Cascading Style Sheets

Cascading Style Sheets, is a style sheet language used to describe the visual presentation of a document written in HTML. CSS describes how elements should be rendered on screen. It enables web designers to control the layout, colors, fonts, and other visual aspects of multiple web pages all at once.

In Cascading Style Sheets (CSS), cascading is the logic that browsers use to determine which CSS rulesets are most important when they conflict with each other. The cascade is central to CSS and is used to solve conflicts when multiple CSS rules apply to an HTML element. For example, if a button is styled with both `button { color: red; }` and `button { color: blue; }`, the cascade will determine that the button's text should be blue.

How to insert css into html (3 ways)

- Inline styling
- Internal styling
- External styling

Inline Styling: In inline styling CSS styles directly apply to the individual HTML elements using the style attribute.

```
<h1 style="color:blue;text-align:center;">This is a heading</h1>
```

Internal Styling: Internal styling in CSS refers to the method of embedding CSS rules directly within the HTML document using the <style> tag.

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: linen;
}

h1 {
```

```
  color: maroon;
  margin-left: 40px;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

External Styling: External CSS is used to style multiple HTML pages with a single style sheet. External CSS contains a separate CSS file with a **.css** extension.

How to link external stylesheet into html document

```
<head>
<link rel="stylesheet" href="mystyle.css">
</head>
```

How to create external stylesheet

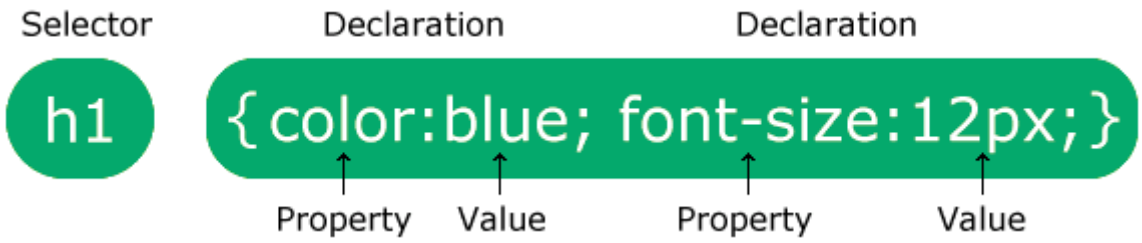
Create a file in a folder with .css extension and write the css code.

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;
  margin-left: 20px;
}
```

CSS SELECTORS

- Simple Selectors
- Pseudo Selectors
- Combinators
- Attribute Selectors



SIMPLE SELECTORS

A simple selector in CSS refers to the most basic form of selecting elements on a webpage. It can target elements based on various criteria like tag name, class, id, attributes, or their relationship with other elements.

1. Element Selectors
2. Universal Selectors
3. Id selectors
4. class selectors
5. grouping selectors

1.Element Selectors

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Element Selectors Example</title>
<style>
    /* Element Selector */
    div {
        color: blue;
        font-size: 18px;
```

```
    }
    p {
        color: green;
        font-size: 14px;
    }
    h1 {
        color: red;
        font-size: 24px;
    }
</style>
</head>
<body>
    <div>This is a div element</div>
    <p>This is a paragraph element</p>
    <h1>This is an h1 heading</h1>
</body>
</html>
```

2.Universal Selectors

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Universal Selector Example</title>
<style>
    /* Universal Selector Represented With *, Will Apply the Styles on all tags */
    * {
```

```
        margin: 0;
        padding: 0;
        box-sizing: border-box;
    }

    body {
        font-family: Arial, sans-serif;
    }

    /* Specific Element Styling */
    div {
        background-color: lightblue;
        padding: 10px;
        margin-bottom: 20px;
    }

    p {
        background-color: lightgreen;
        padding: 5px;
        margin-bottom: 10px;
    }
</style>
</head>
<body>
    <div>
        <h1>Universal Selector Example</h1>
        <p>This is a paragraph inside a div.</p>
```

```
    </div>
    <p>This is a standalone paragraph.</p>
</body>
</html>
```

3.Id selectors

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>ID Selectors Example</title>
<style>
    /* ID Selector, these selectors are represented with # and id's are unique In CSS */
    #mainHeading {
        color: blue;
        font-size: 24px;
    }
    #content {
        background-color: lightgray;
        padding: 10px;
    }
    #footer {
        background-color: darkgray;
        color: white;
        text-align: center;
        padding: 5px;
    }
```

```
</style>
</head>
<body>
    <h1 id="mainHeading">Main Heading</h1>
    <div id="content">
        <p>This is some content inside a div with ID "content".</p>
    </div>
    <div id="footer">
        Footer Section
    </div>
</body>
</html>
```

4.class selectors

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Class Selectors Example</title>
<style>
    /* Class Selector are represented with ( . )  */
    .highlight {
        color: red;
        font-weight: bold;
    }
    .italic {
        font-style: italic;
```

```
    }
    .underline {
        text-decoration: underline;
    }
</style>
</head>
<body>
    <p class="highlight">This text is highlighted</p>
    <p class="italic">This text is italicized</p>
    <p class="underline">This text is underlined</p>
</body>
</html>
```

5.Grouping selectors

Grouping selectors in CSS allow you to apply the same styles to multiple selectors at once.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Grouping Selectors Example</title>
<style>
    /* Grouping Selectors */
    h1, h2, h3 {
        color: blue;
        font-style: italic;
    }
    .info, .warning, .error {
```

```
        border: 1px solid;
        padding: 5px;
        margin-bottom: 10px;
    }
    /* Specific styles for individual selectors */
    .info {
        color: green;
        background-color: lightgreen;
    }
    .warning {
        color: orange;
        background-color: lightyellow;
    }
    .error {
        color: red;
        background-color: lightcoral;
    }
</style>
</head>
<body>
    <h1>Heading 1</h1>
    <h2>Heading 2</h2>
    <h3>Heading 3</h3>
    <div class="info">Information message</div>
    <div class="warning">Warning message</div>
    <div class="error">Error message</div>
</body>
```

</html>

Ways to insert colors in css

1. Color Names:

You can use predefined color names like red, blue, green, etc.

```
.example {
  color: red;
  background-color: blue;
}
```

2. Hexadecimal Notation: Value varies from (a-f,0-9)

```
.example {
  color: #ff0000; /* Red */
  background-color: #0000ff; /* Blue */
}
```

3. RGB Function: Red value(0-255),green value(0-255),blue value(0-255)

```
.example {
  color: rgb(255, 0, 0); /* Red */
  background-color: rgb(0, 0, 255); /* Blue */
}
```

4. HSL Function: H stands for hue value varies between (0-360),saturation(0-100%), Lightness(0-100%)

```
.example {
  color: hsl(0, 100%, 50%); /* Red */
  background-color: hsl(240, 100%, 50%); /* Blue */
}
```

5. Using Variables:

```
:root {
  --primary-color: #ff0000; /* Red */
}
.example {
```

```
  color: var(--primary-color);
}
```

Background properties

Background properties are properties used to control the appearance of an element's background. They allow you to set various attributes such as color, image, position, size, and repetition of the background of an HTML element.

1. **background-color:** Sets the color of the background.

```
div {
   background-color: lightblue;
}
```

2. **background-image:** Sets an image as the background.

```
div {
   background-image: url('example.jpg');
}
```

3. **background-repeat:** Specifies how the background image should be repeated (e.g., repeat, repeat-x, repeat-y, no-repeat).

```
div {
   background-image: url('example.jpg');
   background-repeat: no-repeat;
}
//repeat-x  repeats the image on x axis
//repeat-y  repeats the image on y axis
```

4. **background-position:** Sets the starting position of the background image. such as top ,right, bottom, left, center, right.

```
div {
   background-image: url('example.jpg');
   background-position: top right;
}
```

5. **background-size:** Sets the size of the background image.

   ```
   div {
      background-image: url('example.jpg');
      background-size: cover;
   }
   ```

   cover keyword is used to specify that the background image should cover the entire container while maintaining its aspect ratio. This means that the background image will be scaled up or down as necessary to ensure it covers the entire background area of the element.

   contain keyword is used to specify that the background image should be scaled to fit within the container while preserving its aspect ratio. This means that the entire background image will be visible within the background area of the element, without being cropped or stretched.

6. **background-attachment:** Specifies whether the background image scrolls with the content or remains fixed.

   ```
   div {
      background-image: url('example.jpg');
      background-attachment: fixed;
   }
   ```

Background shorthand property

background: [background-color] [background-image] [background-repeat] [background-attachment] [background-position]

```
.element {
   background: lightblue   url('example.jpg')   repeat-x   fixed  top right;
}
```

Background with multiple images and overlay color:

```
div {
   background-image: url('pattern.jpg'), url('overlay.png');
```

```
    background-position:top left, top right ;
}
//here the first background image displayed on the top left
//here the secondbackground image displayed on the top right
```

CSS text properties

- Text align
- Text decoration
- Text transform
- Text spacing
- Text shadow
- Text direction

color

it is used to specify the color for a text

```
h1{
  color: red;
}
```

Text align

Aligns the text inside an element

1)center: It is used to align the text to center within its container

```
.text {
    text-align: center;
}
```

2)right: it is used to align the text to right within its container

```
.text {
    text-align:right;
```

```
}
```

3)left: it is used to align the text to left within its container and it is default one

```
.text {
   text-align:left;
}
```

4)justify: it lines up the content on the left and right edges of it container

```
.text {
   text-align:justify;
}
```

4)text-align-last: It is used to align the last line of the text. We can apply center , right , left, justify

```
.text {
   text-align-last: center;
}
```

Text decoration

It was used to add decorative styling to text within the html element

```
//it adds underline to the text
.text {
   text-decoration-line: underline;
}
//it adds line above to the text
.text {
   text-decoration-line:overline;
```

```
}
//it draws line through the middle of the text
.text {
    text-decoration-line:line-through;
}
//it removes the text decoration
.text {
    text-decoration-line:none;
}
```

text-decoration-color

It is used to specify the color of the decoration added to the text

```
.text {
    text-decoration: underline;
    text-decoration-color: blue;
}
```

text-decoration-style

it is used to specify the style of the line used for text decoration such as solid, double, dotted,wavy etc..same styles as like borders

```
.text{
    text-decoration: underline;
    text-decoration-style: dashed;
}
```

Text-decoration shorthand property

```
.text{
    text-decoration:1px solid red line-through;
   }
```

Text transform

it is used to control the captialization of the text

```
//transform text to the uppercase
.text {
    text-transform: uppercase;
}

//transform text to the lowercase
.text {
    text-transform:lowercase;
}

//capitalize the first letter of each word in a text
.text {
    text-transform:capitalize;
}
```

Text indent

The text-indent property in CSS is used to specify the indentation of the first line of text in a block-level element.

```
p{
    text-indent: 30px;
}
```

Text direction

The direction and unicode-bidi properties in CSS are used for controlling the directionality of text, especially when dealing with languages that are read from right to left (RTL), such as Arabic or Hebrew

```
p{
```

```
    direction:rtl;
    unicode-bidi: bidi-override;
}
```

Text spacing:

It adjusts the spaces between the characters and words

1)letter-spacing: it controls the space between the characters in a text

```
.text {
   letter-spacing: 2px;
}
```

2)word-spacing: it controls the space between words in a text

```
.text {
  word-spacing: 5px;
}
```

3) white-space: it controls how whitespace inside element is handled . it can be used to collapse and preserve the whitespace

```
.preserve-space {
   white-space:nowrap; //or wrap //pre-wrap
}
```

4)line-height: it is used to specify the space between lines

Text shadow

It is used to add shadow effects to the text

Syntax

text-shadow: **[horizontal offset] [vertical offset] [blur radius] [color];**

- **Horizontal offset:** Specifies the horizontal distance of the shadow. Positive values move the shadow to the right, while negative values move it to the left.
- **Vertical offset:** Specifies the vertical distance of the shadow. Positive values move the shadow downwards, while negative values move it upwards.
- **Blur radius:** Specifies the amount of blurring applied to the shadow. Larger values create a more diffused shadow.
- **Color:** Specifies the color of the shadow.

```
.shadow {
   text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
}
```

CSS box-shadow Property

The box-shadow property in CSS is used to add shadow effects around an element's frame

```
selector {
   box-shadow: [h-offset] [v-offset] [blur] [spread] [color] [inset];
}
.element{
        box-shadow: 0px 0px 5px 5px orange inset;


}
```

CSS Fonts

Font properties are used to control the appearance of the font

1)font-family: specify the font family type used to the text

```
p {
   font-family: Arial, sans-serif;
}
```

2)font-size: it specifies the size of the font

```
h1 {
   font-size: 24px;
}
```

3)font-style : sets the style of the font such as italic or normal

```
em {
   font-style: italic;
}
```

4)font-weight: sets the boldness and thickness of the font

```
strong {
   font-weight: bold;
}
```

5)font-variant: contols the usage of small caps

```
.small-caps {
   font-variant: small-caps;
}
```

Web safe fonts

web-safe fonts are fonts that are widely available across different operating systems and web browsers. These fonts are considered safe to use in web design.

