# FLEX

Flexbox layout is a powerful tool in CSS for creating flexible and responsive layouts. It provides a more efficient way to distribute space among items in a container, even when their size is unknown or dynamic. Here are some important CSS flex properties along with their use cases and examples:

## 1. display: flex

### Definition:

This property establishes a flex container, enabling a flex context for its direct children. It turns the element into a flex container, allowing you to use other flex properties to control the layout of its children.

Example:
HTML:

```
<div class="container">
  <div class="item">Item 1</div>
  <div class="item">Item 2</div>
  <div class="item">Item 3</div>
</div>
```

CSS:

```
.container {
  display: flex;
}
```

### Use Case:

Use display: flex; when you want to create a flex container to arrange its direct children along a flex line.

### Notes:

This property is applied to the parent container.
It enables Flexbox layout for its direct children.

## 2. flex-direction

### Definition:

Determines the direction of the main axis along which flex items are laid out within a flex container. It specifies whether the flex items are laid out in a row or column.
Example:

```
.container {
  display: flex;
  flex-direction: column;
}
```

### Use Case:

Use flex-direction to control the flow direction of flex items within a flex container, altering the main axis direction to suit layout requirements.

### Notes:

This property affects the layout of flex items along the main axis.
It determines the direction in which flex items are arranged.

### Values:

- row: Items are placed in a horizontal line.
- row-reverse: Items are placed in a horizontal line in reverse order.
- column: Items are placed in a vertical line.
- column-reverse: Items are placed in a vertical line in reverse order.

## 3. flex-wrap

### Definition:

Controls whether flex items are forced into a single line or can be wrapped onto multiple lines within a flex container. It specifies whether

flex items are allowed to wrap when there is not enough room on the main axis.
Example:

```
.container {
  display: flex;
  flex-wrap: wrap;
}
```

### Use Case:

Use flex-wrap when you need to control how flex items should behave when there's not enough space along the main axis, allowing them to wrap onto multiple lines or remain in a single line.

### Notes:

This property affects the layout of flex items when they exceed the container's width.

### Values:

- nowrap: Items are all placed in a single line.
- wrap: Items wrap onto multiple lines if needed.
- wrap-reverse: Items wrap onto multiple lines in reverse if needed.

## 4. justify-content

### Definition:

Defines how flex items are aligned along the flex container's main axis. It determines the distribution of space between and around content items along the main axis of a flex container.

Example:

```
.container {
  display: flex;
  justify-content: space-between;
}
```

### Use Case:

Use justify-content to control the alignment and spacing of flex items along the main axis, adjusting their position to achieve the desired layout.

### Notes:

This property distributes space between and around flex items along the main axis.

### Values:

- flex-start: Items are positioned at the start of the container.
- flex-end: Items are positioned at the end of the container.
- center: Items are positioned at the center of the container.
- space-between: Items are evenly distributed with the first item at the start and the last at the end.
- space-around: Items are evenly distributed with equal space around them.

## 5. align-items

Definition:
Specifies how flex items are aligned along the cross-axis of the flex container. It controls the alignment of items within the flex container along the cross-axis.
Example:

```
.container {
  display: flex;
  align-items: center;
}
```

### Use Case:

Use align-items to align flex items along the cross axis, ensuring consistent alignment and spacing of items within the flex container.

### Notes:

This property aligns flex items along the cross-axis.

### Values:

- flex-start: Items are aligned at the start of the cross-axis.
- flex-end: Items are aligned at the end of the cross-axis.
- center: Items are aligned at the center of the cross-axis.
- baseline: Items are aligned at their baselines.
- stretch: Items are stretched to fill the container along the cross-axis.

## 6. align-content

### Definition:

Specifies the alignment of flex lines within the flex container when there is extra space on the cross-axis. It determines how space is distributed between and around flex lines (items wrapped onto multiple lines) along the cross-axis of the flex container.
Example:

```
.container {
  display: flex;
  align-content: space-around;
}
```

### Use Case:

Use align-content to control the spacing and alignment of flex lines within the flex container, especially when there are multiple lines of flex items.

### Notes:

This property is relevant when flex-wrap is set to wrap or wrap-reverse.

### Values:

- flex-start: Lines are packed at the start of the container.
- flex-end: Lines are packed at the end of the container.
- center: Lines are packed at the center of the container.
- space-between: Lines are evenly distributed with the first line at the start and the last line at the end.
- space-around: Lines are evenly distributed with equal space around them.
- stretch: Lines are stretched to fill the container.

## 7. flex-grow

### Definition:

Specifies how much a flex item will grow relative to the rest of the flex items inside the same container. It defines the ability for a flex item to grow if necessary to fill available space along the main axis.
Example:

```
.item {
  flex-grow: 1; /* or any positive number */
}
```

### Use Case:

Use flex-grow to control the relative growth of flex items within a flex container, distributing available space along the main axis based on their flex-grow values.

### Notes:

Flex items with higher flex-grow values will grow proportionally more than items with lower values.

### Values:

- number: A positive number specifies the relative proportion of the available space the flex item should take up. For example, a value of 2 means the item will take up twice as much space as other items with a value of 1.

