FREE HTML/CSS BASIC CLASS

Web Basic Concept

World Wide Web (WWW): The World Wide Web is a system of interlinked hypertext documents accessed via the internet. It allows users to navigate between web pages using hyperlinks.

Website: A collection of related web pages that are typically identified with a common domain name and published on at least one web server.

Web Page: A single document on the web, usually written in HTML (Hypertext Markup Language), containing text, multimedia elements, and hyperlinks.

Web Browser: A software application that retrieves and displays web pages. Examples include Chrome, Firefox, Safari, and Edge.

Web Server: A web server is a software application or a hardware device that processes incoming requests and serves web pages to users over the internet. It handles the communication between the client's web browser and the website's backend, delivering the requested web pages or resources. Common web server software includes Apache, Nginx, Microsoft Internet Information Services (IIS), and others

Web Hosting: Web hosting refers to the service of providing storage space, resources, and infrastructure to make a website accessible on the internet. It involves storing the website's files, databases, and other necessary components on servers that are connected to the internet. A web hosting provider offers different hosting plans, allowing individuals or organizations to rent space on their servers. Web hosting includes various types, such as shared hosting, virtual private server (VPS) hosting, dedicated hosting, and cloud hosting.

HTTP (Hypertext Transfer Protocol): The foundation of any data exchange on the Web. It is a protocol used for transmitting hypertext over the internet.

URL (Uniform Resource Locator): A web address that specifies the location of a resource on the internet. It typically includes the protocol (e.g., http or https), domain name, and the specific path to the resource.
[http://example.com/news/story1.html]

HTML (Hypertext Markup Language): The standard markup language used to create web pages. It defines the structure and layout of a web document using tags.

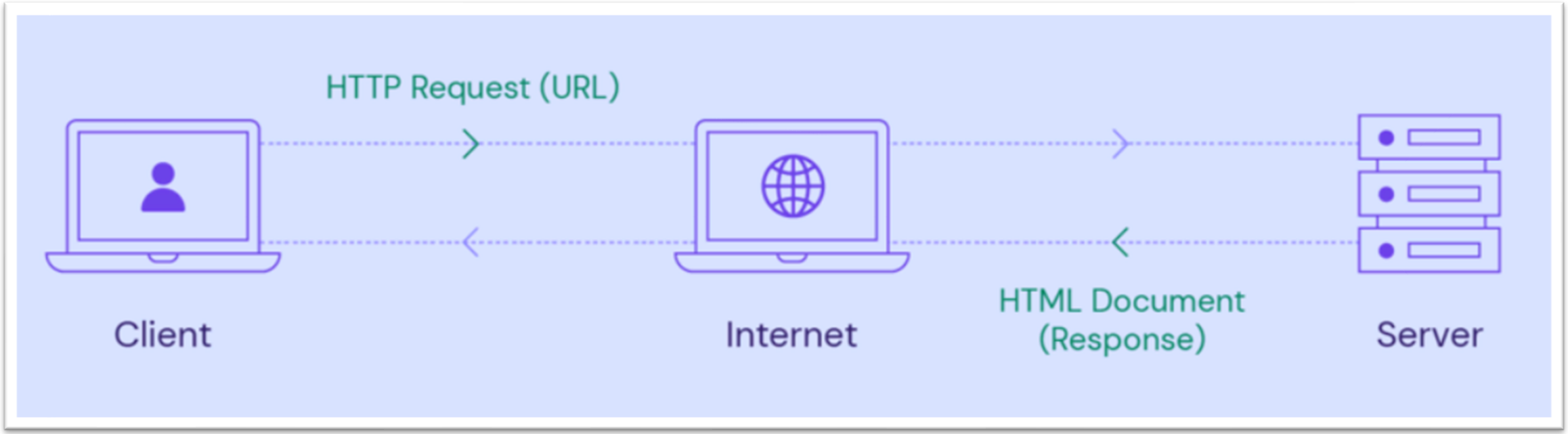
CSS (Cascading Style Sheets): A style sheet language used for describing the look and formatting of a document written in HTML. It separates the content from its presentation.

JavaScript: A programming language that enables interactive web pages. It runs in the browser and can dynamically update the content and respond to user actions.

Web Development: The process of building and maintaining websites. It involves web design, web content development, client-side scripting, server-side scripting, and network security configuration.

Responsive Web Design: Designing websites to provide an optimal viewing and interaction experience across a wide range of devices, from desktop computers to mobile phones.

DNS (Domain Name System): The system that translates human-readable domain names (like www.example.com) into IP addresses that machines use to identify each other on the network.


HTTP Request (URL)
Client
Internet
HTML Document
(Response)
Server

Frontend (Client-Side) Development

HTML (Hypertext Markup Language) : Not a programming language but essential for structuring web content.

CSS (Cascading Style Sheets) : Used for styling and layout. It defines how HTML elements are displayed on the screen.

JavaScript : The primary language for building interactive and dynamic elements on a webpage.

TypeScript : A superset of JavaScript that adds static typing, making it easier to catch errors during development.

React : A JavaScript library for building user interfaces, developed by Facebook.

Angular : A TypeScript-based framework for building dynamic web applications, maintained by Google.

Vue.js : A progressive JavaScript framework for building user interfaces.

Backend (Server-Side) Development:

Node.js (JavaScript/TypeScript): A runtime environment that allows server-side execution of JavaScript and TypeScript. Often used with frameworks like Express.js.

Express.js (JavaScript/Node.js): A minimal and flexible Node.js web application framework.

Java: Known for its platform independence. Popular frameworks include Spring and JavaServer Faces (JSF).

C# (C Sharp): Developed by Microsoft, often used with the ASP.NET framework.

PHP: Widely used for web development, especially for server-side scripting.

Other: Python, Ruby, Django (Python), Ruby on Rails (Ruby).

HTML (Hypertext Markup Language)

HTML (Hypertext Markup Language) is the standard markup language for creating and designing web pages. It is used to structure content on the web, defining elements and their relationships within a document. HTML consists of a **series of elements**, each represented by a **tag**, and these tags are used to enclose or wrap different parts of content to make them appear or behave in a certain way.

Basic Structure: A simple HTML document typically includes the following components:

```
<!DOCTYPE html>
<html>
 <head>
    <title>Page Title</title>
 </head>
 <body>
     <h1>My First Heading</h1>
     <p>My first paragraph.</p>
 </body>
</html>
```

- The **<!DOCTYPE html>** declaration defines that this document is an HTML5 document
- The **<html>** element is the root element of an HTML page
- The **<head>** element contains meta information about the HTML page
- The **<title>** element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The **<body>** element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The **<h1>** element defines a large heading
- The **<p>** element defines a paragraph

What is an HTML Element?

An HTML element is defined by a start tag, some content, and an end tag:
<tagname> Content goes here... **</tagname> // [angle brackets “<“and “>” are html tags]**

HTML attributes

HTML attributes provide additional information about HTML elements and are always included in the opening tag. They are typically added as **name/value pairs**, separated by an equals sign **(=)**. Here's an overview of some common HTML attributes:

class : Used to define one or more class names for an element. Classes are often used to apply CSS styles or JavaScript behaviors.
Example: ...

id : Provides a unique identifier for an element, which can be used for styling with CSS or for targeting with JavaScript.
Example: This is a unique paragraph.

style : Used to apply inline CSS styles to an element.
Example: Styled paragraph.

src : Specifies the source URL for external resources, such as images, scripts, or iframes.
Example: <img src="image.jpg" alt="Description of the image"> **//code**

alt: Provides alternative text for images. It is displayed if the image cannot be loaded and is also used for accessibility.
Example: <img src="image.jpg" alt="Description of the image">

width and height : Used to set the width and height of certain elements, such as images or table cells.
Example: <img src="image.jpg" width="300" height="200" alt="Description">

href : Specifies the URL for hyperlinks, indicating the destination of the link.

Example:
<a href="https://www.example.com">Visit Example.com</a>
<a href="mailto:your@email.com">Send Email</a>
<a href="tel:+1234567890">Call Us</a>

target : Specifies where to open a linked document or where to submit a form. Common values include _blank, _self, _parent, and _top.

Example: <a href="https://www.example.com" target="_blank">Open in a new tab</a>

colspan and rowspan : Used in table cells to specify how many columns or rows a cell should span.

Example: <td colspan="2">Spanning two columns</td>

disabled : Disables an input element, making it non-interactive.

Example: <input type="text" disabled>

placeholder : Provides a hint or example text for the user within an input field.

Example: <input type="text" placeholder="Enter your name">

checked : Used with checkbox and radio input types to set the default checked state.

Example: <input type="checkbox" checked>

multiple : Allows users to select multiple options in a <select> element.

Example: <select multiple>...</select>

value : Sets the initial value for input elements.

Example: <input type="text" value="Default Text">

HTML - The Head Element

The HTML **<head>** element is a container for the following elements**: <meta>, <title>, <link>, <style>, <script>,** and **<base>.**

Meta Tags

The **<meta>** element is typically used to specify the character set, page description, keywords, author of the document, and viewport settings.

Example:

```
<meta charset="UTF-8"> [the character encoding for the document and helps the browser correctly interpret and display characters]
<meta name="description" content="Free Web tutorials"> [Description and Keywords for Search Engines]
<meta name="keywords" content="HTML, CSS, JavaScript">
<meta name="author" content="John Doe">[Author and Copyright Information]
<meta name="copyright" content="© 2023 Your Company">
<meta name="viewport" content="width=device-width, initial-scale=1.0"> [Viewport Settings for Responsive Design]
```

Title

The **<title>** element is required and it defines the title of the document

Example: <title>A Meaningful Page Title</title>

Favicons

Favicons are small icons associated with a website, displayed in the browser's address bar or tab.

Example: <link rel="icon" href="favicon.ico" type="image/x-icon">

Base

The **<base>** element specifies the base URL and/or target for all relative URLs in a page

Example: <base href="https://www.w3schools.com/" target="_blank">

Stylesheets and Scripts

Stylesheets (CSS) and scripts (JavaScript) can be linked or included directly within the <head> element to apply styles and behavior to the document.

Example 1:

```
<style>
        body {background-color: powderblue;}
        h1 {color: red;}
        p {color: blue;}
</style>
```

Example 2: <link rel="stylesheet" href="mystyle.css">

Example 3:

```
<script>
      function myFunction() {
        document.getElementById("demo").innerHTML = "Hello JavaScript!";
      }
</script>
```

Example 4: <script src="script.js"></script>

HTML - The Body Element

Inside the **<body>** tag of an HTML document, you typically include various tags to structure and organize the content of your web page. Here are some commonly used tags for creating a basic layout:

1. **<header> :** Defines a header for a document or a section. Typically contains introductory content or navigation elements.
2. **<nav> :** Represents a navigation menu, often placed within the header or another section of the page.
3. **<main> :** Contains the main content of the document. Typically excludes headers, footers, and navigation elements.
4. **<section> :** Represents a generic section of a document. It's often used to group related content together.//must have title if not use div tag
5. **<article>:** Represents an independent, self-contained piece of content. It can be an article, blog post, or similar items.
6. **<aside>:** Defines content that is tangentially related to the content around it, such as a sidebar or related links.
7. **<footer>:** Represents a footer for a document or a section. Typically contains copyright information, links, or other metadata.
8. ** :** A generic container used to group and style content. It doesn't carry any semantic meaning by itself.

Heading Tags (<h1> to <h6>) : Heading tags are used to define headings in a document, with <h1> as the highest level (most important) heading and <h6> as the lowest level.

Example: <h1>This is a Heading</h1>

Paragraph Tag ( ) : The tag is used to define paragraphs of text.

Example: This is a paragraph of text.

Span Tag (): The tag is similar to , but it is an inline container. It is often used for styling specific portions of text.

Example: This is red text.

Anchor Tag (<a>) : The <a> tag is used to create hyperlinks, allowing users to navigate to other pages or resources.

Example: <a href="https://www.example.com/#section1" target="_blank" >Visit Example.com</a>

Image Tag (<img>) : The <img> tag is used to embed images in a webpage.

Example: <img src="image.jpg" width="300" height="200" alt="Description of the image">

List Tags (<ul>, <ol>, <li>) : <ul> creates an unordered (bulleted) list. <ol> creates an ordered (numbered) list.
<li> defines list items in both unordered and ordered lists.

