**Grid Layout, часть 1. Введение**

В этой теме расскажем ещё об одном способе построения сеток — технологии ”Grid Layout“. По-русски — «гриды». Возникает справедливый вопрос: зачем нужны гриды, когда есть флексбоксы? Вы уже строите сетки с помощью этой технологии и всё нормально работает. Но у гридов есть преимущества: они позволяют описывать сложные сетки малым количеством кода.

Главная идея Grid Layout — разделение пространства на столбцы и строки. Распределять элементы можно сразу в двух направлениях: по горизонтали и вертикали. В отличие от флексбоксов, где главная ось всего одна. Поэтому гриды избавляют от большого количества обёрток вокруг элементов, которые во флексбоксах используются для выравнивания. Мы не хотим противопоставлять эти две технологии, но без сравнений не обойтись.

Когда меняется дизайн сайта или функциональность одного из блоков, разработчику приходится править HTML-разметку. Гриды позволяют менять расположение элементов, не залезая в HTML-код. Да, флексбоксы тоже это умеют, но только по одной оси.

А ещё на гридах разрабатывать адаптивные сайты намного проще, чем на флексах.

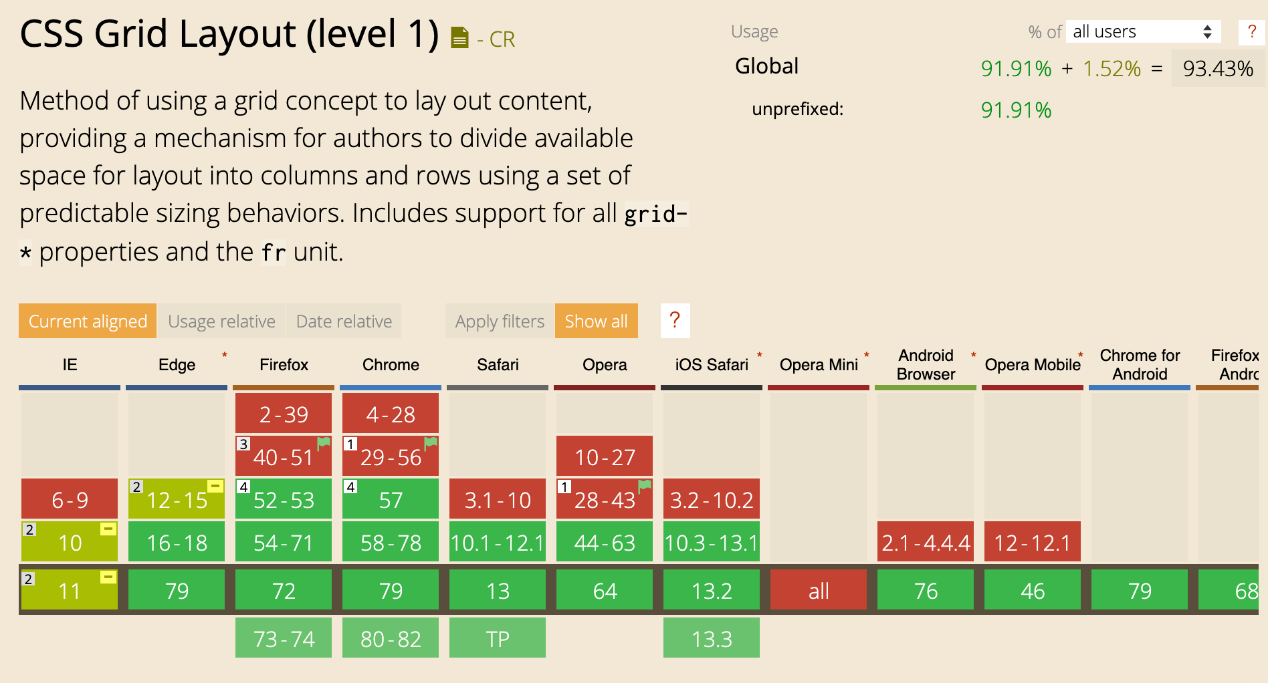
Неужели придётся выбирать между флексбоксами и гридами? Совсем необязательно. Флексбоксы в построении сетки отталкиваются от контента, а гриды ориентированы на структуру. Поэтому эти технологии часто применяют вместе. Гриды — там, где нужна чёткая структура. Например, чтобы создать общую сетку страницы. А флексбоксы — для мест, где нужно выравнивание в строке или колонке. Например, расположить иконки соцсетей в ряд с равными расстояниями между ними.

После изучения этой темы вы сможете создавать сетки страниц не только на флексбоксах, но и на гридах. А главное — вы поймёте, для каких задач какую технологию лучше применять.

**Что такое гриды**

CSS Grid Layout — система, созданная для построения сетки сразу в двух пространствах: по горизонтали и вертикали.

О гридах уже давно говорят. И чаще всего в разговоре о гридах можно услышать: «время гридов ещё не пришло». Это не так: веб-разработчики активно применяют гриды. Согласно [CanIUse](https://caniuse.com/#search=grid), почти все современные браузеры поддерживают гриды:



Согласно сайту CanIUse, поддержка гридов уже очень хорошая

Чтобы превратить элемент в грид-контейнер, нужно задать ему значение свойства display: grid;:

Скопировать кодCSS

.container {

display: grid;

}

Так мы создадим область сетки — «грид-контейнер» (англ. “grid container”). Все элементы внутри этого .container станут грид-элементами. Очень напоминает флекс. Но не будем торопиться с выводами.

Самое явное отличие от флексбоксов — как элементы выстраиваются по умолчанию. Когда мы создаём грид-контейнер, он состоит из одного столбца и одной строки, которые занимают весь его размер. Если добавить внутрь элементы, они выстроятся друг под другом. Для каждого из них создастся новая строка, но элементы всё ещё будут занимать один столбец. Как управлять количеством строк и столбцов, мы расскажем в третьем уроке.

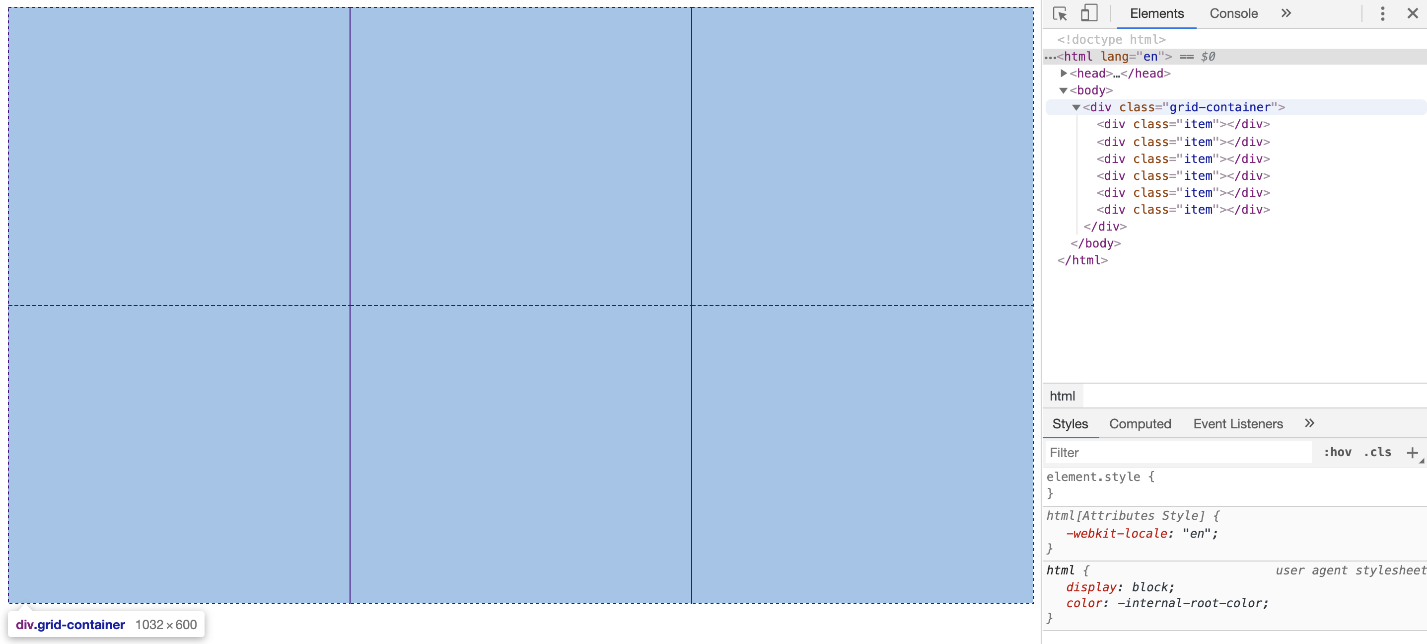
Последовательность грид-элементов в контейнере можно легко менять. Причём гриды справляются с этим намного лучше флексбоксов. Вам понадобится всего несколько строк CSS-кода, и — вуаля! Ваш сайт адаптивный.

Задавать свойства в гридах можно как самому контейнеру, так и его дочерним элементам. Мы подробно разберём эти свойства и устройство гридов в следующих уроках. А в этом уроке вы создадите грид-контейнер.

**Терминология гридов**

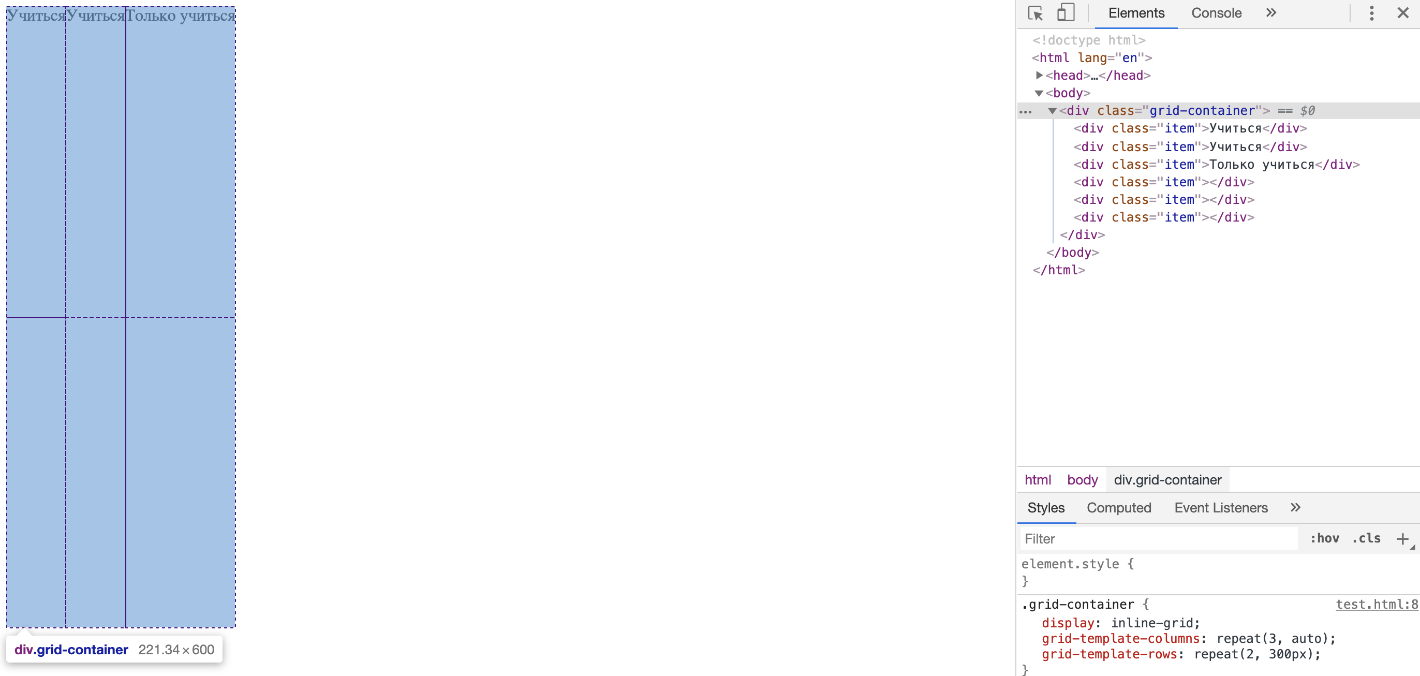
Чтобы понять, как всё устроено, рассмотрим строение гридов и обозначения подробнее.

**Грид-контейнер** (англ. grid container) — набор вертикальных и горизонтальных линий, которые пересекаются и создают области, в которых могут располагаться грид-элементы — все прямые потомки грид-контейнера.



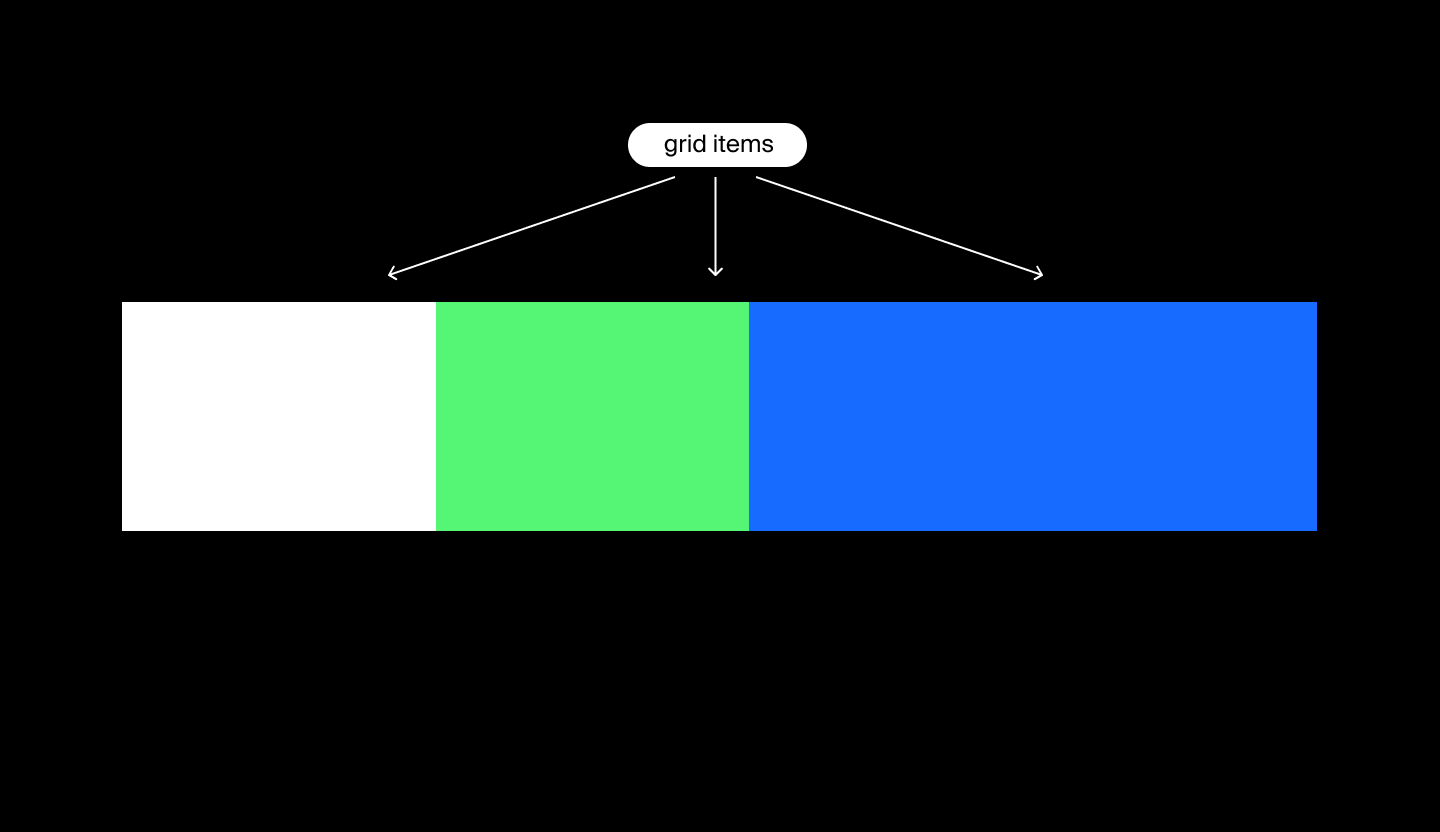
В отладчике контейнер и линии выглядят так

Про грид-контейнер вы уже знаете из предыдущего урока. Добавим, что кроме значения display: grid у него есть ещё одно — inline-grid. В этом случае грид-контейнер будет вести себя как строчный элемент и занимать только то пространство, которое необходимо, чтобы разместить грид-элементы.



В inline-grid контейнере элементы занимают лишь то пространство, которое необходимо для контента внутри

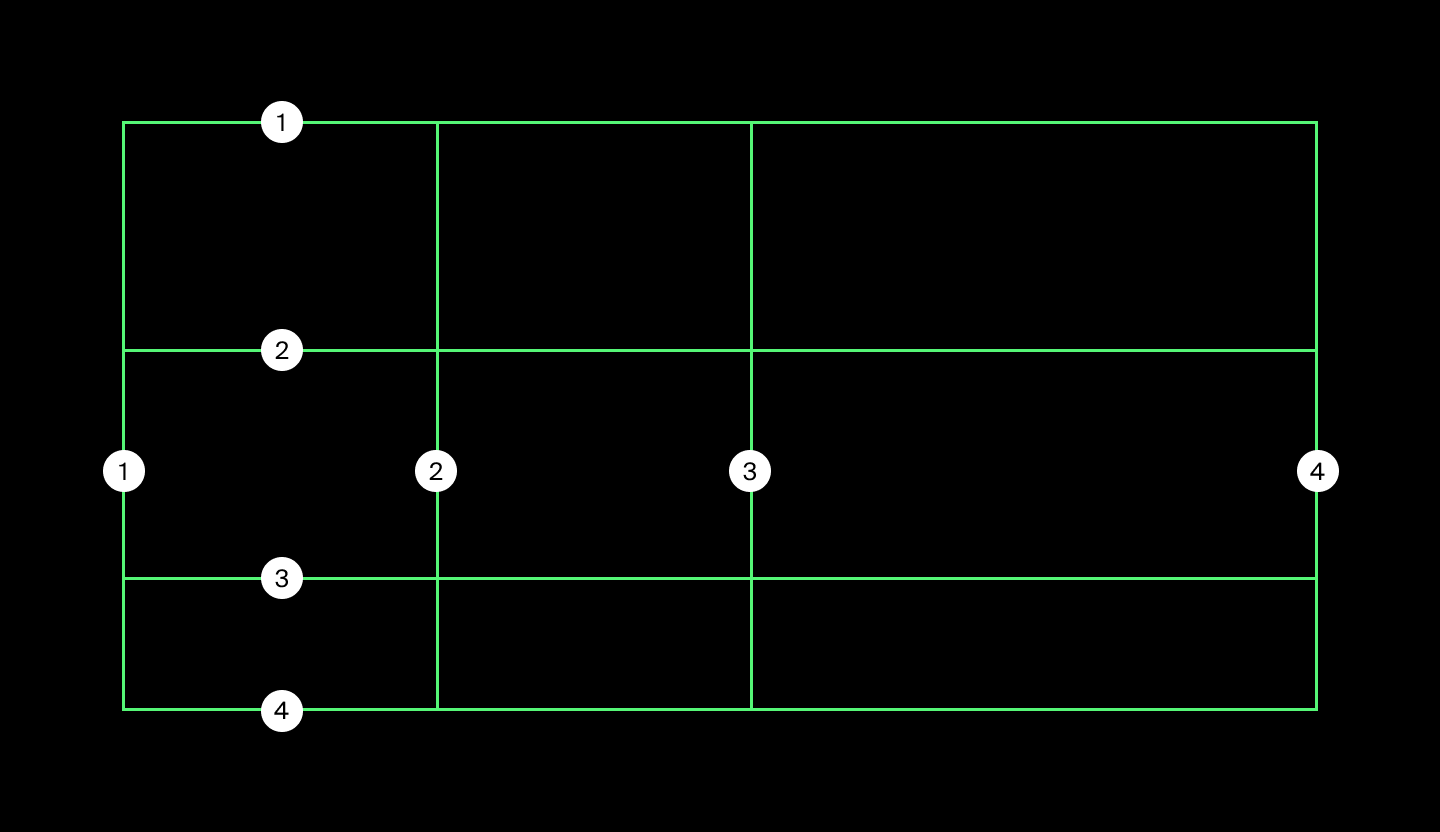
Грид-элементы (англ. grid items) — элементы, которые располагаются в грид-контейнере. То есть прямые потомки того элемента, которому мы задаём display: grid;.



