**Конспект «Гриды: расположение грид-элементов». Раздел 1**

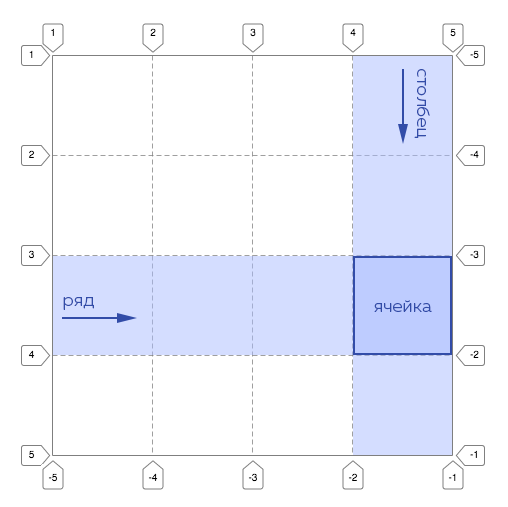
Чтобы сделать элемент грид-контейнером, нужно задать ему соответствующее значение свойства display:

.container {

display: grid;

}

В гриде элементы располагаются по двумерной сетке. То есть грид состоит из рядов и столбцов, располагающихся между линий, которые нумеруются по порядку. Одно «деление» грида называют ячейкой.



Чтобы расположить элемент по сетке внутри грида, нужно задать ему координаты по столбцам и по рядам: с какой линии столбцов и рядов грид-область будет начинаться, на какой линии столбцов и рядов будет заканчиваться. Координаты грид-области на иллюстрации выше в коде описываются так:

/\*

Область

начинается с 4 линии столбцов,

заканчивается на 5 линии столбцов,

начинается на 3 линии рядов,

заканчивается на 4 линии рядов.

\*/

.element {

grid-column-start: 4;

grid-column-end: 5;

grid-row-start: 3;

grid-row-end: 4;

}

Координаты можно отсчитывать не только от начала, но и от конца грида. При этом к индексу линии, от которой ведётся отсчёт, добавляет знак «минус». Координаты той же грид-области можно описать следующим образом:

/\*

Область

начинается со 2 линии столбцов с конца грида,

заканчивается на 1 линии столбцов с конца грида,

начинается на 3 линии рядов с конца грида,

заканчивается на 2 линии рядов с конца грида.

\*/

.element {

grid-column-start: -2;

grid-column-end: -1;

grid-row-start: -3;

grid-row-end: -2;

}

Существует также сокращённый синтаксис для этих свойств. Свойство grid-column объединяет в себе сразу два свойства: grid-column-start/grid-column-end.

Пример:

grid-column: 1 / 3;

/\* Это то же самое, что: \*/

grid-column-start: 1;

grid-column-end: 3;

Аналогично, свойство grid-row — это сокращение для задания пары свойств: grid-row-start/grid-row-end.

Пример:

grid-row: 1 / -2;

/\* Это то же самое, что: \*/

grid-row-start: 1;

grid-row-end: -2;

Если в свойстве grid-row или grid-column не задать второй параметр, то значение останется валидным, но применится только первый параметр.

Грид-элементы могут наслаиваться друг на друга, при этом они начинают себя вести *как будто* абсолютно спозиционированные, при этом на них так же действует свойство z-index. Чем больше значение z-index, тем выше грид-элемент в «стопке».

**Конспект «Гриды: создание раскладки». Раздел 2**

Чтобы задать гриду определённое количество столбцов и рядов, существуют свойства grid-template-columns и grid-template-rows.

Свойство grid-template-columns перечисляет количество и ширину будущих столбцов грида:

/\*

Задаём гриду три столбца,

первый шириной 100px,

второй шириной 200px,

третий — 300px.

\*/

.element {

grid-template-columns: 100px 200px 300px;

}

Аналогично grid-template-columns работает и свойство grid-template-rows, только оно сообщает гриду, сколько рядов он будет содержать и какой они будут высоты:

/\*

Задаём гриду три ряда,

первый высотой 100px,

второй высотой 200px,

третий — 300px.

\*/

.element {

grid-template-rows: 100px 200px 300px;

}

Также есть возможность задавать нефиксированный размер ячейкам. Для этого существует значение auto:

/\*

Задаём гриду два столбца,

первый с нефиксированной шириной,

второй шириной 100px.