Ordered List (<ol>): An ordered list is used for items that have a specific sequence or order. Each item is preceded by a number.

```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
</ol>
```

Unordered List (<ul>): An unordered list is used for items that don't have a specific sequence or order. Each item is typically preceded by a bullet point.

```
<ul>
    <li>Apple</li>
    <li>Orange</li>
    <li>Banana</li>
</ul>
```

Description List (<dl>): A description list is used for terms and their corresponding descriptions.

```
<dl>
    <dt>HTML</dt>
    <dd>HyperText Markup Language</dd>
    <dt>CSS</dt>
    <dd>Cascading Style Sheets</dd>
</dl>
```

Comments <!-- text --!> : Comments are not displayed in the browser but can be useful for developers or anyone reviewing the code. Comments start with <!-- and end with -->.

**Line Breaks
 :** The line break is used to create a new line within the text.

Horizontal rule <hr /> : <hr> tag is used to create a visual break or horizontal line between sections of content.

Table Tags

- The **<table>** element is used to define the table.
- The **<thead>** element contains header rows with **<th>** (table header) elements.
- The **<tbody>** element contains the main content of the table with **<tr>** (table row) and **<td>** (table data/cell) elements.

<table> Attributes

border: Specifies the width of the border around the table. This attribute is rarely used, as styling with CSS is generally preferred.

width: Sets the width of the table.

cellpadding: Specifies the space between the cell content and cell border.

cellspacing: Specifies the space between cells.

<tr> Attributes

bgcolor: Sets the background color for a table row.

<th> and <td> Attributes

colspan: Specifies the number of columns a table header cell or table data cell should span.

rowspan: Specifies the number of rows a table header cell or table data cell should span.

align: Sets the horizontal alignment of the content within a cell. (Deprecated in HTML5, use CSS instead)

valign: Sets the vertical alignment of the content within a cell. (Deprecated in HTML5, use CSS instead)

bgcolor: Sets the background color for a table header cell or table data cell.

Example:

```
<table border="1" cellspacing="5" cellpadding="10">
  <thead>
    <tr bgcolor="#cccccc">
      <th colspan="2">Header spanning two columns</th>
    </tr>
  </thead>
  <tbody>
    <tr bgcolor="#f2f2f2">
      <td rowspan="2">Data spanning two rows</td>
      <td align="center">Centered Text</td>
    </tr>
    <tr>
      <td valign="top">Top-aligned Text</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td colspan="2" bgcolor="#ff0000">Footer Cell</td>
    </tr>
  </tfoot>
</table>
```

Form Tags (<form>, <input>, <button>): <form> defines an HTML form for user input. <input> is used to create various form controls. <button> defines a clickable button.

Text Input (type="text") : This creates a single-line text input field. It's often used for usernames, email addresses, or any short text input.

Number Input (type="number") : This creates a numeric input field, often used for quantities. The min, max, and step attributes define constraints on the allowed values.

Email Input (type="email") : This is used for email addresses and can help browsers validate that the entered text is a valid email format.

Password Input (type="password") : This creates a password input field where the entered characters are typically masked (e.g., as asterisks) for security.

Radio Buttons (type="radio") : Radio buttons allow users to choose only one option from a set of options. The name attribute groups related radio buttons together.

Checkboxes (type="checkbox") : Checkboxes allow users to select one or more options. Each checkbox should have a unique name attribute.

Submit Button (type="submit") : This creates a button that, when clicked, submits the form data to the server.

Reset Button (type="reset") : This creates a button that, when clicked, resets the form fields to their default values.

File Input (type="file") : This allows users to upload files from their local device. The selected file(s) are sent to the server when the form is submitted.

Hidden Input (type="hidden") : This creates an invisible input field. It's used to store data on the client side, which is not displayed but gets submitted when the form is submitted.

Date Input (type="date") : This creates an input field for selecting a date. It usually opens a date picker on supporting browsers.

URL Input (type="url") : This is used for URLs and can help browsers validate that the entered text is a valid URL format.

Range Input (type="range") : This creates a slider control allowing users to choose a numeric value within a specified range.

<textarea> Element : The <textarea> element in HTML is used to create a multiline text input field. Users can input larger amounts of text, such as paragraphs or multiple lines of code.

<button> Element : The <button> element in HTML is used to create a clickable button. It can be used for various purposes, including triggering form submissions.

```html
<form action="/submit" method="post">

  <!-- Text Input -->
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" placeholder="Enter your username" required>

  <!-- Password Input -->
  <label for="password">Password:</label>
  <input type="password" id="password" name="password" placeholder="Enter your password" required>

  <!-- Radio Buttons -->
  <label>Gender:</label>
  <label><input type="radio" name="gender" value="male"> Male</label>
  <label><input type="radio" name="gender" value="female"> Female</label>

  <!-- Checkboxes -->
  <label>Interests:</label>
  <label><input type="checkbox" name="interests" value="reading"> Reading</label>
  <label><input type="checkbox" name="interests" value="coding"> Coding</label>

  <!-- Textarea -->
  <label for="message">Message:</label>
  <textarea id="message" name="message" placeholder="Enter your message" rows="4" required></textarea>

  <!-- Submit Button -->
  <input type="submit" value="Submit">

</form>
```

Block-level Elements

Block-level elements create "**blocks**" or "**boxes**" on a web page, and they typically start on a new line and extend the full width of their container.

Examples:

-
- <p>
- <h1>, <h2>, ..., <h6>
- <ul>, <ol>, <li>
- <table>
- <form>

Behavior

- Block-level elements create a "block" or a "box" that acts as a container for other elements.
- They stack vertically on top of each other, each starting on a new line.
- Block-level elements can contain other block-level and inline-level elements.

Use Cases

- Used for structural elements, like dividing sections of a page.
- Typically used for larger structures like paragraphs, headings, divs, and lists.
- Styling: Commonly manipulated using CSS for layout purposes.
- Can have properties like width, height, margin, and padding.

Inline-level Elements

Inline-level elements do not create a new "block" on the page; instead, they flow within the content and only take up as much width as necessary.

Examples:

-
- <a>
- ,
- <img>
-
- <input>

Behavior

- Inline-level elements flow within the content, appearing on the same line as much content as will fit.
- They only take up as much width as necessary.
- Inline-level elements cannot contain block-level elements but can contain other inline-level elements.

Use Cases

- Used for elements that do not need to start on a new line and only affect the content they enclose.
- Suitable for smaller in-line content like spans, links, and emphasis.

Styling

- Often styled with properties like font, color, and text-decoration.
- Cannot have properties like width or height applied directly.

In HTML and CSS, measurements are used to define the size and dimensions of various elements on a webpage. Here are explanations for some common measurement units:

Pixels (px)

Pixels are the most common unit of measurement in web design. One pixel is a single dot on a screen, and the size of elements in pixels corresponds directly to the screen resolution.

Usage: width: 200px; or font-size: 16px;

Note: While pixels provide precise control, they may not scale well across different devices with varying screen resolutions.

Percentage (%)

Percentage measurements are relative to the parent element. For example, if a parent container is 200 pixels wide and a child element is set to width: 50%;, it will be 100 pixels wide.

Usage: width: 50%; or margin-top: 10%;

Note: Percentages are useful for creating responsive designs that adapt to different screen sizes.

Points (pt)

Description: Points are commonly used in print design but can also be used in web design. One point is equal to 1/72 of an inch. Unlike pixels, points are somewhat scalable, making them suitable for print styles.

Usage: font-size: 12pt;

Note: Points are less common in web design, and pixels or percentages are generally preferred for better cross-device compatibility.

Em

The "em" is a scalable unit that is relative to the font-size of the parent or nearest ancestor. For example, if the font size of a parent element is 16 pixels, 1em would be equivalent to 16 pixels.

Usage: font-size: 1.2em; or margin: 1em;

Note: Using ems can help create more flexible and accessible designs, especially when dealing with text.

Rem

Similar to em, the "rem" (root em) is relative to the font-size of the root element (usually the <html> tag). This makes it more predictable and easier to manage than em.

Usage: font-size: 1.2rem; or margin: 1rem;

Note: Rem units are often preferred for overall layout and spacing.

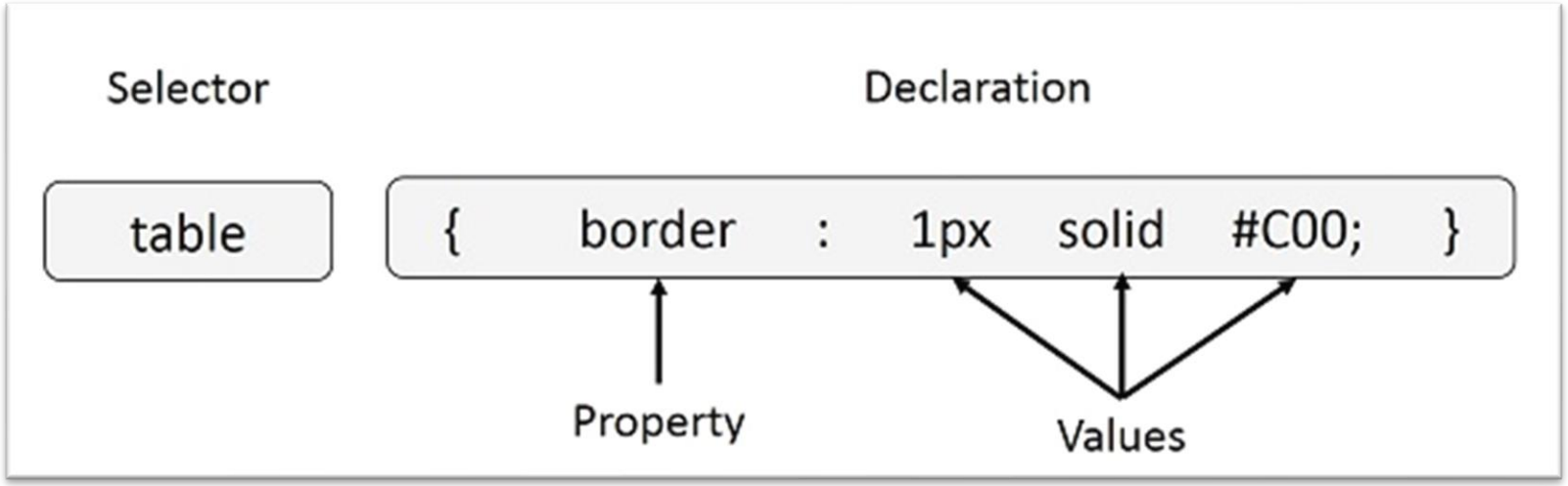
It's important to choose measurement units based on the specific requirements of your design and layout. For responsive and scalable designs, **percentages, ems, and rems** are often preferred over fixed units like pixels or points.

********** Exercies **********

Cascading Style Sheets (CSS)

CSS Syntax Structure

A CSS rule consists of a **selector** and a **declaration block**. The basic syntax looks like this:



How to Insert CSS File

1. Inline Style Sheet

```
<h1 style="color: blue; text-align: center;">This is a heading</h1>
<p style="color: red; font-size: 20px;">This is a paragraph.</p>
```

2. Internal Style Sheet

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Internal CSS Example</title>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
    h1 {
      color: blue;
      text-align: center;
    }
    p {
      color: red;
      font-size: 20px;
    }
  </style>
</head>
```

3. External Style Sheet

```
<link rel="stylesheet" type="text/css" href="styles.css">
```

CSS Selectors

CSS selectors determine which elements in an HTML document will be styled by a particular rule. Here are some common CSS selectors:

Universal Selector (*) : Selects all elements on the page.

```
* {
  /* styles here */
}
```

Element Selector : Selects all instances of a specific HTML element (e.g., <**p**>).

```
p {
  /* Styles for all <p> elements */
}
```

Class Selector(.classname) : Selects elements based on their class attribute (e.g., <**div class="my-class**">).

```
.my-class {
  /* Styles for elements with class "my-class" */
}
```

ID Selector(#id) : Selects a single element based on its ID attribute (e.g., <**div id="my-id**">).

```
#my-id {
  /* Styles for the element with ID "my-id" */
}
```

Descendant Selector (whitespace) : Selects an element that is a descendant of another specified element.

```
div p {
  /* Styles for all <p> elements inside a <div> */
}
```

Child Selector(>) : Selects an element that is a direct child of another specified element.

```
ul > li {
  /* Styles for direct child <li> elements of a <ul> */
}
```

Adjacent Sibling Selector (+) : Selects **an element** that is directly preceded by a specified element.

```
h2 + p {
  /* styles here */
}
```

General sibling selector (~) : The general sibling selector selects **all elements** that are next siblings of a specified element.

```
element1 ~ element2 {
  /* styles here */
}
```

Attribute Selector : Selects elements based on their attributes.

