CSS_3

Agenda

- css units
- flexbox
  - basics
  - advanced

CSS Units: Absolute, Relative

In CSS, units are used to define the size, length, and spacing of elements. These units can be categorized into absolute, relative, and computed units.

Absolute vs. Relative Units

Absolute Units:

- Fixed size regardless of context.
- Examples: `px`, `cm`, `in`, `mm`, `pt`, `pc`.

Relative Units:

- Size relative to another element.
- Examples: `em`, `rem`, `%`, `vh`, `vw`.

Common CSS Units

Pixels (`px`)

- Absolute unit.
- Represents a fixed number of pixels.
- Example:

```
.px {
    font-size: 20px;
```

```
}
```

Root Em (`rem`)

- Relative to the font size of the root element (`<html>`).
- Example:

```
html {
    font-size: 10px; /* Usually set to 16px by default */
}
.rem {
    font-size: 2rem; /* 2 * 10px = 20px */
}
```

Em (`em`)

- Relative to the font size of the parent element.
- Example:

```
.pa {
    font-size: 20px;
}
.em {
    height: 10em; /* 10 * 20px = 200px */
    width: 10em;  /* 10 * 20px = 200px */
    background-color: lightpink;
}
```

Viewport Height (`vh`) and Viewport Width (`vw`)

- `vh`: 1% of the viewport height., 100vh= current height of browser window
- `vw`: 1% of the viewport width., 100vw =current width of browser window
- Example:

```css
.vh-vw {
    height: 20vh; /* 20% of the viewport height */
    width: 50vw;  /* 50% of the viewport width */
    font-size: 2rem;
}
```

Percentages (%)

- Relative to the size of the parent element.
- Example:

```css
.gp {
    height: 200px;
    width: 200px;
    background-color: bisque;
}
.parent {
    height: 50%; /* 50% of its parent's height */
    width: 50%;  /* 50% of its parent's width */
    background-color: lightcoral;
}
.child {
    background-color: lightblue;
    height: 50%; /* 50% of .parent's height */
    width: 50%;  /* 50% of .parent's width */
}
```

Code Examples

HTML Structure

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>CSS Units Demo</title>
    <style>
        .px,
        .rem,
        .vh-vw,
        .em {
            margin-bottom: 20px;
            background-color: lightgreen;
        }

        .px {
            font-size: 20px; /* Absolute unit */
        }

        html {
            font-size: 10px; /* 1rem = 10px */
        }

        .pa {
            font-size: 20px; /* Parent font size */
        }

        .em {
            height: 10em; /* 10 * 20px = 200px */
            width: 10em;  /* 10 * 20px = 200px */
            background-color: lightpink;
        }

        .rem {
            font-size: 2rem; /* 2 * 10px = 20px */
        }

        .vh-vw {
            height: 20vh; /* 20% of the viewport height */
            width: 50vw;  /* 50% of the viewport width */
            font-size: 2rem;
        }

        .gp {
            height: 200px;
            width: 200px;
```

```
            background-color: bisque;
        }

        .parent {
            height: 50%;
            width: 50%;
            background-color: lightcoral;
        }

        .child {
            background-color: lightblue;
            height: 50%; /* 50% of .parent's height */
            width: 50%;  /* 50% of .parent's width */
        }
    </style>
</head>
<body>
    <div class="px">This is a fixed size text (20px)</div>
    <div class="rem">This text is sized using rem (2rem)</div>
    <div class="em">This is a box sized using em</div>
    <div class="vh-vw">This box is sized using vh and vw</div>
    <div class="gp">
        <div class="parent">
            <div class="child">This box is sized using %</div>
        </div>
    </div>
</body>
</html>
```

Explanation

1. **Pixels (`px`)**: An absolute unit where `20px` will always be 20 pixels regardless of the context.
2. **Root Em (`rem`)**: Relative to the root element's font size (`<html>`). If `<html>` has `font-size: 10px`, `2rem` equals `20px`.
3. **Em (`em`)**: Relative to the font size of its parent element. If the parent has `font-size: 20px`, `1em` equals `20px`.

4. **Viewport Height (`vh`) and Width (`vw`)**: Relative to the size of the viewport. `20vh` is 20% of the viewport height, and `50vw` is 50% of the viewport width.
5. **Percentages (`%`)**: Relative to the size of the parent element. A child element with `height: 50%` of a parent with `height: 200px` will have a height of `100px`.

Flexbox

Flexbox is a powerful layout module in CSS that allows you to create flexible and responsive layouts. It is particularly useful for arranging elements in a one-dimensional space either as rows or columns.

1. `display: flex`

To use Flexbox, you need to set the container's display property to `flex`:

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

This makes the container a flex container and its children flex items.

2. `justify-content`

The `justify-content` property is used to align the flex items along the main axis (horizontal by default):

- `flex-start`: Aligns items to the start of the container.
- `center`: Aligns items to the center of the container.
- `flex-end`: Aligns items to the end of the container.
- `space-between`: Distributes items evenly with the first item at the start and the last item at the end.
- `space-around`: Distributes items evenly with equal space around each item.
- `space-evenly`: Distributes items evenly with equal space between and around them.

Examples:

