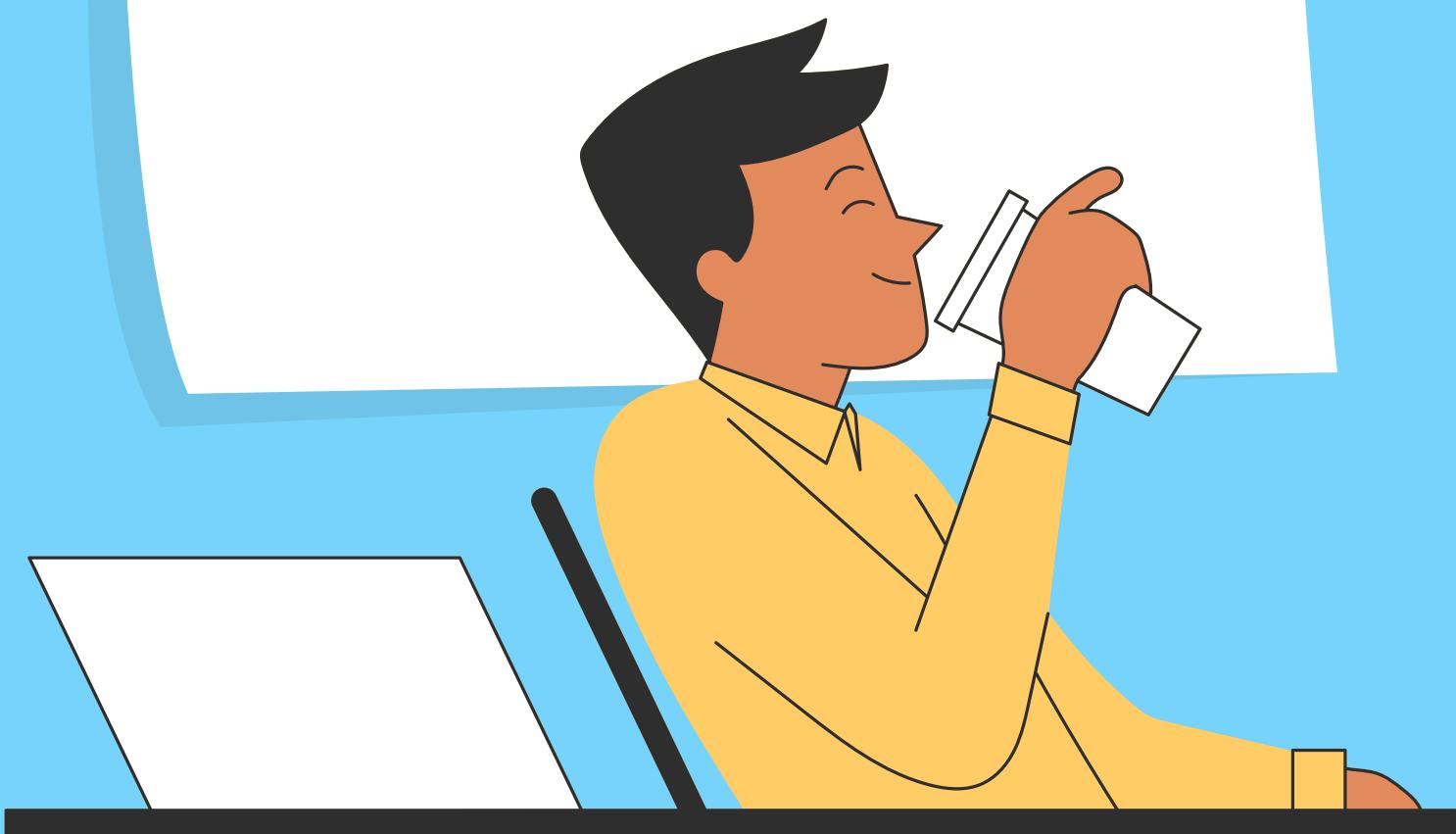




CSS

Cascading Style
Sheets
(Каскадные
таблицы стилей.)

ПЛАН



1

Кросбраузерность, вендорные префиксы.

2

Переходы, анимация.

3

Grid.



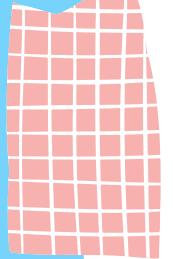
КРОССБРАУЗЕРНОСТЬ

Кроссбраузерность сайтов означает, что веб-страница отображается и функционирует одинаково хорошо в различных веб-браузерах, таких как Google Chrome, Mozilla Firefox, Safari, Microsoft Edge и других. В связи с тем, что разные браузеры могут иметь небольшие различия в реализации стандартов и поддержке CSS и JavaScript, обеспечение кроссбраузерности веб-сайта является важным аспектом разработки.



1. Использование кроссбраузерных CSS и JavaScript: При написании CSS и JavaScript кода следует учитывать поддержку различных браузеров. Избегайте использования устаревших или экспериментальных свойств и функций, которые могут не работать во всех браузерах. Используйте кроссбраузерные альтернативы для поддержки более старых версий браузеров.
2. Использование вендорных префиксов: вендорные префиксы могут быть использованы для обеспечения поддержки определенных функций в разных браузерах. Однако следует быть осторожным и использовать их только там, где это действительно необходимо, и проверять совместимость с последними версиями браузеров.
3. Гибкий дизайн: Стремитесь создавать гибкий и адаптивный дизайн, который может приспосабливаться к различным размерам экранов и разрешениям.

ВЕНДОРНЫЕ ПРЕФИКСЫ



Вендорные префиксы CSS - это специфичные префиксы, которые добавляются к CSS свойствам для обеспечения поддержки браузерами экспериментальных или нестандартных функций CSS, которые могут быть ещё в разработке или не включены в официальные стандарты CSS.

Каждый браузер — это отдельный вендор (от англ. vendor — продавец)

услуг просмотра сайтов, интернета. Отсюда и слово «вендорный».

Буквально это означает, что существуют некие отдельные префиксы —

они же приставки — которые работают в конкретном браузере —

вендоре.

- Основные браузеры используют следующие префиксы:
- -webkit- — Safari, Chrome, Opera 15+ и другие браузеры на основе движка WebKit или Blink.
- -moz- — Firefox и браузеры на движке Gecko.
- -o- — Opera 12 и раньше, на движке Presto.
- -ms- — Internet Explorer и старый Microsoft Edge 12–18.

```
.box {  
    -webkit-border-radius: 5px; /* Вендорный префикс для свойства border-radius */  
    -moz-border-radius: 5px; /* Вендорный префикс для свойства border-radius */  
    -ms-border-radius: 5px; /* Вендорный префикс для свойства border-radius */  
    -o-border-radius: 5px; /* Вендорный префикс для свойства border-radius */  
    border-radius: 5px; /* Стандартное свойство border-radius */  
}
```

В этом примере мы используем вендорные префиксы для свойства `border-radius`, чтобы обеспечить его поддержку в различных браузерах. По мере того, как браузеры обновляются и добавляют поддержку стандартных свойств, вендорные префиксы могут становиться необязательными.

