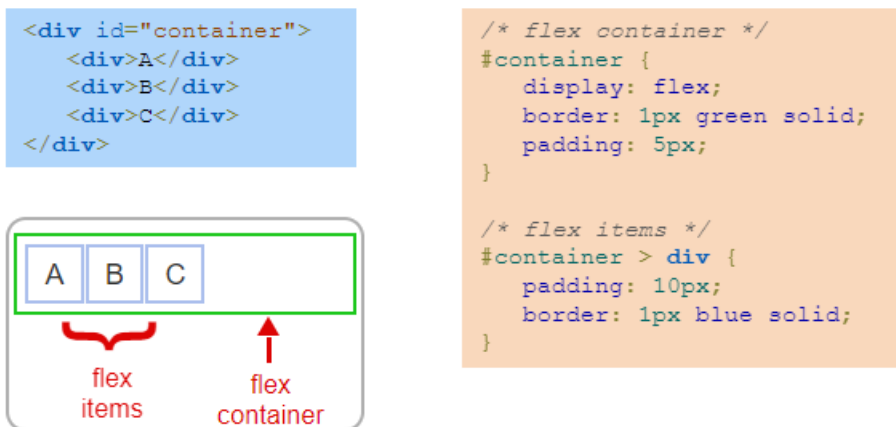


# Flexbox

## Flexbox container and items

The **Flexible Box** or **flexbox** is a CSS layout mode that provides an efficient way to lay out elements in a container so the elements behave predictably when the container is resized or viewed on different screen sizes.

A **flex container** is an element that has the CSS property `display` set to `flex` to create a block-level flex container or `inline-flex` to create an inline flex container. Ex: `<div style="display: flex">`. Flex containers hold flex items. A **flex item** is a child element of a flex container that is positioned and sized according to various CSS flexbox properties.

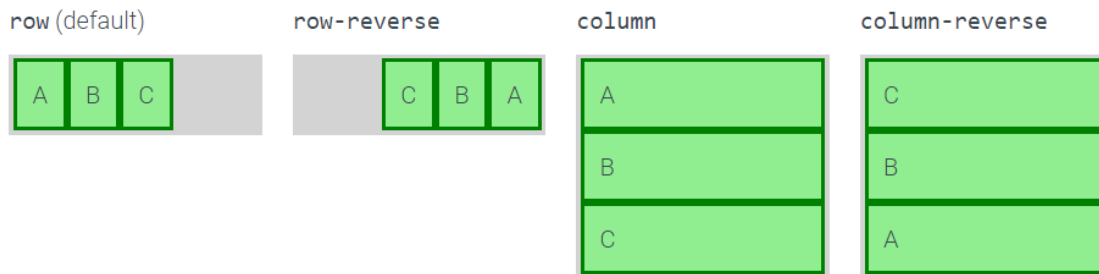


1. Without any CSS, the A, B, and C div elements display vertically, each filling the browser width.
2. Setting the CSS display property to "flex" makes the outer div the flex container. The flex items now display on the same row.
3. The flex items have padding and blue borders.

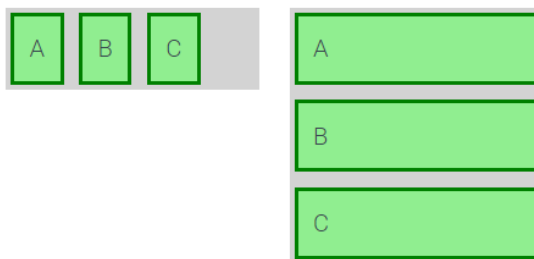
## Flex container properties

Several CSS properties modify the default behavior of a flex container:

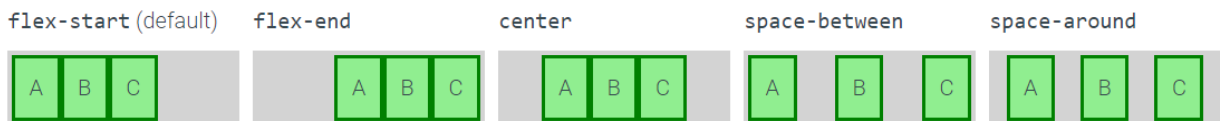
- The ***flex-direction*** property defines the direction of flex items within the container using values:



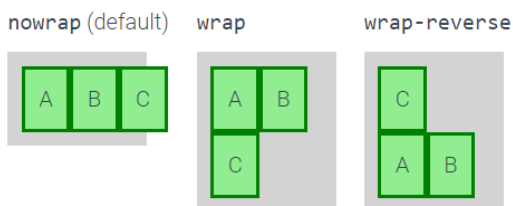
- The ***gap*** property defines the space between flex items. Ex: **gap: 10px;** puts a 10px gap between all items.