Some examples of commonly used web safe fonts include:

1. Arial
2. Helvetica
3. Times New Roman
4. Georgia
5. Courier New
6. Verdana
7. Trebuchet MS
8. Comic Sans MS

Fallback fonts

Fallback fonts, also known as backup fonts or alternative fonts, are fonts specified in CSS that the browser uses as substitutes if the primary font specified is not available or cannot be displayed for some reason.

```
body {
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
}
```

In this example, the browser will first attempt to render the text using the "Helvetica Neue" font. If this font is not available, it will try to use the "Helvetica" font. If "Helvetica" is also unavailable, it will try "Arial". If none of the specified fonts are available, it will default to a generic sans-serif font.

UNDER-STANDING BOX-MODEL

The box model is a fundamental concept in CSS that describes how elements are rendered on a webpage, including their content, padding, border, and margin.

Content: This is the actual content of the element, such as text, images, or other media.

Padding: Padding is the space between the content and the element's border. It adds internal space within the element. Padding can be specified using CSS properties like padding-top, padding-right, padding-bottom, and padding-left, or using the shorthand property padding.

Border: The border surrounds the padding and content of the element. It can be styled using properties like border-width, border-style, and border-color. Borders can have different styles such as solid, dashed, dotted, etc.

Margin: Margins are the space outside the element, separating it from other elements. It creates space between the element and its surrounding elements. Margins can be set using CSS properties like margin-top, margin-right, margin-bottom, and margin-left, or using the shorthand property margin.

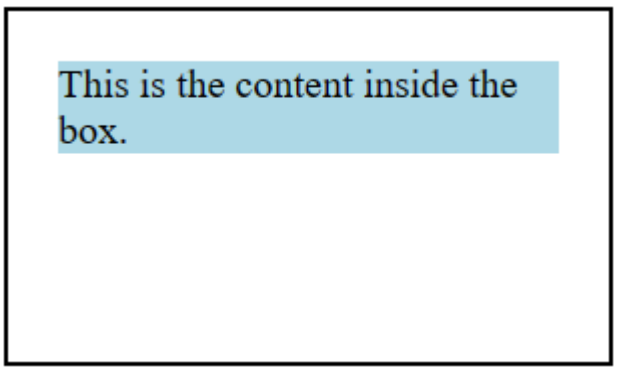
The total width and height of an element are calculated as follows:

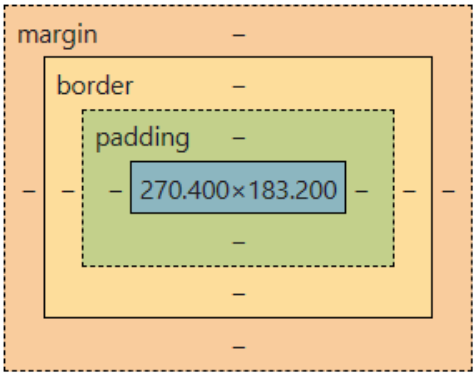
Total Width=Content Width+Padding+Border+Margin

Total Height=Content Height+Padding+Border+Margin

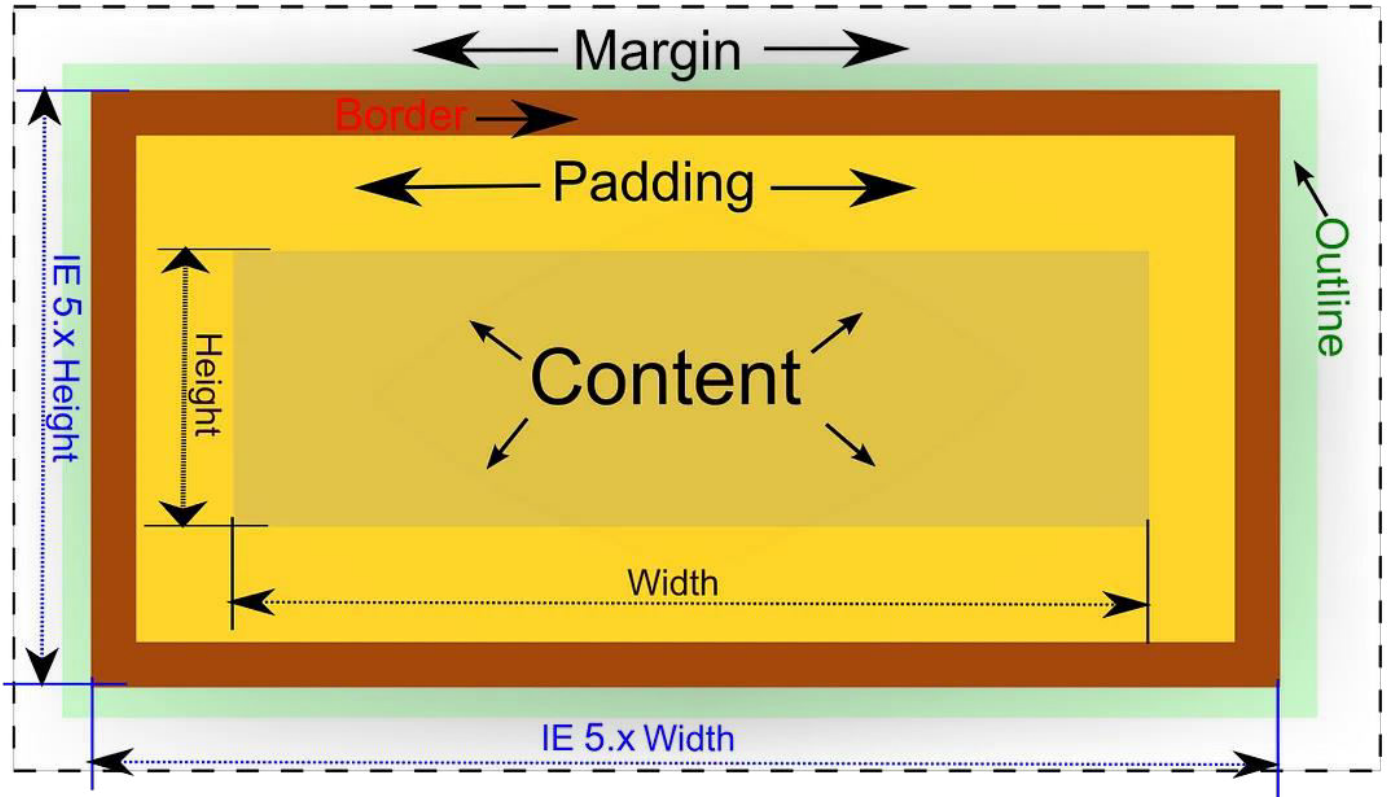
```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Box Model Example</title>
<style>
    .box {
        width: 200px;
        height: 100px;
        padding: 20px;
        border: 2px solid black;
```

```
        margin: 20px;
    }    .content {
        background-color: lightblue;
    }
</style>
</head>
<body>
    <div class="box">
        <div class="content">
            This is the content inside the box.
        </div>
    </div>
</body>
</html>
```





CSS MARGINS, PADDINGS, WIDTH AND HEIGHT, BORDER REFERENCE IMAGE FOR BELOW TOPICS



MARGINS

• Margin Property is used to create space around element's border.

• It can be set for all sides or individually for each side (top, right, bottom, left).

Different Properties of Margins:

1.margin-top: 10px;

2.margin-right: 20px;

3.margin-bottom: 15px;

4.margin-left: 5px;

5.margin: 10px; /* All margins are 10px */

6.margin: 10px 20px; /* Top/bottom: 10px, Right/left: 20px */

7.margin: 10px 20px 15px; /* Top: 10px, Right/left: 20px, Bottom: 15px */

8.margin: 10px 20px 15px 5px; /* Top, Right, Bottom, Left */

Note: Pixels Can Be Changed and Negative Pixels Are Also Allowed

• We Can Set Margins in different ways:

1. Setting Margin for All Sides:

```
.element {
```

```
 margin: 10px; /* Applies 10 pixels margin to all sides */
}
```

2. Setting Margin for Individual Sides:

```
.element {
margin-top: 10px;
margin-right: 20px; /*Applies different pixels for each side */
margin-bottom: 15px;
margin-left: 5px;
}
```

3. Shorthand for Margin:

```
.element {
 margin: 10px 20px 15px 5px; /* top right bottom left */
}
```

NOTE:

• If you provide two values, they apply to top/bottom and right/left respectively.

• If you provide three values, they apply to top, right/left, and bottom respectively

4. Negative Margin

```
.element {
 margin-left: -10px; /* Moves the element 10 pixels to the left */
}
```

5. Auto Margin

```
.element {
 margin: auto; /* Centre's the element horizontally */
}
```

PADDINGS

• Padding Property is used to create space inside an element, between the elements content and its border.

• Padding can be set for all sides or individually for each side (top, right, bottom, left)

Different Properties of Padding

1. padding-top: 10px;
2. padding-right: 20px;
3. padding-bottom: 15px;
4. padding-left: 5px;
5. padding: 10px; /* All paddings are 10px */
6. padding: 10px 20px; /* Top/bottom: 10px, Right/left: 20px */
7. padding: 10px 20px 15px; /* Top: 10px, Right/left: 20px, Bottom: 15px */
8. padding: 10px 20px 15px 5px; /* Top, Right, Bottom, Left */

Note: Pixels Can Be Changed and Negative Pixels Are Not Allowed

• We Can Set Paddings in different ways:

1. Setting Padding for All Sides:

```
.element {
 Padding : 10px; /* Applies 10 pixels padding to all sides */
}
```

2. Setting Padding for Individual Sides:

```
.element {
 padding-top: 10px;
 padding-right: 20px;
 padding-bottom: 15px;
 padding-left: 5px;
}
```

3. Shorthand for Padding:

```
.element {
 padding: 10px 20px 15px 5px; /* top right bottom left */
}
```

• If you provide two values, they apply to top/bottom and right/left respectively.

• If you provide three values, they apply to top, right/left, and bottom respectively.

PADDING AND BACKGROUND COLOR:

• Padding affects the background color of an element. If you set a background color on an element, the padding area will also be

filled with that background color.

BACKGROUND PROPERTY IN CSS

The `background` property in CSS is a shorthand for setting the individual background properties for an element. These properties include `background-color`, `background-image`, `background-repeat`, `background-attachment`, `background-position`, `background-size`, `background-origin`, and `background-clip`.

Individual Background Properties:

1. **background-color**: Sets the background color of an element.

2. **background-image**: Sets one or more background images for an element.

3. **background-repeat**: Sets how background images are repeated.

   - `repeat`: The background image is repeated both vertically and horizontally.

   - `repeat-x`: The background image is repeated only horizontally.

   - `repeat-y`: The background image is repeated only vertically.

   - `no-repeat`: The background image is not repeated.

   - `space`: The background image is repeated as much as possible without clipping. The first and last images are pinned to either side of the element, and whitespace is distributed evenly between the images.

- `round`: The background image is repeated and squished or stretched to fill the space.