Самый простой способ проверить поддержку свойства — найти его на сайте [Can I use](#).

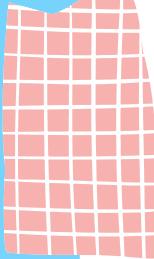
Очень важно указывать сущности (свойства, селекторы и так далее) с вендорными префиксами выше, чем без префиксов:

Это нужно для того, чтобы браузер использовал самую последнюю стабильную реализацию. Если браузер уже поддерживает фичу без префиксов, то применится последнее из перечисленных. А если нет, то сработает подходящая запись из кода выше.

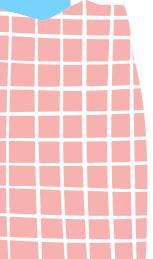
Вендорные префиксы помогают разработчикам использовать новые возможности CSS до того, как они полностью включаются в стандарты.

Важно отметить, что с появлением новых версий браузеров и стандартов CSS, многие функции уже не требуют вендорных префиксов, так как они полностью поддерживаются во всех современных браузерах.

TRANSITION



Переходы (transitions) в CSS позволяют создавать плавные анимации и эффекты перехода между различными стилями элементов. С помощью переходов вы можете контролировать изменение свойств элементов во времени, создавая более динамичный и интерактивный пользовательский опыт.

- 
- Основные свойства, используемые для определения переходов, включают:

1. **transition-property**: Определяет, к каким свойствам должен применяться переход. Вы можете указать одно или несколько свойств через запятую. Например, `transition-property: width, background-color;` означает, что будут применяться переходы для изменений ширины и цвета фона элемента.

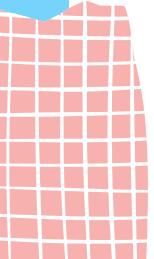
2. **transition-duration**: Определяет длительность перехода в секундах или миллисекундах. Например, `transition-duration: 0.5s;` задает время перехода в полсекунды.

3. **transition-timing-function**: Определяет, как изменения свойств элемента происходят во времени.

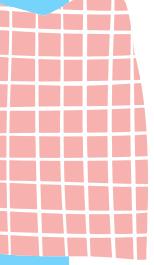
- **ease** - задает эффект перехода с медленным запуском, затем быстрее, затем медленным завершением (по умолчанию)
- **linear** - определяет эффект перехода с одинаковой скоростью от начала до конца
- **ease-in** - задает эффект перехода с медленным запуском
- **ease-out** - определяет эффект перехода с медленным концом
- **ease-in-out** - задает эффект перехода с медленным началом и концом
- **cubic-bezier(n,n,n,n)** - позволяет определить собственные значения в функции cubic-bezier



4. **transition-delay**: Определяет задержку перед началом перехода. Это позволяет установить время ожидания, прежде чем переход начнется. Например, **transition-delay: 1s;** означает, что переход начнется через 1 секунду после того, как свойства будут изменены.



В CSS есть короткая запись для свойства **transition**, которая позволяет объединить все свойства перехода в одной строке. Это делает код более компактным и легким для чтения. Короткая запись **transition** состоит из четырех значений: **transition: property duration timing-function delay**.



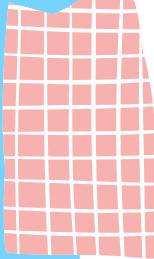
В этом примере используется короткая запись transition для элемента с классом .box. Значение all указывает, что переход будет применяться ко всем свойствам. Значение 1s указывает длительность перехода, ease - функцию изменения времени (с медленным запуском, затем быстрее, затем медленным завершением), а 0.5s - задержку перед началом перехода.

```
.box {  
    width: 200px;  
    height: 200px;  
    border: 2px solid black;  
    background: lightgreen;  
    transition: all 1s ease .5s|
```

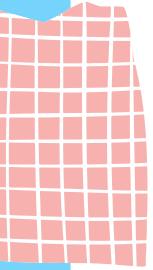


Короткая запись transition может быть удобной, особенно когда нужно применить одинаковые параметры перехода ко множеству свойств. Однако, если вам нужно различать параметры перехода для разных свойств, вы можете продолжать использовать полную запись с transition-property, transition-duration, transition-timing-function и transition-delay.

ANIMATION

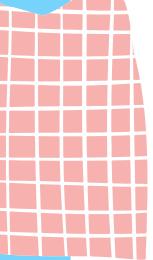


Анимации в CSS позволяют создавать движение и переходы между различными стилями элементов на веб-странице. Они позволяют добавить интерактивность и визуальные эффекты, делая сайт более привлекательным для пользователя.



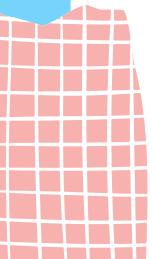
Анимации в CSS основаны на ключевых кадрах (keyframes), которые определяют состояния элемента на различных моментах времени. В каждом ключевом кадре вы указываете, какие свойства элемента должны измениться, чтобы создать желаемый эффект. Затем вы определяете, как эти ключевые кадры должны быть анимированы с помощью свойства `animation`.

```
@keyframes firstAnimation {  
    from {  
        background: red;  
    }  
  
    50% {  
        background: yellow;  
        transform: scale(1.2);  
    }  
  
    to {  
        background: green;  
        transform: scale(1.5);  
    }  
}
```



Основные свойства, используемые для создания анимаций в CSS:

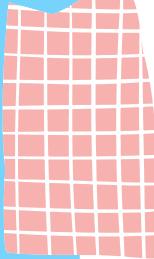
1. **@keyframes**: Определяет набор ключевых кадров для анимации. Внутри @keyframes вы указываете имя анимации и определяете состояния элемента на различных временных моментах, используя проценты от 0% до 100%. Каждый состояния содержит изменения свойств элемента.
2. **animation-name**: Указывает имя набора ключевых кадров (определенного с помощью @keyframes), который будет использоваться для анимации.
3. **animation-duration**: Определяет длительность анимации в секундах или миллисекундах.
4. **animation-timing-function**: Определяет, как изменения свойств элемента происходят во времени. Это может быть линейное изменение (linear), плавное изменение (ease), изменение с ускорением (ease-in), изменение с замедлением (ease-out) и другие. Как и в переходах, вы также можете использовать кастомные функции, определенные с помощью кривых Безье.
5. **animation-delay**: Определяет задержку перед началом анимации. Это позволяет установить время ожидания, прежде чем анимация начнется.
6. **animation-iteration-count**: Определяет количество повторений анимации. Вы можете использовать конкретное число или ключевое слово infinite для бесконечного повторения.