```
.parent {
    display: flex;
    justify-content: center; /* center the items horizontally */
}
```

3. `align-items`

The `align-items` property aligns the flex items along the cross axis (vertical by default):

- `flex-start`: Aligns items to the start of the container.
- `center`: Aligns items to the center of the container.
- `flex-end`: Aligns items to the end of the container.
- `stretch`: Stretches items to fill the container.
- `baseline`: Aligns items such that their baselines align.

Examples:

```
.parent {
    display: flex;
    align-items: center; /* center the items vertically */
}
```

4. `align-self`

The `align-self` property allows you to override the `align-items` property for individual flex items:

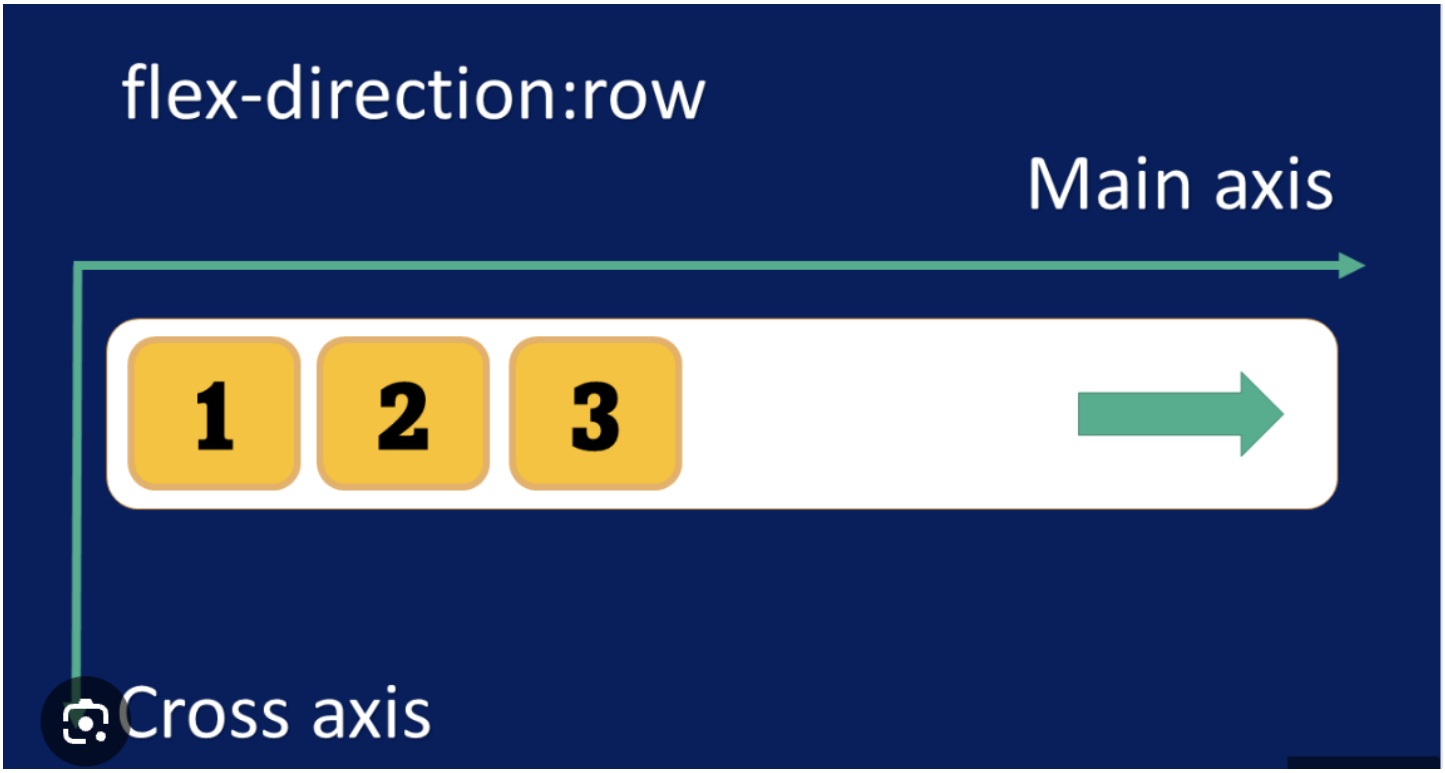
- `auto`: Inherits the value of `align-items`.
- `flex-start`: Aligns the item to the start of the container.
- `center`: Aligns the item to the center of the container.
- `flex-end`: Aligns the item to the end of the container.
- `stretch`: Stretches the item to fill the container.

Examples:

```
.one {
    align-self: flex-start; /* align this item to the start */
}

.two {
    align-self: flex-end; /* align this item to the end */
}
```

`flex-direction` and `flex-wrap`



1. Main Axis and Cross Axis

In Flexbox, the main axis and cross axis are key concepts:

- **Main Axis**: The primary axis along which flex items are laid out. By default, this is horizontal (left to right).
- **Cross Axis**: The perpendicular axis to the main axis. By default, this is vertical (top to bottom).

The direction of these axes can change based on the `flex-direction` property.

2. `flex-direction` Property

The `flex-direction` property defines the direction in which flex items are placed in the flex container. It can take the following values:

- `row` (default): Main axis is horizontal, items are placed from left to right.
- `row-reverse`: Main axis is horizontal, items are placed from right to left.
- `column`: Main axis is vertical, items are placed from top to bottom.
- `column-reverse`: Main axis is vertical, items are placed from bottom to top.

Examples:

```
.container {
    display: flex;
    flex-direction: row; /* Default: horizontal left to right */
}

.container-reverse {
    display: flex;
    flex-direction: row-reverse; /* Horizontal right to left */
}

.container-column {
    display: flex;
    flex-direction: column; /* Vertical top to bottom */
}

.container-column-reverse {
    display: flex;
    flex-direction: column-reverse; /* Vertical bottom to top */
}
```

3. `flex-wrap` Property

The `flex-wrap` property controls whether flex items should wrap onto multiple lines. It can take the following values:

- `nowrap` (default): All flex items are placed in a single line.
- `wrap`: Flex items will wrap onto multiple lines, from top to bottom.
- `wrap-reverse`: Flex items will wrap onto multiple lines, from bottom to top.

Examples:

```
.container {
    display: flex;
    flex-wrap: nowrap; /* Default: single line */
}

.container-wrap {
    display: flex;
    flex-wrap: wrap; /* Multiple lines from top to bottom */
}

.container-wrap-reverse {
    display: flex;
    flex-wrap: wrap-reverse; /* Multiple lines from bottom to top */
}
```

Example Code

Here's an example to demonstrate the usage of `flex-direction` and `flex-wrap`:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flexbox Example</title>
    <style>
        .container {
            display: flex;
            flex-direction: row; /* Try changing this to row-reverse,
column, or column-reverse */
            flex-wrap: wrap; /* Try changing this to nowrap or wrap-
reverse */
```

```
            background-color: rebeccapurple;
            height: 300px;
            width: 100%;
        }

        .item {
            background-color: lightcoral;
            color: white;
            font-weight: bold;
            font-family: sans-serif;
            height: 100px;
            width: 100px;
            margin: 10px;
            text-align: center;
            line-height: 100px;
        }
    </style>
</head>

<body>
    <h2>Flexbox Example</h2>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
        <div class="item">3</div>
        <div class="item">4</div>
        <div class="item">5</div>
        <div class="item">6</div>
    </div>
</body>

</html>
```

Explanation:

1. **Flex Direction**:
    - The `.container` class has `flex-direction: row;` by default, placing items from left to right along the main axis.

- Changing `flex-direction` to `row-reverse`, `column`, or `column-reverse` will change the direction of item placement.

2. **Flex Wrap**:
   - The `.container` class has `flex-wrap: wrap;` allowing items to wrap onto multiple lines from top to bottom.
   - Changing `flex-wrap` to `nowrap` will keep all items in a single line, and `wrap-reverse` will wrap items from bottom to top.

Using these properties, you can control the layout and behavior of flex items within a flex container, creating responsive and flexible designs.