4. **background-attachment**: Sets whether a background image scrolls with the rest of the page or is fixed.

- `scroll`: The background image scrolls with the page.

- `fixed`: The background image is fixed in the viewport.

- `local`: The background image scrolls with the element's contents.

5. **background-position**: Sets the starting position of a background image.

- Values can be specified in length units (px, em, etc.) or percentages.

- Keywords: `top`, `bottom`, `left`, `right`, `center`.

6. **background-size**: Specifies the size of the background image.

- `auto`: Default. The background image is displayed in its original size.

- `cover`: Scales the background image to cover the entire container.

- `contain`: Scales the background image to be as large as possible without clipping.

- Can also specify dimensions like `background-size: 100px 50px`.

7. **background-origin**: Specifies the positioning area of the background images.

- `padding-box`: Default. The background is positioned relative to the padding box.

- `border-box`: The background is positioned relative to the border box.

- `content-box`: The background is positioned relative to the content box.

8. **background-clip**: Specifies the painting area of the background.

- `border-box`: The background extends to the outside edge of the border.

- `padding-box`: The background extends to the outside edge of the padding.

- `content-box`: The background extends to the edge of the content box.

Syntax:

background: [background-color] [background-image] [background-repeat] [background-attachment] [background-position] / [background-size] [background-origin] [background-clip];

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Background Property Example</title>
<style>
  .example {
    width: 300px;
    height: 300px;
    background:
      #ffcc00 /* background-color */
      url('example-image.jpg') /* background-image */
      no-repeat /* background-repeat */
      fixed /* background-attachment */
      center /* background-position */
      / cover /* background-size */
      padding-box /* background-origin */
      border-box; /* background-clip */
    border: 1px solid #000;
    padding: 20px;
  }
</style>
</head>
```

```
<body>

<div class="example">This is a div with a background image.</div>

</body>
</html>
```

- **background-color: #ffcc00**: Sets the background color to a yellow shade.
- **background-image: url('example-image.jpg')**: Sets the background image.
- **background-repeat: no-repeat**: Ensures the background image is not repeated.
- **background-attachment: fixed**: Fixes the background image in the viewport.
- **background-position: center**: Centers the background image.
- **background-size: cover**: Scales the background image to cover the entire container.
- **background-origin: padding-box**: Positions the background relative to the padding box.
- **background-clip: border-box**: Paints the background up to the border box.

Example :

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

```
<title>Background Property One-Sheet</title>
<style>
  .example {
    width: 300px;
    height: 300px;
    background:
      #ffcc00 /* background-color */
      url('example-image.jpg') /* background-image */
      no-repeat /* background-repeat */
      fixed /* background-attachment */
      center /* background-position */
      / cover /* background-size */
      padding-box /* background-origin */
      border-box; /* background-clip */
    border: 1px solid #000;
    padding: 20px;
  }
</style>
</head>
<body>
<div class="example">This is a div with a background image.</div>
</body>
</html>
```

WIDTH AND HEIGHT

Width Properties:

• width: Sets the width of an element. You can use values like pixels (px), percentages (%), viewport width (vw), and more.

• min-width: Specifies the minimum width that an element can have.

Useful for responsive design to prevent elements from becoming too narrow.

• max-width: Specifies the maximum width that an element can have.

Useful for responsive design to limit how wide an element can become.

Height Properties

• height: Sets the height of an element. Similar to width, you can use values like pixels (px), percentages (%), viewport height (vh), and more.

• min-height: Specifies the minimum height that an element can have.

Useful for responsive design to prevent elements from becoming too short.

• max-height: Specifies the maximum height that an element can have.

Useful for responsive design to limit how tall an element can become.

1.Setting Width and Height:

To set the width of an element:

```
.element {
 width: 200px; /* Sets the width to 200 pixels */
}
```

To set the height of an element:

```
.element {
 height: 100px; /* Sets the height to 100 pixels */ }
```

2. Percentage Width and Height:

```
.element {
 width: 50%; /* Sets the width to 50% of its containing element */
 height: 75%; /* Sets the height to 75% of its containing element */
}
```

3. Minimum and Maximum Width and Height:

You can specify minimum and maximum values for width and height using min-width, max-width, min-height, and max-height properties. This is useful for creating responsive layouts or preventing elements from becoming too small or too large:

```
.element {
 min-width: 100px; /* Sets the minimum width to 100 pixels */
 max-width: 500px; /* Sets the maximum width to 500 pixels */
 min-height: 50px; /* Sets the minimum height to 50 pixels */
 max-height: 300px; /* Sets the maximum height to 300 pixels */
}
```

4. Auto Width and Height:

You can use auto to let the browser automatically determine the width or height based on the content:

```
.element {
 width: auto; /* Automatically adjusts the width based on content */
 height: auto; /* Automatically adjusts the height based on content */
}
```

5. Viewport Units:

CSS also supports viewport units (vw and vh) for setting width and height relative to the viewport size:

```
.element {
 width: 50vw; /* Sets the width to 50% of the viewport width */
 height: 75vh; /* Sets the height to 75% of the viewport height */
}
```

BORDERS

• Border Property is the primary way to set border properties around an element.

• We Have Many Controls Over Border

1. Border Style: Specifies the style of the border, such as solid, dashed, dotted, double, groove, ridge, inset, outset, or none.

```
.element {
 border-style: solid; /* Solid border style */
}
```

2. Border Width: Sets the width of the border. You can use values like pixels (px), ems (em), or keywords like thin, medium, thick.

```
.element {
 border-width: 2px; /* Border width of 2 pixels */
}
.element {
 border-bottom-width: 2px; /* Sets the bottom border width to 2 pixels
*/
}
```

3. Border Color: Sets the color of the border. You can use color names, hex codes, RGB values, or the currentColor keyword

```
.element {
 border-color: #333; /* Border color as hex code */
}
```

4. Border Shorthand: The border shorthand property allows you to set all border properties (style, width, color) in one declaration.

```
.element {
 border: 2px solid #333; /* Width, Style, Color */
}
```

5. Individual Borders: You can set borders for specific sides of an element using individual properties like border-top, border-right, border-bottom, and border-left. This allows you to customize borders differently for each side if needed.

```
.element {
 border-top: 2px solid #333; /* Top border */
 border-bottom: 1px dashed #999; /* Bottom border */
}
```

6. Border Radius: Creates rounded corners for borders. You can specify the radius of each corner separately (border-radius-top-left, border-radius-top-right,border-radius-bottom-right, border-radius-bottom-left) or use a single value for all corners.

```
.element {
 border-radius: 5px; /* Applies rounded corners to all corners */
}
```

Example with individual corner radius:

```
.element {
 border-top-left-radius: 10px;
 border-top-right-radius: 5px;
 border-bottom-right-radius: 20px;
 border-bottom-left-radius: 15px;
}
```

7. Border Image: Allows you to use an image as a border instead of a solid color.

This property is used in conjunction with border-width, border-style, and border-color to define the border image.

```
.element {
 border-image: url(border.png) 30 round; /* Border image */
}
```

DISPLAY PROPERTIES AND BEHAVIOUR

1. display: none;

When you set an element's display to none, it completely removes the element from the document layout. The element will not take up any space on the page, and it won't be visible or interactable.

```
.hidden-element {
 display: none;
}
```

The visibility: hidden; CSS property is used to hide an element while still taking up space in the layout. This means that the element becomes invisible, but it still occupies the same space as if it were visible. The visibility property does not affect the layout of other elements around the hidden element.

```
.hidden-element {
```

visibility: hidden;

}

2. display: inline;

Elements with display: inline; behave like inline elements. They flow along with the surrounding text or other inline elements. Inline elements do not start on a new line and only take up as much width as necessary for their content.

.inline-element {

display: inline;

}

3. display: inline-block;

inline-block; behave like inline elements in terms of flowing with the text, but they also have block-level properties like setting width, height, margins, and paddings.

Inline-block elements start on a new line like block-level elements but flow inline with the surrounding content.

.inline-block-element {

display: inline-block;

}

4. display: block;

behave as block-level elements. They start on a new line and occupy the

full width available to them unless specified otherwise.

Block-level elements can have margins, paddings, and widths set, and they

stack vertically one after another.

.block-element {

display: block;

}

DIFFERENCE BETWEEN INLINE AND INLINE-BLOCK

The main difference between inline and inline-block display values in CSS lies in how they interact with other elements and how they handle properties like width, height, margins, and padding.

1. Inline Elements (display: inline;): Inline elements flow along with the surrounding text or other inline elements. They do not start on a new line, and they only take up as much width as necessary for their content.

Inline elements do not have properties like width, height, margins, or paddings applied to them directly. Any attempt to set these properties will not have any effect.

Examples of inline elements include , <a>, , , <img>, <input>, etc.

2. Inline-Block Elements (display: inline-block;):

Inline-block elements behave like inline elements in terms of flowing with the text, but they also have block-level properties like setting width, height, margins, and paddings.

Inline-block elements start on a new line like block-level elements but flow inline with the surrounding content.

Unlike inline elements, inline-block elements can have properties like width, height, margins, paddings, and they will be respected by the browser.

Examples of inline-block elements include , with display: inline-block;, <img> with display: inline-block;, etc.

Inline Element

```
.inline-element {
 display: inline;
 width: 100px; /* This width will not have any effect */
 margin: 10px; /* This margin will not have any effect */
 padding: 5px; /* This padding will not have any effect */
}
```

Inline-Block Element

```
.inline-block-element {
 display: inline-block;
```

```
width: 100px; /* Width will be applied */
margin: 10px; /* Margin will be applied */
padding: 5px; /* Padding will be applied */
}
```

Css combinators

CSS combinators are used to define the relationship between selectors, allowing you to target elements based on their relationship to other elements in the DOM.

There are four main types of combinators:

Descendant Combinator (Space): This combinator is represented by a space between two selectors. It selects all elements that are descendants of a specified element. For example, div p selects all <p> elements inside elements.

```
div p {
color: blue;
}
```

Child Combinator (>): This combinator is represented by the > symbol between two selectors. It selects only those elements matched by the second selector that are direct children of elements matched by the first. For example, div > p selects all <p> elements where the parent is a element.

```
div > p {
  color: red;
}
```

Adjacent Sibling Combinator (+): This combinator is represented by the + symbol between two selectors. It selects only those elements matched by the second selector that are the next sibling

element of the first selector. For example, div + p selects the first <p> element that is placed immediately after elements.

```
div + p {
  color: green;
}
```

General Sibling Combinator (~): This combinator is represented by the ~ symbol between two selectors. It selects every element matched by the second selector that are preceded by an element matched by the first selector. For example, p ~ ul selects every <ul> element that are preceded by a <p> element.

