CSS Flexbox: An Introduction with Examples

Flexbox (Flexible Box Layout) is a powerful layout model in CSS that allows you to align and distribute space among items in a container, even when their size is unknown or dynamic. It provides an efficient and simple way to create flexible, responsive layouts.

Basic Flexbox Syntax

```
css
Copy code
.container {
  display: flex;
  /* Additional flex properties can be applied here */
}

.item {
  /* Flex item properties */
}
```

1. Flex Container and Flex Items

To enable Flexbox, you set the container's `display` property to `flex`. This turns the container into a flex container, and its children automatically become **flex items**.

Example 1: Simple Flexbox Layout

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox Layout Example</title>
  <style>
    .container {
      display: flex;
      border: 1px solid #ccc;
    }

    .item {
      padding: 20px;
      border: 1px solid #000;
      margin: 5px;
      background-color: lightblue;
    }
  </style>
</head>
<body>

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

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

</body>
</html>
```

Explanation:

- **`display: flex;`** turns the `.container` into a flex container.
- The `.item` elements inside the container are automatically laid out in a row (the default flex direction).

2. `flex-direction`

The `flex-direction` property defines the direction in which the flex items are placed inside the flex container. The possible values are:

- `row`: Horizontal, left to right (default).
- `row-reverse`: Horizontal, right to left.
- `column`: Vertical, top to bottom.
- `column-reverse`: Vertical, bottom to top.

Example 2: Using `flex-direction`

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flex Direction Example</title>
  <style>
    .container {
      display: flex;
      flex-direction: column; /* Change direction to vertical */
      border: 1px solid #ccc;
      height: 200px;
    }

    .item {
      padding: 20px;
      border: 1px solid #000;
      margin: 5px;
      background-color: lightgreen;
    }
  </style>
</head>
<body>

  <div class="container">
```

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

</body>
</html>
```

Explanation:

- **`flex-direction: column;`** stacks the items vertically (top to bottom).

3. `justify-content`

The `justify-content` property aligns the flex items along the **main axis** (horizontally, by default). It has several values:

- `flex-start`: Aligns items to the start of the container (default).
- `flex-end`: Aligns items to the end of the container.
- `center`: Centers the items horizontally.
- `space-between`: Distributes items evenly, with space between them.
- `space-around`: Distributes items with equal space around them.
- `space-evenly`: Distributes items with equal space between and around them.

Example 3: Using `justify-content`

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Justify Content Example</title>
  <style>
    .container {
      display: flex;
      justify-content: space-between; /* Distribute space between items */
      border: 1px solid #ccc;
      height: 200px;
    }

    .item {
      padding: 20px;
      border: 1px solid #000;
      margin: 5px;
      background-color: lightcoral;
    }
  </style>
</head>
<body>
```

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

</body>
</html>
```

Explanation:

- **`justify-content: space-between;`** distributes the items evenly with equal space between them along the main axis.

4. `align-items`

The `align-items` property aligns the flex items along the **cross axis** (vertical, by default). The possible values are:

- `flex-start`: Aligns items to the top (or left in `row` direction).
- `flex-end`: Aligns items to the bottom (or right in `row` direction).
- `center`: Aligns items vertically in the center.
- `baseline`: Aligns items along their baseline.
- `stretch`: Stretches items to fill the container (default behavior).

Example 4: Using `align-items`

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Align Items Example</title>
  <style>
    .container {
      display: flex;
      align-items: center; /* Center items vertically */
      border: 1px solid #ccc;
      height: 200px;
    }

    .item {
      padding: 20px;
      border: 1px solid #000;
      margin: 5px;
      background-color: lightyellow;
    }
  </style>
</head>
<body>
```

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

</body>
</html>
```

Explanation:

- **`align-items: center;`** vertically centers the items in the flex container.

5. `flex-wrap`

The `flex-wrap` property allows flex items to wrap onto multiple lines if they overflow the container. It has three values:

- `nowrap` (default): Items do not wrap.
- `wrap`: Items wrap onto the next line.
- `wrap-reverse`: Items wrap in reverse order.

Example 5: Using `flex-wrap`

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flex Wrap Example</title>
  <style>
    .container {
      display: flex;
      flex-wrap: wrap; /* Allow items to wrap */
      border: 1px solid #ccc;
      width: 300px;
    }

    .item {
      flex: 0 0 100px; /* Each item takes up 100px */
      padding: 20px;
      border: 1px solid #000;
      margin: 5px;
      background-color: lightpink;
    }
  </style>
</head>
<body>

  <div class="container">
```

```
    <div class="item">Item 1</div>
    <div class="item">Item 2</div>
    <div class="item">Item 3</div>
    <div class="item">Item 4</div>
    <div class="item">Item 5</div>
  </div>

</body>
</html>
```

Explanation:

- **`flex-wrap: wrap;`** allows items to wrap to the next line when they exceed the container width.

6. `flex-grow`, `flex-shrink`, and `flex-basis`

These three properties control the behavior of flex items within a flex container:

- **`flex-grow`**: Defines how much a flex item should grow relative to other items.
- **`flex-shrink`**: Defines how much a flex item should shrink when there is not enough space.
- **`flex-basis`**: Defines the initial size of the flex item before it grows or shrinks.

Example 6: Using `flex-grow` and `flex-shrink`

```
html
Copy code
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flex Grow and Shrink Example</title>
  <style>
    .container {
      display: flex;
      border: 1px solid #ccc;
      width: 400px;
    }

    .item {
      padding: 20px;
      border: 1px solid #000;
      margin: 5px;
      background-color: lightgreen;
    }

    .grow {
      flex-grow: 2; /* This item will grow more */
    }
```

```
    .shrink {
      flex-shrink: 1; /* This item can shrink */
    }

    .flex-basis {
      flex-basis: 100px; /* Set initial size */
    }
  </style>
</head>
<body>

  <div class="container">
    <div class="item grow">Item 1 (grow)</div>
    <div class="item shrink">Item 2 (shrink)</div>
    <div class="item flex-basis">Item 3 (flex-basis)</div>
  </div>

</body>
</html>
```

Explanation:

- **`flex-grow: 2;`** makes the first item grow twice as much as the other items.
- **`flex-shrink: 1;`** allows the second item to shrink if there's not enough space.
- **`flex-basis: 100px;`** sets the initial size of the third item to 100px.

Conclusion:

Flexbox is a powerful tool for creating responsive, flexible layouts. It provides easy ways to control the arrangement of items within a container, whether you want them to be aligned in rows or columns, centered, spaced evenly, or wrapped. The flexibility of Flexbox makes it an essential part of modern CSS layouts.