительно очень много кода, использующего float).

Принципиальные различия:

`float` присваивается элементам, а `display: flex` или `display: inline-flex`, присваивается контейнерам и оказывают влияние на расположение элементов контейнера.

Когда блоку присвоен `display: flex` или `display: inline-flex`, его элементы игнорируют свойства `float`, `clear` и `vertical-align`.

Аспекты работы flexbox (которые невоплотимы через `float`, кроме *некоторых случаев* многострочной разметки):

1 многострочная разметка (когда элементы могут разрастаться на несколько строк). Достигается через установку контейнеру `flex-wrap: wrap`; или `flex-wrap: wrap-reverse`; . Через `float` элементы разве что могут располагаться по левому краю (через `float: left`;) либо просто по правому краю (через `float: right`) с необходимостью инвертировать порядок расположения элементов. Отдельная большая тема для flexbox из-за его огромных возможностей.

2 расположение элементов в столбец (когда элементы располагаются не в строку). Достигается через установку контейнеру `flex-direction: column`; или `flex-direction: column-reverse`;

3 возможности flex-элементов растягиваться и сужаться с помощью свойств `flex-grow` и `flex-shrink`.

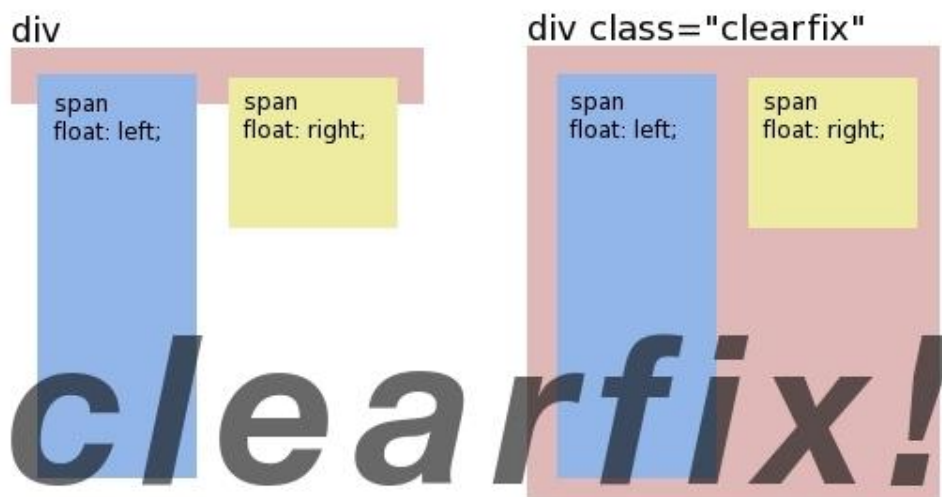
4 как flex-элементы работают с `margin: auto`; (`margin` занимает оставшееся пространство в соответствующем направлении).

5 сочетание flexbox и абсолютного позиционирования.

6 взаимодействие flexbox-элементов с отрицательными `margin` и `z-index`

7 Чтобы контейнер имел ненулевую высоту ему приходится всегда добавлять класс `clearfix`:

```
.clearfix:after {  
  content: "";  
  display: table;  
  clear: both;  
}
```



Для flexbox-контейнера такие "фокусы" не нужны.

8 Нельзя горизонтально центрировать элементы с помощью `float`. Нет значения `float: center`;,. Приходится применять другие приёмы.

9 Также можно менять порядок индивидуальных элементов с помощью свойства `order`.

```
.container {
  display: flex;
  flex-direction: row-reverse;
}
.item {
  /* Просто для демонстрации */
  background-color: orange;
  color: white;
  font-size: 2rem;
  padding: 10px;
  border-radius: 10px;
}
.item:nth-of-type(1) { order: 3; }
.item:nth-of-type(2) { order: 4; }
.item:nth-of-type(3) { order: 1; }
.item:nth-of-type(4) { order: 5; }
.item:nth-of-type(5) { order: 2; }
<div class="container">
  <div class="item">      One      </div>
  <div class="item">      Two      </div>
  <div class="item">      Three     </div>
  <div class="item">      Four     </div>
  <div class="item">      Five     </div>
</div>
```

2. С помощью `float` невозможно гарантировать нахождение элементов на одной строке.

В flexbox такое поведение заложено по умолчанию (`flex-wrap: nowrap`; установлено по умолчанию) и элементы не будут переноситься на новую строку без явного указания контейнеру `flex-wrap: wrap`; (или в редком случае `flex-wrap: wrap-reverse`;).

3. Нет никакого контроля за вертикальным расположением элементов. Всё выравнивается "по верху" без возможности на это повлиять. Также нельзя установить элементам одинаковую высоту без установления фиксированной высоты всем элементам.

В flexbox есть возможность установить вертикальное расположение, как и всем элементу, так и индивидуально элементу. Всем — устанавливаем контейнеру `align-items`, индивидуально — устанавливаем элементу `align-item`.

В общем, как можно увидеть, flexbox несопоставимо мощней и намного выразительней вёрстки прошлых версий CSS.

3. Bootstrap

Bootstrap — это набор инструментов от *Twitter*, который помогает вам разрабатывать свои веб-приложения намного быстрее.

В *Bootstrap* входят *CSS* и *Javascript* файлы, позволяющие создавать страницы с использованием сеток, шаблонов, типографии, таблиц, форм, навигации, всплывающих окон и т.д. Кроме того, разработчики решили проблемы кроссбраузерной совместимости.

Для начала добавьте в свой документ два файла *CSS*: *bootstrap.css* и *bootstrap-responsive.css*.

Вы также можете использовать *url* с сайта *github*: [twitter.github.com](https://github.com/twitter/bootstrap)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Page Title</title>
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
<link href="assets/css/bootstrap.css"rel="stylesheet">
<link href="assets/css/bootstrap-responsive.css"
rel="stylesheet">
</head>
<body>
// Дизайн разметки страницы.
</body>
</html>
```

Подключите следующие файлы *JavaScript* и поместите их в конец документа. Это ускорит загрузку страницы.

```
<script src="assets/js/jquery.js"></script>
<script src="assets/js/bootstrap-
transition.js"></script>
<script src="assets/js/bootstrap-alert.js"></script>
<script src="assets/js/bootstrap-modal.js"></script>
<script src="assets/js/bootstrap-
dropdown.js"></script>
<script src="assets/js/bootstrap-
scrollspy.js"></script>
<script src="assets/js/bootstrap-tab.js"></script>
<script src="assets/js/bootstrap-tooltip.js"></script>
<script src="assets/js/bootstrap-popover.js"></script>
<script src="assets/js/bootstrap-button.js"></script>
<script src="assets/js/bootstrap-
collapse.js"></script>
```



