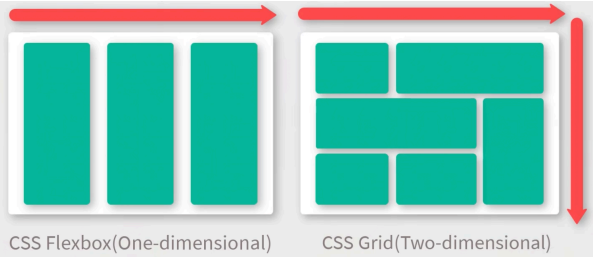
# Advanced CSS

## Section 3: CSS Grid

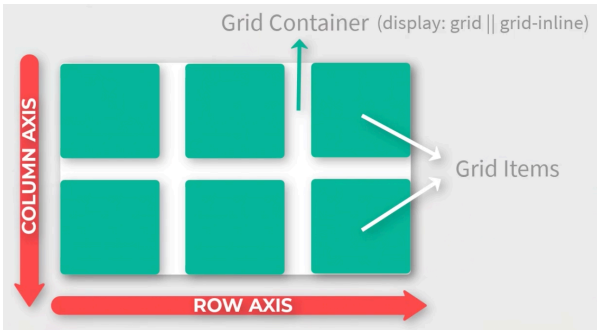
### 3.1 Introduction and How to Create a Grid

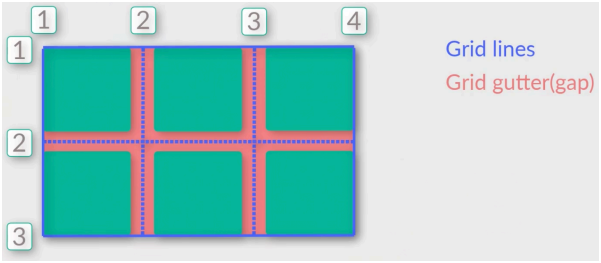
CSS grid allows us to create layout without floats or any older hacks and its is also more powerful compared to CSS Flexbox. CSS Grid is getting more and more popular because it gives more flexibility and control of layout than even before. Both the CSS Flexbox and CSS Grid layout modules work perfectly together and we can utilise the best of both worlds.




The difference between CSS Flexbox and CSS Grid is that Flexbox is one-dimensional while CSS Grid is two-dimensional. Flexbox only allows to align items across rows or columns but not together. CSS Grid allows to align in both the row and column simultaneously.

In the same way as Flexbox where we had flex-container with flex-items CSS grid operates in the same way with grid-container and grid-items. CSS Grid also has row axis as a horizontal direction and a column axis as the vertical direction. In Flexbox we are able to change the horizontal and vertical directions but in the case of CSS Grid we are not able to change the direction of the axis.

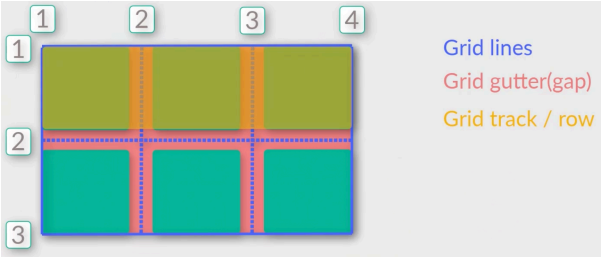


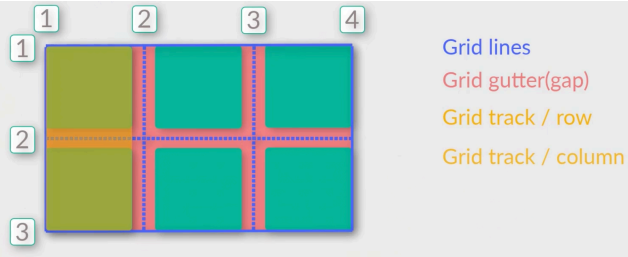




Grid lines are vertical and horizontal lines that divide the CSS Grid and separate the rows and columns.
The number of columns and rows grid lines are always one greater than the total number of rows and columns. In example on the left we have a 3 column by 2 row in which case the grid lines would be 4 by 3.

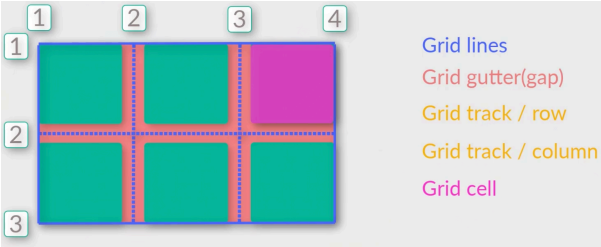
With the use of grid lines we are able to move grid items to different positions. We can also create space between grid items horiztonally and vertically and this space is known as the grid gutter (i.e. the gap between rows and columns). Therefore we have grid-column gap going vertically and grid-row gap going horizontally.