Грид-элементы наполняют грид-контейнер

Грид-элементы распределяются в грид-контейнере, чтобы образовалась нужная нам сетка. Как именно это сделать, мы расскажем чуть позже.

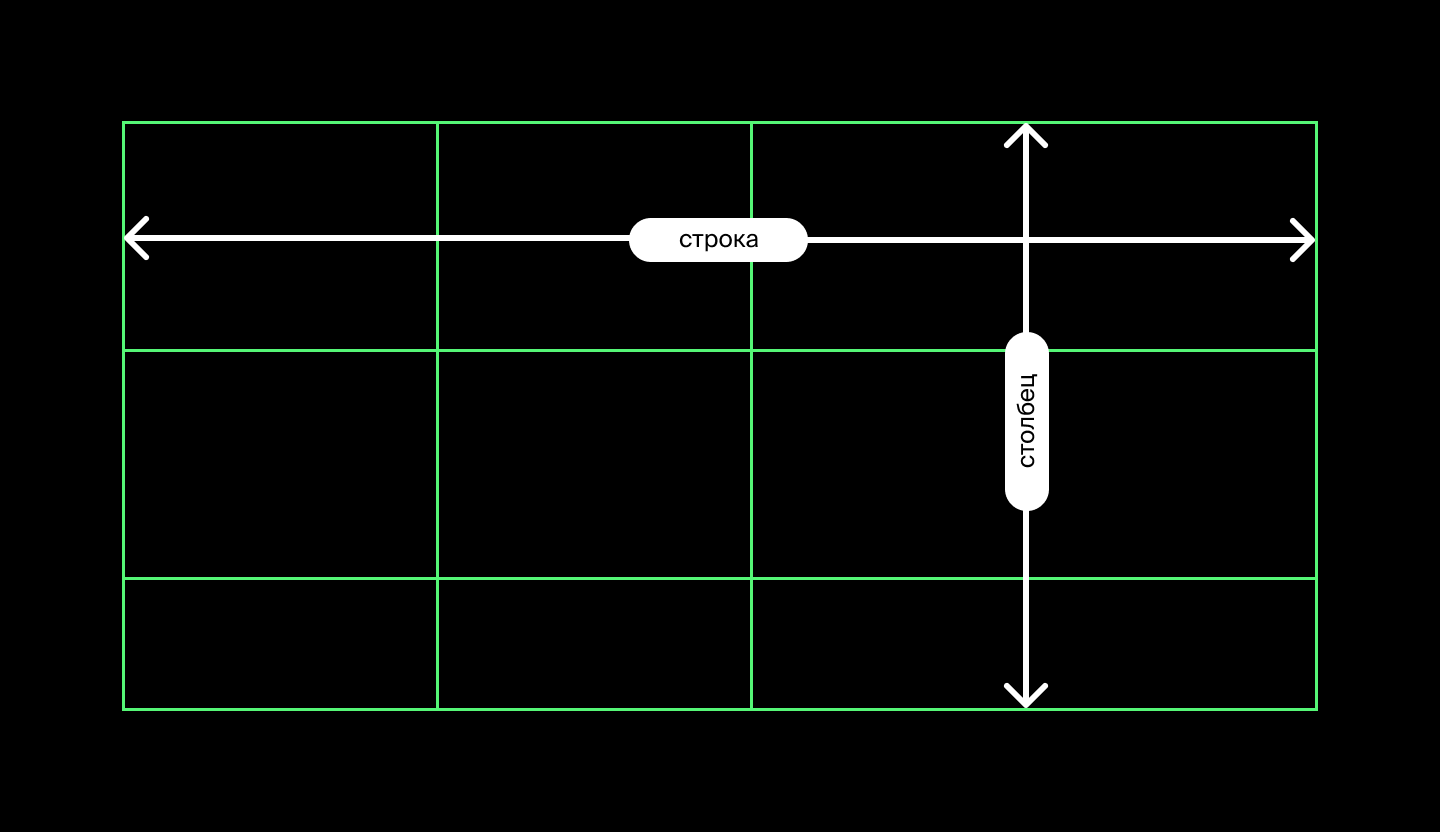
**Грид-линии** (англ. grid lines) — невидимые вертикальные и горизонтальные линии внутри грид-контейнера, которые пересекаются и создают строки и колонки. Линии имеют нумерацию начиная с единицы.



Грид-линии называют порядковыми номерами, начиная с единицы. Бывают горизонтальные и вертикальные линии

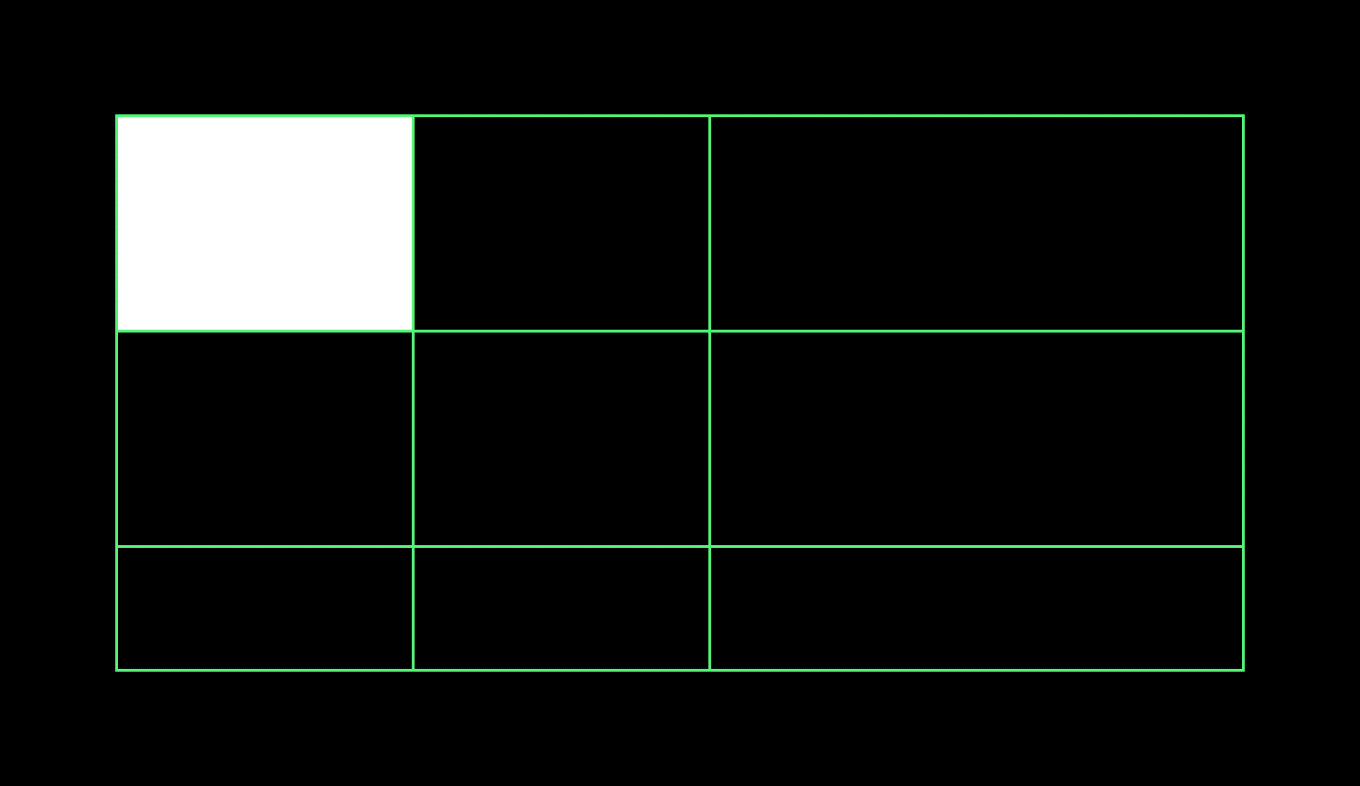
Грид-линии помогают в построении сетки и распределении элементов. Можно указать на какой линии по вертикали и горизонтали должен находиться элемент. Об этом мы тоже расскажем.

Грид-колонки и грид-строки (англ. grid tracks) — результат пересечения грид-линий, то есть пространство, заключённое между ними. Грид-колонки и грид-строки задают сетку страницы. Уже в следующем уроке научимся это делать.



Строки и колонки вместе называют грид-треками

**Грид-ячейки** (англ. grid cells) — получаются в результате пересечения строк и колонок.



Демонстрация отдельной грид-ячейки

Это единственный теоретический урок, дальше полностью погрузимся в практику.

**Столбцы и строки внутри грид-контейнера**

Вы уже немного знаете о гридах. Но нужно ещё научиться применять эту технологию и строить удобные сетки. Вы умеете создавать грид-контейнер, но сам по себе он не решает этой задачи. Элементы внутри него сами себя не выстроят. Нужно указать, где им находиться.

Сделать это можно, разделив пространство внутри грид-контейнера на столбцы и строки. То есть создав явную сетку. Для этого существуют свойства grid-template-columns и grid-template-rows. Значения этих свойств состоят из перечисления размеров грид-полос через пробел. Применяются они к самому контейнеру.

Скопировать кодCSS

.container {

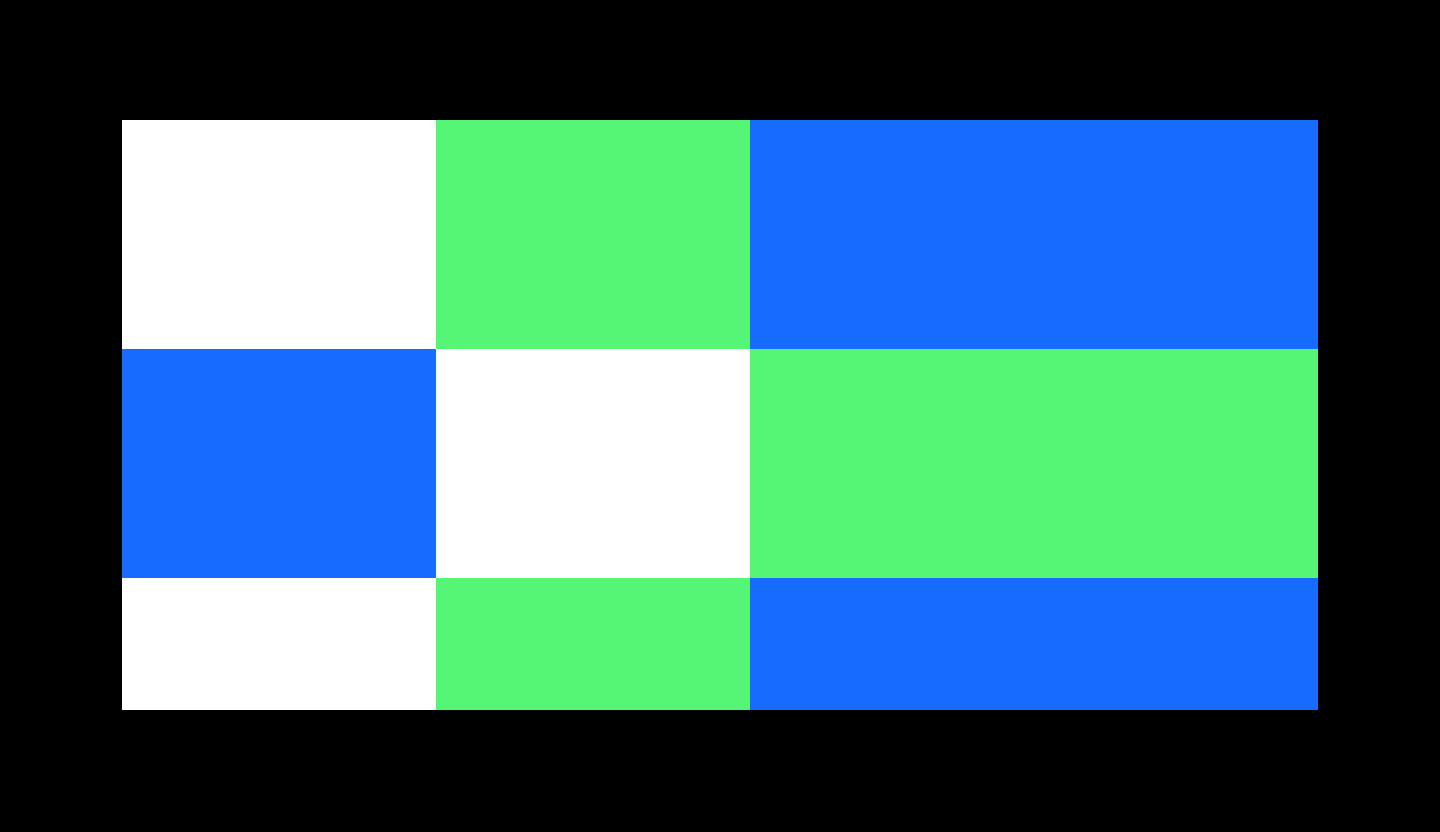
display: grid;

grid-template-columns: 100px 100px 150px;

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

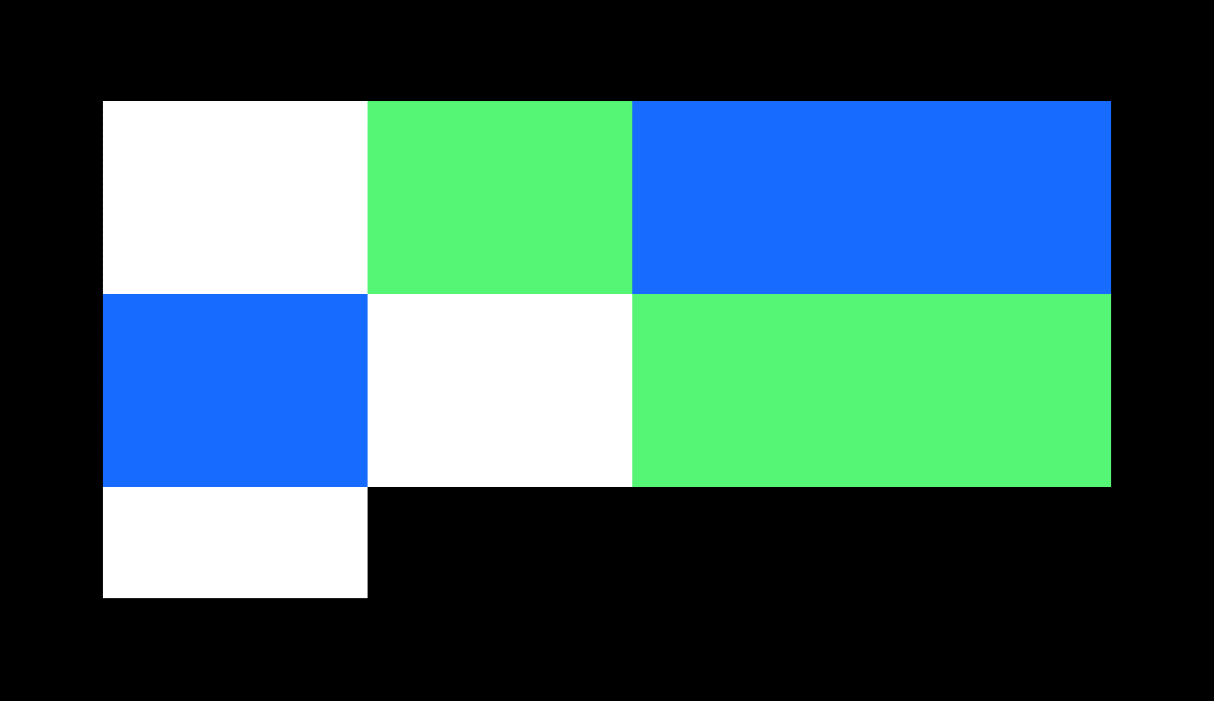
}

В результате мы получим три столбца и три строки. Элементы распределятся по сетке автоматически.



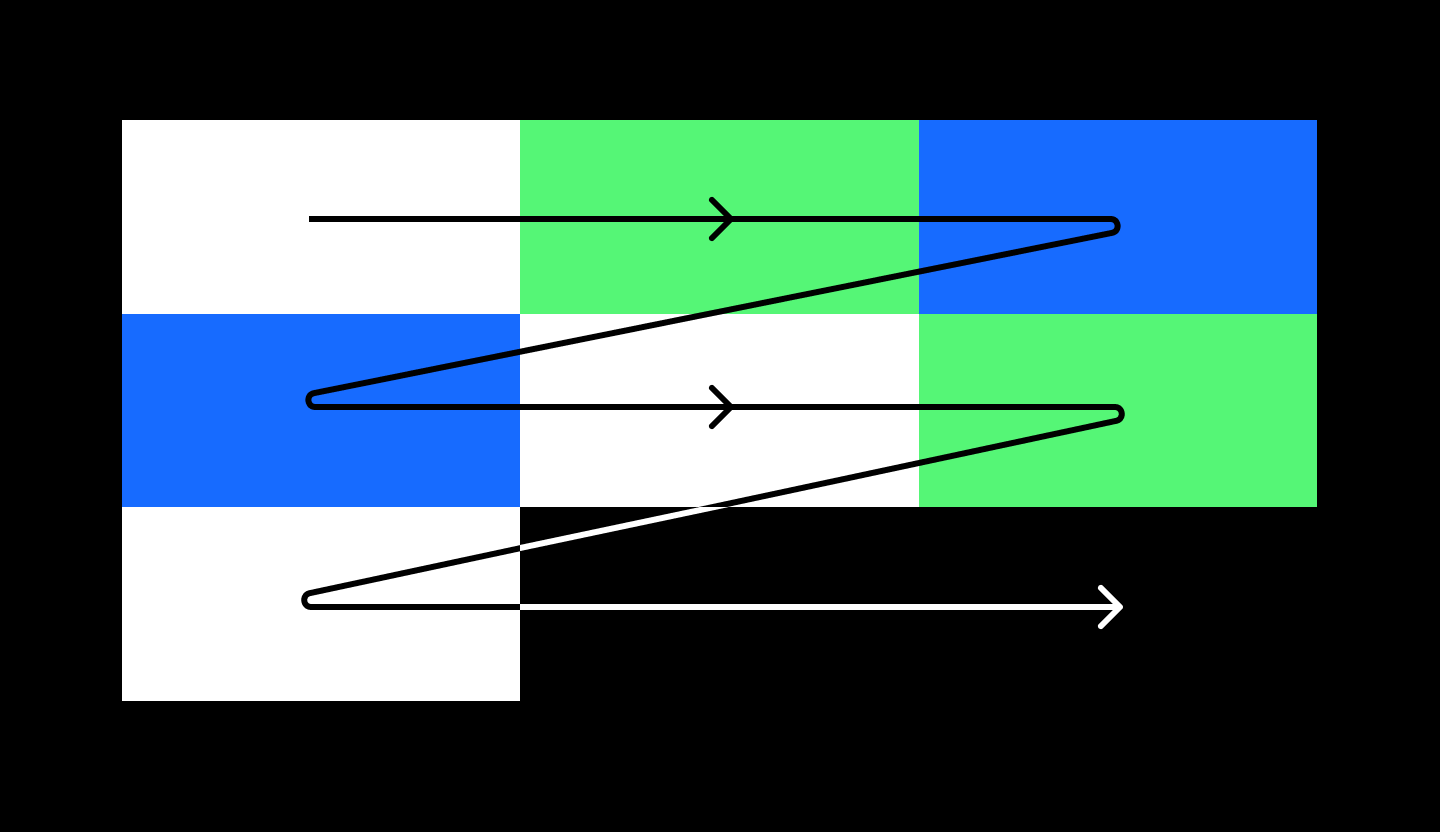
Три колонки и три строки

В примере 9 элементов, поэтому все ячейки заполнились элементами. Если бы элементов было меньше, например 7, то пустыми остались бы две последние ячейки:



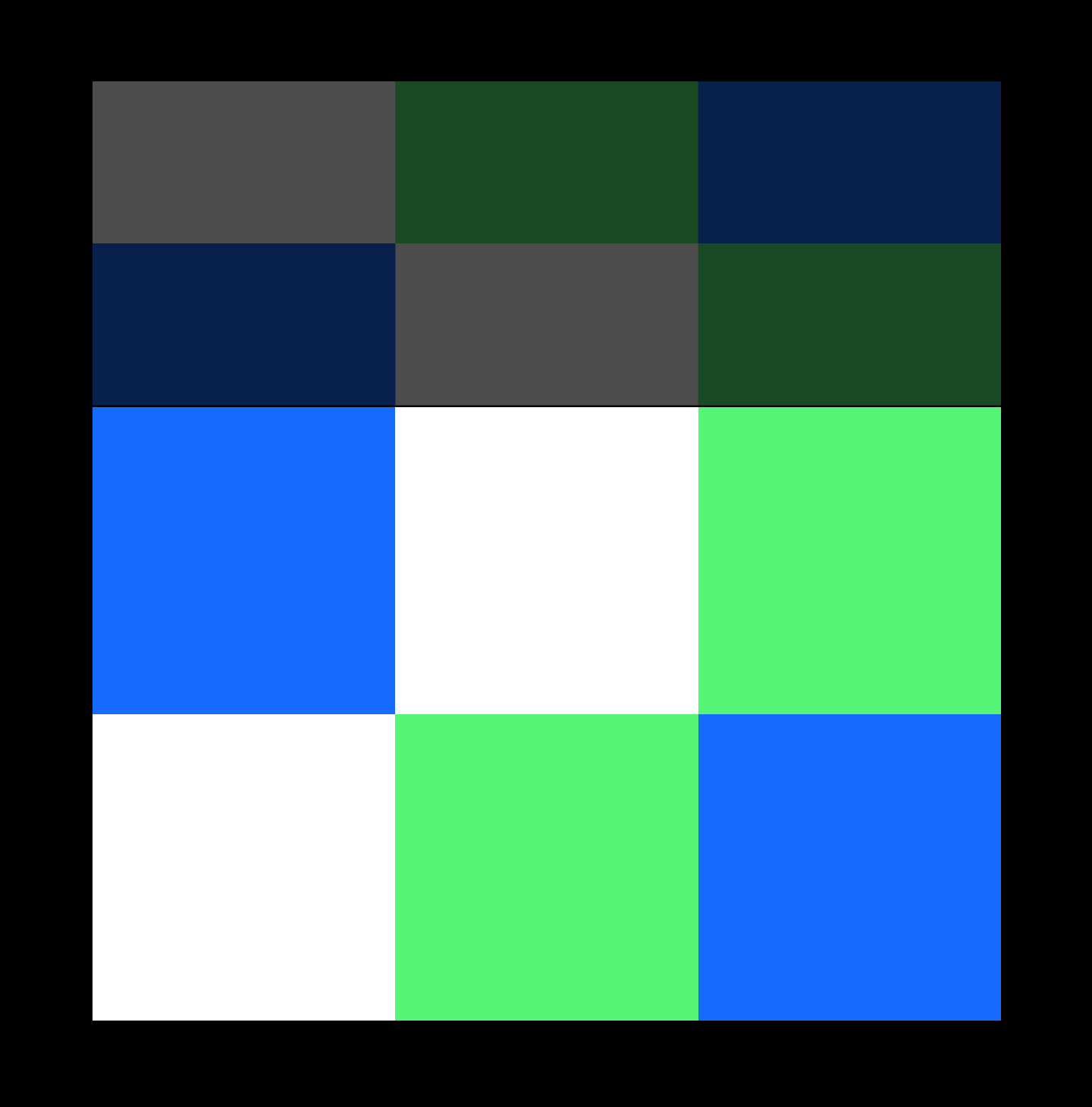
Девять ячеек, но элементов — семь

По умолчанию распределение элементов происходит слева направо и сверху вниз:



Как заполняются грид-ячейки.

Если элементов будет больше, они продолжат выстраиваться по этому же принципу. Но в таком случае появятся неявные ячейки, состоящие из новых строк и колонок. То есть ячейки, которые мы не задавали с помощью свойств. Грид создаёт их сам. По умолчанию неявные ячейки имеют значение auto. Если у них не будет явной ширины за счёт контента, CSS-свойств или свободного пространства, то они схлопнутся.



Мы добавили ещё шесть элементов со значением auto, которые поделили между собой доступное пространство

Размеры колонок могут задаваться в пикселях, процентах или рассчитываться автоматически. Есть и другие способы, мы будем знакомиться с ними постепенно. При значении auto размер грид-полос подстраивается под размер грид-элементов так, чтобы уместился самый большой из них. Другие элементы поделят между собой оставшееся пространство.

После добавления ещё трех ячеек с размером auto, все элементы поделят доступное им пространство.

**Создание отступов**

С построением сетки вы справились. Но какая же сетка обходится без отступов? Будь то товары на странице или блоки сайта, они нужны почти всегда.

Вы можете использовать свойство margin для грид-элементов. Но в таком случае отступы будут и у крайних элементов сетки, придётся сбрасывать их отдельно. К тому же margin будет ужимать элементы, так как отталкиваться они будут не друг от друга, а от краёв грид-линий. Это тоже пришлось бы решать порцией кода.

Чтобы избавиться от подобных проблем, разработчики гридов придумали решение — свойство gap:

Скопировать кодCSS

.container {

display: grid;

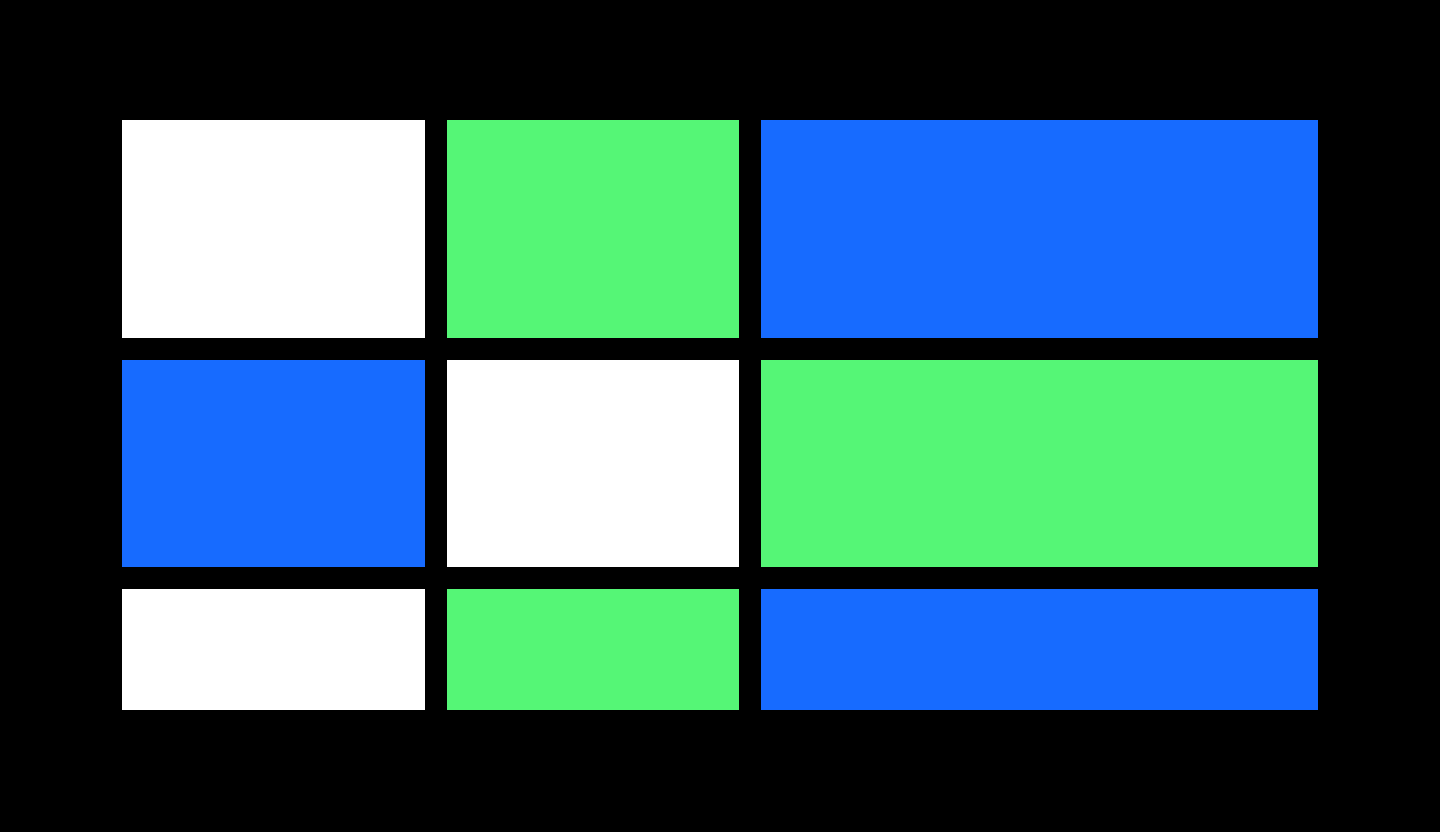
grid-template-columns: 100px 100px 150px;

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

gap: 10px;

}

Между строками и колонками появились зазоры. В них нельзя ничего поместить, но визуально элементы разделены отступами. Между крайними элементами сетки и контейнером их нет, поэтому не нужно ничего обнулять, как в случае с марджинами. Все отступы фиксированные и не изменятся в результате сжатия или растяжения. Чему будут несказанно рады дизайнеры, которые предпочитают, чтобы конечный результат был максимально приближен к макету.



Работа свойства grid-gap

Задавать отступы можно не только всем элементам сразу, но и отдельно строкам или колонкам. Для этого есть свойства column-gap для колонок и row-gap для строк:

Скопировать кодCSS

.container {

display: grid;

grid-template-columns: 100px 100px 150px;

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

column-gap: 20px;

row-gap: 10px;

}

Можно указать оба значения в свойстве gap в формате:

Скопировать кодCSS

.container {

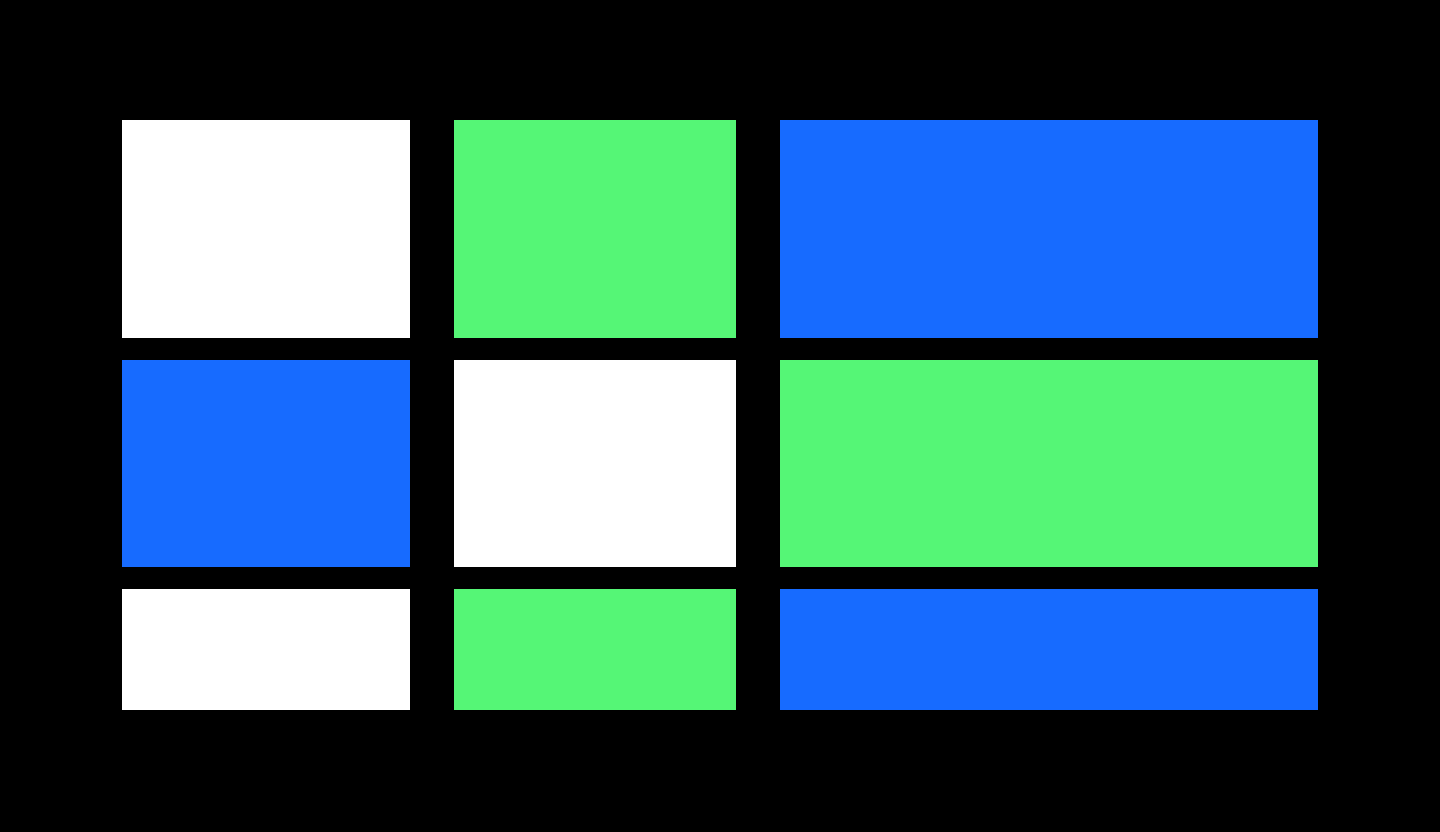
display: grid;

grid-template-columns: 100px 100px 150px;

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

gap: 10px 20px;

}



Grid-gap можно задавать разный — как для строк, так и для колонок

# Функция repeat

Лень — двигатель прогресса. Если что-то можно оптимизировать или работу можно как-то сократить — разработчики обязательно оптимизируют и сократят. Разработчики гридов — не исключение.

Вы уже умеете создавать желаемое количество строк и столбцов любых размеров. Иногда нужно, чтобы они были одинаковой величины. Например, при построении сетки товаров. Чтобы не писать каждый раз одно и то же значение для свойств grid-template-columns и grid-template-rows, придумали свойство repeat.

Синтаксис его довольно прост:

Скопировать кодCSS

grid-template-columns: repeat(*/\* количество столбцов или строк, нужная ширина \*/*);

grid-template-rows: repeat(*/\* количество столбцов или строк, нужная ширина \*/*);

Свойство repeat заметно сокращает код:

Скопировать кодCSS

*/\* вместо этого: \*/*

grid-template-columns: 20% 20% 20% 20% 20%;

*/\* можно написать: \*/*

grid-template-columns: repeat(5, 20%);

Результат будет одинаковый — мы получим пять столбцов, каждый шириной 20% от грид-контейнера.

# Единица измерения: фракция

В третьем уроке мы упоминали, что существуют разные способы задавать размеры. Вы знакомы с процентами, пикселями и значением auto. Все эти способы применяют и в других технологиях, например — во флексбоксах. В этом уроке мы расскажем о новой единице измерения. Её придумали для удобного распределения размеров столбцов и строк в гридах. Обозначается она так fr и называется «фракция».

**Фракция** — единица доступного пространства, которое можно разделить на сколько угодно частей. Когда мы задаём столбцу размер 1fr, он смотрит, есть ли вокруг другие фракции. Если других фракций нет, забирает всё свободное пространство. Если другие фракции есть, смотрит, сколько места они занимают и сколько пространства осталось ему. Элемент размером 2fr будет занимать вдвое больше свободного пространства, чем 1fr, а 3fr — втрое. Это произойдёт только в случае, когда в сетке есть другие строки или колонки с фракцией в качестве размера.

Рассмотрим на примере:

Скопировать кодCSS

.container {

width: 600px;

grid-template-columns: 100px 2fr 3fr;

}

Размер контейнера 600px. У первого столбца фиксированная ширина 100px, значит свободного пространства останется 500px. Пять фракций поделят его между собой. У третьего столбца больше фракций, он заберёт 3/5 части свободного пространства и станет равным 300px. А второму останется 2/5 части, он станет равным 200px.

Мы могли бы использовать свойство auto вместо фракций, но это значение нельзя контролировать. Мы не можем сказать элементу со значением auto: «Будь 5/8 от свободного пространства». А с фракциями можем.

Также фракции помогают избавиться от лишних расчётов размеров. Например, нам нужно создать 6 колонок равной ширины. И вместо того, чтобы рассчитывать, сколько места нужно отвести под каждую колонку, мы просто сделаем их 1fr:

Скопировать кодCSS

grid-template-columns:repeat(6, 1fr);

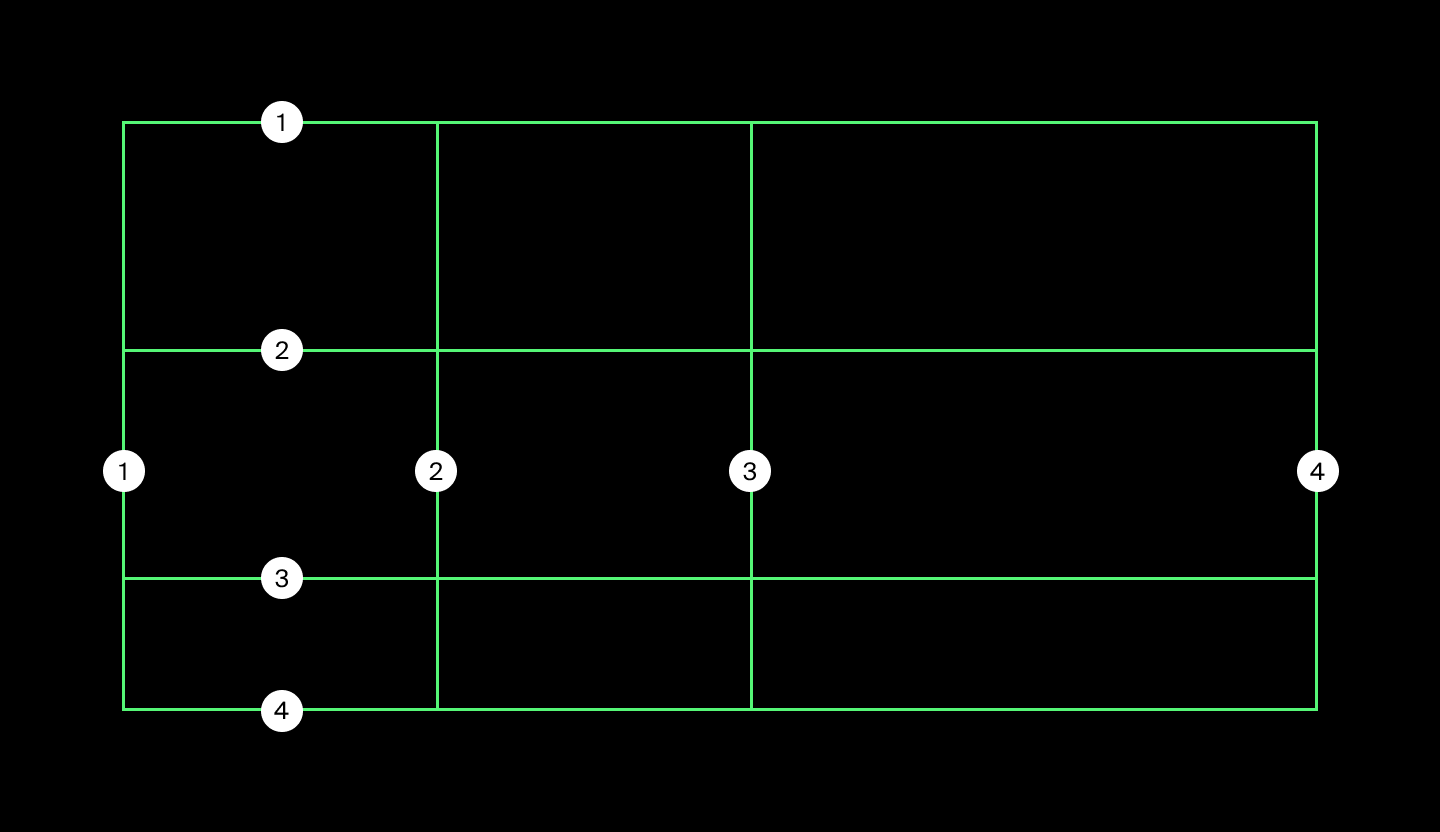
Ещё один плюс фракций — отступы учитываются автоматически и вычитаются из конечной ширины столбца. Это поможет избежать случайного переполнения контейнера.