```
input[type="text"] {
  /* Styles for text input elements */
}
```

Pseudo-class Selector(:pseudo-class) : Selects elements based on their state or position. **Example**. :hover, :active, :focus, :first-child, :last-child, :nth-child(n), :nth-of-type(n), :not(selector), :nth-last-child(n), :checked.

```
a:hover {
  /* Styles for links when hovered */
}
```

Pseudo-element Selector (::pseudo-element) : Selects a specific part of an element. **Example**. :after, :before, :first-letter, first-line, : marker, :selection.

```
p::first-line {
  /* styles here */
}
```

CSS Text

Font Properties

font-family : Defines the font family for text.

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

font-size : Sets the size of the font.
[Eg. h1 - 48px, h2 - 32px, h3 - 28px, h4 - 24px, h5 - 20px, h6 - 18px]

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

font-weight : Specifies the thickness of the font characters.

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

font-style : Defines the style of the font (normal, italic, or oblique).

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

Keyword Values

1. `normal`: Standard weight (usually 400).
2. `bold`: Bold weight (usually 700).
3. `bolder`: Bolder than the inherited weight.
4. `lighter`: Lighter than the inherited weight.

Numeric Values

Numeric values range from 100 to 900, in increments of 100:

- **100**: Thin
- **200**: Extra Light (Ultra Light)
- **300**: Light
- **400**: Normal (Regular)
- **500**: Medium
- **600**: Semi Bold (Demi Bold)
- **700**: Bold
- **800**: Extra Bold (Ultra Bold)
- **900**: Black (Heavy)

Text Color and Decoration

color : Sets the color of the text. The color property accepts color values in different formats, including **named colors**, **hexadecimal values**, **RGB** values, and **HSL** values.

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

text-decoration : Specifies the decoration added to text (**underline, overline, line-through**).

```
a {
  text-decoration: none;
}
```

text-align : Specifies the horizontal alignment of text. (**left, right, center, justify, initial, inherit**)

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

line-height : Sets the height of a line of text.

```
p {
  line-height: 1.5;
}
```

text-indent : Specifies the indentation of the first line in a text block.

```
.indented-paragraph {
  text-indent: 20px;
}
```

text-overflow : Defines how the hidden text content behaves if it's overflowing

```
.example {
  width: 200px;            /* Define a fixed width for the container */
  white-space: nowrap;     /* Prevent the text from wrapping to a new line */
  overflow: hidden;        /* Hide any text that overflows the container */
  text-overflow: ellipsis; /* Display an ellipsis (...) to indicate overflowing text */
}
```

Spacing and Alignment

letter-spacing : Controls the space between characters.

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

word-spacing : Controls the space between words.

```css
p {
  word-spacing: 4px;
}
```

text-transform : Changes the capitalization of text. (**none**, **capitalize**, **uppercase**, **lowercase**)

```css
.uppercase-text {
  text-transform: uppercase;
}
```

white-space : Controls how white space inside an element is handled. (**normal**, **nowrap**, **pre**, **pre-line**, **pre-wrap**)

```css
.no-wrap {
  white-space: nowrap;
}
```

text-shadow : used to add shadow effects to text. Each shadow can have the following values:

Horizontal offset: **Required**. The position of the shadow horizontally.

Vertical offset: **Required**. The position of the shadow vertically.

Blur radius: **Optional**. The radius of the blur. The higher the number, the more blurred the shadow will be.

Color: **Optional**. The color of the shadow.

```css
.example3 {
  text-shadow: 1px 1px 2px #ff0000, -1px -1px 2px #0000ff;
}
```

CSS Background

The background property is used to set the background styling of an element. It can include properties such as **background-color**, **background-image**, **background-repeat, background-attachment**, **background-position**, and **background-size**. Additionally, you can use the **shorthand background property** to combine these values.

background-color

```
.element {
  background-color: #3498db;
}
```

background-image

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

background-image (linear-gradient)

```
.gradient {
    background-image: linear-gradient(45deg, red, blue);
}
```

- **0deg** is a gradient that goes from bottom to top.
- **90deg** is a gradient that goes from left to right.
- **180deg** is a gradient that goes from top to bottom.
- **270deg** is a gradient that goes from right to left.

Ultimate CSS Gradient Generator - ColorZilla.com

- **to top**: From bottom to top
- **to bottom**: From top to bottom
- **to left**: From right to left
- **to right**: From left to right
- **to top left**: From bottom right to top left
- **to top right**: From bottom left to top right
- **to bottom left**: From top right to bottom left
- **to bottom right**: From top left to bottom right

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

background-image (radial-gradient)

shape: Specifies the shape of the gradient. It can be circle or ellipse.
size: Specifies the size of the gradient. It can be keywords like closest-side, farthest-side, closest-corner, farthest-corner, or specific values (like px or %).
position: Specifies the center of the gradient. By default, it's center.
start-color, ..., last-color: Specifies the colors of the gradient. You can have as many color stops as you like.

```
background: radial-gradient(shape size at position, start-color, ..., last-color);
```

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

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

background-position : defines the starting position of the background image. It can take values like **top**, **center**, **bottom**, **left**, **right**, or a combination of values like **top left**.

```
.element {
  background-position: center top;
}
```

Ultimate CSS Gradient Generator - ColorZilla.com

background-repeat : The background-repeat property controls how a background image is repeated. Common values include **repeat**, **no-repeat**, **repeat-x**, and **repeat-y**.

```
.element {
  background-repeat: no-repeat;
}
```

background-attachment : specifies whether the background image is fixed or scrolls with the rest of the page. Values include **scroll** and **fixed**.

```
.element {
  background-attachment: fixed;
}
```

background-size : to specify the size of the background images for elements. Values include **auto**, **cover**, **contain**, **<length-width> <length-height> (eg.px, em, %);**

```
.element {
  background-size: cover;
}
```

Shorthand background Property : You can use the shorthand background property to combine multiple background-related properties into one. When using the **shorthand** property the **order** of the property values is:
background-color, background-image, background-repeat, background-attachment, background-position, background-size
(needs to be specified after background-position with a slash / **)**

```
.element {
  background: #ffcc00 url('example-image.jpg') no-repeat fixed center/contain;
}
```

CSS Box Model

1. **Content**: The actual content of the box, such as text, images, or other elements.
2. **Padding**: The space between the content and the border. Padding is used to create space within the box.
3. **Border**: A border surrounding the padding (if any) and content. The border can have its own style, color, and width.
4. **Margin**: The space outside the border. Margins are used to create space between the box and other elements on the page.

1. content (width and height) : Define the width and height of the content box.

```
.box {
  width: 200px;
  height: 100px;
}
```

2. padding : Padding is the space between the element's content and its border. It controls the inner space of an element, affecting its content but not the overall size of the element.

```
div {
  padding: 10px;
}
```

This sets padding of 10 pixels on all sides of the **div** element.

Shorthand: Like margins, you can use shorthand for individual sides or all sides at once:

```
div {
  padding-top: 10px;
  padding-right: 20px;
  padding-bottom: 10px;
  padding-left: 20px;
}
```

(OR)

```
div {
  padding: 10px 20px;
}
```

3. border : Sets the style, color, and width of the border. Specifies the style of the border, such as **solid**, **dotted**, **dashed**, **double**, etc.

```
.box {
  border: 2px solid #333;
}
```

4. margin : Margin is the space outside the border of an element. It creates space between the element's border and its surrounding elements.

```
div {
  margin: 10px;
}
```

This sets a margin of 10 pixels on all sides of the **div** element.

Shorthand: You can use shorthand to set individual margins for each side:

```
div {
  margin-top: 10px;
  margin-right: 20px;
  margin-bottom: 10px;
  margin-left: 20px;
}
```

(OR)

```
div {
  margin: 10px 20px;
}
```

CSS Layout – Display properties

In CSS, the display property is a versatile and fundamental property that defines the layout behavior of an HTML element. It determines how an element generates boxes and how those boxes interact with each other. Here are some commonly used values for the display property:

block : A block-level element generates a block box that typically starts on a new line and extends the full width of its container.

```
.block-element {
  display: block;
  width: 100%;
  background-color: #f2f2f2;
  padding: 10px;
  margin-bottom: 10px;
}
```

inline : An inline-level element generates an inline box that does not start on a new line and only takes up as much width as necessary.

```
.inline-element {
  display: inline;
  background-color: #e0e0e0;
  padding: 5px;
  margin-right: 10px;
}
```

inline-block : An inline-block element is a combination of both block and inline elements. It generates a block box that flows inline with the surrounding content.

```
.inline-block-element {
  display: inline-block;
  width: 100px;
  height: 50px;
  background-color: #c0c0c0;
  margin-right: 10px;
}
```

none : The **display: none;** property is used to hide an element. The element will not be rendered, and it won't take up any space on the page.

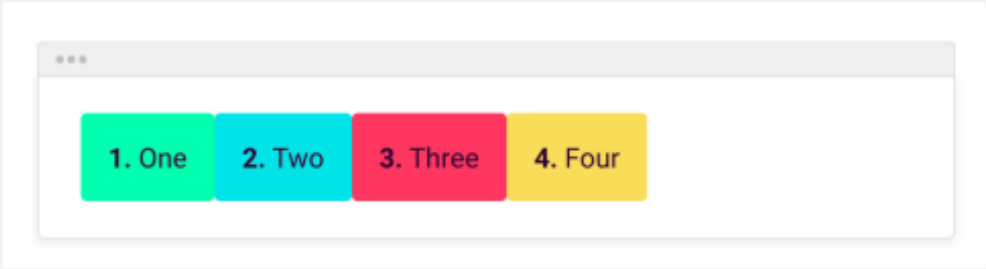
```
.hidden-element {
  display: none;
}
```

flex : The **display: flex;** property establishes a flex container, and the child elements become flex items. It allows for easy alignment and distribution of space along a single axis.

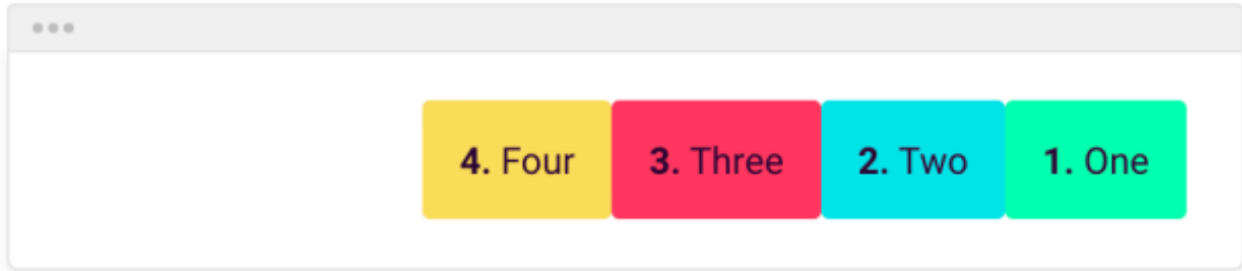
```
.flex-container {
  display: flex;
  justify-content: space-between;
}