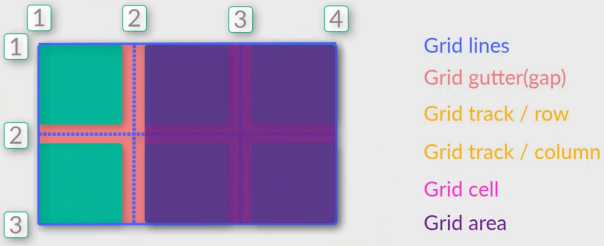
Grid items with grid gaps are called grid-track and so in the case of the horizontal grid track we have what is called the grid track row and for the vertical grid track we have what is called the grid track column.




The single unit of the CSS grid is called a grid cell while any area of the grid is called the grid area.




We need to set the display property to grid or grid-inline. The difference between grid and grid-inline values are almost similar to to the difference between flex and in-line flex values for Flexbox.

Setting the display value to grid is not enough and we need to set other property values for the rows and columns in order to setup the CSS grid.

Below are the various properties that we can use with grid-container and grid-items:

| Grid-Container Property | Description |
|---|---|
| grid-template-columns | Defines the number and the size of the columns e.g. 150px 150px 150px is three columns with the size of 150px each |
| grid-template-rows | Define the number and the size of rows e.g. 150px 150px is two rows with the size of 150px each |
| grid-template-areas | Specifies areas within the grid layout. Name gird items using the grid-area property and then reference to the name in the grid-template-area property |
| grid-template | Shorthand syntax for grid-template-columns, grid-template-row and grid-template-area properties |
| grid-column-gap | Create a grid gap between the columns |
| grid-row-gap | Create a grid gap between the rows |
| grid-gap | Shorthand syntax for grid-column-gap and grid-row-gap properties |

| | |
|---|---|
| justify-items | Align grid-items horizontally inside the cells and by default it is set to stretch |
| align-items | Align grid-items vertically inside the cells and by default it is set to stretch |
| justify-content | Align the grid-track horizontally |
| align-content | Align the grid-track vertically |
| grid-auto-columns | Define the column size of the implicit grid |
| grid-auto-rows | Define the row size of the implicit grid |
| grid-auto-flow | Change the direction and transform implicit rows into implicit columns which by default the value is set to row |

| Gird-Item Properties | Description |
|---|---|
| grid-column-start | Position grid-item horizontally (column) by defining the grid line number for the start position |
| grid-column-end | Position grid-item horizontally (column) by defining the grid line number for the end position |
| grid-column | Shorthand syntax for grid-column-start and grid-column-end properties |
| grid-row-start | Position grid-item vertically (row) by defining the grid line number for the start position |
| grid-row-end | Position grid-item vertically (row) by defining the grid line number for the end position |
| grid-row | Shorthand syntax for grid-row-start and grid-row-end properties |
| grid-area | Shorthand syntax for grid-column-start, grid-column-end, grid-row-start and grid-row-end properties.<br><br>The grid-area property can also be used to assign a name to a grid-item and the grid item can be referenced by the grid-template-areas property |
| justify-self | Align individual grid-item horizontally |
| align-self | Align individual grid-item vertically |
| order | Sets the order to lay out items in the grid-container. Grid-items are sorted by ascending order value and then by their source code oder |

**Disclaimer**: As at May 2020, Mozilla Firefox has the best developer tool for CSS Grid and is recommended to use to understand how CSS Grid works in a clear visual way. This browser allows us to display an overlay grid on our webpage to clearly see the CSS Grid as well as the grid lines and numbers for the lines. This is a feature only available on Mozilla Firefox.

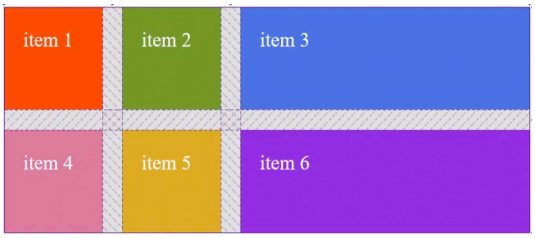
**Important**: At the moment grid gap properties apply the same size of gaps to all column/row and we cannot add different gap sizes for individual column/row. For this we would need to apply different techniques and tricks.

## 3.2 Fractional Units

When defining the grid-template-columns and grid-template-rows values we can use any measurement units such as px, rem, em, etc.