```
p ~ ul {
  color: purple;
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>CSS Combinators Example</title>
<style>
  /* Descendant Combinator (Space) */
  div p {
    color: blue;
  }
```

```
    /* Child Combinator (>) */
    div > p {
        color: red;
    }
    /* Adjacent Sibling Combinator (+) */
    div + p {
        color: green;
    }
    /* General Sibling Combinator (~) */
    p ~ ul {
        color: purple;
    }
</style>
</head>
<body>
    <div>
        <p>This is a descendant of div and should be blue.</p>
        <span>
            <p>This is a child of div, but not a direct child, so it won't be red.</p>
        </span>
        <p>This is a direct child of div and should be red.</p>
```

```
        <p>This is an adjacent sibling of div and should be green.</p>
        <span>
            <p>This is a sibling of the previous paragraph, but not adjacent, so it won't be green.</p>
        </span>
        <ul>
            <li>This ul is preceded by a p, so it should be purple.</li>
        </ul>
        <p>Another paragraph to test sibling combinator.</p>
        <ul>
            <li>This ul is not preceded by a p, so it won't be purple.</li>
        </ul>
    </div>
</body>
</html>
```

CSS POSITIONS

The position CSS property sets how an element is positioned in a document. The top, right, bottom, and left properties determine the final location of positioned elements.

Types of Positions

1.Static Positioning:

- The element is positioned according to the Normal Flow of the document. The top, right, bottom, left, and z-index properties have no effect. This is the default value.
- Default position for all elements.

Properties: position: static;

2.Relative Positioning:

- The element is positioned according to the normal flow of the document, and then offset relative to itself based on the values of top, right, bottom, and left. The offset does not affect the position of any other elements; thus, the space given for the element in the page layout is the same as if position were static.
- Element is positioned relative to its normal position.

Properties: position: relative;

top: value;,

right: value;

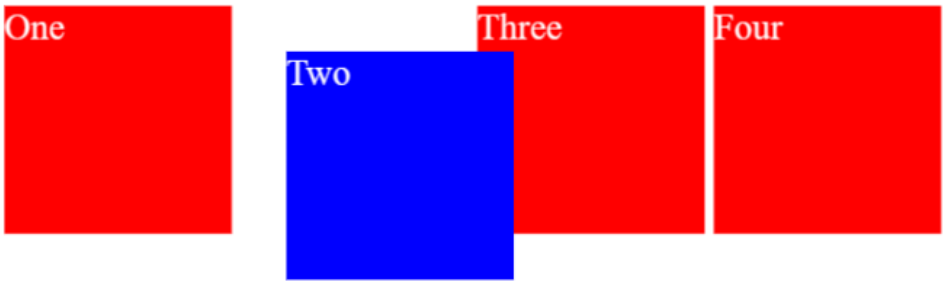
bottom: value;

left: value;

Example:

```
<style>
  .box {
    display: inline-block;
```

```
        width: 100px;
        height: 100px;
        background: red;
        color: white;
    }
    #two {
        position: relative;
        top: 20px;
        left: 20px;
        background: blue;
    }
</style>
 <div class="box" id="one">One</div>
<div class="box" id="two">Two</div>
<div class="box" id="three">Three</div>
<div class="box" id="four">Four</div>
```



3.Absolute Positioning:

- Element is positioned relative to its nearest positioned ancestor.
- If no positioned ancestor, it's positioned relative to the initial containing block (usually the viewport).

Properties: position: absolute;

```
top: value;
right: value;
bottom: value;
left: value;
```

Example:

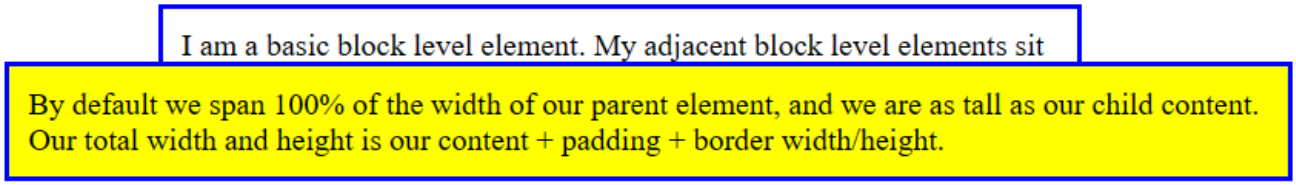
```
<style>
    body {
      width: 500px;
      margin: 0 auto;
    }
    p {
      border: 3px solid blue;
      padding: 10px;
      margin: 10px;
    }
    .positioned {
      position: absolute;
      background-color: yellow;
      top: 30px;
      left: 30px;
    }
</style>
<p>
```

I am a basic block level element. My adjacent block level elements sit on new lines below me.

By default we span 100% of the width of our parent element, and we are as tall

as our child content. Our total width and height is our content + padding +

border width/height.



4.Fixed Positioning:

- The element is removed from the normal document flow, and no space is created for the element in the page layout. The element is positioned relative to its initial containing block, which is the viewport in the case of visual media. Its final position is determined by the values of top, right, bottom, and left.
- Element is positioned relative to the viewport (browser window).
- It stays fixed even when the page is scrolled.

Properties: position: fixed;

top: value;

right: value;

bottom: value;

left: value;

Example:

<style>

```
    * {
        box-sizing: border-box;
    }

    .box {
        width: 100px;
        height: 100px;
        background: red;
        color: white;
    }

    #one {
        position: fixed;
        top: 80px;
        left: 10px;
        background: blue;
    }

    .outer {
        width: 500px;
        height: 300px;
        overflow: scroll;
        padding-left: 150px;
    }
</style>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam congue tortor

eget pulvinar lobortis. Vestibulum ante ipsum primis in faucibus orci luctus

et ultrices posuere cubilia Curae; Nam ac dolor augue. Pellentesque mi mi,

laoreet et dolor sit amet, ultrices varius risus. Nam vitae iaculis elit.

Aliquam mollis interdum libero. Sed sodales placerat egestas. Vestibulum ut

arcu aliquam purus viverra dictum vel sit amet mi. Duis nisl mauris, aliquam

sit amet luctus eget, dapibus in enim. Sed velit augue, pretium a sem

aliquam, congue porttitor tortor. Sed tempor nisl a lorem consequat, id

maximus erat aliquet. Sed sagittis porta libero sed condimentum. Aliquam

finibus lectus nec ante congue rutrum. Curabitur quam quam, accumsan id

ultrices ultrices, tempor et tellus.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam congue tortor

eget pulvinar lobortis. Vestibulum ante ipsum primis in faucibus orci luctus

et ultrices posuere cubilia Curae; Nam ac dolor augue. Pellentesque mi mi,

laoreet et dolor sit amet, ultrices varius risus. Nam vitae iaculis elit.

Aliquam mollis interdum libero. Sed sodales placerat egestas. Vestibulum ut

arcu aliquam purus viverra dictum vel sit amet mi. Duis nisl mauris, aliquam

sit amet luctus eget, dapibus in enim. Sed velit augue, pretium a sem

aliquam, congue porttitor tortor. Sed tempor nisl a lorem consequat, id

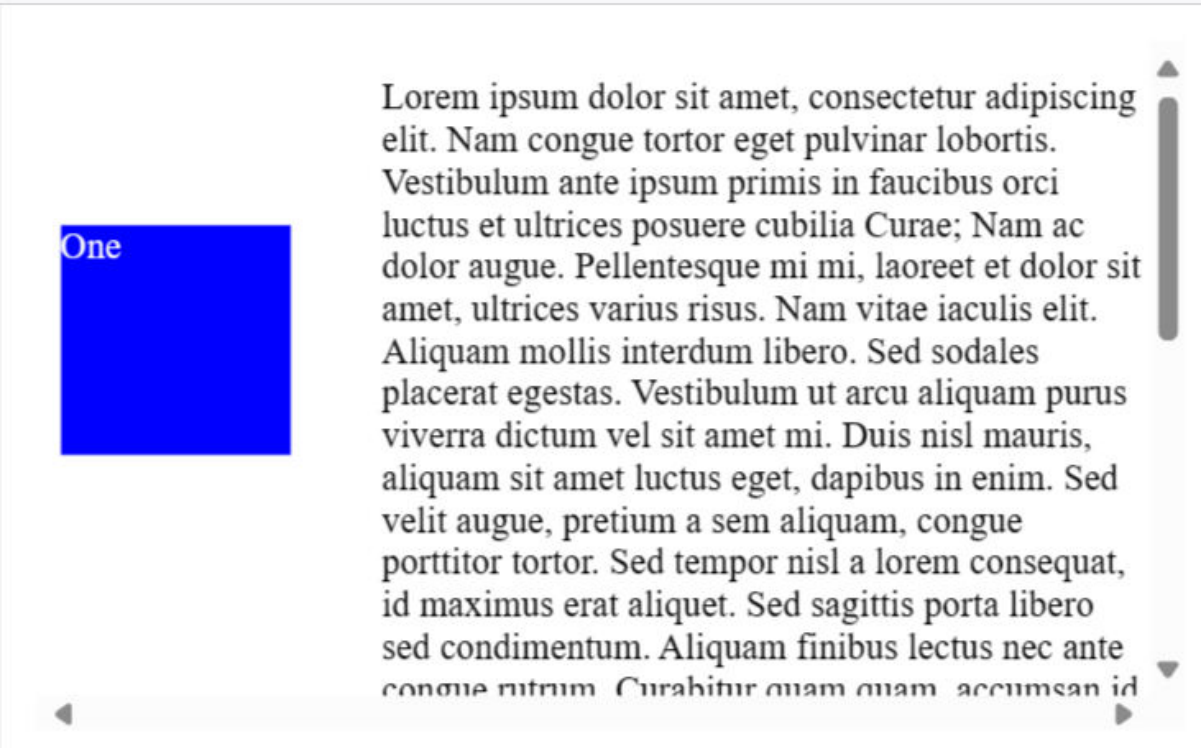
maximus erat aliquet. Sed sagittis porta libero sed condimentum. Aliquam

finibus lectus nec ante congue rutrum. Curabitur quam quam, accumsan id

ultrices ultrices, tempor et tellus.

</p>

One



5.Sticky Positioning:

- The element is positioned according to the normal flow of the document, and then offset relative to its nearest scrolling ancestor and containing block (nearest block-level ancestor), including table-related elements, based on the values of top, right, bottom, and left. The offset does not affect the position of any other elements.
- Element is positioned based on the user's scroll position.
- It acts like position: relative; until the user scrolls to a specified point, then it becomes position: fixed; until it reaches the end of its container.

Properties:

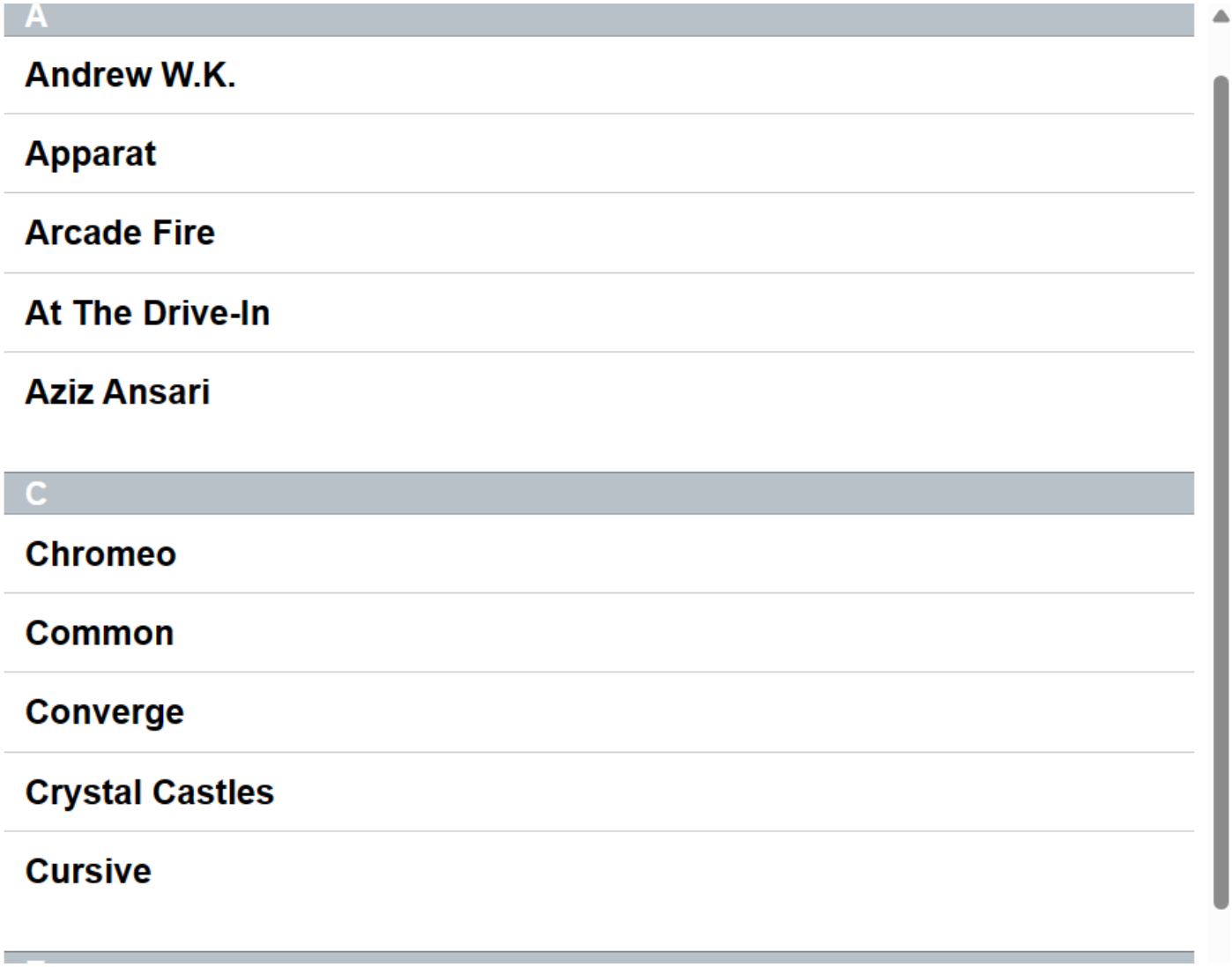
```
position: sticky;
top: value;
right: value;
bottom: value;
left: value;
```