- The ***justify-content*** property justifies the flex items within the container using values:



- The ***flex-wrap*** property determines if or how flex items wrap onto multiple rows when the container is not wide enough to hold all items, using values:




## Flex item properties

A flex item's width is determined by the combination of three CSS properties:

- The ***flex-basis*** property sets the initial length of a flex item. The values can be `auto` (the default), a percentage, or a length unit. The default value `auto` makes the flex item the same initial length as the content.
- The ***flex-grow*** property sets a proportion that determines how much of the available container space should be assigned to the item. The default is 0, meaning the size should be based on the item's content.
- The ***flex-shrink*** property sets a proportion that determines the item's minimum size. The default is 1, meaning the size should shrink at the same rate as other items when the container width shrinks. A value of 0 means the item should not change sizes when the container width shrinks.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="products.html">Products</a></li>
    <li><a href="about.html">About</a></li>
  </ul>
</nav>
```



- Home
- Products
- About

A website's navigation links are displayed in an unordered list.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="products.html">Products</a></li>
    <li><a href="about.html">About</a></li>
  </ul>
</nav>
```

```
nav ul {
  display: flex;
  list-style-type: none;
  padding: 0;
}
```

Home Products About

Making the ul element a flex container places the nav links on the same row.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="products.html">Products</a></li>
    <li><a href="about.html">About</a></li>
  </ul>
</nav>
```

```
nav ul {
  display: flex;
  list-style-type: none;
  padding: 0;
}
```

Home Products About

```
nav li {
  background-color: gold;
  text-align: center;
}
```

Home Products About

By default, the li elements have flex-basis:auto and flex-grow:0, so li elements are only as wide as the item's content.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="products.html">Products</a></li>
    <li><a href="about.html">About</a></li>
  </ul>
</nav>
```

```
nav ul {
  display: flex;
  list-style-type: none;
  padding: 0;
}
```

```
nav li {
  flex-grow: 1;
  background-color: gold;
  text-align: center;
}
```

Home Products About

Home Products About

Changing flex-grow from the default 0 to 1 gives all li elements the same proportion.  
The elements fill the flex container.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="products.html">Products</a></li>
    <li><a href="about.html">About</a></li>
  </ul>
</nav>
```

```
nav ul {
  display: flex;
  list-style-type: none;
  padding: 0;
}
```

```
nav li {
  flex-grow: 1;
  background-color: gold;
  text-align: center;
}
```

```
nav li {
  flex-basis: 100px;
  background-color: gold;
  text-align: center;
}
```

Home Products About

Home Products About

Home Products About

Replacing "flex-grow:1" with "flex-basis:100px" makes each li element 100px wide.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="products.html">Products</a></li>
    <li><a href="about.html">About</a></li>
  </ul>
</nav>
```

```
nav ul {
  display: flex;
  list-style-type: none;
  padding: 0;
}
```

Home Products About

```
nav li {
  flex-grow: 1;
  background-color: gold;
  text-align: center;
}
```

Home Products About

```
nav li {
  flex-basis: 100px;
  background-color: gold;
  text-align: center;
}
```

Home Products About

Resizing the browser changes the container size. When the container shrinks, the li elements shrink to fill the available space.

```
<nav>
  <ul>
    <li><a href="index.html">Home</a></li>
    <li><a href="products.html">Products</a></li>
    <li><a href="about.html">About</a></li>
  </ul>
</nav>
```

```
nav ul {
  display: flex;
  list-style-type: none;
  padding: 0;
}
```

Home Products About

```
nav li {
  flex-grow: 1;
  background-color: gold;
  text-align: center;
}
```

Home Products About

```
nav li {
  flex-basis: 100px;
  flex-shrink: 0;
  background-color: gold;
  text-align: center;
}
```

Home Products Abo

Changing flex-shrink from the default 1 to 0 prevents the li elements from shrinking when the browser is resized.