The auto value allows the column/row to take up the available width.

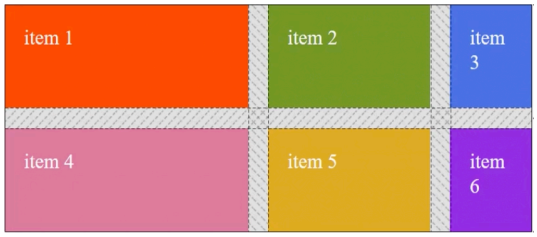
For example, if we have a three column grid and want the last column take up the remaining width available the auto value can be used e.g 150px 150px auto;




If two columns have an auto value and the third column has a measurement value, the two auto value columns will equally take the available width e.g. 150px auto auto;


Instead of using the auto measurement values there is a special measurement unit created for CSS Grid called fractional unit. This is a more flexible measurement unit. Therefore in the second example we can write the value as 150px 1fr 1fr; to achieve the same result.


This unit allows us to divide space into a fraction of the available space. We can manipulate on the size of columns using fractional units for example, suppose we want the second column to be twice as big compared to the first and last column the value would be represented as such: 1fr 2fr 1fr;




In the example of value: 3fr 2fr 1fr; the first column is three times bigger than the last column and the second column is twice as large as the last column. This demonstrates the flexibility and ease of using fractional units with CSS Grid.

The fractional units can be used with both the grid-template-column and the grid-template-rows. Where the units are exactly the same the repeat( ) function can be used to avoid repeating the values over again. This function takes in two arguments the first being the number of repeat and the second the measurement unit e.g repeat(3, 1fr);
Where we want the first column to be a different size and the other two columns to be the same we can write the value as an example: 50% repeat(2, 1fr);

## 3.3 Positioning and Spanning Grid Items

To position grid items in the CSS Grid the grid-line numbers is required. There are a few properties that allow you to position grid items.

The grid-column-start and grid-column-end defines the column line number for the start and end position of the element to be positioned within the grid. These two properties must be used together to define the two positions across the column horizontally.

The grid-row-start and grid-row-end defines the row line number for the start and end position of the element to be positioned within the grid. These two properties must be used together to define the two positions across the row vertically.

The shorthand syntax grid-column and grid-row allows us to easily define the start and end position with few lines of code. The forward slash ( / ) is used to separate the start position and end position line numbers in that order.

```
.item-2 {
    background-color: olivedrab;
    grid-column: 2 / 3;
    grid-row: 2 / 3;
}
```

The grid-area is the shorthand for combining the column and row positions together. The first and second values defines the start position for the column and row while the third and last values define the end positions for the column and row, all separated with a forward slash ( / ).

```
.item-2 {
    background-color: olivedrab;
    grid-area: 2 / 2 / 3 / 3;
}
```

The items that have been defines positions will maintains its position while other items will move around to fit the available space in the grid. The z-index allows us to bring certain elements to the front where elements on the grid overlap.

To stretch an item to the end of the container, rather than writing the specific line number we can use -1 as the value instead. This will span the item across to the end of the container (column or row) regardless if the column grid size was to change later on. This allows the code to be flexible and dynamic. The -1 value only works with the explicit grid.

An alternative approach to stretching an item is to use the span keyword followed by the number of columns/rows to span across. The count will start from the next grid line i.e. line 2 will equal 1 counting up.

```
.item-6 {
    background-color: blueviolet;
    grid-column: 1 / span 2;
}
```

In the example to the left, the column spans across 2 columns. The item therefore stretches to fit between line 1 to line 3 of the grid.

## 3.4 Naming Grid Items and Grid Area

We are able to name grid lines and use those names rather than the grid line numbers. This is an alternative way to positioning elements within the CSS Grid.

To define the names of the columns/rows we would use the square brackets followed by

```
grid-template-rows: [header-start] 1fr [header-end main-start] 1fr 1fr [main-end box-start] 1fr [box-end footer-start] 1fr [footer-end];
grid-template-columns: repeat(4, [col-start] 1fr [col-end]);
```

the name of the column line. This is all done within the grid-template-rows values, see below examples:
We can name the lines for either the columns or rows and also provide them with multiple names. We can then reference these names rather than using the line numbers.

```
.header {
    grid-column: col-start 1 / col-end -1;
    grid-row: header-start / header-end;
}
```

```
.main {
    grid-column: col-start 2 / col-end -1;
    grid-row: main-start / main-end;
}
```

```
.box-1 {
    grid-column: col-start 2 / col-end 2;
    grid-row: box-start / box-end;
}
```

**Disclaimer**: This is an alternative method for positioning which is exactly the same as using the line numbers. We are able to use this method also with the repeat function. It is all personal preference whether using line numbers are line names is easier and convenient for the developer.

