**The University of Azad Jammu and Kashmir, Muzaffarabad**



**Web Design and Development (WDD) Lab**

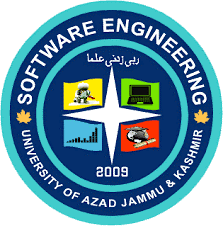
**(SE-3105)**

**Lab Manual**

**Software Engineering**

**(III Year – SEM VI)**

**(2022-26)**

****

**Department of**

**Software Engineering**

**Vision:**

* To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

**Mission:**

* To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers.
* Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

**Program Learning Outcomes (PLOs)**

**General Lab Instructions**

Contents

[Lab 1 9](#_Toc198508422)

[1. Topics Covered 9](#_Toc198508423)

[Lab 2 9](#_Toc198508424)

[1. Topics Covered 9](#_Toc198508425)

[Lab 3 9](#_Toc198508426)

[1. Topics Covered 9](#_Toc198508427)

[Lab 4 10](#_Toc198508428)

[2. Topics Covered 10](#_Toc198508429)

[Lab 1 11](#_Toc198508430)

[What Is the World Wide Web (WWW)? 11](#_Toc198508431)

[Key Terms Made Simple 11](#_Toc198508432)

[What is HTML5? 11](#_Toc198508433)

[Daily Life Example 12](#_Toc198508434)

[Lab 2 19](#_Toc198508435)

[1. Unordered List (<ul>) 19](#_Toc198508436)

[2. Ordered List (<ol>) 19](#_Toc198508437)

[3. Description List (<dl>) 20](#_Toc198508438)

[Nesting Lists 20](#_Toc198508439)

[Form Submission Workflow 21](#_Toc198508440)

[Client-Side vs. Server-Side 21](#_Toc198508441)

[HTML Forms – Complete Overview 22](#_Toc198508442)

[1. Key <form> Tag Attributes 22](#_Toc198508443)

[2. GET vs POST 22](#_Toc198508444)

[3. Basic Form Controls 22](#_Toc198508445)

[4. Input Types (With Attributes & Description) 23](#_Toc198508446)

[5. Select Dropdown (Single and Multiple Options) 23](#_Toc198508447)

[6. Radio Buttons vs Checkboxes 24](#_Toc198508448)

[7. Basic Inline Validation (No JavaScript) 24](#_Toc198508449)

[CSS Selectors 24](#_Toc198508450)

[1. Tag (Element) Selector 24](#_Toc198508451)

[2. Class Selector (.class) 24](#_Toc198508452)

[3. ID Selector (#id) 25](#_Toc198508453)

[4. Group Selector 25](#_Toc198508454)

[5. Descendant Selector 25](#_Toc198508455)

[6. Child Selector (>) 25](#_Toc198508456)

[7. Universal Selector (\*) 26](#_Toc198508457)

[8. Attribute Selector 26](#_Toc198508458)

[9. Pseudo-class Selector (Intro Only) 26](#_Toc198508459)

[Selector Summary Table 26](#_Toc198508460)

[CSS Float 27](#_Toc198508461)

[1. Syntax of float 27](#_Toc198508462)

[2. Behavior of Floated Elements 27](#_Toc198508463)

[3. Clear Property 27](#_Toc198508464)

[Layout Creation with Float 27](#_Toc198508465)

[6. Limitations of Float Layouts 28](#_Toc198508466)

[Lab 3 29](#_Toc198508467)

[CSS Flexbox 29](#_Toc198508468)

[2. Flexbox Structure 29](#_Toc198508469)

[3. Flex Container Properties (Parent Only) 29](#_Toc198508470)

[A. display: flex 29](#_Toc198508471)

[B. flex-direction 29](#_Toc198508472)

[C. flex-wrap 29](#_Toc198508473)

[D. justify-content 30](#_Toc198508474)

[E. align-items 30](#_Toc198508475)

[F. align-content 30](#_Toc198508476)

[G. gap 31](#_Toc198508477)

[4. Flex Item Properties (Children Only) 31](#_Toc198508478)

[A. flex-grow 31](#_Toc198508479)

[B. flex-shrink 31](#_Toc198508480)

[C. flex-basis 31](#_Toc198508481)

[D. flex (Shorthand) 32](#_Toc198508482)

[E. align-self 32](#_Toc198508483)

[CSS Box Model 32](#_Toc198508484)

[Definition 32](#_Toc198508485)

[Structure (Inside to Outside) 33](#_Toc198508486)

[Width and Height Calculation 33](#_Toc198508487)

[Box-Sizing Property 33](#_Toc198508488)

[Example Comparison 34](#_Toc198508489)

[HTML Size Units 34](#_Toc198508490)

[1. Absolute Units 34](#_Toc198508491)

[2. Relative Units 34](#_Toc198508492)

[Microsoft Clarity 35](#_Toc198508493)

[Why it's useful 35](#_Toc198508494)

[How to Use 35](#_Toc198508495)

[Google Analytics 35](#_Toc198508496)

