

## ****Part 1: Introduction to CSS****

### ****1. What is CSS?****

**CSS (Cascading Style Sheets)** is a language used to style HTML elements. While **HTML** provides the structure of a webpage, **CSS controls the appearance and layout.**

📌 **Example without CSS (Plain HTML):**

<!DOCTYPE html>

<html lang="en">

<head>

<title>My First Page</title>

</head>

<body>

<h1>Hello, World!</h1>

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

</body>

</html>

👉 The text appears **unstyled**, with a default black color and basic font.

📌 **Example with CSS (Styled HTML):**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Styled Page</title>

<style>

h1 { color: blue; }

p { font-size: 18px; color: green; }

</style>

</head>

<body>

<h1>Hello, World!</h1>

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

</body>

</html>

✅ Now, the heading is **blue**, and the paragraph text is **green and larger**.

### ****2. How Does CSS Work?****

CSS works by **selecting HTML elements** and **applying styles** to them. The basic structure of a CSS rule is:

selector {

property: value;

}

🔹 **Selector** → Targets an HTML element (e.g., h1, p, .class, #id).  
🔹 **Property** → The style you want to apply (e.g., color, font-size).  
🔹 **Value** → The setting for the property (e.g., blue, 16px).

📌 **Example:**

p {

color: red;

font-size: 20px;

}

👉 This will make all <p> elements red with a font size of 20px.

### ****3. Ways to Apply CSS****

There are **three main ways** to apply CSS to an HTML document:

#### ****(A) Inline CSS (Applied directly to an element)****

Inline CSS is written **inside the HTML tag** using the style attribute.  
📌 **Example:**

<p style="color: red; font-size: 20px;">This is a red paragraph.</p>

✅ **Pros:** Quick and easy for small changes.  
❌ **Cons:** Not reusable and makes the HTML file messy.

#### ****(B) Internal CSS (Written inside**** <style> ****in**** <head>****)****

Internal CSS is written inside a <style> block within the <head> section of an HTML file.

📌 **Example:**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Internal CSS Example</title>

<style>

h1 { color: blue; }

p { font-size: 18px; color: green; }

</style>

</head>

<body>

<h1>Hello, World!</h1>

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

</body>

</html>

✅ **Pros:** Keeps CSS separate from HTML elements.  
❌ **Cons:** Not reusable across multiple pages.

#### ****(C) External CSS (Best Practice – Linked in a Separate File)****

External CSS is written in a separate file (style.css) and linked to the HTML file using the <link> tag inside <head>.

📌 **Steps to Use External CSS:**  
1️⃣ **Create an HTML file (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<title>External CSS Example</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<h1>Hello, World!</h1>

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

</body>

</html>

2️⃣ **Create a CSS file (style.css)**

h1 {

color: blue;

}

p {

font-size: 18px;

color: green;

}

✅ **Pros:** Keeps CSS completely separate, making code cleaner and easier to manage.  
❌ **Cons:** Requires an additional file.

### ****4. Activity: Testing the Different CSS Methods****

🛠 **Objective:** Students apply inline, internal, and external CSS on a simple HTML page.

1️⃣ **Step 1: Create an HTML page (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<title>CSS Test</title>

</head>

<body>

<h1 style="color: red;">This is Inline CSS</h1>

<p>This paragraph will be styled using Internal CSS.</p>

<div>This div will be styled using External CSS.</div>

</body>

</html>

2️⃣ **Step 2: Add Internal CSS inside <head>**

<style>

p {

color: blue;

font-size: 18px;

}

</style>

3️⃣ **Step 3: Create an External CSS file (style.css)**

div {

color: green;

font-weight: bold;

}

4️⃣ **Step 4: Link the External CSS in <head>**

<link rel="stylesheet" href="style.css">

✅ **Expected Output:**

* The **heading (h1)** should be **red** (inline CSS).
* The **paragraph (p)** should be **blue and bigger** (internal CSS).
* The **div text** should be **green and bold** (external CSS).

### ****5. Recap & Key Takeaways****

✅ CSS is used to **style** HTML pages.  
✅ **Three ways to apply CSS:** Inline, Internal, and External.  
✅ **Best practice:** Use **External CSS** for organized and reusable code.

# ****Part 2: CSS Selectors and Basic Styling****

## ****1. What are CSS Selectors?****

**CSS selectors** allow us to target specific HTML elements and apply styles to them. They help in controlling the look and feel of different parts of a webpage.

