

HyperText Markup Language

HTML & CSS

Урок 9

Grid Layout

Оглавление

Что такое Grid Layout?	4
Знакомство с Grid Layout	4
Основные понятия и компоненты сетки	4
Как создать Grid-контейнер?.....	7
Строка, столбец	13
Определение количества и размера столбцов	14
Определение количества и размера строк.....	15
Размеры линий сетки и единицы определения треков сетки	17
Размеры линий сетки.....	17
Значение для определения треков сетки.....	18
Гибкие размеры треков: единица измерения fr	19
Автоматические размеры	20
Функция repeat	21

Позиционирование элементов и Grid-линии	24
Позиционирование элементов сетки с помощью номеров линий сетки	24
Сокращенные формы записи для позиционирования	27
Объединение ячеек с помощью ключевого слова span.....	29
Именованные линии сетки	29
Именованные строки и столбцы и ключевое слово span	31
Области сетки	31
Выравнивание в Grid	35
Выравнивание всей сетки.....	35
Выравнивание всех элементов сетки.....	37
Выравнивание отдельных элементов сетки	39
Grid или Flexbox: что выбрать?	42
Flexbox-подход.....	46
Grid-подход	47
Объединяем два подхода	49
Практические примеры использования.....	52
Домашнее задание.....	60

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе [Adobe Acrobat Reader](#).

Что такое Grid Layout?

Знакомство с Grid Layout

Появление CSS Grid Layout — это важное событие во Front-end разработке, призванное упростить построение веб-макетов. CSS уже давно используется для компоновки элементов на странице. Сначала веб-разработчики использовали таблицы, позже — плавающие элементы (свойство `float`), позиционирование и строчные блоки. Однако эти инструменты и средства не всегда гибко и качественно выполняли работу по структурированию страниц. С появлением модели Flexbox ситуация улучшилась, но эта модель рассчитана на одномерные макеты. Правильно будет сказать, что Flexbox выполняет свои задачи, а Grid — свои.

Когда и как лучше использовать Flexbox и Grid? К этому вопросу мы еще вернемся в конце урока, а сейчас давайте поближе познакомимся с Grid.

Grid Layout — система двумерной компоновки, используемая для создания дизайна пользовательского интерфейса. В основе макета лежит сетка, разделяющая страницу на строки и столбцы, и позволяющая управлять размещением, размерами и выравниванием дочерних элементов. Благодаря свойству явно размещать элементы в сетке, Grid позволяет кардинально преобразовывать структуру макета, не изменяя при этом разметку документа.

Основные понятия и компоненты сетки

Прежде чем мы перейдем к рассмотрению возможностей Grid, необходимо ознакомиться с основными компонентами сетки.

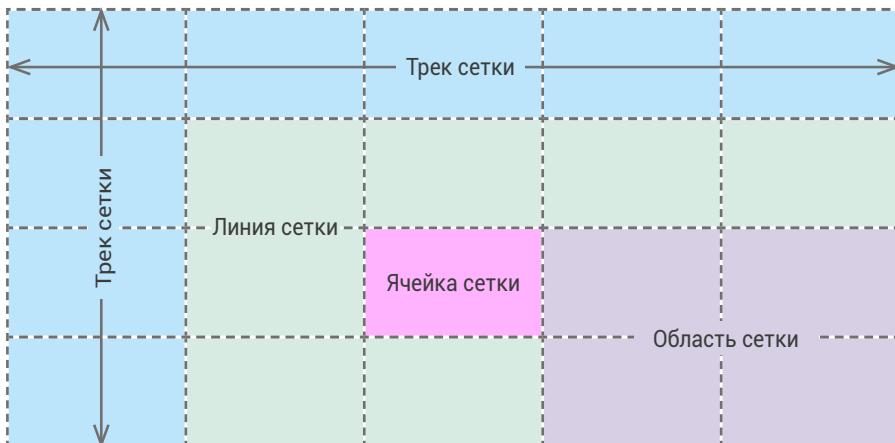


Рисунок 1

- **Контейнер сетки (Grid Container)** — главный родительский элемент для всех элементов сетки.
- **Элемент сетки (Grid Item)** — дочерний элемент (прямой потомок) контейнера сетки.
- **Линия сетки (Grid Line)** — линия, которая разделяет элементы сетки. Она может быть горизонтальной или вертикальной.
- **Трек сетки (Grid Track)** — пространство между двумя параллельными линиями сетки. Можно считать, что это строка или столбец сетки.
- **Ячейка сетки (Grid Cell)** — пространство между двумя соседними строками и двумя соседними столбцами линий сетки, аналогично ячейке в таблицы. Это область, в которую можно что-то поместить. Ячейка является наименьшей единицей сетки, на которую можно ссылаться при позиционировании элементов сетки.

- **Область сетки (Grid Area)** — любое пространство между четырьмя линиями сетки может состоять из любого количества ячеек. Область сетки может быть не больше одной ячейки или размером со все ячейки сетки.

Далее в уроке мы подробно рассмотрим эти компоненты и научимся использовать их для управления сеткой.

Как создать Grid-контейнер?

Для построения макета на основе сетки, сначала необходимо определить контейнер сетки.

Контейнер сетки — это блок, который создает область с сеткой и устанавливает контекст форматирования по типу сетки, то есть дочерние элементы располагаются согласно правилам компоновки сетки, а не блочной компоновки. Контейнер сетки определяется при помощи `display: grid` или `display: inline-grid`. Как только мы это сделаем, все прямые дочерние элементы контейнера автоматически станут элементами сетки.

Рассмотрим небольшой пример. Создадим разметку:

```
<div class="grid-container">
    <div class="item-1">child-item-1</div>
    <div class="item-2">child-item-2</div>
    <div class="item-3">child-item-3</div>
</div>
```

Зададим стили:

```
.grid-container {
    display: grid;
    padding: 15px 30px;
    border-radius: 5px;
    background-color: #0d233d;
    color: #ffffff;
    font-size: 20px;
    font-family: Arial, Helvetica, sans-serif;
}
```

```
.grid-container > div {  
    margin: 10px 0;  
    padding: 0.3em;  
    border-radius: 3px;  
    font-size: 130%;  
    font-weight: bold;  
}  
  
.item-1 {  
    background-color: #aa2f26;  
}  
  
.item-2 {  
    background-color: #01796f;  
}  
  
.item-3 {  
    background-color: #1769aa;  
}
```

Результат:



Рисунок 2

Для элемента с классом `grid-container` мы задали `display: grid`. Таким образом все три непосредственных дочерних элемента этого контейнера автоматически стали элементами сетки.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Когда мы создаем контейнер сетки, сетка по умолчанию имеет один столбец и одну строку, которые занимают полный размер контейнера.

Контейнеры сетки не являются блочными контейнерами, поэтому некоторые CSS-свойства в контексте контейнера сетки работать не будут.

Свойство контейнера сетки `display` объявляет, что текущий элемент будет использоваться как контейнер сетки и устанавливает новый контекст форматирования для его дочерних элементов.

Возможные значения:

- `grid` — генерирует контейнер сетки уровня блока.
- `inline-grid` — генерирует контейнер сетки уровня строки.
- `subgrid` — если контейнер также является элементом другой сетки (вложенный контейнер), можно указать, что параметры для текущей сетки следует взять из родительской.