.flex-item {
  flex: 1;
}
```

Flex-direction => Defines how flexbox items are ordered within a flexbox container.
Row,row-reverse, column, column-reverse

flex-direction: row;



flex-direction: row-reverse;



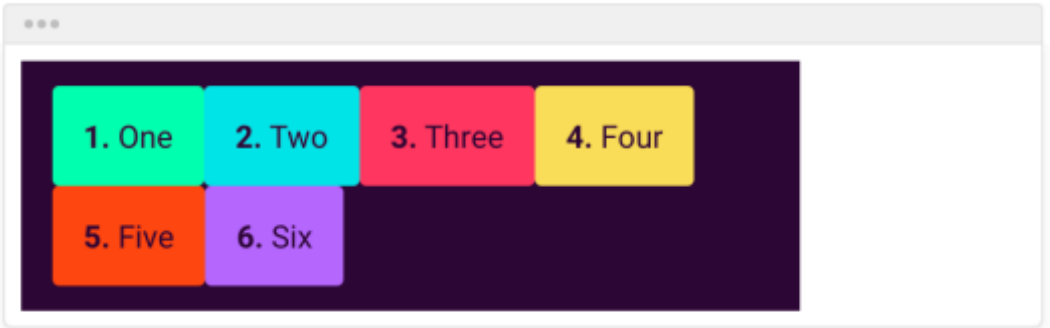
flex-direction: column;

flex-direction: column-reverse;

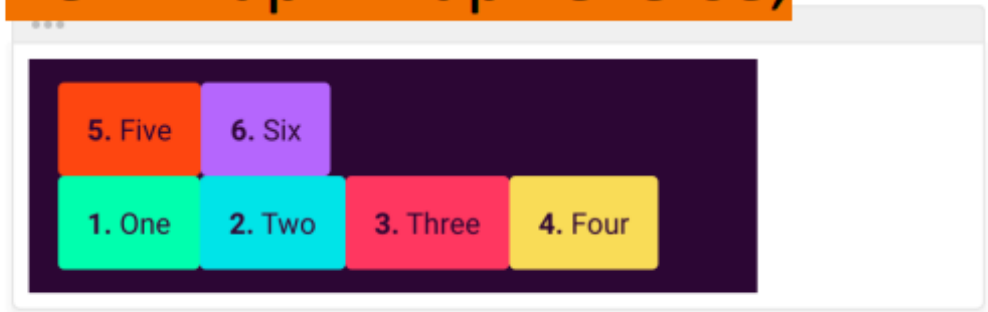
Flex-wrap => Defines if flexbox items appear on a single line or on multiple lines within a flexbox container.

flex-wrap: nowrap;

flex-wrap: wrap;

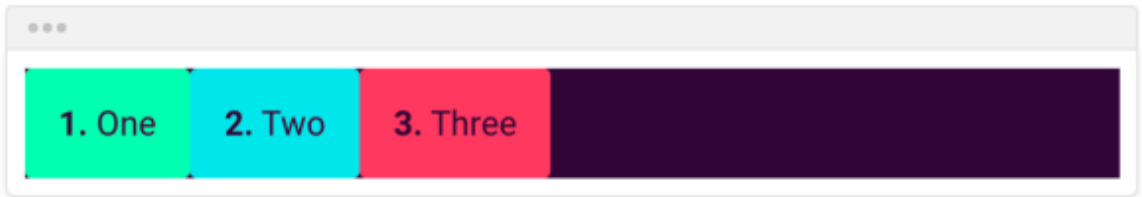


flex-wrap: wrap-reverse;

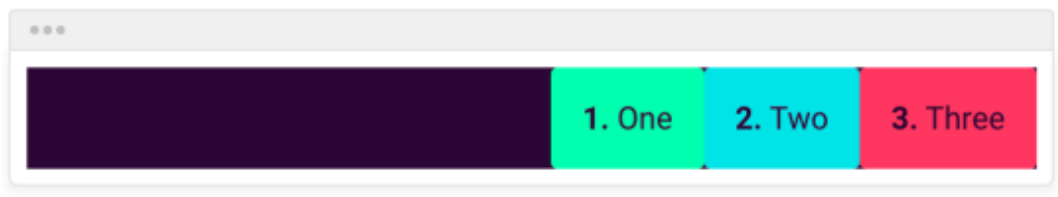


justify-content => Defines how flexbox/grid items are aligned according to the main axis, within a flexbox/grid container.

justify-content: flex-start;



justify-content: flex-end;



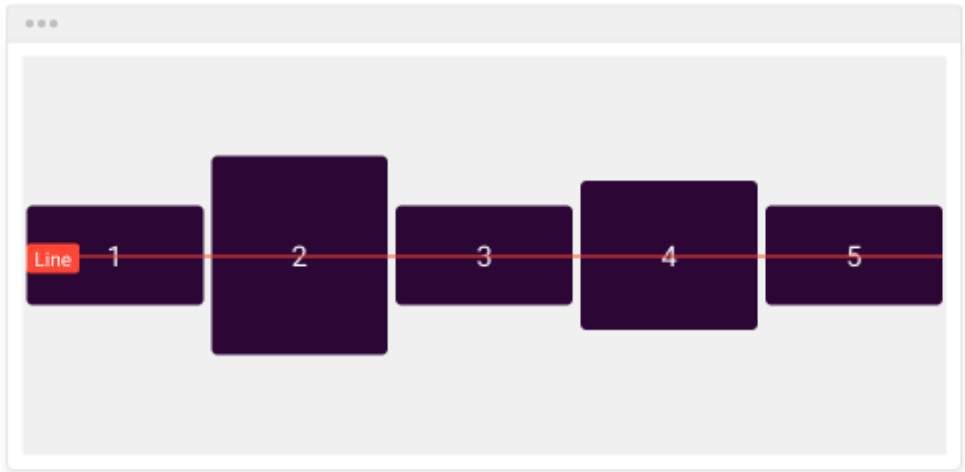
justify-content: center;

justify-content: space-between;

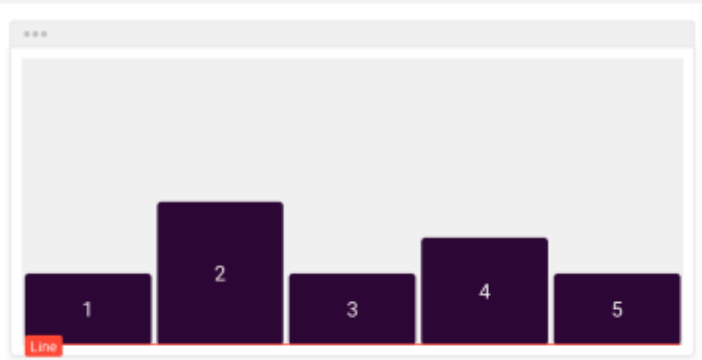
justify-content: space-around;

Align-items => Defines how flexbox items are aligned according to the cross axis, within a line of a flexbox container.

align-items: flex-start;



align-items: flex-end;



align-items: center;

CSS Layout - Positioning

The position property is a fundamental part of CSS layout, and it allows you to control the positioning of elements. Here are the values you can use with the position property.

- **Static (**Default value) : Elements are positioned according to the normal flow of the document.

```
.example {
  position: static;
}
```

- **Relative:** Positioned relative to its normal position. You can then move it using the top, right, bottom, and left properties.

```
.example {
  position: relative;
  top: 10px;
  left: 20px;
}
```

- **Absolute:** Positioned relative to its closest positioned ancestor. If no positioned ancestor is found, it's positioned relative to the initial containing block (usually the <html> element).

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

- **Fixed:** Positioned relative to the browser window. It doesn't move when the page is scrolled.

```
.example {
  position: fixed;
  top: 10px;
  right: 20px;
}
```

- **Sticky:** Acts like relative until an element crosses a specified point during scrolling, then it becomes fixed.

```
.example {
  position: sticky;
  top: 10px;
}
```

- **Z-index:** Determines the stack order of positioned elements. Elements with a higher z-index are stacked above those with a lower z-index.

```
.example {
  position: absolute;
  z-index: 2;
}
```

CSS Lists

List Style Type : The list-style-type property defines the appearance of the list item marker (bullet, number, etc.).Common values for list-style-type include **disc**, **circle**, **square** for **unordered lists**, and **decimal**(1, 2) , **lower-alpha**(a, b), **upper-roman**(I, II) for **ordered lists**. To remove default list styling (bullets or numbers), you can set list-style-type to **none**.

```
ul {
  list-style-type: disc; /* For unordered lists */
}

ol {
  list-style-type: decimal; /* For ordered lists */
}
```

```
ul, ol {
  list-style-type: none;
}
```

List Style Position : The list-style-position property specifies whether the list item marker should be **inside** or **outside** the content flow.

```
ul {
  list-style-position: outside; /* Default */
}

ol {
  list-style-position: inside;
}
```

List Style Image : The list-style-image property allows you to use a custom image as the list item marker.

```
ul {
  list-style-image: url('bullet-image.png'); /* For unordered lists */
}
```

CSS Table

Table Layout

width: Specifies the width of the table.
border-collapse : Combines table borders into a single border, eliminating any space between table cells.

```
table {
  width: 100%;
  border-collapse: collapse;
}
```

Table Cell Padding and Spacing

padding : Adds space within each table cell.
border-spacing : Sets the space between table borders.

```
td, th {
  padding: 8px;
  border-spacing: 0;
}
```

Border and Border Color:

border : Sets the border of the table, header cells (th), and data cells (td).
border-color : Specifies the color of the border.

```
table, th, td {
  border: 1px solid #ddd;
}
```

CSS Layout – Float and clear

The float property in CSS is used for positioning and aligning elements. It was traditionally used for creating multi-column layouts, but with the advent of Flexbox and Grid, its use for layout purposes has decreased. However, it is still used for certain scenarios, such as wrapping text around images.

```
.float-example {
  float: left; /* or right */
  width: 200px; /* Set a width for the floated element */
  margin: 0 20px 20px 0; /* Margin to create space between floated elem
}
```

Floats values:

- **none**: **Default**. The element does not float.
- **left**: Float the element to the left.
- **right**: Float the element to the right.

Clearing Floats: Floating elements can affect the layout of subsequent elements. To prevent this, the clear property is used on an element after the floated elements.

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

Clear values

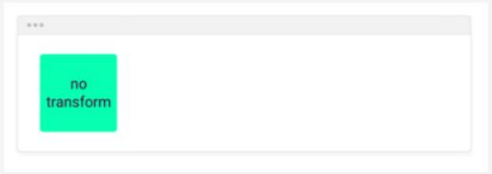
- **left**: No floating elements allowed on the left side.
- **right**: No floating elements allowed on the right side.
- **both**: No floating elements allowed on either side.
- **none**: Allows floating elements on both sides.

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

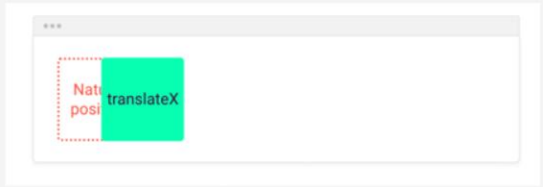
CSS Transform

Translate => Move the element.

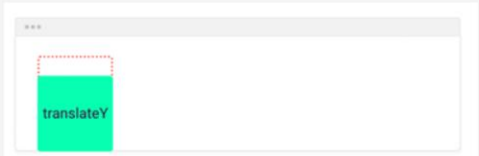
transform: none;



transform: translateX(40px);



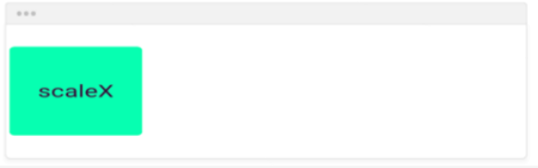
transform: translateY(20px);



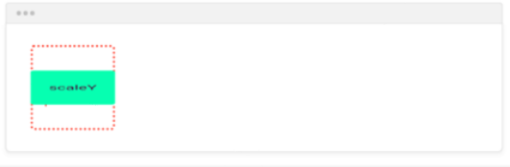
transform: translate(20px, -10%);

Scale the element.

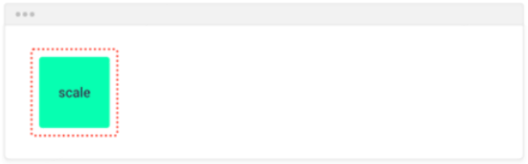
transform: scaleX(1.5);



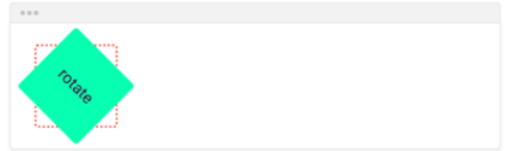
transform: scaleY(0.4);



transform: scale(0.8, 0.8);

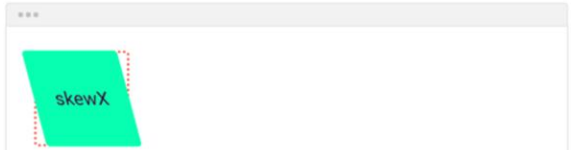


transform: rotate(45deg);

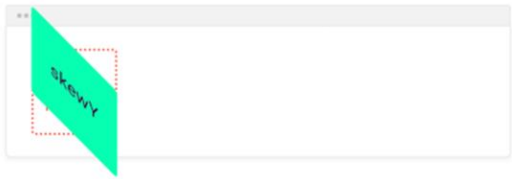


Skew the element

transform: skewX(15deg);



transform: skewY(45deg);



CSS Transition

The transition property in CSS is used to create smooth transitions between different states of an element. It allows you to define the **duration**, **timing function**, and **delay** for the transition, making it easy to animate changes to CSS **properties**.

The transition property is a shorthand for the following individual properties:

- **transition-property**: Specifies the name of the CSS property to which the transition is applied.
- **transition-duration**: specifies how many seconds (s) or milliseconds (ms) a transition effect takes to complete.
- **transition-timing-function**: Specifies the timing function to be used for the transition. [ease, linear, ease-in, ease-out, ease-in-out,…)
- **transition-delay**: Specifies the delay before the transition starts.

transition - CSS: Cascading Style Sheets | MDN (mozilla.org)

```css
.example2 {
  width: 100px;
  height: 100px;
  background-color: blue;
  transition: width 2s ease-in-out 0.5s;
}
.example2:hover {
  width: 200px;
}
```

CSS Layout – Width , Height

Width Property: The width property is used to set the width of an element. It can be set using various units such as pixels (px), percentages (%), em units (em), viewport width (vw), and more.

