# HTML

# CSS

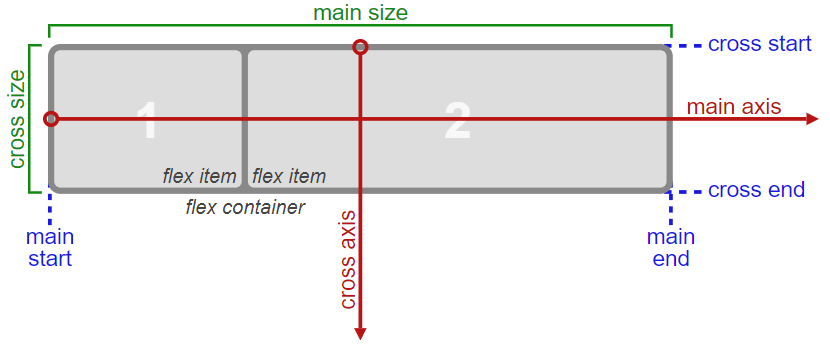
## Flex-box

Placing the CSS property **display: flex**; on an element allows you to use other flex properties to build a responsive page.

Adding **display: flex** to an element turns it into a flex container. This makes it possible to align any children of that element into rows or columns. You do this by adding the **flex-direction** property to the parent item and *setting it* **to row or column**. Creating a row will align the children horizontally, and creating a column will align the children vertically.

Other options for flex-direction are row-reverse and column-reverse.

Note: The default value for the flex-direction property is row.



for each(row & column), the direction the flex items are arranged is called the main axis.

For **a row**, this is a horizontal line that cuts through each item. And for **a column**, the main axis is a vertical line through the items.

There are several options for how to space the flex items along the line that is the main axis. One of the most commonly used is **justify-content: center;**, which aligns all the flex items to the center inside the flex container.

Others options include:

**flex-start**: aligns items to the start of the flex container. For a row, this pushes the items to the left of the container. For a column, this pushes the items to the top of the container. This is the default alignment if no justify-content is specified.

**flex-end:** aligns items to the end of the flex container. For a row, this pushes the items to the right of the container. For a column, this pushes the items to the bottom of the container.

**space-between**: aligns items to the center of the main axis, with extra space placed between the items. The first and last items are pushed to the very edge of the flex container. For example, in a row the first item is against the left side of the container, the last item is against the right side of the container, then the remaining space is distributed evenly among the other items.

**space-around**: similar to space-between but the first and last items are not locked to the edges of the container, the space is distributed around all the items with a half space on either end of the flex container.

**space-evenly**: Distributes space evenly between the flex items with a full space at either end of the flex container.

The **align-items** property is similar to *justify-content*.

The different values available for **align-items** include:

**flex-start**: aligns items to the start of the flex container. For rows, this aligns items to the top of the container. For columns, this aligns items to the left of the container.

**flex-end**: aligns items to the end of the flex container. For rows, this aligns items to the bottom of the container. For columns, this aligns items to the right of the container.

**center**: align items to the center. For rows, this vertically aligns items (equal space above and below the items). For columns, this horizontally aligns them (equal space to the left and right of the items).

**stretch**: stretch the items to fill the flex container. For example, rows items are stretched to fill the flex container top-to-bottom. This is the default value if no align-items value is specified.

**baseline**: align items to their baselines. Baseline is a text concept, think of it as the line that the letters sit on.

**flex-wrap** property tells CSS to wrap items. This means extra items move into a new row or column. The break point of where the wrapping happens depends on the size of the items and the size of the container.

CSS also has options for the direction of the wrap:

**nowrap**: this is the default setting, and does not wrap items.

**wrap**: wraps items from left-to-right if they are in a row, or top-to-bottom if they are in a column.

**wrap-reverse**: wraps items from right-to-left if they are in a row, or bottom-to-top if they are in a column.

the **flex-shrink** property. When it's used, it allows an item *to shrink if the flex container is too small*. Items shrink when the width of the parent container is smaller than the combined widths of all the flex items within it.

The *flex-shrink property takes numbers as values*. The higher the number, the more it will shrink compared to the other items in the container. For example, if one item has a flex-shrink value of 1 and the other has a flex-shrink value of 3, the one with the value of 3 will shrink three times as much as the other.

The opposite of flex-shrink is **the flex-grow** property. The flex-grow property controls the size of items when the parent container expands.

The **flex-basis** property specifies the initial size of the item before CSS makes adjustments with flex-shrink or flex-grow.

The units used by the flex-basis property are the same as other size properties (px, em, %, etc.). The value auto sizes items based on the content.

There is a shortcut available to set several *flex properties at once*. The **flex-grow**, **flex-shrink**, and **flex-basis** properties can all be set together by using the **flex** property.

The default property settings are **flex: 0 1 auto**;

The **order** property is used to tell CSS the order of how flex items appear in the flex container. By default, items will appear in the same order they come in the source HTML. The property takes numbers as values, and negative numbers can be used.

The final property for flex items is **align-self**. This property allows you to adjust each item's alignment individually, instead of setting them all at once. This is useful since other common adjustment techniques using the *CSS properties float, clear, and vertical-align do not work on flex items*.

**align-self** accepts the same values as **align-items** and will override any value set by the align-items property.

## Grid

Turn any HTML element into a grid container by setting its display property to grid. This gives you the ability to use all the other properties associated with CSS Grid.

Note: In CSS Grid, the parent element is referred to as the container and its children are called items.