Рассмотрим пример создания контейнера сетки. Создадим несколько HTML-элементов:

```
<div class="container">
  <div class="grid-container">display: grid;</div>

  <p>
    Lorem, ipsum dolor sit amet consectetur
```

```
adipisicing elit. Possimus cumque  
necessitatibus ipsum suscipit cupiditate  
iusto commodi, aut, expedita tempore  
perspiciatis asperiores fugit illo aperiam.  
</p>  
  
<div class="inline-grid-container">  
    display: inline-grid;  
</div>  
  
<p>  
    Lorem ipsum dolor sit, amet consectetur  
    adipisicing elit. Quidem deleniti,  
    necessitatibus fugit, voluptates quod minus  
    molestias minima ex, laudantium quasi  
    quo adipisci voluptas nesciunt perspiciatis.  
</p>  
</div>
```

Зададим для них стили:

```
.container {  
    padding: 30px;  
    border-radius: 5px;  
    background-color: #0d233d;  
    color: #ffffff;  
    font-size: 20px;  
    font-family: Arial, Helvetica, sans-serif;  
}  
  
.container > div {  
    padding: 0.3em;  
    border-radius: 3px;  
    font-size: 130%;  
    font-weight: bold;  
}
```

```
.grid-container {  
    display: grid;  
    background-color: #b2a429;  
}  
  
.inline-grid-container {  
    display: inline-grid;  
    background-color: #aa2f26;  
}  
  
p {  
    color: rgb(245, 248, 249);  
}
```

Результат:

display: grid;

 Lorem, ipsum dolor sit amet consectetur adipisicing elit. Possimus cumque necessitatibus ipsum suscipit cupiditate iusto commodi, aut, expedita tempore perspiciatis asperiores fugit illo aperiam.

display: inline-grid;

 Lorem ipsum dolor sit, amet consectetur adipisicing elit. Quidem deleniti, necessitatibus fugit, voluptates quod minus molestias minima ex, laudantium quasi quos adipisci voluptas nesciunt perspiciatis.

Рисунок 3

В примере показано, что, когда мы задаем для свойства **display** значение **grid**, мы определяем контейнер сетки, который будет вести себя как блочный элемент, то есть занимать все доступное пространство родительского элемента, даже если содержимое сетки будет небольшим.

Если задать значение `inline-grid`, то поведение контейнера сетки будет похоже на поведение строчных элементов, то есть сетка будет занимать ровно столько же места, сколько и ее содержимое.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Строка, столбец

Количество строк и столбцов определяется с помощью свойств `grid-template-rows` и `grid-template-columns`.

Например, создадим разметку:

```
<div class="grid">
  <div class="item-1">1</div>
  <div class="item-2">2</div>
  <div class="item-3">3</div>
  <div class="item-4">4</div>
  <div class="item-5">5</div>
  <div class="item-6">6</div>
  <div class="item-7">7</div>
  <div class="item-8">8</div>
  <div class="item-9">9</div>
</div>
```

Добавим стили:

```
.grid {
  display: grid;
  padding: 20px;
  background-color: #0d233d;
  border-radius: 5px;
  font-family: Arial, Helvetica, sans-serif;
}

.grid div {
  padding: 0.3em;
  border-radius: 3px;
  background-color: #482880;
  color: #ffffff;
  font-size: 130%;
  font-weight: bold;
```

```
    text-align: center;  
}  
  
.grid div:nth-child(odd) {  
    background-color: #1769aa;  
}  
}
```

Результат:

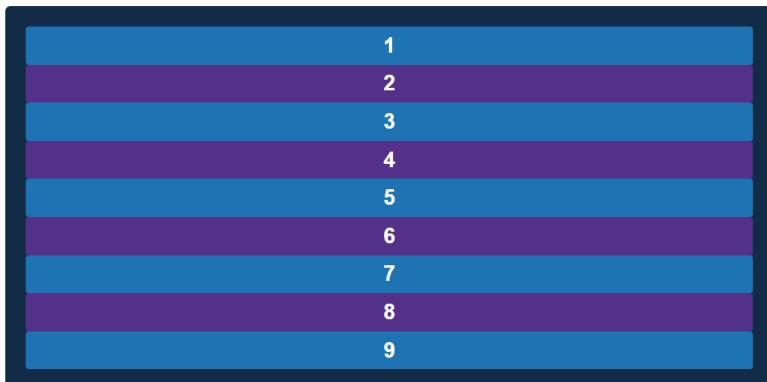


Рисунок 4

Пока что все элементы в сетке расположены последовательно друг над другом.

Определение количества и размера столбцов

Чтобы задать количество и размеры столбцов, используется свойство `grid-template-columns`:

```
.grid {  
    display: grid;  
    grid-template-columns: 100px 200px 300px;  
}  
}
```

Результат:

1	2	3
4	5	6
7	8	9

Рисунок 5

Мы создали 3 столбца и для каждого из них задали ширину `100px`, `200px` и `300px` соответственно. Теперь элементы в сетке расположились в 3 строках и 3 столбцах.

- ▶ С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в [CodePen](#) по [ссылке](#).

Определение количества и размера строк

Чтобы задать количество и размеры строк, используется свойство `grid-template-rows`:

```
grid-template-rows: 50px 150px 100px;
```

Результат:

1	2	3
4	5	6
7	8	9

Рисунок 6

Для каждой строки мы задали высоту **50px** **150px** и **100px** соответственно.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в *CodePen* по [ссылке](#).

Размеры линий сетки и единицы определения треков сетки

Размеры линий сетки

Часто возникает необходимость задать размеры для линий сетки. Такие линии создаются только между ячейками, по краям сетки их нет.

Свойства, которые задают размеры линий сетки:

- `grid-column-gap` — устанавливает ширину вертикальной линии сетки (промежуток между столбцами).
- `grid-row-gap` — устанавливает высоту горизонтальной линии сетки (промежуток между строками).
- `grid-gap` — сокращенная форма записи для `grid-column-gap+grid-row-gap`.

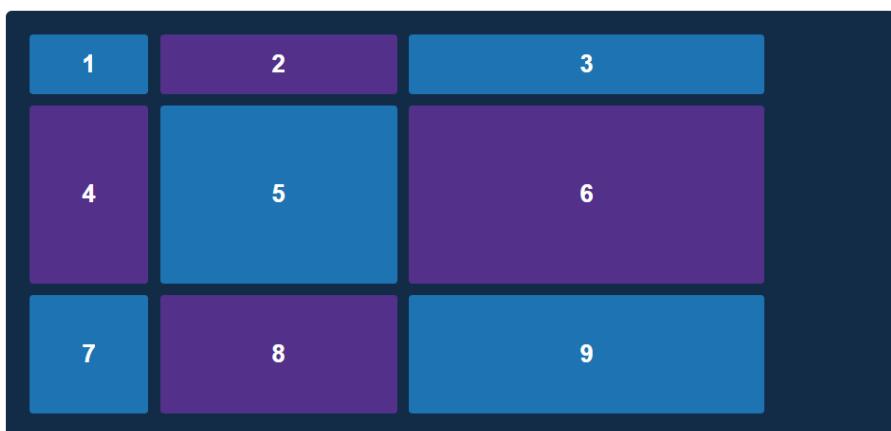


Рисунок 7

Зададим размер в **10px** для вертикальной и горизонтальной линий:

```
grid-gap: 10px;
```

Результат представлен на рисунке 7. Из примера видно, что между элементами сетки появились промежутки, поскольку мы задали ширину для линий сетки, равную **10px**.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в *CodePen* по [ссылке](#).

Значение для определения треков сетки

Размеры треков сетки можно задавать с помощью различных значений, используя *относительные единицы длины*, например, **em**, **vh**, **vw**; *абсолютные единицы длины px*; и *проценты %*. Размеры в **%** высчитываются по ширине или высоте контейнера-сетки.

Зададим значение для размеров треков сетки в разных единицах измерения:

```
grid-template-columns: 100px 10em 30%;  
grid-template-rows: 10em 100px 100px;
```

Результат представлен на рисунке 8. Теперь первый столбец сетки занимает фиксированную ширину **100px**, второй — **10em** — ширина элемента будет пропорционально зависеть от размера шрифта, использованного для текста в блоке. В данном случае ширина элемента в 10 раз больше размера шрифта. Третий столбец занимает **30%** от ширины контейнера сетки.

