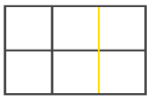
Grid Layout – Advanced

It is assumed that you have learned the basics of grid layout although we will be doing a quick review here.

The CSS rules are applied to the parent element that then becomes a grid container and to the children elements that then become the grid items.

To start the grid layout you have to define a container element as grid using display:grid and for that same element you will then define the columns and rows using the grid-template-columns and grid-template-rows CSS properties. The placement of each child of the grid container (the parent) will be done by using grid-column and grid-row (other CSS properties are available for that purpose too).

There are some terms that belong to the grid layout terminology:



Grid Line – the dividing lines that make up the structure of the grid (vertical = column grid lines or horizontal = row grid lines). The grid line resides on either side of a row or column

Grid Track – the space between two adjacent grid lines and you can think of them like the columns or rows of the grid

Grid Cell – the space between two adjacent row and two adjacent column grid lines. It is a single "unit" of the grid.

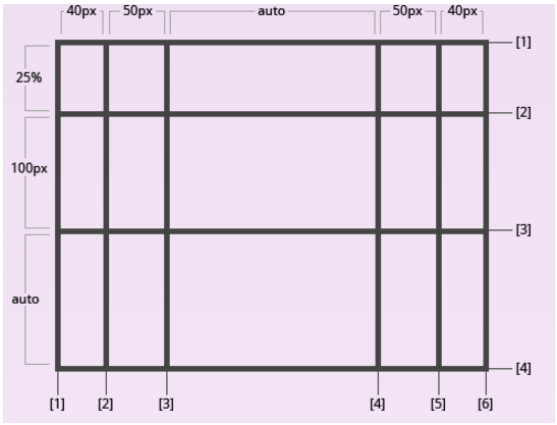
Grid Area – the total space surrounded by four grid lines. A grid area may be comprised of any number of grid cells. The image on the side shows the grid area between row grid lines 1 and 3 and column grid lines 1 and 3

CSS Properties for grid container

grid-template-columns, **grid-template-rows**: define the columns and rows of the grid with a space-separated list of values. The values will represent the track size and the space between will represent the grid line. The track size can be a length, a percentage, or a fraction of the free space in the grid (using the fr unit). You have also the option of defining the line name with a name of your choice.

Example:

```
.container {
  grid-template-columns: 40px 50px auto 50px 40px;
  grid-template-rows: 25% 100px auto;
}
```



Example with the naming of lines:

```
.container {
  grid-template-columns: [first] 40px [line2] 50px [line3] auto [col4-start] 50px [five] 40px [end];
  grid-template-rows: [row1-start] 25% [row1-end] 100px [third-line] auto [last-line];
```

}

A line can have more than one name – in the code below the second line has two names (row1-end and row2-start):

.container {
grid-template-rows: [row1-start] 25% [row1-end row2-start] 25% [row2-end];
}

When defining the values for the grid-template-rows and grid-template-columns you can use the repeat() function to streamline things, if your definition contains repeating parts.

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.

- ***<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 | …";
}

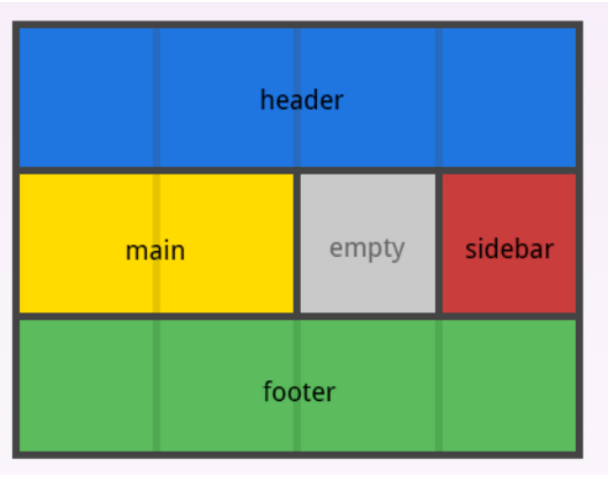
See below an HTML and CSS that uses the grid-template-areas to define areas of the grid and then set the HTML elements to be in that specific area of the grid:

```
<!DOCTYPE html>
<html lang="en">
<head>
<style>
header { grid-area: header; }
nav { grid-area: menu; }
main { grid-area: main; }
aside { grid-area: right; }
footer { grid-area: footer; }

.grid-container {
  display: grid;
  grid-template-areas:
  'header header header header header header'
  'menu main main main right right'
  'menu footer footer footer footer footer';
  grid-gap: 10px;
  background-color: lightblue;
  padding: 10px;
}

header, nav, main, aside, footer {
  background-color: rgba(0, 0, 255, 0.4);
  text-align: center;
  padding: 20px 0;
  font-size: 2em;
  color: white;
```

```
}
</style>
</head>
<body>

<h1>Using Grid Template Areas</h1>

<p>You can use the <em>grid-area</em> property to name grid items.</p>
<p>You can refer to the name when you set up the grid layout, by using the <em>grid-template-areas</em> property on the grid container.</p>
<p>This grid layout contains six columns and three rows:</p>

<div class="grid-container">
  <header>Header</header>
  <nav>Menu</nav>
  <main>Main</main>
  <aside>Right</aside>
  <footer>Footer</footer>
</div>

</body>
</html>
```

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.