\*/

.element {

grid-template-columns: auto 100px;

}

/\*

Задаём гриду три ряда,

первый высотой 100px,

второй с нефиксированной высотой,

третий высотой 200px.

\*/

.element {

grid-template-rows: 100px auto 200px;

}

При заданных свойствах grid-template-columns и grid-template-rows грид-элементы вписываются в заданную сетку автоматически. При этом часть грид-элементов также может иметь чёткие координаты в гриде. Комбинируя задание явного расположения грид-элементов и их автоматическое распределение, можно строить сложные и одновременно гибкие сетки.

Ещё один механизм создания раскладки грида заключается в использовании свойств grid-template-areas и grid-area. В значении свойства grid-template-areas визуально «по клеточкам» описывается структура грида.

Пример:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

HTML:

<div class="grid-container">

<div class="grid-element-1"></div>

<div class="grid-element-2"></div>

<div class="grid-element-3"></div>

</div>

CSS:

.grid-container {

grid-template-areas:

"red yellow green"

"red yellow green"

"red yellow green";

}

.grid-element-1 {

grid-area: red;

}

.grid-element-2 {

grid-area: yellow;

}

.grid-element-3 {

grid-area: green;

}

Свойство grid-template-areas позволяет некоторые ячейки грида помечать как пустые. Для этого вместо буквенного именования области используется символ точки .

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

.grid-container {

grid-template-areas:

". . green"

"red yellow green"

"red yellow green";

}

Свойство gap позволяет добавлять равномерный интервал между рядами и столбцами. Чтобы добавить интервал только между рядами, используется свойство row-gap, а только между столбцами — column-gap.

.grid-container {

gap: 10px; /\* Между рядами и столбцами интервал 10px \*/

column-gap: 20px; /\* Между столбцами интервал 20px \*/

row-gap: 30px; /\* Между рядами интервал 30px \*/

}

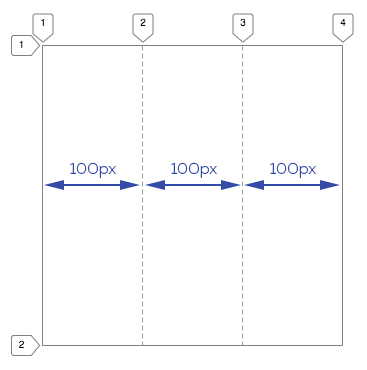
**Автоматические координаты элементов в гриде: столбцы**

Окей, мы научились экономить время и использовать именованные области! Круто!

Но давайте теперь вернёмся в самое начало, [к первым заданиям этой части](https://htmlacademy.ru/courses/269/run/1). В них мы имели дело с гридом фиксированной ширины 4 на 4 ячейки. Внимательные студенты наверняка заметили загадочные свойства grid-template-columns и grid-template-rows. Что же это за свойства?

Свойство grid-template-columns перечисляет количество и ширину будущих столбцов грида. К примеру, запись grid-template-columns: 100px 100px 100px; говорит гриду:

Ты будешь иметь три столбца, каждый шириной 100px.



А что же в это время происходит с рядами?

Если не задавать параметры для рядов, то их количество становится переменным. Порядок действий такой:

1. Все грид-элементы выстраиваются в один ряд согласно объявленной раскладке столбцов.
2. Если грид-элементов больше, чем столбцов, то *не вписавшиеся* в один ряд элементы переносятся на следующий ряд и заполняют его.
3. Новые ряды формируются до тех пор, пока все грид-элементы не будут вписаны в раскладку по столбцам. Если последний ряд заполняется элементами не полностью, то оставшееся место в гриде остаётся пустым.

Рассмотрим пример. Возьмём грид с пятью элементами, которому задан один столбец. По раскладке получается, что в первый ряд грида может поместиться всего один грид-элемент. Остальные элементы будут переноситься на новые ряды, и таким образом сформируют пять рядов, то есть выстроятся в столбец:

Если же мы увеличим количество столбцов до двух, то в один ряд будут помещаться уже по два элемента. То есть всего элементы сформируют три ряда, в первом и втором ряду будет по два элемента, а в третьем — один:

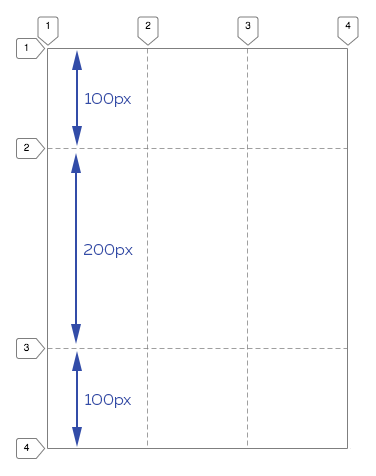
Что касается высоты рядов, то она распределится равномерно, чтобы ряды заполнили собой всё пространство грида.