```css
.container {
  width: 80%; /* 80% of the parent container width */
}

.box {
  width: 200px; /* Fixed width in pixels */
}

.image {
  width: 100%; /* Takes the full width of its container */
}
```

Height Property: Similarly, the height property is used to set the height of an element. Like width, it can be specified using various units such as pixels, percentages, em units, viewport height (vh), and more.

```css
.container {
  height: 400px; /* Fixed height in pixels */
}

.box {
  height: 50%; /* 50% of the parent container height */
}

.image {
  height: 100vh; /* Takes the full height of the viewport */
}
```

Max-width and Max-height: You can also use max-width and max-height properties to set the maximum width and height that an element can have. This is useful when you want to limit the size of an element.

```
.container {
  max-width: 1200px; /* Maximum width of 1200 pixels */
  max-height: 600px; /* Maximum height of 600 pixels */
}
```

Min-width and Min-height: Similarly, min-width and min-height properties set the minimum width and height for an element. This is useful when you want to ensure that an element is at least a certain size.

```
.container {
  min-width: 300px; /* Minimum width of 300 pixels */
  min-height: 150px; /* Minimum height of 150 pixels */
}
```

Usage scenarios:

Responsive Web Design: max-width is commonly used in responsive web design to ensure that elements do not become too wide on larger screens, while min-width ensures that they do not become too narrow on smaller screens.

Images and Media: max-width is commonly applied to images and media elements to prevent them from exceeding certain dimensions and breaking the layout.

CSS Media Queries

CSS media queries are written in a specific format that follows the @media rule. Here's the basic syntax for CSS media queries:

```
/* Default styles for all devices */

@media media_type and (media_feature: value) {
  /* Styles for devices that match the media query conditions */
}
```

media_type : Specifies the type of media to which the styles should apply. Common values include **all**, **screen**, **print**, **speech**, etc.

media_feature : Defines the condition or feature to be checked. Common features include **width**, **height**, **min-width**, **max-width**, **orientation**, **resolution**, etc.

value : Specifies the value or range of values for the media feature.

Width-Based Media Query:

```
/* Styles for screens with a maximum width of 600 pixels */
@media screen and (max-width: 600px) {
  /* Styles go here */
}
```

Device Orientation Media Query:

```
/* Styles for screens in landscape orientation */
@media screen and (orientation: landscape) {
  /* Styles go here */
}
```

Combining Multiple Conditions:

```
/* Styles for screens with a width between 600 and 1024 pixels in landscape orientation */
@media screen and (min-width: 600px) and (max-width: 1024px) and (orientation: landscape) {
  /* Styles go here */
}
```

Resolution-Based Media Query:

```
/* Styles for screens with a minimum resolution of 300dpi */
@media screen and (min-resolution: 300dpi) {
  /* Styles go here */
}
```

Mobile Devices:

```css
/* Default styles for all devices */

/* Styles for devices with a maximum width of 767 pixels (typically mobile devices) */
@media screen and (max-width: 767px) {
  /* Mobile styles go here */
}
```

Tablets:

```css
/* Default styles for all devices */

/* Styles for devices with a width between 768 and 1024 pixels (typically tablets) */
@media screen and (min-width: 768px) and (max-width: 1024px) {
  /* Tablet styles go here */
}
```

Small Laptops and Desktops:

```css
/* Default styles for all devices */

/* Styles for devices with a width between 1025 and 1366 pixels (small laptops and desktops) */
@media screen and (min-width: 1025px) and (max-width: 1366px) {
  /* Laptop/desktop styles go here */
}
```

Large Desktops:

```css
/* Default styles for all devices */

/* Styles for devices with a minimum width of 1367 pixels (large desktops) */
@media screen and (min-width: 1367px) {
  /* Large desktop styles go here */
}
```

What is SASS and SCSS?

- Sass is a CSS is a preprocessor scripting language.
- SCSS syntax is very similar to CSS, but it allows for the use of **variables**, **nesting**, **mixins**, and other programming constructs. SCSS has a file extension of **.scss**. The aim is to make the coding process simpler and more efficient.
- .sass syntax removes semicolons and curly brackets, and is arguably easier to type

1. Install Node.js (https://nodejs.org/en)
2. npm install -g sass
3. sass --watch input.scss output.css (OR) sass --watch assets/scss/styles.scss:assets/css/styles.css

SCSS Variables : variables are defined using the "**$**" symbol followed by the **variable name**. You can assign **colors**, **font stacks**, or any CSS value.

```scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

SCSS Nesting : Sass allows us to write our styles in a nested hierarchy, like we would in HTML. We can place selectors inside the code block of other selectors and Sass will automatically combine the outer rule's selector with that of the inner rule.

```scss
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

```css
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

```scss
.form {
  border: 1px solid #ccc;

  &__submit {
    padding: 1rem .5rem;

    &--disabled {
      cursor: no-drop;
    }
  }
}
```

```css
.form {
  border: 1px solid #ccc;
}

.form__submit {
  padding: 1rem 0.5rem;
}

.form__submit--disabled {
  cursor: no-drop;
}
```

SCSS Partials : A partial is a Sass file named with a **leading underscore**. You might name it something like **_partial.scss**. The underscore lets Sass know that the file is only a partial file and that it should not be generated into a CSS file. Sass partials are used with the **@use** rule.

SCSS Modules : You don't have to write all your Sass in a single file. You can split it up however you want with the @use rule. This rule loads another Sass file as a module, which means you can refer to its variables, mixins, and functions in your Sass file with a namespace based on the filename.

```scss
// _base.scss
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

```scss
// styles.scss
@use 'base';

.inverse {
  background-color: base.$primary-color;
  color: white;
}
```

```scss
//_variables.scss

$black: black;
```

```scss
//styles.scss
@use 'foundation/variables' as vars;

h1 {
    color: vars.$black;
}
```

Notice : we're using **@use 'base';** in the **styles.scss** file. When you use a file you don't need to include the file extension. Sass is smart and will figure it out for you.

SCSS Mixin : A mixin is one or more styles grouped together, allowing us to reuse them multiple times throughout our stylesheet without rewriting the code each time.

- To define a mixin, use the **@mixin** directive followed by a **name** and any **parameters** it may accept.
- To use (include) a mixin within a selector, use the **@include** directive followed by the **mixin** name and any required arguments.

```scss
@mixin theme($theme: DarkGray) {
  background: $theme;
  box-shadow: 0 0 1px rgba($theme, .25);
  color: #fff;
}

.info {
  @include theme;
}
.alert {
  @include theme($theme: DarkRed);
}
.success {
  @include theme($theme: DarkGreen);
}
```

```css
.info {
  background: DarkGray;
  box-shadow: 0 0 1px rgba(169, 169, 169, 0.25);
  color: #fff;
}

.alert {
  background: DarkRed;
  box-shadow: 0 0 1px rgba(139, 0, 0, 0.25);
  color: #fff;
}

.success {
  background: DarkGreen;
  box-shadow: 0 0 1px rgba(0, 100, 0, 0.25);
  color: #fff;
}
```

```
// Define a mixin for responsive styles
@mixin responsive-styles($breakpoint) {
  @media (min-width: $breakpoint) {
    @content; // Placeholder for styles to be included inside the mixin
  }
}
```

```
// Example usage of the 'responsive-styles' mixin
.button {
  background-color: #3498db;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;

  @include responsive-styles(768px) {
    padding: 12px 24px;
    font-size: 16px;
  }

  @include responsive-styles(1024px) {
    padding: 14px 28px;
    font-size: 18px;
  }
}
```

css

```
.button {
  background-color: #3498db;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

@media (min-width: 768px) {
  .button {
    padding: 12px 24px;
    font-size: 16px;
  }
}

@media (min-width: 1024px) {
  .button {
    padding: 14px 28px;
    font-size: 18px;
  }
}
```

SCSS Extend/Inheritance : Sass allows us to inherit properties from other selectors to reduce the amount of code we have to type and/or combining we have to do.

scss

```scss
/* Define a base style */
%base-button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  display: inline-block;
}

/* Extend the base style in different buttons */
.primary-button {
  @extend %base-button;
  background-color: #3498db;
  color: white;
}

.secondary-button {
  @extend %base-button;
  background-color: #2ecc71;
  color: white;
}
```

css

```css
.primary-button, .secondary-button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  display: inline-block;
}

.primary-button {
  background-color: #3498db;
  color: white;
}

.secondary-button {
  background-color: #2ecc71;
  color: white;
}
```

Using **@extend** with **Class Selectors**

scss

```scss
/* Define a base class */
.base-button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  display: inline-block;
}

/* Extend the base class in different buttons */
.primary-button {
  @extend .base-button;
  background-color: #3498db;
  color: white;
}

.secondary-button {
  @extend .base-button;
  background-color: #2ecc71;
  color: white;
}
```

css

```css
.base-button, .primary-button, .secondary-button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  display: inline-block;
}

.primary-button {
  background-color: #3498db;
  color: white;
}

.secondary-button {
  background-color: #2ecc71;
  color: white;
}
```

Using **@extend** with **Pseudo-classes** and **Pseudo-elements**

scss

```scss
/* Define a base class with pseudo-classes */
.base-button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  display: inline-block;

  &:hover {
    background-color: darken(#3498db, 10%);
  }
}

/* Extend the base class */
.primary-button {
  @extend .base-button;
  background-color: #3498db;
  color: white;
}

.secondary-button {
  @extend .base-button;
  background-color: #2ecc71;
  color: white;
}
```

CSS

```css
.base-button, .primary-button, .secondary-button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  display: inline-block;
}

.base-button:hover, .primary-button:hover, .secondary-button:hover {
  background-color: #2a7ab7; /* darken(#3498db, 10%) */
}

.primary-button {
  background-color: #3498db;
  color: white;
}

.secondary-button {
  background-color: #2ecc71;
  color: white;
}
```

SCSS Operators

Operators are one or more symbols in Sass that is special to the compiler and allows us to perform various operations within our application.

Operator	Description	Example
+	Adds operands	5 + 3 = 8
-	Subtract the second operand from the first	5 - 3 = 2
*	Multiplies the first operand with the second	5 * 3 = 15
/	Divide the first operand with the second	5 / 3 = 1
%	Modulus. Returns the remainder of a division	5 % 3 = 2

```
@use "sass:math";

.container {
  display: flex;
}

article[role="main"] {
  width: math.div(600px, 960px) * 100%;
}

aside[role="complementary"] {
  width: math.div(300px, 960px) * 100%;
  margin-left: auto;
}
```

```
.container {
  display: flex;
}

article[role=main] {
  width: 62.5%;
}

aside[role=complementary] {
  width: 31.25%;
  margin-left: auto;
}
```

scss

```scss
$base-font-size: 16px;
$line-height: 1.5;
$padding: 20px;
$margin: 10px;
$color: #3498db;
$darker-color: darken($color, 10%);
$width: 100px;
$height: $width / 2;

.container {
  font-size: $base-font-size;
  line-height: $line-height * 1em;
  padding: $padding;
  margin: $margin + 5px;
  color: $color;
  background-color: $darker-color;
  width: $width;
  height: $height;
  border: 1px solid $color;

  @media (min-width: 768px) {
    width: $width * 2;
  }

  @if $width > 50px {
    border-radius: 10px;
  }

  .nested {
    font-size: $base-font-size + 4px;
  }
}
```

css

```css
.container {
  font-size: 16px;
  line-height: 1.5em;
  padding: 20px;
  margin: 15px;
  color: #3498db;
  background-color: #2a7ab7;
  width: 100px;
  height: 50px;
  border: 1px solid #3498db;
  border-radius: 10px;
}

@media (min-width: 768px) {
  .container {
    width: 200px;
  }
}

.container .nested {
  font-size: 20px;
}
```

These conditional statements (**@if, @else if, and @else**) in SCSS provide powerful tools for making your stylesheets more dynamic and adaptable to different conditions and variables.

Structure of FLOCSS

The typical structure of a FLOCSS project might look like this:

```css
scss/
|
├── foundation/
|   ├── _reset.scss
|   ├── _variables.scss
|   ├── _mixins.scss
|   ├── _typography.scss
|   └── _base.scss
|
├── layout/
|   ├── _grid.scss
|   ├── _header.scss
|   ├── _footer.scss
|   └── _sidebar.scss
|
├── component/
|   ├── _button.scss
|   ├── _card.scss
|   ├── _form.scss
|   └── _modal.scss
|
└── main.scss
```

Tailwind CSS (Tailwind CLI - Installation)

Terminal

```
› npm install -D tailwindcss
› npx tailwindcss init
```

tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ["./src/**/*.{html,js}"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

src/input.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

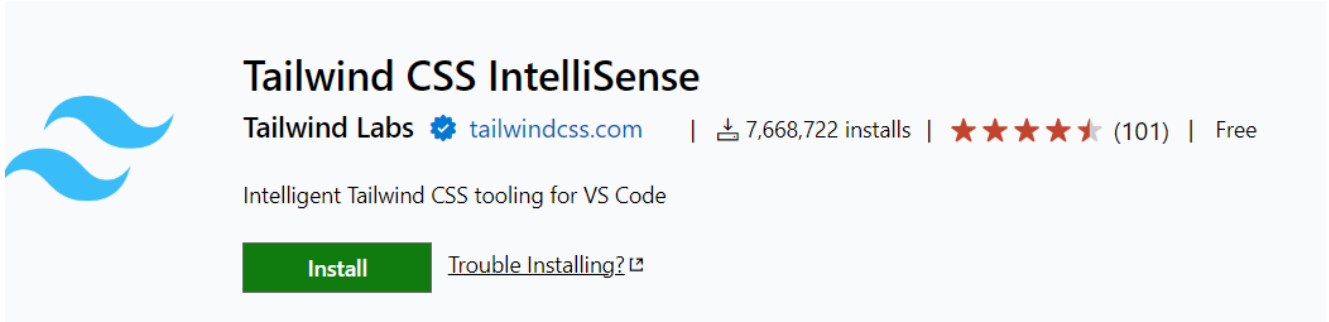
Terminal

```
› npx tailwindcss -i ./src/input.css -o ./src/output.css --watch
```

src/index.html

```
<!doctype html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale
  <link href="./output.css" rel="stylesheet">
</head>
<body>
  <h1 class="text-3xl font-bold underline">
    Hello world!
  </h1>
</body>
</html>
```

Tailwind CSS (Editor Setup)



Tailwind CSS (Optimizing for Production)

```
npx tailwindcss -i ./src/styles.css -o ./build.css --minify
```

https://heroicons.dev/?size=27

Adding Custom Styles

Overriding Default Values: If you want to override default values instead of extending them, define them without using the **extend** key.

Customizing Other Theme Properties: You can customize other aspects of the theme, such as spacing and border radius.

tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  theme: {
    screens: {
      sm: '480px',
      md: '768px',
      lg: '976px',
      xl: '1440px',
    },
    colors: {
      'blue': '#1fb6ff',
      'pink': '#ff49db',
      'orange': '#ff7849',
      'green': '#13ce66',
      'gray-dark': '#273444',
      'gray': '#8492a6',
      'gray-light': '#d3dce6',
    },
    fontFamily: {
      sans: ['Graphik', 'sans-serif'],
      serif: ['Merriweather', 'serif'],
    },
    extend: {
      spacing: {
        '128': '32rem',
        '144': '36rem',
      },
      borderRadius: {
        '4xl': '2rem',
      }
    }
  }
}
```

```
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base {
  h1 {
    @apply text-4xl font-extrabold;
  }
}

@layer components {
  .card {
    @apply bg-white p-6 shadow-md rounded-md;
  }
}

@layer utilities {
  .bg-theme {
    background-color: #3490dc;
  }
}
```

Class	Font Size	Line Height
`text-xs`	`0.75rem` (12px)	`1rem`
`text-sm`	`0.875rem` (14px)	`1.25rem`
`text-base`	`1rem` (16px)	`1.5rem`
`text-lg`	`1.125rem` (18px)	`1.75rem`
`text-xl`	`1.25rem` (20px)	`1.75rem`
`text-2xl`	`1.5rem` (24px)	`2rem`
`text-3xl`	`1.875rem` (30px)	`2.25rem`
`text-4xl`	`2.25rem` (36px)	`2.5rem`
`text-5xl`	`3rem` (48px)	`1`
`text-6xl`	`3.75rem` (60px)	`1`
`text-7xl`	`4.5rem` (72px)	`1`
`text-8xl`	`6rem` (96px)	`1`
`text-9xl`	`8rem` (128px)	`1`

Name	Class Prefix	Minimum Width
`sm`	`sm:`	`640px`
`md`	`md:`	`768px`
`lg`	`lg:`	`1024px`
`xl`	`xl:`	`1280px`
`2xl`	`2xl:`	`1536px`

Class	Font Weight Value	Description
`font-thin`	`100`	Thin
`font-extralight`	`200`	Extra Light
`font-light`	`300`	Light
`font-normal`	`400`	Normal (Regular)
`font-medium`	`500`	Medium
`font-semibold`	`600`	Semi Bold
`font-bold`	`700`	Bold
`font-extrabold`	`800`	Extra Bold
`font-black`	`900`	Black

Value	rem	px
0	0	0px
0.5	0.125rem	2px
1	0.25rem	4px
1.5	0.375rem	6px
2	0.5rem	8px
2.5	0.625rem	10px
3	0.75rem	12px
3.5	0.875rem	14px
4	1rem	16px
5	1.25rem	20px
6	1.5rem	24px
8	2rem	32px
10	2.5rem	40px

Value	rem	px
12	3rem	48px
16	4rem	64px
20	5rem	80px
24	6rem	96px
32	8rem	128px
40	10rem	160px
48	12rem	192px
56	14rem	224px
64	16rem	256px
72	18rem	288px
80	20rem	320px
96	24rem	384px

By default the spacing scale is inherited by the **padding**, **margin**, **width**, **minWidth**, **maxWidth**, **height**, **minHeight**, **maxHeight**, **gap**, **inset**, **space**, **translate**, **scrollMargin**, and **scrollPadding**

1. Layout & Spacing

p-	Padding	**p-4** for padding of 1rem
m-	Margin	**m-2** for margin of 0.5rem
px-	Horizontal padding (left and right)	**px-4**
py-	Vertical padding (top and bottom)	**py-2**
mx-	Horizontal margin	**mx-auto** for auto margin on the left and right
my-	Vertical margin	**my-4** for vertical margin of 1rem
w-	Width	**w-full** for width of 100%
h-	Height	**h-screen** for full viewport height

2. Typography

text-	Text size or color	**text-lg** for large text size, **text-red-500** for red text
font-	Font weight	**font-bold** for bold text
leading-	Line height	**leading-tight** for tight line height
tracking-	Letter spacing	**tracking-wide** for wide letter spacing

3. Flexbox & Grid

flex	Display flex	**flex** to make an element a flex container
grid	Display grid	**grid** for a grid layout
justify-	Justify content	**justify-center** to center items horizontally in flex/grid
items-	Align items	**items-center** to vertically align items in the center
gap-	Gap between flex or grid items	**gap-4** for a gap of 1rem

4. Borders

border-	Border width or color	**border-2** for 2px border width, **border-blue-500** for a blue border
rounded-	Border radius	**rounded-lg** for large rounded corners, **rounded-full** for fully circular borders
border-t-	Border for top side	**border-t-4** for 4px border on the top

5. Background

bg-	Background color or image	**bg-blue-500** for blue background, **bg-cover** for a background image that covers the container
bg-gradient-to-	Background gradient direction	**bg-gradient-to-r** for a gradient from left to right

6. Positioning

absolute, relative, fixed	Sets positioning	**absolute** to absolutely position an element
top-, right-, bottom-, left-	Set distance from sides	**top-0** for 0 distance from the top
z-	Z-index	**z-10** for a z-index of 10

7. Colors

text-	Text color	**text-red-500**
bg-	Background color	**bg-green-400**
border-	Border color	**border-yellow-300**
shadow-	Box shadow	**shadow-lg** for a large shadow

8. Sizing

w-	Width	**w-64** for a width of 16rem
h-	Height	**h-32** for a height of 8rem
min-w-	Minimum width	**min-w-full** for a minimum width of 100%
max-w-	Maximum width	**max-w-md** for a maximum width of the medium breakpoint

9. Utility Prefixes for Effects

shadow-	Box shadow	**shadow-sm**, **shadow-md**
opacity-	Opacity level	**opacity-50** for 50% opacity
transition-	CSS transitions	**transition-all** for transition on all properties
duration-	Duration of transition	**duration-500** for a 500ms transition

10. State Modifiers

hover:, foucs:, active, visited:, disabled:	State Modifiers	**hover:bg-blue-500**
group	Group Modifiers	**focus:outline-none**
sm:, md:, lg:, xl:, 2xl:	Responsive Modifiers	**sm:text-lg** applies larger text size on small screens

Styling based on parent state
group-{modifier}

```html
<div class="group p-4 bg-gray-200 rounded-md hover:bg-gray-300 transition">
  <h2 class="text-lg font-bold">Hover over me!</h2>
  <p class="mt-2 text-gray-700 group-hover:text-blue-500">
    I change color when the parent is hovered.
  </p>
  <button class="mt-4 px-4 py-2 bg-gray-500 text-white rounded hover:bg-gray-600 group-hover:bg-blue-500">
    Click me
  </button>
</div>
```

```
<div class="grid grid-cols-3 md:grid-cols-4 lg:grid-cols-6">
  <!-- ... -->
</div>
```

Responsive breakpoints
sm, md, lg, xl, 2xl

Styling direct children
***-{modifier}**

```
<div>
  <h2>Categories<h2>
  <ul class="*:rounded-full *:border *:border-sky-100 *:bg-sky-50 *:px-2 *:py-0.5 da
    <li>Sales</li>
    <li>Marketing</li>
    <li>SEO</li>
    <!-- ... -->
  </ul>
</div>
```

Pseudo-elements
before and after

```
<label class="block">
  <span class="after:content-['*'] after:ml-0.5 after:text-red-500 block text-sm font
    Email
  </span>
  <input type="email" name="email" class="mt-1 px-3 py-2 bg-white border shadow-sm b
</label>
```

```
<div class="bg-[url('/img/hero-pattern.svg')]">
  <!-- ... -->
</div>
```

Arbitrary Values
[]

What is Javascript?

Javascript is interpreted, lightweight scripting language. We can use javascript in web, mobile and desktop application development. It can use both client and server side. Well know javascript frameworks are Angular, React, VueJs, Express and Nest.

JS Writing Code

Internal JavaScript

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Inline JavaScript Example</title>
    <!-- You can also include CSS and other meta tags here -->
</head>
<body>

<h1>Inline JavaScript Example</h1>

<!-- Inline JavaScript -->
<script>
    // JavaScript code here
    alert("Hello, world!");
</script>

</body>
</html>
```

External JavaScript File

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>External JavaScript Example</title>
    <!-- You can include CSS and other meta tags here -->
    <!-- Link to external JavaScript file -->
    <script src="script.js"></script>
</head>
<body>

<h1>External JavaScript Example</h1>

<!-- Call the function defined in script.js -->
<button onclick="greet('Alice')">Greet Alice</button>

</body>
</html>
```

JS Displaying Output

1. Using console.log()

```
console.log("Hello, world!");
```

2. Alert Boxes

```
alert("Hello, world!");
```

3. Writing to the HTML Document

You can also write output directly to the HTML document using various methods, such as **document.write()**, modifying the DOM elements, or setting the **innerHTML** property of an element.

```
<script>
    function displayOutput() {
        // Get the element where you want to display the output
        let outputElement = document.getElementById("output");

        // Set innerHTML of the element to display output
        outputElement.innerHTML = "Hello, world!";
    }
</script>
```

```
document.write("Hello, world!");
```

JS Variables

- In JavaScript, variables are containers for **storing** data **values**.
- In Javascript we can declare variable with "**var**", "**let**" and "**const**" keywords.
- You can denote "**var**" is functional scope and it can also **re-assign** and **re-declare**. Variable re-declaration can also effect to depended function and condition, so you should reconsider and check carefully before you do.
- "**let**" is block scope and you can **re-assign** but **re-declaration** doesn't allow. It's **good** to use for common case.
- "**const**" also block scope. Different between "const" and "let" is you can't **re-assign** to "**const**" variable. If you don't wish to re-assign your variable value, you must use "const"

	Re-Assign	Re-Declare
Var	TRUE	TRUE
Let	TRUE	FALSE
Const	FALSE	FALSE

Variable Naming Rules

- Variables must begin with a letter (a-z, A-Z), underscore (_), or dollar sign ($).
- Subsequent characters can also include numbers (0-9).
- JavaScript is case-sensitive (name and Name are different variables).