Свойство `animation-fill-mode` в CSS определяет, как стили элемента применяются до и после выполнения анимации.

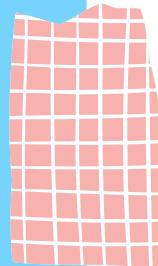
1. **none**: Это значение по умолчанию. Стили элемента не будут изменяться до или после выполнения анимации.
2. **forwards**: Стили элемента будут сохранены после выполнения анимации и будут соответствовать последнему состоянию анимации. То есть элемент останется в последнем состоянии анимации, как если бы анимация продолжалась. Это полезно, когда вы хотите, чтобы элемент оставался в итоговом состоянии анимации после ее выполнения.
3. **backwards**: Стили элемента будут устанавливаться в начальное состояние анимации до ее выполнения. То есть элемент будет начинать анимацию со своего начального состояния, даже если анимация не началась. Это полезно, когда вы хотите, чтобы элемент находился в начальном состоянии анимации до ее начала.
4. **both**: Стили элемента будут сохранены как до, так и после выполнения анимации. Сначала они будут устанавливаться в начальное состояние анимации, затем после выполнения анимации они будут сохранены в последнем состоянии. Это сочетание `forwards` и `backwards`.

GRID



CSS Grid Layout или просто гриды — это удобная технология для раскладки элементов на веб-страницах. В отличие от флексбоксов, одновременно работающих только с одним измерением, гриды дают возможность работать одновременно с двумя: горизонталью и вертикалью.

Грид-контейнер: родительский элемент, к которому применяется свойство `display: grid`.



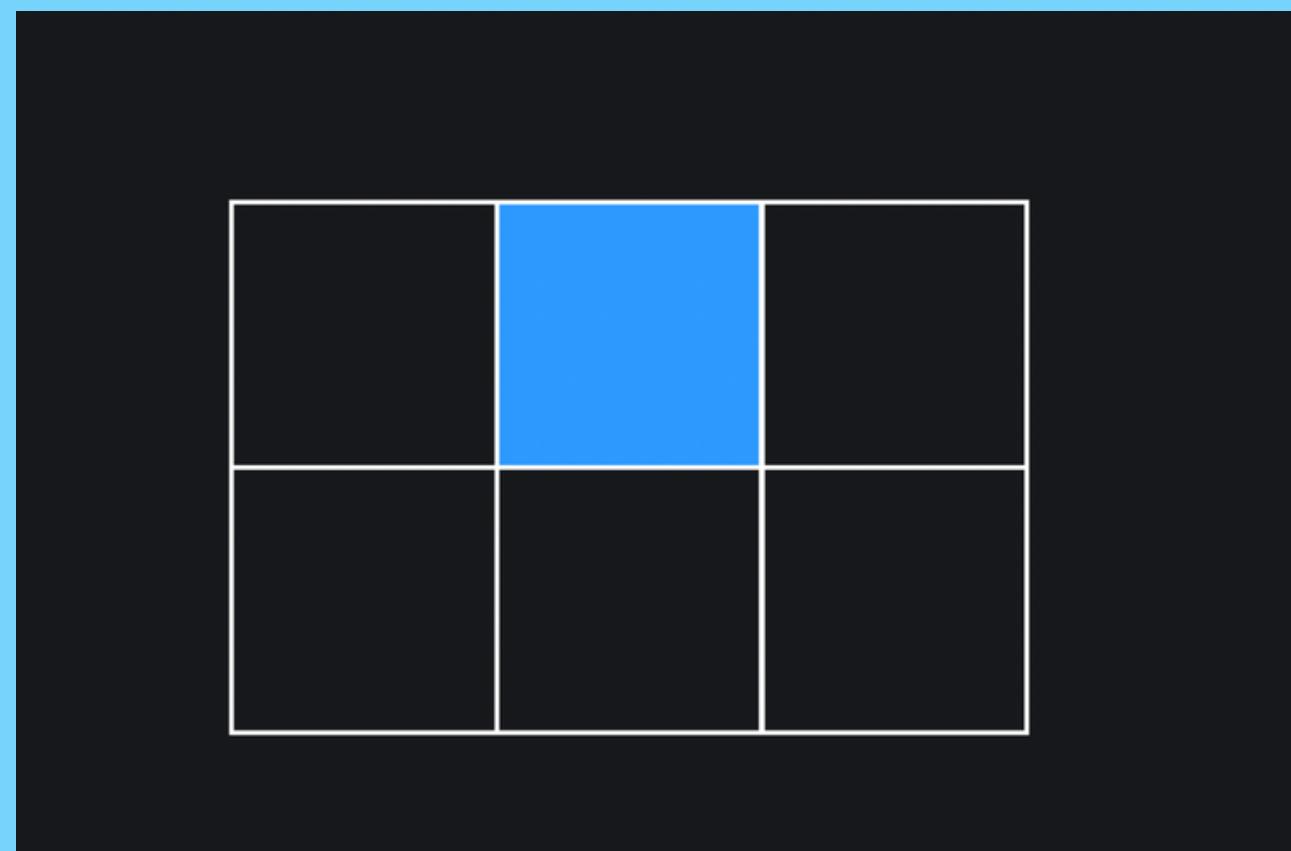
Грид-элемент: дочерний элемент, прямой потомок грид-контейнера.

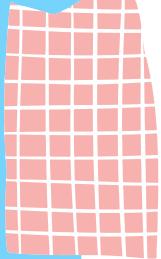
Подчиняется правилам раскладки гридов.

Грид-линия: разделительная линия, формирующая структуру грида. Может быть как вертикальной (грид-линия колонки), так и горизонтальной (грид-линия ряда). Располагается по обе стороны от колонки или ряда.

Используется для привязки грид-элементов.

Грид-ячейка: пространство между соседними грид-линиями. Единица грид-сетки.





Грид-контейнер: родительский элемент, к которому применяется свойство `display: grid`.

Грид-элемент: дочерний элемент, прямой потомок грид-контейнера.

Подчиняется правилам раскладки гридов.

Грид-линия: разделительная линия, формирующая структуру грида. Может быть как вертикальной (грид-линия колонки), так и горизонтальной (грид-линия ряда). Располагается по обе стороны от колонки или ряда.

Используется для привязки грид-элементов.