Ну, а чтобы всё стало совсем понятно, перейдём к практике.

**Автоматические координаты элементов в гриде: столбцы и ряды**

Аналогично grid-template-columns работает и свойство grid-template-rows, только оно сообщает гриду сколько рядов он будет содержать и какой они будут высоты. К примеру, запись grid-template-rows: 100px 200px 100px; говорит гриду:

Ты будешь иметь три ряда, первый высотой 100px, второй высотой 200px и третий — 100px.



Если при этом столбцам не заданы явные параметры, то по умолчанию будет существовать всего один столбец, элементы в котором растянутся на всю его ширину.

В случае, если в гриде есть ряды, которым не задана явная высота в grid-template-rows, оставшееся свободное пространство по высоте распределится среди них равномерно.

Таким образом, зная об особенностях распределения областей в гридах и используя свойства grid-template-columns и grid-template-rows, можно создавать нужные вам сетки.

# Нефиксированная ширина столбцов

Выполняя прошлые задания, мы убедились, что если не задавать строкам и столбцам грида явные размеры, то они по умолчанию будут занимать всё свободное пространство. Давайте разберёмся, как мы можем извлечь из этого пользу.

Рассмотрим пример, когда нам нужно задать явную ширину первому и последнему столбцу трёхколоночного грида, а среднему столбцу отдать всё оставшееся пространство по ширине, не фиксируя его размеры.

Сейчас мы умеем задавать размеры столбцов с помощью фиксированных значений, например, вот так:

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

Но если мы хотим оставить среднему столбцу автоматический размер, то в объявлении должно появиться ключевое слово auto. То есть код станет таким:

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

# Нефиксированная ширина столбцов и рядов

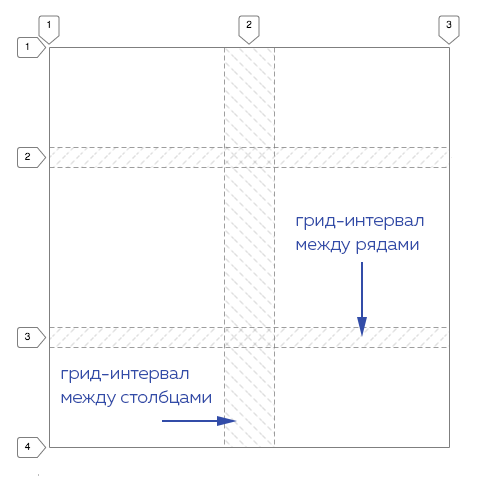
Значение auto можно использовать и в определении столбцов, и в определении строк. Можно даже делать это одновременно.

К примеру, давайте создадим такой грид: вся ширина грида распределяется на два столбца поровну автоматически, первый и последний ряды имеют фиксированные размеры.

**Грид-интервал: свойства row-gap и column-gap**

Не всегда нам необходимы одинаковые интервалы между рядами и столбцами, и в этом случае к нам на помощь приходят свойства column-gap и row-gap.

Как вы уже догадались, первое свойство задаёт интервал между столбцами, а второе — между рядами.



.grid {

row-gap: 20px; /\* Грид-интервал между рядами \*/

column-gap: 50px; /\* Грид-интервал между столбцами \*/

}

На самом деле с помощью свойства gap тоже можно задавать интервал отдельно для столбцов и для рядов. Если задать gap не одно, а два значения, первое будет определять интервал между рядами, а второе — между столбцами. Второе значение задаётся через пробел после первого:

.grid {

/\*

Грид-интервал между рядами 20px,

грид-интервал между столбцами 50px.

\*/

gap: 20px 50px;

}

Свойства column-gap и row-gap раньше назывались grid-column-gap и grid-row-gap соответственно. Как и gap, они были переименованы и теперь используются не только с гридами.