## 8. flex-shrink

### Definition:

Determines the ability for a flex item to shrink if necessary. It specifies how flex items will shrink relative to each other along the main axis when there isn't enough space in the flex container.
Example:

```
.item {
  flex-shrink: 1; /* or any positive number */
}
```

### Use Case:

Use flex-shrink to control the ability of flex items to shrink proportionally when the available space along the main axis is insufficient to accommodate all items at their preferred sizes.

### Notes:

Flex items with higher flex-shrink values will shrink more than items with lower values.

### Values:

- number: A positive number specifies the factor by which the flex item can shrink relative to other flex items. For example, a value of 2 means the item can shrink twice as much as other items with a value of 1.

## 9. flex-basis

### Definition:

Specifies the initial main size of a flex item before any available space is distributed. It defines the default size of an element before the remaining space is distributed.
Example:

```
.item {
  flex-basis: 100px;
}
```

### Use Case:

Use a flex-basis to set the initial size of flex items along the main axis, providing a basis for calculating their final sizes within the flex container.

### Notes:

This property sets the initial size of flex items before any available space is distributed among them.

### Values:

- length: Specifies the initial size of the flex item before any available space is distributed. It can be a length value like px, %, em, etc., or auto.

## 10. flex

### Definition:

A shorthand property for setting the flex-grow, flex-shrink, and flex-basis properties in a single declaration. It combines the flex-grow, flex-shrink, and flex-basis properties into a single convenient shorthand property.

Example:

```
.item {
  flex: 1 1 auto; /* flex-grow, flex-shrink, flex-basis */
}
```

### Use Case:

Use flex when you want to set all three flex properties (flex-grow, flex-shrink, and flex-basis) in a single declaration, simplifying the syntax and making the code more concise.

### Notes:

The first value represents flex-grow.
The second value represents flex-shrink.
The third value represents flex-basis.

### Values:

- flex-grow: Specifies how much the item will grow relative to the rest of the flex items.
- flex-shrink: Specifies how much the item will shrink relative to the rest of the flex items.
- flex-basis: Specifies the initial main size of the item before any available space is distributed.

## 11. flex-flow

### Definition:

This property is a shorthand for setting both the flex-direction and flex-wrap properties at the same time. It allows you to specify the direction of the main axis and whether flex items should wrap or not in a single declaration.

Example:

```
.container {
  flex-flow: row wrap;
}
```

### Use Case:

Use flex-flow when you need to set both the direction of flex items and whether they should wrap or remain in a single line, providing a concise way to manage the flow of flex items within a flex container.

### Notes:

This property combines the functionality of flex-direction and flex-wrap into a single declaration.

### Values:

Same as flex-direction and flex-wrap values.

## 12. order

### Definition:

Specifies the order in which flex items are displayed within a flex container. By default, flex items are displayed in the order they appear in the source code. However, the order property allows you to control the visual order independently of the source order.
Example:

```
.item {
  order: 2; /* Display item 2 before item 1 visually */
}
```

### Use Case:

Use order when you need to rearrange the visual order of flex items without changing their position in the source code, providing flexibility in controlling the display order of items within a flex container.

### Notes:

Lower order values position items earlier in the display order.
Negative values are also allowed.

### Values:

number: Specifies the order in which the item should appear within the flex container. Lower values appear earlier in the display order.

## 13. align-self

### Definition:

This property allows individual flex items to override the alignment set by the align-items property for their respective cross-axis.
Example:

```
.item {
  align-self: flex-start;
}
```

### Use Case:

Use align-self when you need to align a specific flex item differently from the other items in the flex container, providing fine-grained control over individual item alignment.

### Notes:

align-self works similarly to align-items, but it applies only to the individual flex item, overriding the alignment set by the container's align-items property.

Values:

Same as align-items values, but applied only to the individual flex item, overriding the alignment set by the container's align-items property.

## Conclusion

In conclusion, mastering CSS Flexbox properties opens up a world of possibilities for creating dynamic and responsive layouts in web design. By understanding and effectively utilising properties like display, flex-direction, flex-wrap, justify-content, align-items, and align-content, developers gain granular control over the arrangement and alignment of elements within a flex container. Whether it's arranging items in rows or columns, controlling wrapping behaviour, or evenly distributing space between or around items, Flexbox offers intuitive solutions for diverse layout requirements.

Furthermore, flex-grow, flex-shrink, and flex-basis empower developers to create flexible and adaptable layouts that respond seamlessly to changes in viewport size or content. With flex, developers can set these properties succinctly, streamlining the code and enhancing maintainability. Additionally, the order property provides the flexibility to manipulate the visual order of elements independently of their source order, facilitating complex layouts while maintaining semantic integrity.

Moreover, the flex-flow property simplifies the management of both flex-direction and flex-wrap, offering a convenient shorthand for controlling the flow of flex items within a container. Lastly, the align-self property allows for fine-grained control over the alignment of individual items within the flex container, enabling developers to override the default alignment set by the container's align-items property. In essence, CSS Flexbox provides a powerful toolkit for building modern, responsive web layouts, empowering developers to easily create visually stunning and user-friendly interfaces.

## References


1. https://developer.mozilla.org/en-US/docs/Web/CSS/flex
2. https://developer.mozilla.org/en-US/docs/Web/CSS/order
3. https://developer.mozilla.org/en-US/docs/Web/CSS/align-content
4. https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content