# CSS Selectors & Units: Class, ID, px, em, %, vh, vw

## *1. Class Selector (.)*

Definition:

- A class selector in CSS is used to select one or more HTML elements based on the class attribute.

Syntax:

```
.className {
  property: value;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .highlight {
            color: white;
            background-color: blue;
            padding: 10px;
        }
    </style>
</head>
<body>
    <p class="highlight">This is a highlighted paragraph.</p>
</body>
</html>
```

Real-time Use Case:

Used to style multiple elements with the same look, like buttons, cards, or sections in a webpage.

## *2. ID Selector (#)*

Definition:

- An ID selector in CSS selects a single unique element based on its id attribute.

Syntax:

```
#idName {
  property: value;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        #main-header {
            font-size: 24px;
            color: red;
            text-align: center;
        }
    </style>
</head>
<body>
    <h1 id="main-header">Welcome to My Website</h1>
</body>
</html>
```

Real-time Use Case:

Used for styling unique elements like navigation bars, specific sections, or page headers.

***3. px (Pixels)***

Definition:

- px (pixels) is an absolute unit representing a fixed size on the screen.

Syntax:

```
element {
  property: 16px;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        p {
            font-size: 20px;
        }
    </style>
</head>
<body>
    <p>This text has a font size of 20 pixels.</p>
</body>
</html>
```

Real-time Use Case:

Used for setting exact sizes for fonts, margins, padding, or borders when consistent spacing is needed.

***4. em (Relative to Parent Font Size)***

Definition:

- em is a relative unit where 1em equals the font size of the parent element.

Syntax:

```
element {
  property: 1.5em;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            font-size: 20px;
        }
        p {
            font-size: 1.5em; /* 1.5 times the font size of div (20px * 1.5 = 30px) */
        }
    </style>
</head>
<body>
    <div>
        <p>This text size is relative to its parent.</p>
    </div>
</body>
</html>
```

Real-time Use Case:

Used for responsive typography where text scales based on its parent container.

***5. % (Percentage)***

Definition:

- % is a relative unit based on the parent element's size.

Syntax:

```
element {
  width: 50%;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            width: 50%;
            background-color: lightblue;
            padding: 20px;
        }
    </style>
</head>
<body>
    <div>This div takes 50% of the screen width.</div>
</body>
</html>
```

Real-time Use Case:

Used for fluid layouts where elements adjust dynamically based on screen size.

***6. vh (Viewport Height)***

Definition:

- vh is a relative unit where 1vh equals 1% of the viewport height.

Syntax:

```
element {
  height: 50vh;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            height: 50vh;
            background-color: lightcoral;
        }
    </style>
</head>
<body>
    <div>This div is 50% of the viewport height.</div>
</body>
</html>
```

Real-time Use Case:

Used for creating full-screen sections or hero banners.

***7. vw (Viewport Width)***

Definition:

- vw is a relative unit where 1vw equals 1% of the viewport width.

Syntax:

```
element {
  width: 50vw;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            width: 50vw;
            background-color: lightgreen;
        }
    </style>
</head>
<body>
    <div>This div is 50% of the viewport width.</div>
</body>
</html>
```

Real-time Use Case:

Used for creating responsive elements that adjust based on screen width

**Summary Table**

| Selector/Unit | Definition | Use Case |
|---|---|---|
| `.class` | Selects multiple elements with the same class | Styling buttons, text, or sections |
| `#id` | Selects a unique element | Styling specific headers, navigation bars |
| `px` | Absolute unit (fixed size) | Exact spacing, font sizes |
| `em` | Relative to parent font size | Responsive typography |
| `%` | Relative to parent size | Fluid layouts |
| `vh` | Relative to viewport height | Full-screen sections |
| `vw` | Relative to viewport width | Responsive width adjustments |

## CSS Box Model: Margin, Padding, Border, Width, Height

What is the CSS Box Model?

The CSS Box Model is a fundamental concept in web design that describes how elements are structured and spaced on a webpage. Every HTML element is treated as a rectangular box consisting of the following layers:

- Content – The actual content of the element (text, image, etc.).
- Padding – The space between the content and the border.
- Border – The outline surrounding the padding and content.
- Margin – The space outside the border that separates the element from others.

Why is the Box Model Important?

- It helps in layout design by allowing precise control over spacing and sizing.
- It ensures consistent design across different screen sizes and resolutions.
- It is crucial for responsive design and avoiding layout shifts.