# Расположение элементов в грид-контейнере

Как распределять элементы в контейнере автоматически, мы объяснили. Теперь перейдём к самому интересному.

Представьте, что какой-то из элементов, например картинка, должен занимать не одну ячейку, а несколько. Ну потому что так красивее. Гриды справятся с этой задачей как никто другой — помогут грид-линии.

Во втором уроке мы упоминали, что каждая грид-линия имеет свой порядковый номер, начиная с 1.



Грид-линии в контейнере

Мы можем использовать эту нумерацию, чтобы указать элементу, где ему начинаться, а где заканчиваться. Для этого разработчики гридов придумали специальные свойства: grid-column-start и grid-column-end для вертикальных грид-линий, grid-row-start и grid-row-end для горизонтальных. Эти свойства применяются к самомý грид-элементу.

Например, у нас есть сетка из 6 элементов:

Скопировать кодHTML

*<!-- Разметка -->*

<div class="container">

<div class="block block\_size\_big">1</div>

<div class="block">2</div>

<div class="block">3</div>

<div class="block">4</div>

<div class="block">5</div>

<div class="block">6</div>

</div>

Скопировать кодCSS

*/\* Стили: \*/*

.container {

display: grid;

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

grid-template-rows: 1fr 1fr 1fr;

gap: 40px 5px;

}

Все элементы распределятся автоматически в три колонки и в три ряда. Но мы хотим, чтобы первый блок занимал не одну ячейку, а две. То есть от первой линии до третьей. Для этого запишем ему свойства grid-column-start и grid-column-end.

Скопировать кодCSS

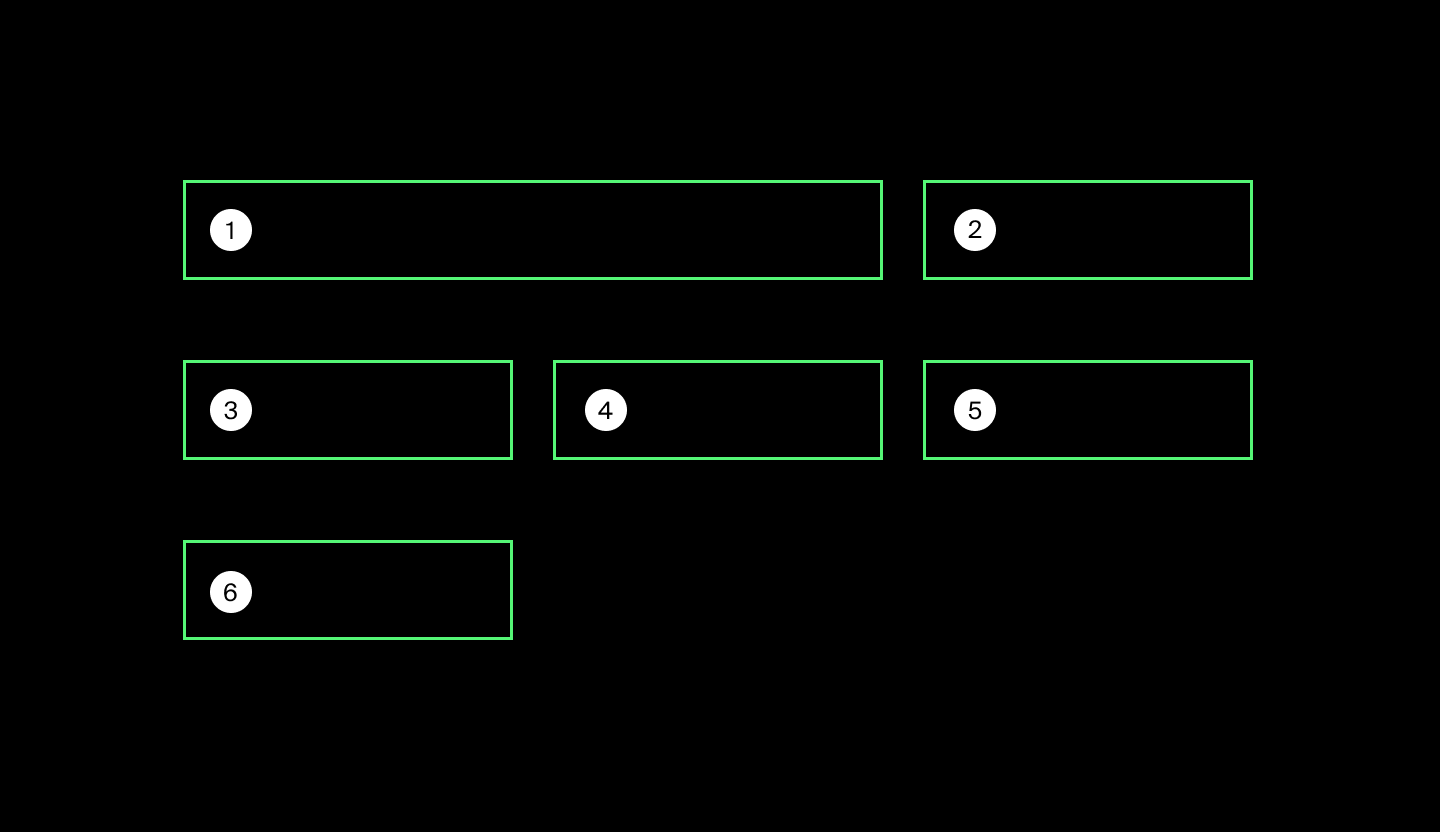
.block\_size\_big {

grid-column-start: 1;

grid-column-end: 3;

}

Теперь элементы выглядят так:



Первый элемент растянулся с 1 по 3 линию по горизонтали

Теперь скажем первому элементу занять пять ячеек по вертикали. То есть с первой по шестую линию. Для этого допишем ему свойства grid-row-start и grid-row-end.

Скопировать кодCSS

.block\_size\_big {

grid-column-start: 1;

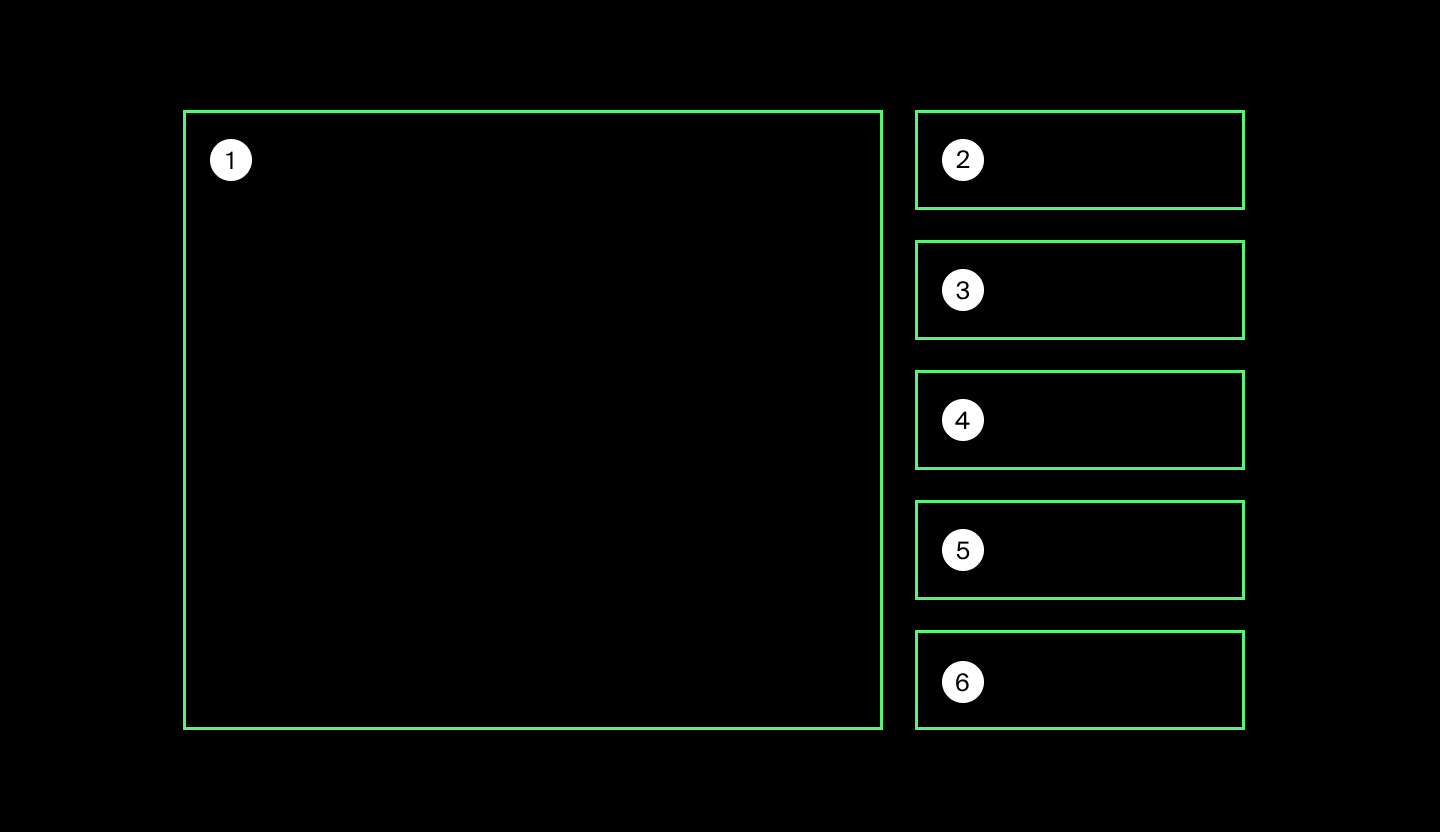
grid-column-end: 3;

grid-row-start: 1;

grid-row-end: 6;

}

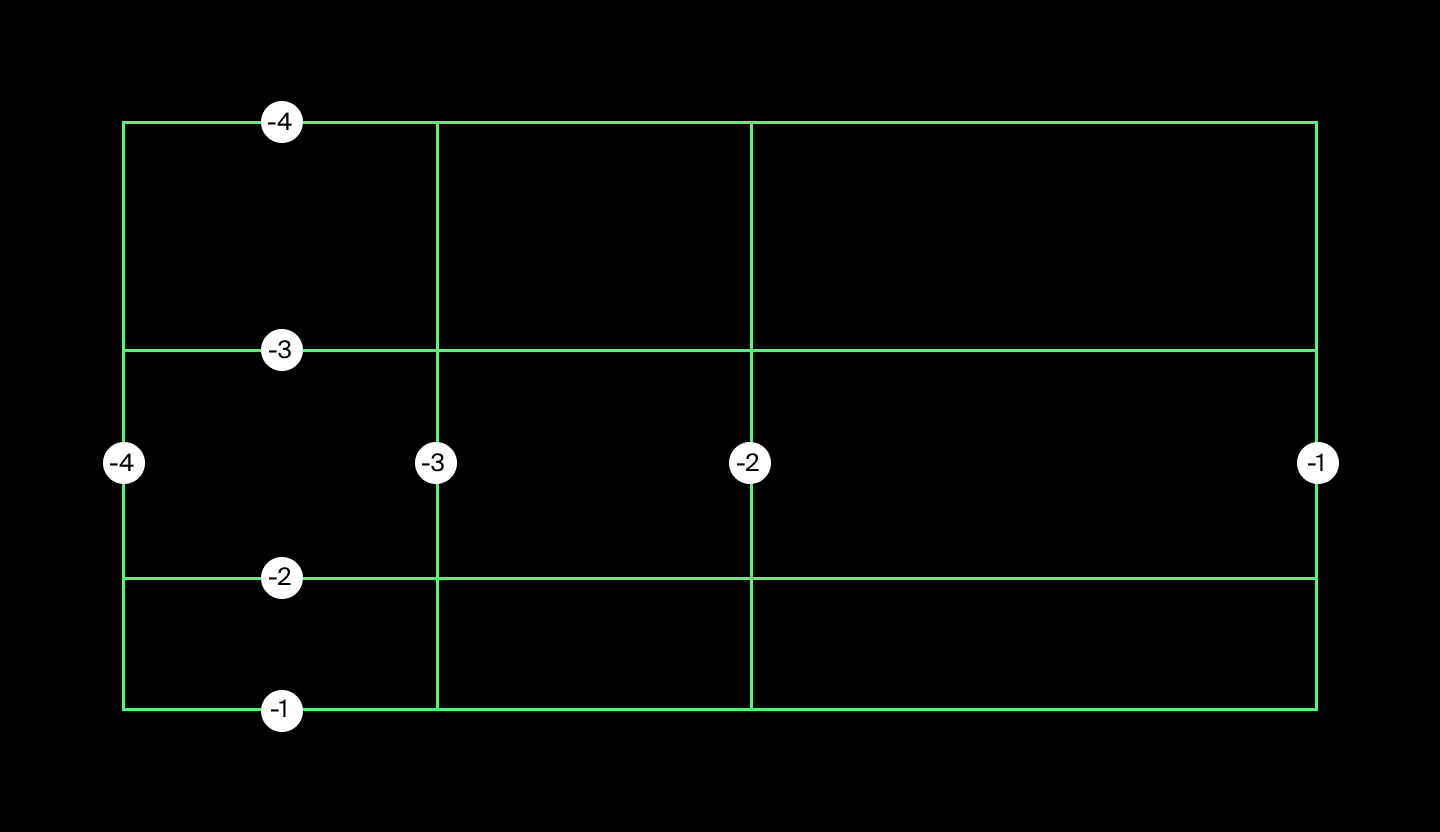
Получим такую сетку:



Первый элемент растянулся с 1 по 6 линию по вертикали

Остальные элементы подстроились к новым условиям. Мы распределяем элементы так, как нам захочется, не меняя разметки.

Но это ещё не всё. Грид-линии имеют также отрицательные номера: –1, −2, −3 и т. д.



Обозначение линий отрицательными числами

Это даёт возможность перемещать элемент по сетке, например, с начала в конец. Переместим блок block\_size\_big.

Скопировать кодCSS

.block\_size\_big {

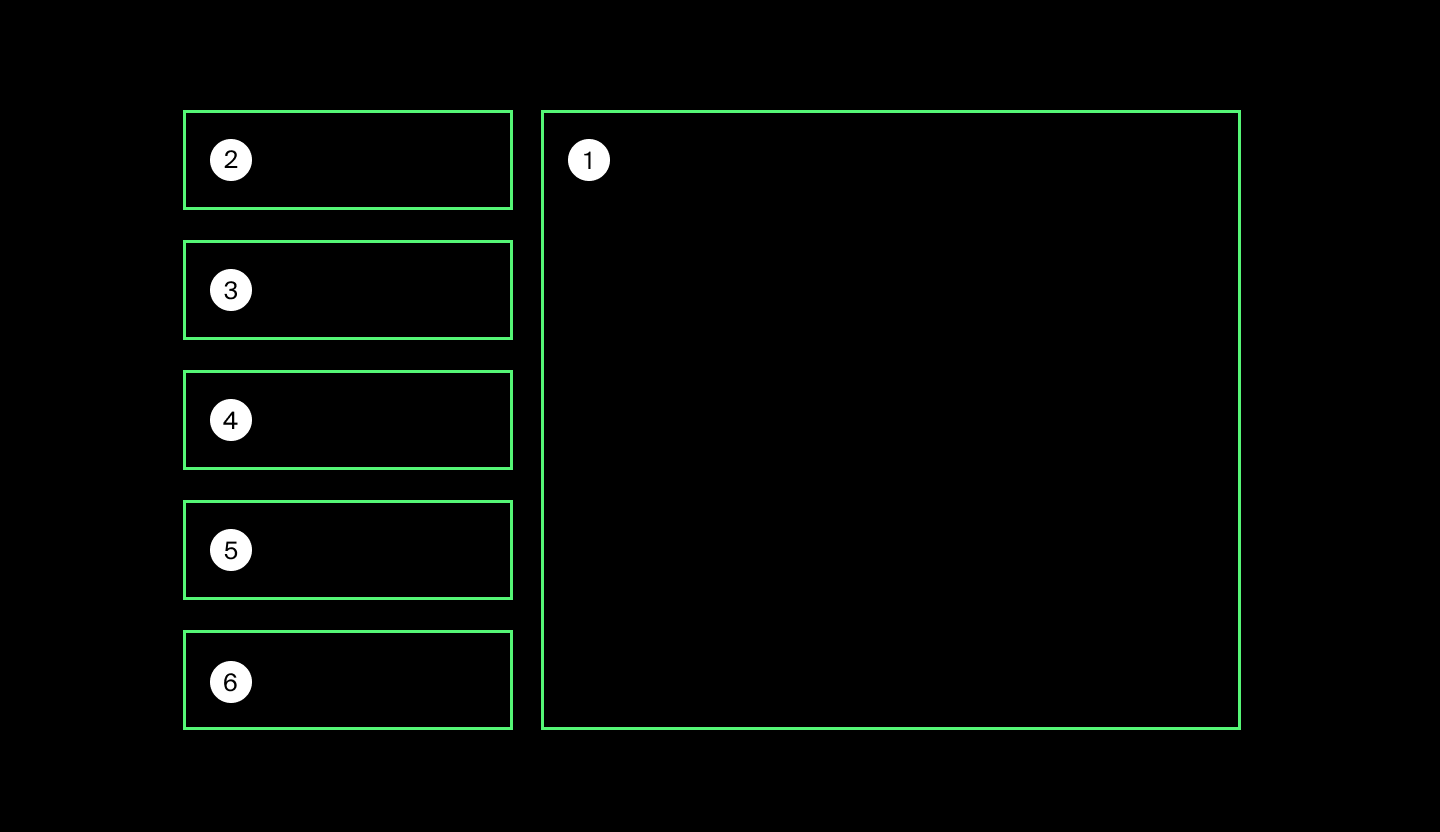
grid-column-start: -1;

grid-column-end: -3;

grid-row-start: 1;

grid-row-end: 6;

}



Первый элемент переместился в конец сетки

Элемент оказался совсем в другом месте, так как мы сказали ему «начинаться с минус первой линии и заканчиваться минус третьей».

Разработчики позаботились и о сокращённой записи свойств: grid-row и grid-column. Указываем начало грид-линии, ставим слеш /, указываем конец. Если указать только одно значение этому свойству, второе по умолчанию станет auto и будет охватывать одну ячейку:

Скопировать кодCSS

.block {

grid-row: 1;

grid-column: 2/4;

}

Чтобы каждый раз не считать линии, им можно давать имена. Цифровые названия линий останутся доступными. Имена линиям указывают в квадратных скобках на этапе создания строк и столбцов:

Скопировать кодCSS

grid-template-rows: [aside-start] 300px [aside-end];

Чтобы задать элементу расположение, отсчитывая от линии с именем aside-start, нужно написать:

Скопировать кодCSS

grid-row: aside-start / 4;

Имя линии может быть любым, кроме ключевого слова span. Словом span указывают, какое количество строк или столбцов должен занимать элемент до или после какой-то линии. Например:

Скопировать кодCSS

.block {

grid-column-start: 2;

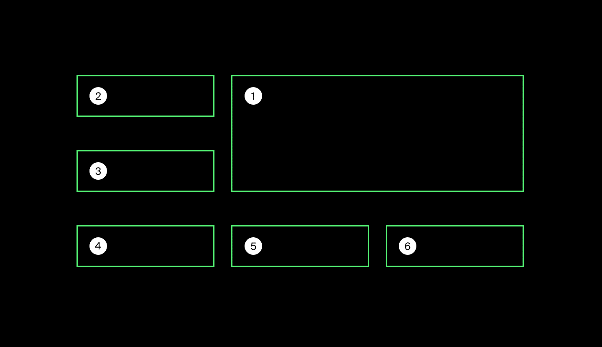
grid-column-end: span 2;

grid-row-start: span 2;

grid-row-end: 3;

}

Элемент займёт две строки, заканчиваясь во второй строке, и два столбца, начинаясь во второй колонке.



