**What is CSS Grid?**

* CSS Grid Layout is a **two-dimensional** grid-based layout.
* Good alternative to tables, floats or inline-block.

**Grid Container**

The element on which display: grid is applied. It’s the direct parent of all the grid items. In this example container is the grid container.

<div class="container">

<div class="item item-1"> </div>

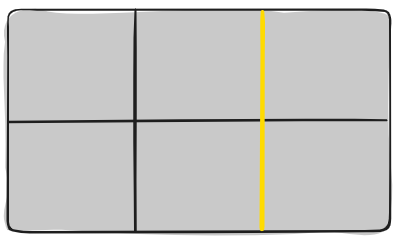
<div class="item item-2"> </div>

<div class="item item-3"> </div>

</div>

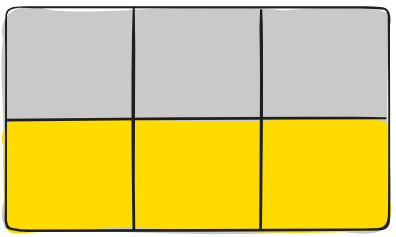
**Grid Line**

The **dividing** **lines** that make up the structure of the grid. They can be either vertical (“column grid lines”) or horizontal (“row grid lines”) and reside on either side of a row or column. Here the yellow line is an example of a column grid line.



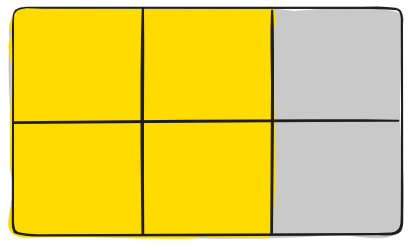
**Grid Track**

The space between two adjacent grid lines. You can think of them as the columns or rows of the grid. Here’s the grid track between the second and third-row grid lines.



**Grid Area**

The total space surrounded by four grid lines. A grid area may be composed of any number of grid cells. Here’s the grid area between row grid lines 1 and 3, and column grid lines 1 and 3.



**Grid Item**

The children (*i.e. direct descendants*) of the grid container. Here the item elements are grid items, but sub-item isn’t.

<div class="container">

<div class="item"> </div>

<div class="item">

<p class="sub-item"> </p>

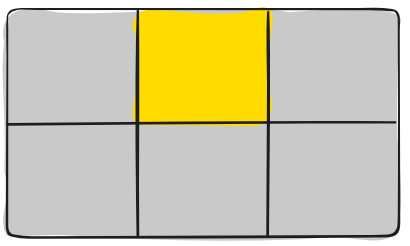
</div>

<div class="item"> </div>

</div

**Grid Cell**

The space between two adjacent row and two adjacent column grid lines. It’s a single “unit” of the grid. Here’s the grid cell between row grid lines 1 and 2, and column grid lines 2 and 3.



**Properties**

**Properties for the Parents**

**Display**

Defines the element as a grid container and establishes a new grid formatting context for its contents.

Values:

* **grid** – generates a block-level grid
* **inline-grid** – generates an inline-level grid

.container {

display: grid | inline-grid;

}

**grid-template-columns / grid-template-rows**

Defines the columns and rows of the grid with a space-separated list of values. The values represent the track size, and the space between them represents the grid line.

Values:

* **<track-size>** – can be a length, a percentage, or a fraction of the free space in the grid (using the fr unit)
* **<line-name>** – an arbitrary name of your choosing

.container {

grid-template-columns: ... ...;

/\* e.g.

1fr 1fr

minmax(10px, 1fr) 3fr

repeat(5, 1fr)

50px auto 100px 1fr

\*/

grid-template-rows: ... ...;

/\* e.g.

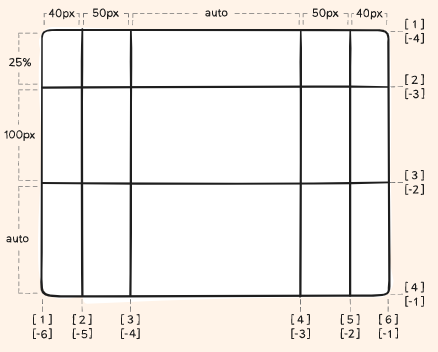
min-content 1fr min-content

100px 1fr max-content

\*/

}

Grid lines are automatically assigned positive numbers from these assignments (-1 being an alternate for the very last row).