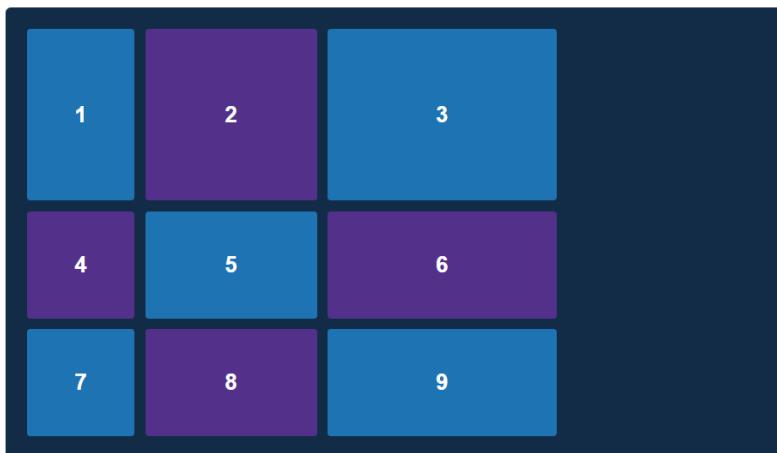


Рисунок 8

Для строк заданы размеры `10em`, `100px` и `100px` соответственно.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в *CodePen* по [ссылке](#).

Гибкие размеры треков: единица измерения `fr`

`Fr` — единица измерения длины, которая позволяет создавать гибкие треки. При определении размеров треков, общий размер фиксированных строк или столбцов вычитается из доступного пространства контейнера-сетки. Оставшееся пространство делится между строками и столбцами с гибкими размерами пропорционально их коэффициенту. Если сумма размеров всех гибких треков меньше 1, они будут занимать только соответствующую часть оставшегося пространства, а не расширяться, чтобы заполнить все пространство полностью.

Если ширина или высота контейнера-сетки не заданы, треки сетки масштабируются по их содержимому, сохраняя при этом пропорции.

Зададим значение для размеров треков в единицах **fr**:

```
grid-template-columns: 2fr 1fr 1fr;
```

Результат:

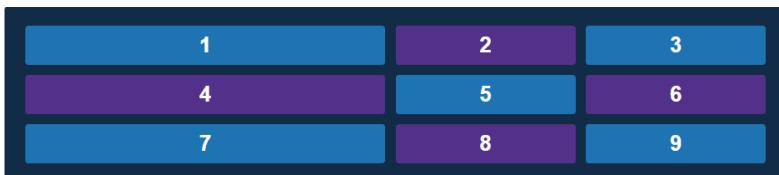


Рисунок 9

В примере выше сетка разделена на 3 столбца. Первый столбец занимает половину ширины контейнера, а второй и третий распределили оставшееся пространство поровну между собой. Аналогичный результат получим, если зададим размеры для столбцов следующим образом: **50% 25% 25%**.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Автоматические размеры

Значение **auto** ориентируется на содержание элементов сетки одного трека. Минимальный размер трека рассматривается как минимальный размер элемента сетки, то есть Grid использует свойства **min-width** или **min-height**. Максимальный размер трека устанавливается

из расчета на значение свойства **max-content**. Трек может растягиваться при использовании свойств **align-content** и **justify-content**.

Зададим размеры треков, используя значения **auto**:

```
grid-template-columns: auto auto auto;  
grid-template-rows: auto auto auto;
```

Результат:

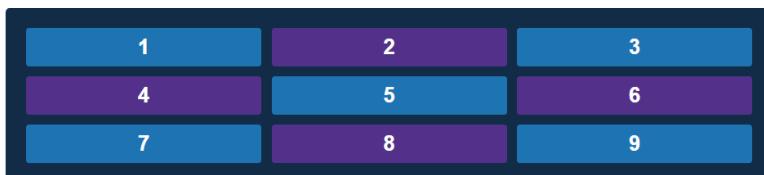


Рисунок 10

Из примера видно, что каждая строка и столбец заняли поровну часть пространства контейнера сетки.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Функция repeat

Функция **repeat()** позволяет создавать фрагменты списка треков, которые повторяются. Это позволяет записать в более компактной форме большое количество одинаковых по размерам столбцов или строк.

Для примера создадим несколько столбцов с одинаковыми размерами:

```
grid-template-columns: 1fr 1fr 1fr;
```

Такое написание можно заменить на:

```
grid-template-columns: repeat(3, 1fr);
```

Рассмотрим примеры использования функции `repeat()`:

```
grid-template-columns: repeat(2, 1fr 2fr);
```

Результат:

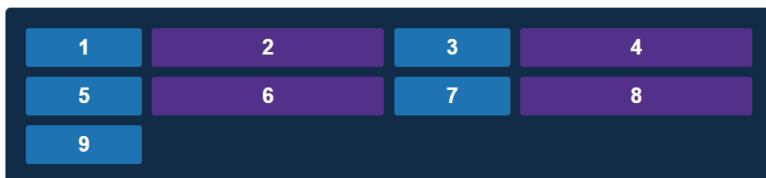


Рисунок 11

Сейчас одна строка сетки состоит из столбцов, которые занимают ширину в `1fr` и `2fr` соответственно и повторяются 2 раза в пределах одной строки. Последний элемент сетки занимает ширину в `1fr`. Если мы добавим еще один блок в сетку, он будет занимать `2fr`, следующий — `1fr` и т.д.

Еще один пример:

```
grid-template-columns: 50px repeat(2, 1fr 2fr);
```

Результат:



Рисунок 12

Теперь первый элемент в строке занимает фиксированную ширину в **100px**, а следующие 4 блока занимают ширину **1fr** и **2fr** соответственно и повторяются 2 раза в пределах строки.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Позиционирование элементов и Grid-линии

Свойства размещения и компоновки позволяют свободно размещать и изменять порядок расположения элементов сетки таким образом, что визуальное представление может значительно отличаться от порядка элементов в html-документе.

Позиционирование элементов сетки определяется расположением линий сетки и диапазоном сетки — количеством треков, которые занимает элемент. По умолчанию элемент сетки занимает одну ячейку трека на каждой оси.

Свойства позиционирования элементов сетки — `grid-row-start`, `grid-row-end`, `grid-column-start` и `grid-column-end` и их краткая запись `grid-row`, `grid-column` и `grid-area` позволяют определить размещение элемента сетки.

При размещении элементов мы скорее отдаём предпочтение линиям, а не трекам, поскольку позиционирование в основном зависит от индексов начальных и конечных линий сетки, а не от порядковых номеров треков.

Позиционирование элементов сетки с помощью номеров линий сетки

Можно определить расположение элементов, устанавливая номера начальной и конечной линий для элементов сетки.

Свойства:

- `grid-column-start` — задает начальную линию расположения элемента в строке.

- **grid-column-end** — задает конечную линию расположения элемента в строке.
- **grid-row-start** — задает начальную линию расположения элемента в столбце.
- **grid-row-end** — задает конечную линию расположения элемента в столбце.

Зададим некоторые CSS-свойства для элементов:

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

В примере сетка состоит из 4 строк и 3 столбцов. Мы размещаем первый элемент сетки, применяя к ней свойства `grid-column-start`, `grid-column-end`, `grid-row-start` и `grid-row-end`. Двигаясь слева направо, первый элемент начинается с вертикальной линии 1 и продолжается до вертикальной линии 4, которая находится справа в нашей сетке. Он начинается на горизонтальной линии 1 и заканчивается на горизонтальной линии 2, то есть элемент занимает 3 столбца и 1 строку сетки.

Второй элемент начинается с вертикальной линии 1 и заканчивается на вертикальной линии 2, то есть элемент занимает 1 столбец и 1 строку.

Третий элемент располагается по аналогии со вторым элементом и занимает столько же ячеек.

Четвертый элемент начинается с вертикальной линии 3 и заканчивается на вертикальной линии 4, начинается на горизонтальной линии 2 и заканчивается на горизонтальной линии 4, то есть занимает 1 столбец и 2 строки.