Example:

```
<style>
  * {
    box-sizing: border-box;
  }
  dl>div {
    background: #fff;
    padding: 24px 0 0 0;
  }
  dt {
    background: #b8c1c8;
    border-bottom: 1px solid #989ea4;
    border-top: 1px solid #717d85;
    color: #fff;
    font:
      bold 18px/21px Helvetica,
      Arial,
      sans-serif;
    margin: 0;
    padding: 2px 0 0 12px;
    position: -webkit-sticky;
    position: sticky;
    top: -1px;
  }
  dd {
    font:
```

```
            bold 20px/45px Helvetica,
            Arial,
            sans-serif;
        margin: 0;
        padding: 0 0 0 12px;
        white-space: nowrap;
    }
    dd+dd {
        border-top: 1px solid #ccc;
    }
</style>
<dl>
    <div>
        <dt>A</dt>
        <dd>Andrew W.K.</dd>
        <dd>Apparat</dd>
        <dd>Arcade Fire</dd>
        <dd>At The Drive-In</dd>
        <dd>Aziz Ansari</dd>
    </div>
    <div>
        <dt>C</dt>
        <dd>Chromeo</dd>
        <dd>Common</dd>
        <dd>Converge</dd>
```

```
        <dd>Crystal Castles</dd>
        <dd>Cursive</dd>
    </div>
    <div>
        <dt>E</dt>
        <dd>Explosions In The Sky</dd>
    </div>
    <div>
        <dt>T</dt>
        <dd>Ted Leo &amp;amp; The Pharmacists</dd>
        <dd>T-Pain</dd>
        <dd>Thrice</dd>
        <dd>TV On The Radio</dd>
        <dd>Two Gallants</dd>
    </div>
</dl>
```



Z-index

The z-index property in CSS is used to control the stacking order of positioned elements on a webpage. It allows you to specify which elements should appear in front of or behind others when they overlap. The z-index property can take several values:

- auto: The stack order is equal to its parent's.

- <integer>: A positive or negative integer that specifies the stack order. Elements with a higher integer value will appear in front of elements with a lower value.

- inherit: The element inherits the z-index value from its parent.

It's important to note that z-index only works on positioned elements, which are elements with a position property set to anything other than

static (e.g., relative, absolute, fixed, or sticky). Additionally, z-index affects the stacking order of elements within the same stacking context. A stacking context is formed by elements with a z-index value other than auto, or by elements with a position property set to absolute or relative and a z-index value of auto.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Z-Index Example</title>
    <link rel="stylesheet" href="styles.css">
    <style>
      body {
    margin: 0;
    padding: 0;
    font-family: Arial, sans-serif;
}
.wrapper {
    position: relative;
    width: 100%;
    height: 200px;
    background-color: #f0f0f0;
    padding: 20px;
    box-sizing: border-box;
}
```

```
.dashed-box {
    position: relative;
    z-index: 1;
    border: 2px dashed #000;
    height: 80px;
    margin-bottom: 10px;
    margin-top: 20px;
    background-color: #fff;
}
.gold-box {
    position: absolute;
    z-index: 3; /* This will put .gold-box above .green-box and .dashed-box */
    background: gold;
    width: 80%;
    left: 60px;
    top: 30px;
    height: 70px;
}
.green-box {
    position: absolute;
    z-index: 2; /* This will put .green-box above .dashed-box */
    background: lightgreen;
    width: 20%;
    left: 65%;
```

```
    top: -25px;
    height: 70px;
    opacity: 0.9;
}
    </style>
</head>
<body>
    <div class="wrapper">
        <div class="dashed-box"></div>
        <div class="gold-box"></div>
        <div class="green-box"></div>
    </div>
</body>
</html>
```

Float

float property is used to specify whether an element should be floated to the left or right within its container, allowing other elements

1. Syntax:
float: left | right | none | inherit;

2.Values:

- **left**: The element floats to the left side of its containing block, allowing content to flow around it on the right side.

- **right**: The element floats to the right side of its containing block, allowing content to flow around it on the left side.
- **none**: The element does not float. It remains in the normal flow of the document.

3.**Example**

```
.float-left {
    float: left;
}
.float-right {
    float: right;
}
```

4. Usage:

- **Creating Float Layouts**: Historically, floats were commonly used for creating multi-column layouts before the advent of Flexbox and CSS Grid. By floating elements to the left or right, designers could create columns and make content flow around them.
- **Image Alignment**: Floats are often used to align images within text, allowing the text to wrap around the image.
- **Clearing Floats**: When floating elements, it's common to use the **clear** property to ensure that subsequent content does not wrap around floated elements unintentionally. This prevents elements from being affected by previous floats.

5.Float Clearing Techniques:

- **clear: left | right | both | none**: This property specifies whether an element can be positioned next to floating elements that are on the left side, right side, or both sides, or whether it should be moved down below any floating elements.

- Clearfix Hack: This is a popular technique used to contain floats within a parent element without having to explicitly set a height. It involves applying a clearfix class to the parent element.

```
.clearfix::after {
    content: "";
    display: table;
    clear: both;
}
```

OUTLINE

The outline property in CSS is used to draw a line around an element, outside the border, to highlight it or denote focus

Outline Property:

Syntax: outline: <outline-width> <outline-style> <outline-color>;

The outline property can have up to three values:

outline-width: Specifies the thickness of the outline. Default value is medium.

outline-style: Specifies the style of the outline, such as solid, dashed, double, etc. Default value is none.

outline-color: Specifies the color of the outline. Default value is the text color of the element.

```
.highlighted {
    outline: 2px solid red; /* Thick red solid outline */
}
.focused {
    outline: 2px dotted blue; /* Thick blue dotted outline */
}
```

You can adjust the values (width, style, color) of the outline based on your design requirements.

Box shadow

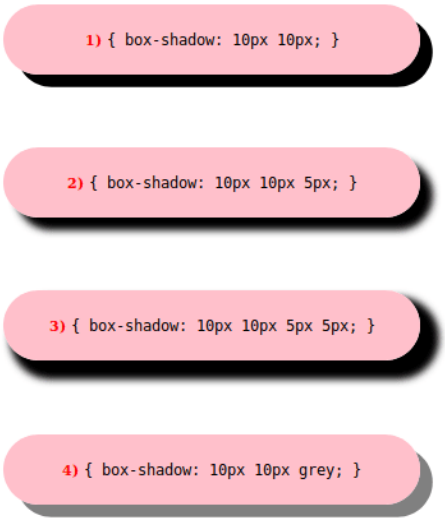
It is used to add shadow effects to the Box

Syntax

box-shadow: **[horizontal offset] [vertical offset] [blur radius] [color];**

- **Horizontal offset:** Specifies the horizontal distance of the shadow. Positive values move the shadow to the right, while negative values move it to the left.
- **Vertical offset:** Specifies the vertical distance of the shadow. Positive values move the shadow downwards, while negative values move it upwards.
- **Blur radius:** Specifies the amount of blurring applied to the shadow. Larger values create a more diffused shadow.
- **Color:** Specifies the color of the shadow.

```
.shadow {
   box-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
}
```



Absolute length units:-

px (Pixels): It's a fixed-size unit that does not change with the screen's resolution.

in (Inches), **cm (Centimeters)**, **mm (Millimeters)**: These units represent physical lengths. They are useful when you need precise control over print styles.

Relative length units:-

% (Percentage): Percentages are relative to the parent element. For example, setting width: 50% on an element means it will take up 50% of its parent's width.

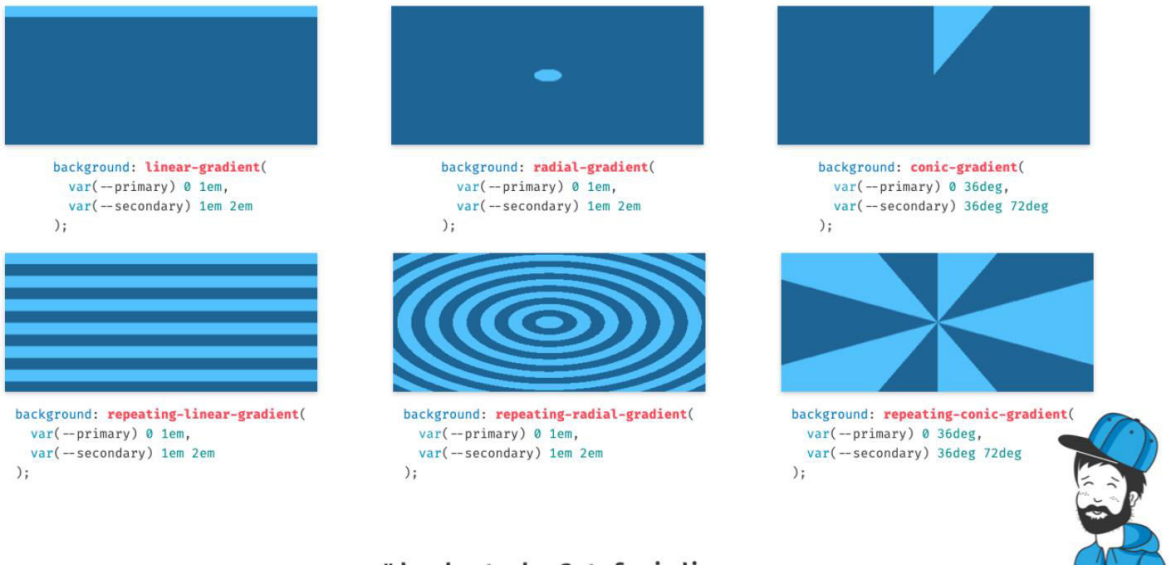
em: em unit is relative to the font-size of its parent element, if an parent element's font-size is 16px, then 1em is equal to 16px

rem: Similar to em, but it's relative to the font-size of the root element (usually the <html> element).

vw (Viewport Width), **vh (Viewport Height)**: These units are relative to the size of the browser viewport. For example, 1vw is equal to 1% of the viewport width

Gradients

gradients allow you to create smooth transitions between two or more specified colors. They are defined using the linear-gradient() , radial-gradient() and conic-gradient.



Linear Gradients:

- Defined using the **linear-gradient()** function.
- Creates a gradient that transitions linearly along a specified angle.
- Syntax: **linear-gradient(direction, color1, color2, ...)**.

```
.element {
    background: linear-gradient(to right, red, blue);
}
```

Radial Gradients:

- Defined using the **radial-gradient()** function.
- Creates a gradient that transitions radially from the center to the outer edges.
- Syntax: **radial-gradient(shape size at position, color1, color2, ...)**.

```
.element {
    background: radial-gradient(circle, red, blue);
}
```

background-image: radial-gradient(closest-side at 60% 55%, red, yellow, black);

Conic Gradients:

- Defined using the **conic-gradient()** function.
- Creates a gradient that create a color transition that starts from a specified starting angle and moves around in a circular manner, ending at a specified ending angle.
- Syntax: **conic-gradient(from angle,color1, color2, color3)**.

```
.element {
  background: conic-gradient(from 45deg, red, yellow, lime, blue);
}
```

Repeating Gradients:

- You can create repeating gradients by using the **repeating-linear-gradient()** or **repeating-radial-gradient()** or **repeating-conic -gradient()** functions.