Note that a line can have more than one name. For example, here the second line will have two names: row1-end and row2-start:

.container {

grid-template-rows: [row1-start] 25% [row1-end row2-start] 25% [row2-end];

}

If your definition contains repeating parts, you can use the repeat() notation to streamline things:

.container {

grid-template-columns: repeat(3, 20px [col-start]);

}

Which is equivalent to this:

.container {

grid-template-columns: 20px [col-start] 20px [col-start] 20px [col-start];

}

If multiple lines share the same name, they can be referenced by their line name and count.

.item {

grid-column-start: col-start 2;

}

The fr unit allows you to set the size of a track as a fraction of the free space of the grid container. For example, this will set each item to one third the width of the grid container:

.container {

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

}

The free space is calculated after any non-flexible items. In this example the total amount of free space available to the fr units doesn’t include the 50px:

.container {

grid-template-columns: 1fr 50px 1fr 1fr;

}

**grid-template-areas**

Defines a grid template by referencing the names of the **grid** **areas** which are specified with the **grid-area** property. Repeating the name of a **grid** **area** causes the content to span those cells. A **period** signifies an **empty** **cell**. The syntax itself provides a visualization of the structure of the grid.

Values:

* **<grid-area-name>** – the name of a grid area specified with grid-area
* **.** – a period signifies an empty grid cell
* **none** – no grid areas are defined

.container {

grid-template-areas:

"<grid-area-name> | . | none | ..."

"...";

}

Example:

.item-a {

grid-area: header;

}

.item-b {

grid-area: main;

}

.item-c {

grid-area: sidebar;

}

.item-d {

grid-area: footer;

}

.container {

display: grid;

grid-template-columns: 50px 50px 50px 50px;

grid-template-rows: auto;

grid-template-areas:

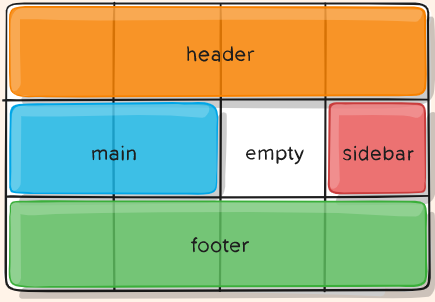
"header header header header"

"main main . sidebar"

"footer footer footer footer";

}

That’ll create a grid that’s four columns wide by three rows tall. The entire top row will be composed of the **header** area. The middle row will be composed of two **main** areas, one empty cell, and one **sidebar** area. The last row is all **footer**.



Each row in your declaration needs to have the same number of cells.

You can use any number of adjacent periods to declare a single empty cell. As long as the periods have no spaces between them, they represent a single cell.

**grid-template**

A shorthand for setting grid-template-rows, grid-template-columns, and grid-template-areas in a single declaration.

Values:

* **none** – sets all three properties to their initial values
* **<grid-template-rows> / <grid-template-columns>** – sets grid-template-columns and grid-template-rows to the specified values, respectively, and sets grid-template-areas to none

**column-gap / row-gap / grid-column-gap / grid-row-gap**

Specifies the size of the grid lines. You can think of it like setting the width of the gutters between the columns/rows.

Values:

* **<line-size>** – a length value

Example:

.container {

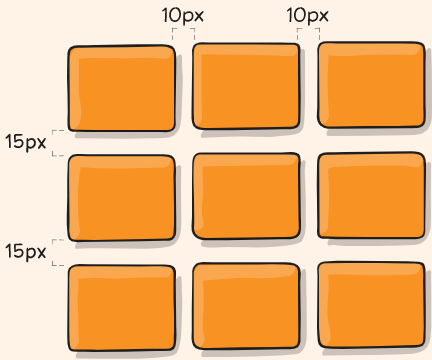
grid-template-columns: 100px 50px 100px;

grid-template-rows: 80px auto 80px;

column-gap: 10px;

row-gap: 15px;

}



The gutters are only created between the columns/rows, not on the outer edges.

.container {

/\* standard \*/

gap: <grid-row-gap> <grid-column-gap>;

}

**Justify Items**

Aligns grid items along the **inline (row) axis.** This value applies to all grid items inside the container.

