Unit III: Cascading Style Sheets

ℹ️ 3.1 Introduction: Cascading Style Sheets (CSS); CSS Syntax

Introduction to Cascading Style Sheets (CSS)

🌟 What is CSS?

CSS (Cascading Style Sheets) is a stylesheet language used to describe the **presentation** (look and feel) of a web page written in HTML.

Purpose of CSS:

- Controls **layout, colors, fonts, spacing, positioning, animations**, and more.
- Keeps the **content (HTML)** separate from **presentation (CSS)**.
- Makes it easier to maintain and update the look of a website.

🏗️ Structure Overview:

```
Website Project/
├── index.html      → HTML content (structure)
└── style.css       → CSS stylesheet (design & layout)
```

🧩 CSS Syntax Breakdown:

```css
selector {
  property: value;
}
```

Component	Description	Example
Selector	Specifies which HTML element(s) you want to style, different types on the basis of how the selection is to be made	`p` , `h1` , `.class` , `#id`
Property	The style attribute you want to change	`color` , `font-size` , `margin`
Value	The value assigned to the property	`red` , `16px` , `10px`

Declaration Block	Contains one or more **property: value** pairs inside `{}`	`{ color: red; font-size: 16px; }`

🔥 Example: Basic CSS Syntax

```css
/* This styles all <p> elements */
p {
  color: blue;
  font-size: 16px;
  text-align: justify;
}
```

- **Selector:** `p` → targets all `<p>` tags.
- **Properties & Values:**
  - `color: blue;` → sets text color.
  - `font-size: 16px;` → sets font size.
  - `text-align: justify;` → aligns text.

🏷️ Types of CSS Selectors:

Selector Type	Syntax Example	What it Selects
Element Selector	`p`, `h1`, `div`	Targets **all** elements of a specific type
Class Selector	`.classname`	Targets **all elements** with the specified class
ID Selector	`#idname`	Targets **one unique element** with the given ID
Universal Selector	`*`	Targets **all elements** on the page
Group Selector	`h1, p, div`	Targets **multiple selectors** at once
Attribute Selector	`input[type="text"]`	Targets elements with a specific attribute
Pseudo-class Selector	`a:hover`	Targets elements in a specific state (e.g., hover)
Pseudo-element Selector	`p::first-letter`	Targets specific parts of elements (e.g., first letter)

🎯 Quick Examples:

```css
/* Element Selector */
h1 { color: red; }

/* Class Selector */
.button { background-color: blue; }

/* ID Selector */
#header { font-size: 20px; }
```

```
/* Universal Selector */
* { margin: 0; padding: 0; }

/* Group Selector */
h1, p, div { border: 1px solid black; }

/* Attribute Selector */
input[type="text"] { border: 2px solid green; }

/* Pseudo-class Selector */
a:hover { color: orange; }

/* Pseudo-element Selector */
p::first-letter { font-size: 24px; color: purple; }
```

🚀 Key Difference:

Selector	Prefix	Used For	Uniqueness
Class	`.`	Style multiple elements	Can be reused
ID	`#`	Style a **single unique** element	Must be unique
Element	None	Style all elements of that type	Common use

🎯 Cascading Concept in CSS

The **"Cascading"** part means:

- **Multiple stylesheets or rules can apply to the same element.**
- The browser decides which rule to apply based on:
  1. **Specificity**
  2. **Importance (`!important`)**
  3. **Source order (later styles override earlier ones)**

📝 Practice Problem:

Given this HTML:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    h1 { color: red; }
    .special { color: green; }
    #unique { color: blue; }
  </style>
</head>
<body>
  <h1 id="unique" class="special">Hello World</h1>
</body>
</html>
```

Question: What will be the final color of the `<h1>` text? Why?

Final color: **Blue**

Why?

- `h1 { color: red; }` → general rule.
- `.special { color: green; }` → class selector.
- `#unique { color: blue; }` → ID selector.

ID selectors have the highest specificity, so `#unique` overrides others.

📋 Recap Table: Introduction to CSS & Syntax

Key Point	**Details**
Full Form	Cascading Style Sheets
Purpose	Controls design & layout of web pages
Syntax	`selector { property: value; }`
Selector Types	Element (`p`), Class (`.class`), ID (`#id`)
Multiple Styles Conflict	Resolved by Specificity, Importance, and Source Order
Separation of Content & Design	HTML → Structure, CSS → Presentation
Example	`p { color: blue; font-size: 16px; }`

💠 3.2 Inserting CSS: Inline, Internal, External

🎯 What's the purpose of inserting CSS?

To apply styles to your HTML, you need to **insert CSS** into your HTML file. There are **three main ways** to do it:

Type	**Where it's written**	**When to use**
Inline	Inside an HTML element (using `style` attribute)	For **quick, small, one-off styles**
Internal	Inside `<style>` tag in `<head>` of HTML	For **styling a single HTML file**
External	In a separate `.css` file, linked to HTML	For **styling multiple pages** (best practice!)

🌟 1. Inline CSS:

Syntax:

```
<h1 style="color: red; font-size: 20px;">Hello</h1>
```

✅ **Advantages:**

- Quick and easy for testing.

❌ **Disadvantages:**

- Not reusable.
- Hard to maintain.
- Mixes content with style (bad practice).

🌟 2. Internal CSS:

Syntax:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    p {
      color: green;
      font-size: 16px;
    }
  </style>
</head>
<body>
  <p>This is a paragraph.</p>
</body>
</html>
```

✅ **Advantages:**

- Styles all elements in the page.