**grid-template-columns** property on a grid container as demonstrated below:

.container {

display: grid;

grid-template-columns: 50px 50px;

}

To adjust the rows manually, use the **grid-template-rows** property in the same way you used **grid-template-columns**

**fr:** sets the column or row to a fraction of the available space,

**auto**: sets the column or row to the width or height of its content automatically,

**%**: adjusts the column or row to the percent width of its container.

To add a gap between the columns, use the **grid-column-gap, grid-row-gap** property like this:

grid-column-gap: 10px;

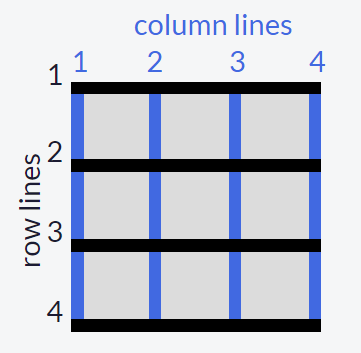
grid-row-gap: 5px;

**grid-gap** is a shorthand property for **grid-row-gap** and **grid-column-gap** that's more convenient to use. If **grid-gap** has one value, it will create a gap between all rows and columns. However, if there are two values, it will use the first one to set the gap between the rows and the second value for the columns.

The **grid-column** property is the first one for use on the grid items themselves.

The hypothetical horizontal and vertical lines that create the grid are referred to as lines. These lines are numbered starting with 1 at the top left corner of the grid and move right for columns and down for rows, counting upward.

This is what the lines look like for a 3x3 grid:



To control the amount of columns an item will consume, you can use the **grid-column** property in conjunction with the line numbers you want the item to start and stop.

Here's an example:

grid-column: 1 / 3;

This will make the item start at the first vertical line of the grid on the left and span to the 3rd line of the grid, consuming two columns.

you can make items consume multiple rows just like you can with columns, using the **grid-row** property on a grid item.

In CSS Grid, the content of each item is located in a box which is referred to as a cell. You can align the content's position within its cell horizontally using the **justify-self** property on a grid item. By *default*, this property has a value of *stretch*, which will make the content fill the whole width of the cell. This CSS Grid property accepts other values as well:

start: aligns the content at the left of the cell,

center: aligns the content in the center of the cell,

end: aligns the content at the right of the cell.

Just as you can align an item horizontally, there's a way to align an item vertically as well. To do this, you use the **align-self** property on an item. This property accepts all of the same values as justify-self

Align All Items Horizontally using **justify-items**.

**Align-items** property on *a grid container* will set the vertical alignment for *all the items in our grid*.

You can **group cells** of your grid together into an area and give the area a custom name. Do this by using **grid-template-areas** on the container like this:

grid-template-areas:

"header header header"

"advert content content"

"footer footer footer";

The code above merges the top three cells together into an area named header, the bottom three cells into a footer area, and it makes two areas in the middle row; advert and content. Note: Every word in the code represents a cell and every pair of quotation marks represent a row. In addition to custom labels, you can use a period (.) to designate an empty cell in the grid.

After creating an area's template for your grid container, you can place an item in your custom area by referencing the name you gave it. To do this, you use **the grid-area** property on an item like this:

.item1 {

grid-area: header;

}

*This lets the grid know that you want the item1 class to go in the area named header*. In this case, the item will use the entire top row because that whole row is named as the header area.

Use grid-area Without Creating an Areas Template:

The grid-area property can be used in another way. If your grid doesn't have an areas template to reference, you can create an area on the fly for an item to be placed like this:

item1 { grid-area: 1/1/2/4; }

This is using the line numbers you learned about earlier to define where the area for this item will be. The numbers in the example above represent these values:

grid-area: horizontal line to start at / vertical line to start at / horizontal line to end at / vertical line to end;

So the item in the example will consume the rows between lines 1 and 2, and the columns between lines 1 and 4.

the **repeat** function to specify the number of times you want your column or row to be repeated, followed by a comma and the value you want to repeat.

grid-template-rows: repeat(100, 50px);

You can also repeat multiple values with the repeat function and insert the function amongst other values when defining a grid structure. Here's what that looks like:

grid-template-columns: repeat(2, 1fr 50px) 20px;

This translates to:

grid-template-columns: 1fr 50px 1fr 50px 20px;

**Limit Item Size Using the minmax Function**

There's another built-in function to use with grid-template-columns and grid-template-rows called **minmax.** It's used to limit the size of items when the grid container changes size. To do this you need to specify the acceptable size range for your item. Here is an example:

grid-template-columns: 100px minmax(50px, 200px);

In the code above, grid-template-columns is set to create two columns; the first is 100px wide, and the second has the minimum width of 50px and the maximum width of 200px.

**Create Flexible Layouts Using auto-fill**

The repeat function comes with an option called auto-fill. This allows you to automatically insert as many rows or columns of your desired size as possible depending on the size of the container. You can create flexible layouts when combining auto-fill with minmax, like this:

repeat(auto-fill, minmax(60px, 1fr));

When the container changes size, this setup keeps inserting 60px columns and stretching them until it can insert another one. Note: If your container can't fit all your items on one row, it will move them down to a new one.

**Create Flexible Layouts Using auto-fit**

**auto-fit** works almost identically to **auto-fill**. The only difference is that when the container's size exceeds the size of all the items combined, auto-fill keeps inserting empty rows or columns and pushes your items to the side, while auto-fit collapses those empty rows or columns and stretches your items to fit the size of the container.

Note: If your container can't fit all your items on one row, it will move them down to a new one.