

**Structure of an HTML Document:**

An HTML document follows a standard structure.

<!DOCTYPE html> defines the document type as HTML5.

<html> is the root element that wraps all content.

<head> contains metadata like title, links, and styles.

<body> holds the visible content of the webpage.

**Basic HTML Tags & Syntax:**

<h1> to <h6> define headings (h1 is the largest, h6 is the smallest).

<p> is used for paragraphs.

<br> adds a line break.

<hr> creates a horizontal separator.

<strong> makes text bold.

<em> italicizes text.

<small> reduces text size.

<mark> highlights text with a yellow background.

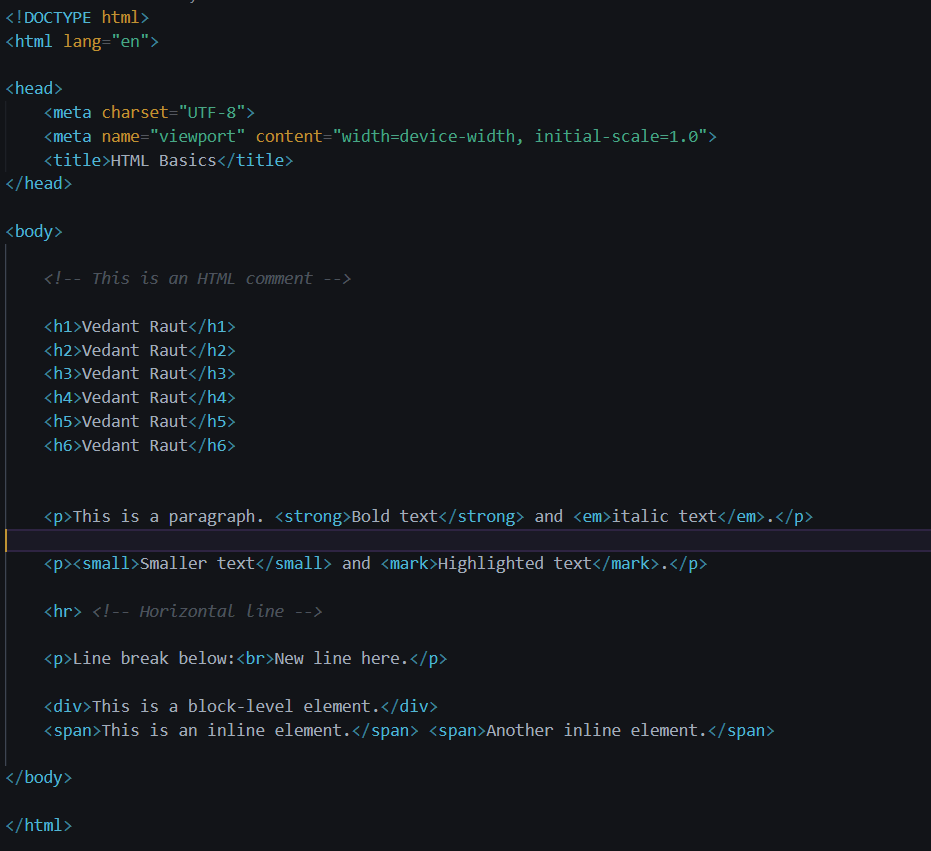
**HTML Comments:**

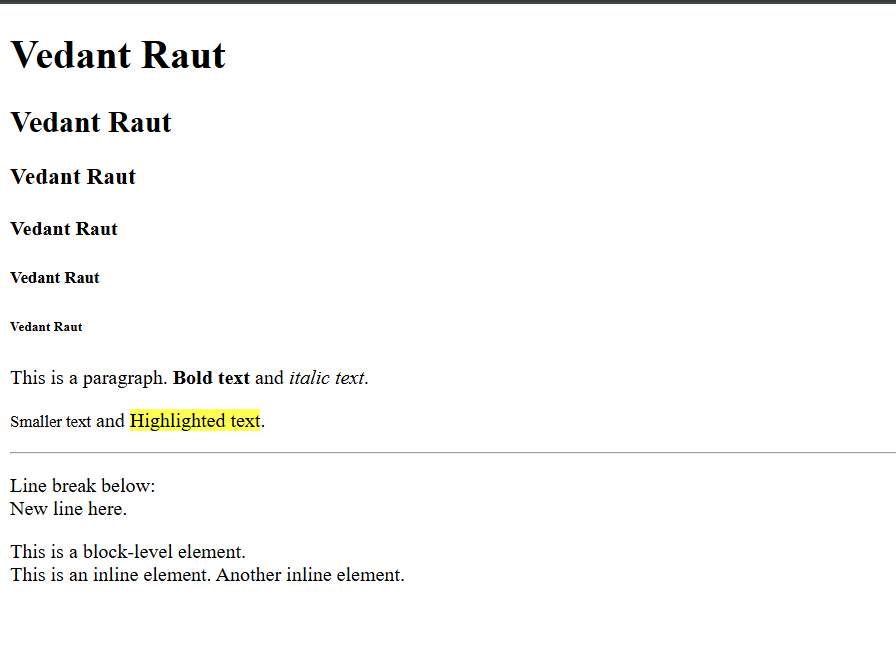
Use <!-- Comment --> to add comments in the code. These are ignored by the browser but help in documentation.

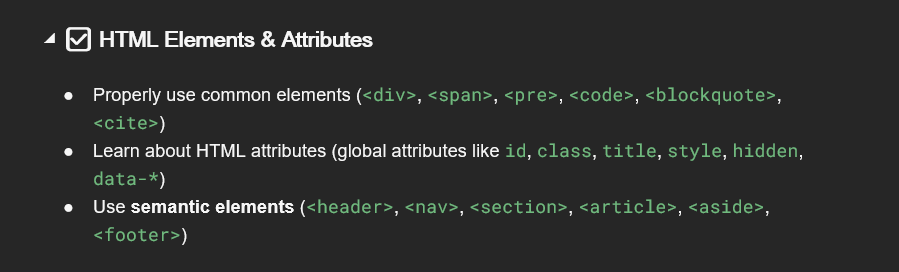
**Block-Level vs Inline Elements:**

Block-level elements take up the full width of the parent container and start on a new line, like <div>, <p>, and <h1>.

Inline elements take only as much width as needed and don’t start on a new line, like <span>, <a>, and <strong>.







**HTML Elements & Attributes:**

**HTML Elements & Attributes**

**Common HTML Elements:**

<div> is a generic container for grouping elements.  
<span> is an inline container used for styling specific parts of text.  
<pre> preserves whitespace and displays text exactly as written in the HTML.  
<code> is used to represent programming code snippets.  
<blockquote> is used for quoting external content, usually displayed with indentation.  
<cite> represents a reference to a creative work, such as a book, article, or website.