Если элемент будет занимать только одну ячейку сетки, можно опустить значение `-end`, как показано в следующем примере:

```
.item-5 {  
    grid-column-start: 1;  
    grid-row-start: 3;  
}  
  
.item-6 {  
    grid-column-start: 2;  
    grid-row-start: 3;  
}
```

Результат:

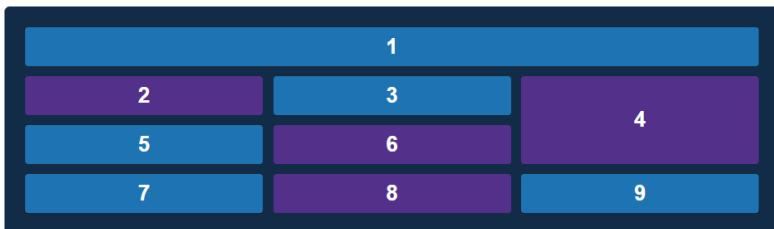


Рисунок 13

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Сокращенные формы записи для позиционирования

Мы можем достичь такого же результата, как в предыдущем примере, используя сокращенный синтаксис, который объявляет значение для начальной и конечной линий сетки сразу.

Свойства:

- **grid-column** — задает начальную и конечную линию расположения элемента в строке.
- **grid-row** — задает начальную и конечную линию расположения элемента в столбце.
- **grid-area** — задает начальную и конечную линию расположения элемента в строке и столбце.

В следующем примере элемент начинается с вертикальной линии 1 и заканчивается на вертикальной линии 4, начинается на горизонтальной линии 1 и заканчивает-

ся на горизонтальной линии 2, то есть занимает 3 столбца и 1 строку:

```
grid-column: 1 / 4;
grid-row: 1 / 2;
```

Если элемент занимает только одну ячейку, можно задать только значение для начальной линии сетки:

```
grid-row: 1;
```

- ▶ С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Можно определить сразу целую область, которую элемент будет занимать:

```
grid-area: 2 / 3 / 4 / 4;
```

Порядок значений следующий: `row-start / column-start / row-end / column-end`.

В примере выше элемент начинается с горизонтальной линии 2 и заканчивается на горизонтальной линии 4, начинается с вертикальной линии 3 и заканчивается на вертикальной линии 4, то есть занимает 2 строки и 1 столбец.

Опять-таки, если элемент занимает только одну ячейку, можно задать только значение для исходных линий сетки:

```
grid-area: 2 / 1;
```

- ▶ С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Объединение ячеек с помощью ключевого слова `span`

Ключевое слово `span` дает возможность объединять ячейки без указания начальной и конечной линии трека.

В следующем примере элемент начинается с вертикальной линии 1 и заканчивается на вертикальной линии 2, начинается на горизонтальной линии 1 и объединяет 3 ячейки по вертикали, т.е. заканчивается на горизонтальной линии 4. Элемент занимает 1 столбец и 3 строки.

```
grid-column: 1;
grid-row: 1 / span 3;
```

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в *CodePen* по [ссылке](#).

Именованные линии сетки

Хотя на линии сетки можно ссылаться по их числовому индексу, именованные линии облегчают понимание и использование свойств размещения сетки. Имена линиям можно задать явно, используя свойства `grid-template-rows` и `grid-template-columns` или неявно — путем создания именованных областей сетки с помощью свойства `grid-template-areas`.

Имя линии может быть любым. При задании в значении свойства, его следует взять в квадратные скобки. Ключевое слово `span` не может использоваться в качестве названия линии сетки.

Даже если мы задаем линиям имена, мы все равно можем обращаться к ним по индексу.

В следующем примере создается 3 строки и 3 столбца и для каждой линии сетки задается имя:

```
grid-template-columns: [col1-start] auto [col2-start]
                      auto [col3-start] auto [col3-end];
grid-template-rows:    [row1-start] auto [row2-start]
                      auto [row3-start] auto [row3-end];
```

На рисунке показано расположение именованных линий:

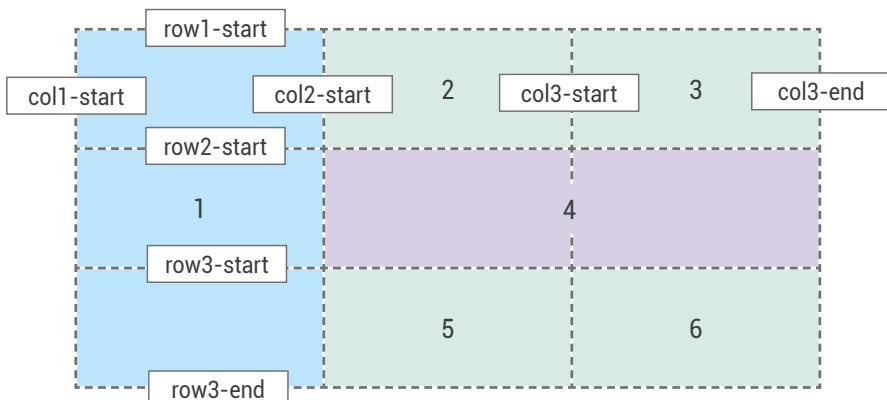


Рисунок 14

Теперь для позиционирования элементов в сетке можно использовать имена линий сетки, например:

```
grid-column: col1-start / col2-start;
grid-row: row1-start / row3-end;
```

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в [CodePen](#) по [ссылке](#).

Именованные строки и столбцы и ключевое слово `span`

Строчкам и столбцам можно также задать одно и то же имя, а потом задавать расположение элементов по этим именам и использовать ключевое слово `span` для объединения ячеек.

В следующем примере создаются 3 строки и 3 столбца, и для каждой строки и столбца задается одинаковое название:

```
grid-template-columns: [col] auto [col] auto [col] auto;
grid-template-rows:    [row] auto [row] auto [row] auto;
```

Теперь для позиционирования элементов в сетке можно использовать имена строк и столбцов, а для объединения ячеек — ключевое слово `span`:

```
grid-column: col 1;
grid-row: row / span 3;
```

Если элемент располагается в первой строке или столбце, то номер можно не писать.

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в [CodePen](#) по [ссылке](#).

Области сетки

Мы можем создать именованные участки сетки для размещения содержимого. Для этого мы сначала определяем элементы нашей разметки для сетки, используя свойство `grid-area`, например:

```
.item-1 {
    grid-area: area1;
}

.item-2 {
    grid-area: area2;
}

.item-3 {
    grid-area: area3;
}
```

А потом мы можем использовать свойство `grid-template-areas`, чтобы сказать, где именно на сетке должны располагаться эти элементы:

```
grid-template-areas:
    'area1 area1 area1'
    'area3 area4 area2'
    'area5 area6 area2';
```

Результат:

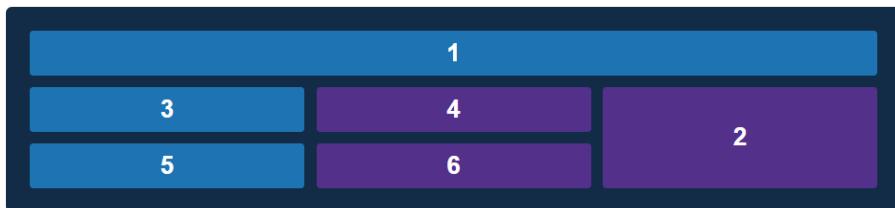


Рисунок 15

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Этот подход очень удобен при создании и компоновке небольших макетов. Рассмотрим еще один пример. Создадим несколько HTML-элементов:

```
<div class="grid">
    <div class="item-1">header</div>
    <div class="item-2">menu</div>
    <div class="item-3">main</div>
    <div class="item-4">right</div>
    <div class="item-5">footer</div>
</div>
```

Добавим стили:

```
.grid {
    display: grid;
    grid-template-areas:
        'header header header header header header'
        'menu main main main right right'
        'menu footer footer footer footer footer';
    grid-gap: 10px;
    padding: 20px;
    border-radius: 5px;
    background-color: #0d233d;
    font-family: Arial, Helvetica, sans-serif;
}

.grid div {
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 0.3em;
    border-radius: 3px;
    color: #ffffff;
    font-size: 130%;
    font-weight: bold;
}
```

```
.item-1 {  
    grid-area: header;  
    background-color: #aa2f26;  
}  
  
.item-2 {  
    grid-area: menu;  
    background-color: #b2a429;  
}  
  
.item-3 {  
    grid-area: main;  
    background-color: #1769aa;  
}  
  
.item-4 {  
    grid-area: right;  
    background-color: #01796f;  
}  
  
.item-5 {  
    grid-area: footer;  
    background-color: #482880;  
}
```

Результат:



Рисунок 16

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в [CodePen](#) по [ссылке](#).