### ****2. Types of CSS Selectors****

There are different types of CSS selectors, each serving a unique purpose. Let's go through the most commonly used ones.

### ****(A) Universal Selector (****\*****)****

🔹 This applies styles to **all elements** on a webpage.  
📌 **Example:**

\* {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

}

👉 This sets the font for all elements and removes default margins and padding.

### ****(B) Element Selector (****tagname****)****

🔹 This applies styles to **all occurrences of a specific HTML tag**.  
📌 **Example:**

h1 {

color: blue;

}

p {

font-size: 18px;

}

👉 All <h1> elements will be blue, and all <p> elements will have a font size of 18px.

### ****(C) Class Selector (****.classname****)****

🔹 Targets elements with a **specific class**.  
📌 **Example:**

.alert {

color: red;

font-weight: bold;

}

📌 **Usage in HTML:**

html

CopyEdit

<p class="alert">This is an important message!</p>

👉 Any element with the alert class will have **red, bold text**.

### ****(D) ID Selector (****#idname****)****

🔹 Targets a **specific element** using an **ID** (must be unique per page).  
📌 **Example:**

#main-heading {

color: green;

text-align: center;

}

📌 **Usage in HTML:**

<h1 id="main-heading">Welcome to My Website</h1>

👉 The heading with id="main-heading" will be **green and centered**.

🔹 **Difference Between Class and ID:**

| **Feature** | **.class (Class Selector)** | **#id (ID Selector)** |
| --- | --- | --- |
| Usage | Can be applied to multiple elements | Should be unique (one per page) |
| Syntax | .classname {} | #idname {} |

### ****(E) Grouping Selector (****selector1, selector2****)****

🔹 Used to style **multiple elements at once**.  
📌 **Example:**

h1, h2, p {

color: darkblue;

}

👉 **All <h1>, <h2>, and <p>** elements will have **dark blue text**.

### ****(F) Descendant Selector (****parent child****)****

🔹 Selects elements **inside a specific parent element**.  
📌 **Example:**

div p {

color: purple;

}

📌 **Usage in HTML:**

<div>

<p>This paragraph will be purple.</p>

</div>

<p>This paragraph outside the div will not be affected.</p>

👉 **Only <p> inside a <div> will turn purple**.

### ****(G) Pseudo-Classes (e.g.,**** :hover****,**** :first-child****)****

🔹 Used to define **special states** of an element.

📌 **Example (:hover - Changes style when the user hovers over an element):**

button:hover {

background-color: darkblue;

color: white;

}

📌 **Usage in HTML:**

<button>Hover over me!</button>

👉 The button’s background changes to **dark blue** when hovered.

📌 **Example (:first-child - Targets the first child element inside a parent):**

p:first-child {

color: red;

}

📌 **Usage in HTML:**

<div>

<p>This will be red.</p>

<p>This will not be affected.</p>

</div>

👉 **Only the first <p> inside <div> turns red**.

### ****3. Activity: Practicing CSS Selectors****

🛠 **Objective:** Students apply different selectors to style a webpage.

📌 **Step 1: HTML Structure (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>CSS Selectors Practice</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<h1 id="main-heading">Welcome to CSS Selectors</h1>

<p class="intro">This paragraph has a class applied.</p>

<p>This paragraph does not have a class.</p>

<div class="container">

<p>This paragraph is inside a div.</p>

</div>

<button>Hover over me</button>

</body>

</html>

📌 **Step 2: CSS Styling (style.css)**

/\* Universal Selector \*/

\* {

font-family: Arial, sans-serif;

}

/\* Element Selector \*/

h1 {

color: blue;

}

/\* Class Selector \*/

.intro {

color: green;

font-weight: bold;

}

/\* ID Selector \*/

#main-heading {

text-align: center;

font-size: 24px;

}

/\* Descendant Selector \*/

.container p {

color: purple;

}

/\* Pseudo-Class \*/

button:hover {

background-color: darkblue;

color: white;

}

✅ **Expected Output:**

* The **h1** will be **blue** and **centered**.
* The paragraph with .intro class will be **green and bold**.
* The paragraph inside .container will be **purple**.
* The button will change **color on hover**.

### ****4. Recap & Key Takeaways****

✅ **Selectors help target elements** in HTML to apply styles.  
✅ The most important selectors are **Element, Class, ID, Grouping, Descendant, and Pseudo-Classes**.  
✅ **Best practice:** Use **class selectors** for reusable styles instead of IDs.