JS Data Types

JavaScript has several data types that you can use to store and manipulate different kinds of data. These data types can be categorized into **primitive** and **non-primitive** (or reference) types.

Primitive Data Types

Primitive data types are **immutable** and are directly assigned values.

1. String

- Used for **text**.
- Enclosed in single **quotes** (' '), **double quotes** (" "), or **backticks** (` `) for template literals.

```
let name = "Alice";
let greeting = 'Hello';
let message = `Hi, ${name}!`;
```

2. Number

Represents both **integer** and **floating-point numbers**.

```
let age = 30;
let price = 19.99;
```

3. Boolean

Represents **true** or **false**.

```
let isActive = true;
let isComplete = false;
```

4. Null

Represents the intentional **absence** of any object value.

```
let emptyValue = null;
```

5. Undefined

Represents a variable that has been declared but **not yet assigned a value**.

```
let notAssigned;
console.log(notAssigned); // Output: undefined
```

Non-Primitive (Reference) Data Types

Non-primitive data types are mutable and are accessed by reference.

1. Object

Represents a collection of **key-value pairs**.

```
let person = {
  name: 'Alice',
  age: 30
};
```

2. Array

Represents an ordered list of **values**.

```
let numbers = [1, 2, 3, 4, 5];
```

3. Function

Represents a **block of code** designed to perform a particular task.

```
function greet(name) {
  return `Hello, ${name}!`;
}
```

Checking Data Types: You can check the type of a variable using the **typeof** operator.

```
console.log(typeof name); // Output: string
```

Type Conversion: You can convert values from one type to another.

Implicit Conversion: JavaScript sometimes automatically converts types.

```
let result = '5' + 10; // '510' (string concatenation)
let sum = '5' - 2; // 3 (string converted to number)
```

Explicit Conversion: You can manually convert types using functions like **String()**, **Number()**, **Boolean()**, etc.

```
let num = 42;
let str = String(num); // "42"

let strNum = '123';
let int = Number(strNum); // 123

let truthy = 1;
let bool = Boolean(truthy); // true
```

String Method	Description
length	Returns the **length** of the string.
charAt(index**)**	Returns the **character** at the specified index.
indexOf(value**)**	Returns the **index** of the first occurrence of the specified value.
lastIndexOf(value**)**	Returns the **index** of the last occurrence of the specified value.
includes(value**)**	Checks if the string contains the specified value. Returns **true or false**.
substring(start, end**)**	Extracts **characters** from the string between the start and end indices. [substring(0, 5) extracts characters starting from index 0 and up to, but not including, index 5]
slice(start, end**)**	Extracts a section of the string and returns it as a **new string**.
substr(start, length**)**	Extracts parts of the **string**, starting at the specified position and for the specified number of characters.
toUpperCase()	Converts the string to uppercase letters.
toLowerCase()	Converts the string to lowercase letters.
trim()	Removes whitespace from **both sides** of the string.
replace(searchValue, newValue**)**	Replaces the specified value with another value.
split(separator**)**	Splits the string into an array of substrings.
concat(...strings**)**	Joins two or more strings.
repeat(count**)**	Returns a new string with a specified number of copies of the string it was called on.
startsWith(value**)**	Checks if the string starts with the specified value.

```javascript
let str = "JavaScript is fun!";

// length
console.log(str.length); // Output: 17

// charAt()
console.log(str.charAt(0)); // Output: J

// indexOf()
console.log(str.indexOf("is")); // Output: 11

// lastIndexOf()
console.log(str.lastIndexOf("a")); // Output: 3

// includes()
console.log(str.includes("fun")); // Output: true

// substring()
console.log(str.substring(0, 10)); // Output: JavaScript

// slice()
console.log(str.slice(0, 10)); // Output: JavaScript
console.log(str.slice(-4)); // Output: fun!

// substr()
console.log(str.substr(11, 3)); // Output: fun

// toUpperCase()
console.log(str.toUpperCase()); // Output: JAVASCRIPT IS FUN!

// toLowerCase()
console.log(str.toLowerCase()); // Output: javascript is fun!
```

```javascript
// trim()
let strWithSpaces = "   JavaScript is fun!   ";
console.log(strWithSpaces.trim()); // Output: JavaScript is fun!

// replace()
console.log(str.replace("fun", "awesome")); // Output: JavaScript is awesome!

// split()
console.log(str.split(" ")); // Output: ["JavaScript", "is", "fun!"]

// concat()
let str1 = "Hello";
let str2 = "World";
console.log(str1.concat(", ", str2, "!")); // Output: Hello, World!

// repeat()
console.log(str1.repeat(3)); // Output: HelloHelloHello

// startsWith()
console.log(str.startsWith("JavaScript")); // Output: true

// endsWith()
console.log(str.endsWith("fun!")); // Output: true
```

Number Method	Description
toString()	Converts the number to a string.
toFixed(digits)	Formats the number to a specified number of decimal places.
toExponential(digits)	Converts the number to exponential notation with a specified number of decimal places.
toPrecision(digits)	Formats the number to a specified length.
valueOf()	Returns the primitive value of the number object.
Number.isInteger(value)	Checks if the value is an integer. Returns true or false.
Number.isNaN(value)	Checks if the value is NaN (Not-a-Number). Returns true or false.
Number.isFinite(value)	Checks if the value is a finite number. Returns true or false.
Number.parseFloat(value)	Parses a string and returns a floating-point number.
Number.parseInt(value, radix)	Parses a string and returns an integer, with an optional radix (base).

```javascript
let num = 123.456;

// toString()
console.log(num.toString()); // Output: "123.456"

// toFixed()
console.log(num.toFixed(2)); // Output: "123.46"
console.log(num.toFixed(0)); // Output: "123"

// toExponential()
console.log(num.toExponential(2)); // Output: "1.23e+2"
console.log(num.toExponential(4)); // Output: "1.2346e+2"

// toPrecision()
console.log(num.toPrecision(4)); // Output: "123.5"
console.log(num.toPrecision(6)); // Output: "123.456"

// valueOf()
let numObj = new Number(456);
console.log(numObj.valueOf()); // Output: 456

// Number.isInteger()
console.log(Number.isInteger(123)); // Output: true
console.log(Number.isInteger(123.456)); // Output: false

// Number.isNaN()
console.log(Number.isNaN(NaN)); // Output: true
console.log(Number.isNaN(123)); // Output: false

// Number.isFinite()
console.log(Number.isFinite(123)); // Output: true
console.log(Number.isFinite(Infinity)); // Output: false
```

```javascript
// Number.parseFloat()
console.log(Number.parseFloat("123.456")); // Output: 123.456
console.log(Number.parseFloat("123abc")); // Output: 123

// Number.parseInt()
console.log(Number.parseInt("123.456")); // Output: 123
console.log(Number.parseInt("123abc")); // Output: 123
console.log(Number.parseInt("101", 2)); // Output: 5 (parses "101" as a binary number)
```

Array Method	Description
push(element)	Adds one or more elements to the end of an array, and returns the new length of the array.
pop()	Removes the last element from an array and returns that element.
shift()	Removes the first element from an array and returns that element.
unshift(element)	Adds one or more elements to the beginning of an array, and returns the new length of the array.
concat(array)	Returns a new array by merging (concatenating) existing arrays or values.
join(separator)	Joins all elements of an array into a string.
slice(start, end)	Extracts a section of an array and returns a new array.
splice(start, deleteCount, item1, item2, ...)	Changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.
indexOf(element)	Returns the first index at which a given element can be found in the array, or -1 if it is not present.
lastIndexOf(element)	Returns the last index at which a given element can be found in the array, or -1 if it is not present.
forEach(callback)	Executes a provided function once for each array element.
map(callback)	Creates a new array populated with the results of calling a provided function on every element in the calling array.
filter(callback)	Creates a new array with all elements that pass the test implemented by the provided function.

Array Method	Description
reduce(callback, initialValue)	Applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value.
reduceRight(callback, initialValue)	Applies a function against an accumulator and each element in the array (from right to left) to reduce it to a single value.
some(callback)	Checks if at least one element in the array passes a test implemented by the provided function.
every(callback)	Checks if all elements in the array pass a test implemented by the provided function.
find(callback)	Returns the value of the first element in the array that satisfies the provided testing function. Otherwise, undefined is returned.
findIndex(callback)	Returns the index of the first element in the array that satisfies the provided testing function. Otherwise, -1 is returned.
sort(compareFunction)	Sorts the elements of an array in place and returns the sorted array.
reverse()	Reverses the order of the elements of an array in place.
includes(element)	Determines whether an array includes a certain element, returning true or false as appropriate.
isArray(value)	Checks if a value is an array. Returns true or false.

```javascript
let numbers = [1, 2, 3, 4, 5];

// push()
numbers.push(6);
console.log(numbers); // Output: [1, 2, 3, 4, 5, 6]

// pop()
let lastElement = numbers.pop();
console.log(lastElement); // Output: 6
console.log(numbers); // Output: [1, 2, 3, 4, 5]

// shift()
let firstElement = numbers.shift();
console.log(firstElement); // Output: 1
console.log(numbers); // Output: [2, 3, 4, 5]

// unshift()
numbers.unshift(0);
console.log(numbers); // Output: [0, 2, 3, 4, 5]

// concat()
let moreNumbers = [6, 7, 8];
let mergedArray = numbers.concat(moreNumbers);
console.log(mergedArray); // Output: [0, 2, 3, 4, 5, 6, 7, 8]

// join()
console.log(numbers.join(", ")); // Output: "0, 2, 3, 4, 5"

// slice()
let slicedArray = numbers.slice(1, 4);
console.log(slicedArray); // Output: [2, 3, 4]

// splice()
numbers.splice(2, 0, 1.5); // Insert 1.5 at index 2
console.log(numbers); // Output: [0, 2, 1.5, 3, 4, 5]

// indexOf()
console.log(numbers.indexOf(3)); // Output: 3

// forEach()
numbers.forEach(function(element) {
console.log(element * 2);
});
// Output:
// 0
// 4
// 3
// 6
// 8
// 10

// map()
let doubledNumbers = numbers.map(function(element) {
return element * 2;
});
console.log(doubledNumbers); // Output: [0, 4, 3, 6, 8, 10]

// filter()
let filteredNumbers = numbers.filter(function(element) {
return element > 3;
});
console.log(filteredNumbers); // Output: [4, 5]

// reduce()
let sum = numbers.reduce(function(accumulator, currentValue) {
return accumulator + currentValue;
}, 0);
console.log(sum); // Output: 15

// sort()
numbers.sort();
console.log(numbers); // Output: [0, 1.5, 2, 3, 4, 5]

// reverse()
numbers.reverse();
console.log(numbers); // Output: [5, 4, 3, 2, 1.5, 0]

// includes()
console.log(numbers.includes(3)); // Output: true
console.log(numbers.includes(6)); // Output: false

// isArray()
console.log(Array.isArray(numbers)); // Output: true
console.log(Array.isArray(mergedArray)); // Output: true
console.log(Array.isArray("Hello")); // Output: false
```

JS DOM

DOM stands for Document Object Model. It is a programming interface for web documents that provides a structured representation of the HTML document as a tree-like structure.

Definition and Purpose

Document: Refers to the **HTML document** loaded in the browser.

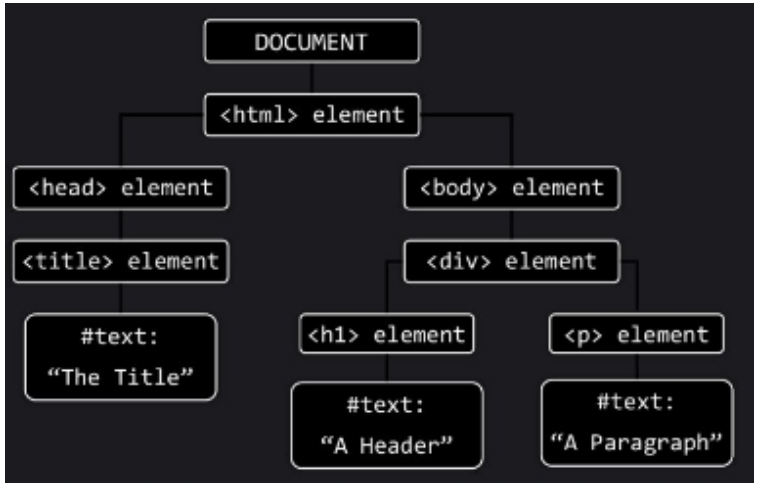
Object: Each **HTML element** (like <body>, <p>, , etc.) in the document is represented as an object in the DOM tree.