**HTML Attributes:**

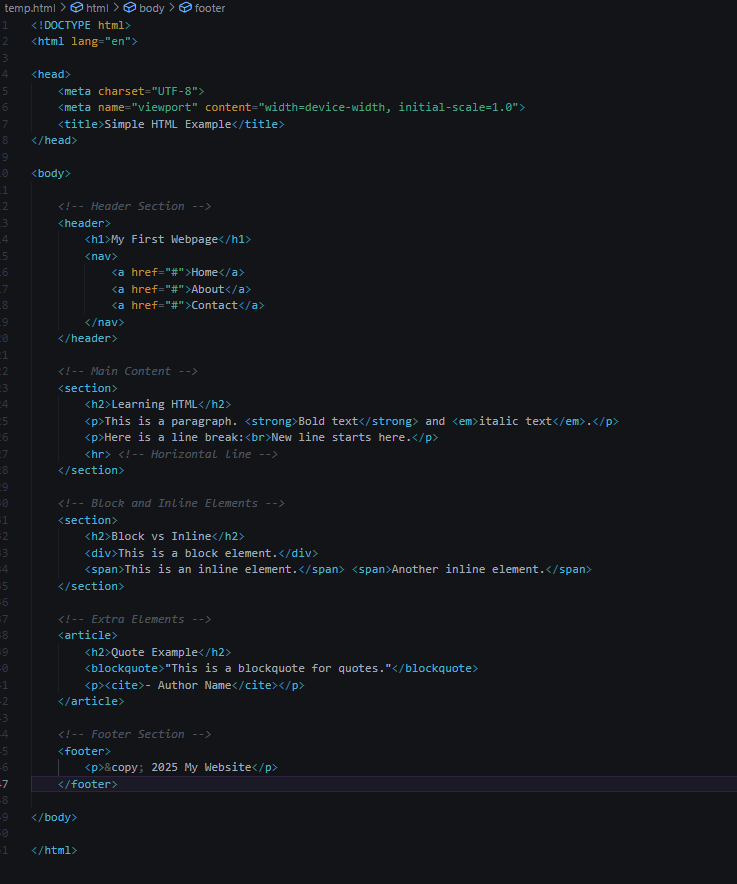
Attributes provide additional information about an element and are written inside the opening tag.

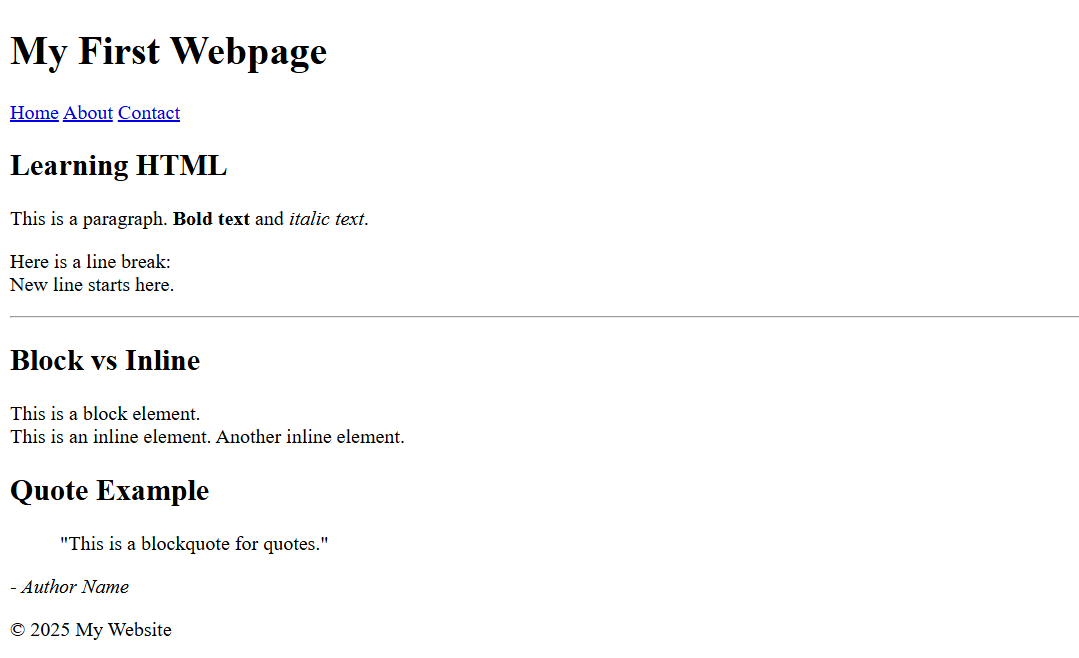
id assigns a unique identifier to an element.  
class groups multiple elements with the same styles or behaviors.  
title provides extra information, often shown as a tooltip on hover.  
style allows inline CSS styling for an element.  
hidden hides an element from display but keeps it in the HTML structure.  
data-\* stores custom data attributes that can be used in JavaScript for interactivity.

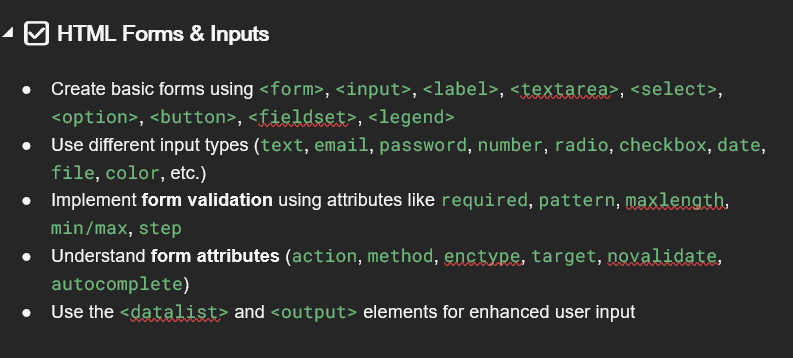
**Semantic Elements:**

Semantic elements give meaning to web content, making it more readable and SEO-friendly.

<header> represents the top section of a webpage or a section.  
<nav> contains navigation links for the website.  
<section> groups related content within a page.  
<article> represents self-contained content, like blog posts or news articles.  
<aside> contains additional information, like sidebars or advertisements.  
<footer> is used for the bottom section of a webpage, often containing copyright or contact info.







**HTML Forms & Inputs**

**Basic Form Elements:**

<form> defines a form for user input.  
<input> is used to take user input in different formats.  
<label> is used to describe form inputs.  
<textarea> allows users to enter multi-line text.  
<select> creates a dropdown menu with <option> elements.  
<button> adds a clickable button.  
<fieldset> groups related form fields, and <legend> provides a title for the group.