Notice this code below and the image that would represent the layout of this grid:




```
.item-a { grid-area: header; }
.item-b { grid-area: main; }
.item-c { grid-area: sidebar; }
.item-d { grid-area: footer; }
.container {
    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"; }
```

Notice that the grid layout has 4 columns defined but in the **grid-template-areas** that define the second row, you see **main main . sidebar** – the **.** (period) is exactly to set an empty grid as you see the gray area in the image above. But all the rows have 4 grid areas defined!

Instead of using **grid-template-columns**, **grid-template-rows**, and **grid-template-areas**, you can use the **grid-template** as a shorthand with these possible values:

- ***none*** - sets all three properties to their initial values
- ***subgrid*** - sets grid-template-rows and grid-template-columns to subgrid, and grid-template-areas to its initial value
- ***<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

If you use grid-template, then the code below:

```
.container {
    grid-template:
        [row1-start] 25px "header header header" [row1-end]
        [row2-start] "footer footer footer" 25px [row2- end]
    / auto 50px auto;
}
```

Would replace this one:

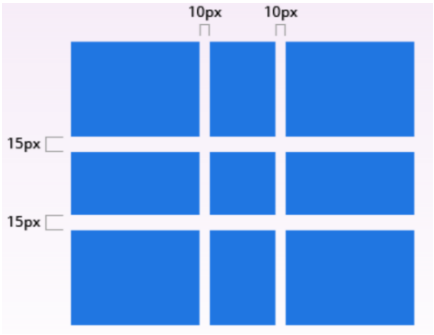
```
.container {
    grid-template-rows: [row1-start] 25px [row1-end row2-start] 25px [row2-end];
    grid-template-columns: auto 50px auto;
    grid-template-areas:
    "header header header"
    "footer footer footer";
}
```

grid-column-gap and **grid-row-gap** - specifies the size of the grid lines. It is like setting the width of the gutters between the columns/rows, not on the outer edges.

grid-gap – a shorthand for **grid-row-gap** and **grid-column-gap**

For example:

```
.container{
  grid-template-columns: 100px 50px 100px;
  grid-template-rows: 80px auto 80px;
  grid-gap: 10px 15px;
}
```



justify-items - aligns the content inside a grid item **along the row axis** (as opposed to **align-items** which aligns **along the column axis**). This value applies to all grid items inside the container. Here are the possible values that **justify-items** can receive:
.container { justify-items: start | end | center | stretch; }

The values for **justify-items** are:
start - aligns the content to the left end of the grid area
center - aligns the content in the center of the grid area

end - aligns the content to the right end of the grid area
stretch - fills the whole width of the grid area (this is the default)

align-items - aligns the content inside a grid item **along the column axis** (as opposed to **justify-items** which aligns **along the row axis**). This value applies to all grid items inside the container.
.container { align-items: start | end | center | stretch; }

The values for **align-items** are:
start - aligns the content to the top of the grid area
center - aligns the content in the center of the grid area
end - aligns the content to the bottom of the grid area
stretch - fills the whole height of the grid area (this is the default)

justify-content - property aligns the grid **along the row axis** (as opposed to **align-content** which aligns the grid **along the column axis**).
.container { justify-content: start | end | center | stretch | space-around | space-between | space-evenly; }

The values for **justify-content** are:
start - aligns the grid to the left end of the grid container
end - aligns the grid to the right end 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

align-content - property aligns the grid **along the column axis** (as opposed to **justify-content** which aligns the grid **along the row axis**).
.container { align-content: start | end | center | stretch | space-around | space-between | space-evenly; }

The values for **align-content** are:
start - aligns the grid to the top of the grid container
end - aligns the grid to the bottom 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

grid-auto-columns and **grid-auto-rows** - specifies the size of any auto-generated grid tracks (aka implicit grid tracks). Implicit grid tracks get created when you explicitly position rows or columns (via grid-template-rows/grid-template-columns) that are out of range of the defined grid.

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

.container {
grid-auto-columns: <track-size> ...;
grid-auto-rows: <track-size> ...;
}

Note: the **grid-template-columns** property overrides the **grid-auto-columns** and the **grid-template-rows** overrides the grid-auto-rows.

grid-auto-flow - controls how the auto-placement algorithm works, how auto-placed items get inserted in the grid.