## The flex property

The shorthand property ***flex*** specifies `flex-grow`, `flex-shrink`, and `flex-basis` together. Ex: `flex: 0 1 auto;` is the same as `flex-grow: 0;` `flex-shrink: 1;` `flex-basis: auto;`.

1.

```
<body>
  <header>Header</header>

  <!-- Flexbox layout -->
  <div id="container">
    <nav>Nav</nav>
    <main>Main</main>
    <aside>Aside</aside>
  </div>

  <footer>Footer</footer>
</body>
```

```
#container {
  display: flex;
}

nav {
  flex: 0 1 20%;
}

main {
  flex: 0 1 60%;
}

aside {
  flex: 0 1 20%;
}
```

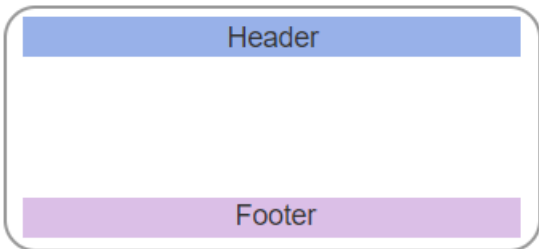
`<header>` and `<footer>` span the entire width of `<body>`, but the `<div>` is a flex container that displays the flex items on the same row.

2.

```
<body>
  <header>Header</header>

  <!-- Flexbox layout -->
  <div id="container">
    <nav>Nav</nav>
    <main>Main</main>
    <aside>Aside</aside>
  </div>

  <footer>Footer</footer>
</body>
```



```
#container {
  display: flex;
}

nav {
  flex: 0 1 20%;
}

main {
  flex: 0 1 60%;
}

aside {
  flex: 0 1 20%;
}
```

flex-grow

<nav>, <main>, and <aside> all have flex-grow = 0, so all three flex items' width should be based on each item's content.

3.



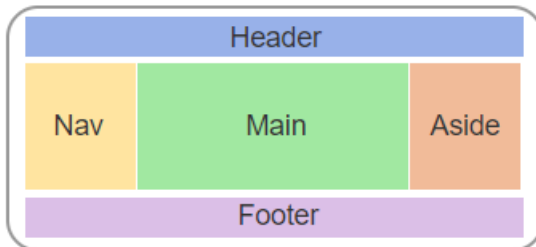
```

<body>
  <header>Header</header>

  <!-- Flexbox layout -->
  <div id="container">
    <nav>Nav</nav>
    <main>Main</main>
    <aside>Aside</aside>
  </div>

  <footer>Footer</footer>
</body>

```



```

#container {
  display: flex;
}

nav {
  flex: 0 1 20%;
}

main {
  flex: 0 1 60%;
}

aside {
  flex: 0 1 20%;
}

```

flex-grow      flex-shrink      flex-basis

<nav> occupies 20% of the row, <main> occupies 60%, and <aside> occupies 20%.  
 20% + 60% + 20% = 100% of the row.

4.

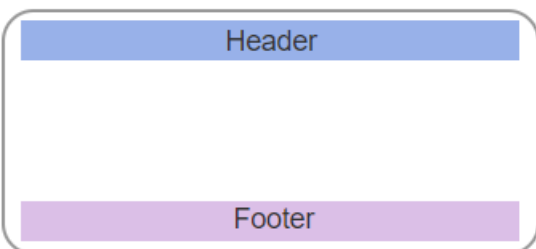
```

<body>
  <header>Header</header>

  <!-- Flexbox layout -->
  <div id="container">
    <nav>Nav</nav>
    <main>Main</main>
    <aside>Aside</aside>
  </div>

  <footer>Footer</footer>
</body>

```



```

#container {
  display: flex;
}

nav {
  flex: 0 1 20%;
}

main {
  flex: 0 1 60%;
}

aside {
  flex: 0 1 20%;
}

```

flex-grow      flex-shrink

## Grid container and grid items

**Grid layout** is a CSS layout mode that divides a webpage into a rectangular grid in which to position page elements. Grid layout is ideal for designing two-dimensional webpage layouts.