**Input Types:**

Text fields → <input type="text">  
Email → <input type="email">  
Password → <input type="password">  
Number → <input type="number">  
Radio buttons → <input type="radio">  
Checkbox → <input type="checkbox">  
Date picker → <input type="date">  
File upload → <input type="file">  
Color picker → <input type="color">

**Form Validation Attributes:**

required makes a field mandatory.  
pattern ensures input matches a regex pattern.  
maxlength limits character count.  
min and max set numeric range limits.  
step controls numeric input increments.

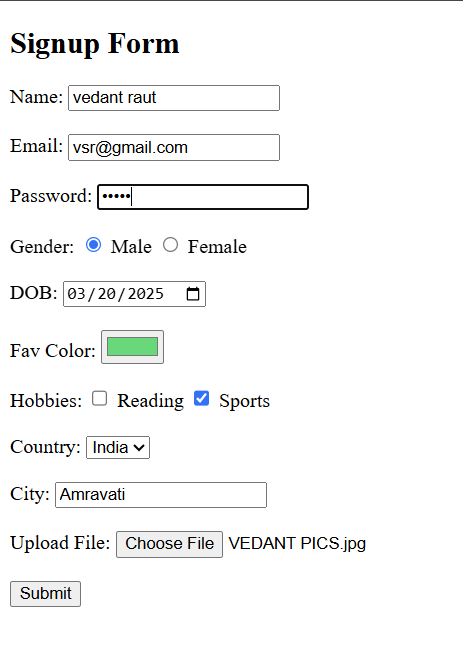
**Form Attributes:**

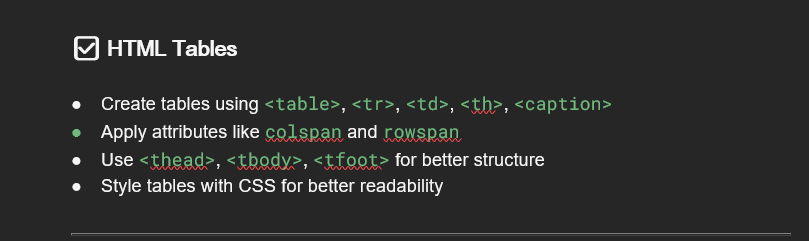
action defines the URL where form data is sent.  
method specifies HTTP method (GET or POST).  
enctype specifies encoding for file uploads.  
target decides where to display the response.  
novalidate disables form validation.  
autocomplete suggests previously entered values.

**Enhanced Inputs:**

<datalist> provides autocomplete suggestions for an input field.  
<output> displays calculated or dynamic results.







HTML tables are used to display data in rows and columns. We use <table> to create a table, <tr> for table rows, <td> for table data, and <th> for table headers.

We can merge columns using colspan and merge rows using rowspan. For better structure, we use <thead>, <tbody>, and <tfoot>

We use <table> to create a table.

<tr> is used for table rows.

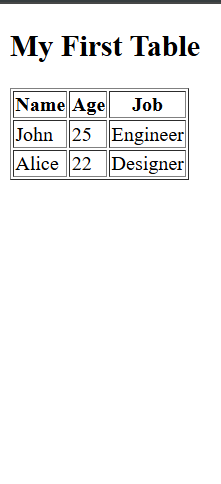
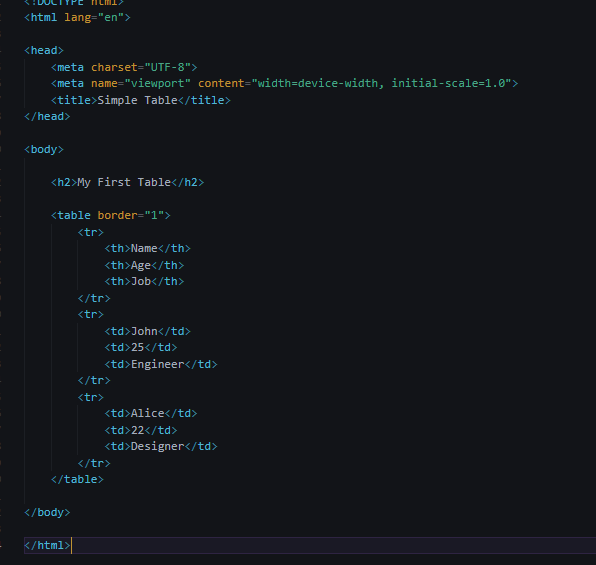
<td> is used for table data (cells).

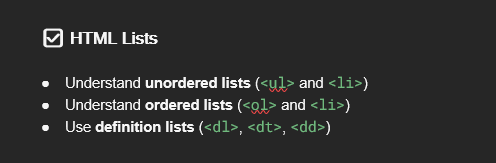
<th> is used for table headers (bold and centered by default).

We can merge columns using colspan.

We can merge rows using rowspan.

For better structure, we use <thead>, <tbody>, and <tfoot>.

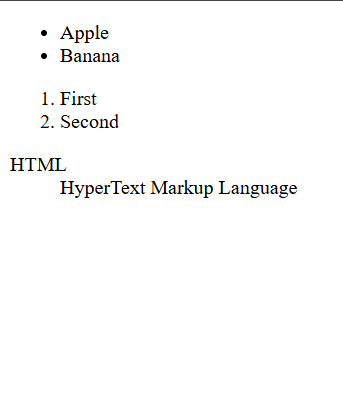
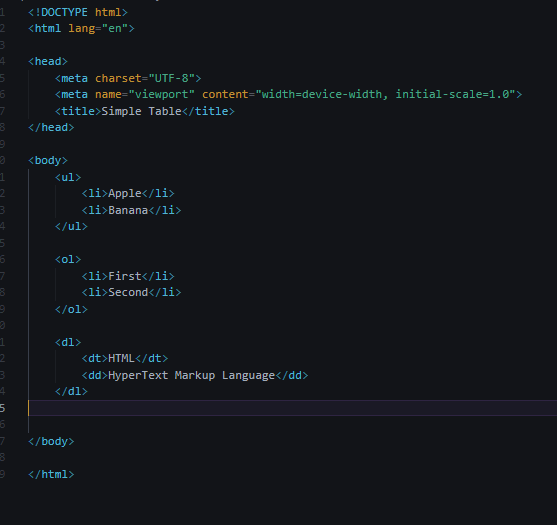


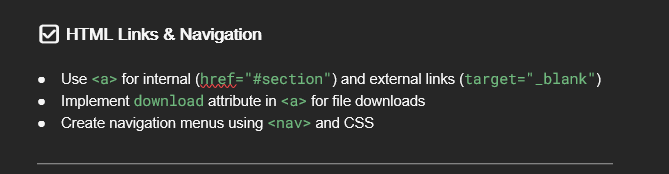


**HTML Lists**

Lists are used to organize content in a structured way.