Демонстрация работы кода с ключевым словом span

Если не указывать начало элемента, по умолчанию он начнётся с той линии, на которой расположен изначально.

Печатать каждый раз **grid-column-start** и **grid-column-end** может быть немного утомительно. К счастью, есть краткая форма **grid-column**, которая принимает оба значения сразу через косую черту: **/**.

Например, **grid-column: 2 / 4** скажет grid-элементу начаться на второй вертикальной grid-линии и закончиться на четвёртой.

А теgерь попробуйте использовать сокращённое свойство **grid-row**. **Определяет позицию grid-элемента внутри строк: Grid-column-start /** **grid-column-end**

Например grid-row: 3/span3

Если вас также утомляет печатать **grid-column** и **grid-row**, есть сокращённое свойство и для этого. **grid-area** принимает 4 значения, разделенные косой чертой **/**: **grid-row-start**, **grid-column-start**, **grid-row-end** и **grid-column-end**.

Пример: **grid-area: 1 / 1 / 3 / 6;**.

Если grid-элементы не имеют конкретного расположения с **grid-area**, **grid-column**, **grid-row** и т.д., они автоматически размещаются, следуя порядку, написанному в коде. Мы можем изменить это с помощью свойства **order**, которое является одним из преимуществ CSS Grid Layout перед табличной разметкой.

Изначально все grid-элементы имеют **order**, равный **0**, но этому свойству можно присвоить любое положительное или отрицательное значение, примерно как у **z-index**.

**grid-template** — сокращённый вариант комбинации **grid-template-rows** и **grid-template-columns**.

Например, **grid-template: 50% 50% / 200px** создаст grid-сетку с двумя строками по 50% каждая и одним столбцом шириной 200 пикселей.

# Грид-области

В этом уроке начнётся настоящая магия.

Представьте, что для построения сетки больше не нужно запоминать имена линий и задавать каждому элементу расположение по вертикали и горизонтали, как в предыдущем задании. Представьте, шапку сайта, которая занимает три колонки. А контенту внутри мы можем просто указать, какую колонку ему занять: логотип поместить в левую, меню — в среднюю, контакты — в правую. С появлением гридов такое стало возможным.

**Грид-область** (англ. grid-area) — это прямоугольная область, которая состоит из одной или нескольких грид-ячеек.

Для того, чтобы разделить грид-контейнер со строками и столбцами на области, разработчики придумали свойство grid-template-areas. Оно задаётся самому контейнеру, в качестве значения — список с именами областей.

Возьмём контейнер с тремя столбцами и четырьмя строками. Поделим его на области, где хотим разместить шапку, подвал, блок с новостями, промо-блок и сайдбар — боковое меню.

Скопировать кодCSS

.container {

display: grid;

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

grid-template-rows: repeat(4, 1fr);

grid-template-areas:

"header header header"

"news news aside"

"promo promo aside"

". footer footer"

}

Каждую ячейку сетки необходимо заполнить, иначе свойство grid-template-areas не сработает. Имя области может быть любым. Мы советуем выбирать имя, которое лучше всего отражает содержимое. Если какую-то область нужно оставить пустой, вместо имени ставится точка или несколько точек.

Для того чтобы сетка обрела нужный вид, воспользуемся свойством grid-area. Оно задаётся грид-элементу и указывает на область, в которой он должен поместиться.

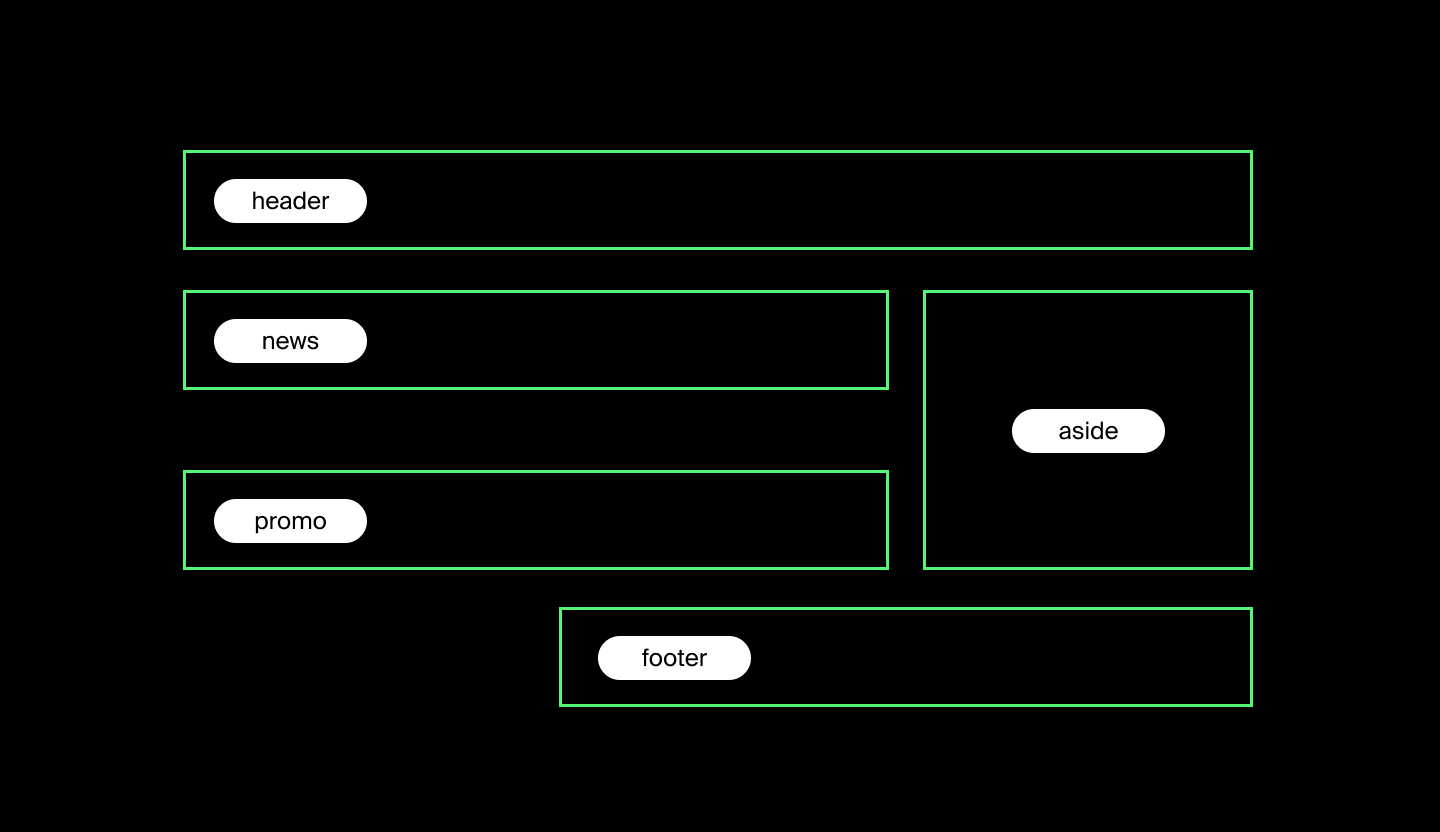
Скопировать кодCSS

.header {

grid-area: header;

}

Шапка займёт всю верхнюю строку. Останется распределить остальные элементы. В итоге у нас получится сетка:



Демонстрация работы свойства grid-template-areas

У этого способа построения сетки есть несколько нюансов. Грид-область обязательно должна быть прямоугольником. Также можно создать только одну область для каждого имени — разъединённые области создать не получится:

Скопировать кодCSS

*/\* Такой код работать не будет \*/*

grid-template-areas:

"header header header"

"news news aside"

"promo promo aside"

"footer news footer";

Но очень вероятно, что и это станет возможным.

# Заключение

Мы рассказали об основных возможностях гридов и почему не стоит бояться их использовать. Теперь вы знаете, что не нужно противопоставлять флексбоксы и гриды. Они призваны решать общую задачу, у каждой технологии есть сильные стороны, которые можно использовать вместе.

Знаний, которые вы получили, хватит, чтобы начать использовать гриды в проекте: строить сетки страниц и размещать в них содержимое. Но это только начало, в этой теме вы поняли лишь суть технологии. В следующей теме мы расскажем про новые свойства и тонкости работы с грид-элементами, которые помогут делать сайты адаптивными.

По традиции несколько полезных источников для закрепления материала:

[Интерактивная игра для быстрого запоминания свойств](https://cssgridgarden.com/#ru).

[Основные понятия Grid Layout](https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout).

[Документация Grid Layout](https://www.w3.org/TR/css-grid-1/) (на английском языке).

[Шпаргалка по свойствам гридов](https://css-tricks.com/snippets/css/complete-guide-grid/) (на английском языке).

# Grid Layout, часть 2. Введение

В первой части вы изучили основные принципы Grid Layout. Вы узнали, как устроены грид-сетки и их контейнеры, почему они удобны и чем гриды отличаются от флексбоксов. Этих основ достаточно, чтобы уже использовать гриды в проектах.

Во второй части вы научитесь работать с грид-элементами внутри контейнера: двигать, выравнивать и накладывать друг на друга. Вы больше узнаете о неявных треках и задании размеров. Именно они отвечают за «резиновость» сетки.

После этой темы вы поймёте, почему гриды так удобны для адаптивности. Вы научитесь делать отзывчивый сайт проще и быстрее — без тонны кода и нудных расчётов. Эти знания пригодятся в проектной работе в конце спринта.

**Размеры неявных строк и колонок**

В первой части курса по гридам мы уже говорили о неявных треках. Они появляются, если элементов в контейнере больше, чем ячеек в явно заданных строках и столбцах свойствами grid-template-rows и grid-template-columns. Это может произойти при добавлении новых картинок в галерею или блоков на сайт. Предыдущая сетка из-за этого не сломается, но внешний вид страницы пострадает.

Всё дело в различиях размеров треков. Неявные треки по умолчанию auto и отталкиваются от размеров содержимого в них, поэтому могут быть больше или меньше, чем явные.



Элементы, которые не поместились в явные ячейки, расположись в новых неявных строках

Для таких случаев разработчики гридов придумали свойства управления размерами: grid-auto-rows для неявных строк и grid-auto-columns для столбцов. Их задают грид-контейнеру так же, как параметры явных треков.

Рассмотрим простую сетку с 11 элементами:

Скопировать кодHTML

<div class="container">

<div class="block">1</div>

<div class="block">2</div>

<div class="block">3</div>

<div class="block">4</div>

<div class="block">5</div>

<div class="block">6</div>

<div class="block">7</div>

<div class="block">8</div>

<div class="block">9</div>

<div class="block">10</div>

<div class="block">11</div>

</div>

Зададим ей явные строки и колонки:

Скопировать кодCSS

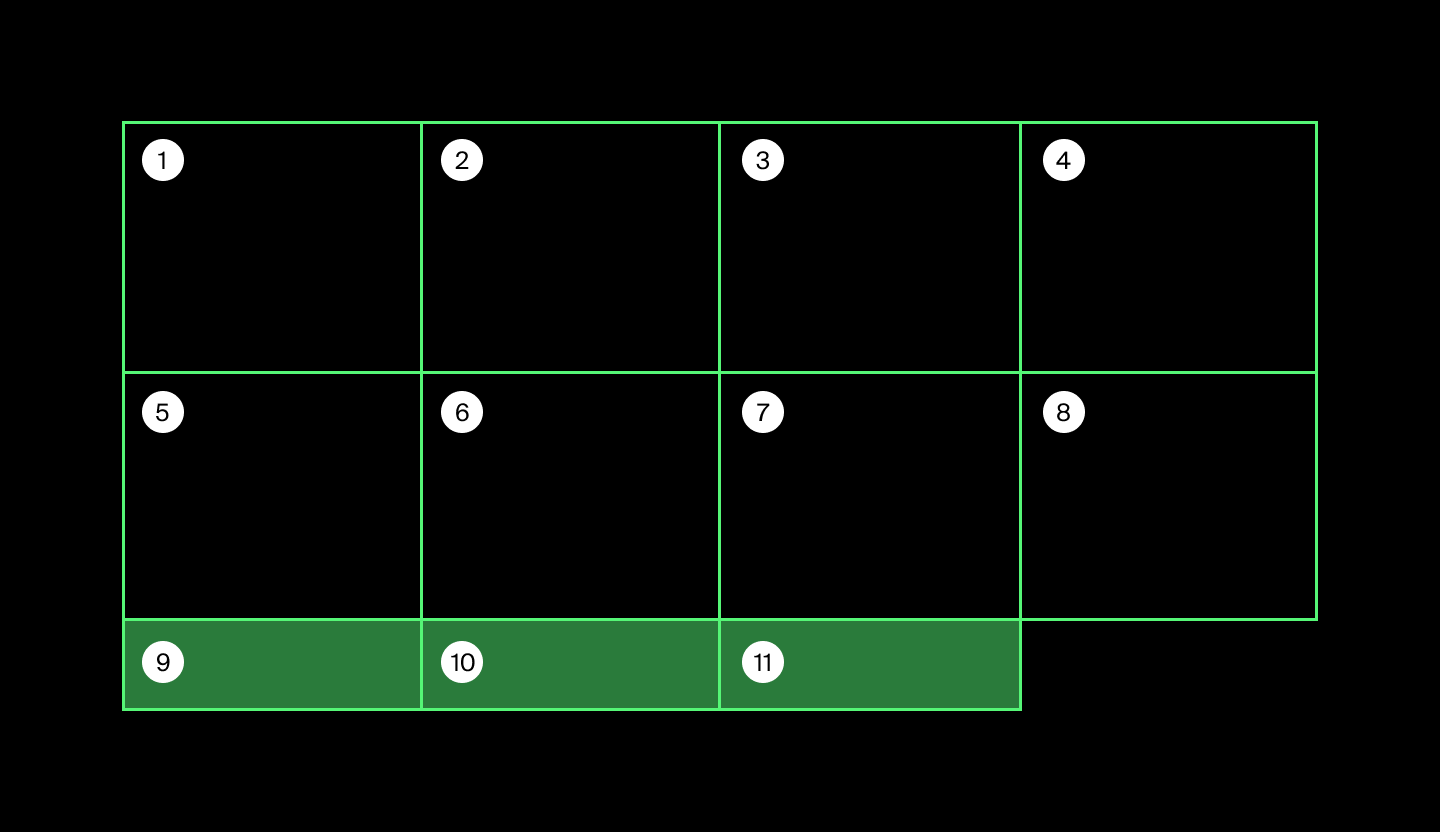
.container {

display: grid;

grid-template-rows: repeat(2, 100px);

grid-template-columns: repeat(4, 100px);

}



Элементы, которые не поместились в сетке, выбиваются из общих размеров

На помощь приходит свойство grid-auto-rows. Добавим его к стилям:

Скопировать кодCSS

.container {

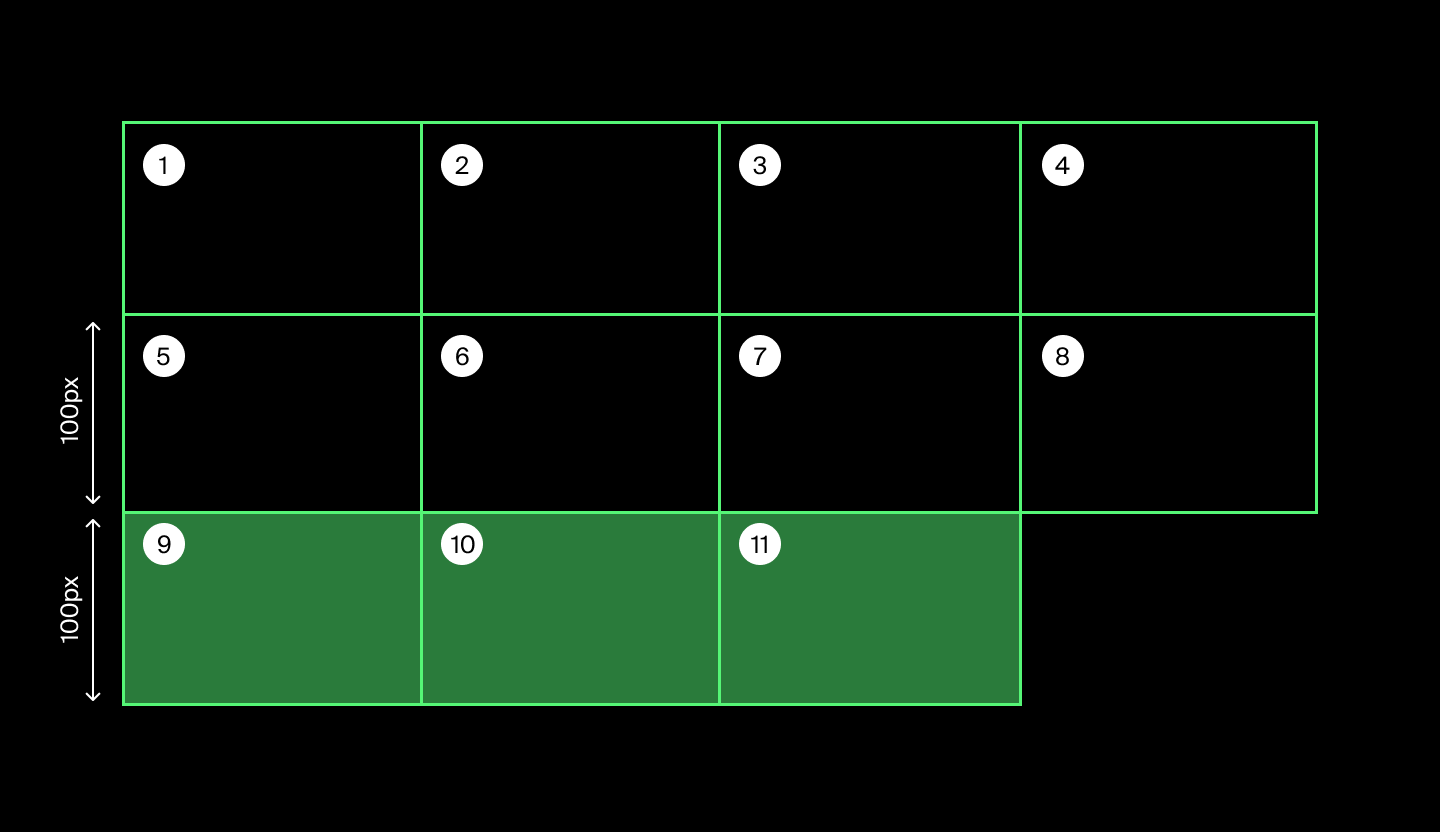
display: grid;

grid-template-rows: repeat(2, 100px);

grid-template-columns: repeat(4, 100px);

grid-auto-rows: 100px;

}



Теперь все неявные треки будут высотой в 100px

Если нужна разная высота неявных строк, свойству grid-auto-rows задают несколько значений — по аналогии с grid-template-rows.