❌ **Disadvantages:**

- Affects **only one page**.
- Styles are not reusable across pages.

🌟 3. External CSS:

Structure:

```
Website Project/
├── index.html
└── styles.css
```

Linking External CSS:

HTML (index.html):

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

CSS (styles.css):

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

h1 {
  color: navy;
}
```

✅ Advantages:

- Reusable across **multiple pages**.
- Cleaner separation of content & design.
- Easy to maintain.

❌ Disadvantages:

- Requires separate HTTP request to load CSS file (can affect load time if not optimized).

📊 Comparison Table:

Method	**Location**	**Scope**	**Pros**	**Cons**
Inline	Inside HTML element	Single element	Quick, simple	Not reusable, hard to maintain
Internal	`<style>` in `<head>`	Whole HTML page	Styles in one place, affects full page	Can't be reused across pages
External	Separate `.css` file	Multiple pages	Best practice, reusable, clean, maintainable	Needs extra file load, but efficient overall

---

📝 Practice Problem:

👉 You have **two HTML files**: `home.html` and `about.html`.

You want both to have the same header and paragraph styles.

Which method should you use? Why?

Use External CSS.

Because it allows you to link the same `.css` file to both `home.html` and `about.html`, keeping styles consistent and easy to maintain.

📋 Recap Table: Inserting CSS

CSS Method	Syntax Location	Usage	Example
Inline	`style=""` attribute in element	Single element styling	`<p style="color: red;">Hello</p>`
Internal	`<style>` in `<head>`	Styling one HTML page	`<style> p { color: green; } </style>`
External	Separate `.css` file linked via `<link>`	Styling multiple pages	`<link rel="stylesheet" href="styles.css">`

🆔 3.3 CSS ID and Class Selectors

🎯 Why use ID and Class Selectors?

They give you **precise control** over which elements to style:

Selector	Purpose	Uniqueness
ID Selector	Styles **one unique element**	Must be unique in page
Class Selector	Styles **multiple elements sharing the same class**	Can be reused

---

🌟 1. ID Selector

Syntax:

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

HTML Example:

```html
<h1 id="main-heading">Welcome!</h1>
```

CSS Example:

```css
#main-heading {
  color: blue;
  font-size: 24px;
}
```

✅ **Key Point:**

The ID value is **unique** – it should be used **once per page**.

🌟 2. Class Selector

Syntax:

```css
.classname {
  property: value;
}
```

HTML Example:

```html
<p class="highlight">This is important text.</p>
<p class="highlight">Another important paragraph.</p>
```

CSS Example:

```css
.highlight {
  background-color: yellow;
  font-weight: bold;
}
```

✅ **Key Point:**

Classes are **reusable – multiple elements** can have the same class.

📝 Practice Problem:

Here's an HTML snippet:

```html
<h1 id="title">Hello</h1>
<p class="text">This is a paragraph.</p>
```

```
<p class="text">Another paragraph.</p>
```

Write CSS to:

1. Make the `<h1>` color red.
2. Make all paragraphs with class `text` color green.

Solution:

```
#title {
  color: red;
}

.text {
  color: green;
}
```

📊 Difference Table: ID vs Class

Feature	**ID Selector**	**Class Selector**
Syntax	`#idname {}`	`.classname {}`
Uniqueness	Must be **unique per page**	Can be used on **multiple elements**
Usage	For **single, unique elements**	For **grouping similar elements**
Priority	Higher specificity	Lower specificity
Example	`#header { color: blue; }`	`.menu-item { color: red; }`

🧠 Important Note on Specificity:

If an element has **both an ID and a Class**, **ID styles will override Class styles** due to higher specificity.

📋 Recap Table: ID and Class Selectors

Selector Type	**Symbol**	**Use Case**	**Example in HTML**	**Example in CSS**
ID Selector	`#`	Unique, single element styling	`<div id="unique-box"></div>`	`#unique-box { border: 1px solid; }`
Class Selector	`.`	Group multiple similar elements	`<p class="note"></p>`	`.note { color: green; }`

🎨 3.4 Colors, Backgrounds, Borders, Text, Font, List, Table

🎨 Colors in CSS

Setting Text Color

The `color` property in CSS is used to define the color of the text within an element.

Syntax:

```css
selector {
  color: value;
}
```

Examples:

```css
/* Using a color name */
p {
  color: red;
}

/* Using a HEX value */
h1 {
  color: #00ff00;
}

/* Using an RGB value */
span {
  color: rgb(0, 0, 255);
}

/* Using an HSL value */
div {
  color: hsl(240, 100%, 50%);
}
```

Color Values

CSS supports various ways to define colors:

- **Color Names:** Predefined names like `red`, `blue`, `green`, etc.
- **HEX Values:** A hexadecimal representation, e.g., `#ff0000` for red.
- **RGB Values:** Defines colors using Red, Green, and Blue components, e.g., `rgb(255, 0, 0)`.

- **HSL Values:** Stands for Hue, Saturation, and Lightness, e.g., `hsl(0, 100%, 50%)`.

Note: If the `color` property is not set, it inherits the color from its parent element.

🖼️ Backgrounds in CSS

The `background` property is a shorthand for setting various background properties in one declaration.

Syntax:

```
selector {
  background: [color] [image] [position] [size] [repeat] [attachment] [origin] [clip];
}
```

Examples:

```
/* Setting a background color */
body {
  background-color: lightblue;
}

/* Setting a background image */
div {
  background-image: url('background.jpg');
  background-size: cover;
  background-repeat: no-repeat;
}