- **row** - tells the auto-placement algorithm to fill in each row in turn, adding new rows as necessary (this is the default value)
- **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

.container { grid-auto-flow: row | column | row dense | column dense }

Here is an example using some of those grid properties we have seen so far:

```
<!DOCTYPE html>
<html lang="en">
<head>
<style>
.part3 { grid-column: auto / span 2; }

.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  grid-template-rows: auto auto;
  grid-gap: 10px;
  background-color: lightblue;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>
</head>
<body>

<h1>Using grid-auto-flow</h1>

<p>This property controls how auto-placed items are inserted in the grid.</p>
<p>This grid has three columns and two rows.</p>
```

```
<p>The "part3" spans two columns. We will now see where the other elements will be inserted depending on the value of the <strong>grid-auto-flow</strong>.</p>
<h2>grid-auto-flow: row</h2>
<div class="grid-container" style="grid-auto-flow: row;">
  <div class="part1">1</div>
  <div class="part2">2</div>
  <div class="part3">3</div>
  <div class="part4">4</div>
</div>
<h2>grid-auto-flow: row dense</h2>
<p>The "dense" value fills holes in the grid:</p>
<div class="grid-container" style="grid-auto-flow: row dense;">
  <div class="part1">1</div>
  <div class="part2">2</div>
  <div class="part3">3</div>
  <div class="part4">4</div>
</div>

</body>
</html>
```

grid – a shorthand used to set all of the following properties in a single declaration.

- ***none*** - sets all sub-properties to their initial values
- ***<grid-template-rows> / <grid-template-columns>*** - sets grid-template-rows and grid-template-columns to the specified values, respectively, and all other sub-properties to their initial values
- ***<grid-auto-flow> [<grid-auto-rows> [/ <grid-auto-columns>]] -*** accepts all the same values as grid-auto-flow, grid-auto-rows and grid-auto-columns, respectively. If grid-auto-columns is omitted, it is set to the value specified for grid-auto-rows. If both are omitted, they are set to their initial values.

.container { grid: none | <grid-template-rows> / <grid-template-columns> | <grid-auto-flow> [<grid-auto-rows> [/ <grid-auto-columns>]]; }

If using this shorthand, you could have:

```
.container {
grid: 200px auto / 1fr auto 1fr;
}
```

Instead of:

```
.container {
  grid-template-rows: 200px auto;
  grid-template-columns: 1fr auto 1fr;
  grid-template-areas: none;
}
```

You could also have:

```
.container {
grid: column 1fr / auto;
}
```

Instead of:

```
.container {
  grid-auto-flow: column;
  grid-auto-rows: 1fr;
  grid-auto-columns: auto;
}
```

This is another possibility:

```
.container {
  grid: [row1-start] "header header header" 1fr   [row1-end]
      [row2-start] "footer footer footer" 25px        == [row2-end]
      / auto 50px auto;
}
```

To substitute this one below:

```
.container {
  grid-template-areas:
    "header header header"
    "footer footer footer";
  grid-template-rows: [row1-start] 1fr [row1-end] [row2-start] 25px [row2-end];
  grid-template-columns: auto 50px auto;
}
```

CSS Properties for the children (grid items)

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.

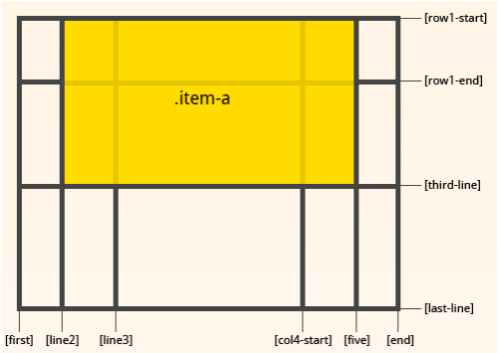
grid-column-start/grid-row-start is the line where the item begins, and **grid-column-end/grid-row-end** is the line where the item ends.

- ***<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

If no **grid-column-end/grid-row-end** is declared, the item will span 1 track by default.

For example, the code below would give something as the image you see here on the right side:

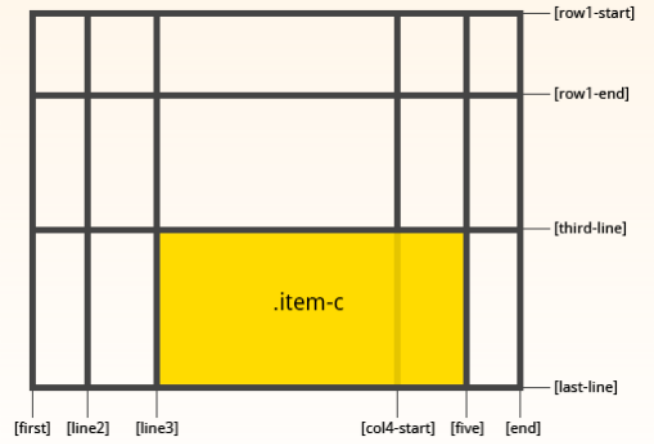
```
.item-a {
grid-column-start: 2;
 grid-column-end: five;
grid-row-start: row1-start
grid-row-end: 3
}
```



grid-column ; **grid-row** - shorthand for **grid-column-start** + **grid-column-end**, and **grid-row-start** + **grid-row-end**, respectively.
<start-line> / <end-line> - each one accepts all the same values as the longhand version, including span