A **grid container** is an element that has the CSS property `display` set to `grid` to create a block-level grid container or `inline-grid` to create an inline grid container. Ex: `<div style="display: grid">`. A **grid item** is a child element of a grid container that is by default placed into a single grid cell.

The **grid-template-columns** property defines the grid container's number of columns and optionally the width of each column. Ex: `grid-template-columns: 50px 90px auto auto;` specifies 4 values that create 4 columns: the first is 50px wide, the second is 90px wide, and the third and fourth columns are automatically sized to fit the remainder of the grid width.

```
<div id="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
```

```
#grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
}

#grid-container > div {
  text-align: center;
  background: lightgreen;
  border: 3px solid green;
  padding: 20px;
}
```

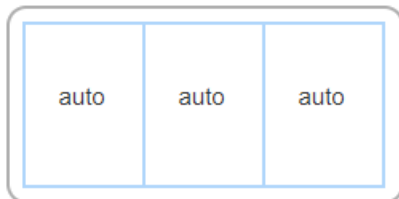


The "display: grid;" declaration makes the `<div>` with id `grid-container` a block-level grid.

```
<div id="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
```

```
#grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
}

#grid-container > div {
  text-align: center;
  background: lightgreen;
  border: 3px solid green;
  padding: 20px;
}
```

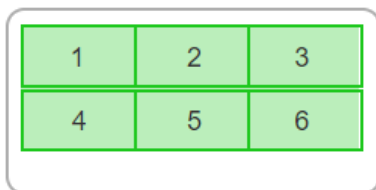


The grid-template-columns property is assigned 3 "auto" values, so the grid container will contain 3 equally-sized columns.

```
<div id="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
```

```
#grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
}

#grid-container > div {
  text-align: center;
  background: lightgreen;
  border: 3px solid green;
  padding: 20px;
}
```



Six <div> elements are children of the grid container, so each element becomes a grid item. The div child selector puts a green border around each grid item.

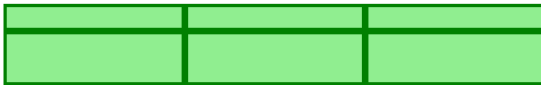
## Controlling the grid container

The default behavior of a grid container can be modified with various CSS properties:

- The **gap** property defines the gap between each grid row and column.  
Ex: `gap: 5px 25px;` puts a 5px gap between each row and a 25px gap between each column.



- The **grid-template-rows** property defines the height of each row. Ex: `grid-template-rows: 20px 40px;` makes the first row 20px tall and the second row 40px tall.



- The **justify-content** property horizontally aligns the grid items inside the grid container using values:
  - start** - Aligns grid flush with the grid container's starting edge.



- end** - Aligns grid flush with the grid container's ending edge.



- center** - Aligns grid in the center of the grid container.



- stretch** - Stretches the grid items to fill the grid container width.



- space-around** - Places equal spacing between grid items with half the space on either side of the grid container.



- space-between** - Places equal spacing between grid items with no space on either side of the grid container.

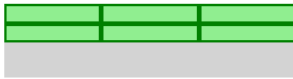


- space-evenly** - Places equal spacing between grid items, including the sides of the grid container.

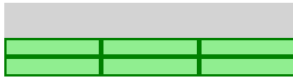


- The **align-content** property vertically aligns the grid items inside the grid container using values:

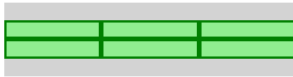
- **start** - Aligns grid flush with the grid container's starting edge.



- **end** - Aligns grid flush with the grid container's ending edge.



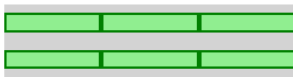
- **center** - Aligns grid in the center of the grid container.



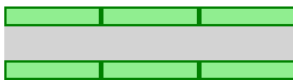
- **stretch** - Stretches the grid items to fill the grid container height.



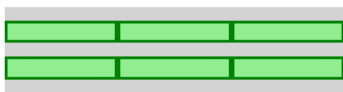
- **space-around** - Places equal spacing between grid items with half the space on either side of the grid container.



- **space-between** - Places equal spacing between grid items with no space on either side of the grid container.



- **space-evenly** - Places equal spacing between grid items, including the sides of the grid container.