Скопировать кодCSS

.container {

display: grid;

grid-template-rows: 100px;

grid-template-columns: repeat(4, 100px);

grid-auto-rows: 100px 300px;

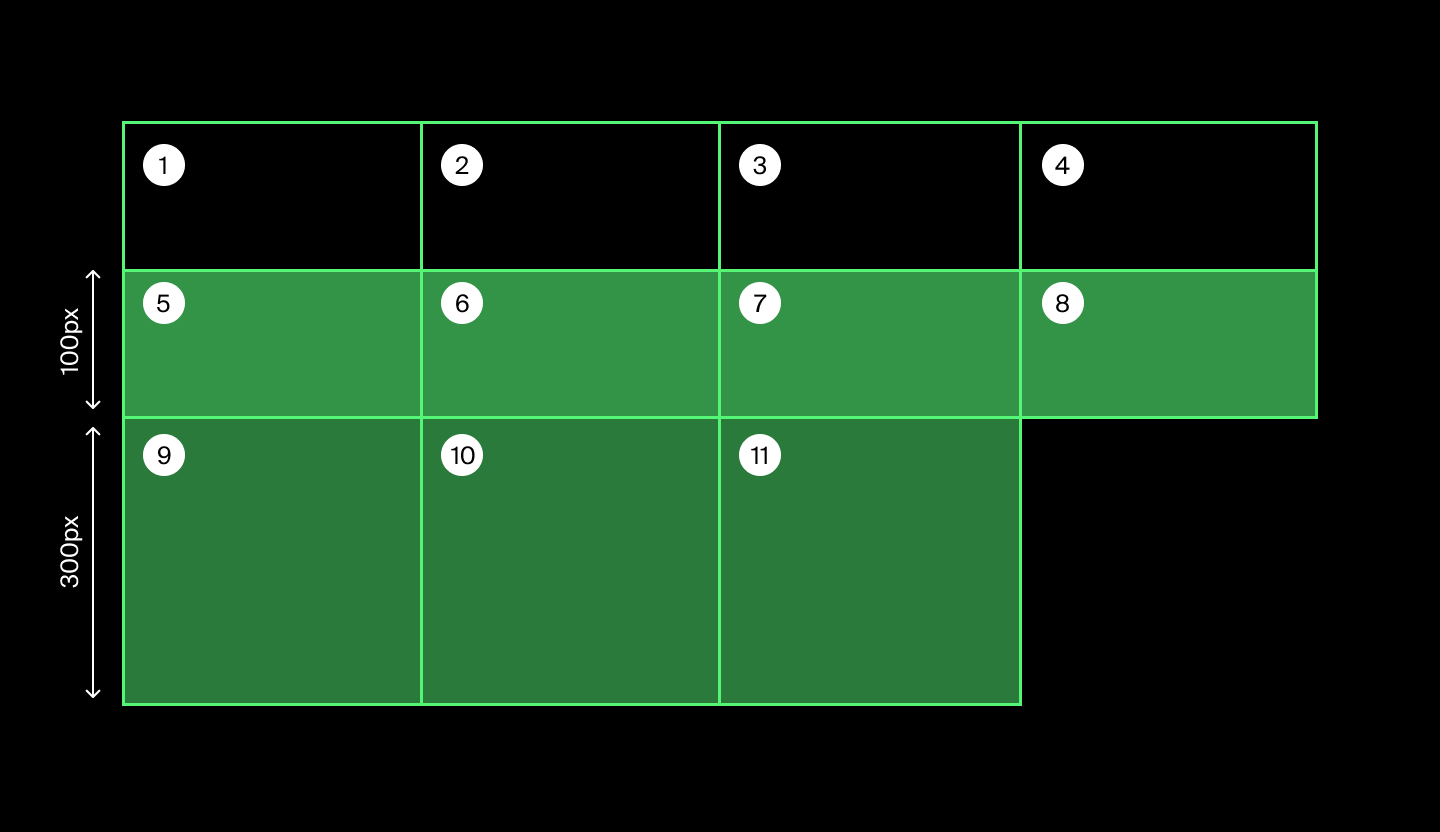
}

*/\* высота первого трека 100px,*

*а второго — 300px,*

*высота третьего трека снова — 100px,*

*а четвёртого — 300px и т.д. \*/*



Первый неявный трек стал 100px в высоту, а второй — 300px

По умолчанию дополнительные треки:

* добавляются сверху вниз (↓), образуя новые строки,
* и заполняются слева направо (→), образуя столбцы.

Свойство grid-auto-columns нужно, чтобы помимо размеров управлять поведением неявных треков. Это свойство мы рассмотрим в следующем уроке, а пока перейдём к практике.

# Свойство grid-auto-flow

В прошлом уроке вы увидели, что неявные треки добавляются снизу сразу после явных. Этим поведением можно управлять свойством grid-auto-flow. По умолчанию его значение row. Чтобы вместо строк добавлялись неявные столбцы, нужно его поменять на column:

Скопировать кодCSS

.container {

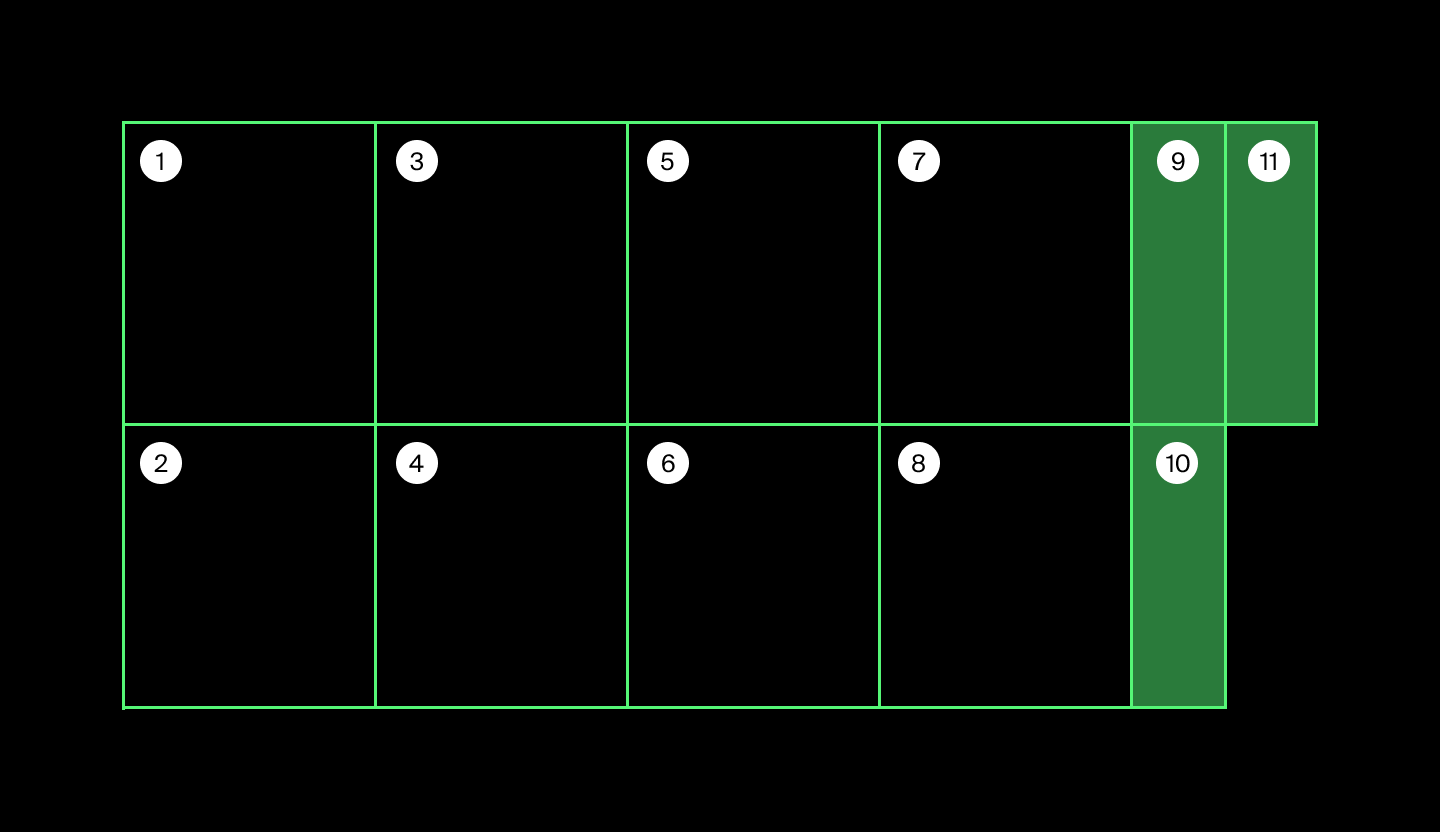
display: grid;

grid-template-rows: repeat(2, 100px);

grid-template-columns: repeat(4, 100px);

grid-auto-flow: column; */\* теперь будут добавляться неявные столбцы \*/*

}



Неявные треки теперь добавляются слева направо

Теперь столбцы будут заполняться сверху вниз и при необходимости добавляться новые.

Свойство grid-auto-columns управляет размерами этих треков:

Скопировать кодCSS

.container {

display: grid;

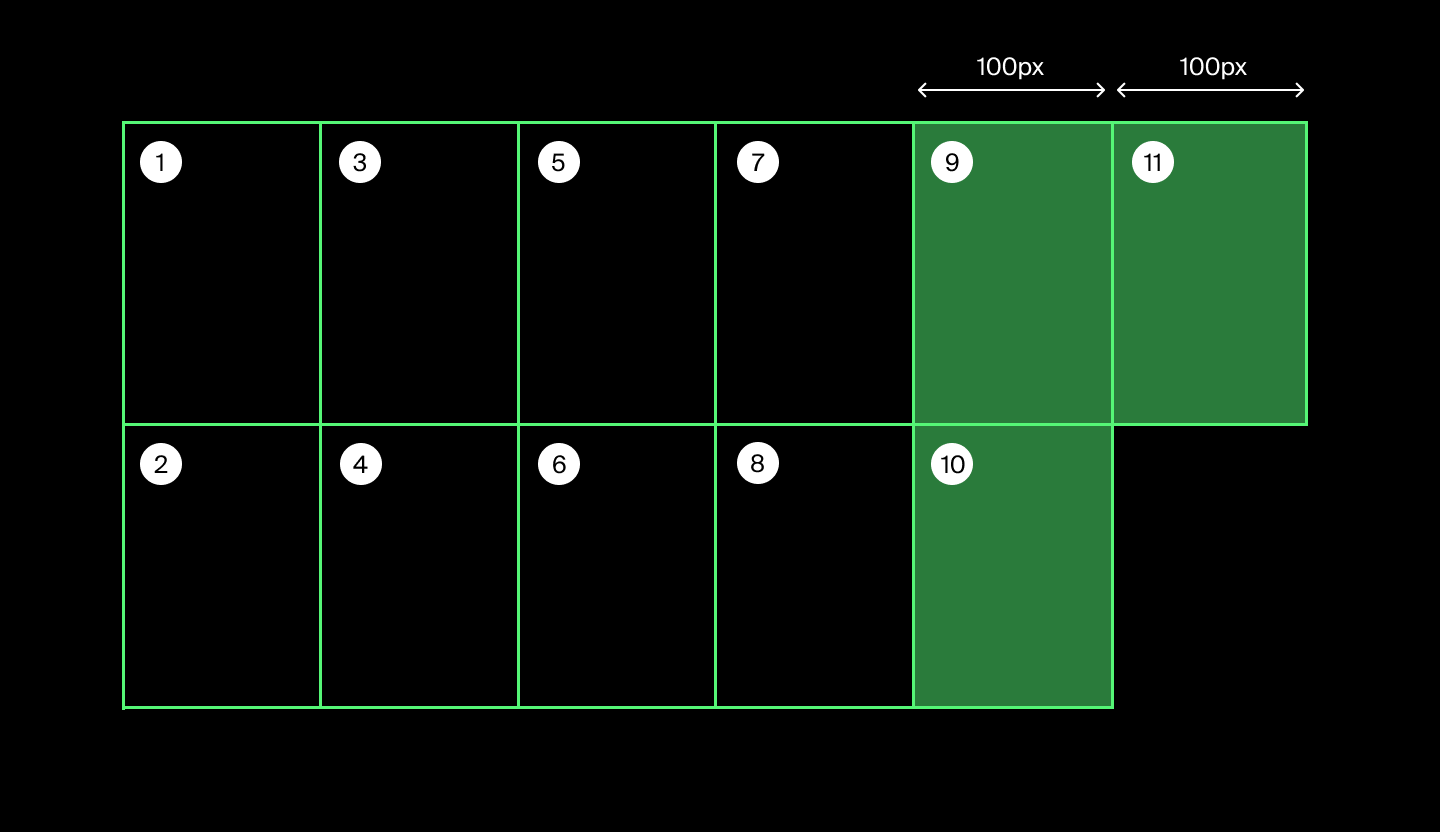
grid-template-rows: repeat(2, 100px);

grid-template-columns: repeat(4, 100px);

grid-auto-flow: column;

grid-auto-columns: 100px; */\* ширина неявных столбцов 100px \*/*

}



Теперь все ячейки одинаковой ширины

У grid-auto-flow есть необычное значение dense. Представьте, что вы укладываете вещи в поездку оптимально компактно, чтобы они уместились в чемодан с минимумом пустого места. То же произойдёт с элементами сетки, если использовать dense.

Применим это для простой сетки из шапки, основного контента и подвала:

Скопировать кодCSS

.container {

display: grid;

width: 500px;

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

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

grid-auto-flow: dense;

}



Пока новых элементов нет, ничего не происходит

Как только появятся элементы поменьше, они займут пробелы в сетке.



Новый блок занял свободное место

Значение dense помогает заполнять пробелы в сетке, которые появляются из-за разных размеров элементов. Так сетка остаётся компактной, и это круто. Не используйте dense в местах, где порядок элементов важен. Иначе меню сайта или подвал окажется не там, где вы ожидаете его увидеть.

# Функции minmax() и fit-content()

Мы много говорим о размерах колонок, строк и элементов. Именно подбор размеров и единиц измерений — главная фишка гридов. Она и позволяет создавать адаптивные сайты.

Фракция удобна тем, что треки сами пересчитывают свою долю доступного свободного пространства. Но использовать фракцию не всегда выгодно. Расскажем и о других способах задания размеров.

Контролировать сжатие и растяжение элементов удобно. Когда вы верстаете закреплённые боковые панели, хорошо, если они не больше или не меньше какой-нибудь определённой величины — в зависимости от разрешения устройства и контента.

## Минимальный и максимальный размер. Функция minmax()

Найти золотую середину помогает minmax(). Эта функция имеет два параметра: минимальный размер min и максимальный max. Трек или элемент с таким размером растягивается и сжимается в пределах этих значений.

Скопировать кодCSS

.container {

display: grid;

grid-template-columns: 1fr 1fr minmax(300px, 1fr);

*/\* третий столбец не может быть меньше 300px и больше одной фракции \*/*

}

Контейнер растянется на всё свободное пространство, но не ужмётся меньше 300px.

Значения параметров min и max можно задавать в пикселях, процентах, фракциях. Но есть два специфических значения — max-content и min-content. Их можно использовать самостоятельно или в рамках функции minmax().

### Значение max-content

Значение max-content сообщает строке или столбцу: «Будь такой ширины, чтобы вмещать содержимое без переносов и максимально компактно, без свободного пространства по краям». Это удобно для подписей к картинкам. Значение max-content размещает подпись на одной строке без переносов.

Скопировать кодCSS

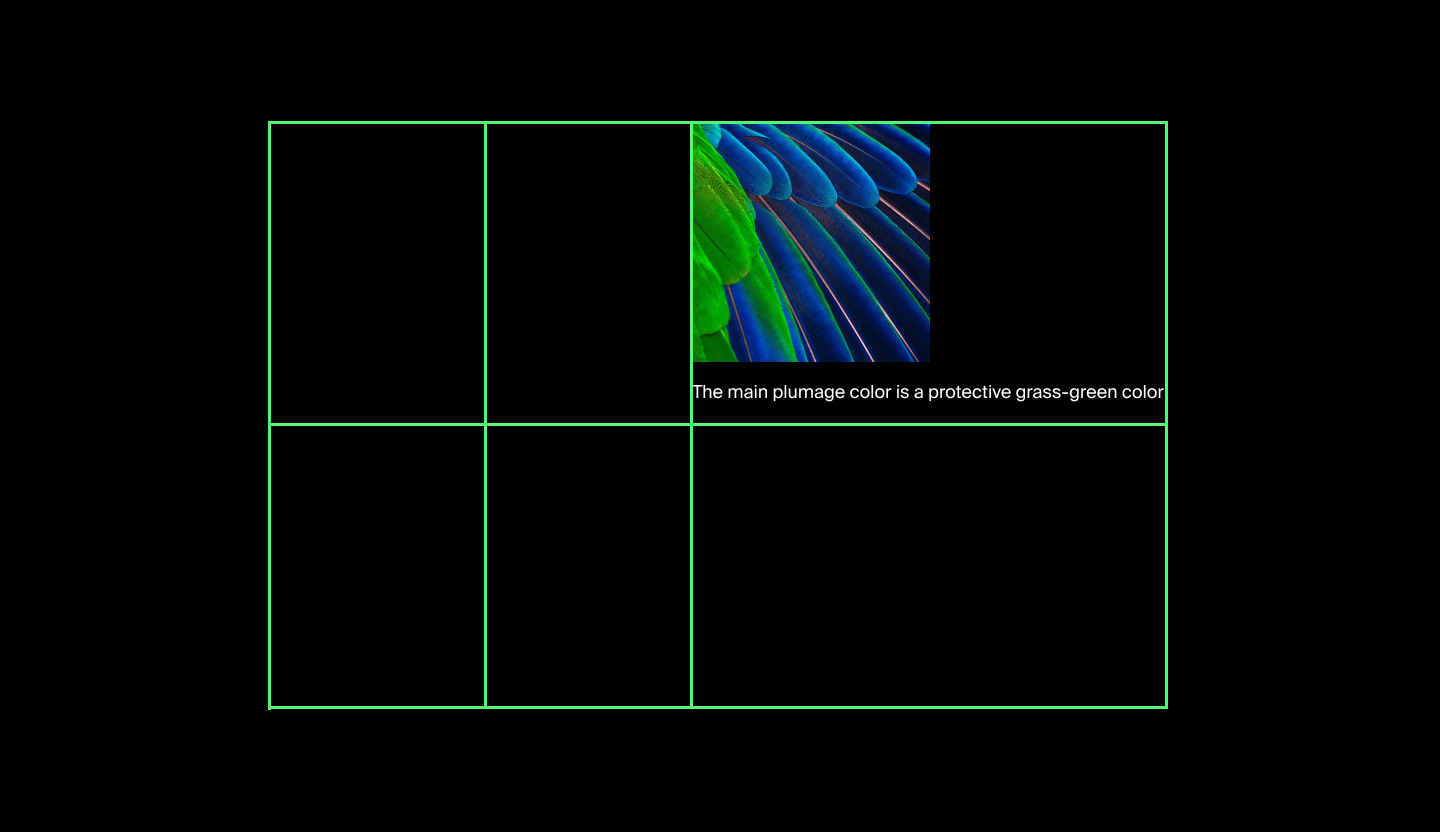
.container {

display: grid;

grid-template-rows: 1fr 1fr;

grid-template-columns: 1fr 1fr max-content;

}



Подпись к картинке осталась цельной и не перенеслась

Но оставлять контент без переносов рискованно. Если мы зададим значение max-content столбцу со всеми обещаниями «начать с понедельника», представьте, какой ширины получится этот столбец!

### Значение min-content

Для длинных списков подходит свойство min-content. Оно стремится перенести всё, что переносится, чтобы содержимое ячейки занимало как можно меньше места. Значение min-content сообщает треку: «Ориентируйся на самый маленький — картинку или слово.