### *1. Margin*

Definition:

- The margin property creates space outside the element's border, separating it from other elements.

Syntax:

```
element {
  margin: value; /* Can be px, %, em, auto */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            width: 200px;
            height: 100px;
            background-color: lightblue;
            margin: 20px; /* Adds space around the element */
        }
    </style>
</head>
<body>
    <div>This box has a margin of 20px.</div>
</body>
</html>
```

Real-time Use Case:

- Used to create spacing between buttons, sections, or images in a webpage.
- Example: Adding space between form fields in a contact form.

### *2. Padding*

Definition:

- The padding property creates space inside an element, between the content and its border.

Syntax:

```
element {
  padding: value; /* Can be px, %, em */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            width: 200px;
            height: 100px;
            background-color: lightcoral;
            padding: 20px; /* Adds space inside the element */
        }
    </style>
</head>
<body>
    <div>This box has 20px padding.</div>
</body>
</html>
```

Real-time Use Case:

- Used to increase readability by adding space inside elements like buttons and cards.
- Example: Increasing padding in a navigation bar to make links easier to click.

***3. Border***

Definition:

- The border property defines the outline around an element.

Syntax:

```
element {
  border: width style color;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            width: 200px;
            height: 100px;
            background-color: lightgreen;
            border: 5px solid black; /* Thick solid black border */
        }
    </style>
</head>
<body>
    <div>This box has a 5px solid border.</div>
</body>
</html>
```

Real-time Use Case:

- Used for highlighting sections like input fields in a form.
- Example: Adding red borders to input fields for error messages.

### *4. Width*

Definition:

- The width property defines how wide an element should be.

Syntax:

```
element {
  width: value; /* Can be px, %, vw, em */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            width: 50%; /* 50% of the parent container */
            background-color: orange;
        }
    </style>
</head>
<body>
    <div>This box takes 50% of the width.</div>
</body>
</html>
```

Real-time Use Case:

- Used for fluid layouts where elements adapt to different screen sizes.
- Example: Making images responsive by setting width: 100%.

### *5. Height*

Definition:

- The height property defines how tall an element should be.

Syntax:

```
element {
  height: value; /* Can be px, %, vh, em */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            height: 100px; /* Fixed height */
            background-color: yellow;
        }
    </style>
</head>
<body>
    <div>This box has a height of 100px.</div>
</body>
</html>
```

Real-time Use Case:

- Used to maintain consistent design across different elements.
- Example: Fixing the height of a navbar for uniformity.

## Complete Example: Box Model in Action

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .box {
            width: 300px;
            height: 150px;
            background-color: lightblue;
            padding: 20px;
            border: 5px solid black;
            margin: 30px;
        }
    </style>
</head>
<body>
    <div class="box">This box follows the CSS Box Model.</div>
</body>
</html>
```

How This Works:

- Width: 300px
- Height: 150px
- Padding: 20px (adds space inside)
- Border: 5px solid black
- Margin: 30px (adds space outside)

Summary Table

| Property | Definition | Use Case |
|---|---|---|
| `margin` | Space outside an element | Creating gaps between sections |
| `padding` | Space inside an element | Adding space inside buttons for better readability |
| `border` | Outline around an element | Highlighting form fields or creating card-like designs |
| `width` | Defines how wide an element is | Making layouts responsive |
| `height` | Defines how tall an element is | Setting fixed heights for headers or footers |

Why Understanding the Box Model is Essential?

- Avoids Layout Issues – Ensures proper spacing between elements.
- Helps in Responsive Design – Allows elements to scale dynamically.
- Improves User Experience – Enhances readability and interaction.
- Optimizes Page Layout – Prevents overlapping and misalignment issues.

## Typography in CSS

### *What is Typography?*

Typography in CSS refers to the styling and arrangement of text on a webpage. It involves selecting fonts, adjusting sizes, spacing, alignment, and decoration to enhance readability and aesthetics.

***Why is Typography Important?***

- Improves readability – Ensures text is easy to read.
- Enhances user experience – A well-structured text layout improves engagement.
- Boosts accessibility – Proper font choices help users with visual impairments.
- Creates a professional look – Typography sets the tone for a website's design.
-

***1. font-family***

Definition:

- The font-family property specifies the font type for an element. If the preferred font is unavailable, fallback fonts are used.

Syntax:

```
element {
  font-family: "Font Name", fallback-font, generic-family;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        p {
            font-family: "Arial", sans-serif;
        }
    </style>
</head>
<body>
    <p>This text uses the Arial font.</p>
</body>
</html>
```

Real-time Use Case:

- Used to set brand identity with a consistent font style.
- Example: News websites use serif fonts for a professional look, while tech blogs prefer sans-serif for modern appeal.

***2. font-size***

Definition:

- The font-size property defines the text size. It can be in px (pixels), em (relative to parent), %, vw (viewport width), rem (relative to root font size).

Syntax:

```
element {
  font-size: value; /* px, em, %, vw, rem */
}
```

Sample Code:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        p {
            font-size: 20px;
        }
    </style>
</head>
<body>
    <p>This text is 20px in size.</p>
</body>
</html>
```

Real-time Use Case:

- Used to control text hierarchy for better readability.
- Example: Headings (H1) are large, paragraphs are smaller, ensuring a structured layout.

***3. text-align***

Definition:

- The text-align property defines how text is aligned within its container.

Values:

- left – Aligns text to the left (default).
- right – Aligns text to the right.
- center – Centers text.
- justify – Aligns text to both left and right margins, adding spacing between words.

Syntax:

```css
element {
  text-align: left | right | center | justify;
}
```

Sample Code:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        p {
            text-align: center;
        }
    </style>
</head>
<body>
    <p>This text is centered.</p>
</body>
</html>
```

Real-time Use Case:

- Used to align headings, paragraphs, and buttons based on design requirements.
- Example: Justified text is used in newspapers to create a clean look.

### *4. text-decoration*

Definition:

- The text-decoration property controls text styling such as underlining, overlining, and strikethrough.

Values:

- none – No decoration.
- underline – Adds an underline.
- overline – Adds a line above the text.
- line-through – Strikes through the text.
- blink – (Deprecated) Makes text blink.

Syntax:

```
element {
  text-decoration: none | underline | overline | line-through;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        a {
            text-decoration: none;
        }
    </style>
</head>
<body>
    <a href="#">This is a link without an underline.</a>
</body>
</html>
```

Real-time Use Case:

- Used to style hyperlinks and headings for better UI.
- Example: Removing the underline from navigation links in a menu.

### *5. letter-spacing*

Definition:

- The letter-spacing property defines the spacing between characters in a text.

Syntax:

```
element {
  letter-spacing: value; /* px, em */
}
```

Sample Code:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        p {
            letter-spacing: 2px;
        }
    </style>
</head>
<body>
    <p>This text has 2px letter spacing.</p>
</body>
</html>
```

Real-time Use Case:

- Used to improve text readability or for aesthetic effects.
- Example: Increasing spacing in branding elements like logos.

## Complete Example: Applying Typography Properties

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .text-style {
            font-family: "Georgia", serif;
            font-size: 18px;
            text-align: justify;
            text-decoration: underline;
            letter-spacing: 1px;
        }
    </style>
</head>
<body>
    <p class="text-style">This paragraph applies typography properties.</p>
</body>
</html>
```

***How This Works:***

- Font-family: Uses a serif font (Georgia).
- Font-size: Sets text size to 18px.
- Text-align: Justifies text.
- Text-decoration: Underlines text.
- Letter-spacing: Increases space between characters.

***Summary Table:***

| Property | Definition | Use Case |
|---|---|---|
| `font-family` | Defines the font type | Ensures consistent branding |
| `font-size` | Defines text size | Establishes text hierarchy |
| `text-align` | Aligns text | Centers headlines, justifies paragraphs |
| `text-decoration` | Styles text (underline, strikethrough) | Enhances UI for links and titles |
| `letter-spacing` | Adjusts space between characters | Improves readability, branding |

## Typography in CSS: Font-Family, Font-Size, Text-Align, Text-Decoration, Letter-Spacing

### *What is Typography?*

- Typography refers to the style, arrangement, and appearance of text in web design. It plays a crucial role in readability, aesthetics, and user experience.

### *Why is Typography Important?*

- Improves Readability – Makes content easy to read and scan.
- Enhances Visual Appeal – Creates an attractive and professional design.
- Establishes Brand Identity – Different fonts and styles convey different moods.
- Affects User Engagement – Good typography ensures users stay longer on a webpage.

### *1. Font-Family*

Definition:

- The font-family property specifies the typeface (font) to be used for text. It can have multiple values (fallback fonts), where the browser picks the first available one.

Syntax:

```
element {
  font-family: "Primary Font", "Fallback Font", generic-family;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        p {
            font-family: "Arial", "Helvetica", sans-serif;
        }
    </style>
</head>
<body>
    <p>This text uses Arial font.</p>
</body>
</html>
```

Real-time Use Case:

- Using serif fonts (e.g., Times New Roman) for professional websites.
- Using sans-serif fonts (e.g., Arial, Roboto) for modern, clean designs.

### *2. Font-Size*

Definition:

The font-size property sets the size of the text. It can be specified in:

- px (pixels) – Fixed size

- em – Relative to parent element
- % – Relative to the default size
- rem – Relative to the root element

Syntax:

```
element {
  font-size: value; /* px, em, %, rem */
}
```

## Colors & Backgrounds in CSS

Colors and backgrounds in CSS help in designing visually appealing web pages by controlling the appearance of elements. These properties enhance readability, aesthetics, and user experience.

### *1. Color*

Definition:

- The color property in CSS sets the text color of an element.

Syntax:

```
element {
  color: value; /* Can be name, hex, rgb, hsl */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        p {
            color: blue; /* Text color set to blue */
        }
    </style>
</head>
<body>
    <p>This text is blue.</p>
</body>
</html>
```

Real-time Use Case:

- Used to style headings, paragraphs, links, and buttons for better visibility.
- Example: Setting the primary theme color for a website (e.g., brand colors).

### *2. Background-Color*

Definition:

- The background-color property sets the background color of an element.

Syntax:

```
element {
  background-color: value; /* Can be name, hex, rgb, hsl */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            background-color: lightgray;
            padding: 20px;
        }
    </style>
</head>
<body>
    <div>This box has a light gray background.</div>
</body>
</html>
```

Real-time Use Case:

- Used for sections, navigation bars, buttons, and highlights in a webpage.
- Example: Dark mode feature using background colors.

### *3. Background-Image*

Definition:

- The background-image property sets an image as the background of an element.

Syntax:

```
element {
  background-image: url("image.jpg");
  background-size: cover; /* Ensures image covers the full area */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            width: 300px;
            height: 200px;
            background-image: url("https://via.placeholder.com/300");
            background-size: cover;
            color: white;
            padding: 20px;
        }
    </style>
</head>
<body>
    <div>This box has a background image.</div>
</body>
</html>
```

Real-time Use Case:

- Used for hero sections, banners, and card backgrounds.
- Example: Using a cityscape image as a background for a travel website.

### *4. Opacity*

Definition:

- The opacity property sets the transparency level of an element. A value of 1 means fully visible, and 0 means completely transparent.

Syntax:

```
element {
  opacity: value; /* Range: 0 to 1 */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            width: 200px;
            height: 100px;
            background-color: blue;
            opacity: 0.5; /* 50% transparent */
            color: white;
            text-align: center;
            padding: 20px;
        }
    </style>
</head>
<body>
    <div>This box has 50% opacity.</div>
</body>
</html>
```

Real-time Use Case:

- Used to create hover effects, overlays, and glass morphism designs.
- Example: Making a login form background slightly transparent over an image.

**Summary Table**

| Property | Definition | Use Case |
|---|---|---|
| `color` | Sets the text color | Text styling, branding colors |
| `background-color` | Sets the background color | Section backgrounds, buttons |
| `background-image` | Adds an image as background | Hero sections, banners |
| `opacity` | Controls transparency of an element | Overlays, hover effects |

## Positioning in CSS: Static, Relative, Absolute, Fixed, Sticky

***What is positioning in CSS?***

Positioning in CSS is the method used to control the placement of elements on a webpage. It defines how an element is positioned relative to other elements or the viewport.

***Why is Positioning Important?***

- Precise Layout Control – Allows developers to place elements exactly where they need them.
- Better User Experience – Helps in designing intuitive and interactive layouts.
- Responsive Design – Makes elements adapt to different screen sizes.
- Layering Elements – Enables elements to overlap or stay fixed as per design needs.

### *1. Static Positioning*

Definition:

- Position: static; is the default position for all HTML elements. The element follows the normal document flow and cannot be moved using top, left, right, or bottom.

Syntax:

```
element {
  position: static;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        p {
            position: static;
            background-color: lightgray;
        }
    </style>
</head>
<body>
    <p>This paragraph has static positioning.</p>
</body>
</html>
```

Real-time Use Case:

- Used when no special positioning is required.
- Example: Regular paragraphs and headings that follow the natural document flow.

### *2. Relative Positioning*

Definition:

- Position: relative; moves the element relative to its original position. It does not affect surrounding elements.

Syntax:

```
element {
  position: relative;
  top: 10px; /* Moves down */
  left: 20px; /* Moves right */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        div {
            position: relative;
            left: 30px;
            top: 20px;
            background-color: lightblue;
            padding: 10px;
        }
    </style>
</head>
<body>
    <div>This box is moved 30px right and 20px down.</div>
</body>
</html>
```

Real-time Use Case:

- Used for slightly adjusting an element's position without affecting others.
- Example: Shifting a button slightly for better alignment.

### *3. Absolute Positioning*

Definition:

- Position: absolute; positions the element relative to its nearest positioned ancestor (not the document). If no positioned ancestor exists, it moves relative to <html>.

Syntax:

```
element {
  position: absolute;
  top: 50px;
  left: 100px;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .container {
            position: relative;
            width: 300px;
            height: 200px;
            background-color: lightgray;
        }
        .box {
            position: absolute;
            top: 50px;
            left: 100px;
            background-color: orange;
            padding: 10px;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="box">Absolute Positioned Box</div>
    </div>
</body>
</html>
```

Real-time Use Case:

- Used for dropdown menus, tooltips, and popups.
- Example: Placing an image caption over a picture.

### *4. Fixed Positioning*

Definition:

- Position: fixed; positions an element relative to the viewport (browser window). It remains in place even when scrolling.

Syntax:

```
element {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
}
```

Sample Code:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .navbar {
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            background-color: black;
            color: white;
            padding: 10px;
        }
    </style>
</head>
<body>
    <div class="navbar">Fixed Navigation Bar</div>
    <p style="margin-top: 50px;">Scroll down to see the effect.</p>
</body>
</html>
```

Real-time Use Case:

- Used for sticky headers, floating chat buttons, and sidebars.
- Example: Keeping a navbar visible while scrolling.

***5. Sticky Positioning***

Definition:

- Position: sticky; switches between relative and fixed. It acts relative until a certain scroll position is reached, then it becomes fixed.

Syntax:

```
element {
  position: sticky;
  top: 20px; /* Sticks when scrolled past 20px */
}
```

Sample Code:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .sticky-box {
            position: sticky;
            top: 10px;
            background-color: yellow;
            padding: 10px;
        }
    </style>
</head>
<body>
    <p>Scroll down to see the sticky effect.</p>
    <div class="sticky-box">I will stick when you scroll.</div>
    <p style="height: 2000px;">Long content to enable scrolling.</p>
</body>
</html>
```

Real-time Use Case:

- Used for sticky headers, side navigation, or table headers in a long list.
- Example: Keeping a "Back to Top" button visible when scrolling.

## Comparison Table

| Position | Moves Relative To | Stays in Normal Flow? | Affects Other Elements? | Scroll Behavior |
|---|---|---|---|---|
| `static` | Default position | ✅ Yes | ✅ Yes | Moves with page |
| `relative` | Itself | ✅ Yes | ❌ No | Moves with page |
| `absolute` | Nearest positioned ancestor | ❌ No | ❌ No | Moves with page |
| `fixed` | Viewport | ❌ No | ❌ No | Stays fixed |
| `sticky` | Normal flow until scrolled | ✅ Yes | ❌ No | Sticks when scrolled |

## Flexbox & Grid in CSS

### *What are Flexbox and Grid?*

Flexbox (Flexible Box Layout)

- Flexbox is a one-dimensional layout model that arranges items either in a row or a column. It provides efficient alignment and distribution of space among items, even if their sizes are unknown.

### *Grid (CSS Grid Layout)*

- Grid is a two-dimensional layout model that allows content to be arranged both in rows and columns. It is ideal for designing complex layouts with precise alignment.

### *Why Are Flexbox and Grid Important?*

| Feature | Flexbox | Grid |
|---|---|---|
| Layout Type | One-dimensional (Row or Column) | Two-dimensional (Rows and Columns) |
| Alignment | Controls space between elements | Precise row and column control |
| Best For | Small components like **navbars, cards** | Large page layouts like **dashboards** |
| Responsiveness | Easy for **dynamic content** | Ideal for **structured layouts** |

Real-Time Example

- Flexbox: Used for navigation menus, buttons, and forms.
- Grid: Used for website layouts, dashboards, and galleries.

### *1. display: flex*

Definition:

The display: flex; property turns an element into a flex container, making its direct children flexible.

Syntax:

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

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .container {
            display: flex;
            background-color: lightgray;
            padding: 10px;
        }
        .box {
            background-color: blue;
            color: white;
            padding: 20px;
            margin: 5px;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="box">Box 1</div>
        <div class="box">Box 2</div>
        <div class="box">Box 3</div>
    </div>
</body>
</html>
```