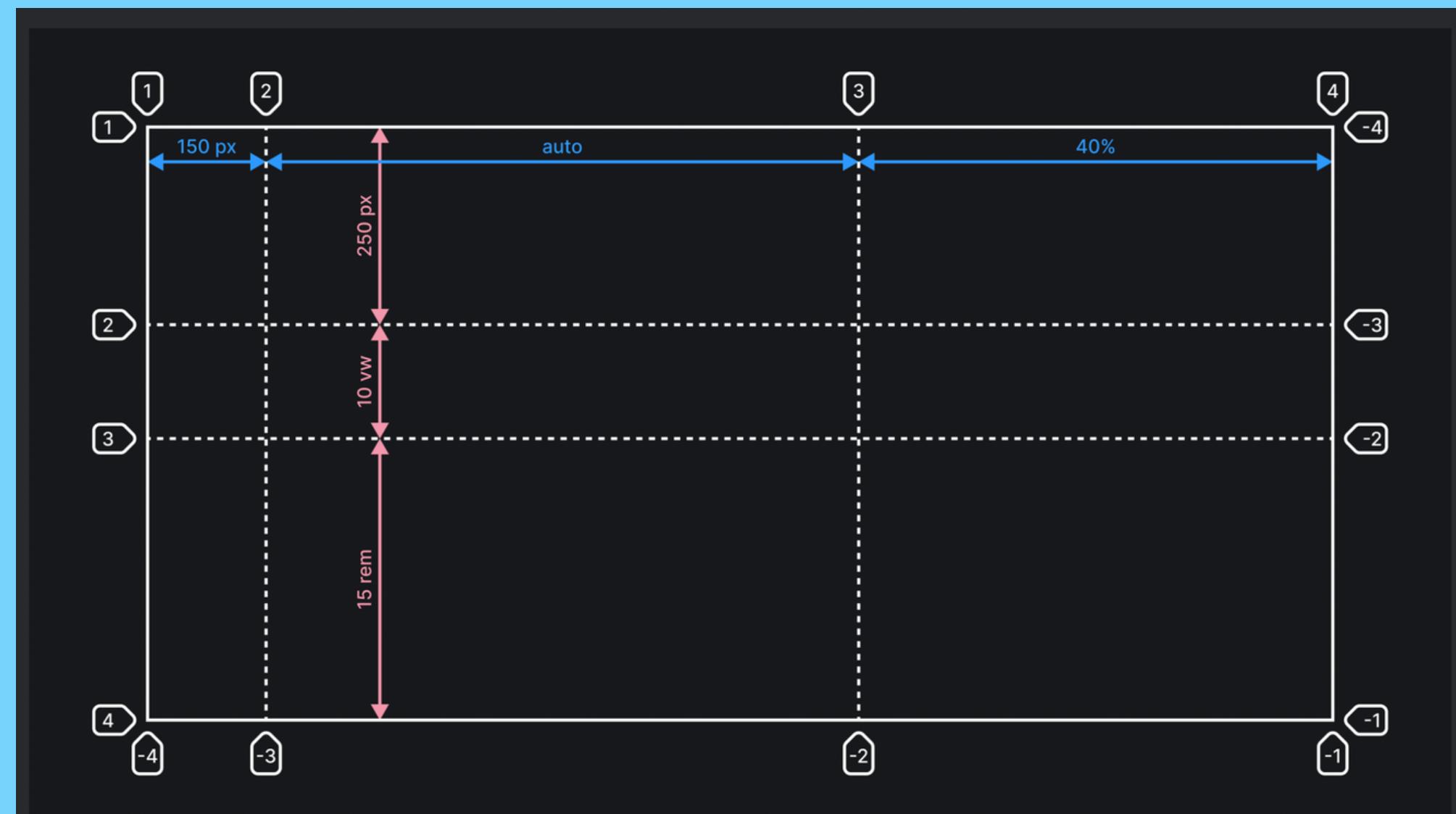
Грид-ячейка: пространство между соседними грид-линиями. Единица грид-сетки.

Грид-полоса: пространство между двумя соседними грид-линиями. Может быть проще думать о грид-полосе как о ряде или колонке.

display: grid: Основное свойство, которое применяется к родительскому элементу, чтобы создать сетку. Устанавливает элемент-контейнер в режим сетки.

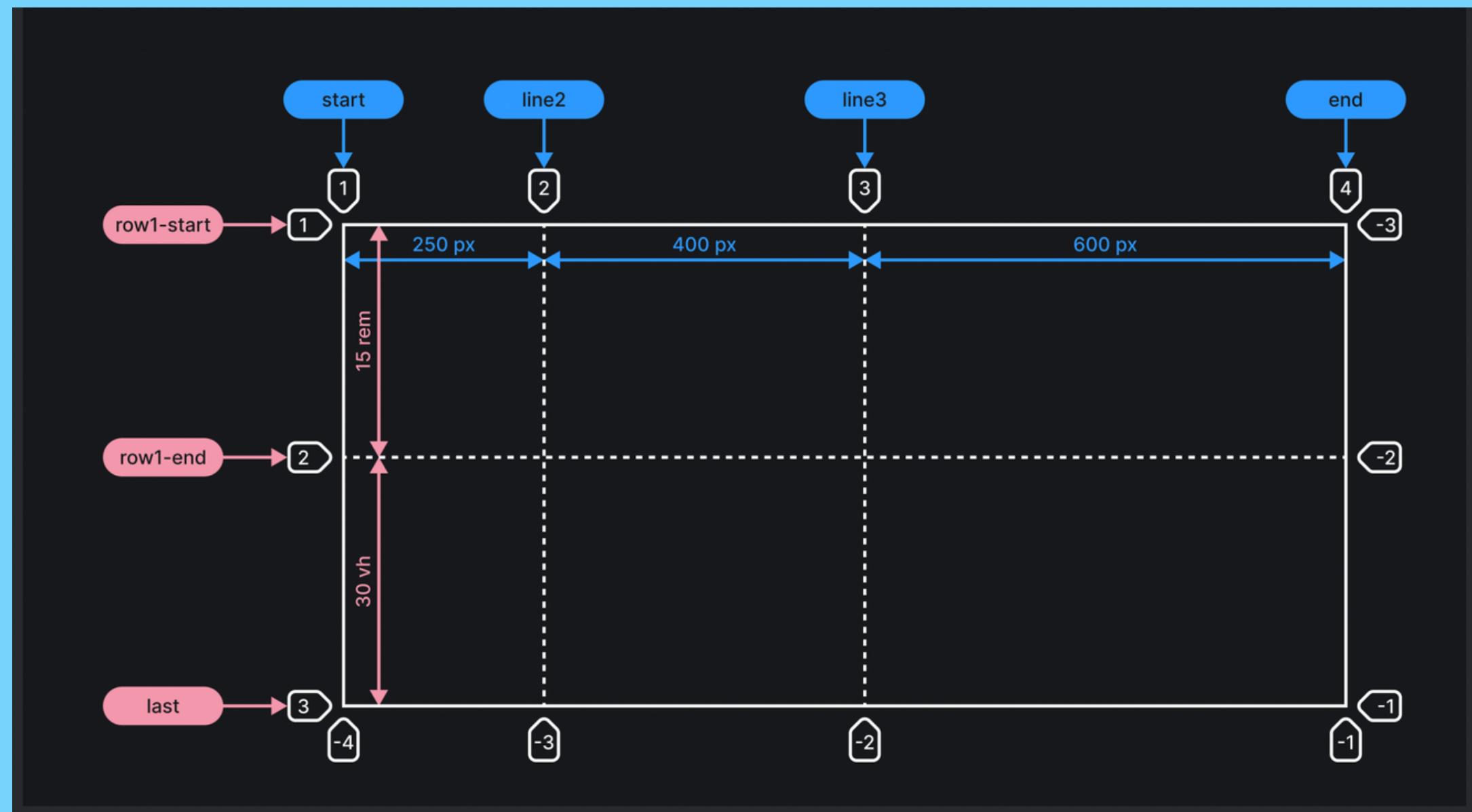
grid-template-columns и **grid-template-rows**: Эти свойства определяют структуру сетки, задавая количество и размеры столбцов и строк, соответственно

```
1 .container {  
2   display: grid;  
3   /* Будет создано 3 колонки */  
4   grid-template-columns: 150px auto 40%;  
5   /* Будет создано 3 ряда */  
6   grid-template-rows: 250px 10vw 15rem;  
7 }
```