The code below would get something like the image shown on the right side:

```
.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 **-** property can be used as an even shorter shorthand for **grid-row-start** + **grid-column-start** + **grid-row-end** + **grid-column-end**.

The use of **grid-area** will be related to the area names given in the **grid-template-areas** property.

```
.item {
  grid-area: <name> | <row-start> / <column-start> / <row-end> / <column-end>;
}
```

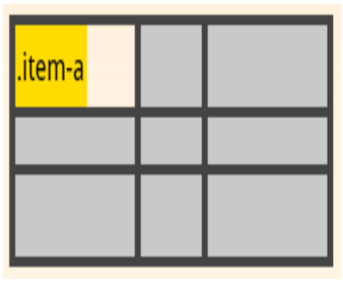
justify-self - aligns the content inside a grid item along the row axis. The values of this property can be:

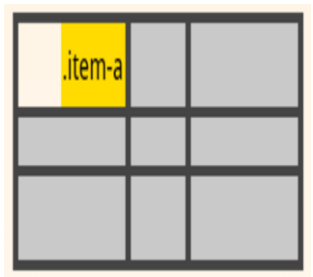
- ***start*** - aligns the content to the left end of the grid area
- ***end*** - aligns the content to the right end of the grid area
- ***center*** - aligns the content in the center of the grid area
- ***stretch*** - fills the whole width of the grid area (this is the default)

align-self **-** aligns the content inside a grid item along the column axis. The values of this property can be:

- ***start*** **-** aligns the content to the top of the grid area
- ***end*** **-** aligns the content to the bottom of the grid area
- ***center*** **-** aligns the content in the center of the grid area
- ***stretch*** **-** fills the whole height of the grid area (this is the default)

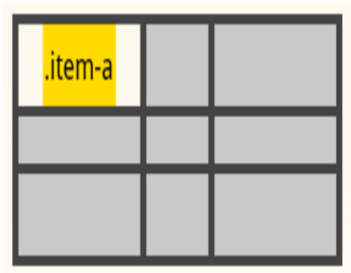
Note: The value applies to the content **inside a single grid item**.





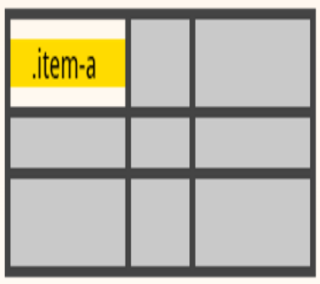
.item-a {
justify-self: start | end | center | stretch;
}

The image on the left shows the effect of **justify-self** to the element with **item-a** class. Starting on the upper left that is showing the **start** value, then on the right top you see the **end** value, on the left bottom you see the **center** value, and on the bottom right you see the **stretch**



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

The image on the left shows the effect of **align-self** to the element with **item-a** class. Starting on the upper left that is showing the **start** value, then on the right top you see the **end** value, on the left bottom you see the **center** value, and on the bottom right you see the **stretch**



Compare the CSS in the files: **fitcontent.html** with **withoutfitcontent.html**. What does happen when you shrink the browser? What about when you expand the browser again?

The minmax (min, max)

You can use this to define the min and max width or height of the grid track. It can be used in grid-template-columns, or grid-template-rows, or both.

The **min** value needs to be smaller than **max** otherwise **max** will be ignored.

Open **minmax.html** and notice the **minmax()** applied to the first column of the grid meaning that min width will be 150px and max width will be 250px

Adding some responsiveness?

Yes, this can be done with the @media query and then use CSS to reorganize the grid system you had for laptop.

For example:

```
@media only screen and (max-width: 600px) {
  .grid {
    grid-template-columns: 100%;
    grid-template-rows: auto 350px auto auto;
    width: 90%; }
  .grid-area-1 { grid-column: 1/2; grid-row: 1/2; }
  .grid-area-2 { grid-column: 1/2 ; grid-row: 2/3; }
  .grid-area-3 { grid-column: 1/2 ; grid-row: 3/4; }
  .grid-area-4 { grid-column: 1/2; grid-row: 4/5; }
}
```

Extra Resources

The dark side of grids – with great tips for accessibility

A complete guide to grids – from CSS Tricks