# ****Part 3: CSS Box Model and Layout (40 Minutes)****

## ****1. Introduction to the CSS Box Model****

Every HTML element is treated as a **box** in CSS. The **CSS Box Model** is a fundamental concept that helps in understanding how elements are sized and spaced on a webpage.

### ****The Box Model Components****

Each element consists of **four layers** from the inside out:

1. **Content** – The actual text or image inside the box.
2. **Padding** – Space **between the content and the border**.
3. **Border** – A line surrounding the element.
4. **Margin** – Space **outside the border** that separates elements.

📌 **Visual Representation:**

+------------------------------+ <-- Margin (Outer Space)

| +------------------------+ | <-- Border

| | +------------------+ | | <-- Padding

| | | Content Area | | | <-- Content

| | +------------------+ | |

| +------------------------+ |

+------------------------------+

## ****2. Understanding Each Part of the Box Model****

### ****(A) Content****

This is where text, images, or other elements are displayed.

📌 **Example:**

<div class="box">This is the content.</div>

.box {

width: 200px;

height: 100px;

background-color: lightblue;

}

👉 The content takes **200px width** and **100px height**.

### ****(B) Padding**** (Space Inside the Box)

🔹 padding adds space **inside the element** between the content and the border.  
📌 **Example:**

.box {

padding: 20px;

}

👉 The content gets **20px of space around it inside the border**.

🔹 **Padding can be set for each side separately:**

.box {

padding-top: 10px;

padding-right: 15px;

padding-bottom: 20px;

padding-left: 25px;

}

👉 Instead of writing four lines, we can shorthand:

.box {

padding: 10px 15px 20px 25px; /\* Top, Right, Bottom, Left \*/

}

### ****(C) Border**** (Outer Edge of the Box)

🔹 Borders are used to **wrap around an element**.  
📌 **Example:**

.box {

border: 5px solid black;

}

👉 This creates a **5px thick black border**.

🔹 **Border Style Options:**

|  |  |
| --- | --- |
| **Style** | **Example** |
| Solid | border: 2px solid black; |
| Dashed | border: 2px dashed black; |
| Dotted | border: 2px dotted black; |
| Double | border: 4px double black; |
| Groove | border: 3px groove black; |

### ****(D) Margin**** (Space Between Elements)

🔹 margin **creates space around the element, outside the border**.  
📌 **Example:**

.box {

margin: 30px;

}

👉 Adds **30px space around the box**.

🔹 **Margin can be set individually for each side:**

.box {

margin-top: 10px;

margin-right: 15px;

margin-bottom: 20px;

margin-left: 25px;

}

👉 Shorthand:

.box {

margin: 10px 15px 20px 25px; /\* Top, Right, Bottom, Left \*/

}

## ****3. Box Model in Action****

📌 **Example:**

<div class="box">This is a box model example.</div>

css

CopyEdit

.box {

width: 200px;

height: 100px;

padding: 20px;

border: 5px solid black;

margin: 30px;

background-color: lightblue;

}

👉 **How the total width is calculated?**

mathematica

CopyEdit

Total Width = Width + Padding + Border + Margin

= 200px + (20px \* 2) + (5px \* 2) + (30px \* 2)

= 310px

## ****4. Activity: Box Model Experiment****

🛠 **Objective:** Adjust padding, margin, and border to see how the box changes.

📌 **Step 1: HTML Structure (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>CSS Box Model Practice</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="box">Experiment with Box Model</div>

</body>

</html>

📌 **Step 2: CSS Styling (style.css)**

.box {

width: 250px;

height: 120px;

padding: 15px;

border: 3px solid red;

margin: 20px;

background-color: yellow;

text-align: center;

font-weight: bold;

}

✅ **Expected Output:**

* A **yellow box** with a **red border**, **15px padding**, and **20px margin**.
* The text inside is **centered and bold**.

## ****5. Advanced Topic:**** box-sizing ****Property****

By default, **padding and border increase an element’s size**. We can **fix this issue** using box-sizing.

📌 **Example:**

.box {

width: 200px;

height: 100px;

padding: 20px;

border: 5px solid black;

box-sizing: border-box;

}

👉 box-sizing: border-box; **ensures the total width remains 200px** instead of increasing due to padding and border.

## ****6. Recap & Key Takeaways****

✅ Every element follows the **Box Model** (Content, Padding, Border, Margin).  
✅ Use margin to control **spacing between elements**.  
✅ Use padding to create **space inside the element**.  
✅ Use box-sizing: border-box; to **keep consistent element sizes**.