Можно явно именовать грид-линии, используя для этого квадратные скобки:

```
css
1 .container {
2   display: grid;
3   grid-template-columns: [start] 250px [line2] 400px [line3] 600px [end];
4   grid-template-rows: [row1-start] 15rem [row1-end] 30vh [last];
5 }
```



Если нужны одинаковые колонки или ряды, то можно воспользоваться функцией `repeat()`.

Будет создано 3 колонки по 250 пикселей:

css

```
1 .container {  
2   display: grid;  
3   grid-template-columns: repeat(3, 250px);  
4 }
```

С появлением гридов у нас появилась и новая единица измерения: fr
fr (от fraction — доля, часть) отвечает за свободное пространство внутри
грид-контейнера.

Например, этот код создаст три колонки, каждая из которых будет
занимать 1/3 ширины родителя:

```
1 .container {  
2   display: grid;  
3   grid-template-columns: repeat(3, 1fr);  
4 }
```

Что аналогично записи:

css

```
1 .container {  
2   display: grid;  
3   grid-template-columns: 1fr 1fr 1fr;  
4 }
```

Свободное пространство рассчитывается после того, как место отдано всем фиксированным размерам. К примеру, в коде ниже сначала будет создана колонка шириной 200 пикселей, а затем свободное пространство — ширина родителя минус 200 пикселей — будет поделено между остальными колонками. Каждая будет занимать ширину $(100\% - 200\text{px}) / 2$:

css

```
1 .container {  
2   display: grid;  
3   grid-template-columns: 1fr 200px 1fr;  
4 }
```

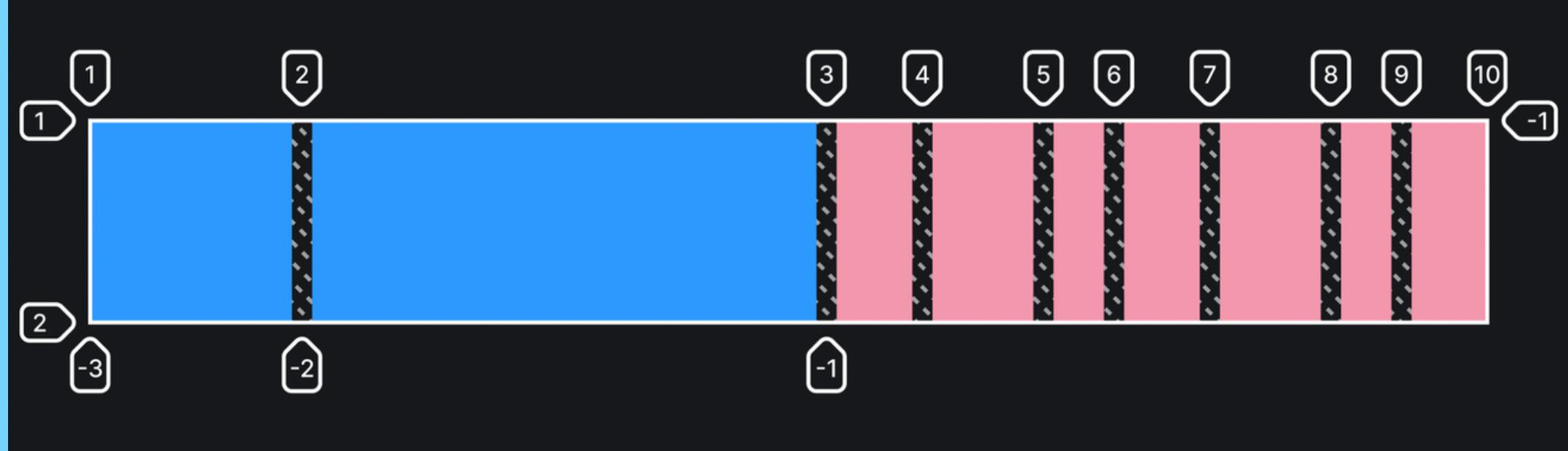
grid-auto-columns, grid-auto-rows

Скопировать ссылку "grid-auto-columns, grid-auto-rows"

Если элементов внутри грид-контейнера больше, чем может поместиться в заданные явно ряды и колонки, то для них создаются автоматические, неявные ряды и колонки. При помощи свойств `grid-auto-columns` и `grid-auto-rows` можно управлять размерами этих автоматических рядов и колонок.

```
1 .container {  
2   display: grid;  
3   grid-template-rows: 50px 140px;  
4   grid-auto-rows: 40px;  
5   gap: 20px;  
6 }
```

```
1 .container {  
2   display: grid;  
3   grid-template-columns: 200px 500px;  
4   grid-auto-columns: 75px 100px 50px;  
5   grid-auto-flow: column;  
6   gap: 20px;  
7 }
```