Карточки товаров

.cards-list {

display: grid;

grid-template-columns: repeat(auto-fit, 140px);

gap: 15px;

margin: 0;

padding: 20px;

}

**Свойство align-items**

По умолчанию грид-элементы занимают всё доступное пространство по высоте. Такое поведение можно изменить с помощью свойства align-items. Оно задаётся грид-контейнеру и управляет выравниванием грид-элементов по вертикали.

.grid-container {

display: grid;

align-items: start;

}

У align-items могут быть следующие значения:

* stretch — значение по умолчанию; элементы начинаются у верхней границы и растягиваются на всю высоту.
* start — элементы выстраиваются по верхней границе и, если содержимого немного, не тянутся на всю высоту;
* end — элементы выстраиваются по нижней границе;
* center — элементы располагаются по центру;
* не сработает.
* **repeat**
* Если все колонки в грид-контейнере должны быть одинаковой ширины, то удобно использовать значение-функцию repeat. В скобках после repeat указывают количество колонок и их ширину. Значения разделяют запятой:
* grid-template-columns: repeat(количество колонок, ширина колонки);
* **auto-fit**
* Если количество колонок зависит от ширины контейнера, используют специальное значение auto-fit. Его указывают в скобках после repeat вместо числа колонок:
* grid-template-columns: repeat(auto-fit, ширина колонки);
* **repeat (auto-fit, 100px)**
* width: 200px;
* width: 300px;
* width: 400px;
* **minmax**
* Чтобы ширина колонок изменялась пропорционально свободному пространству в контейнере, используют значение-функцию minmax.
* Его указывают в repeat вместо фиксированной ширины колонок. В скобках после minmax задают минимальный и максимальный размеры колонок, они разделяются запятой:
* repeat(auto-fit, minmax(минимальный размер, максимальный размер));
* В minmax в качестве максимального значения часто используют единицу измерения fr. Она позволяет колонкам увеличивать ширину до тех пор, пока свободного пространства в контейнере не хватит на ещё одну колонку.
* **minmax (100px, 1fr)**
* width: 200px;
* width: 299px;
* width: 300px;
* **Значение minmax**
* Мы сделали так, чтобы количество колонок в списке зависело от ширины страницы. Но если пространство по ширине не получается разделить на количество колонок без остатка, то справа остаётся много свободного места.
* Чтобы от него избавиться, можно увеличить ширину колонок. Но фиксированная ширина проблему не решит — если страница ещё увеличится, то свободное пространство появится снова. Поэтому нужно, чтобы ширина колонок изменялась динамически и зависела от наличия свободного пространства.
* Это можно сделать с помощью ещё одного значения-функции minmax. Его указывают в repeat вместо фиксированной ширины колонок. В скобках после minmax задают минимальный и максимальный размеры колонок, они разделяются запятой:
* repeat(auto-fit, minmax(минимальный размер, максимальный размер));
* Код ниже позволит колонкам изменять ширину в зависимости от имеющегося в контейнере свободного пространства. При этом колонки не станут меньше 100px и не растянутся больше, чем на 150px:
* .grid-container {
* display: grid;
* grid-template-columns: repeat(auto-fit, minmax(100px, 150px));
* }
* В minmax в качестве максимального значения часто используют единицу измерения fr. Она позволяет колонкам увеличивать ширину до тех пор, пока свободного пространства в контейнере не хватит на ещё одну колонку.
* **minmax (100px, 1fr)**
* width: 200px;
* width: 299px;
* width: 300px;
* В шаблоне списка укажем с помощью minmax минимальный и максимальный размеры колонок и понаблюдаем, как перестроится список, если изменить ширину страницы. После этого посмотрим, что случится, если добавить в список карточку, чьё содержимое шире колонки.
* **Свойства grid-column и grid-row**
* Чтобы растянуть элемент на несколько колонок используют свойство grid-column. Число после ключевого слова span указывает число колонок, которые элемент должен занять:
* .element {
* grid-column: span 2;
* }
* Растянуть элемент на несколько рядов можно с помощью свойства grid-row. Ключевое слово span в нём означает количество рядов, которые элемент должен занять:
* .long-element {
* grid-row: span 2;
* }
* Свойства grid-column и grid-row можно использовать одновременно.
* **Свойство grid-auto-flow**
* Свойство grid-auto-flow управляет автозаполнением грид-контейнера.
* .grid-container {
* display: grid;
* grid-auto-flow: row;
* }
* Значение по умолчанию row говорит располагать элементы в том порядке, в котором они идут в разметке, и при необходимости создавать новые ряды:
* 1
* 2
* 3. Карточка, занимающая две колонки.
* 4
* 5
* Но если указать значение dense, то контейнер будет заполняться так, чтобы не было пропусков:
* 1
* 2
* 3. Карточка, занимающая две колонки.
* 4
* 5
* Значение dense заставляет грид-контейнер заполнять пустые ячейки первым подходящим по размеру грид-элементом. При этом визуальный порядок на странице может отличаться от порядка элементов в разметке. Если порядок элементов важен, лучше это значение не использовать.
* **Свойство grid-column**
* Мы добавили в список новую карточку, содержимое которой не помещается в одну колонку. Дизайнер настаивает, что уменьшать текст и картинку нельзя. Как быть? Разрешим карточке занять две колонки.
* Сделать это можно с помощью свойства grid-column:
* .element {
* grid-column: span 2;
* }
* Элемент из примера растянется на две колонки:
* 1
* 2
* 3
* 4. Карточка, занимающая две колонки.
* 5
* Чтобы растянуть грид-элемент, используют слово span, а после указывают число колонок, которые элемент должен занять:
* // Этот элемент растянется на три колонки
* .wide-element {
* grid-column: span 3;
* }
* // А этот – на четыре
* .very-wide-element {
* grid-column: span 4;
* }
* Нам нужно, чтобы карточка в списке заняла две колонки. Добавим ей дополнительный класс и используем свойство grid-column, чтобы растянуть её. Поэкспериментируем с разными значениями span и разной шириной страницы.