```
.element {
  background: repeating-linear-gradient(to right, red, blue 50px);
}
```

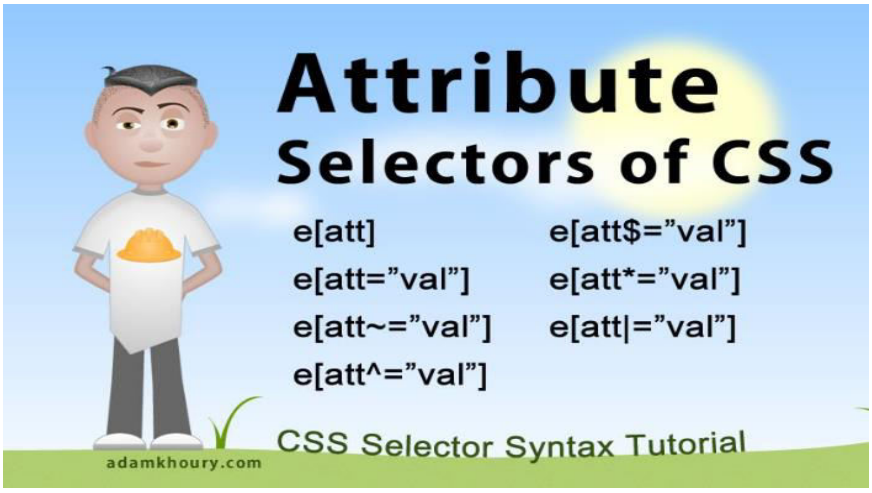
Attribute selectors

Attribute selectors are a powerful way to select HTML elements based on their attributes and attribute values. They allow for more specific and flexible styling compared to class or ID selectors.

Basic Syntax

element[attribute] { /* styles */ }

This selects all elements of the specified type that have the given attribute, regardless of its value.



Types of Attribute Selectors

1. **[attribute]**: Selects elements with the specified attribute.

[target] { background-color: lightgray; }

This selects all elements that have the **target** attribute.

2. **[attribute=value]**: Selects elements with the specified attribute and exact value.

input[type="text"] { border: 1px solid black; }

This selects all **<input>** elements where **type** is exactly **text**.

3. **[attribute~=value]**: Selects elements with the specified attribute whose value is a whitespace-separated list of words, one of which is exactly the specified value.

[class~="button"] { padding: 10px; }

This selects elements that have a **class** attribute containing the word "button" (e.g., **class="button primary"**).

4. **[attribute^=value]**: Selects elements with the specified attribute whose value starts with the specified value.

a[href^="https"] { color: green; }

This selects **<a>** elements where the **href** attribute value starts with "https".

5. **[attribute$=value]**: Selects elements with the specified attribute whose value ends with the specified value.

img[src$=".jpg"] { border: 2px solid black; }

This selects **<img>** elements where the **src** attribute value ends with ".jpg".

6. **[attribute|=value]**: Selects elements with the specified attribute whose value is either exactly the specified value or starts with the specified value followed by a hyphen (usually used for language codes).

[lang|="en"] { color: blue; }

This selects elements with a **lang** attribute value that is exactly "en" or starts with "en-" (e.g., **lang="en-US"**).

7. **[attribute*="value"]**

The [attribute*="value"] selector is used to select elements whose attribute value contains a specified value.

The following example selects all elements with a class attribute value that contains "rent":

```
[class*="rent"] {
  background: yellow;
}
```

Pseudo-classes and pseudo-elements in CSS

Pseudo-classes and pseudo-elements in CSS are powerful tools that allow you to style elements based on their state or to style specific parts of an element.

Pseudo-classes

Pseudo-classes are used to define a special state of an element. They can be combined with a CSS selector to apply styles based on the element's state. For example, you can change the style of an element when the user hovers over it, or when a link is visited.

:hover: This pseudo-class is used to style an element when the user's pointer hovers over it. For example, changing the background color of a element on hover:

```
div:hover {
background-color: blue;
}
```

:active: This pseudo-class is used to style an element when it is activated, such as when a user clicks on it. For example, changing the background color of a element when it is clicked:

```
.box:active {
background-color: orange;
}
```

:visited: This pseudo-class is used to style links that have been visited by the user.

:focus: This pseudo-class is used to style an element when it has focus, such as when a user tabs to an input field.

ALL ELEMENTS OF PSEUDO-CLASS

Selector	Example	Example description
:active	a:active	Selects the active link
:checked	input:checked	Selects every checked <input> element
:disabled	input:disabled	Selects every disabled <input> element
:empty	p:empty	Selects every <p> element that has no children
:enabled	input:enabled	Selects every enabled <input> element
:first-child	p:first-child	Selects every <p> elements that is the first chil parent
:first-of-type	p:first-of-type	Selects every <p> element that is the first <p> its parent
:focus	input:focus	Selects the <input> element that has focus
:hover	a:hover	Selects links on mouse over
:in-range	input:in-range	Selects <input> elements with a value within a range
:invalid	input:invalid	Selects all <input> elements with an invalid valu

:lang(*language*)	p:lang(it)	Selects every <p> element with a lang attribute starting with "it"
:last-child	p:last-child	Selects every <p> elements that is the last child parent
:last-of-type	p:last-of-type	Selects every <p> element that is the last <p> its parent
:link	a:link	Selects all unvisited links
:not(selector)	:not(p)	Selects every element that is not a <p> elemen
:nth-child(n)	p:nth-child(2)	Selects every <p> element that is the second ch parent
:nth-last-child(n)	p:nth-last-child(2)	Selects every <p> element that is the second ch parent, counting from the last child
:nth-last-of-type(n)	p:nth-last-of-type(2)	Selects every <p> element that is the second < of its parent, counting from the last child
:nth-of-type(n)	p:nth-of-type(2)	Selects every <p> element that is the second < of its parent
:only-of-type	p:only-of-type	Selects every <p> element that is the only <p> its parent

:only-child	p:only-child	Selects every <p> element that is the only child parent
:optional	input:optional	Selects <input> elements with no "required" att
:out-of-range	input:out-of-range	Selects <input> elements with a value outside a range
:read-only	input:read-only	Selects <input> elements with a "readonly" attr specified
:read-write	input:read-write	Selects <input> elements with no "readonly" att
:required	input:required	Selects <input> elements with a "required" attri specified
:root	root	Selects the document's root element
:target	#news:target	Selects the current active #news element (clicke containing that anchor name)
:valid	input:valid	Selects all <input> elements with a valid value
:visited	a:visited	Selects all visited links

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>Pseudo-classes Example</title>
    <style>
        /* Styling for links */
        a:link {
            color: blue;
        }
        a:visited {
            color: purple;
        }
        a:hover {
            color: red;
        }
        a:active {
            color: yellow;
        }

        /* Styling for a div on hover */
        div:hover {
            background-color: blue;
        }
        /* Styling for a button on hover */
        button:hover {
```

```
            color: blue;
        }
        /* Styling for an input field on focus */
        input:focus {
            border: 2px solid blue;
            outline: none;
        }
        /* Styling for a div with a specific class on hover */
        .highlight:hover {
            color: #ff0000;
        }
        /* Styling for a div with a specific class on active */
        .highlight:active {
            background-color: yellow;
        }
        /* Styling for a div with a specific class on focus */
        .highlight:focus {
            border: 2px solid blue;
        }
    </style>
</head>
<body>
    <h1>Pseudo-classes Example</h1>
```

```
    <!-- Links -->
    <p>
        <a href="#">Link 1</a>
        <a href="#">Link 2</a>
    </p>
    <!-- Div with hover effect -->
    <div>Hover over me!</div>
    <!-- Button with hover effect -->
    <button>Hover over me!</button>
    <!-- Input field with focus effect -->
    <input type="text" placeholder="Focus on me!">
    <!-- Div with specific class and hover effect -->
    <div class="highlight">Hover over me!</div>
    <!-- Div with specific class and active effect -->
    <div class="highlight">Click me!</div>
    <!-- Div with specific class and focus effect -->
    <div class="highlight">Focus on me!</div>
</body>
</html>
```

Pseudo-elements

Pseudo-elements are used to style specific parts of an element. They can be used to insert content before or after the content of an element, or to style the first letter or line of an element.

::before and ::after: These pseudo-elements allow you to insert content before or after the content of an element. For example, adding a decorative element before the content of a :

```
div::before {
content: ">";
color: red;
}
```

::first-letter and ::first-line: These pseudo-elements allow you to style the first letter or line of an element. For example, changing the font size of the first letter of a paragraph:

```
p::first-letter {
font-size: 2em;
}
```

Pseudo-classes and pseudo-elements provide a way to apply styles based on the state of an element or to style specific parts of an element without altering the HTML structure. They are essential tools for creating dynamic and interactive web designs

PSEUDO ELEMENTS

Selector	Example	Example description
::after	p::after	Insert content after every <p> element
::before	p::before	Insert content before every <p> element

::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
::marker	::marker	Selects the markers of list items
::selection	p::selection	Selects the portion of an element that is selected b

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Pseudo-elements Example</title>
<style>
    /* Style the first letter of a paragraph */
    p::first-letter {
        font-size: 24px;
        color: red;
    }
    /* Style the first line of a paragraph */
    p::first-line {
```

```
        font-weight: bold;
    }
    /* Style the content before an element */
    p::before {
        content: "Before: ";
        color: blue;
    }
    /* Style the content after an element */
    p::after {
        content: " (After)";
        color: green;
    }
    /* Style the placeholder text of an input */
    input::placeholder {
        color: gray;
        font-style: italic;
    }
</style>
</head>
<body>
    <p>This is a paragraph.</p>
    <input type="text" placeholder="Enter text here">
</body>
```

</html>

CSS Flexbox

CSS Flexbox is a one-dimensional layout model that allows you to design complex web layouts more efficiently and intuitively. It provides a flexible way to distribute space among elements in a container and align them in various ways.

Parent properties

1. **Flex Container (display: flex):**
    - To enable Flexbox layout for a container, set its display property to flex.
    - The children of a flex container become flex items.

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

2. **Flex Direction (flex-direction):**
    - Determines the main axis direction along which flex items are laid out.
    - Values: row (default), row-reverse, column, column-reverse.

```
.container {
    flex-direction: row;        /* Default: horizontal */
    flex-direction: row-reverse; /* Horizontal reversed */
    flex-direction: column;     /* Vertical */
    flex-direction: column-reverse; /* Vertical reversed */
}
```

3. **Justify Content (justify-content):**
    - Aligns flex items along the main(horizontal) axis of the flex container.

- Values: flex-start, flex-end, center, space-between, space-around, space-evenly.

```
.container {
    justify-content: flex-start; /* Default: aligned at the start */
    justify-content: flex-end;   /* Aligned at the end */
    justify-content: center;     /* Centered */
    justify-content: space-between; /* Evenly distributed with space between */
    justify-content: space-around; /* Evenly distributed with space around */
    justify-content: space-evenly; /* Evenly distributed with equal space around them */
}
```

4. **Align Items (align-items):**
   - Aligns flex items along the cross axis of the flex container.
   - Values: stretch (default), flex-start, flex-end, center, baseline.

```
.container {
    align-items: stretch;       /* Default: stretch to fill container */
    align-items: flex-start;    /* Align items at the start */
    align-items: flex-end;      /* Align items at the end */
    align-items: center;        /* Center items */
    align-items: baseline;      /* Align items along the baseline */
}
```

5. **Align Content (align-content):**
   - Aligns multiple lines of flex items when there's extra space in the cross axis.
   - Only applicable when there are multiple lines of flex items.

- Values: flex-start, flex-end, center, space-between, space-around, space-evenly.