/* Setting multiple background images */
section {
  background-image: url('image1.png'), url('image2.png');
  background-position: left top, right bottom;
}
```

Background Properties

- **background-color:** Sets the background color of an element.
- **background-image:** Sets one or more background images for an element.
- **background-position:** Sets the starting position of a background image.
- **background-size:** Specifies the size of the background images.
- **background-repeat:** Sets if/how a background image will be repeated.
- **background-attachment:** Sets whether a background image scrolls with the rest of the page or is fixed.
- **background-origin:** Specifies the positioning area of the background images.

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

🖌️ Borders in CSS

Borders are used to define the boundary of an element. The `border` property is a shorthand for setting the width, style, and color of an element's border.

Syntax:

```
selector {
  border: [width] [style] [color];
}
```

Examples:

```
/* Setting a solid border */
p {
  border: 2px solid black;
}

/* Setting a dashed border */
div {
  border: 1px dashed #ff0000;
}

/* Setting individual border sides */
h1 {
  border-top: 5px double blue;
  border-right: 2px solid green;
  border-bottom: 5px double blue;
  border-left: 2px solid green;
}
```

Border Properties

- **border-width:** Specifies the width of the border.
- **border-style:** Specifies the style of the border (e.g., `none`, `solid`, `dashed`, `double`).
- **border-color:** Specifies the color of the border.

Note: The `border-color` property can accept color names, HEX values, RGB values, or HSL values. If `border-color` is not set, it inherits the color of the element. ṆciteÖturn0search5Ộ

📝 Text in CSS

CSS provides various properties to style text within an element.

Examples:

```css
/* Setting text alignment */
p {
  text-align: center;
}

/* Setting text decoration */
a {
  text-decoration: none;
}

/* Setting text transformation */
h2 {
  text-transform: uppercase;
}

/* Setting text indentation */
blockquote {
  text-indent: 50px;
}

/* Setting line height */
p {
  line-height: 1.5;
}
```

Text Properties

- **text-align:** Specifies the horizontal alignment of text (`left`, `right`, `center`, `justify`).
- **text-decoration:** Specifies the decoration added to text (`none`, `underline`, `overline`, `line-through`).
- **text-transform:** Controls the capitalization of text (`none`, `capitalize`, `uppercase`, `lowercase`).
- **text-indent:** Specifies the indentation of the first line of text.
- **line-height:** Sets the height between lines of text.

🔤 Fonts in CSS

Fonts determine the appearance of text characters. The `font` property is a shorthand for setting various font properties.

Syntax:

```css
selector {
  font: [style] [variant] [weight] [size]/[line-height] [family];
}
```

Examples:

```
/* Setting font family */
body {
  font-family: Arial, sans-serif;
}

/* Setting font size */
p {
  font-size: 16px;
}

/* Setting font weight */
h1 {
  font-weight: bold;
}

/* Setting font style */
em {
  font-style: italic;
}

/* Using the shorthand font property */
div {
  font: italic small-caps bold 16px/1.5 'Times New Roman', serif;
}
```

Font Properties

- **font-family:** Specifies the font family for text.
- **font-size:** Specifies the size of the font.
- **font-weight:** Specifies the weight (boldness) of the font (`normal`, `bold`, `bolder`, `lighter`, or numerical values from 100 to 900).
- **font-style:** Specifies the style of the font (`normal`, `italic`, `oblique`).
- **font-variant:** Specifies whether the text should be displayed in small-caps.
- **line-height:** Sets the space between lines of text.

Note: When specifying `font-family`, it's a good practice to provide a fallback font in case the primary font is not available.

📋 Lists in CSS

CSS provides various properties to style both ordered (`<ol>`) and unordered (`<ul>`) lists, allowing for customization of list markers, positions, and images.

List Style Type

The `list-style-type` property specifies the appearance of the list item marker.

Values:

Value	Description
`disc`	Solid circle (default for `<ul>`).
`circle`	Hollow circle.
`square`	Solid square.
`decimal`	Numbers (default for `<ol>`).
`decimal-leading-zero`	Numbers with leading zeros (01, 02, 03, etc.).
`lower-roman`	Lowercase Roman numerals (i, ii, iii, etc.).
`upper-roman`	Uppercase Roman numerals (I, II, III, etc.).
`lower-alpha`	Lowercase letters (a, b, c, etc.).
`upper-alpha`	Uppercase letters (A, B, C, etc.).

Example:

```
/* Unordered list with square markers */
ul {
  list-style-type: square;
}

/* Ordered list with uppercase Roman numerals */
ol {
  list-style-type: upper-roman;
}
```

HTML:

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
</ul>

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

Rendered Output:

- **Unordered List:**
  - ■ Item 1
  - ■ Item 2
- **Ordered List:**
  - I. First
  - II. Second

List Style Position

The `list-style-position` property specifies the position of the list-item markers (bullet points or numbers).

Values:

Value	Description
`outside`	Marker is outside the list item (default).
`inside`	Marker is inside the list item, causing text indentation.

Example:

```css
/* Markers outside the list item */
ul.outside {
  list-style-position: outside;
}

/* Markers inside the list item */
ul.inside {
  list-style-position: inside;
}
```

HTML:

```html
<ul class="outside">
  <li>Outside Marker</li>
  <li>Outside Marker</li>
</ul>

<ul class="inside">
  <li>Inside Marker</li>
  <li>Inside Marker</li>
</ul>
```

Rendered Output:

- **Outside Marker:**
  - Outside Marker

    - Outside Marker
- **Inside Marker:**
    - Inside Marker
    - Inside Marker

List Style Image

The `list-style-image` property sets an image as the list item marker.

Example:

```
/* Using a custom image as list marker */
ul.custom-marker {
  list-style-image: url('marker.png');
}
```

HTML:

```
<ul class="custom-marker">
  <li>Custom Marker Item 1</li>
  <li>Custom Marker Item 2</li>
</ul>
```

Rendered Output:

- ![marker.png] Custom Marker Item 1
- ![marker.png] Custom Marker Item 2

Note: Ensure the image URL is correct and accessible for the markers to display properly.

Shorthand Property

The `list-style` property is a shorthand for setting `list-style-type`, `list-style-position`, and `list-style-image` in one declaration.

Syntax:

```
selector {
  list-style: [list-style-type] [list-style-position] [list-style-image];
}
```

Example:

```
/* Shorthand for setting list style */
ul {
  list-style: square inside url('marker.png');
}
```

This sets the list to have square markers, positioned inside, with a custom image as the marker.

Practice Problem

Problem: Style the following list so that:

1. The list uses lowercase Greek letters as markers.
2. The markers are positioned inside the list items.
3. If the browser doesn't support Greek letters, it should fall back to decimal numbers.

HTML:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>List Practice</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <ol class="custom-list">
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
  </ol>
</body>
</html>
```

CSS (`styles.css`):

```css
/* Answer */
.custom-list {
  list-style-type: lower-greek;
  list-style-position: inside;
}

/* Fallback for browsers that don't support lower-greek */
.custom-list {
  list-style-type: decimal;
}
```

Rendered Output:

- α First item
- β Second item
- γ Third item

Note: The fallback ensures compatibility with browsers that may not support the `lower-greek` value.

Recap Table: Lists in CSS

Property	Description	Example
`list-style-type`	Specifies the marker style for list items.	`ul { list-style-type: square; }`
`list-style-position`	Specifies the position of the list-item markers.	`ul { list-style-position: inside; }`
`list-style-image`	Sets an image as the list item marker.	`ul { list-style-image: url('marker.png'); }`
`list-style`	Shorthand for setting all list style properties.	`ul { list-style: square inside; }`

📊 Tables in CSS

CSS provides properties to style HTML tables, enhancing their appearance and readability.

Table Borders

The `border` property specifies the border of the table and its cells.

Example:

```css
/* Adding borders to table, th, and td */
table, th, td {
  border: 1px solid black;
}
```

HTML:

```html
<table>
  <tr>
    <th>Header 1</th>
    <th>Header 2</th>
  </tr>
</table>
```

📦 3.5 CSS Box Model, Normal Flow Box Layout: Basic Box Layout

📦 CSS Box Model

In CSS, the **Box Model** is a fundamental concept that describes the structure of elements on a webpage. Every element is considered as a rectangular box, which consists of the following components:

1. **Content**: The actual content of the box, such as text, images, or other media.
2. **Padding**: The space between the content and the border. It clears an area around the content, inside the border.
3. **Border**: A border that surrounds the padding (if any) and content.
4. **Margin**: The outermost layer that clears an area outside the border. It creates space between the current box and adjacent boxes.
   This structure is crucial for controlling the design and layout of web pages.

Visual Representation:

```
+---------------------------+
|      Margin          |
| +---------------------+ |
| |     Border       | |
| | +---------------+ | |
| | |  Padding    | | |
| | | +---------+ | | |
| | | | Content | | | |
| | | +---------+ | | |
| | +---------------+ | |
| +---------------------+ |
+---------------------------+
```

Example:

```
.element {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
  margin: 10px;
}
```

In this example:

- **Content width**:Ő200pỢ
- **Padding**:Ő20px (on all sidesỢ
- **Border**:Ő5px solid blacỢ
- **Margin**:Ő10px (on all sidesỢ
  ŐThe total width of the element would be calculated asỢ
  Ő

`Total width = content width + (padding * 2) + (border * 2) + (margin * 2)Ợ Ő` Total width = 200px + (20px * 2) + (5px * 2) + (10px * 2) = 290pxỢ
ŐSimilarly, the total height would be calculated by adding the content height, padding, border, and margin in the vertical directionỢ

🌐 Normal Flow in CSS

Normal Flow refers to the default layout behavior of elements in a web page before any CSS positioning or floating is applie. Understanding normal flow is essential for controlling the layout and ensuring that elements appear as intended across different devices and screen sizes.

Characteristics of Normal Flow:

- **Block-level elements** These elements (e.g., `<div>`, `<p>`, `<h1>`) occupy the full width available and stack vertically, one after the other. Each block-level element starts on a new line.
- **Inline elements** Elements like `<span>`, `<a>`, and `<strong>` are inline by default. They do not start on a new line and only take up as much width as necessary. Inline elements are laid out horizontally within a line box.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Normal Flow Example</title>
  <style>
    .block {
      background-color: lightblue;
      margin: 10px 0;
    }
    .inline {
      background-color: lightgreen;
    }
  </style>
</head>
<body>
  <div class="block">Block-level Element 1</div>
  <div class="block">Block-level Element 2</div>
  <p>This is a paragraph with <span class="inline">inline elements</span> like <a href="#" class
="inline">links</a> and <strong class="inline">bold text</strong>.</p>
</body>
</html>
```

In this example:

- The `<div>` elements are block-level and stack vertically with margins separating them.
- The `<span>`, `<a>`, and `<strong>` elements are inline and flow within the paragraph text without starting new lines.
  ŐUnderstanding normal flow is crucial for creating predictable and maintainable layouts. By default, elements are laid out in normal flow, and designers can then apply CSS properties like `float`, `position`, or `display` to alter this behavior as neede.Ȯ ŅciteŐturn0searchỘỢ

🖥️ 3.6 Display Property, Padding, Margin, Positioning: Relative, Float, Absolute

🖥️ CSS `display` Property

The `display` property in CSS determines how an element is displayed on the webpage. It plays a crucial role in layout design, affecting the flow and arrangement of elements.

Common Display Values

1. **`block`**: The element occupies the full width available, starting on a new line. Block-level elements can have width and height set, and margins and padding are respected. Examples include `<div>`, `<p>`, and `<h1>` elements.

   Example:

   ```
   <div style="display: block; background-color: lightblue;">
     This is a block-level element.
   </div>
   ```

2. **`inline`**:ŐThe element does not start on a new line and only takes up as much width as necessary. Width and height properties do not apply, but padding and margins (left and right) are respected. Examples include `<span>`, `<a>`, and `<strong>` elementsỢ

   Example:

   ```
   <span style="display: inline; background-color: lightgreen;">
     This is an inline element.
   </span>
   ```

3. **`inline-block`**:ŐCombines characteristics of both `block` and `inline`. The element flows inline but respects width and height propertiesỢ

   Example:

   ```
   <div style="display: inline-block; width: 100px; height: 50px; background-color: lightcoral;">
     This is an inline-block element.
   ```

```
</div>
```

4. `none`:ŐThe element is not displayed and does not occupy any space in the layoutỢ

   Example:

```
<p style="display: none;">
  This paragraph will not be displayed.
</p>
```

Practice Problems

1. **Identifying Display Types:**

   Given the following HTML elements, identify their default `display` property values:

   - `<header>`
   - `<span>`
   - `<section>`
   - `<em>`

2. **Applying Display Properties:**

   How would you modify a `<div>` to behave like an inline element using the `display` property?

Recap Table: CSS `display` Property

Display Value	Description	Can Set Width/Height	Starts on New Line
`block`	Element occupies full width and starts on a new line.	Yes	Yes
`inline`	Element occupies only necessary width and does not start on a new line.	No	No
`inline-block`	Element flows inline but respects width and height properties.	Yes	No
`none`	Element is not displayed and does not occupy space in the layout.	N/A	N/A

🛠️ CSS `padding` Property

The `padding` property in CSS is used to generate space around an element's content, inside of any defined borders. It enhances the readability and visual structure of web content by providing necessary spacing.

Syntax and Usage

The `padding` property can be applied in several ways:

1. **Shorthand Property**: Allows setting padding for all four sides in one declaration.

   ```
   padding: top right bottom left;
   ```

   For example:

   ```
   padding: 10px 20px 15px 5px;
   ```

   This sets:

   - Top padding to 10px
   - Right padding to 20px
   - Bottom padding to 15px
   - Left padding to 5px

2. **Individual Properties**: Allows setting padding for each side separately.

   ```
   padding-top: 10px;
   padding-right: 20px;
   padding-bottom: 15px;
   padding-left: 5px;
   ```

3. **Single Value**: Applies the same padding to all four sides.

   ```
   padding: 10px;
   ```

4. **Two Values**: The first value applies to the top and bottom, the second to the left and right.

   ```
   padding: 10px 20px;
   ```

5. **Three Values**: The first value applies to the top, the second to the left and right, and the third to the bottom.

   ```
   padding: 10px 20px 15px;
   ```

Acceptable Units

Padding values can be defined using various units:

- **Length Units**: Such as pixels (`px`), ems (`em`), rems (`rem`), points (`pt`), centimeters (`cm`), etc.
- **Percentages**: Relative to the width of the containing element.

Note: Negative values for padding are not allowed.

Examples

1. **Uniform Padding**:

   ```css
   .box {
     padding: 20px;
   }
   ```

   This applies 20px padding to all sides of the `.box` element.

2. **Asymmetric Padding**:

   ```css
   .container {
     padding: 10px 15px 20px 25px;
   }
   ```

   This sets different padding values for each side of the `.container` element.

3. **Percentage Padding**:

   ```css
   .content {
     padding: 5%;
   }
   ```

   This sets the padding to 5% of the width of the containing element.

Impact on Element Size

Padding affects the total size of an element. By default, the width and height properties define the size of the content box. Padding adds to this size, increasing the overall dimensions.

Example:

```css
.box {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
}
```

In this case, the total width of the `.box` element is calculated as:

- Content width: 200px
- Padding: 20px (left) + 20px (right) = 40px
- Border: 5px (left) + 5px (right) = 10px

Total width = 200px + 40px + 10px = 250px

To include padding and border within the specified width and height, you can use the `box-sizing` property set to `border-box`.