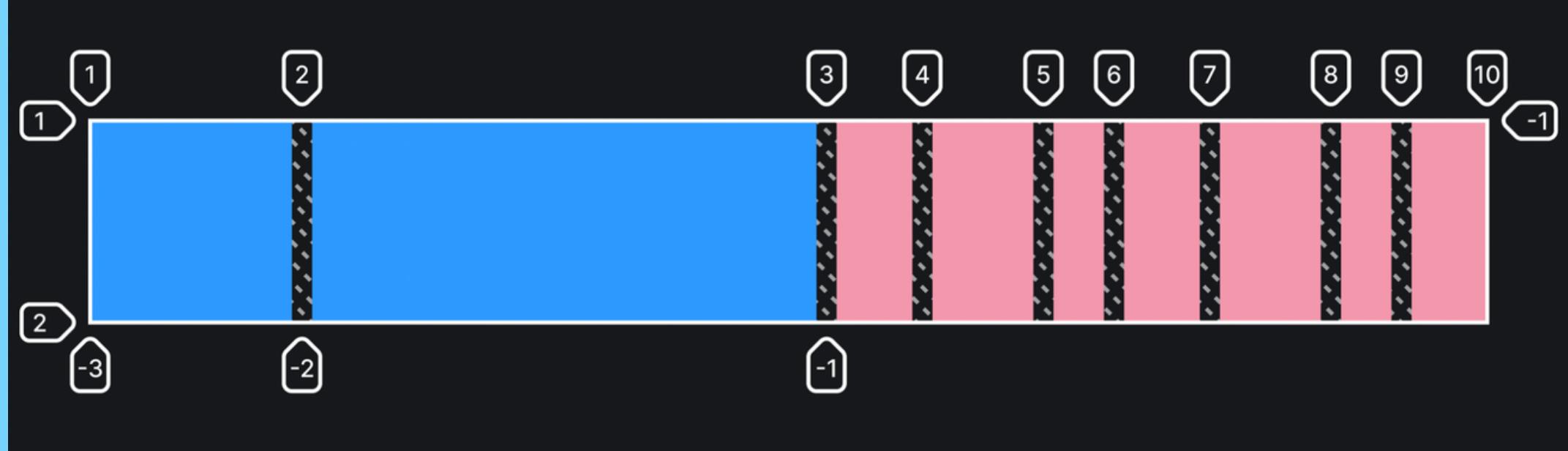


Важно указать при помощи `grid-auto-flow: column`, что элементы должны вставлять в колонки, чтобы элементы без контента были видны. Как видите, автоматически создаются колонки размером 75, 100 и затем 50 пикселей. И так до тех пор, пока элементы не закончатся.

grid-auto-flow

Если грид-элементов больше, чем явно объявленных колонок или рядов, то они автоматически размещаются внутри родителя. А вот каким образом — в ряд или в колонку — можно указать при помощи свойства `grid-auto-flow`. По умолчанию значение у этого свойства `row`, лишние элементы будут выстраиваться в ряды в рамках явно заданных колонок.

```
1 .container {  
2   display: grid;  
3   grid-template-columns: 200px 500px;  
4   grid-auto-columns: 75px 100px 50px;  
5   grid-auto-flow: column;  
6   gap: 20px;  
7 }
```



Важно указать при помощи `grid-auto-flow: column`, что элементы должны вставлять в колонки, чтобы элементы без контента были видны. Как видите, автоматически создаются колонки размером 75, 100 и затем 50 пикселей. И так до тех пор, пока элементы не закончатся.

grid-auto-flow

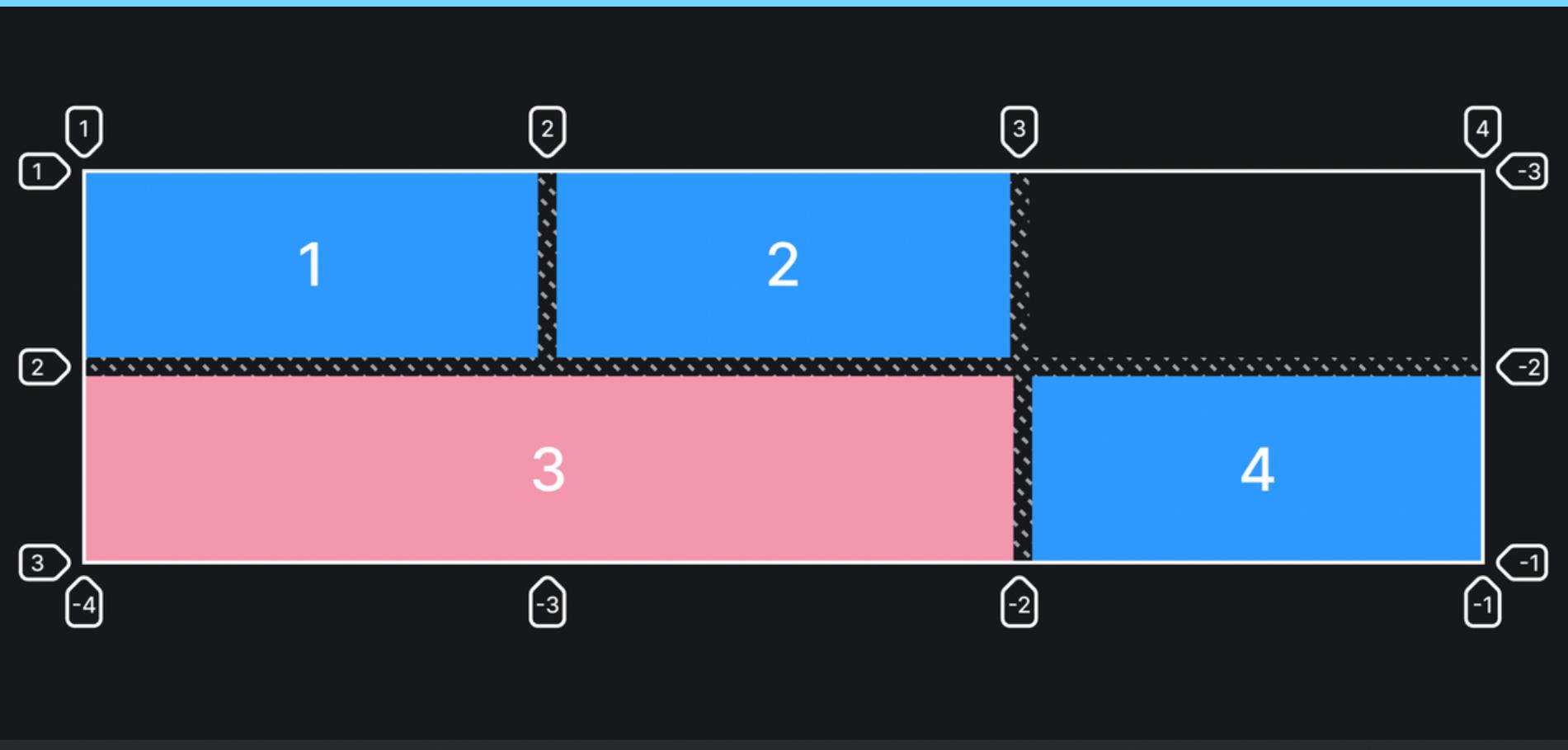
Если грид-элементов больше, чем явно объявленных колонок или рядов, то они автоматически размещаются внутри родителя. А вот каким образом — в ряд или в колонку — можно указать при помощи свойства `grid-auto-flow`. По умолчанию значение у этого свойства `row`, лишние элементы будут выстраиваться в ряды в рамках явно заданных колонок.