```
.container {
    align-content: stretch;     /* Default: lines stretch to fill space */
    align-content: flex-start;  /* Lines packed to the start */
    align-content: flex-end;    /* Lines packed to the end */
    align-content: center;      /* Lines centered */
    align-content: space-between; /* Lines evenly distributed with space between them */
    align-content: space-around; /* Lines evenly distributed with space around them */
    align-content: space-evenly; /* Lines evenly distributed with equal space around them */
}
```

6. Flex wrap (flex-wrap):

- property specifies whether the flex items should wrap or not.
- Values: wrap or no-wrap or wrap-reverse.

```
.container {
    flex-wrap: nowrap;          /* Default: no wrap */
    flex-wrap: wrap;            /* Wrap items */
    flex-wrap: wrap-reverse;    /* Wrap items in reverse order */
}
```

7. row-gap | column-gap | gap

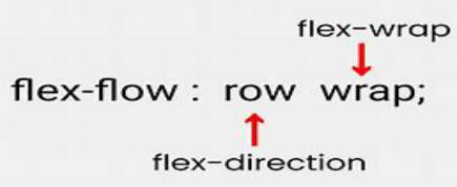
- row-gap creates the space between the rows
- column-gap creates the space between the columns

- gap is a shorthand for setting both **column-gap** and **row-gap** simultaneously

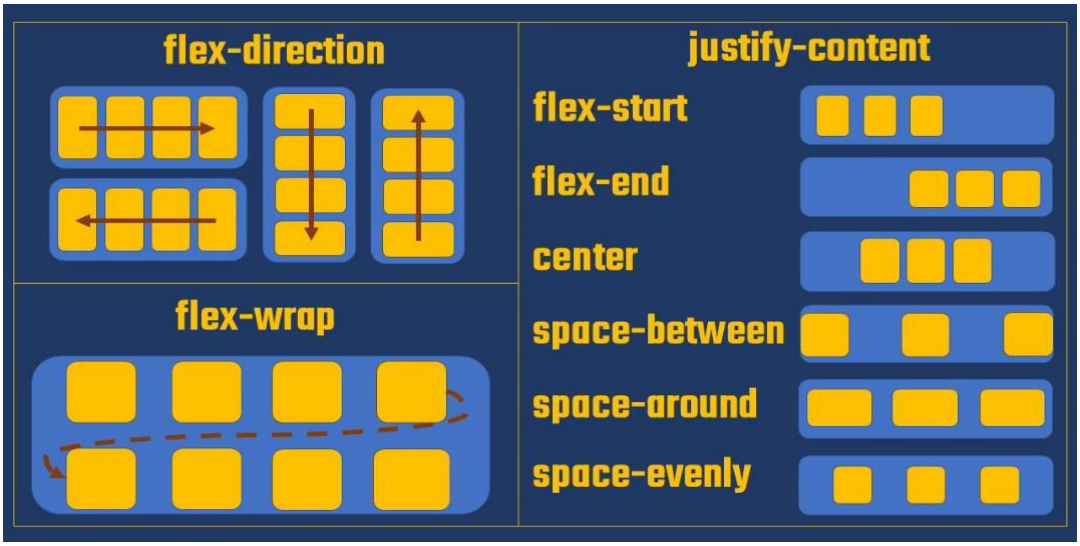
```
.container {
    gap: 20px;          /* creates a gap of 20px on both rows and columns*/
    row-gap:10px;          /*creates a gap of 10px between the rows */
    column-gap:10px;    /* creates a gap of 10px between the column */
    }
```

8. flex-flow

A shorthand for flex-direction and flex-wrap.




```
.container {
    flex-flow: row wrap;       /* Combines direction and wrapping */
}
```



Child properties

1. **Flex Basis (flex-basis):**
   - Specifies the initial size of a flex item before free space is distributed.
   - Default value is auto, which means the item's size is based on its content size.

   ```
   .item {
       flex-basis: 200px; /* Default is auto */
   }
   ```

2. **Flex Grow (flex-grow):**
   - Defines how much a flex item can grow relative to other flex items.
   - A value of 0 means the item won't grow.
   - A higher value means it will grow more compared to other flex items.

   ```
   .item {
       flex-grow: 1; /* Default is 0, no growth */
   }
   ```

3. **Flex Shrink (flex-shrink):**
   - Defines how much a flex item can shrink relative to other flex items.
   - A value of 0 means the item won't shrink.
   - A higher value means it will shrink more compared to other flex items.

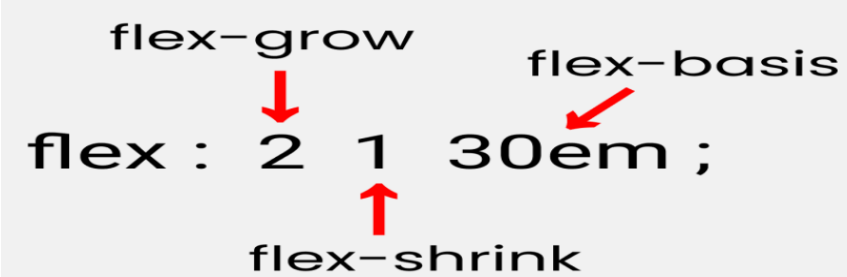
   ```
   .item {
       flex-shrink: 1; /* Default is 1, items shrink */
   ```

}

4. **Flex (flex shorthand):**

- Combines flex-grow, flex-shrink, and flex-basis properties into a single shorthand property.

```
.item {
    flex: 1 1 200px; /* flex-grow: 1; flex-shrink: 1; flex-basis: 200px; */
}
```



5. **Order (order):**

- Specifies the order in which a flex item appears in the flex container.
- By default, items have an order of 0.
- Lower values come first, and higher values come later.

```
.item {
    order: 1;   /* Default is 0 */
}
```

Grid layout

A grid layout is a multi-dimensional design approach where content is organized into rows and columns

The **display: grid;** property in CSS is used to create grid layouts. It turns an HTML element into a grid container, allowing you to easily organize its child elements into rows and columns.

```
<style>
    .container {
      display: grid;
      grid-template-columns: 100px 100px 100px; /* Three columns each 100px wide */
      grid-template-rows: 100px 100px; /* Two rows each 100px tall */
      gap: 10px; /* Gap between grid items */
    }
    .item {
      background-color: #f0f0f0;
      padding: 20px;
      text-align: center;
    }
  </style>
<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 class="item">7</div>
    <div class="item">8</div>
    <div class="item">9</div>
  </div>
```

- We create a grid container by setting **display: grid;**.
- We define three columns, each 100px wide, and two rows, each 100px tall, using **grid-template-columns** and **grid-template-rows**.
- The **gap** property adds a 10px gap between grid items.

Parent properties

1. **display:grid**

*This is the most fundamental property to create a grid layout. Setting **display: grid;** on an element turns it into a grid container.

2. **grid-template-columns || grid-template-rows**

* grid-template-columns defines the number and size of columns in the grid. You can specify the width of each column individually.

* Similar to grid-template-columns, but for defining the rows in the grid

3. **grid-row-gap | grid-column-gap | gap**

*grid-row-gap creates the space between the rows

*grid-column -gap creates the space between the columns

* gap is a shorthand for setting both **grid-column-gap** and **grid-row-gap** simultaneously.

4. **Justify Content (justify-content):**

- Aligns the grid along the row axis (horizontal alignment) within the grid container
- Values: start, end, center, space-between, space-around, space-evenly.

5. **Align Items (align-items):**
    - Aligns grid items along the column axis (vertical alignment) within the grid cells
    - Values: start, end, center, space-between, space-around, space-evenly.
6. **Align Content (align-content):**
    - Aligns the grid along the column axis (vertical alignment) within the grid container

Child Properties

1. **grid-column-start** and **grid-column-end**: Specifies the starting and ending grid lines for the grid item along the column axis. You can also use the shorthand property **grid-column** to set both start and end values.

   .item1{

   grid-column-start:2;

   grid-column-end:5;

   }

2. **grid-row-start** and **grid-row-end**: Specifies the starting and ending grid lines for the grid item along the row axis. Similarly, you can use the shorthand property **grid-row** to set both start and end values.

```
.item1{
grid-row-start:2;
grid-row-end:5;
}
```

3. **grid-column**: A shorthand for **grid-column-start** and **grid-column-end**, allowing you to specify both values in a single property.

```
.item1{
grid-column:2/5;
}
```

4. **grid-row**: A shorthand for **grid-row-start** and **grid-row-end**, allowing you to specify both values in a single property.

```
.item1{
grid-row:2/5;
}
```

How to use grid template areas in media queries for effective layout

Grid-template-areas:

The CSS grid-template-areas property is a powerful feature of CSS Grid Layout, allowing you to define areas within a grid container. These areas can be used to assign grid items to specific sections of the grid, making it easier to design complex layouts with a readable and maintainable structure.

Syntax:

```
grid-template-areas:
  "header header header"
  "sidebar main main"
  "footer footer footer";
```

Each string represents a row in the grid, and each space-separated value within the string represents a grid area.

1. **Naming Areas:**
   - Areas are named using any valid CSS identifier.
   - Each area name should be consistent across the grid.
   - Use a **.** (period) to represent an empty cell in the grid.

2. **Defining Grid Items:**
   - Use the grid-area property to assign a grid item to a named area.

```
.header {
  grid-area: header;
}
.sidebar {
  grid-area: sidebar;
}
.main {
  grid-area: main;
}
.footer {
  grid-area: footer;
}
```

3. **Responsive Design**: Change grid areas with media queries.

```
@media (max-width: 600px) {
  .container {
    grid-template-areas:
      "header"
      "main"
      "sidebar"
      "footer";
  }
}
```

Responsiveness of websites

Responsiveness in web design refers to the ability of a website to adapt and display appropriately across various devices and screen sizes. A responsive website adjusts its layout, content, and design elements dynamically to ensure an optimal viewing experience for users on desktops, laptops, tablets, and smartphones.

Breakpoints

Breakpoints in responsive design refer to specific points at which the layout of a website changes to accommodate different screen sizes or device types.

common breakpoints used in responsive design:

Extra Small (XS): Typically for devices with a width of up to **576 pixels,** such as small smartphones in portrait orientation.

Small (SM): Devices with a width between **576 pixels and 768 pixels**, often including larger smartphones and small tablets in portrait orientation.

Medium (MD): Devices with a width between **768 pixels and 992 pixels**, including most tablets in landscape orientation.

Large (LG): Devices with a width between **992 pixels and 1200 pixels**, covering larger tablets and small laptops.

Extra Large (XL): Devices with a width of **1200 pixels** or higher, including desktop monitors and larger laptops.

Media queries

Media queries in CSS are a powerful tool for applying styles based on the conditions of the device rendering the content. They are essential for creating responsive web designs that adapt to various screen sizes, resolutions, and other device characteristics.

Media queries are a way for websites to adapt their layout and design based on different conditions, such as screen size, device orientation, or even the type of device being used.

Media Query Syntax:

```
@media screen and (max-width: 600px) {
  /* CSS rules for screens with a maximum width of 600px */
}
```

Explanation:

@media: This tells the browser that we're about to specify different styles based on certain conditions.

screen: This specifies that we're targeting screens (like computer monitors or smartphones) rather than other types of media like print.

(max-width: 600px): This is the condition we're checking. In this example, it means "if the screen width is 600 pixels or less."

The rules inside the curly braces {} will apply only if the condition inside the parentheses is met.

Example

```
/* Styles for desktop */
.sidebar {
  width: 30%; /* Sidebar takes up 30% of the screen width */
}

/* Media query for smaller screens */
@media screen and (max-width: 600px) {
  .sidebar {
    width: 100%; /* Sidebar takes up 100% of the screen width */
  }
```

```
}
```

Explanation:

Initially, the sidebar takes up 30% of the screen width, which is suitable for larger screens.

But when the screen width becomes 600 pixels or less (like on smartphones), the sidebar switches to taking up 100% of the screen width, making it easier to navigate on smaller devices.

Media queries adjustments using inline-block

```
<style>
    .inlineitem {
        display: inline-block;
        width: 23%;
        background-color: aquamarine;
        height: 20vh;
        border: 2px solid red;
    }

    @media screen and (max-width: 768px) {
        .inlineitem {
            display: block;
            width: 100%;
            padding: 30px;
            height: 10vh;
        }
    }
```

```
    </style>

<body>
    <div class="inlineitem">Item 1</div>
    <div class="inlineitem">Item 2</div>
    <div class="inlineitem">Item 3</div>
    <div class="inlineitem">Item 4</div>
</body>
```

Media queries adjustment flex

```
    <style>
        .container {
            display: flex;
            justify-content: space-evenly;
        }

        .child {
            background-color: aqua;
            padding: 20px;
            border: 2px solid red;
            width: 100%;
        }

        @media screen and (max-width: 800px) {
```

```
        .container {
            flex-direction: column;
        }
    }
</style>
<body>
    <div class="container">
        <div class="child">Child 1</div>
        <div class="child">Child 2</div>
        <div class="child">Child 3</div>
    </div>
</body>
```

Media queries adjustment using grid

```
    <style>
        .container {
            display: grid;
            grid-template-columns: auto auto auto;
            grid-template-rows: 100px 100px 100px 100px 100px 100px;
        }

        .child {
            background-color: aqua;
```

```
            padding: 20px;
            border: 2px solid red;
            width: 100%;
            font-size: 100px;
        }

        @media screen and (min-width: 800px) and (max-width: 1199px)
{
            .container {
                grid-template-columns: auto auto;
            }
        }

        @media screen and (max-width: 799px) {
            .container {
                grid-template-columns: auto;
            }

            .child {
                font-size: 10px;
            }
        }
    </style>
<body>
    <div class="container">
```

```
        <div class="child">Child 1</div>
        <div class="child">Child 2</div>
        <div class="child">Child 3</div>
        <div class="child">Child 4</div>
        <div class="child">Child 5</div>
        <div class="child">Child 6</div>
    </div>
</body>
```

Media queries adjustment using grid template areas

HTML Structure: Uses the grid template areas defined in the CSS to structure the content.