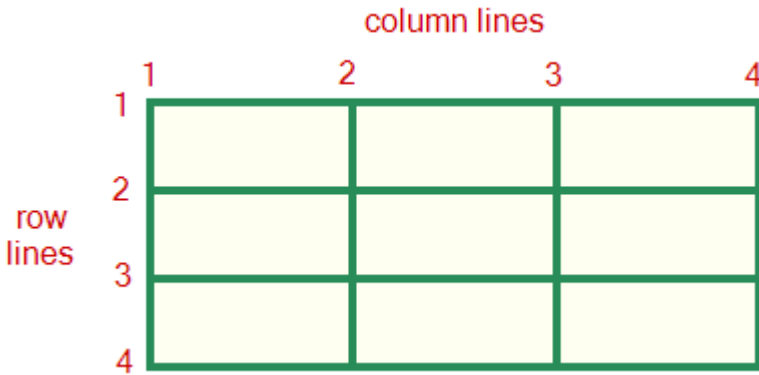


The `justify-content` and `align-content` properties have no effect unless the grid width or height is less than the grid container's width or height.

## Controlling grid item placement

A grid item by default appears in a single row and column based on the ordering of the grid item within the grid container. However, grid items may be positioned at specific grid locations using the column line and row line numbers as illustrated in the figure below.

Figure Row and column lines.

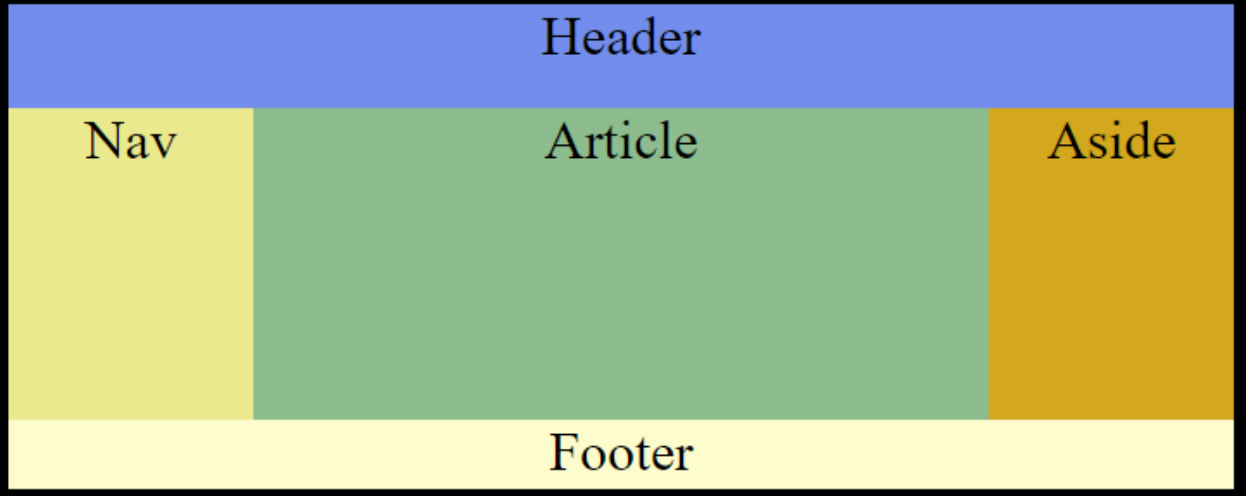


A grid item may be placed in a specific row or column or span multiple rows and/or columns using various following CSS properties:

- The ***grid-row*** property lists the grid item's starting and ending row line numbers. Ex: `grid-row: 1 / 3;` makes the grid item start at row line 1 and end at row line 3, so the grid item spans 2 rows.
- The ***grid-column*** property lists the grid item's starting and ending column line numbers. Ex: `grid-column: 1 / 4;` makes the grid item start at column line 1 and end at column line 4, so the grid item spans 3 columns.
- The ***grid-area*** property lists the grid item's starting and ending row and column numbers. Ex: `grid-area: 1 / 2 / 3 / 4;` makes the grid item start at row line 1 and column line 2 and end at row line 3 and column line 4, so the grid item spans 2 rows and 2 columns.

### Naming grid items

- Grid items may be assigned names with the `grid-area` property. The grid container's ***grid-template-areas*** property specifies the grid layout using the named grid items.
- Figure : Layout goal for the Participation Activity below.



.