Values:

* **start** – aligns items to be flush with the start edge of their cell
* **end** – aligns items to be flush with the end edge of their cell
* **center** – aligns items in the center of their cell
* **stretch** – fills the whole width of the cell (this is the default)

Examples:

|  |  |
| --- | --- |
| .container {  justify-items: start;  } |  |
| .container {  justify-items: end;  } |  |
| .container {  justify-items: center;  } |  |
| .container {  justify-items: stretch;  } |  |

**align-items**

Aligns grid items along the block (column) axis (as opposed to justify-items which aligns along the inline (row) axis). This value applies to all grid items inside the container.

Values:

* **stretch** – fills the whole height of the cell (this is the default)
* **start** – aligns items to be flush with the start edge of their cell
* **end** – aligns items to be flush with the end edge of their cell
* **center** – aligns items in the center of their cell
* **baseline** – align items along text baseline.

|  |  |
| --- | --- |
| .container {  align-items: start;  } |  |
| .container {  align-items: end;  } |  |
| .container {  align-items: center;  } |  |
| .container {  align-items: stretch;  } |  |

#### **place-items**

place-items sets both the align-items and justify-items properties in a single declaration.

Values:

* <align-items> / <justify-items> – The first value sets align-items, the second value justify-items. If the second value is omitted, the first value is assigned to both properties.

For example, this can be very useful for super quick multi-directional centering:

.center {

display: grid;

place-items: center;

}

**justify-content**

Sometimes the **total size** of your **grid** might be **less** than the size of its **grid** **container**. This could happen if all of your **grid items** are sized with **non-flexible** units like **px**. In this case you can set the alignment of the grid within the grid container.

Values:

* **start** – aligns the grid to be flush with the start edge of the grid container
* **end** – aligns the grid to be flush with the end edge of the grid container
* **center** – aligns the grid in the center of the grid container
* **stretch** – resizes the grid items to allow the grid to fill the full width of the grid container
* **space-around** – places an even amount of space between each grid item, with half-sized spaces on the far ends
* **space-between** – places an even amount of space between each grid item, with no space at the far ends
* **space-evenly** – places an even amount of space between each grid item, including the far ends

|  |  |
| --- | --- |
| .container {  justify-content: start;  } |  |
| .container {  justify-content: end;  } |  |
| .container {  justify-content: center;  } |  |
| .container {  justify-content: stretch;  } |  |
| .container {  justify-content: space-around;  } |  |
| .container {  justify-content: space-between;  } |  |
| .container {  justify-content: space-evenly;  } |  |

**align-content**

Sometimes the **total size** of your **grid** might be less than the size of its **grid container**. This could happen if all of your **grid items** are sized with **non-flexible** units like **px**. In this case you can set the alignment of the grid within the grid container.

Values:

* **start** – aligns the grid to be flush with the start edge of the grid container
* **end** – aligns the grid to be flush with the end edge of the grid container~
* **center** – aligns the grid in the center of the grid container
* **stretch** – resizes the grid items to allow the grid to fill the full height of the grid container
* **space-around** – places an even amount of space between each grid item, with half-sized spaces on the far ends
* **space-between** – places an even amount of space between each grid item, with no space at the far ends
* **space-evenly** – places an even amount of space between each grid item, including the far ends

|  |  |
| --- | --- |
| .container {  align-content: start;  } |  |
| .container {  align-content: end;  } |  |
| .container {  align-content: center;  } |  |
| .container {  align-content: stretch;  } |  |
| .container {  align-content: space-around;  } |  |
| .container {  align-content: space-between;  } |  |
| .container {  align-content: space-evenly;  } |  |

**place-content**

place-content sets both the align-content and justify-content properties in a single declaration.

Values:

* **<align-content> / <justify-content>** – The first value sets align-content, the second value justify-content. If the second value is omitted, the first value is assigned to both properties.

All major browsers except *Edge* support the place-content shorthand property.

**grid-auto-columns / grid-auto-rows**

Specifies the size of any auto-generated grid tracks (*aka implicit grid tracks*). Implicit tracks get created when there are **more grid items than cells in the grid** or **when a grid item is placed outside of the explicit grid**.