Real-Time Use Case:

- Used for horizontal navigation bars, buttons, and card layouts.
- Example: Aligning social media icons side by side in a footer.

### *2. grid-template-columns*

Definition:

- The grid-template-columns property defines column structure for a CSS Grid layout.

Syntax:

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr; /* 3 columns */
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .container {
            display: grid;
            grid-template-columns: 1fr 1fr 1fr;
            gap: 10px;
            background-color: lightgray;
            padding: 10px;
        }
        .box {
            background-color: green;
            color: white;
            padding: 20px;
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="box">Column 1</div>
        <div class="box">Column 2</div>
        <div class="box">Column 3</div>
    </div>
</body>
</html>
```

Real-Time Use Case:

- Used for gallery layouts, dashboards, and product listings.
- Example: Displaying blog posts in a grid layout.

### *3. align-items (Flexbox & Grid)*

Definition:

- The align-items property aligns flex or grid items along the cross-axis (vertical by default in flexbox).

Syntax (Flexbox):

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

Syntax (Grid):

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

Sample Code (Flexbox):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .container {
            display: flex;
            align-items: center;
            height: 200px;
            background-color: lightgray;
        }
        .box {
            background-color: red;
            color: white;
            padding: 20px;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="box">Box 1</div>
        <div class="box">Box 2</div>
        <div class="box">Box 3</div>
    </div>
</body>
</html>
```

Real-Time Use Case:

- Used for horizontal navigation bars, buttons, and card layouts.
- Example: Aligning social media icons side by side in a footer.

### *4. justify-content (Flexbox & Grid)*

Definition:

- The justify-content property controls how items are distributed along the main axis (horizontal by default in flexbox).

Syntax (Flexbox):

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

Syntax (Grid):

```
.container {
  display: grid;
  justify-content: center;
}
```

Sample Code (Flexbox):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .container {
            display: flex;
            justify-content: space-around;
            background-color: lightgray;
            padding: 10px;
        }
        .box {
            background-color: purple;
            color: white;
            padding: 20px;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="box">Box 1</div>
        <div class="box">Box 2</div>
        <div class="box">Box 3</div>
    </div>
</body>
</html>
```

Real-Time Use Case:

- Used for evenly spacing buttons, menus, or items inside a container.
- Example: Arranging icons inside a navbar with equal spacing.

## Comparison Table: Flexbox vs Grid

| Property | Flexbox | Grid |
|---|---|---|
| **Layout Type** | One-dimensional | Two-dimensional |
| **Best For** | Small UI components like menus, buttons | Full page layouts |
| **Alignment** | `align-items`, `justify-content` | `align-items`, `grid-template-columns` |
| **Responsive Design** | Easy for **dynamic layouts** | Precise **column-based** layouts |
| **Use Case** | Navbar, Cards, Form Elements | Dashboards, Image Galleries |

## CSS Transitions & Animations

***What are Transitions and Animations in CSS?***

CSS Transitions and Animations are used to create smooth visual effects without JavaScript.

### *1. Transitions*

Transitions allow changes in CSS properties over time when triggered (e.g., hover, focus).

- Best for small effects like hover changes, color fades, or button effects.
- Example: Changing a button color smoothly when hovered.

### *2. Animations*

Animations define a sequence of keyframes to create more complex movements.

- Best for moving elements, loading animations, or complex UI effects.
- Example: A bouncing ball effect or a typing effect.

Real-Time Use Cases

| Feature | Use Case |
| --- | --- |
| Transitions | Hover effects, fading images, button interactions |
| Animations | Loading spinners, pop-up effects, moving banners |

### *1. transition*

Definition:

- The transition property controls how quickly an element changes from one style to another.

Syntax:

```
.element {
  transition: property duration timing-function delay;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .button {
            background-color: blue;
            color: white;
            padding: 10px 20px;
            border: none;
            transition: background-color 0.5s ease-in-out;
        }

        .button:hover {
            background-color: red;
        }
    </style>
</head>
<body>
    <button class="button">Hover Me</button>
</body>
</html>
```

Real-Time Use Case:

- Used for smooth hover effects on buttons, links, and cards.
- Example: Button color change on hover in a navigation menu.

### *2. animation*

Definition:

- The animation property applies predefined keyframes to an element to create continuous movement.

Syntax:

```
.element {
  animation: name duration timing-function delay iteration-count direction;
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .box {
            width: 100px;
            height: 100px;
            background-color: green;
            animation: move 2s infinite alternate;
        }

        @keyframes move {
            from {
                transform: translateX(0);
            }
            to {
                transform: translateX(200px);
            }
        }
    </style>
</head>
<body>
    <div class="box"></div>
</body>
</html>
```

Real-Time Use Case:

- Used for loading indicators, animated banners, and attention-grabbing elements.
- Example: Sliding notifications on websites.

### *3. keyframes*

Definition:

- The @keyframes rule defines animation steps and controls movement.

Syntax:

```
@keyframes animation-name {
  from {
    property: value;
  }
  to {
    property: value;
  }
}
```

Sample Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        .circle {
            width: 50px;
            height: 50px;
            background-color: red;
            border-radius: 50%;
            animation: bounce 1s infinite alternate;
        }
        @keyframes bounce {
            0% {
                transform: translateY(0);
            }
            100% {
                transform: translateY(-50px);
            }
        }
    </style>
</head>
<body>
    <div class="circle"></div>
</body>
</html>
```

Real-Time Use Case:

- Used for bouncing effects, pulsing buttons, and animated logos.
- Example: Live chat button bouncing to attract user attention.

## Comparison: Transition vs Animation

| Feature | Transition | Animation |
|---|---|---|
| Trigger | Needs an event (hover, click) | Runs automatically |
| Duration | Controlled by `transition` | Defined by `@keyframes` |
| Complexity | Simple property changes | Advanced movement & effects |
| Use Case | Button hover effects, fading images | Loading spinners, pop-up effects |

## CSS Media Queries

### *What are Media Queries?*

CSS Media Queries allow web pages to adapt to different screen sizes and devices.

- They help create responsive designs that look good on desktops, tablets, and mobile devices.
- They use breakpoints to change styles based on the screen width, height, or resolution.

### *Real-Time Use Case*

- Making a website mobile-friendly by adjusting font size, layout, or visibility.
- Hiding/showing elements based on screen size.
- Changing navigation menus from horizontal (desktop) to vertical (mobile).

Definition

- The @media rule applies different CSS styles based on the device's screen size.

Syntax

```
@media screen and (max-width: 768px) {
  /* CSS rules for screens smaller than 768px */
}
```

- max-width: 768px → Applies styles when the screen width is 768px or smaller.

- min-width: 769px → Applies styles when the screen width is 769px or larger.

Sample Code

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <style>
        body {
            font-size: 20px;
            background-color: lightblue;
        }

        @media screen and (max-width: 768px) {
            body {
                font-size: 16px;
                background-color: lightcoral;
            }
        }
    </style>
</head>
</head>
<body>
    <p>Resize the window to see the background color and font size change.</p>
</body>
</html>
```

How it Works?

- On large screens (above 768px), the background is light blue, and text is 20px.
- On small screens (below 768px), the background changes to light coral, and text becomes 16px.

Real-Time Use Cases

| Use Case | Description |
| --- | --- |
| **Mobile-Friendly Design** | Adjusts layout, font size, and images for better readability on smaller screens. |
| **Responsive Navigation** | Converts a desktop menu into a mobile-friendly dropdown. |
| **Hiding Elements** | Hides sidebars or less important elements on smaller screens. |

Example: Responsive Navigation Menu

```css
nav {
    display: flex;
    justify-content: space-around;
    background-color: navy;
    padding: 10px;
}

.menu {
    display: flex;
    gap: 20px;
}

@media screen and (max-width: 768px) {
    .menu {
        flex-direction: column;
        gap: 10px;
    }
}
```

- Desktop: Menu items appear in a row.
- Mobile: Menu items appear in a column for better readability.