* <ul> creates an **unordered list** (bullets).
* <ol> creates an **ordered list** (numbers).
* <dl> creates a **definition list** with <dt> for the term and <dd> for the definition.

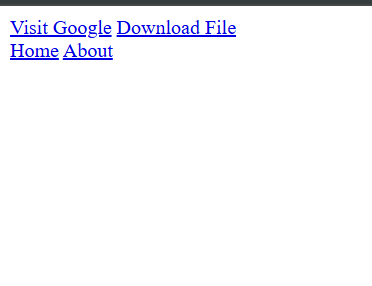
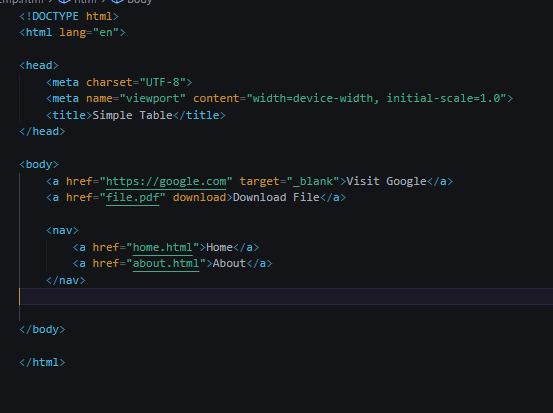


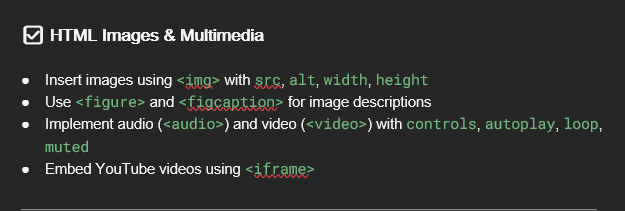


**HTML Links & Navigation**

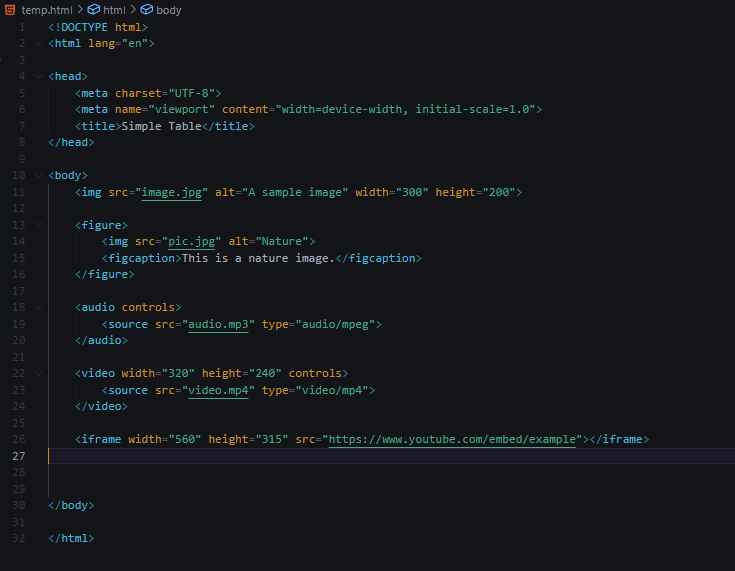
Links let users navigate between pages.

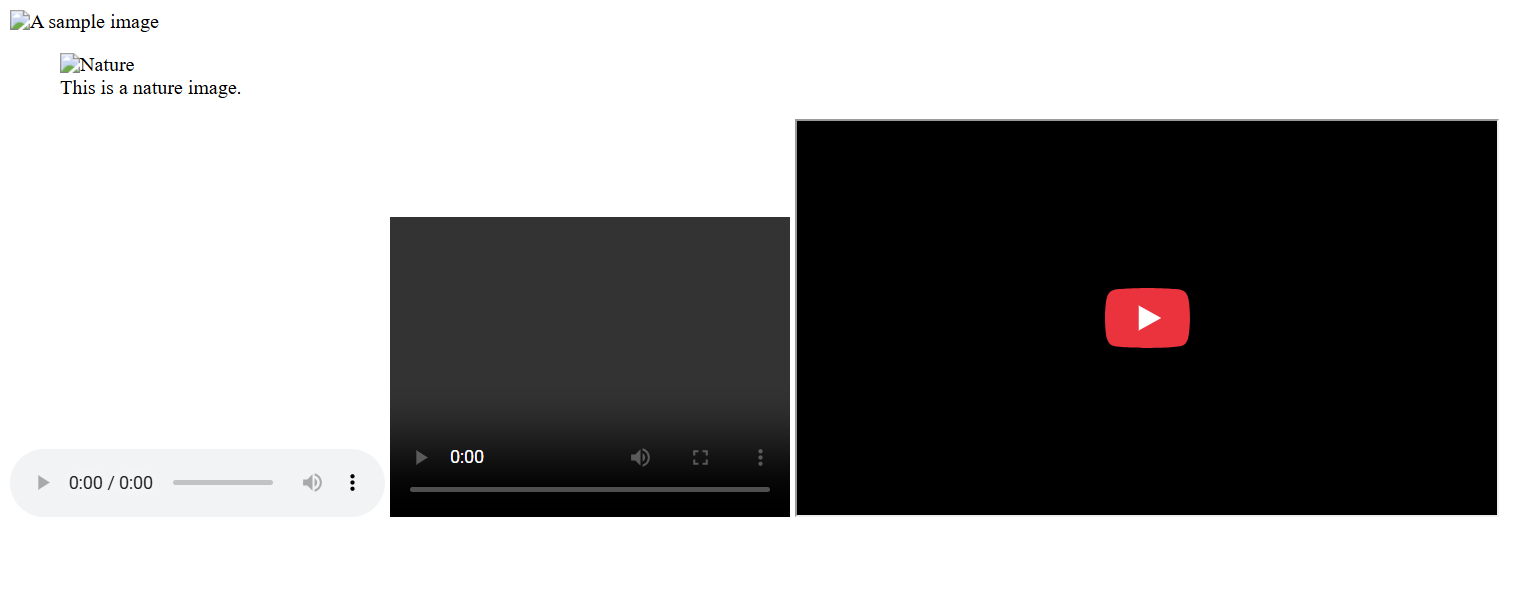
* <a href="URL"> creates a link.
* target="\_blank" opens links in a new tab.
* download allows file downloads.
* <nav> is used for navigation menus.