The final alternative method for positioning grid items is by naming grid areas. To create a grid area the grid-template-area property is required. This allows us to give a name to each cell inside of the defined grid container. The name's of each cell must be placed between double quotation marks.

```
grid-template-rows: repeat(5, 1fr);
grid-template-columns: repeat(4, 1fr);
grid-template-areas: "header header header ."
                     "sidebar main main main"
                     "sidebar main main main"
                     "sidebar box-1 box-2 box-3"
                     "footer footer footer footer";
```

In the example, the names can visually represent the grid areas. This is building the entire page layout using the area names. This requires every cell to be given a name otherwise it will not work.

To maintain one cell to be empty this requires a period ( . ) in its place.

```
.header {
    grid-area: header;
}
```

The element requires the grid-area property in order to position the element to the grid area name that was defined in the grid-template-area property.

**Disclaimer**: This method is convenient when you already know the layout required for the webpage and can allow you to write very little CSS code.

## 3.5 Explicit and Implicit Grid

A explicit grid is a grid that is explicitly defined using the grid-template properties. The implicit grid is a grid that is not defined by us but is automatically created to fit content onto the gird container where there is no more space available on the explicit grid.

To manipulate and control a implicit grid there are a few properties in that can be used for example grid-auto-columns and grid-auto-rows are used to define the size of the implicit grid. If we set the grid-auto-flow from the default row value to column the grid-auto-rows property will no longer work and we should use the grid-auto-column property to define the size of the implicit columns instead.

## 3.6 Aligning Grid Items and Grid Tracks

In CSS Grid there are almost similar properties to align elements on the Grid compared to Flexbox.

The justify-items property is used to align items horizontally in a cell and by default is set to stretch. There are three other properties it can take i.e. start, centre and end. The grid-item will take up the width that is required to display its content and then it will move according to value, other than stretch, that is set as the value. The start aligns the grid-item to the left side while the end aligns to the right side of the grid cell.

The align-items works exactly the same as the justify-items property but aligns the grid-items in the vertical direction. The grid-item will take up the necessary height that is required to display its content before it will move using the start, center or end values.

Both alignment properties work across all grid-items. To specify for individual grid-items we would have to use the justify-self and align-self properties. This property would be applied to the individual element we wish to change the alignment on either the horizontal or vertical axis of the grid-cell. This will override the global alignment from the justify-items and align-items properties.

To align the grid-track itself in the grid-container there are two properties justify-content and align-content. These properties can take different values such as start, center, end, space-between, space-around and space-evenly. This is similar to Flexbox and will align the whole grid-track in the container space horizontally/vertically.

## 3.7 max-content, min-content and minmax( )

The max-content, min-content and minmax( ) function are another measurement unit we can use when defining the grid-template-columns and grid-template-rows properties. This can be used with the other measurement units including the fraction units.

The max-content will make the grid-item to take up the maximum width/height required to display the content of the grid item.

The min-content will make the grid-item to take up the minimum width/height required to display the content of the grid item and will wrap the content. For example, a grid-item with text, the minimum width will be the largest world and the rest of the text content will wrap inside the grid-cell. This is the main difference between min-content and max-content properties.

The minmax( ) function allows us to define a range of sizes of rows and columns and the size will be accessible depending on the various situations. This takes in two arguments the first is the minimum range value and the second is the maximum range value.

If there is enough space then the grid-item will always take up the maximum value and as soon as there is not enough space then the width of the column starts to decrease. Once the content reaches the minimum value the content will maintain the minimum value and will not shrink anymore.

---

## 3.8 Auto-fill and Auto-fit

The auto-fill and auto-fit values can be used when defining the grid-template-columns and grid-template-rows properties and allows us to auto-size columns inside a container.

```
grid-template-columns: repeat(auto-fill, minmax(80px, 1fr));
```

The auto-fill value fills the row with as many column as it can fit. This creates implicit columns whenever a new column can fit.

```
grid-template-columns: repeat(auto-fit, minmax(80px, 1fr));
```

The auto-fit value will fit the available columns inside the container so that they take up any available space. This will not create any empty columns. This allows us to make our grid layout responsive because when there is not enough space the items will start wrapping and the number of columns will decrease as the browser screen size decreases.

The minmax( ) function can also be used with the auto-fit value.

**Disclaimer**: In many cases the auto-fill is not that much useful compared to the auto-fit value which is much more useful and powerful when creating responsive layouts.

**END**