```
<script src="assets/js/bootstrap-
carousel.js"></script>
<script src="assets/js/bootstrap-
typeahead.js"></script>
```

Общая структура

1. **header** – шапка сайта, верхняя область в которой находится логотип, главное меню (иногда), доп. инфа (контакты, доп. меню, переключатели языков) и т.д.
2. **footer** – подвал сайта, самая нижняя часть веб-страницы в которой могут находиться дополнительные меню, блок с подпиской, блок с контактами, социальные иконки, копирайт и т.д.
3. **основная часть** – находится как правило сразу под шапкой сайта. В основной части содержимое варьируется в зависимости от тематики сайта, если это блок, то будет некий контент (пост, страница и т.д.), если это интернет магазин, то тут будет товар (страница товара, категория товаров и т.д.).
4. **sidebar** – боковая панель сайта, она бывает не во всех дизайнах, а в некоторых их может быть несколько. Сайдбар как правило находится слева или справа от основного контента и содержит в себе различные виджеты.

Сетки

Bootstrap включает отзывчивую, изначально ориентированную на мобильные устройства сетку, которая масштабируется, начиная с 12 столбцов как на устройствах или при изменении окна просмотра.

Это сетка, используемая Twitter Bootstrap имеет ширину 940px и поддерживает 12 колонок. Класс `span1` имеет ширину 40px.



Сетки используют для построения макетов страниц посредством компоновки строк и колонок, которые содержат контент.

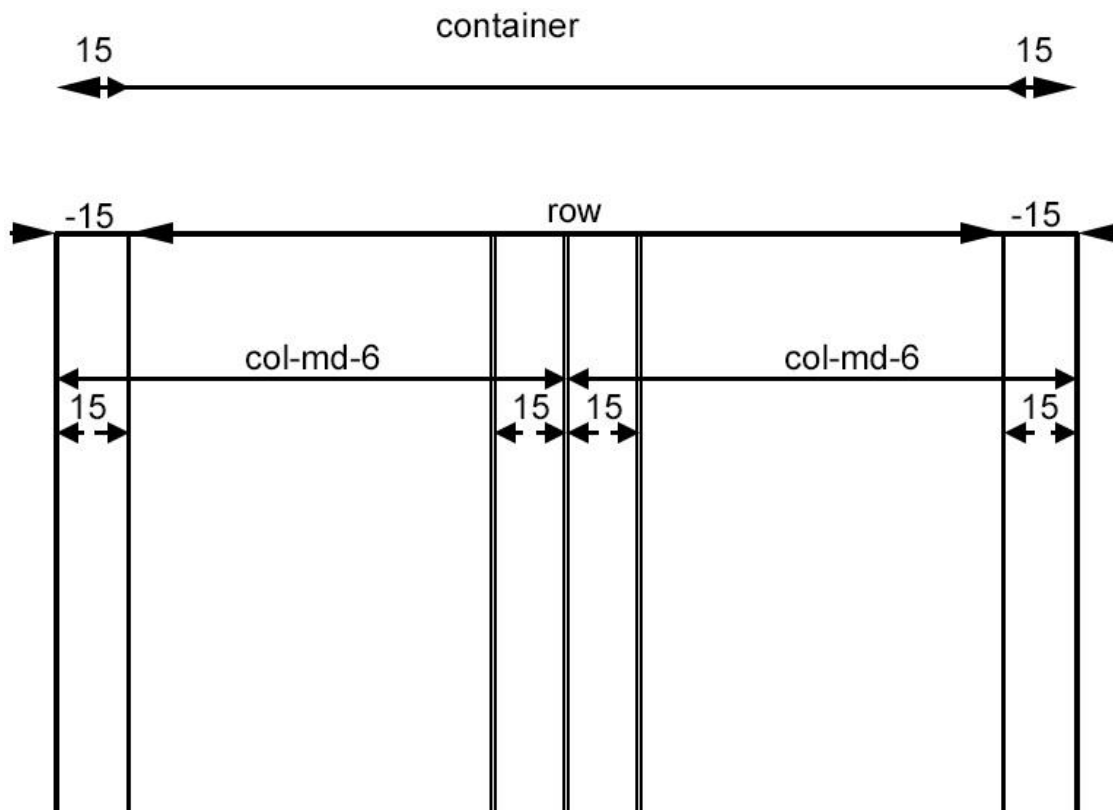
Контейнер-обертка

Bootstrap требуется охватывающий элемент, чтобы обернуть контент сайта и стать домом для нашей сеточной системы. Для своего проекта вы можете использовать один из двух вариантов.

Используйте **.container** для отзывчивого с фиксированной шириной контейнера (максимальная ширина блока 1170px).

Используйте **.container-fluid** для контейнера с шириной на всю область просмотра.

Чтобы понять все эти принципы было еще проще взгляните на изображение ниже:



Особенности верстки с применением сетки

Ниже представлен код разметки, для формирования четырех столбцов при помощи сетки:

```
<div class="container">
  <div class="row">
    <div class="col-md-3"><p>Column 1</p></div>
    <div class="col-md-3"><p>Column 2</p></div>
    <div class="col-md-3"><p>Column 3</p></div>
    <div class="col-md-3"><p>Column 4</p></div>
  </div>
</div>
```

Как видно из кода выше, для формирование структуры сайта используются три основных класса: **.container**, **.row**, **.col-***.

.container – блок обертки всего содержимого, задает основную ширину сайта

.row – блок обертки для колонок, имеет отрицательный отступ

.col-* – непосредственно блок колонки

Теперь давайте немного подробнее поговорим о правилах построения макета с применением сетки.

Строки (**.rows**) должны быть размещены в пределах **.container** (фиксированной ширины) или **.container-fluid** (полной ширины) для правильного выравнивания и заполнения.

Используйте строки (**.rows**) для создания горизонтальной группы столбцов.

Содержание должно быть размещены в столбцах, и только колонки могут быть непосредственными потомками строк.

Стандартные классы сетки, такие как **.row** и **.col-xs-4** используются для быстрого создания сетки макетов.

Колонки имеют внутренний отступ (**padding**), который для первой и последней колонки компенсируется отрицательным отступом **margin** в **.row**.

Столбцы сетки создаются с учетом максимального количества столбцов – 12, т.е. чтобы создать три блока, то нужно использовать класс **.col-md-4**.

Если в одном ряду размещено более 12 колонок (суммарное значение ширины всех колонок), то каждая последующая колонка будет перенесена на новый ряд.

Настройки сетки

	(<768px)	>=768px)	(>=992px)	(>=1200px)
Поведение сетки	Горизонтальное всегда	Сжатая изначально		
Ширина контейнера Container width	None (auto)	750px	970px	1170px
Префикс класса Class prefix	.col-xs-	.col-sm-	.col-md-	.col-lg-
# колонок	12			
Ширина колонки Column width	Auto	~62px	~81px	~97px
Ширина зазора Gutter width	30px (15px с каждой стороны колонки)			

Вложение	Yes
Смещение	Yes
Упорядочение колонок	Yes

Используя класс **.col-md-***, вы создаете сетку, которая начинает складываться (образуя стек) на мобильных устройствах; при этом на средних устройствах ячейки расположены горизонтально. Расположите колонки сетки в любой **.row**.

HTML

```


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>


.col-md-1</div>



.col-md-8</div>


.col-md-4</div>



.col-md-4</div>


.col-md-4</div>


.col-md-4</div>



.col-md-6</div>


.col-md-6</div>



Пример: жидкий контейнер. Поменяйте сетку с фиксированной шириной на сетку с шириной на все окно браузера при помощи замены .container на .container-fluid.


```

HTML

```
<div class="container-fluid">
  <div class="row">
    ...
  </div>
</div>
```

Пример: мобильники и настольные устройства. Не хотите, чтобы ваши колонки складывались на мобильных устройствах? Применяйте к колонкам классы для небольших и средних устройств: **.col-xs-***, **.col-md-***.

HTML