Возможные значения:

- `row` (значение по умолчанию) — автоматически размещаемые элементы выстраиваются в ряды.
- `column` — автоматически размещаемые элементы выстраиваются в колонки.
- `dense` — браузер старается заполнить дырки (пустые ячейки) в разметке, если размеры элементов позволяют. Можно сочетать с остальными значениями.

Принципы работы этого свойства удобнее всего изучать на примере, когда есть большой блок, который не помещается в одну грид-ячейку.

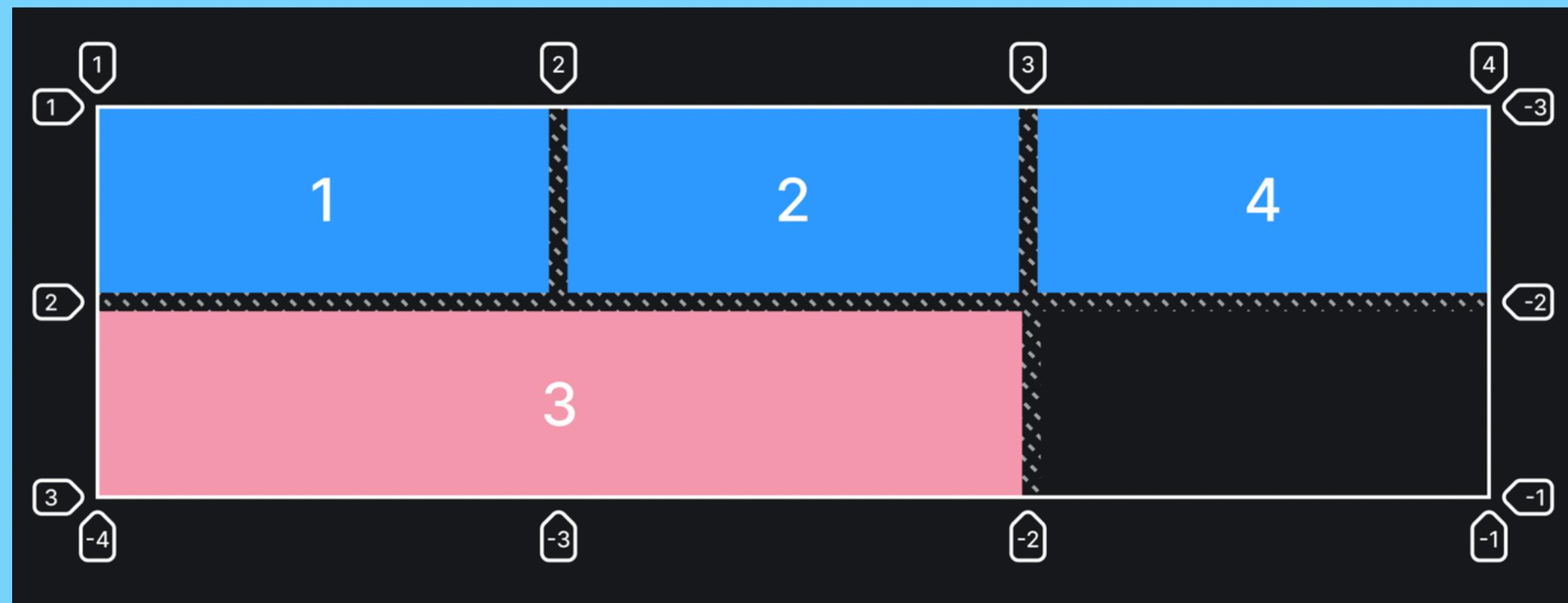
```
1 .container {  
2   display: grid;  
3   /* Три колонки */  
4   grid-template-columns: auto auto auto;  
5   /* Два ряда */  
6   grid-template-rows: auto auto;  
7   /* Автоматическое размещение в ряд */  
8   grid-auto-flow: row;  
9   /* Отступы между ячейками */  
10  gap: 20px;  
11 }  
12  
13 .item3 {  
14   /* Занимает один ряд и  
15   растягивается на две колонки */  
16   grid-column: span 2;  
17 }
```



Как видите, третий элемент не поместился в последнюю ячейку первого ряда и был перенесён в следующий ряд. Следующий за ним четвёртый элемент встал в ближайшую доступную ячейку во втором ряду.

Добавим к значению свойства `grid-auto-flow` слово `dense`:

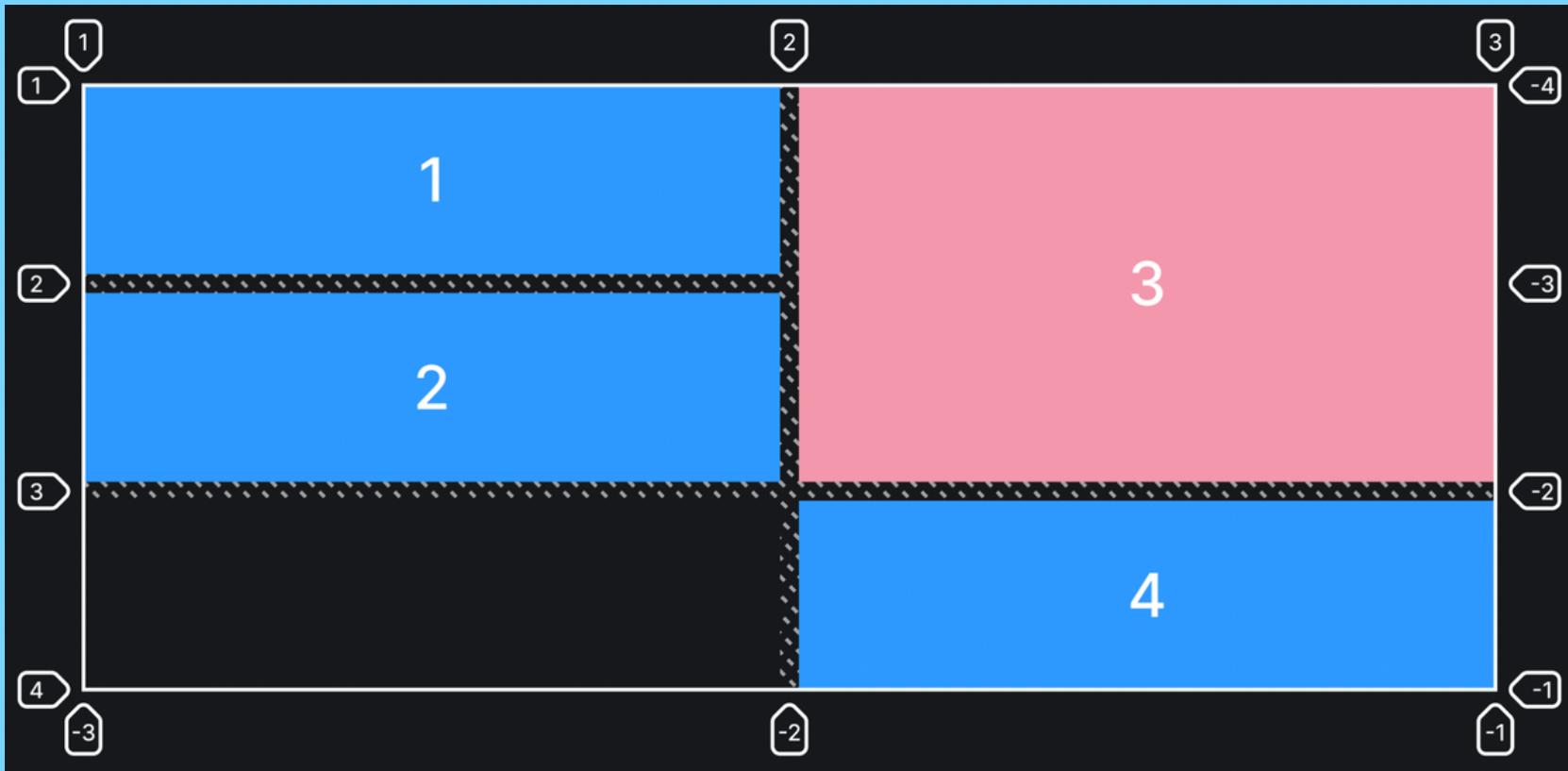
```
1 .container {  
2     /* Автоматическое размещение в ряд */  
3     grid-auto-flow: row dense;  
4 }
```