Выравнивание в Grid

Grid содержит функции выравнивания для того, чтобы мы могли контролировать, как именно выравниваются элементы, размещенные в области сетки, и как выравнивается вся сетка.

Выравнивание всей сетки

Иногда размер сетки может быть меньше, чем размер самого контейнера сетки. Такое может случиться, если заданный размер строк и столбцов является фиксированным (в `px`). В этом случае мы можем установить выравнивание для сетки в контейнере с помощью свойств `justify-content` и `align-content`.

Свойство `justify-content` выравнивает всю сетку относительно вертикальной оси.

Значения:

- `start` — выравнивает сетку по левому краю контейнера.
- `end` — выравнивает сетку по правому краю контейнера.
- `center` — выравнивает сетку по центру.
- `stretch` — растягивает сетку на всю ширину контейнера.
- `space-around` — размещает единицу пустого пространства между элементами и половину единицы по краям.
- `space-between` — размещает единицу пустого пространства между элементами, без отступов по краям.

- **space-evenly** — размещает единицу пустого пространства между элементами, включая края сетки.
Пример использования:

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

Результат:



Рисунок 17

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Свойство **align-content** выравнивает сетку относительно горизонтальной оси.

Значения:

- **start** — выравнивает сетку по верхнему краю контейнера.
- **end** — выравнивает сетку по нижнему краю контейнера.
- **center** — выравнивает сетку по центру.
- **stretch** — растягивает сетку на всю высоту контейнера.
- **space-around** — размещает единицу пустого пространства между элементами и половину единицы по краям.
- **space-between** — размещает единицу пустого пространства между элементами, без отступов по краям.

- **space-evenly** — размещает единицу пустого пространства между элементами, включая края сетки.

Пример использования:

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

Результат:

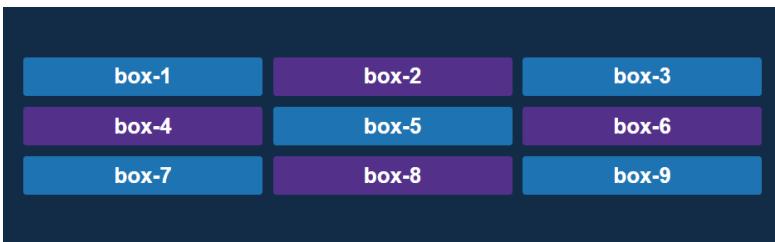


Рисунок 18

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Выравнивание всех элементов сетки

Иногда возникает необходимость выровнять контент внутри ячеек. Мы можем установить выравнивание для контента с помощью свойств **justify-items** и **align-items**. Данные значения применяются ко всем ячейкам сетки.

Свойство **justify-items** выравнивает содержимое внутри ячейки относительно вертикальной оси.

Значения:

- **start** — выравнивает содержимое по левому краю ячейки.

- **end** — выравнивает содержимое по правому краю ячейки.
- **center** — выравнивает содержимое по центру.
- **stretch** — растягивает содержимое на всю ячейку (значение по умолчанию).

Пример использования:

```
justify-items: center;
```

Результат:



Рисунок 19

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Следующий пример более наглядно демонстрирует работу этого свойства:

- [Ссылка](#) на пример в CodePen.

Свойство **align-items** выравнивает содержимое внутри ячейки относительно горизонтальной оси.

Значения:

- **start** — выравнивает содержимое по верхнему краю ячейки.
- **end** — выравнивает содержимое по нижнему краю ячейки.

- **center** — выравнивает содержимое по центру.
- **stretch** — растягивает содержимое на всю ячейку (значение по умолчанию).

Пример использования:

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

Результат:

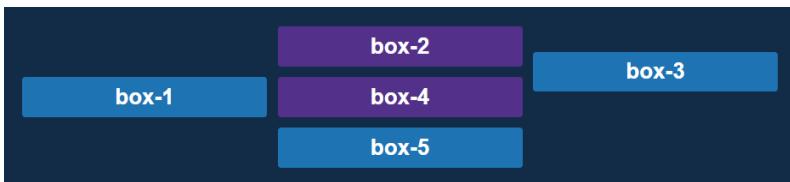


Рисунок 20

- ▶ С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Следующий пример более наглядно демонстрирует работу этого свойства:

- ▶ [Ссылка](#) на пример в CodePen.

Выравнивание отдельных элементов сетки

Установить индивидуальное поведение для ячейки можно с помощью свойств **justify-self** и **align-self**.

Свойство **justify-self** выравнивает текущий элемент относительно вертикальной оси.

Значения:

- **start** — выравнивает элемент по левому краю зоны.
- **end** — выравнивает элемент по правому краю зоны.

- **center** — выравнивает элемент по центру зоны.
- **stretch** — растягивает элемент по горизонтали на все свободное место (значение по умолчанию).

Пример использования:

```
justify-self: center;
```

Результат:



Рисунок 21

- ▶ С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Следующий пример более наглядно демонстрирует работу этого свойства:

- ▶ [Ссылка](#) на пример в CodePen.

Свойство **align-self** выравнивает текущий элемент относительно горизонтальной оси.

Значения:

- **start** — выравнивает элемент по верхнему краю зоны.
- **end** — выравнивает элемент по нижнему краю зоны.
- **center** — выравнивает элемент по центру зоны.

- **stretch** — растягивает элемент по вертикали на все свободное место (значение по умолчанию).

Пример использования:

```
align-self: center;
```

Результат:

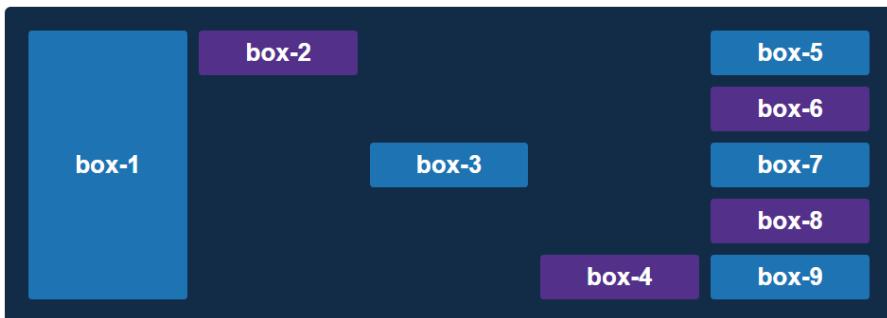


Рисунок 22

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в [CodePen](#) по [ссылке](#).

Следующий пример более наглядно демонстрирует работу этого свойства:

- [Ссылка](#) на пример в [CodePen](#).

Grid или Flexbox: что выбрать?

За последние несколько лет Flexbox стал очень популярным среди Front-end разработчиков. И это неудивительно, поскольку с помощью Flexbox стало гораздо проще создавать динамические шаблоны и выравнивать содержимое внутри контейнеров.

Однако со временем появился Grid, а у него очень много возможностей аналогичных Flexbox. В одних случаях он даже лучше, чем Flexbox, а в других — совсем нет.

Основное различие между двумя подходами в том, что **Flexbox** — это система одномерной компоновки, а **Grid** — двумерной.

То есть если мы создаем объекты в одном направлении, например, кнопку в **header**, то лучше применять Flexbox.

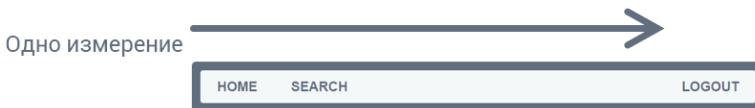


Рисунок 23

В этом случае Flexbox предоставит нам больше гибкости, чем использование Grid. Также с ним будет проще работать в дальнейшем, а самое главное — мы избежим лишней писанины.