# ****Part 4: CSS Flexbox – Responsive Layouts****

## ****1. Introduction to Flexbox****

**Flexbox (Flexible Box Layout)** is a CSS layout model that helps design **responsive web pages** by making it easy to align and distribute elements efficiently.

🔹 **Why use Flexbox?**  
✔ Makes **alignment** and **positioning** of elements easier  
✔ Helps create **responsive designs**  
✔ Eliminates the need for **complex floats or positioning**

## ****2. Understanding Flexbox Terminology****

Flexbox consists of **two main components**:

1. **Flex Container** – The parent element that holds flex items.
2. **Flex Items** – The child elements inside the container.

📌 **Example: Basic Flexbox Structure**

<div class="container">

<div class="box">1</div>

<div class="box">2</div>

<div class="box">3</div>

</div>

css

CopyEdit

.container {

display: flex;

}

👉 By setting display: flex;, all child elements **become flex items**.

## ****3. Main Properties of Flexbox****

### ****(A) display: flex****

Defines a **flex container**, making all children **flex items**.

.container {

display: flex;

}

### ****(B) flex-direction**** (Row vs. Column)

Determines how items are arranged in the flex container.

|  |  |
| --- | --- |
| **Value** | **Description** |
| row (default) | Items placed **horizontally (left to right)** |
| row-reverse | Items placed **horizontally (right to left)** |
| column | Items placed **vertically (top to bottom)** |
| column-reverse | Items placed **vertically (bottom to top)** |

📌 **Example: Arranging items in a column**

.container {

display: flex;

flex-direction: column;

}

### ****(C) justify-content**** (Horizontal Alignment)

Controls **alignment along the main axis**.

|  |  |
| --- | --- |
| **Value** | **Description** |
| flex-start | Items align **at the beginning** (default) |
| flex-end | Items align **at the end** |
| center | Items align **at the center** |
| space-between | **First and last item at edges, space in between** |
| space-around | **Equal space around all items** |

📌 **Example: Centering items horizontally**

.container {

display: flex;

justify-content: center;

}

### ****(D) align-items**** (Vertical Alignment)

Controls **alignment along the cross-axis**.

|  |  |
| --- | --- |
| **Value** | **Description** |
| stretch (default) | Items stretch to fill the container |
| flex-start | Items align **at the top** |
| flex-end | Items align **at the bottom** |
| center | Items align **in the middle** |
| baseline | Items align **by their text baseline** |

📌 **Example: Centering items vertically**

.container {

display: flex;

align-items: center;

}

### ****(E) flex-wrap**** (Wrap Items in Multiple Lines)

By default, flexbox **keeps all items in one line**.  
We can allow items to **wrap into multiple lines** using flex-wrap.

|  |  |
| --- | --- |
| **Value** | **Description** |
| nowrap | Items stay **on a single line** (default) |
| wrap | Items move **to the next line if needed** |
| wrap-reverse | Items **wrap in reverse order** |

📌 **Example: Allowing items to wrap**

.container {

display: flex;

flex-wrap: wrap;

}

### ****(F) gap**** (Spacing Between Items)

Instead of using margins, we can create spacing **between flex items** using gap.

📌 **Example: Adding space between items**

css

CopyEdit

.container {

display: flex;

gap: 20px;

}

### ****(G) flex-grow**** (Item Expansion)

Defines **how much an item should grow** relative to others.

* **Default value** = 0 (items don’t grow).
* Higher values mean **more space** is given to that item.

📌 **Example: Making one item take up extra space**

.box:nth-child(2) {

flex-grow: 2; /\* Grows twice as much as others \*/

}

### ****(H) flex-shrink**** (Item Reduction)

Defines **how much an item should shrink** when there’s not enough space.

* **Default value** = 1 (shrinks normally).
* 0 means the item **will not shrink**.

📌 **Example: Preventing an item from shrinking**

.box:nth-child(2) {

flex-shrink: 0;

}

## ****4. Activity: Responsive Card Layout****

🛠 **Objective:** Create a responsive **flexbox-based card layout**.

📌 **Step 1: HTML Structure (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Flexbox Practice</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="container">

<div class="card">Card 1</div>

<div class="card">Card 2</div>

<div class="card">Card 3</div>

</div>

</body>

</html>

📌 **Step 2: CSS Styling (style.css)**

/\* Flexbox Container \*/

.container {

display: flex;

justify-content: space-around;

align-items: center;

flex-wrap: wrap;

gap: 20px;

}

/\* Card Styling \*/

.card {

width: 200px;

height: 150px;

background-color: lightblue;

display: flex;

justify-content: center;

align-items: center;

font-size: 18px;

font-weight: bold;

border: 2px solid navy;

border-radius: 10px;

}

✅ **Expected Output:**

* **Three cards** evenly spaced, centered inside the container.
* Cards adjust when resized (flex-wrap enabled).