Теперь четвёртый элемент встал в ряд, но занял при этом пустую ячейку в первом ряду. Браузер старается закрыть все дырки в сетке, переставляя подходящие элементы на свободные места.

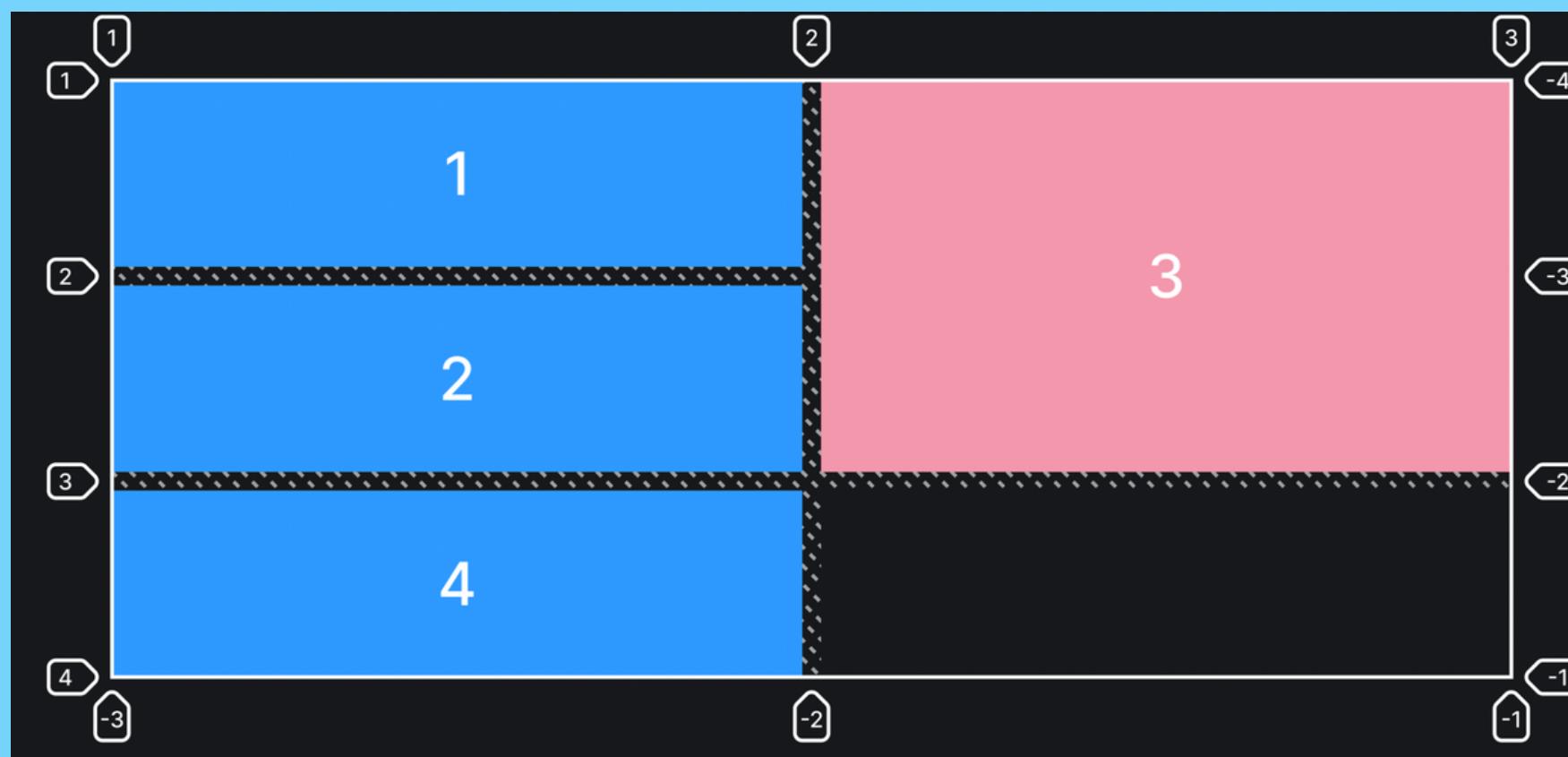
Посмотрим пример со значением
column:

```
1 .container {  
2   grid-template-columns: auto auto;  
3   grid-template-rows: auto auto auto;  
4   /* Автоматическое размещение в колонку */  
5   grid-auto-flow: column;  
6 }  
7  
8 .item3 {  
9   grid-row: span 2;  
10 }
```



Видим аналогичную картину: третий элемент не поместился в последнюю ячейку первой колонки и встал во вторую колонку. Следующий за ним четвёртый элемент встал ниже во второй колонке. Добавим `dense` к текущему значению:

```
1 .container {  
2   /* Автоматическое размещение в ряд */  
3   grid-auto-flow: column dense;  
4 }
```



В результате четвёртый элемент занял пустую ячейку в первой колонке.



6

THANK YOU!

Have a
great day
ahead.