```
.box {
  width: 200px;
  padding: 20px;
  border: 5px solid black;
  box-sizing: border-box;
}
```

With `box-sizing: border-box`, the total width remains 200px, as padding and border are included within this width.

📄 Practice Problems

1. **Calculating Total Width and Height**:

   Given the following CSS rule:

   ```
   .element {
     width: 150px;
     height: 100px;
     padding: 10px 20px;
     border: 5px solid;
   }
   ```

   - Calculate the total width and height of the `.element` box.

2. **Setting Padding with Percentages**:

   If a container has a width of 500px, what is the padding in pixels when the following CSS is applied?

   ```
   .container {
     padding: 5%;
   }
   ```

3. **Using Shorthand Property**:

   Write the shorthand CSS property to apply the following padding:

   - Top: 10px
   - Right and Left: 15px
   - Bottom: 20px

🔑 Recap Table: CSS `padding` Property

Property	Description	Example
`padding`	Sets padding for all four sides	`padding: 10px;`

`padding-top`	Sets padding for the top side	`padding-top: 10px;`
`padding-right`	Sets padding for the right side	`padding-right: 10px;`
`padding-bottom`	Sets padding for the bottom side	`padding-bottom: 10px;`
`padding-left`	Sets padding for the left side	`padding-left: 10px;`
Shorthand (4 values)	Top, Right, Bottom, Left padding in one line	`padding: 10px 15px 20px 5px;`
Units	px, %, em, rem, pt, cm, etc.	`padding: 5%;`
Negative Values	**Not Allowed**	
Impact on Size	Adds space inside border, increases element size	
Box-Sizing Fix	Use `box-sizing: border-box` to contain padding	`box-sizing: border-box;`

📌 CSS Margins

🌟 What is Margin in CSS?

The **margin** in CSS is the space **outside** an element's border. It creates space between the element and its neighboring elements.

Margin Properties:

Property	Description	Example
`margin`	Sets all four margins at once	`margin: 10px;`
`margin-top`	Sets top margin	`margin-top: 20px;`
`margin-right`	Sets right margin	`margin-right: 15px;`
`margin-bottom`	Sets bottom margin	`margin-bottom: 25px;`
`margin-left`	Sets left margin	`margin-left: 30px;`

Shorthand Syntax:

```
margin: top right bottom left;
```

Examples:

```
margin: 10px 20px 30px 40px;
```

Meaning:

- Top margin = 10px
- Right margin = 20px
- Bottom margin = 30px
- Left margin = 40px

Shorter versions:

Syntax	Meaning
`margin: 10px;`	All sides = 10px
`margin: 10px 20px;`	Top & Bottom = 10px, Right & Left = 20px
`margin: 10px 20px 30px;`	Top = 10px, Right & Left = 20px, Bottom = 30px

🌐 Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
   .box {
     width: 200px;
     height: 100px;
     background-color: lightblue;
     margin: 20px 30px 40px 50px;
   }
  </style>
</head>
<body>

  <div class="box">This box has margin</div>

</body>
</html>
```

Explanation:

- Adds 20px margin on top, 30px on the right, 40px on the bottom, 50px on the left.

🎯 Practice Problem:

Question:

What margin values will you use to apply:

- Top margin: 15px
- Right margin: 25px
- Bottom margin: 35px
- Left margin: 45px

Answer:

```
margin: 15px 25px 35px 45px;
```

⚠️ Important Points:

1. **Margins can have negative values!**
    - Example: `margin: -10px;` pulls the element closer.
2. **Auto Margin:**
    - Used for centering block elements horizontally.
    - Example:

    ```
    margin: 0 auto;
    ```

🔥 Recap Table:

Feature	Explanation
What is Margin?	Space **outside** the element's border
Shorthand Syntax	`margin: top right bottom left;`
Auto Margin	`margin: 0 auto;` centers block elements horizontally
Negative Margin	Possible, pulls the element inward
Individual Properties	`margin-top`, `margin-right`, `margin-bottom`, `margin-left`

📌 CSS Positioning

CSS **Positioning** controls how an element is positioned in the document flow or relative to its container.

🌟 Position Property Values:

Value	Description
`static`	Default. Elements follow normal document flow (top to bottom, left to right).
`relative`	Positions the element **relative to its normal position**. Allows shifting using `top`, `left`, etc.
`absolute`	Positions the element **relative to the nearest positioned ancestor** (non-static). Removes from flow.
`fixed`	Positions the element **relative to the viewport**. Stays fixed even when scrolling.
`sticky`	Element toggles between `relative` and `fixed` based on scroll position.

🎯 Today, we will focus on these three:

1. **Relative**
2. **Absolute**

3. **Float** (Note: Not part of `position`, but used for positioning)

1️⃣ Position: Relative

- Moves the element **relative to its normal position**.
- Still occupies space in the document flow.

Syntax:

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

Effect:

- Moves the element **down by 10px** and **right by 20px**.
- Other elements treat it as if it's still in the normal spot.

2️⃣ Position: Absolute

- Removed from normal document flow.
- Positioned **relative to nearest positioned ancestor**.
    - If no ancestor has `position: relative/absolute/fixed`, positions relative to `<html>` (viewport).

Syntax:

```css
.parent {
  position: relative; /* Acts as reference */
}

.child {
  position: absolute;
  top: 10px;
  left: 20px;
}
```

Effect:

- `.child` moves **10px from top & 20px from left of `.parent`**, not affecting siblings.

3️⃣ Float Property

- Used to **float elements left or right**, making text or inline content wrap around it.

- Removed partially from normal flow but still occupies space.

Syntax:

```
.image {
  float: left;
  margin-right: 10px;
}
```

🌐 Example (All Three):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    .relative-box {
      width: 150px;
      height: 100px;
      background-color: lightgreen;
      position: relative;
      top: 20px;
      left: 30px;
    }

    .absolute-parent {
      position: relative;
      width: 300px;
      height: 200px;
      background-color: lightgray;
    }

    .absolute-child {
      position: absolute;
      top: 10px;
      left: 10px;
      width: 100px;
      height: 100px;
      background-color: coral;
    }

    .float-box {
      float: left;
      width: 100px;
      height: 100px;
      background-color: lightblue;
      margin-right: 10px;
    }
  </style>
</head>
<body>

  <div class="relative-box">Relative</div>

  <div class="absolute-parent">
    <div class="absolute-child">Absolute</div>
  </div>