Но если мы собираемся создать целый макет в двух измерениях, то есть использовать как столбцы, так и строки, то больше подойдет Grid.

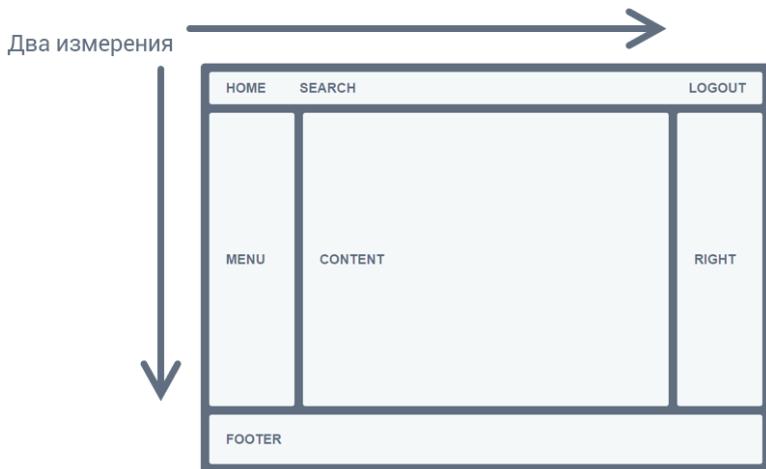


Рисунок 24

В этом случае с Grid мы получим больше функциональности и возможностей для компоновки, и наша разметка будет проще, а код станет понятнее для дальнейшей работы.

Мы, конечно, можем сочетать оба подхода. В приведенном выше рисунке было бы идеально использовать Grid для всего макета, а потом Flexbox для выравнивания содержимого внутри `header`.

На данный момент большинство браузеров поддерживает Flexbox, и все же есть несколько браузеров, о которых нам нужно помнить для обеспечения кроссбраузерности. Это Internet Explorer 10, который внедрил `display: flexbox` версию спецификации с префиксом `-ms-`, UC Browser, который все еще поддерживает спецификацию версии 2009 года с `display: box` с префиксом `-webkit-` и Internet Explorer 11, который, хоть и поддерживает эту технологию, но не без багов. Также Flexbox не поддержи-

вают старые версии некоторых браузеров. Сайт <https://caniuse.com>, куда постоянно добавляется свежая и актуальная информация по поддержке браузерами CSS-свойств, показывает следующий результат:

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android
6-9		2-21 22-27	4-20 21-28	3.1-6 6.1-8	10-11.5 15-16	12.1 7-8.4	3.2-6.1 9-13.3		2.1-4.3 4.4-4.4.4	12 12.1	
10		12-80 28-75	28-75 29-80	9-13 17-67							
11		81	76	81	13.1	68	13.4	all	81	46	81
		77-78	83-85	TP							68

Рисунок 25

Несмотря на то, что Grid появился сравнительно недавно, его уже поддерживает большинство современных браузеров. Однако Internet Explorer 10 и Internet Explorer 11 реализуют старую версию спецификации, которая не поддерживает сокращенную форму записи `grid`. Grid не поддерживается старыми версиями некоторых браузеров и мобильной версией браузера Opera.

Отображение поддержки Grid браузерами на вышеупомянутом сайте:

IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android
6-9		2-39 40-51	4-28 29-56		10-27		3.2-10.2				
10		12-15 52-53	57	3.1-10 10.1-13	28-43 44-67		10.3-13.3	2.1-4.4.4	12-12.1		
11		16-80 54-75	58-80	81	13.1	68	13.4	all	81	46	81
		77-78	83-85	TP							68

Рисунок 26

Еще одно кардинальное отличие между двумя технологиями заключается в том, что Flexbox берет за основу

контент, в то время как в основе Grid лежит сам макет и его общая разметка. Звучит все это достаточно абстрактно, поэтому давайте рассмотрим конкретный пример, который позволит нам легче понять эти различия.

В следующем примере мы создадим header для макета с помощью двух подходов — Grid и Flexbox, и попробуем понять, чем же отличаются эти технологии в плане их использования.

Создадим некоторую разметку:

```
<header>
  <div>Home</div>
  <div>Search</div>
  <div>Logout</div>
</header>
```

Зададим стили:

```
header {
  padding: 10px;
  background-color: #0d233d;
  color: #fff;
  font-family: Arial, Helvetica, sans-serif;
  text-transform: uppercase;
}

header div {
  margin: 0 20px;
  padding: 10px 0;
}
```

До использования Flexbox блоки будут располагаться один над другим:

```
HOME  
SEARCH  
LOGOUT
```

Рисунок 27

Flexbox-подход

Когда мы укажем `display: flex`, все элементы аккуратно встанут в строку:

```
header {  
    display: flex;  
}
```

Мы получим следующий результат:

```
HOME  SEARCH  LOGOUT
```

Рисунок 28

Для того, чтобы разместить кнопку Logout справа, мы просто укажем нужный элемент и зададим ему внешний отступ:

```
header > div:nth-child(3) {  
    margin-left: auto;  
}
```

Это даст нам такой результат:

```
HOME  SEARCH  LOGOUT
```

Рисунок 29

При использовании данного подхода, мы дали элементам самим решить, как именно им расположиться.

В этом заключается основное отличие между Flexbox и Grid, и сейчас это станет еще понятнее после того, как мы переделаем **header** с использованием Grid.

Grid-подход

Хоть Grid и не задумывался для создания одномерных header'ов, это все же хороший пример для понимания разницы между двумя подходами к компоновке.

Мы можем создать наш **header** несколькими разными способами с помощью Grid. В этом примере мы увидим достаточно понятный способ, где у Grid есть десять колонок, каждая из которых является частью формирования ширины элемента:

```
header {  
    display: grid;  
    grid-template-columns: repeat(10, 1fr);  
}
```

Результат будет выглядеть аналогичным использованию Flexbox:



HOME SEARCH LOGOUT

Рисунок 30

Давайте подробнее рассмотрим, в чем же разница. Зайдем в **Developer Tools** в браузере, чтобы увидеть границы:

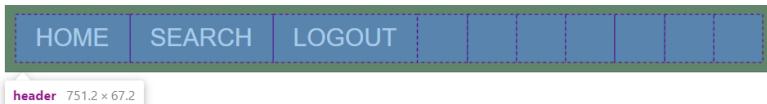


Рисунок 31

Ключевое отличие тут в том, что мы должны сначала определить столбцы макета, начиная с определения ширины столбцов и потом добавляя содержание в доступные ячейки сетки.

Этот подход заставил нас четко определить, на сколько столбцов мы хотим разделить **header**.

Пока мы не изменим сетку, получается, что мы застряли с этими десятью столбцами. Это ограничение, с которым мы бы не столкнулись, работая только с Flexbox.

Для того, чтобы поместить кнопку **Logout** справа, мы расположим этот элемент в десятом столбце:

```
header > div:nth-child(3) {
    grid-column: 10;
}
```

Вот как это будет выглядеть при детальном отображении расположения столбцов:

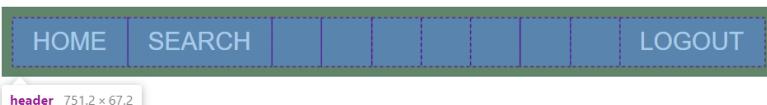


Рисунок 32

Мы не могли просто задать этому элементу **margin-left: auto;**, потому что кнопка **Logout** уже была помещена в конкретную ячейку макета, а именно в третий столбец.