```
<div class="row">
<div class="col-xs-12 col-md-8">.col-xs-12 .col-md-8
</div>
<div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
<!-- Columns start at 50% wide on mobile and bump up
to 33.3% wide on desktop -->
<div class="row">
<div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
<div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
<div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4</div>
</div>
<!-- Columns are always 50% wide, on mobile and
desktop
<div class="row">
  <div class="col-xs-6">.col-xs-6</div>
  <div class="col-xs-6">.col-xs-6</div>
</div>
```

Пример: мобильники, планшеты и настольные устройства

Основываясь на предыдущем примере создаем еще более мощный и динамический макет с классом для планшетов **.col-sm-***.

HTML

```
<div class="row">
  <div class="col-xs-12 col-sm-6 col-md-8">.col-xs-12
.col-sm-6 .col-md-8</div>
  <div class="col-xs-6 col-md-4">.col-xs-6 .col-md-4
</div>
</div>
<div class="row">
```

```

    <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4
</div>
    <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4
</div>
    <!-- Optional: clear the XS cols if their content
doesn't match in height -->
    <div class="clearfix visible-xs-block"></div>
    <div class="col-xs-6 col-sm-4">.col-xs-6 .col-sm-4
</div>
</div>

```

Пример: смещение колонки на новую строку. Если в строке расположено более 12 колонок, каждая группа дополнительных колонок, как самостоятельная единица сместится на новую линию.

HTML

```

<div class="row">
<div class="col-xs-9">.col-xs-9</div>
<div class="col-xs-4">.col-xs-4<br>Since 9 + 4 = 13
>12, this 4-column-wide div gets wrapped onto a new
line as one contiguous unit.</div>
<div class="col-xs-6">.col-xs-6<br>Subsequent columns
continue along the new line.</div>
</div>

```

Сброс для адаптивных колонок. Имея сетку из четырех колонок вы, возможно, столкнетесь с проблемой, когда для соответствующих точек останова ваши столбцы не выровнялись правильно, так как один столбец выше другого. Чтобы исправить это, используйте комбинацию класса **.clearfix** и отзывчивых классов.

HTML

```

<div class="row">
    <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3
</div>
    <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3
</div>
    <!-- Add the extra clearfix for only the required
viewport -->
    <div class="clearfix visible-xs-block"></div>
    <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3
</div>
    <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3
</div>
</div>

```

В конце колонки необходимо очистить смещение.

HTML

```

<div class="row">
  <div class="col-sm-5 col-md-6">.col-sm-5 .col-md-6
</div>
  <div class="col-sm-5 col-sm-offset-2 col-md-6 col-
md-offset-0">.col-sm-5 .col-sm-offset-2 .col-md-6
.col-md-offset-0</div>
</div>
<div class="row">
  <div class="col-sm-6 col-md-5 col-lg-6">.col-sm-6
.col-md-5 .col-lg-6</div>
  <div class="col-sm-6 col-md-5 col-md-offset-2 col-
lg-6 col-lg-offset-0">.col-sm-6 .col-md-5 .col-md-
offset-2 .col-lg-6 .col-lg-offset-0</div>
</div>

```

Смещение колонок. Сместите колонки направо, используя класс **.col-md-offset-***. Эти классы увеличивают левое поле колонки на * колонок. Например, **.col-md-offset-4** сместит **.col-md-4** на четыре колонки.

HTML

```

<div class="row">
  <div class="col-md-4">.col-md-4</div>
  <div class="col-md-4 col-md-offset-4">.col-md-4
.col-md-offset-4</div>
</div>
<div class="row">
  <div class="col-md-3 col-md-offset-3">.col-md-3
.col-md-offset-3</div>
  <div class="col-md-3 col-md-offset-3">.col-md-3
.col-md-offset-3</div>
</div>
<div class="row">
  <div class="col-md-6 col-md-offset-3">.col-md-6
.col-md-offset-3</div>
</div>

```

Вложенные колонки. Чтобы вложить ваш контент в существующую сетку, добавьте новую **.row** и установите **.col-md-*** колонок внутри уже существующей **.col-sm-*** колонки.

HTML

```

<div class="row">
  <div class="col-sm-9">
    Level 1: .col-sm-9
    <div class="row">
      <div class="col-xs-8 col-sm-6">
        Level 2: .col-xs-8 .col-sm-6

```

```

    </div>
    <div class="col-xs-4 col-sm-6">
      Level 2: .col-xs-4 .col-sm-6
    </div>
  </div>
</div>
</div>

```

Порядок колонок. Порядок для колонок сетки можно изменить при помощи классов **.col-md-push-*** и **.col-md-pull-***.

HTML

```

<div class="row">
  <div class="col-md-9 col-md-push-3">.col-md-9 .col-
md-push-3</div>
  <div class="col-md-3 col-md-pull-9">.col-md-3 .col-
md-pull-9</div>
</div>

```

Использование Bootstrap для дизайна блога

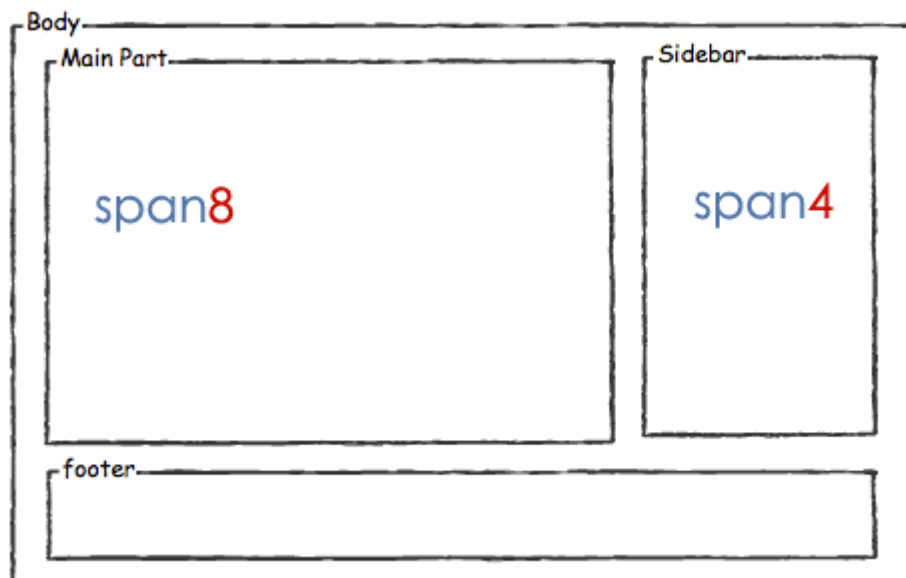
Разметка: *Div* «container» разделён на две части: на основную область **span8** для статей и боковую панель **span4** для другой информации. Здесь применен класс **row-fluid** для гибкого дизайна и правильного отображения на разных устройствах. Это позволяет автоматически изменять размеры *div*'ов и изображений.

```

<div class="container">
<div class="row-fluid">
<div class="span8">
// Основная часть
</div>
<div class="span4">
// Боковая панель
</div>
</div>
</div>

```

Схема разметки приведенного кода.




Основная часть

Здесь расположен список статей, содержащий заголовок, описание и картинку. `btn` — это класс кнопки «read more» («читать далее»).

```
<div class="span8">
// Основная часть
<div class="row">
// Список постов
<div class="span7">
<h2>Article Title.</h2>
<p>Article Description.</p>
<p><img src='ArticleImage.png' class="row-fluid"></p>
<p><a class="btn" href="#">Read More »</a></p>
</div>
</div>
// Конец списка постов
</div>
</div>
```

Для разных цветов кнопок, просто примените к ним следующие классы.

Button	class=""
	<code>btn</code>
	<code>btn btn-primary</code>
	<code>btn btn-info</code>
	<code>btn btn-success</code>
	<code>btn btn-warning</code>
	<code>btn btn-danger</code>
	<code>btn btn-inverse</code>

Боковая панель

Это блок для информации об авторе, виджетов, последних постов и других материалов.