  <div class="float-box">Float</div>
  <p>This text wraps around the floated box. Notice how float pulls the box left.</p>

</body>
</html>
```

🔥 Difference Table:

Property	Document Flow	Relative To	Affects Siblings	Scroll Behavior
`relative`	Stays	Its original position	No	Moves with scroll
`absolute`	Removed	Nearest positioned ancestor	No	Moves with scroll
`float`	Partially	Normal flow (left/right)	Yes, affects flow	Moves with scroll

---

🎯 Practice Problem:

Question:

Given this structure:

```
<div class="container">
  <div class="box">Box</div>
</div>
```

Make `.box` positioned 50px from the left & top of `.container` using `absolute`.

Answer:

```
.container {
  position: relative;
}

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

📝 Recap Table:

Concept	Key Point
`relative`	Moves element relative to its normal spot. Space occupied remains.
`absolute`	Moves element relative to nearest positioned ancestor. Removed from normal flow.
`float`	Moves element left/right, text/content wraps around.
Important	For `absolute`, always check if parent has `position: relative/absolute/fixed`.

📌 3.7 CSS3 Borders, Box Shadows, Text Effects & Text Shadows

We'll break this topic into four key parts:

1. **CSS Borders**
2. **Box Shadows**
3. **Text Effects**
4. **Text Shadows**

1️⃣ CSS Borders

Borders define the outline around an element.

Border Properties:

Property	Description	Example
`border-width`	Thickness of the border	`border-width: 3px;`
`border-style`	Type of border line	`solid`, `dashed`, `dotted`
`border-color`	Color of the border	`border-color: red;`
`border-radius`	Rounds the corners (CSS3)	`border-radius: 10px;`

Shorthand Syntax:

```
border: 3px solid blue;
```

Example:

```
<div style="border: 3px dashed green; border-radius: 10px; padding: 10px;">
  Rounded Dashed Border
</div>
```

2️⃣ Box Shadows

Adds shadow effects around elements.

Syntax:

```
box-shadow: offset-x offset-y blur-radius spread-radius color;
```

Term	Meaning	Example
`offset-x`	Horizontal shadow (positive → right, negative → left)	`5px`
`offset-y`	Vertical shadow (positive → down, negative → up)	`10px`
`blur-radius`	How blurry the shadow is	`8px`
`spread`	How much the shadow expands or contracts	`2px`
`color`	Shadow color	`rgba(0,0,0,0.5)`

Example:

```
div {
  width: 200px;
  height: 100px;
  background-color: lightblue;
  box-shadow: 5px 10px 8px 2px rgba(0,0,0,0.5);
}
```

Result:

- Shadow 5px to the right, 10px down, blurred by 8px, spread by 2px, color semi-transparent black.

3️⃣ Text Effects (CSS3)

Some notable text effects include:

Property	Description	Example
`text-transform`	Controls capitalization	`uppercase`, `lowercase`, `capitalize`
`letter-spacing`	Space between letters	`letter-spacing: 2px;`
`word-spacing`	Space between words	`word-spacing: 5px;`
`text-decoration`	Underline, overline, line-through	`underline`, `overline`, `none`

`text-align`	Aligns text (left, right, center, justify)	`text-align: center;`
`line-height`	Line spacing	`line-height: 1.5;`

Example:

```
p {
  text-transform: uppercase;
  letter-spacing: 2px;
  word-spacing: 5px;
  text-align: justify;
}
```

4️⃣ Text Shadows

Adds shadow effects to text.

Syntax:

```
text-shadow: offset-x offset-y blur-radius color;
```

Term	Meaning	Example
`offset-x`	Horizontal shadow	`2px`
`offset-y`	Vertical shadow	`4px`
`blur-radius`	Shadow blur	`3px`
`color`	Shadow color	`rgba(0,0,0,0.5)`

Example:

```
h1 {
  text-shadow: 2px 4px 3px rgba(0,0,0,0.5);
}
```

🌟 Full Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    .box {
      width: 200px;
      height: 100px;
      background-color: lightgreen;
      border: 3px solid blue;
      border-radius: 10px;
      box-shadow: 5px 5px 10px 2px gray;
      text-align: center;
      line-height: 100px;
      text-shadow: 2px 2px 5px black;
      letter-spacing: 2px;
      word-spacing: 5px;
      text-transform: uppercase;
    }
  </style>
</head>
<body>

  <div class="box">Styled Box</div>

</body>
</html>
```

🎯 Difference Table:

Feature	Applies To	Description
`border`	Box Element	Outline around an element
`box-shadow`	Box Element	Shadow effect behind box
`text-shadow`	Text Content	Shadow effect behind text
`border-radius`	Box Element	Rounds corners
Text Effects	Text Content	Styling like uppercase, spacing, alignment, etc

---

📝 Recap Table:

Concept	Key Usage Example
`border`	`border: 2px solid red;`
`border-radius`	`border-radius: 10px;`
`box-shadow`	`box-shadow: 5px 5px 8px gray;`
`text-shadow`	`text-shadow: 2px 2px 5px black;`
Text Effects	`text-transform` , `letter-spacing` , `word-spacing`

📱 3.8 Basics of Responsive Web Design (RWD); Media Queries (Media Types, Viewport)

Responsive Web Design

🌟 What is Responsive Web Design?

Responsive Web Design means creating web pages that **adapt to different screen sizes and devices** (desktops, tablets, smartphones) without needing multiple versions of the site.

🚀 Key Principles of RWD:

Principle	Description
Flexible Layouts (Fluid Grid)	Uses percentages instead of fixed units like pixels. Layout adjusts as screen changes.
Flexible Media	Images and media scale within their containing elements.
Media Queries	Apply different CSS styles based on device's screen size, resolution, etc.

🏗️ Simple Responsive Layout Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    body {
      margin: 0;
    }
    .container {
      width: 90%;
      margin: auto;
    }
    .box {
      width: 100%;
      background-color: lightblue;
      padding: 20px;
      margin: 10px 0;
    }
  </style>
</head>
<body>

  <div class="container">
    <div class="box">Responsive Box 1</div>
    <div class="box">Responsive Box 2</div>
  </div>

</body>
</html>
```

Notice we used **percentages** for width → adjusts to screen width.