Объединяем два подхода

А теперь давайте посмотрим, как применять оба подхода вместе, объединяя **header** со всем нашим макетом. Начнем с построения самого макета:

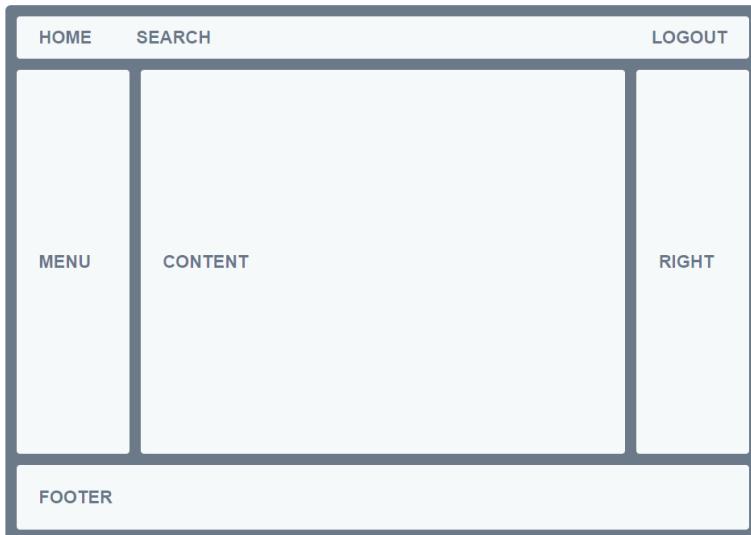


Рисунок 33

Пропишем разметку:

```
<div class="container">
  <header>
    <div>Home</div>
    <div>Search</div>
    <div>Logout</div>
  </header>
  <nav>Menu</nav>
  <main>Content</main>
  <aside>Right</aside>
  <footer>Footer</footer>
</div>
```

Зададим некоторые стили:

```
.container {  
    display: grid;  
    grid-template-columns: repeat(12, 1fr);  
    grid-template-rows: auto 350px auto;  
}
```

Расставим элементы по сетке:

```
header {  
    grid-column: span 12;  
}  
  
nav {  
    grid-column: span 2;  
}  
  
main {  
    grid-column: span 8;  
}  
  
aside {  
    grid-column: span 2;  
}  
  
footer {  
    grid-column: span 12;  
}
```

Теперь просто добавим **header**. Применим к нему Flexbox, хотя он уже является элементом нашего Grid:

```
header {  
    display: flex;  
}
```

Далее мы можем установить кнопку **Logout** справа:

```
header > div:nth-child(3) {  
    margin-left: auto;  
}
```

Мы создали макет, который использует как Grid, так и Flexbox:

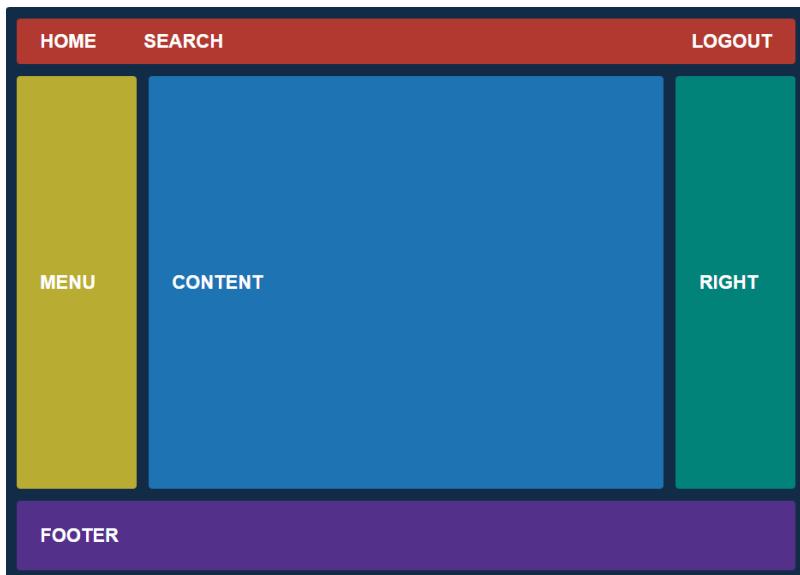


Рисунок 34

- С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Теперь у вас есть четкое понимание основ и конкретных различий между Flexbox и Grid, и самое важное — это понимание того, как использовать их вместе.

Практические примеры использования

Чтобы закрепить пройденный материал на практике, создадим небольшую адаптивную галерею изображений.

Наша галерея будет состоять из блоков с изображениями различной ширины и высоты. Также мы добавим галерее адаптивное поведение для красивого отображения на различных устройствах. Результат будет примерно такой:

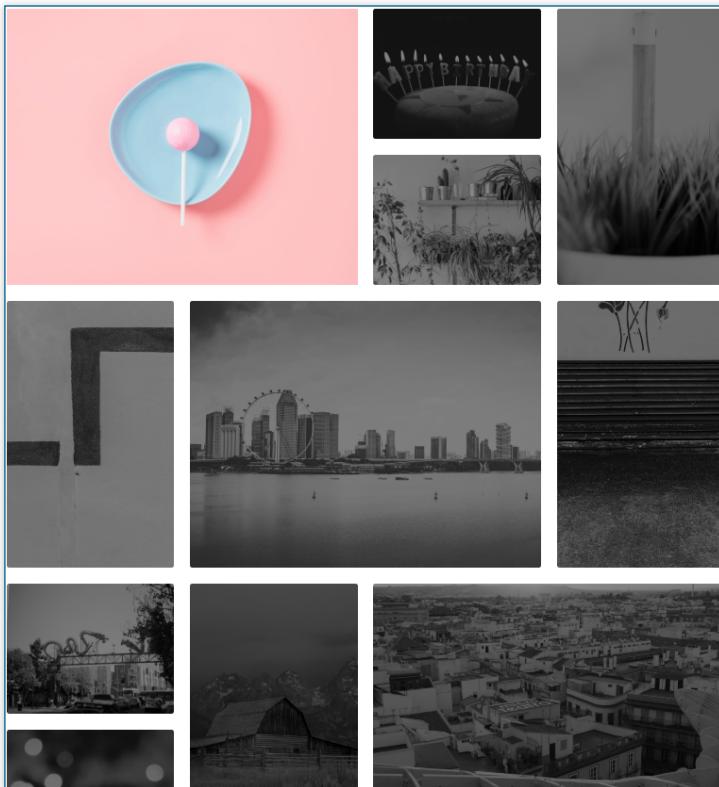


Рисунок 35

Определим основной контейнер, который будет содержать изображения и будет выступать в роли сетки:

```
<div class="gallery"></div>
```

Зададим стили для этого элемента:

```
.gallery {  
    width: 80%;  
    margin: 40px auto;  
}
```

Добавим блок с одним изображением в элемент с классом **gallery**:

```
<div class="image">  
      
</div>
```

Мы будем использовать случайные изображения, поэтому они каждый раз будут меняться в галерее. Изображение можно взять с сайта <https://source.unsplash.com>.

Добавим минимум двадцать таких изображений. Для блоков с картинками задаем следующие стили:

```
.gallery .image {  
    position: relative;  
    height: 100%;  
    width: 100%;  
    overflow: hidden;  
}
```

Пока что все блоки с картинками расположены друг над другом. Применим свойство `grid` для элемента с классом `gallery` и определим количество столбцов в сетке:

```
display: grid;  
grid-template-columns: auto auto auto auto;
```

По умолчанию наша галерея будет содержать 4 столбца. Зададим размеры линий сетки:

```
grid-gap: 2vh;
```

Вот так сейчас выглядит одна строка галереи:



Рисунок 36

Зададим также некоторые стили для изображений:

```
.gallery .image img {  
    height: 100%;  
    width: 100%;  
    object-fit: cover;  
}
```

Теперь в одном блоке показывается только определенная часть изображения, а `object-fit: cover` центриру-

ет само изображение в блоке. Сейчас галерея выглядит примерно так:

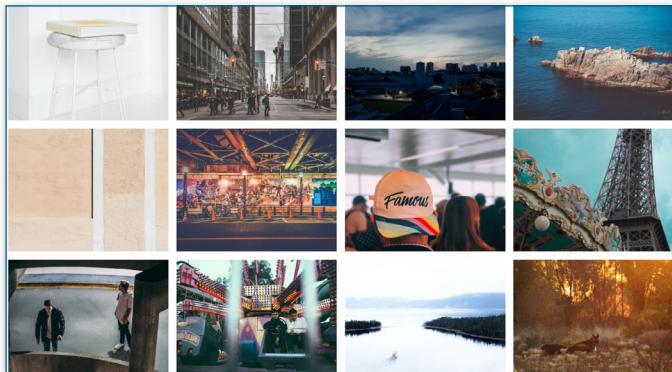


Рисунок 37

Каждый блок с изображением занимает 1 столбец и 1 строку сетки. Изменим количество ячеек для первого элемента галереи:

```
.gallery .image:first-child {  
    grid-column-start: span 2;  
    grid-row-start: span 2;  
}
```

Теперь этот элемент занимает 2 столбца и 2 строки:



Рисунок 38

Изменим расположение и размеры других элементов, например так:

```
.gallery .image:nth-child(2n+1) {  
    grid-row-start: span 2;  
}  
  
.gallery .image:nth-child(4n+6) {  
    grid-column-start: span 2;  
    grid-row-start: span 2;  
}
```

Примерно вот так сейчас выглядит наша галерея:

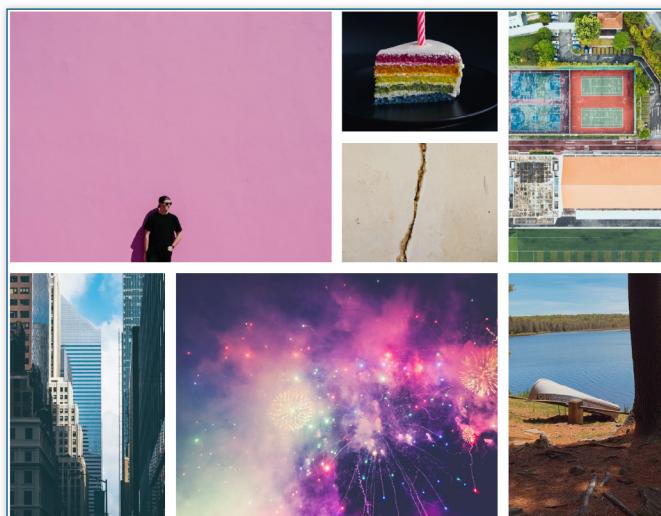


Рисунок 39

Поскольку некоторые элементы могут не помещаться в отведенную для них область, применим для сетки свойство **grid-auto-flow**, чтобы исправить отображение:

```
grid-auto-flow: dense;
```

Добавим изображениям немного интерактивности:

```
.gallery .image img {  
    ...  
    border-radius: 2px;  
    filter: brightness(0.5) grayscale(1);  
    transition: 0.3s ease-in-out;  
}  
  
.gallery .image img:hover {  
    filter: brightness(1) grayscale(0);  
    transform: scale(1.1);  
}
```

Примерный вид галереи:

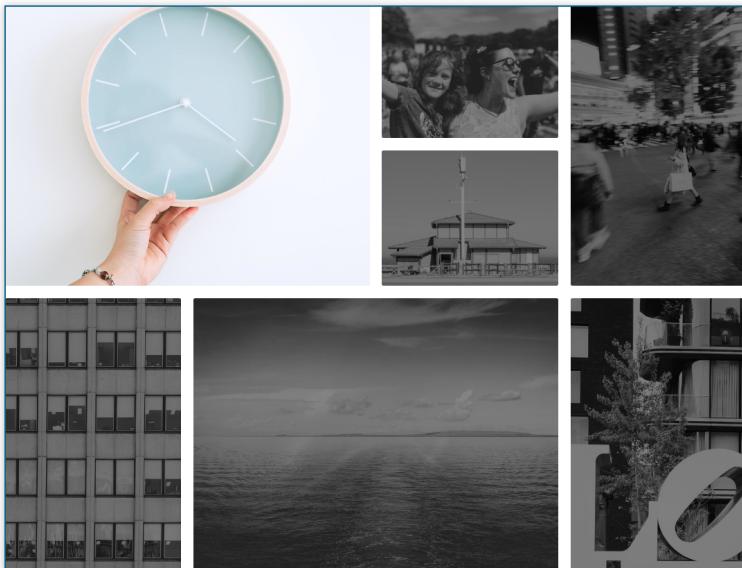


Рисунок 40

Зададим адаптивное поведение галереи для отображения на разных экранах устройств:

```
@media only screen and (min-width : 480px)
    and (max-width : 768px) {
    .gallery {
        display: grid;
        grid-template-columns: auto auto auto;
    }
}

@media only screen and (min-width : 320px)
    and (max-width : 480px) {
    .gallery {
        display: grid;
        grid-template-columns: auto auto;
    }
}

@media only screen and (max-width: 320px) {
    .gallery {
        display: block;
        max-width: 240px;
    }

    .gallery .image {
        display: block;
        margin: 2% 0;
    }

    .gallery .image img{
        filter: brightness(1) grayscale(0);
    }
}
```

Отображение галереи на разных экранах показано на рисунке 41.

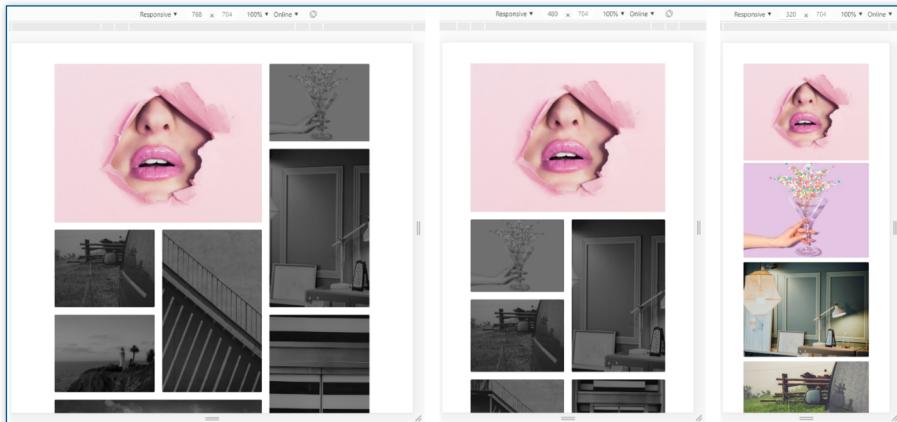


Рисунок 41

- ▶ С полным кодом примера можно ознакомиться в прикрепленных к уроку файлах или просмотреть в CodePen по [ссылке](#).

Полезные ссылки по теме:

1. <https://drafts.csswg.org/css-grid>;
2. https://www.w3schools.com/css/css_grid.asp;
3. <https://gridbyexample.com>;
4. <https://css-tricks.com/snippets/css/complete-guide-grid>;
5. <https://learn-cssgrid.com>.

Домашнее задание

- Предлагается закрепить все знания, приобретенные на уроке, с помощью интересной мини-игры. Это очень простая и милая игра, в которой мы поливаем морковь и опрыскиваем сорняки, попутно осваивая Grid Layout. Полезные советы от разработчиков помогают осилить каждый уровень игры. Ссылка на сайт с игрой: <https://cssgridgarden.com>.



Рисунок 42

- Вторым заданием будет сверстать свой адаптивный сайт-портфолио, используя сочетание подходов Grid и Flexbox.

Сайт может состоять, например, из главной страницы, страницы с работами студента и страницы, в которой рассказывается об авторе сайта. Сайт также может быть одностраниценным — **Landing Page**.

Макет сайта и отображение работ студента нужно организовать с использованием Grid.

Меню, расположение иконок ссылок на социальные сети и другие элементы можно задать при помощи Flexbox.

Вдохновение и идею для своего сайта можно почерпнуть, просмотрев готовые сайты портфолио по ссылке: <https://colorlib.com/wp/cat/portfolio>.