Values:

* **<track-size>** – can be a length, a percentage, or a fraction of the free space in the grid (using the fr unit)

To illustrate how *implicit grid* tracks get created, think about this:

|  |  |
| --- | --- |
| .container {  grid-template-columns: 60px 60px;  grid-template-rows: 90px 90px;  } |  |

This creates a 2 x 2 grid.

But now imagine you use grid-column and grid-row to position your grid items like this:

|  |  |
| --- | --- |
| .item-a {  grid-column: 1 / 2;  grid-row: 2 / 3;  }  .item-b {  grid-column: 5 / 6;  grid-row: 2 / 3;  } |  |

We told .item-b to start on column line 5 and end at column line 6, but we never defined a column line 5 or 6. Because we referenced lines that don’t exist, implicit tracks with widths of 0 are created to fill in the gaps. We can use grid-auto-columns and grid-auto-rows to specify the widths of these implicit tracks:

|  |  |
| --- | --- |
| .container {  grid-auto-columns: 60px;  } |  |

**grid-auto-flow**

If you have grid items that you don’t explicitly place on the grid, the auto-placement algorithm kicks in to automatically place the items. This property controls how the auto-placement algorithm works.

Values:

* **row** – tells the auto-placement algorithm to fill in each row in turn, adding new rows as necessary (default)
* **column** – tells the auto-placement algorithm to fill in each column in turn, adding new columns as necessary
* **dense** – tells the auto-placement algorithm to attempt to fill in holes earlier in the grid if smaller items come up later

Note that **dense** only changes the visual order of your items and might cause them to appear out of order, which is bad for accessibility.

Examples:

Consider this HTML:

.container {

display: grid;

grid-template-columns: 60px 60px 60px 60px 60px;

grid-template-rows: 30px 30px;

grid-auto-flow: row;

}

When placing the items on the grid, you only specify spots for two of them (we set grid-auto-flow to row, which is the default):

|  |  |
| --- | --- |
| .item-a {  grid-column: 1;  grid-row: 1 / 3;  }  .item-e {  grid-column: 5;  grid-row: 1 / 3;  } |  |

If we instead set grid-auto-flow to **column**, item-b, item-c and item-d flow down the columns:

|  |  |
| --- | --- |
| .container {  display: grid;  grid-template-columns: 60px 60px 60px 60px 60px;  grid-template-rows: 30px 30px;  grid-auto-flow: column;  } |  |

**Grid**

**Properties for the Children**

**grid-column-start / grid-column-end / grid-row-start / grid-row-end**

Determines a grid item’s location within the grid by referring to specific grid lines. grid-column-start / grid-row-start is the line where the item begins, and grid-column-end/grid-row-end is the line where the item ends.

Values:

* **<line>** – can be a number to refer to a numbered grid line, or a name to refer to a named grid line
* **span <number>** – the item will span across the provided number of grid tracks
* **span <name>** – the item will span across until it hits the next line with the provided name
* **auto** – indicates auto-placement, an automatic span, or a default span of one

.item {

grid-column-start: <number> | <name> | span <number> | span <name> | auto;

grid-column-end: <number> | <name> | span <number> | span <name> | auto;

grid-row-start: <number> | <name> | span <number> | span <name> | auto;

grid-row-end: <number> | <name> | span <number> | span <name> | auto;

}

See the two examples below:

|  |  |
| --- | --- |
| .item-a {  grid-column-start: 2;  grid-column-end: five;  grid-row-start: row1-start;  grid-row-end: 3;  } |  |
| .item-b {  grid-column-start: 1;  grid-column-end: span col4-start;  grid-row-start: 2;  grid-row-end: span 2;  } |  |

**grid-column / grid-row**

Shorthand for grid-column-start + grid-column-end, and grid-row-start + grid-row-end, respectively.

Values:

* **<start-line> / <end-line>** – each one accepts all the same values as the longhand version, including span

.item {

grid-column: <start-line> / <end-line> | <start-line> / span <value>;

grid-row: <start-line> / <end-line> | <start-line> / span <value>;

}

|  |  |
| --- | --- |
| .item-c {  grid-column: 3 / span 2;  grid-row: third-line / 4;  } |  |

If no end line value is declared, the item will span 1 track by default.