## ****5. Recap & Key Takeaways****

✅ Flexbox simplifies **alignment and layout control**.  
✅ justify-content aligns items **horizontally**.  
✅ align-items aligns items **vertically**.  
✅ flex-wrap allows items to **wrap onto multiple lines**.  
✅ gap adds **spacing between elements**.

# ****📌 Part 5: CSS Grid – Advanced Layouts (40 Minutes)****

## ****1. Introduction to CSS Grid****

**CSS Grid Layout** is a powerful system for **creating complex, responsive web layouts** with precise control over rows and columns.

🔹 **Why use CSS Grid?**  
✔ Creates **multi-dimensional layouts** (both rows and columns)  
✔ More control over **placement of elements**  
✔ Simplifies **complex designs**

## ****2. Understanding CSS Grid Terminology****

CSS Grid consists of **two main components**:

1. **Grid Container** – The parent element that defines the grid.
2. **Grid Items** – The child elements placed inside the grid.

📌 **Example: Basic Grid Structure**

<div class="grid-container">

<div class="grid-item">1</div>

<div class="grid-item">2</div>

<div class="grid-item">3</div>

<div class="grid-item">4</div>

</div>

css

CopyEdit

.grid-container {

display: grid;

}

👉 By setting display: grid;, all child elements **become grid items**.

## ****3. Defining Columns and Rows****

CSS Grid allows us to define **grid columns and rows** using grid-template-columns and grid-template-rows.

📌 **Example: Creating a 2-column, 2-row grid**

.grid-container {

display: grid;

grid-template-columns: 200px 200px;

grid-template-rows: 100px 100px;

}

🔹 This creates **two columns (200px each)** and **two rows (100px each)**.

### ****(A) Using**** fr ****for Flexible Sizing****

Instead of fixed sizes (px), we can use **fractional units (fr)** to make the grid responsive.

📌 **Example: Creating a flexible grid**

.grid-container {

display: grid;

grid-template-columns: 1fr 2fr;

grid-template-rows: 1fr 1fr;

}

🔹 The first column takes 1fr (one part), while the second takes 2fr (twice as much space).

### ****(B)**** grid-gap ****(Spacing Between Grid Items)****

We can add spacing **between rows and columns** using gap.

📌 **Example: Adding gaps**

.grid-container {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 20px;

}

🔹 This creates **three equal columns** with a 20px gap between them.

### ****(C) Spanning Items Across Multiple Columns/Rows****

We can make grid items span **multiple columns or rows** using:

* grid-column: start / end;
* grid-row: start / end;

📌 **Example: Making an item span two columns**

.grid-item:nth-child(1) {

grid-column: span 2;

}

🔹 The first item will **stretch across two columns**.

## ****4. Activity: Responsive Grid Layout****

🛠 **Objective:** Create a **simple webpage layout using CSS Grid**.

📌 **Step 1: HTML Structure (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Grid Practice</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<div class="grid-container">

<div class="header">Header</div>

<div class="sidebar">Sidebar</div>

<div class="content">Content</div>

<div class="footer">Footer</div>

</div>

</body>

</html>

📌 **Step 2: CSS Styling (style.css)**

/\* Grid Layout \*/

.grid-container {

display: grid;

grid-template-columns: 1fr 3fr;

grid-template-rows: auto auto auto;

gap: 20px;

}

/\* Header, Sidebar, Content, Footer \*/

.header {

grid-column: span 2;

background-color: lightblue;

text-align: center;

padding: 20px;

font-size: 20px;

font-weight: bold;

}

.sidebar {

background-color: lightgray;

padding: 20px;

}

.content {

background-color: white;

padding: 20px;

border: 2px solid black;

}

.footer {

grid-column: span 2;

background-color: lightblue;

text-align: center;

padding: 10px;

}

✅ **Expected Output:**

* **Header spans across two columns**
* **Sidebar on the left, content on the right**
* **Footer spans across two columns**

## ****5. Recap & Key Takeaways****

✅ grid-template-columns defines **columns**.  
✅ grid-template-rows defines **rows**.  
✅ gap adds **spacing** between grid items.  
✅ grid-column and grid-row control **spanning** across multiple columns/rows.