Скопировать кодCSS

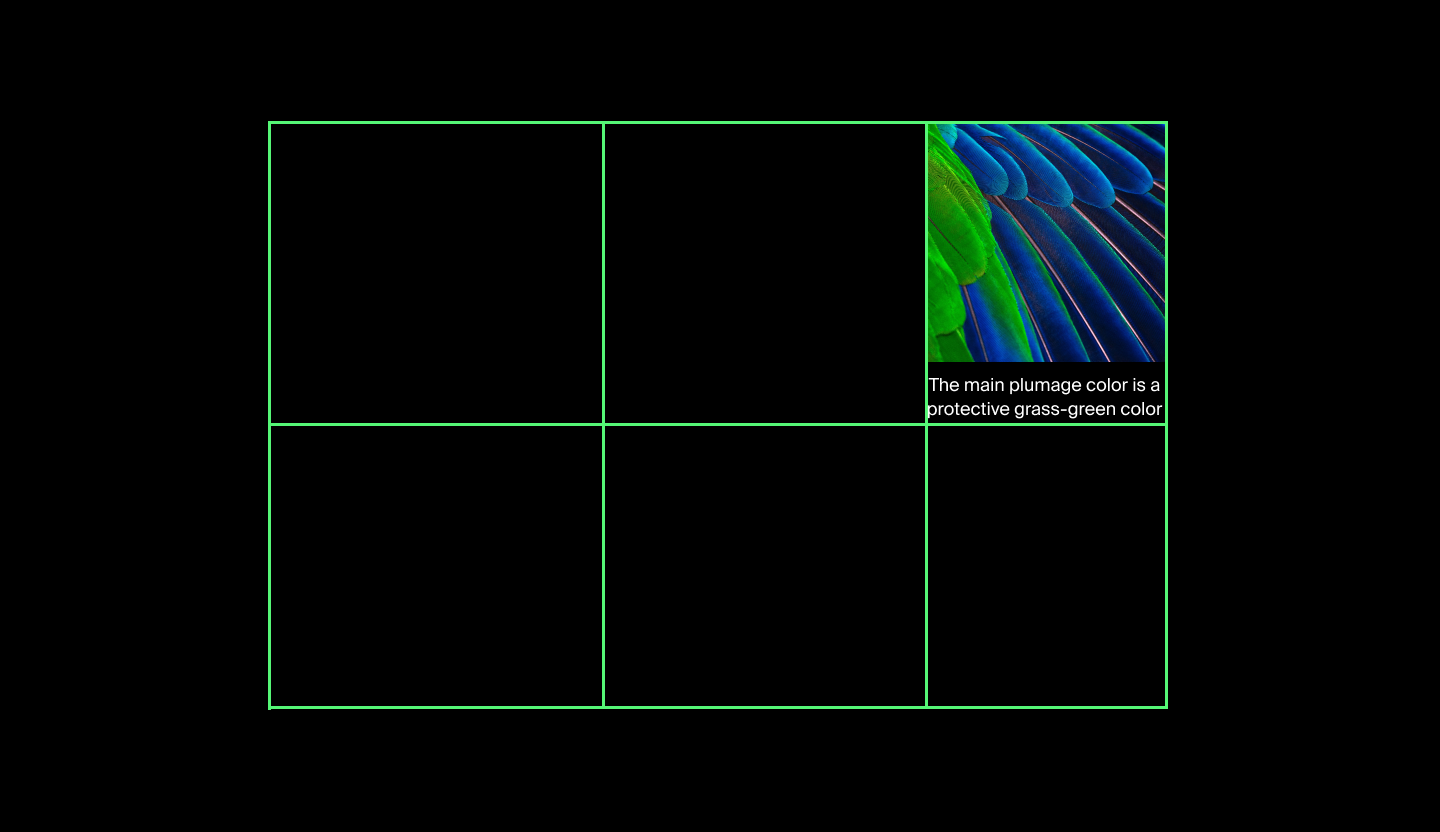
.container {

display: grid;

grid-template-rows: 1fr 1fr;

grid-template-columns: 1fr 1fr min-content;

}



Размер третьей колонки ужался под контент и стал равен ширине картинки

Но по-настоящему удобство этих значений вы оцените в команде с minmax.

Скопировать кодCSS

grid-template-rows: minmax(*/\*минимальное значение\*/*, */\*максимальное значение\*/*);

### Функция minmax() с параметрами min-content и max-content

Трек со значением minmax(min-content, max-content) будет ориентироваться на контент: не ужмётся меньше содержимого, но и не займёт лишнее место, когда контента мало.

Скопировать кодCSS

.container {

display: grid;

grid-template-columns: 1fr 1fr minmax(min-content, max-content);

}

Правая колонка не растянется и не сожмётся сильнее, чем это необходимо.

Функцией minmax(min-content, max-content) мы ограничили размеры трека:

задали минимальное значение «сузиться до размеров самого большого неделимого элемента: слова или картинки»;

и максимальное значение «вмещать весь контент без переносов и пустых полей вокруг элементов».

Сочетание функции minmax() со свойствами min-content и max-content удобно, потому что оно определяет размер трека как «не больше и не меньше, чем требуется».

## Ограничение по размеру. Функция fit-content()

Трек с fit-content ориентируется на контент, но всегда стремится к указанному в нём значению. Колонка занимает ширину под размер своего контента. Но она не сможет превзойти значение, указанное в fit-content.

Звучит сложно, приведём аналогию.

Функция fit-content() — это своего рода пропуск на аттракционы по весу. Когда вес посетителя меньше заявленного, он может попасть на аттракцион. Если же вес превышает заданный предел — доступ закрыт, фейс-контроль на входе будет непреклонен.

Вот как это работает:

Скопировать кодCSS

.container {

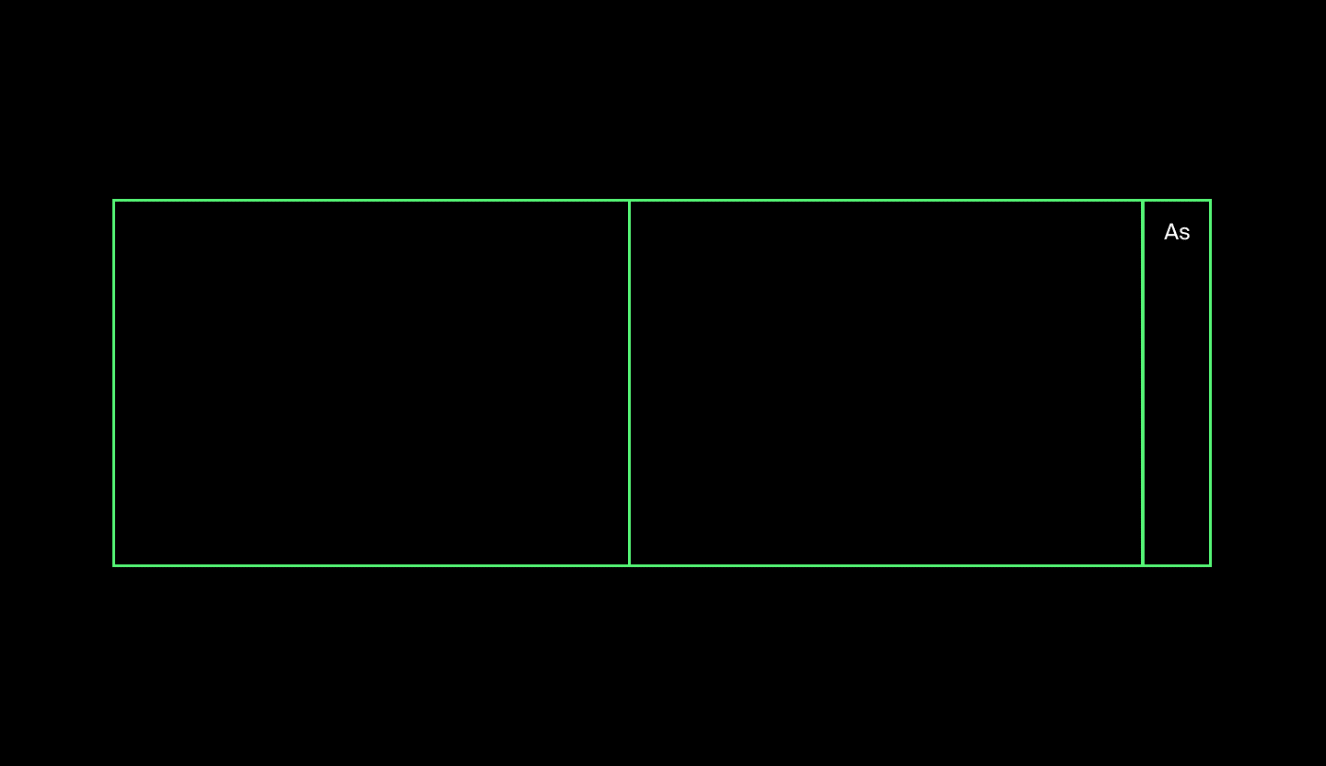
display: grid;

grid-template-rows: 1fr 1fr 1fr;

grid-template-columns: 1fr 1fr fit-content(200px);

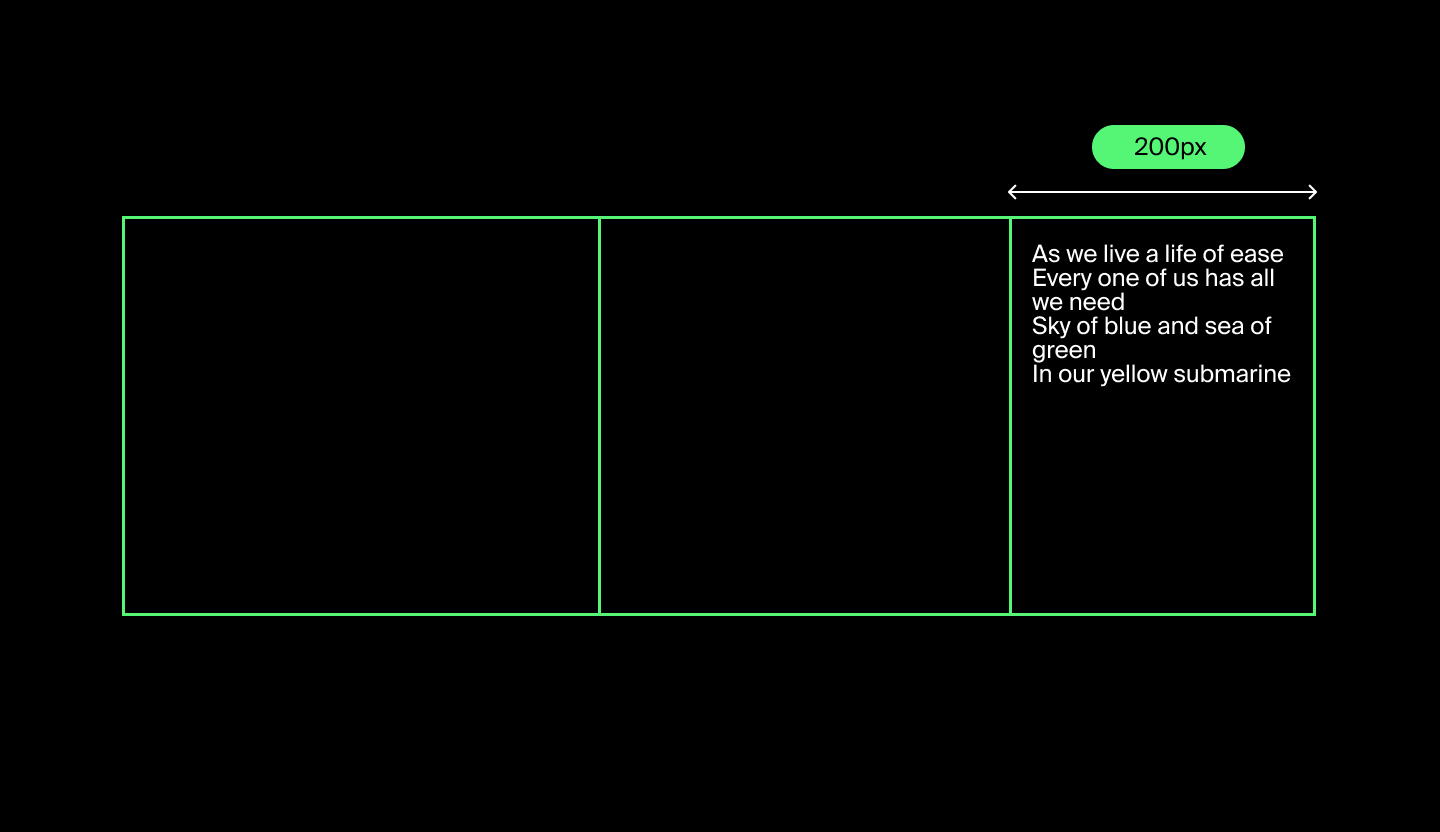
}

Мы установили третьей колонке предельный размер в 200px. Шире этого значения колонка не станет.



Здесь контента меньше 200px, поэтому колонка проходит фейс-контроль и ужимается по своим размерам

Добавим контента, чтобы колонка растянулась:



Тут контента намного больше. Фейс-контроль не пройден, придётся ужиматься до 200px

Ограничивать размер трека удобнее всего одной из двух функций: minmax() и fit-content().

Функция minmax() работает особенно хорошо со свойствами min-content и max-content. Трек со значением minmax(min-content, max-content) не ужмётся меньше своего контента, но и не займёт больше места, чем контенту необходимо.

Функция fit-content() задаёт только верхнюю границу размера трека. Трек со значением fit-content(200px) ужмётся под размер своего контента, но не сможет растянуться больше 200px.

page {

display: grid;

font-weight: 400;

font-size: 20px;

grid-template-columns: minmax(min-content, 1fr) 3fr 3fr minmax(245px, 1fr);

/\*первой колонке (navigation) дает размер не меньше контента, но не более одной фрацции – для боковой панели навигации – самое то\*/   
/\*последней колонке (banner) дает размер не меньше 245, но не более одной фрацции – остальные свойства ниже, для селектора\*/

grid-template-rows: 9em auto 9em;

grid-template-areas:

"header header header header"

"navigation main main banner"

"footer footer footer footer";

font-family: Roboto, Arial, sans-serif;

}

.content {

grid-area: main;

background: #fffffb;

display: grid;

grid-template-columns: repeat(2, minmax(min-content, max-content));

/\*содержиме (картинка+текст) красиво укладываются в grid-контейнер, а текст переносится в зависимости от размера окна\*/

gap: 1.7em;

padding: 0 1em 6em 0;

}

.banners\_\_list {

list-style-type: none;

display: grid;

grid-template-columns: repeat(2, fit-content(70%));

/\*содержиме банера с бОльшим количеством текста занимает больше пространства, но не ужимает другие блоки сильнее, чем задано в fit-content – 70% \*/

gap: 10px;

margin: 0;

}

Функция fit-content() задаёт только верхнюю границу размера трека.

# Параметры auto-fill и auto-fit

Теперь вы знаете, как регулировать размеры треков. Следующий шаг — управлять их количеством.

Разработчикам не всегда известно количество колонок — они просто хотят уместить столько, сколько получится. Это типично для каталогов товаров. Так у Андрея на ноуте помещается 6 карточек, а у Саши на моноблоке — уже 8. А для дизайна иногда требуется просто перенести карточки на новую строку.

Для этого существуют значения auto-fill и auto-fit, которые используются в связке с уже знакомым вам repeat().

Скопировать кодCSS

grid-template-columns: repeat(*/\* auto-fill или auto-fit \*/*, */\* размер колонки \*/*);

## Значение auto-fill

Рассмотрим контейнер с шестью элементами:

Скопировать кодHTML

<div class="container">

<div class="block">1</div>

<div class="block">2</div>

<div class="block">3</div>

<div class="block">4</div>

<div class="block">5</div>

<div class="block">6</div>

</div>

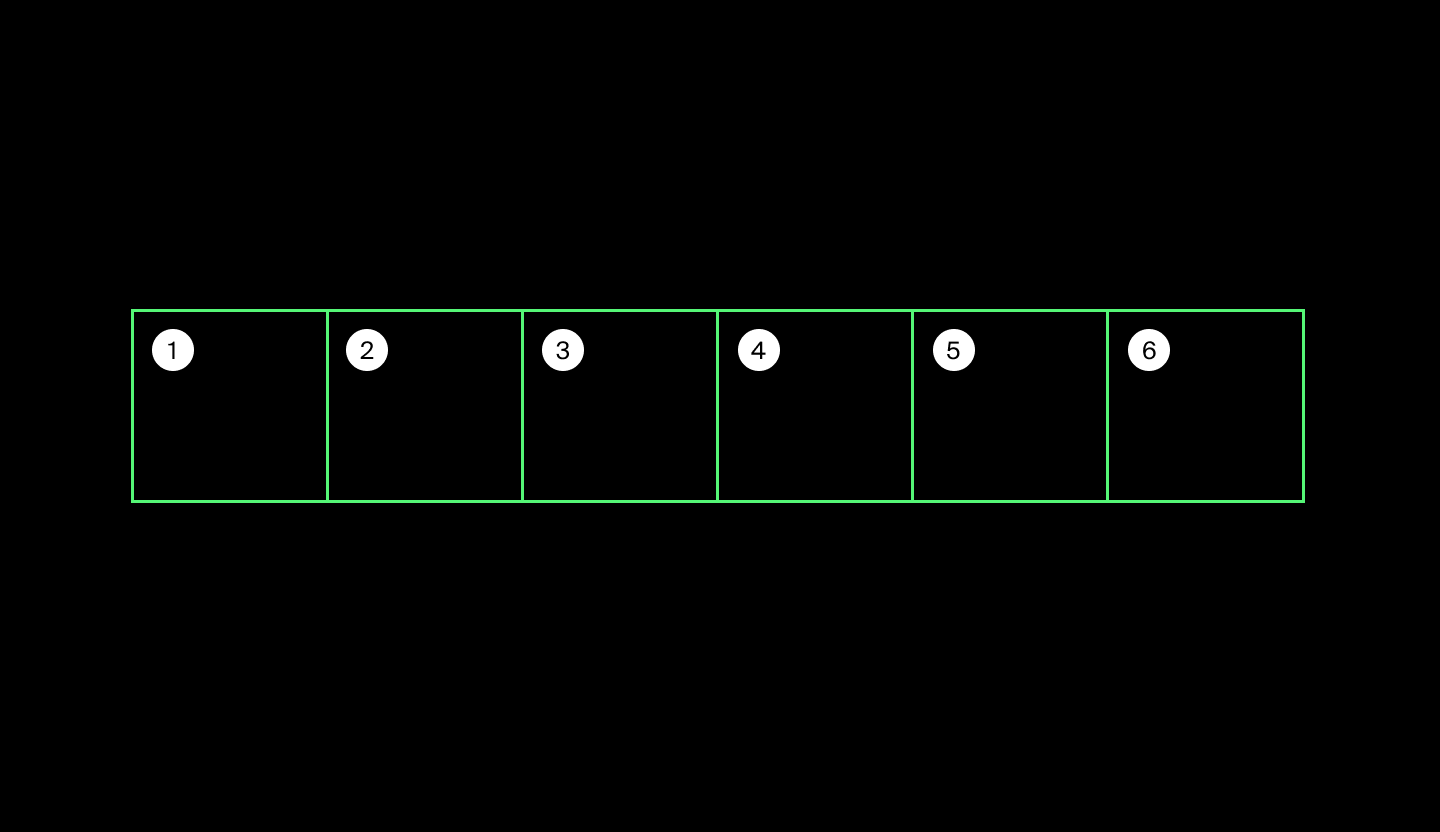
Скопировать кодCSS

.container {

display: grid;

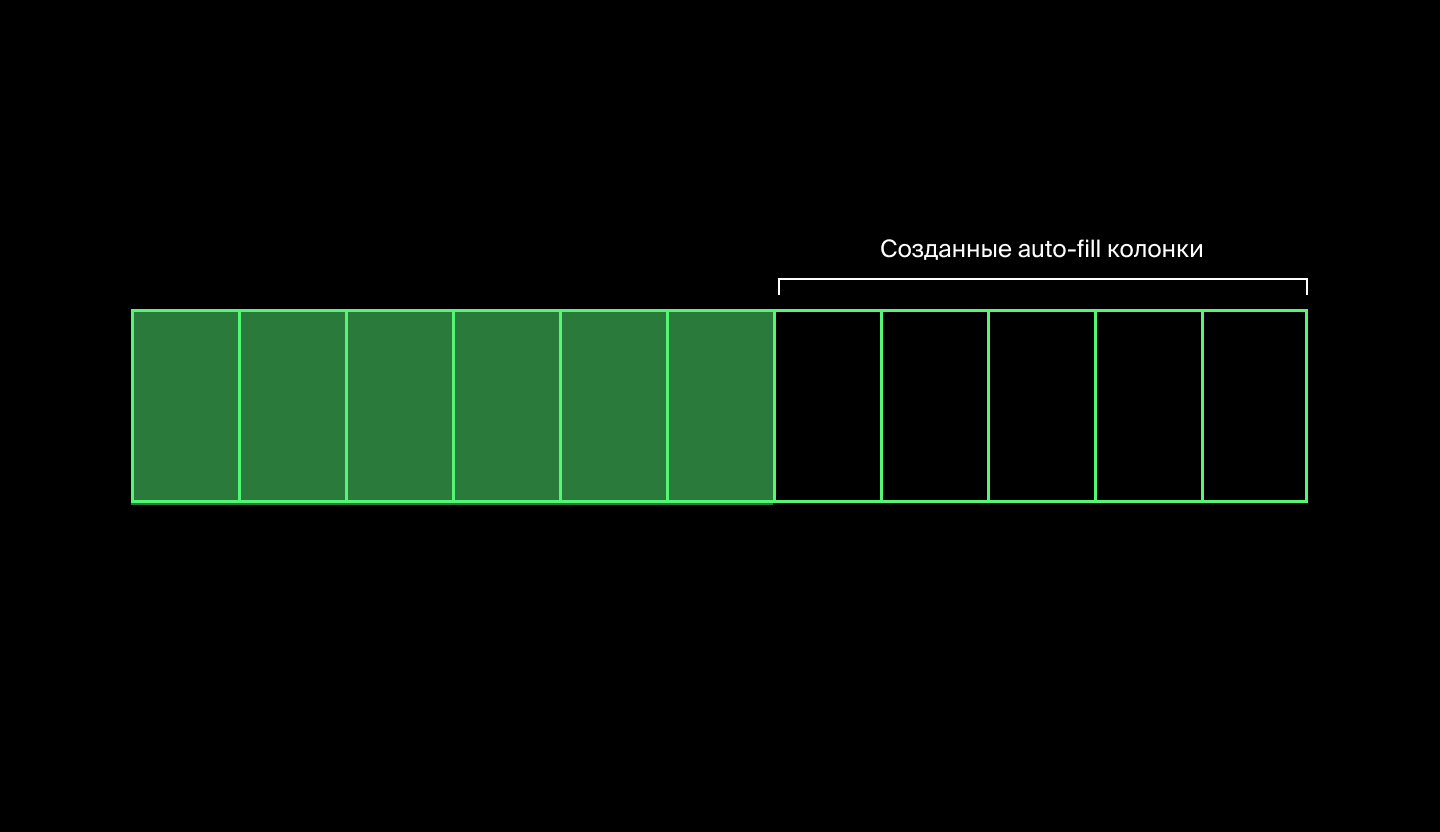
grid-template-columns: repeat(auto-fill, 100px);