**grid-area**

Gives an item a name so that it can be referenced by a template created with the grid-template-areas property. Alternatively, this property can be used as an even shorter shorthand for grid-row-start + grid-column-start + grid-row-end + grid-column-end.

Values:

* **<name>** – a name of your choosing
* **<row-start> / <column-start> / <row-end> / <column-end>** – can be numbers or named lines

.item {

grid-area: <name> | <row-start> / <column-start> / <row-end> / <column-end>;

}

Examples:

As a way to assign a name to the item:

.item-d {

grid-area: header;

}

As the short-shorthand for grid-row-start + grid-column-start + grid-row-end + grid-column-end:

|  |  |
| --- | --- |
| .item-d {  grid-area: 1 / col4-start / last-line / 6;  } |  |

**justify-self**

Aligns a grid item inside a cell along the inline (row) axis. This value applies to a grid item inside a single cell.

Values:

* **start** – aligns the grid item to be flush with the start edge of the cell
* **end** – aligns the grid item to be flush with the end edge of the cell
* **center** – aligns the grid item in the center of the cell
* **stretch** – fills the whole width of the cell (this is the default)

.item {

justify-self: start | end | center | stretch;

}

|  |  |
| --- | --- |
| .item-a {  justify-self: start;  } |  |
| .item-a {  justify-self: end;  } |  |
| .item-a {  justify-self: center;  } |  |
| .item-a {  justify-self: stretch;  } |  |

To set **alignment for all the items** in a grid, this behavior can also be set on the grid container via the justify-items property.

**align-self**

Aligns a grid item inside a cell along the block (column). This value applies to the content inside a single grid item.

Values:

* start – aligns the grid item to be flush with the start edge of the cell
* end – aligns the grid item to be flush with the end edge of the cell
* center – aligns the grid item in the center of the cell
* stretch – fills the whole height of the cell (this is the default)

.item {

align-self: start | end | center | stretch;

}

|  |  |
| --- | --- |
| .item-a {  align-self: start;  } |  |
| .item-a {  align-self: end;  } |  |
| .item-a {  align-self: center;  } |  |
| .item-a {  align-self: stretch;  } |  |

To align all the items in a grid, this behavior can also be set on the grid container via the align-items property.

**place-self**

place-self sets both the align-self and justify-self properties in a single declaration.

Values:

* **auto** – The “default” alignment for the layout mode.
* **<align-self> / <justify-self>** – The first value sets align-self, the second value justify-self. If the second value is omitted, the first value is assigned to both properties.

|  |  |
| --- | --- |
| .item-a {  place-self: center;  } |  |
| .item-a {  place-self: center stretch;  } |  |

All major browsers except *Edge* support the place-self shorthand property.

**Special Units & Functions**

**fr units**

They essentially mean “portion of the remaining space”. So a declaration like:

grid-template-columns: 1fr 3fr;

Means, loosely, 25% 75%. Except that those percentage values are much more firm than fractional units are.

**Sizing Keywords**

When sizing rows and columns, you can use all the lengths you are used to, like **px**, **rem**, **%**, etc, but you also have keywords:

* **min-content**: the minimum size of the content. Imagine a line of text like “E pluribus unum”, the min-content is likely the width of the word “pluribus”.
* **max-content**: the maximum size of the content. Imagine the sentence above, the max-content is the length of the whole sentence.
* **auto**: this keyword is a lot like fr units, except that they “lose” the fight in sizing against fr units when allocating the remaining space.
* **fit-content**: use the space available, but never less than min-content and never more than max-content.
* **fractional units**: see above

**Sizing Functions**

The minmax() function does exactly what it seems like: it sets a minimum and maximum value for what the length is able to be. This is useful for in combination with relative units. Like you may want a column to be only able to shrink so far.

grid-template-columns: minmax(100px, 1fr) 3fr;

**The repeat() Function and Keywords**

The repeat() function can save some typing:

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

/\* easier: \*/

grid-template-columns: repeat(8, 1fr);

But repeat() can get extra fancy when combined with keywords:

* **auto-fill**: Fit as many possible columns as possible on a row, even if they are empty.
* **auto-fit**: Fit whatever columns there are into the space. Prefer expanding columns to fill space rather than empty columns.

grid-template-columns:

repeat(auto-fit, minmax(250px, 1fr));