Model: The DOM represents the document as a **hierarchical structure of objects** that can be manipulated and interacted with using scripting languages like JavaScript.

Why DOM is Important

- **Interactive Web Pages**: The DOM enables dynamic interaction with web pages. It allows developers to respond to **user actions** (like clicks and keystrokes) and **update the content or style of elements** accordingly.
- **Dynamic Content**: Websites can dynamically **update content without reloading** the entire page, making web applications more responsive and interactive.
- **Cross-Browser Compatibility**: The DOM provides a standardized way to access and manipulate web documents, ensuring consistency across different web browsers.

DOM Tree




```
<!DOCTYPE html>
<html>
<head>
  <title> The Title </title>
</head>
<body>
  <h1> A Header </h1>
  <p> A Paragraph </p>
</body>
</html>
```

Node Relationships

The nodes in the node tree have a hierarchical relationship to each other. The terms **parent**, **child**, and **sibling** are used to describe the relationships.

- <html> is the root or top node meaning it has no parents.
- <html> is the parent of <head> and <body>
- <head> is the first child of <html>. <body> is the last child of <html>
- <head> has one child:<title>
- <body> has two children: <h1> and <p>
- <h1> is the first child of <body>. <p> is the last child of <body>
- <head> and <body> are siblings.
- <h1> and <p> are siblings.

Note: elements are nodes. Additionally text, attributes and comments are considered as nodes.

JS DOM Selectors

With JavaScript, we can select any elment from the DOM tree using document object.

- To get their content.
- To change their content.
- To style them.
- To get or change their attributes.
- To remove them.

1. Select the Topmost Elements

- document.documentElement selects <html>
- document.head selects <head>
- Document.body selects <body>

2. Select an Element By ID (getElementById)

```
let element = document.getElementById("myElement");
element.innerHTML = "New Content";
```

3. **Select Elements By Class Name** (getElementsByClassName)

- Selects all elements with the given class name.
- Returns an array of objects.

```
let elements = document.getElementsByClassName("myClass");
for (let i = 0; i < elements.length; i++) {
  elements[i].style.color = "blue";
}
```

4. **Select Elements By Tag Name** (getElementsByTagName)

- Selects all elements with the given tag name.
- Returns an array of objects.

```
let elements = document.getElementsByTagName("p");
for (let i = 0; i < elements.length; i++) {
  elements[i].style.color = "blue";
}
```

5. **Select Elements By Name** (getElementsByName)

- Selects all elements with the given attribute name.
- Returns an array of objects.

```
let elements = document.getElementsByName("name");
```

6. Select Elements Using CSS Selectors (querySelectorAll)

- Selects all elements that match the specified **selector**
- Returns an array of objects.

```
let usernameInputs = document.querySelectorAll('input[name="username"]');
usernameInputs.forEach(input => {
  console.log(input.value);  // Outputs: JohnDoe, JaneDoe
});
```

7. Select Elements Using CSS Selector (querySelector)

- You can use CSS attribute selectors within **querySelector** to select the **first element** with a specific input name.

```
let usernameInput = document.querySelector('input[name="username"]');
console.log(usernameInput.value);  // Outputs: JohnDoe
```

Navigating Element Nodes

The following document object properties are used to navigate **nodes**:

- parentNode
- firstChild
- lastChild
- childNodes[index]
- previousSibling
- nextSibling

The properties mentioned above will navigate not just **element nodes**, but also **text** and **comment nodes**.

The following document object properties are used to navigate **element nodes**:

- parentElement
- firstElementChild
- lastElementChild
- children[index]
- previousElementSibling
- nextElementSibling

JS DOM HTML

1. Getting and replacing element's content (innerHTML)

Getting: returns the content of an HTML element.

Replacing: to replace the content of and HTML element.

```
<div id="content">Original Content</div>

<script>
let contentDiv = document.getElementById('content');
console.log(contentDiv.innerHTML);  // Outputs: Original Content

contentDiv.innerHTML = '<p>New <strong>HTML</strong> Content</p>';
console.log(contentDiv.innerHTML);  // Outputs: <p>New <strong>HTML</strong> Content</p>
</script>
```

2. Getting and replacing element's content (textContent)

```
<div id="content"><p>Original <strong>HTML</strong> Content</p></div>

<script>
let contentDiv = document.getElementById('content');
console.log(contentDiv.textContent);  // Outputs: Original HTML Content

contentDiv.textContent = 'New Text Content';
console.log(contentDiv.textContent);  // Outputs: New Text Content
</script>
```

3. Getting and replacing element's content (innerText)

```
contentDiv.innerText = 'The text content has been changed!';
```

4. Creating Element Nodes (createElement)

The **createElement** method creates a new **HTML element** specified by its **tag name**. You can then set its attributes, properties, and append it to the DOM.

```
let newElement = document.createElement(tagName);
```

```
<div id="container"></div>

<script>
let container = document.getElementById('container');

// Create a new paragraph element
let newParagraph = document.createElement('p');
newParagraph.textContent = 'This is a new paragraph.';

// Append the new paragraph to the container div
container.appendChild(newParagraph);
</script>
```

Creating Text Node (createTextNode)

The **createTextNode** method creates a **new text node** containing the **specified text**. This text node can then be appended to an element.

```
let newText = document.createTextNode(text);
```

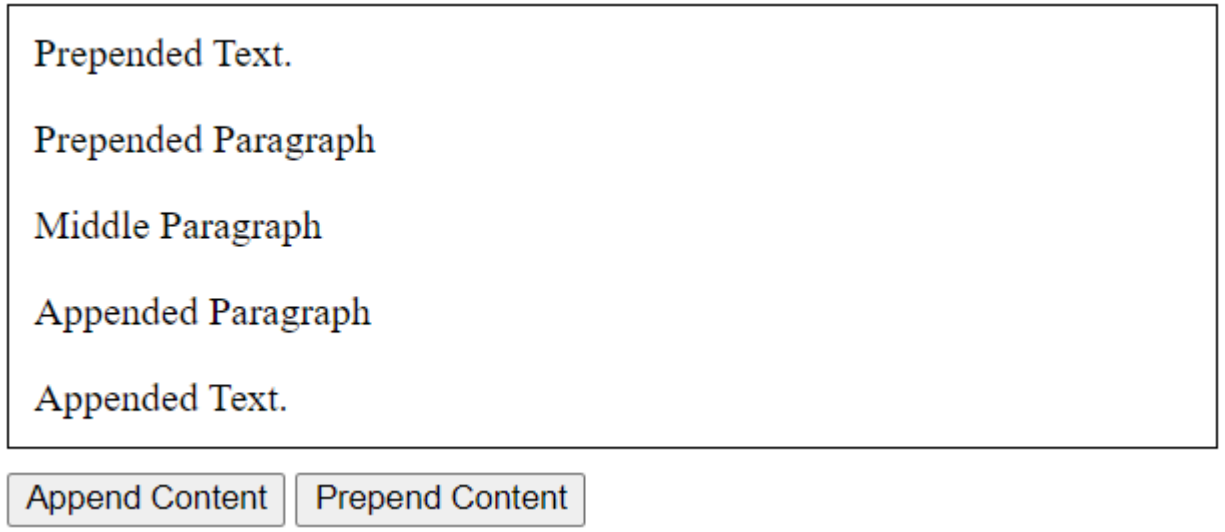
```
<div id="container"></div>

<script>
let container = document.getElementById('container');

// Create a new text node
let newText = document.createTextNode('This is a new text node.');

// Append the new text node to the container div
container.appendChild(newText);
</script>
```

Inserting to Elements (prepend, append)

In JavaScript, the **append** and **prepend** methods are used to insert **elements** or **text nodes** at specific positions within a parent element.

- The **append** method inserts nodes or strings at the **end** of a parent element.
- The **prepend** method inserts nodes or strings at the **beginning** of a parent element.

```
parentElement.append(...nodesOrDOMStrings);

parentElement.prepend(...nodesOrDOMStrings);
```




```
<div id="container">
  <p>Middle Paragraph</p>
</div>

<button id="appendButton">Append Content</button>
<button id="prependButton">Prepend Content</button>

<script>
let container = document.getElementById('container');
let appendButton = document.getElementById('appendButton');
let prependButton = document.getElementById('prependButton');

appendButton.addEventListener('click', function() {
  let newParagraph = document.createElement('p');
  newParagraph.textContent = 'Appended Paragraph';
  container.append(newParagraph, ' Appended Text.');
});

prependButton.addEventListener('click', function() {
  let newParagraph = document.createElement('p');
  newParagraph.textContent = 'Prepended Paragraph';
  container.prepend('Prepended Text. ', newParagraph);
});
</script>
```

Removing Elements (remove)

To remove an element, use the **node.remove()** method.

```
removeButton.addEventListener('click', function() {
  // Find all dynamically added elements
  let dynamicElements = container.querySelectorAll('.dynamic');

  // If there are dynamic elements, remove the last one
  if (dynamicElements.length > 0) {
    dynamicElements[dynamicElements.length - 1].remove();
  }
});
```

Getting Attribute Value (getAttribute)

To remove an element, use the **node.remove()** method.

```
element.getAttribute(attributeName);
```

Setting Attribute Value (setAttribute)

The **setAttribute** method is used to **add a new attribute** or **change the value** of an existing attribute on an element.

```
element.setAttribute(attributeName, attributeValue);
```

JS DOM CSS or Styling

JavaScript allows you to dynamically manipulate the **styles** of **HTML elements** using the **DOM**. This can be done in several ways, including modifying the **style property** of an element, using **CSS classes.**

Styling Elements With DOM

```
element.style.propertyName = "value";
```

```
document.getElementById("myElement").style.backgroundColor = "lightblue";
document.getElementById("myElement").style.border = "2px solid red";
```

Adding and Removing CSS Classes

```
element.classList.add("className");
element.classList.remove("className");
element.classList.toggle(className);
```

```
document.getElementById("myElement").classList.add("highlight");
document.getElementById("myElement").classList.remove("highlight");
document.getElementById("myElement").classList.toggle('highlight');
```

Using cssText Property

```
element.style.cssText = "property1: value1; property2: value2; ...";
```

```
document.getElementById("myElement").style.cssText = "background-color: pink; border: 4px solid purple;";
```

Note: template literals are enclosed by backticks (`) instead of single or double quotes.

JS DOM Events

With JavaScript and DOM, we can listen to events that happen to elemnts.

1. Inline Event Handlers

```
<element event="eventHandler()"> ... </element>
```

```
<button onclick="alert('Button clicked!')">Click Me</button>
```

2. Event Handler Properties

```
element.event = function() { ... };
```

```
<button id="myButton">Click Me</button>

<script>
  document.getElementById('myButton').onclick = function() {
    alert('Button clicked!');
  };
</script>
```

3. addEventListener Method

```
element.addEventListener('event', function(event) { ... });
```

```
<button id="myButton">Click Me</button>

<script>
  document.getElementById('myButton').addEventListener('click', function() {
    alert('Button clicked!');
  });
</script>
```

Used for...of : Replaced **for...in** with **for...of** for looping over the **HTMLCollection**. This ensures that only the child elements are processed, and not any non-indexed properties.

Using Array.from(): This converts the **HTMLCollection** to an array, allowing you to use array methods and simplifying your code.

Mouse Events: Triggered by mouse actions.

click: Triggered when an element is clicked.

dblclick: Triggered on a double-click.

mouseover: Triggered when the mouse hovers over an element.

mouseout: Triggered when the mouse leaves an element.

mousemove: Triggered when the mouse moves within an element.

Keyboard Events: Triggered by keyboard actions.

keydown: Triggered when a key is pressed down.

keyup: Triggered when a key is released.

keypress: Triggered when a key is pressed and released (deprecated).

Form Events: Triggered by form actions.

submit: Triggered when a form is submitted.

change: Triggered when the value of an input changes.

focus: Triggered when an input gains focus.

blur: Triggered when an input loses focus.

Window Events: Triggered by actions on the browser window.

load: Triggered when the entire page is loaded.

resize: Triggered when the window is resized.

scroll: Triggered when the user scrolls the page.

Touch Events (for mobile devices):

touchstart: Triggered when a touch point is placed on the touch surface.

touchmove: Triggered when a touch point is moved along the touch surface.

touchend: Triggered when a touch point is removed from the touch surface.