

CSS Flexbox & Grid

1. CSS Flexbox (Flexible Box Layout)

Flexbox is a layout model in CSS designed to make it easier to align and distribute space among items in a container, even when their size is unknown or dynamic. It works along a single axis (either a row or a column) and is ideal for one-dimensional layouts.

Basic Concepts of Flexbox:

- **Flex Container:** The parent element that holds the flex items. It is set by applying `display: flex;` or `display: inline-flex;` to an element.
- **Flex Items:** The child elements within the flex container.

Syntax to make an element a flex container:

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

Key Flexbox Properties:

1. Container Properties:

- **flex-direction:** Defines the direction of the flex container's items (row or column).
 - `row`: Default, items are placed in a horizontal line (left to right).
 - `column`: Items are placed vertically (top to bottom).
 - `row-reverse`: Items are placed in a reverse horizontal direction.
 - `column-reverse`: Items are placed in reverse vertical direction.

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

- **justify-content:** Aligns items along the main axis (horizontal in a row, vertical in a column).
 - `flex-start`: Aligns items at the start of the container.
 - `center`: Aligns items at the center of the container.
 - `flex-end`: Aligns items at the end of the container.
 - `space-between`: Distributes items with equal spacing between them.
 - `space-around`: Distributes items with equal spacing around them.

```
.container {  
  justify-content: center; /* Centers items along the main axis */  
}
```

- **align-items:** Aligns items along the cross axis (vertical in a row, horizontal in a column).
 - **stretch:** Items stretch to fill the container (default).
 - **flex-start:** Aligns items to the top of the container (for rows).
 - **flex-end:** Aligns items to the bottom (for rows).
 - **center:** Aligns items to the center.
 - **baseline:** Aligns items based on their baseline (useful for text alignment).

```
.container {
  align-items: center; /* Vertically centers items in a row
  layout */
}
```

- **flex-wrap:** Controls whether items should wrap onto the next line when there's not enough space.
 - **nowrap:** Default, items will not wrap.
 - **wrap:** Items will wrap onto multiple lines.
 - **wrap-reverse:** Items will wrap in reverse order.

```
.container {
  flex-wrap: wrap; /* Allows items to wrap onto the next line */
}
```

2. Item Properties:

- **flex:** Defines how an item will grow and shrink relative to others within the container.
 - **Syntax:** `flex: <flex-grow> <flex-shrink> <flex-basis>;`
 - **flex-grow:** Defines how much an item will grow relative to others (default is 0).
 - **flex-shrink:** Defines how much an item will shrink relative to others (default is 1).
 - **flex-basis:** Defines the initial size of the item before it starts growing or shrinking.

```
.item {
  flex: 1; /* All items will grow equally to fill the container
  */
}
```

- **align-self:** Allows individual flex items to override the `align-items` property and align themselves differently along the cross axis.
 - **auto:** Default, follows the container's `align-items`.
 - **flex-start, flex-end, center, baseline, stretch:** Same options as `align-items`.

```
.item {
  align-self: center; /* Aligns this particular item to the
  center */
}
```

Flexbox 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;
  justify-content: space-between;
  align-items: center;
}

.item {
  width: 100px;
  height: 100px;
  background-color: lightblue;
  margin: 10px;
}
```

In this example, the items are evenly spaced across the container and centered vertically.

2. CSS Grid Layout

The **CSS Grid Layout** is a powerful two-dimensional layout system that enables precise control over both rows and columns. It's great for building complex, responsive web layouts.

- **Grid Container:** The parent element that holds the grid items. It is set by applying `display: grid;` to an element.
- **Grid Items:** The child elements that make up the grid.

Syntax to make an element a grid container:

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

Key Grid Properties:

1. Container Properties:

- **grid-template-columns:** Defines the columns of the grid.
 - You can set specific widths for each column or use `fr` (fractional unit) to distribute space equally.

```
.container {
  grid-template-columns: 1fr 1fr 1fr; /* Three equal-width
columns */
}
```

- **grid-template-rows:** Defines the rows of the grid.
 - Similar to columns, you can set fixed heights or use `fr` units.

```
.container {
  grid-template-rows: 200px 100px; /* Two rows: first 200px
  high, second 100px high */
}
```

- **grid-gap (gap):** Defines the space between grid items.

```
.container {
  grid-gap: 10px; /* 10px gap between grid items */
}
```

- **grid-template-areas:** Defines a grid template using named areas, making it easier to visualize the layout.

```
.container {
  grid-template-areas:
    "header header header"
    "main sidebar sidebar"
    "footer footer footer";
}
```

2. Item Properties:

- **grid-column:** Specifies where an item should start and end in terms of columns.

```
.item {
  grid-column: 1 / 3; /* Item spans from column 1 to column 3 */
}
```

- **grid-row:** Specifies where an item should start and end in terms of rows.

```
.item {
  grid-row: 2 / 4; /* Item spans from row 2 to row 4 */
}
```

Grid Example:

html

```
<div class="container">
  <div class="item header">Header</div>
  <div class="item main">Main Content</div>
  <div class="item sidebar">Sidebar</div>
  <div class="item footer">Footer</div>
</div>
```

CSS

```
.container {
  display: grid;
  grid-template-columns: 2fr 1fr; /* Two columns: main content is twice as
  wide */
  grid-template-rows: auto 1fr auto; /* Three rows: header, content, footer
  */
}
```

```
    grid-gap: 10px;
}

.header {
    grid-column: 1 / 3;
}

.main {
    grid-column: 1;
}

.sidebar {
    grid-column: 2;
}

.footer {
    grid-column: 1 / 3;
}
```

In this example, the grid has two columns (with the first column twice as wide), and three rows. The `header` spans both columns, and the `footer` also spans both columns.

Summary of Key Concepts:

Flexbox:

- **One-dimensional layout:** Works in a row or column direction.
- Use properties like `flex-direction`, `justify-content`, `align-items`, and `flex-wrap` for positioning and spacing items.
- Flexible items grow and shrink to fit the container.

CSS Grid:

- **Two-dimensional layout:** Works with both rows and columns.
- Use properties like `grid-template-columns`, `grid-template-rows`, `grid-gap`, and `grid-template-areas` to control the layout.
- Items can span multiple rows and columns using `grid-column` and `grid-row`.

Both **Flexbox** and **Grid** are powerful layout tools, with Flexbox being best for simple, linear layouts and Grid being more suitable for complex, two-dimensional layouts.