```
        <div class="container">
            <div class="header">header</div>
            <div class="nav">navbar</div>
            <div class="section">section 1</div>
            <div class="article">article 1</div>
            <div class="aside">aside</div>
            <div class="footer">footer</div>
        </div>
```

Grid Areas: Assigns grid areas to different sections of the layout.

```
        .header { grid-area: header; }
        .nav { grid-area: nav; }
        .article { grid-area: article; }
        .section { grid-area: section; }
```

```
.aside { grid-area: aside; }
.footer { grid-area: footer; }
```

Container Grid Layout: Defines a grid layout for larger screens with specified grid areas.

```
.container {
    display: grid;
    grid-template-areas:
    "header header header"
    "nav nav nav"
    "section section aside"
    "article article aside"
    "footer footer footer";
}
```

Media Query for Small Screens: Adjusts the grid layout for screens smaller than 800px

```
@media screen and (width < 800px) {
    .container {
        grid-template-areas:
        "header"
        "nav"
        "section"
        "article"
        "aside"
        "footer";
        width: 100%;  /* Make container full width for small screens
*/
```

```
    }
}
```

2d,3d Transforms

CSS transforms allow you to manipulate the appearance of HTML elements in various ways, including translating (moving), scaling (resizing), rotating, and skewing elements. These transformations can be applied in both 2D and 3D space, offering a wide range of visual effects to enhance web designs.

2D Transforms

2D transforms include functions like translate(), scale(), rotate(), and skew(). These functions allow you to move, resize, rotate, and skew elements along the X and Y axes.

Translate: Moves an element along the X and Y axes.

```
.element {
  transform: translateX(100px) translateY(50px);
}
```

Scale: Resizes an element.

```
.element {
  transform: scale(1.5);
}
```

Rotate: Rotates an element clockwise or counterclockwise.

```
.element {
  transform: rotate(45deg);
}
```

Skew: Skews an element along the X and Y axes.

```
.element {
  transform: skew(30deg, 20deg);
}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Transform Example</title>
<style>
    .container {
        display: flex;
        justify-content: space-around;
        margin-top: 50px;
    }

    .translate-div {
        width: 100px;
        height: 100px;
        background-color: lightblue;
        margin: 10px;
        transition: transform 0.3s ease;
    }
```

```
.translate-div:hover {
    transform: translateX(50px);
}

.scale-div {
    width: 100px;
    height: 100px;
    background-color: lightgreen;
    margin: 10px;
    transition: transform 0.3s ease;
}

.scale-div:hover {
    transform: scale(1.5);
}

.rotate-div {
    width: 100px;
    height: 100px;
    background-color: lightcoral;
    margin: 10px;
    transition: transform 0.3s ease;
```

```
    }

    .rotate-div:hover {
        transform: rotate(45deg);
    }

    .skew-div {
        width: 100px;
        height: 100px;
        background-color: lightsalmon;
        margin: 10px;
        transition: transform 0.3s ease;
    }

    .skew-div:hover {
        transform: skewX(30deg);
    }
</style>
</head>
<body>
    <div class="container">
        <div class="translate-div">Translate</div>
        <div class="scale-div">Scale</div>
```

```
            <div class="rotate-div">Rotate</div>
            <div class="skew-div">Skew</div>
        </div>
    </body>
    </html>
```

3D Transforms

3D transforms introduce depth to the transformations, allowing for more complex effects. Functions like translate3d(), scale3d(), rotate3d(), and perspective() are used for 3D transformations.

Translate3d: Moves an element in 3D space.

```
.element {
  transform: translate3d(100px, 50px, 200px);
}
```

Scale3d: Resizes an element in 3D space.

```
.element {
  transform: scale3d(1.5, 1.5, 1.5);
}
```

Rotate3d: Rotates an element in 3D space.

```
.element {
  transform: rotate3d(1, 1, 1, 45deg);
}
```

Perspective: Defines the perspective from which the 3D elements are viewed.

```
.element {
```

```
  transform: perspective(500px) rotateX(45deg);
}
```

To enable 3D transformations, you might need to use the transform-style property and set it to preserve-3d. This ensures that child elements also adopt the 3D transformations.

CSS transforms are a powerful tool for creating visually engaging and interactive web designs

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>3D Transform Example</title>
<style>
    .container {
        display: flex;
        justify-content: space-around;
        margin-top: 50px;
    }
    .rotate-x-div {
        width: 100px;
        height: 100px;
        background-color: lightblue;
```

```
    margin: 10px;
    transition: transform 0.3s ease;
}

.rotate-x-div:hover {
    transform: rotateX(180deg);
}

.rotate-y-div {
    width: 100px;
    height: 100px;
    background-color: lightgreen;
    margin: 10px;
    transition: transform 0.3s ease;
}

.rotate-y-div:hover {
    transform: rotateY(180deg);
}

.rotate-3d-div {
    width: 100px;
    height: 100px;
```

```
        background-color: lightcoral;
        margin: 10px;
        transition: transform 0.3s ease;
    }

    .rotate-3d-div:hover {
        transform: rotate3d(1, 1, 1, 180deg);
    }
</style>
</head>
<body>
    <div class="container">
        <div class="rotate-x-div">Rotate X</div>
        <div class="rotate-y-div">Rotate Y</div>
        <div class="rotate-3d-div">Rotate 3D</div>
    </div>
</body>
</html>
```

Transition

transition property in CSS allows you to control the transition of CSS properties over a specified duration. It consists of several sub-properties that define various aspects of the transition effect:

1. **Property**: This specifies the CSS property or properties to which the transition effect should be applied.

   example: **transition-property: width**

2. **Duration**: This specifies the duration over which the transition should occur. It can be specified in seconds (**s**) or milliseconds (**ms**).

   example: **transition-duration: 5s;**

3. **Timing Function**: This property defines the speed curve of the transition, determining how intermediate values are calculated between the start and end points of the transition. There are various timing functions available, such as **ease**, **linear**, **ease-in**, **ease-out**, **ease-in-out**, and custom cubic bezier functions.

   example: **transition-function: ease-out;**

4. **Delay**: This property specifies a delay before the transition effect starts. It can also be specified in seconds (**s**) or milliseconds (**ms**).

   example: **transition-delay:2s;**.

   Trasition on hover

```
.el {
  background-color: #3498db;
  transition: background-color 2s ease;
}
```

```
.el:hover {
  background-color: #2980b9;
}
```

Slide down menu

```
.menu {
  height: 0;
  overflow: hidden;
  transition: height 0.3s ease;
}
.menu:hover {
  height: 100px; /* Or whatever height you want */
}
```

Transition shorthand property:

shorthand property **transition** in CSS allows you to specify all the transition properties (property, duration, timing function, and delay) in one declaration

transition: property duration timing-function delay;

Animations

CSS animations allow you to create more complex and dynamic animations than transitions. They enable you to specify keyframes that define the style changes at various points during the animation's duration.

Keyframes – it defines the stages of the animation

```
@keyframes animationName {
  0% { /* style at start */ }
  50% { /* style at midpoint */ }
  100% { /* style at end */ }
}
```

Or

```
@keyframes name{
from {  /* style at start */ }
 to{ /* style at end */ }
}
```

```
selector {
  animation: animationName duration timing-function delay iteration-count direction fill-mode play-state;
}
```

1) animationName: Name of the animation defined by **@keyframes**.

animation-name: slidein;

2) duration: Duration of the animation.

animation-duration: 2s;

3) delay (optional): Delay before the animation starts.

animation-delay: 1s;

4) Iteration-count (optional): Specifies the number of times the animation should repeat.

Values: infinite, a number

animation-iteration-count: infinite;

5) animation-timing-function: Specifies the speed curve of the animation (e.g., ease, linear, ease-in, ease-out, ease-in-out).

- ease: Starts slow, speeds up, then slows down (cubic-bezier(0.25, 0.1, 0.25, 1.0)).
- linear: Constant speed from start to finish (cubic-bezier(0.0, 0.0, 1.0, 1.0)).
- ease-in: Starts slow, then accelerates (cubic-bezier(0.42, 0.0, 1.0, 1.0)).
- ease-out: Starts fast, then decelerates (cubic-bezier(0.0, 0.0, 0.58, 1.0)).
- ease-in-out: Starts slow, speeds up, then slows down (cubic-bezier(0.42, 0.0, 0.58, 1.0)).
- cubic-bezier(n, n, n, n): Custom cubic-bezier curve defined by four control points

animation-timing-function: ease-in-out;

6) direction (optional): Specifies whether the animation should play in reverse or alternate directions.

Values: normal, reverse, alternate, alternate-reverse

animation-direction: alternate;

normal

- The animation plays forward from the beginning to the end of the keyframes.
- For example, if the keyframes define an animation that moves an element from left to right, it will follow that direction.

reverse

- The animation plays backward from the end to the beginning of the keyframes.
- If the keyframes define an animation that moves an element from left to right, using reverse will make the element move from right to left instead

alternate

- The animation alternates direction with each iteration. It plays forward on odd-numbered cycles (1, 3, 5, etc.) and backward on even-numbered cycles (2, 4, 6, etc.).
- This creates a "ping-pong" effect where the animation reverses direction each time it completes.

alternate-reverse

- The animation alternates direction with each iteration, but it starts by playing backward on the first cycle (1, 3, 5, etc.) and forward on the second cycle (2, 4, 6, etc.).
- This also creates a "ping-pong" effect, but in the opposite initial direction compared to alternate.

7) fill-mode (optional): Specifies how styles are applied before and after the animation.

Values: none, forwards, backwards, both

- **None**: The element will not retain any styles from the keyframes before the animation starts or after it ends. The element will return to its original state
- **Forwards**: after the animation ends, the element retains the styles defined in the last keyframe.
- **Backwards**: The element retainsthe styles defined in the first keyframe before the animation starts.
- **Both**: The element retains the styles from the first keyframe before the animation starts and the styles from the last keyframe after the animation ends.

animation-fill-mode: forwards;

Example

```
@keyframes move {
  0% { transform: translateX(0); } /* At the start, no translation */
  100% { transform: translateX(200px); } /* At the end, move 200px to the right */
}

.moving-element {
  animation: move 2s ease-in-out infinite; /* Apply the move animation */
}
```

Animation shorthand property

animation: name duration timing-function delay iteration-count direction fill-mode;