🎯 What are Media Queries?

Media Queries allow you to apply specific CSS **based on conditions like screen width, height, orientation, resolution, etc.**

Basic Syntax:

```
@media media-type and (condition) {
  /* CSS rules */
}
```

🔑 Common Conditions:

Condition	Description	Example
`min-width`	Minimum viewport width	`@media (min-width: 600px)`
`max-width`	Maximum viewport width	`@media (max-width: 768px)`
`orientation`	Device orientation (`portrait` or `landscape`)	`@media (orientation: portrait)`
`resolution`	Screen resolution	`@media (min-resolution: 300dpi)`

📏 Viewport Meta Tag:

Always include this tag for proper scaling on mobile devices:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Without this, the responsive CSS might not behave correctly on smaller screens!

📚 Media Types Overview:

Media Type	Description
`all`	Default, applies to all devices
`screen`	Computer screens, tablets, smartphones
`print`	For printed documents
`speech`	For screen readers

Example: Responsive Design with Media Queries:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    .box {
      width: 100%;
      padding: 20px;
      background-color: lightgreen;
      text-align: center;
    }

    /* For screens wider than 768px */
    @media (min-width: 768px) {
      .box {
        background-color: lightcoral;
        width: 50%;
      }
    }
  </style>
</head>
<body>

  <div class="box">Resize Me!</div>

</body>
</html>
```

🔍 **Explanation:**

- On mobile (less than 768px), the box is green and takes up 100% width.
- On larger screens (desktop/tablet), it becomes coral-colored and shrinks to 50%.

📝 Recap Table:

Concept	Description
Responsive Web Design	Adapts layout to various devices & screen sizes
Flexible Layouts	Use % units instead of fixed px
Flexible Media	Images & videos scale with container
Media Queries	Apply CSS rules based on device features (width, orientation, etc.)
Viewport Meta Tag	Essential for proper scaling on mobile
Common Conditions	`min-width`, `max-width`, `orientation`, `resolution`

🧩 Practice Task:

Try:

1. Create a `<div>` with a background color.
2. Use a media query to change the background color when screen width > 600px.
3. Make the width of the box 100% on small screens, and 50% on larger screens.

🚀 3.9 Introduction to Bootstrap (Basic Concepts and Installation)

Introduction to Bootstrap

🌟 What is Bootstrap?

Bootstrap is a **popular, open-source front-end framework** used to build responsive, mobile-first websites quickly and efficiently. It provides pre-designed **CSS classes, components, and JavaScript plugins**.

🔑 Key Features of Bootstrap:

Feature	Description
Responsive Grid System	12-column flexible grid for layout design
Pre-designed Components	Buttons, forms, navbars, modals, cards, etc.
Customizable	Can customize using Sass variables
Mobile-First Approach	Prioritizes mobile design, scales up to larger screens
Cross-browser Compatibility	Works across all major browsers
Built-in JavaScript Plugins	Includes jQuery-based components like dropdowns, carousels, modals

---

🏗️ Bootstrap File Structure Overview:

```
project-folder/
├── index.html
├── css/
│   └── bootstrap.min.css
├── js/
│   └── bootstrap.bundle.min.js
└── img/
```

🛠️ How to Install Bootstrap

1️⃣ Using CDN (Simplest Way):

Include Bootstrap via Content Delivery Network (CDN) links:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesh
eet">
</head>
<body>

  <button class="btn btn-primary">Bootstrap Button</button>

  <!-- Bootstrap JS Bundle -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></scri
pt>
</body>
</html>
```

✅ **Pros:** No download needed, easy to set up.
❌
Cons: Requires internet connection.

2️⃣ Download and Host Locally:

1. Go to **Bootstrap Official Website**
2. Download the compiled CSS and JS files.
3. Link them locally in your HTML:

```
<link rel="stylesheet" href="css/bootstrap.min.css">
<script src="js/bootstrap.bundle.min.js"></script>
```

✅ **Pros:** No dependency on external servers.
❌
Cons: Manual updates required.

3️⃣ Using Package Managers (Advanced Developers):

You can also install Bootstrap using:

Tool	Command

npm	`npm install bootstrap`
yarn	`yarn add bootstrap`

📐 Bootstrap Grid Example:

```
<div class="container">
  <div class="row">
    <div class="col-sm-6 bg-info">Column 1</div>
    <div class="col-sm-6 bg-warning">Column 2</div>
  </div>
</div>
```

Explanation:

- `.container` centers the content.
- `.row` creates a row.
- `.col-sm-6` divides row into two columns, each 50% width on small and larger screens.

📝 Recap Table:

Concept	Description
Bootstrap	Front-end CSS framework for responsive design
Installation (CDN)	Quick setup using Bootstrap's hosted CSS/JS files
Installation (Download)	Download Bootstrap files locally
Grid System	12-column flexible layout
Pre-built Components	Buttons, forms, navbars, modals, etc.
Mobile-First	Starts with mobile design, scales up