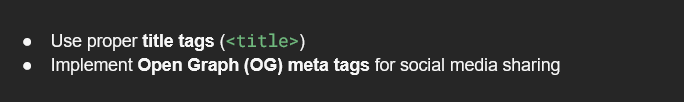
  
**HTML Images & Multimedia**

* <img> adds images using src (source) and alt (description).
* <figure> with <figcaption> provides image descriptions.
* <audio> and <video> are used for media with controls, autoplay, loop, and muted.
* <iframe> embeds external content like YouTube videos.





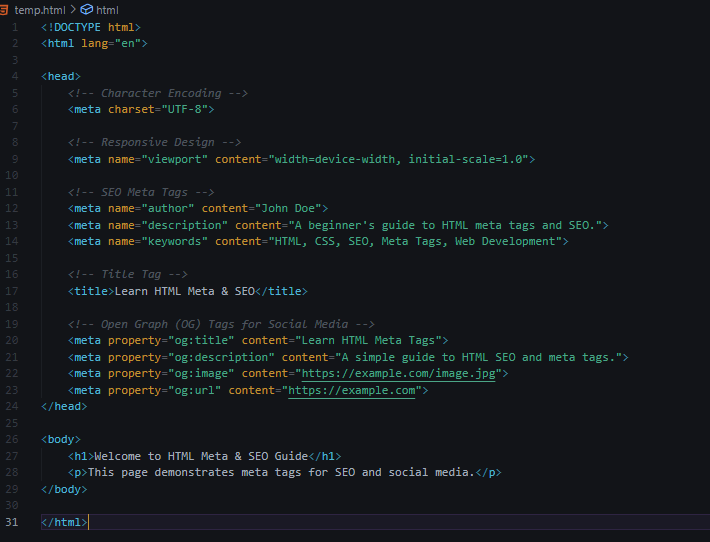


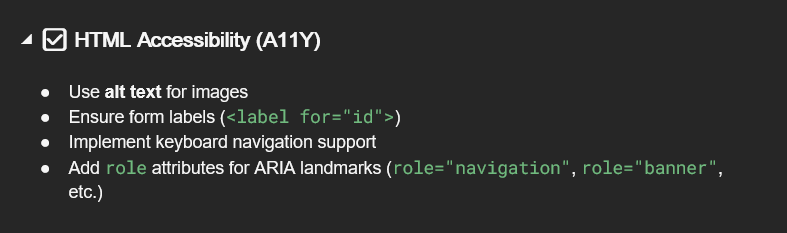


**HTML Meta & SEO**

Meta tags provide important information about a webpage.

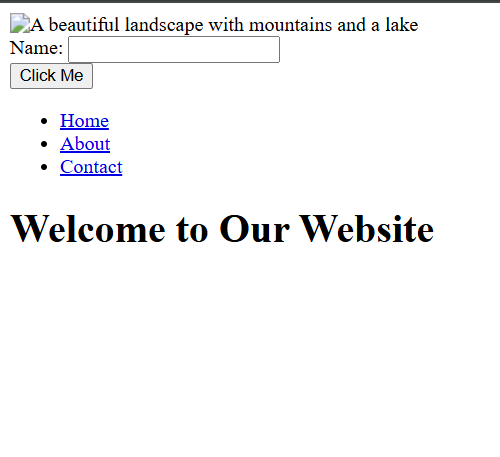
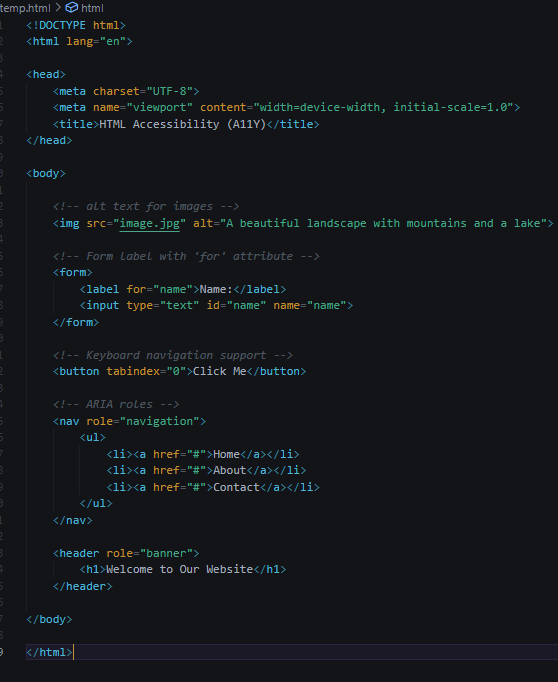
* <meta charset="UTF-8"> sets the character encoding.
* <meta name="viewport" content="width=device-width, initial-scale=1.0"> makes the site responsive.
* <meta name="author" content="Your Name"> specifies the author.
* <meta name="description" content="This is a webpage"> helps search engines.
* <meta name="keywords" content="HTML, SEO, Web Development"> improves search ranking.