```
<div class="span4 ">
// Боковая панель
<div class="well sidebar-nav">
<ul class="nav nav-list">
<li class="nav-header">About Me</li>
<li>Author Data</li>
<li class="nav-header">Recent Posts</li>
<li>link1</li>
<li>link1</li>
<li>link1</li>
<li class="nav-header">Advertisements</li>
<li>ad1</li>
<li>ad2</li>
</ul>
</div>
</div>
```

Навигация

Панель навигации в фиксированном верхнем меню.

```
<div class="navbar navbar-fixed-top">
<div class="navbar-inner">
// Кнопка меню для iPhone
<a class="btn btn-navbar" data-toggle="collapse" data-
target=".nav-collapse">
<span class="icon-bar"></span>
```

```
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</a>
  <a class="brand" href="#">9LESSONS.info</a>
<div class="nav-collapse">
<ul class="nav">
<li class="active"><a href="#">Home</a></li>
<li><a href="#">Tutorials</a></li>
<li><a href="#">Demos</a></li>
</ul>
</div>
</div>
</div>
```

Медиа-запросы.

Для сетки используются следующие медиа-запросы:

```
/* Совсем маленькие устройства (phones, меньше 768px) */
```

```
/* Медиа-запросы отсутствуют, так как эти стили стоят в
Bootstrap по умолчанию */
```

```
/* Небольшие устройства (планшеты, 768px и выше) */
```

```
@media (min-width: @screen-sm-min) { ... }
```

```
/* Средние устройства (мониторы, 992px and up) */
```

```
@media (min-width: @screen-md-min) { ... }
```

```
/* Большие устройства (большие мониторы, 1200px and up) */
```

```
@media (min-width: @screen-lg-min) { ... }
```

4 CSS Grid

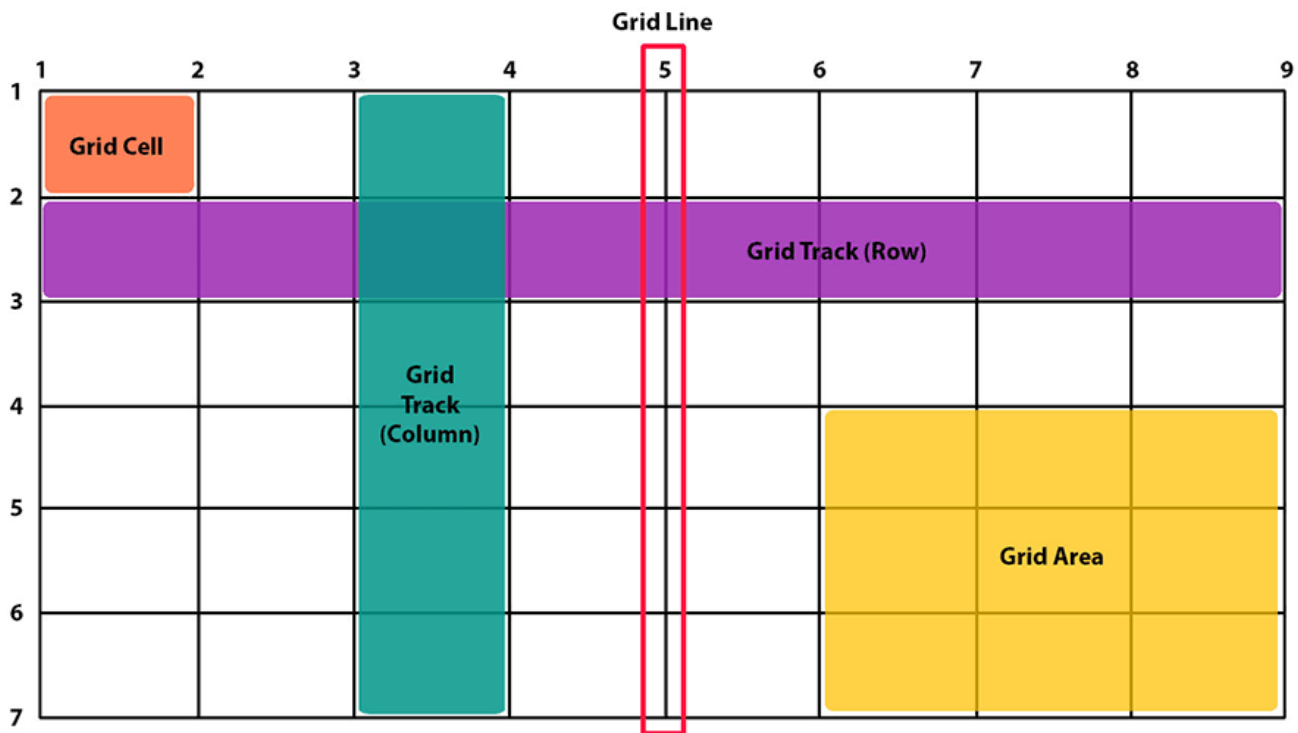
A) Введение

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 браузеры начали его поддержку в марте 2017 года. CSS Grid скоро станет неотъемлемой частью набора инструментов любого фронт-энд разработчика.

CSS Grid это новая модель шаблона, идеальная для сайтов, форм, галерей и всего, что требует точного и отзывчивого позиционирования.

Flexbox уже позволил разработчикам начать уходить от флоат элементов, хотя Flexbox работает только в одном измерении. Grid CSS же это делает в двух, таким образом лучше подходя для создания сложных шаблонов.

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.



Grid container — это набор пересекающихся горизонтальных и вертикальных grid линий, которые делят пространство grid контейнера на grid области, в которые могут быть помещены grid элементы. Внутри grid контейнера есть два набора grid линий: один определяет ось столбцов, другой определяет ось строк.

Grid lines — это горизонтальные и вертикальные разделители grid контейнера. Эти линии находятся по обе стороны от столбца или строки. Автор может задать для данного элемента имя или числовой индекс, которые может использовать дальше в стилях. Нумерация начинается с единицы.

Grid track — это пространство между двумя смежными grid линиями, вертикальными или горизонтальными.

Grid cell — это наименьшая неделимая единица grid контейнера на которую можно ссылаться при позиционировании grid элементов. Образуется на пересечении grid строки и grid колонки.

Grid area — это пространство внутри grid контейнера, в которое может быть помещен один или больше grid элементов. Этот элемент может состоять из одной или более grid ячеек.

Каждый элемент тесно связан друг с другом и отвечает за определенную часть grid контейнера.

Б) Создаем Grid - виртуальную сетку

Два главных компонента CSS Grid – это обертка (**wrapper**) и элементы, которые она содержит.

Чтобы создать **Grid-элемент**, просто нужно выставить элементу-контейнеру свойство **display: grid**. Это автоматически сделает всех прямых потомков этого элемента — **grid - элементами**.

HTML разметка для CSS Grid выглядит вот так:

```
<div id="grid">
<div>1</div>
<div>2</div>
<div>3</div>
<div>4</div>
</div>
```

По-факту, это не будет полноценным `grid` - элементом, пока мы не применим правило CSS для него:

```
#grid
{
display: grid;
}
```

Это правило применяется к внешнему `<div>`, так как ему было назначено ID `#grid`.

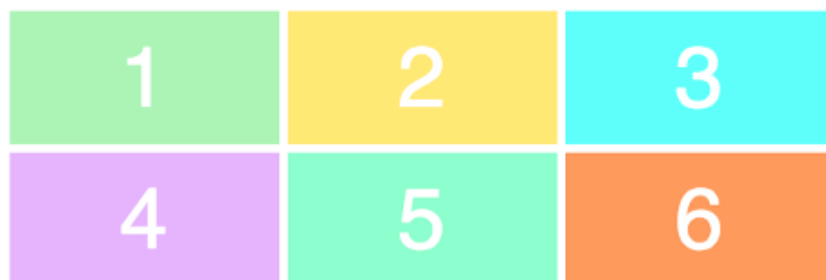
В) Колонки и строки

Чтобы создать **grid -сетку**, нам нужно определить **колонки (columns)** и **строки (rows)**.

Давайте создадим три колонки и две строки, используя свойства **grid-template-row** и **grid-template-column**.