**Свойство grid-row**

Мы заставили карточку растянуться на несколько колонок. Дизайнеру идея понравилась, и он решил ни в чём себя не ограничивать. Так в списке появилась карточка, которой требуется две колонки *и два ряда*. Она уже добавлена в разметку, осталось только написать правильные стили.

Растянуть элемент на несколько рядов можно с помощью свойства grid-row. Ключевое слово span в нём работает так же, как в grid-column, только означает не количество колонок, которые элемент должен занять, а количество рядов:

.long-element {

grid-row: span 2;

}

Элемент из примера растянется на два ряда:

1

2

3  
Карточка, занимающая два ряда.

4

5

Свойства grid-column и grid-row можно использовать одновременно. Например, такой код заставит элемент занять три колонки и два ряда:

.element {

grid-column: span 3;

grid-row: span 2;

}

**Свойство grid-auto-flow**

Мы сделали так, чтобы карточка с классом huge занимала два ряда и две колонки. Из-за этого в первом ряду образовалась пустая ячейка. Так произошло потому, что в ряду осталось место только для одной колонки, а следующая карточка занимает две. В результате карточка перенеслась на новый ряд, а предыдущий ряд остался незаполненным.

Такое поведение можно изменить с помощью свойства grid-auto-flow. Оно управляет автозаполнением грид-контейнера.

.grid-container {

display: grid;

grid-auto-flow: row;

}

Значение по умолчанию row говорит располагать элементы в том порядке, в котором они идут в разметке, и при необходимости создавать новые ряды:

1

2

3. Карточка, занимающая две колонки.

4

5

Но если указать значение dense, то контейнер будет заполняться так, чтобы не было пропусков:

1

2

3. Карточка, занимающая две колонки.

4

5

Значение dense заставляет грид-контейнер заполнять пустые ячейки первым подходящим по размеру грид-элементом. При этом визуальный порядок на странице может отличаться от порядка элементов в разметке. Если порядок элементов важен, лучше это значение не использовать.

В нашем списке порядок карточек не имеет значения, но желательно заполнять ряды полностью. Чтобы этого добиться, добавим списку карточек свойство grid-auto-flow со значением dense. Затем изменим ширину страницы и понаблюдаем, как элементы меняются местами.

Наконец, уберём у <body> фиксированную ширину. Будем изменять ширину окна мини-браузера и посмотрим, как меняются размеры колонок и перестраивается список карточек.