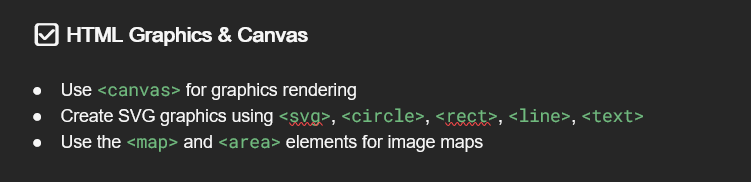




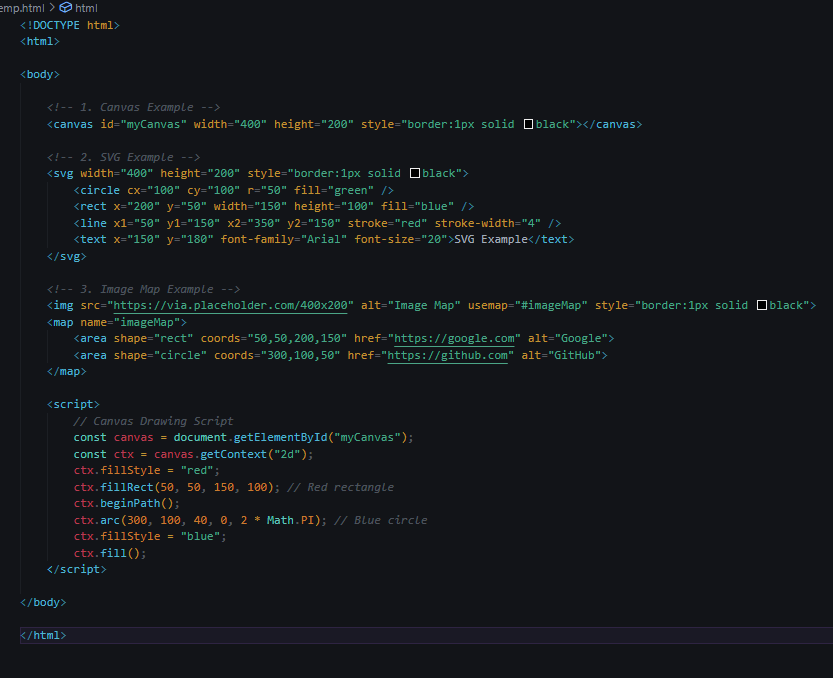
**HTML Accessibility (A11Y)**

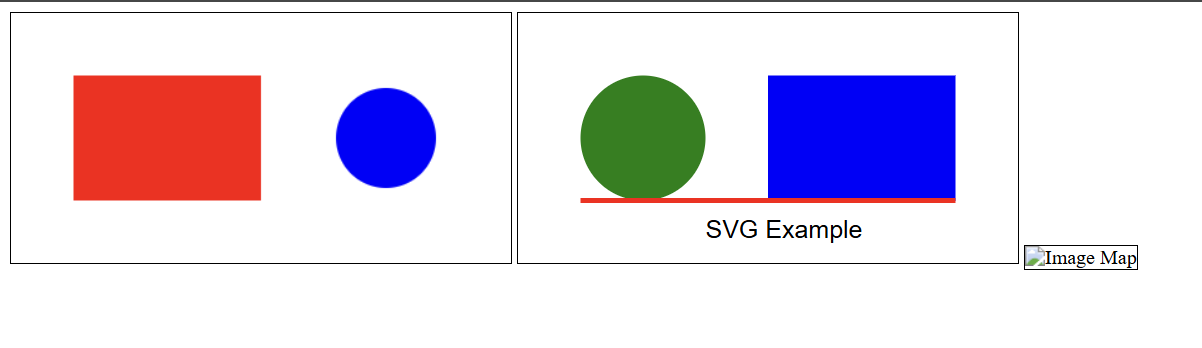
1. **alt** (Alternative text for images) – Provides a textual description of an image for screen readers.
2. **label** – Associates a label with a form element using the for attribute, improving accessibility.
3. **Keyboard Navigation** – Ensuring users can navigate using the Tab key and other keyboard shortcuts.
4. **role** – Defines ARIA (Accessible Rich Internet Applications) roles to improve accessibility for screen readers.



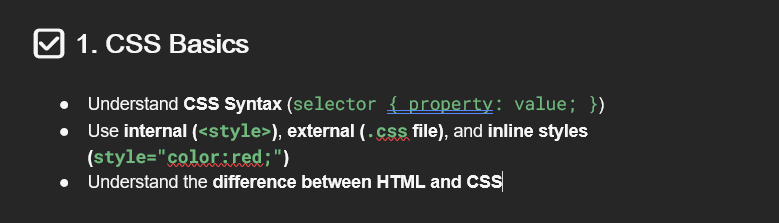


1. **<canvas>**: A JavaScript-based drawing surface for dynamic, pixel-based graphics (e.g., games, charts).
2. **SVG**: XML-based vector graphics (resolution-independent) with elements like <circle>, <rect>, and <line>.
3. **Image Maps**: Define clickable areas on an image using <map> and <area>, linked to URLs.





CSS



**CSS Basics**

1. **CSS Syntax:**  
   CSS (Cascading Style Sheets) is used to style HTML elements. The basic syntax follows this structure:

E.g.: selector {

property: value;

}

**Selector**: Targets an HTML element.

**Property**: Specifies the style attribute (e.g., color).

**Value**: Defines the style (e.g., red).

E.g.: p {

color: blue;

font-size: 16px;

}

1. CSS can be applied in three ways:

**Inline CSS:**

<p style="color: red;">This is a red paragraph.</p>

**Internal CSS:**

**<style>**

**h1 {**

**font-family: Arial, sans-serif;**

**color: green;**

**}**

</style>

**External CSS:**

**styles.css**

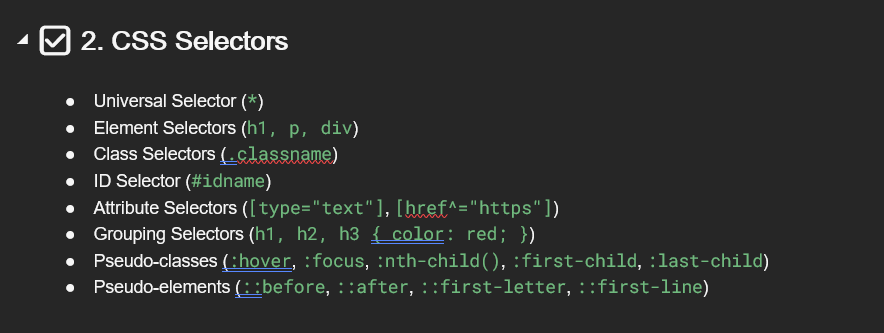
**body {**

**background-color: lightgray;**

**}**

**HTML**

**<link rel="stylesheet" href="styles.css">**



**Universal Selector (\*)**

* Selects all elements on the page.
* h1, p, div {

margin: 10px;

}

**Element Selector (h1, p, div)**

* Targets specific HTML elements.
* \* {

font-family: Arial, sans-serif;

}

**Class Selector (.classname)**

* Selects elements with a specific class.

.classname {

color: blue;

font-weight: bold;

}

**Attribute Selector ([type="text"], [href^="https"])**

* Selects elements based on their attributes.