```
.wrapper {
display: grid;
grid-template-columns: 100px 100px 100px;
grid-template-rows: 50px 50px;
}
```

Введя три значения для **grid-template-columns**, мы получаем сетку, состоящую из трех колонок. Аналогично для **grid-template-rows**: 2 значения и две строки.



Г) Расположение элементов

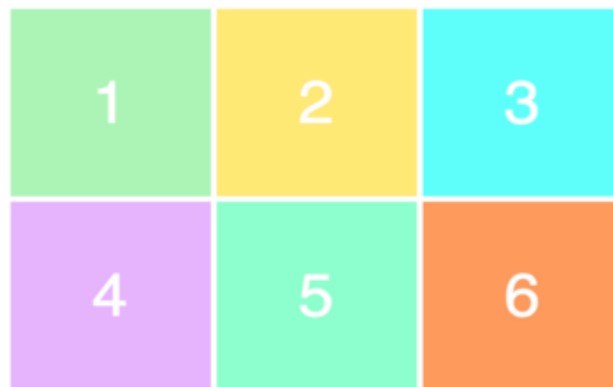
Теперь вам необходимо научиться располагать элементы внутри сетки. Здесь во всей красе предстает главная супер-сила CSS Grid – возможность легко и просто создавать разметку страниц.

Давайте создадим сетку 3x3, используя ту же разметку, что и ранее.

```
.wrapper {
display: grid;
grid-template-columns: 100px 100px 100px;
```

```
    grid-template-rows: 100px 100px 100px;  
}
```

Вот что получилось:



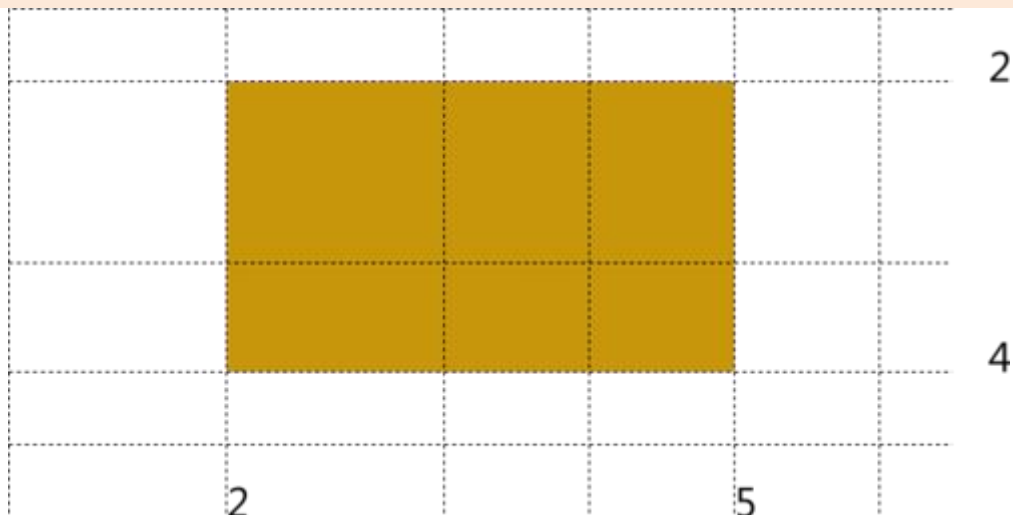
Обратите внимание, что мы видим сетку 3x2 вместо 3x3. Это потому, что в нашей сетке всего 6 элементов, и если мы добавим еще 3 элемента, то третий ряд также заполнится.

Чтобы позиционировать и изменять размеры наших элементов мы используем **grid-column** и **grid-row** свойства.

По умолчанию, элемент занимает пространство от указанной линии до следующей (поэтому такая нумерация совпадает с нумерацией ячеек и все выглядит так, как будто мы говорим, в какую ячейку разместить элемент).

Чтобы растянуть элемент на несколько ячеек сетки, можно использовать свойства **grid-row-span** и **grid-column-span**:

```
#item {  
  grid-column: 2;  
  grid-column-span: 3;  
  grid-row: 2;  
  grid-row-span: 2;  
}
```



Опционально для привязки элемента к сетке можно указать не только начальную, но и конечную линию.

```
#item {  
  grid-column: 2;  
  grid-row: 2 4;  
}
```

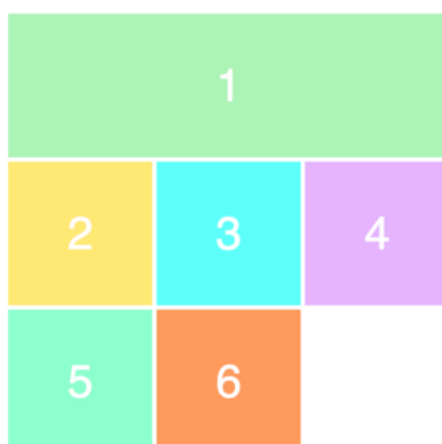


В отличие от механизма «**span**», который говорит, на сколько ячеек элемент должен растянуться по горизонтали или вертикали, данная возможность позволяет четко указать, на какой линии элемент должен закончиться. Это также удобно использовать в сочетании с возможностью именования отдельных линий сетки.

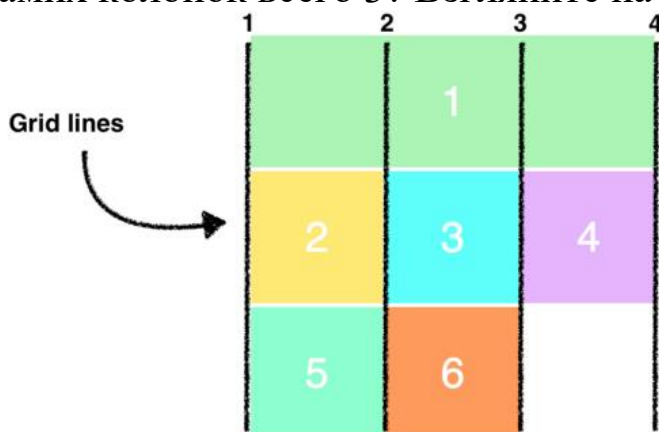
Таким образом, чтобы задать, как именно элемент будет размещен по сетке, необходимо указать, к какой линии по вертикали и горизонтали он будет привязан:

```
.item1 {  
  grid-column-start: 1;  
  grid-column-end: 4;  
}
```

Этот код делает так, чтобы **item1** начинался в первой строке и первой колонке сетки и заканчивался в четвертой колонке. Иными словами, **item1** теперь будет занимать весь ряд.



Наверное, вас немного сбивает с толку, почему у нас 4 разделения на колонки, но самих колонок всего 3? Взгляните на эту картинку:



Когда `item1` занял весь первый ряд, он вытеснил остальные элементы вниз.

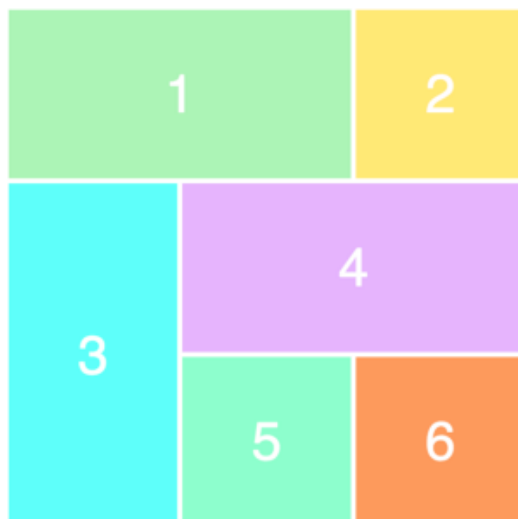
А вот таким образом можно записать этот синтаксис в упрощенном виде:

```
.item1 {  
    grid-column: 1 / 4;  
}
```

Чтобы вы полностью поняли концепцию, давайте снова изменим элементы.

```
.item1 {  
    grid-column-start: 1;  
    grid-column-end: 3;  
}  
.item3 {  
    grid-row-start: 2;  
    grid-row-end: 4;  
}  
.item4 {  
    grid-column-start: 2;  
    grid-column-end: 4;  
}
```

Вот как этот код отобразится на странице.



Д) Повторяющиеся блоки

Сам шаблон описывается в круглых скобках, после чего в квадратных указывается количество повторений.

```
#grid {  
  display: grid;  
  grid-columns: 24px (120px 24px) [4];  
  grid-rows: (1fr 24px) [3];  
}
```

Е) Единицы измерения

При описании ширины колонок и высоты строк (размеров блоков) можно использовать следующие единицы и значения:

линейные размеры — стандартные единицы указания длины, определенные в модуле CSS3 Values and Units, например, pt, px, em и т. д.;

проценты — размер трека в процентах от размеров контейнера с сеткой (хотя если высота или длина сетки зависит от контента, результат будет неопределенным);

max-content — ключевое слово для указания максимальной длины из *максимальных* длин элементов в треке;

min-content — ключевое слово для указания максимальной длины из *минимальных* длин элементов в треке;

minmax(min, max) — задает диапазон значений (принцип работы можно описать как $\text{minmax}(p, q) = \text{max}(p, \text{min}(\text{fill-available}, q))$) — максимум из нижнего порога и минимума доступного пространства и верхнего порога);

auto — ключевое слово, эквивалентное $\text{minmax}(\text{min-content}, \text{max-content})$.

доли (fraction) — неотрицательное число с последующей единицей измерения *fr*, размер каждой доли берется пропорциональным указанному числу (см. подробнее ниже);

ж) Доли

Давайте попробуем разобраться, как работают доли (fraction value)? Сетка занимает некоторое пространство по ширине и высоте. Оно может зависеть от контента, быть жестко фиксированным или занимать все доступное пространство во внешнем контейнере. Далее, при описании треков части колонок и строк вы можете явно задать, какого размера они должны быть, для еще какой-то части вы можете указать, что их длина зависит от контента.

Теперь, если из доступной длины, отведенной под сетку по вертикали или горизонтали, вычесть сумму всех таких явных или «контентных» длин, оставшееся пространство распределяется между остальными треками пропорционально указанному в них долям (размер доли, деленный на сумму всех таких долей):



На примере выше это три столбца с ширинами в соотношении 2:1:1 и две строки с высотами в соотношении 5:2.

Пример

```
#grid {  
  display: grid;  
  grid-columns: 100px 1fr max-content minmax(min-content, 1fr)  
}
```

Здесь определяются следующие линии:

1. Стартовая первая линия.
2. Линия в 100px от первой.
3. Еще одна линия на расстоянии 1/2 оставшегося после всех расчётов пространства — от второй линии.
4. И еще одна линия, чье расстояние от третьей равно максимальному из размеров контента элементов в колонке.
5. Наконец, последняя линия, расположенная от четвертой на расстоянии, равном либо минимальной длине элементов в колонке, либо 1/2 оставшегося пространства, смотря, что больше.

3) Привязка элементов

Теперь, когда элементы «привязаны» к линиям сетки, возникает естественный вопрос: а как же они располагаются между линиями?

Привязка элементов к границам ячейки контролируется с помощью свойств **grid-column-align** и **grid-row-align**.

Для управления можно использовать следующие значения:

- start
- end
- center
- stretch

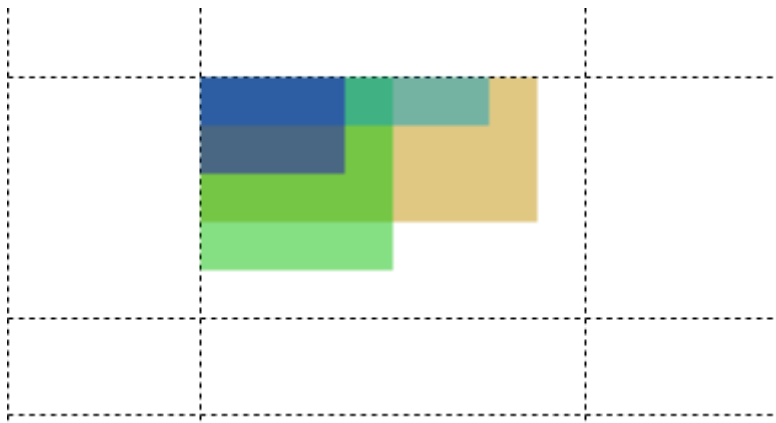
По умолчанию используется значение **stretch**.

start/start			start/stretch
	start/center	start/end	
center/start	center/center	center/end	center/stretch
end/start	end/center	end/end	end/stretch
stretch/start	stretch/center	stretch/end	stretch/stretch

И) Управление слоями

Следующий важный момент: расположение элементов внутри сетки с наложениями. Что происходит, если, к примеру, два элемента привязаны к одним и тем же линиям, либо накладываются при расширении на несколько ячеек?

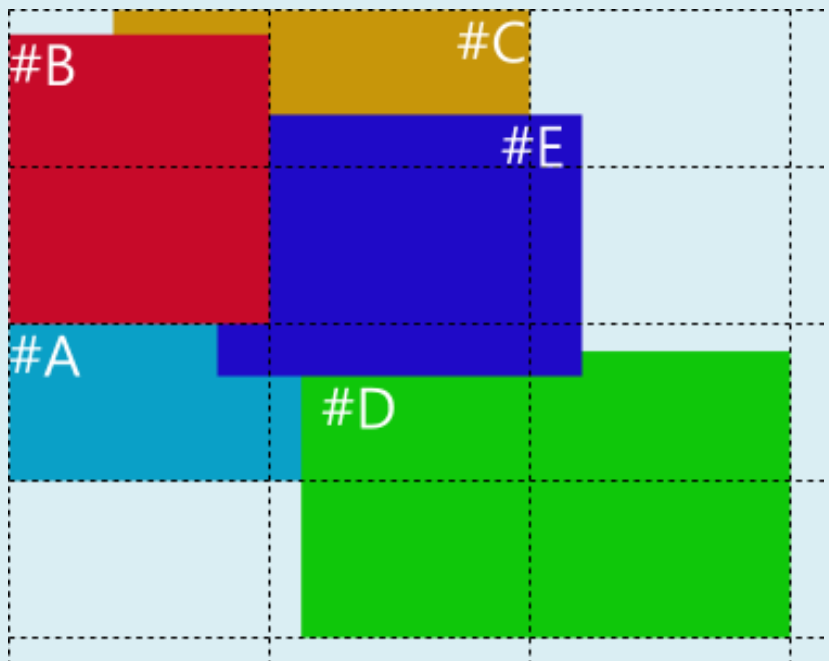
Прежде всего, важно понять следующий нюанс: элементы, размещаемые внутри сетки, не влияют напрямую на расположение друг друга. Если вы десять элементов привяжете, скажем, ко второй линии по горизонтали и третьей по вертикали, то по умолчанию они все расположатся один на другом так, как будто каждый из них привязан к соответствующему углу. Элементы могут влиять только на размеры треков, если они завязаны, в свою очередь, на размеры контента.



Чтобы управлять порядком отображения таких слоев, текущая версия спецификации расширяет возможности **z-index**, позволяя управлять слоями элементов внутри сетки.

Пример использования:

```
#grid {
display: grid;
grid-columns: (1fr) [3];
grid-rows: (1fr) [4];}
#A { grid-column:1;
grid-row:3;
grid-column-span:2;}
#B { grid-column:1;
grid-row:1;
grid-row-span:2;
/* grid-layer: 10; */
z-index:10;
margin-top:10px;}
#C { grid-column:1;
grid-row:1;
grid-column-span:2;
margin-left:50px;}
#D { grid-column:2;
grid-row:3;
grid-row-span:2;
grid-column-span:2;
margin:10px 0 0 10px;}
#E { grid-column:2;
grid-row:2;
/* grid-layer: 5; */
z-index:5;
margin: -20px;}
```



К) Именованные линии сетки

Для удобства линиям можно давать названия. Это делается вставкой в соответствующих местах строковых значений при описании треков (можно давать несколько имен, если это имеет практический смысл, например, с точки зрения семантики разметки):

```
#grid { display: grid;
grid-columns: "nav" "first" 150px "content" 1fr "last";
grid-rows: "header" 50px "content" 1fr "footer" 50px;}
```

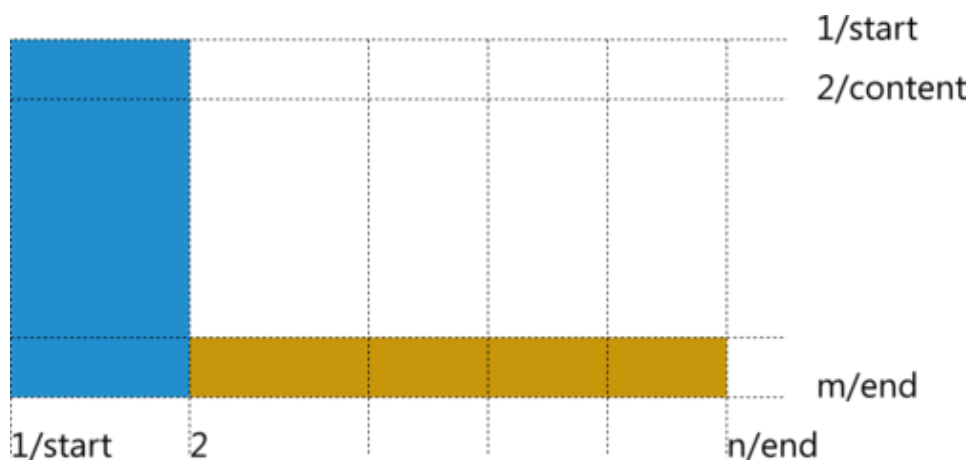
Далее при описании привязки элементов можно ссылаться на эти имена:

```
#menu {
grid-column: "nav";
grid-row: "content";}
#header {
grid-column: "content";
grid-row: "header";}
#article {
grid-column: "content";
grid-row: "content";}
```



Также спецификация вводит 4 заранее именованных линии — вертикальные и горизонтальные **start** и **end**, фактически, обрамляющие всю сетку. Это позволяет, к примеру, расположить элемент «от второго столбца и до последнего», не задумываясь об общем количестве столбцов.

```
#menu {
grid-column: 1;
grid-row: start end;
}
#footer {
grid-column: 2 end;
grid-row: 3;
}
```



Л) Именованные ячейки и шаблоны

Еще один способ размещения элементов по сетке заключается в использовании шаблонов, позволяющих описать виртуальную структуру блоков:

```
#grid {
display: grid;
grid-template: "ln"
"ma"
"ba"
"ff";
grid-columns: auto minmax(min-content, 1fr);
grid-rows: auto minmax(min-content, 1fr) auto auto;
}
```

logo	navigation
menu	article
banner	article
footer	footer

При этом для размещения элемента с привязкой к той или иной виртуальной ячейке достаточно сослаться на нее соответствующим правилом:

```
#article {
grid-cell: "a";
}
```

Такой подход оказывается особенно удобным, если в зависимости от различных условий, например, разрешения экрана, вам нужно менять расположение элементов и даже переделать саму сетку.

В подобной ситуации **Grid Layout** хорошо сочетается с **Media Queries**:

```
@media (orientation: portrait) {
  #grid {
    display: grid;
    grid-template: "ln"
      "ma"
      "ba"
      "ff";
    grid-columns: auto minmax(min-content, 1fr);
    grid-rows: auto minmax(min-content, 1fr) auto auto;
  }
}
@media (orientation: landscape) {
  #grid {
    display: grid;
    grid-template: "ln"
      "ma"
      "mb"
      "sf";
    grid-columns: auto minmax(min-content, 1fr);
    grid-rows: auto minmax(min-content, 1fr) auto auto;
  }
}
#article {
  grid-cell: "a";
}
```

logo	navigation	
menu	article	
menu	banner	
space	footer	

Обратите внимание, что привязка самой статьи к именованной ячейке при этом не меняется.

Примеры

```
body {
  display: grid;
  grid: "header header header" 80px
        "nav section aside" 1fr
        "footer footer footer" 50px
        / 15% 1fr 18%;
  min-height: 100vh;
}
header { grid-area: header; }
nav { grid-area: nav; }
section { grid-area: section; }
aside { grid-area: aside; }
footer { grid-area: footer; }
/* стили для наглядности */
body{ margin:0; font-family: sans-serif; }
body > *{ padding:1em; }
header { background: #daebe8; }
nav { background: #cfe0e8; }
section { background: #b7d7e8; }
aside { background: #bccad6; }
footer { background: #b3d4ce; }
```

Каркас HTML страницы