}



Сетка, полученная командой repeat(auto-fill, 100px)

Глаза говорят нам, что колонок 6, но это не так. Дело в том, что свойство auto-fill стремится заполнить колонками всё доступное пространство. Когда элементы заканчиваются, auto-fill создаёт пустые колонки. Их можно увидеть через инспектор в браузере, но особенно они заметны на большом мониторе:



Так auto-fill создаёт пустые колонки, чтобы заполнить всё доступное пространство

## Значение auto-fit

Значение auto-fit тоже заполняет всё доступное пространство колонками, но в отличие от auto-fill схлопывает пустые и отдаёт больше места под заполненные:

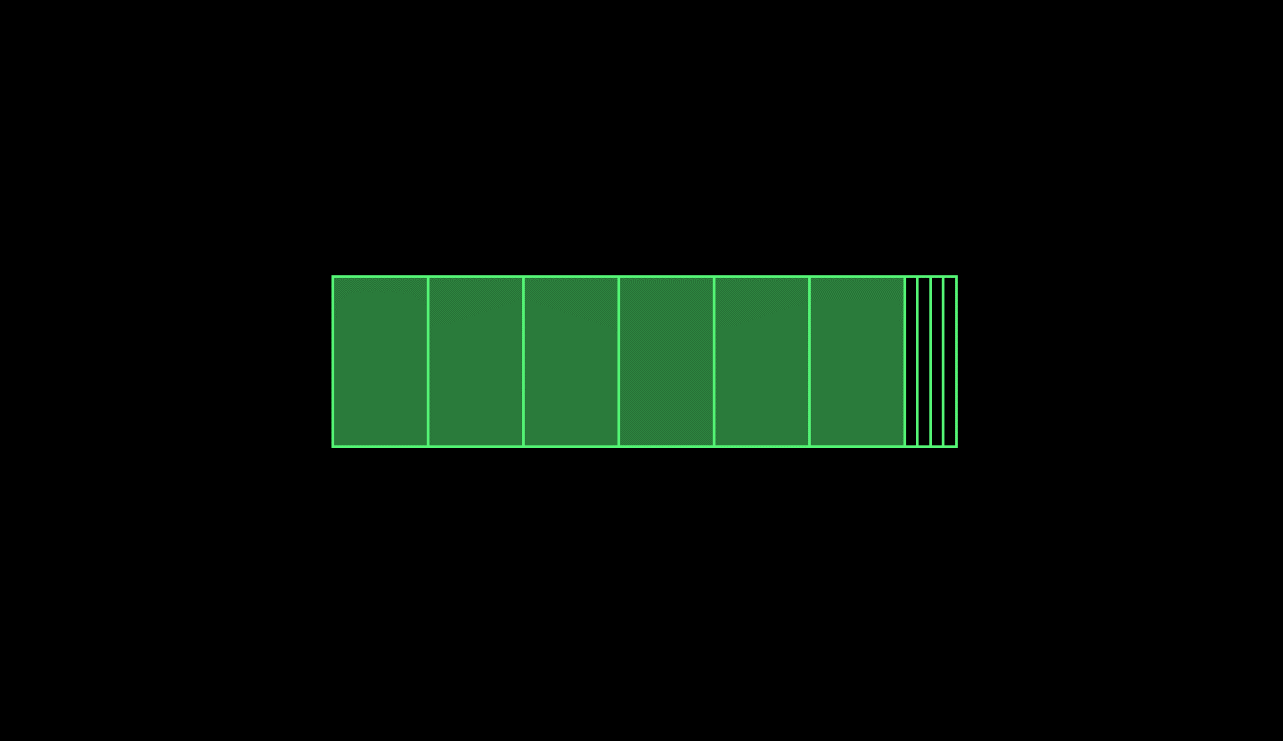
Скопировать кодCSS

.container {

display: grid;

grid-template-columns: repeat(auto-fit, 100px);

}



Со свойством repeat(auto-fit, 100px) неявные треки схлопнулись

При этом пустые колонки всё ещё существуют, просто с нулевым размером.

Вместо пикселей значения в auto-fill и auto-fit можно задавать во фракциях — тогда получится по-настоящему резиновая сетка.

Для создания такой сетки установите функцией minmax() минимальную и максимальную ширину колонок:

Скопировать кодCSS

.container {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));

}

Так auto-fit и minmax создают адаптивный каркас сайта.

Так колонки не разорвёт от контента: нет жёстких границ, в которые он обязан уместиться.

Значения auto-fill и auto-fit имеют схожее поведение. Разницу можно увидеть через функцию minmax() при изменении ширины окна браузера.

# Выравнивание содержимого строк и колонок

Вы уже выравнивали элементы во флексбоксах. В гридах разработчики не стали изобретать велосипед, поэтому работают те же самые свойства.

Свойство justify-items выравнивает элементы по горизонтали, т. е. по строкам. Во флексбоксах такого нет, зато есть justify-content.

По умолчанию у justify-items значение stretch. Его можно заменить на start, end или center. Во флексбоксах вы их уже встречали, только с приставкой flex-.

Свойство align-items выравнивает элементы по вертикали, т. е. по столбцам. Значения те же, что и у justify-items. Попробуйте в виджете ниже и посмотрите, как это работает:

Свойства justify-content и align-content помогут выровнять всю сетку. Кроме значений stretch, start, end и center у них есть space-around, space-between и space-evenly.

# Выравнивание элементов

Гриды позволяют выравнивать и отдельные элементы: свойством justify-self по горизонтальной оси и align-self по вертикальной. Похожие свойства есть во флексбоксах.

Выравниванием вы можете добиться эффекта наложения элементов друг на друга. В следующем уроке покажем, как.

# Наложение элементов

Наложение элементов помогает создавать декоративные эффекты. Это умение полезно, когда нужно переместить блоки, не трогая разметку.

Для наложения элементов требуются координаты — грид-области, номера или имена линий. Если задать группе элементов одинаковые координаты, они перекроют друг друга.

Посмотрим на примере.

На сайте есть грид-контейнер 3х3 и два элемента в сетке — шапка сайта и блок с навигацией. Менеджеры хотят провести A/B-тест: нужно переместить навигацию в шапку сайта. Менять разметку нежелательно: это что-то сломает в стилях или гипотеза не подтвердится. Тогда придётся с болью возвращать всё в исходное состояние. Вы должны придумать другой способ наложить их друг на друга.

Тут помогут грид-координаты:

Скопировать кодCSS

.header {

grid-column: 1/4;

grid-row: 1/2;

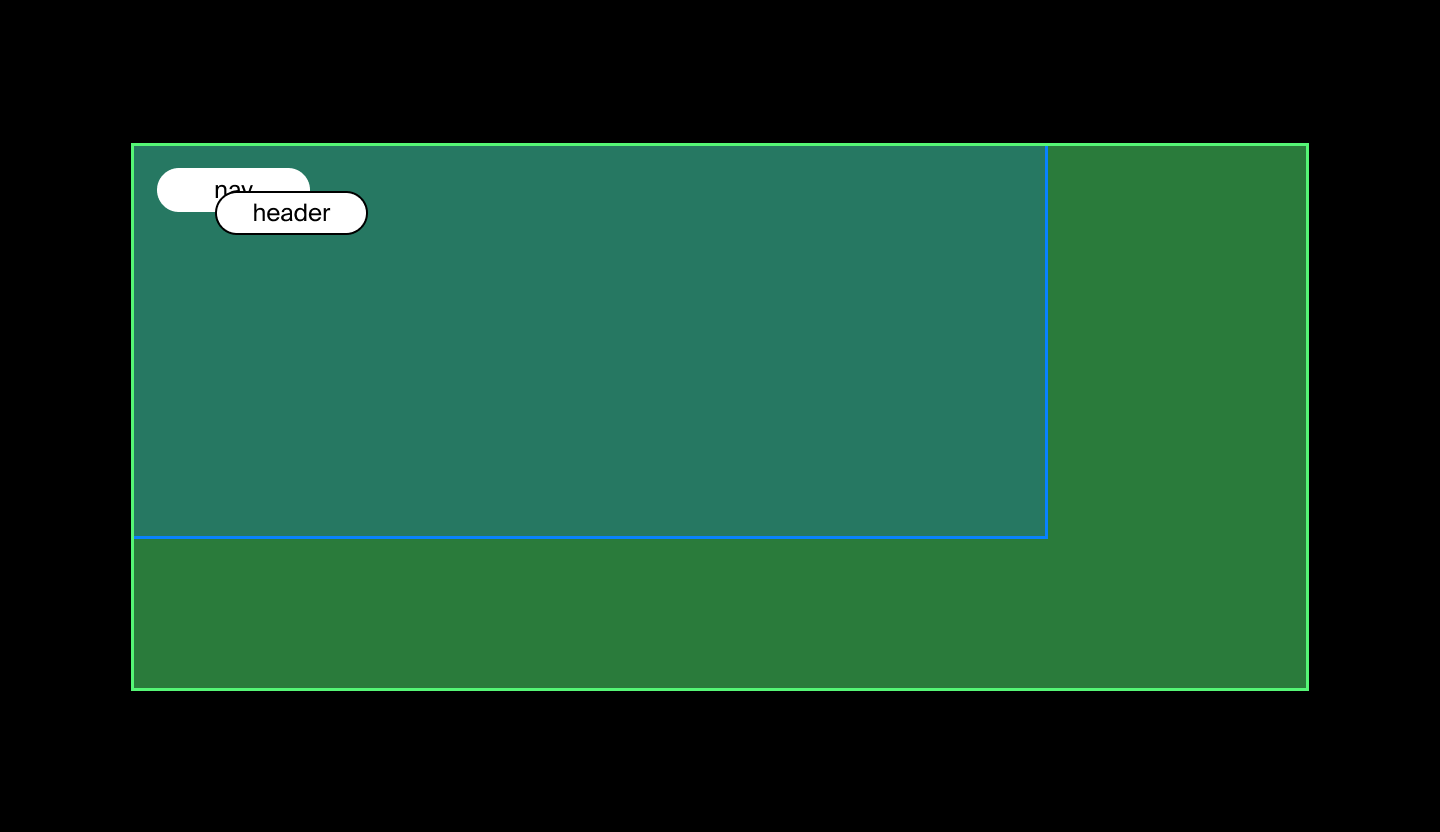
}

.nav {

grid-column: 1/3;

grid-row: 1/2;

}



Значения grid-column и grid-row пересекаются, накладывая элементы друг на друга

Теперь шапка сайта перекрывает и блокирует навигацию. Исправить это позволяет свойство z-index — оно работает и в гридах. Чем больше значение z-index, тем «выше» располагается элемент:

Скопировать кодCSS

.header {

grid-column: 1/4;

grid-row: 1/2;

}

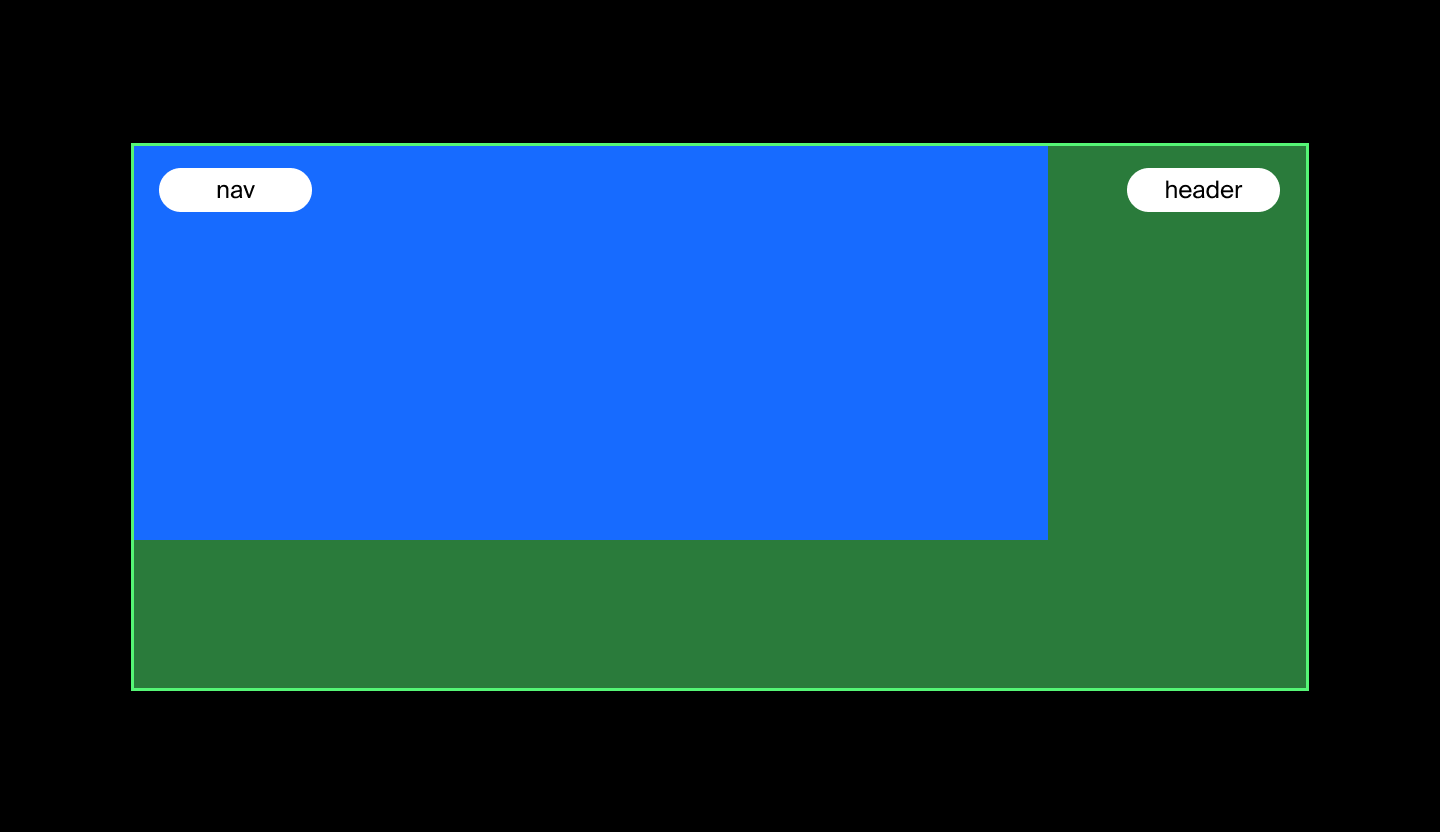
.nav {

grid-column: 1/3;

grid-row: 1/2;

z-index: 1;

}



Свойство z-index позволило расположить навигацию поверх шапки

Осталось выровнять навигацию уже знакомым свойством align-self:

Скопировать кодCSS

.header {

grid-column: 1/4;

grid-row: 1/2;

}

.nav {

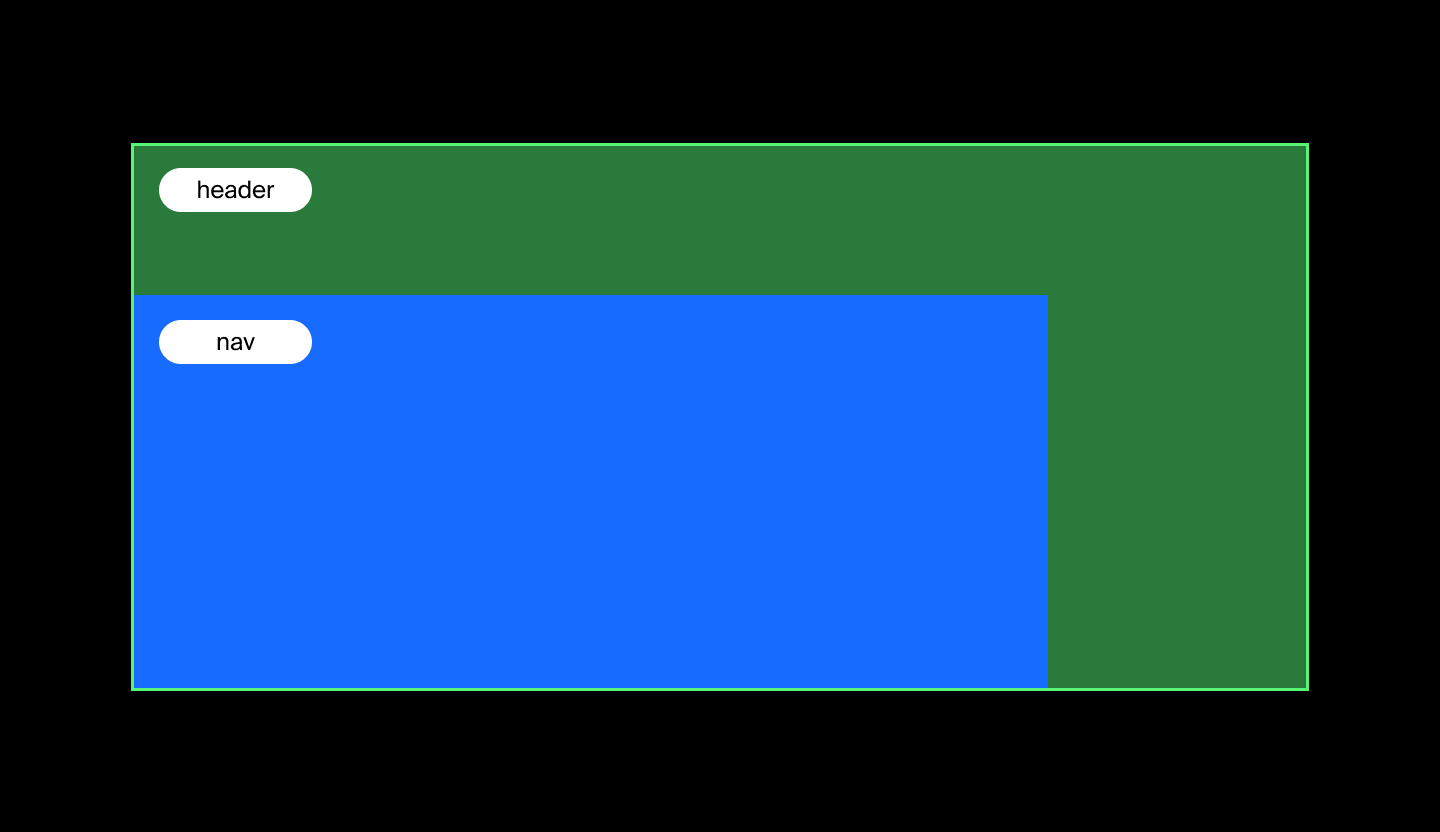
grid-column: 1/3;

grid-row: 1/2;

z-index: 1;

align-self: end;

}



Свойство align-self позволило изменить расположение шапки, не меняя разметки

Этот способ наложения — временное решение. Вёрстка начнёт ломаться, когда в блоках станет больше элементов или их размеры изменятся. Если гипотеза подтвердится, наложение придётся переделать. Но в бизнесе такие быстрые решения уместны.