#idname {

background-color: yellow;

padding: 5px;

}

**Grouping Selector (h1, h2, h3 {})**

* Applies the same styles to multiple elements.

h1, h2, h3 {

color: red;

}

**Pseudo-classes (:hover, :focus, etc.)**

* Apply styles when an element is in a specific state.

button:hover {

background-color: lightblue;

}

input:focus {

border-color: purple;

}

**Pseudo-elements (::before, ::after, ::first-letter, ::first-line)**

* Style parts of an element (not the whole element).

p::first-letter {

font-size: 2em;

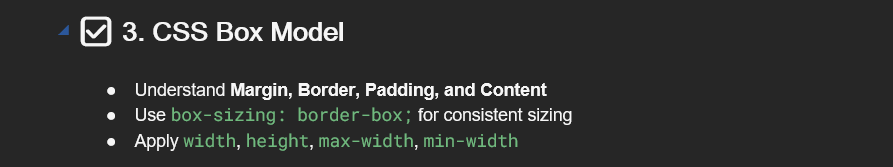
color: brown;

}

p::first-line {

font-weight: bold;

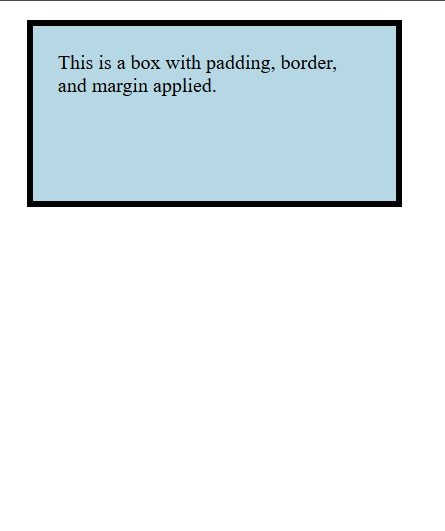
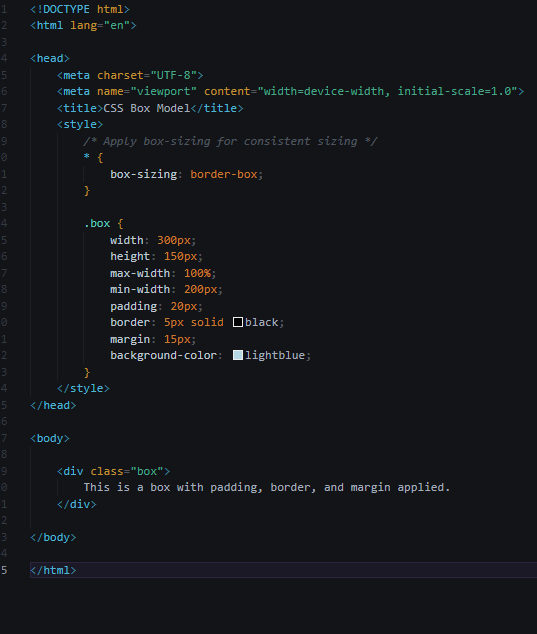
}

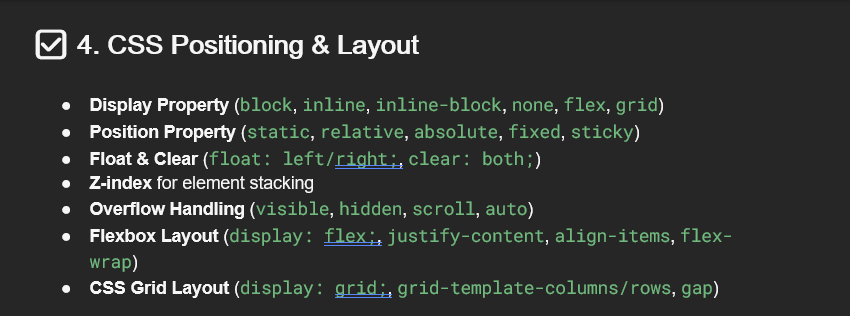


**CSS Box Model – Theory**

The **CSS Box Model** describes how elements are structured on a web page. Every element is a rectangular box consisting of **Content, Padding, Border, and Margin**.

* **Content**: The actual text or image inside the element.
* **Padding**: Space between the content and the border.
* **Border**: The boundary around the element.
* **Margin**: Space outside the border, separating elements.





**CSS Positioning & Layout – Theory**

CSS provides different techniques to control the layout and positioning of elements on a web page. Below are the key concepts:

**1. Display Property**

Defines how an element behaves in the layout.

* block: Takes up full width, starts on a new line. (e.g., <div>, <p>)
* inline: Stays in the same line without breaking (e.g., <span>, <a>)
* inline-block: Like inline, but allows width and height adjustments.
* none: Hides the element completely.
* flex: Enables the **Flexbox layout** for better alignment.
* grid: Enables the **Grid layout** for complex designs.

**2. Position Property**

Controls the placement of elements.

* static: Default positioning, follows normal document flow.
* relative: Moves the element **relative to its original position**.
* absolute: Moves the element relative to its nearest **positioned ancestor** or the entire page.
* fixed: Stays fixed in place, even when scrolling.
* sticky: Acts like relative but sticks when scrolling past a certain point.

**3. Float & Clear**

Float allows elements to be placed to the left or right of other content, while **clear** prevents elements from wrapping around floated elements.

* float: left/right; Moves elements to the left/right.
* clear: both; Ensures elements don’t wrap around floated ones.

**4. Z-Index**

Controls the **stacking order** of elements.

* Elements with a **higher z-index value** appear **on top** of lower ones.
* Only works with elements that have position: relative, absolute, fixed.

**5. Overflow Handling**

Determines what happens if content **exceeds the element’s box**.

* visible: Default; content overflows normally.
* hidden: Hides overflowing content.
* scroll: Adds a scrollbar if content overflows.
* auto: Scrollbar appears only if needed.

**6. Flexbox Layout**

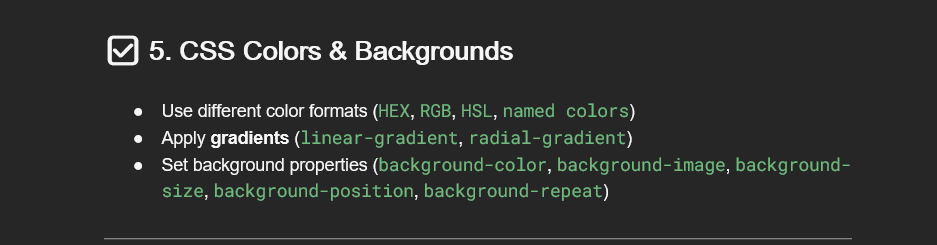
A powerful way to align and distribute space among elements in a container.

* display: flex; Enables Flexbox.
* justify-content: Aligns items horizontally (center, space-between, etc.).
* align-items: Aligns items vertically (center, flex-start, flex-end).
* flex-wrap: Allows items to wrap onto multiple lines if needed.

**7. CSS Grid Layout**

A **two-dimensional layout system** for arranging elements in rows and columns.

* display: grid; Enables Grid.
* grid-template-columns/rows: Defines the structure of the grid.
* gap: Adds spacing between grid items.



**CSS Colors & Backgrounds – Theory**

CSS provides multiple ways to style colors and backgrounds for better design and user experience. Below are the key concepts:

**1. Different Color Formats**

CSS supports multiple color formats for defining colors:

* HEX → Example: #ff5733 (Hexadecimal format)
* RGB → Example: rgb(255, 87, 51) (Red, Green, Blue)
* HSL → Example: hsl(9, 100%, 60%) (Hue, Saturation, Lightness)
* **Named Colors** → Example: red, blue, green, etc.