```
<body>
  <header>header</header>
  <nav>nav</nav>
  <section>section</section>
  <aside>aside</aside>
  <footer>footer</footer>
</body>
```

header

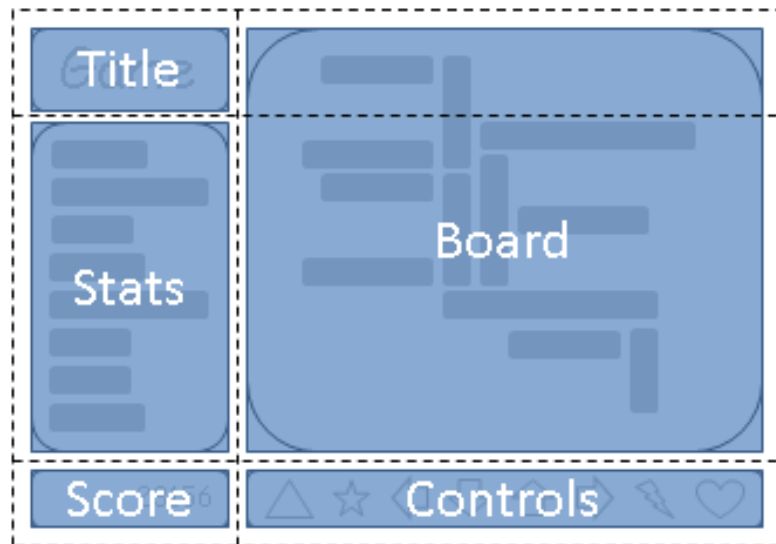
nav

section

aside

footer

Каркас для игрового приложения



Две колонки:

1. подстраивается под контент,
2. получает все оставшееся пространство (но не может быть меньше минимального размера контента в ней или controls блока в этой же колонке)

Три ряда:

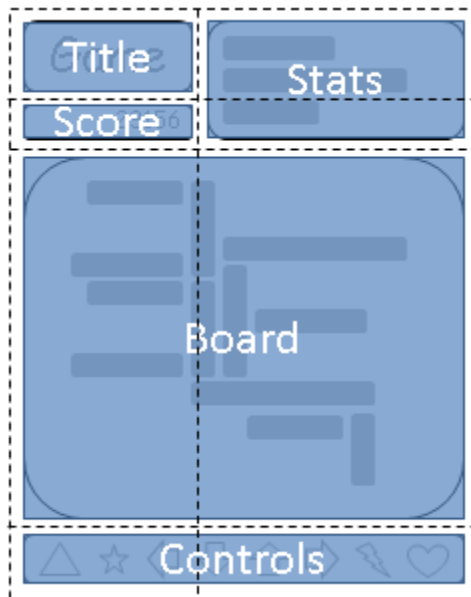
1. подстраивается под контент,
2. получает все свободное пространство (но не может быть меньше минимального размера любого из блоков в этом ряду)
3. подстраивается под контент.

```
#grid {
  display: grid;
  grid-template-columns:
    /* 1 */ auto
    /* 2 */ 1fr;
  grid-template-rows:
    /* 1 */ auto
    /* 2 */ 1fr
    /* 3 */ auto;
  /* растянем на всю высоту */
  height: 100vh; }
/* указывает позиции блоков в сетке с помощью
координат */
#title      { grid-column: 1; grid-row: 1; }
#score      { grid-column: 1; grid-row: 3; }
#stats      { grid-column: 1; grid-row: 2; }
#board      { grid-column: 2; grid-row: 1 / span 2; }
#controls   { grid-column: 2; grid-row: 3; justify-
self: center; }
```

Каркас HTML страницы

```
<div id="grid">
  <div id="title">Game Title</div>
  <div id="score">Score</div>
  <div id="stats">Stats</div>
  <div id="board">Board</div>
  <div id="controls">Controls</div>
</div>
```

Усложним задачу. Теперь нам нужно сделать так чтобы при повороте мобильного устройства, каркас менялся и получалось так:



В этом случае удобнее будет использовать области Grid сетки.

```
/* книжная ориентация */
@media (orientation: portrait) {
  #grid { display: grid;
    /* создадим структуру сетки и укажем названия областей, такая структура будет работать по умолчанию и подходить для альбомной ориентации. */
    grid-template-areas:
      "title stats"
      "score stats"
      "board board"
      "ctrls ctrls";
    /* укажем размеры для рядов и колонок. */
    grid-template-columns: auto 1fr;
    grid-template-rows: auto auto 1fr auto;
    height: 100vh;
  }
}
/* альбомная ориентация */
```

```

@media (orientation: landscape) {
  #grid {
    display: grid;

    /* создадим структуру сетки и укажем названия
    областей, Такая структура будет работать по
    умолчанию и подходить для альбомной ориентации.*/
    grid-template-areas:
      "title board"
      "stats board"
      "score ctrls";
    /* укажем размеры для рядов и колонок. */
    grid-template-columns: auto 1fr;
    grid-template-rows: auto 1fr auto;
    height: 100vh;
  }
}

/*расположим элементы в именованные областях
сетки */
#title      { grid-area: title; }
#score      { grid-area: score; }
#stats      { grid-area: stats; }
#board      { grid-area: board; }
#controls   { grid-area: ctrls; justify-self:
center; }

/* стили для наглядности */
body{ margin:0; font-family: sans-serif; }
#grid > *{ padding:1em; }
#title      { background: #daebe8; }
#score      { background: #cfe0e8; }
#stats      { background: #b7d7e8; }
#board      { background: #bccad6; }
#controls   { background: #b7d7e8; }

```

Каркас HTML страницы

```

<div id="grid">
  <div id="title">Game Title</div>
  <div id="score">Score</div>
  <div id="stats">Stats</div>
  <div id="board">Board</div>
  <div id="controls">Controls</div>
</div>

```

CSS Grid или Bootstrap?

1. Разметка станет проще. Замена Bootstrap на CSS Grid сделает HTML более чистым. Хотя это не самое важное преимущество, оно, вероятно, первое, которое вы заметите.

Bootstrap

Давайте для начала рассмотрим разметку, необходимую для создания этого сайта в Bootstrap. Каждая строка должна быть с отдельным тэгом `<div>`. Для обозначения макета должны использоваться имена классов (`col-xs-2`). Когда этот шаблон усложняется, HTML тоже. Если это адаптивный сайт, тэги выглядят как правило ещё сложнее

CSS Grid

Теперь давайте посмотрим на способ реализации того же в CSS Grid. Возможно использовать семантические элементы, но мы будем придерживаться `div`, чтобы сравнение с Bootstrap выглядело понятнее.

Можно заметить сразу, что эта разметка проще. Это просто контейнер для сетки и позиции внутри неё. И в отличие от Bootstrap, эта разметка не станет слишком сложной с ростом сложности макета страницы.

В примере с Bootstrap от вас не требовалось добавлять CSS, в CSS Grid в этом конечно есть необходимость. Для некоторых это может быть аргументом в пользу Bootstrap: вам не нужно беспокоиться о CSS для создания простой сетки — вы просто строите макет в HTML.

Но связь между разметкой и макетом на самом деле — уязвимость, если речь идёт о гибкости. Предположим, вы хотите менять макет в зависимости от размера экрана. Например, поднимать меню в верхний ряд, для просмотра с мобильных устройств.

CSS Grid

Делать это с CSS Grid очень просто. Мы добавляем `media`-запрос и перемешиваем разные блоки как хотим. Возможность создания макета подобным образом — не заботясь о том, как написан HTML — называется независимостью от порядка в коде (`source order independence`), и это гигантская победа для разработчиков и дизайнеров.

Bootstrap

Если бы мы захотели сделать то же самое в Bootstrap, нам бы потребовалось менять HTML. Нам нужно было бы поднять тег `menu` в верхний ряд, в добавок к `header`, потому что `menu` во втором ряду в заложниках.

Сделать такое с наличием медиа-запроса — не тривиальная задача. Такое можно проверить только вместе с HTML и CSS, и вам бы пришлось возиться с JavaScript. + Больше никаких ограничений 12 колонками.

Этот пример показывает преимущество CSS Grid.