**2. Gradients**

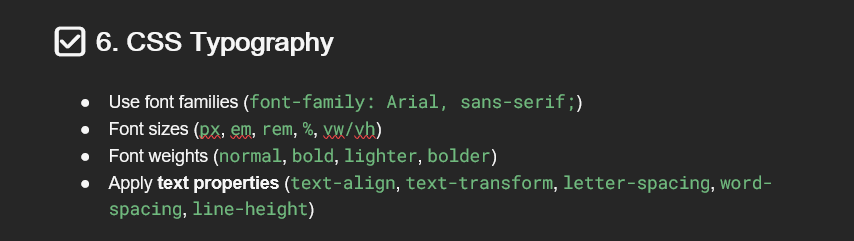
Gradients create smooth color transitions instead of solid colors.

* **Linear Gradient** (linear-gradient()) → Creates a gradient in a straight line.
  + Example: background: linear-gradient(to right, red, blue);
* **Radial Gradient** (radial-gradient()) → Creates a circular gradient.
  + Example: background: radial-gradient(circle, red, blue);

**3. Background Properties**

CSS offers various background properties for styling elements:

* background-color: Sets a solid background color.
* background-image: Adds an image as a background.
* background-size: Controls how the image fits (cover, contain, or custom size).
* background-position: Defines the position (center, top left, etc.).
* background-repeat: Controls repetition (repeat, no-repeat, repeat-x, repeat-y).



**CSS Typography – Theory**

Typography in CSS is used to style text, making it more readable and visually appealing. Below are the key typography concepts:

**1. Font Families**

Defines the font style for text.

* Example: font-family: Arial, sans-serif;
* Uses a fallback mechanism where the first font is preferred, and the second is used if unavailable.

**2. Font Sizes**

Controls text size using different units:

* px → Fixed size (e.g., 16px)
* em → Relative to parent element (1em = parent size)
* rem → Relative to root element (1rem = root font size)
* % → Percentage of the parent font size
* vw/vh → Relative to viewport width/height

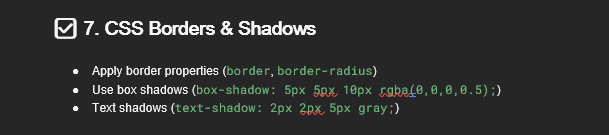
**3. Font Weights**

Adjusts the thickness of the font:

* normal → Default weight
* bold → Thicker text
* lighter → Lighter than normal
* bolder → Heavier than bold

**4. Text Properties**

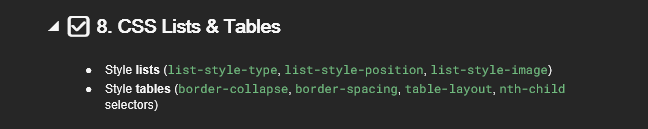
* text-align → Aligns text (left, right, center, justify)
* text-transform → Changes case (uppercase, lowercase, capitalize)
* letter-spacing → Adjusts space between letters
* word-spacing → Adjusts space between words
* line-height → Sets line spacing for better readability



**CSS Borders & Shadows – Theory**

Borders and shadows enhance the appearance of elements, giving them depth and emphasis.

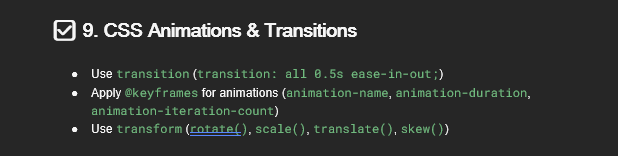
1. **Border Properties**: Defines the boundary of an element. Example: border: 2px solid black;
2. **Border Radius**: Rounds the corners of an element. Example: border-radius: 10px;
3. **Box Shadow**: Adds a shadow around an element. Example: box-shadow: 5px 5px 10px rgba(0,0,0,0.5);
4. **Text Shadow**: Creates shadow effects for text. Example: text-shadow: 2px 2px 5px gray;



**CSS Lists & Tables – Theory**

Lists and tables help organize content efficiently.

1. **Lists**: Styled using properties like list-style-type, list-style-position, and list-style-image.
2. **Tables**: Styled with border-collapse, border-spacing, table-layout, and nth-child for better readability.



**CSS Animations & Transitions – Theory**

Animations and transitions enhance user experience by adding motion effects.

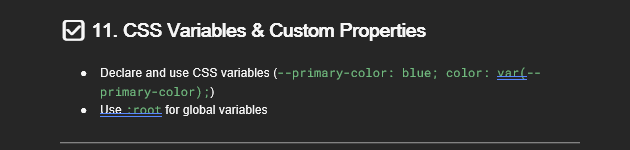
1. **Transitions**: Smoothly changes properties over a duration. Example: transition: all 0.5s ease-in-out;
2. **Keyframes**: Defines animations using @keyframes. Example: animation: slide 2s infinite;
3. **Transforms**: Modifies an element’s appearance using rotate(), scale(), translate(), and skew().



**CSS Media Queries & Responsive Design – Theory**

Responsive design ensures content adapts to different screen sizes.

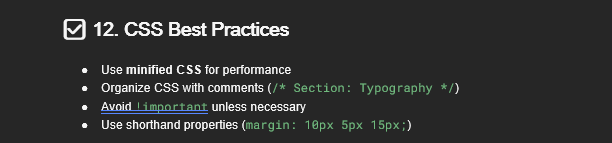
1. **Media Queries**: Use @media to apply styles based on screen width. Example: @media (max-width: 768px) {}
2. **Flexible Units**: Use vw, vh, %, rem, em for scalable designs.
3. **Mobile-First vs Desktop-First**: Prioritize mobile layout (min-width) or desktop layout (max-width) based on design needs.



**CSS Variables & Custom Properties – Theory**

CSS variables, also known as custom properties, enable reusability and maintainability in stylesheets.

1. **Declaring Variables**: Use the -- prefix to define a variable. Example: --primary-color: blue;
2. **Using Variables**: Apply with the var() function. Example: color: var(--primary-color);
3. **Global Variables with :root**: Declaring variables inside :root makes them accessible throughout the document.



**CSS Best Practices – Theory**

Following best practices improves code efficiency and maintainability.

1. **Minified CSS**: Reducing file size enhances performance. Example: Using tools like CSS Minifier.
2. **Organizing with Comments**: Helps structure code. Example: /\* Section: Typography \*/
3. **Avoiding !important**: Should only be used when necessary, as it overrides all styles.
4. **Shorthand Properties**: Combine multiple properties into one. Example: margin: 10px 5px 15px; saves space.