[What it Does 35](#_Toc198508497)

[How to Set Up 35](#_Toc198508498)

[Overlay Gradient 36](#_Toc198508499)

[Usage Example (CSS) 36](#_Toc198508500)

[Explanation 36](#_Toc198508501)

[Lab 4 37](#_Toc198508502)

[What is CSS Grid? 37](#_Toc198508503)

[Basic Terminology 37](#_Toc198508504)

[Creating a Grid Container 37](#_Toc198508505)

[1. Grid Template: Rows and Columns 37](#_Toc198508506)

[2. Grid Gaps 38](#_Toc198508507)

[3. Naming Grid Lines 38](#_Toc198508508)

[4. Placing Items with Line Numbers 38](#_Toc198508509)

[5. Placing Items with grid-column-start, grid-column-end, etc. 38](#_Toc198508510)

[6. Spanning Multiple Rows or Columns 38](#_Toc198508511)

[7. Grid Auto Flow 39](#_Toc198508512)

[8. Auto Rows and Auto Columns 39](#_Toc198508513)

[10. justify-items and align-items 39](#_Toc198508514)

[11. justify-content and align-content 39](#_Toc198508515)

[12. justify-self and align-self 40](#_Toc198508516)

[Comparison of Table, Float, Flexbox, and Grid Layouts 40](#_Toc198508517)

[1. CSS Specificity & Precedence 41](#_Toc198508518)

[Precedence Order (Highest to Lowest) 41](#_Toc198508519)

[Specificity Rules (for selectors) 41](#_Toc198508520)

[Image Attributes (Corrected and Complete) 42](#_Toc198508521)

[1. src (File Path) 42](#_Toc198508522)

[2. alt (Alternative Text) 42](#_Toc198508523)

[3. width and height 42](#_Toc198508524)

[4. object-fit 42](#_Toc198508525)

[5. object-position 43](#_Toc198508526)

[CSS Positioning 43](#_Toc198508527)

[Types of Positioning in CSS 43](#_Toc198508528)

[Offset Properties 44](#_Toc198508529)

[Stacking Elements: z-index 44](#_Toc198508530)

# Lab 1

## Topics Covered

* Working of WWW.
* Introduction to HTML5 and basic web page structure in detail explain
* HTML Tags inline, block level, empty, Attributes - Using common HTML tags (headings, paragraphs, images, picture, links)
* Code execution flow
* Designing page layouts with tables – email example
* Applying table attributes (rows pan, cols pan, borders)
* Styling web pages using internal CSS

# Lab 2

## Topics Covered

* Lists
* HTML form working completely explain
* All about HTML Forms (form attributes get, post, validation, action, input types, select, radio, checkbox, basic inline validation,)
* CSS Selectors: ID (#id) and Class (.class) usage, tag other ways to select
* Layout creation using CSS Float property

# Lab 3

## Topics Covered

* Flex
* Flex term (align items, align content, justify item, justify content, align self, align content, flex wrap, flex grow, flex shrink, flex basis, flex gap)
* Layout creation using flex
* CSS Box Model
* Html size units
* Microsoft clarity
* What fonts
* Google analytics
* Google fonts
* Overlay gradient

# Lab 4

## Topics Covered

* CSS selectors and there specifity
* Image attributes (object-fit, positioning, alt, width, height, file path)
* Grid
* Grid styling (all that were in flex, fr, auto row and columns width)
* Layout creation using Grid
* CSS Positioning

# Lab 5

## Topics Covered

* Transform (translate, rotate, scale etc)
* Translate
* Animate (keyframes)
* Media querries

# Lab 1

## ****What Is the World Wide Web (WWW)?****

The **World Wide Web (WWW)** is like a huge **online library** full of websites and web pages. You can open any page using a **browser** (like Chrome or Firefox), just like opening a book to read.

The **Internet** is the road system that connects computers all over the world. The **WWW** is the collection of websites that travel on those roads.

## ****Key Terms Made Simple****

|  |  |  |
| --- | --- | --- |
| Term | What It Means | Daily Life Example |
| **Browser** | App to open websites (Chrome, Firefox, etc.) | Your phone used to call and order pizza |
| **Editor/IDE** | Where you actually write code | Notepad, VS Code |
| **URL** | Website address (e.g., [www.google.com](http://www.google.com)) | Phone number of the pizza shop |
| **DNS** | Finds real server address from website name | Operator who gives the shop's location |
| **Web Server** | Stores and sends web pages | The pizza shop that prepares the pizza |
| **HTTP/HTTPS** | Rules for sending/receiving websites | The delivery guy who brings the pizza |
| **IP Address** | Actual computer address of the website | House number of the pizza shop |

## ****What is HTML5?****

**HTML5** stands for **Hyper Text Markup Language version 5**.  
It is the **main language** used to create **web pages**.

* Think of HTML like the **skeleton** of a human body — it gives the structure.
* It tells the browser **what to display**: headings, text, images, links, etc.

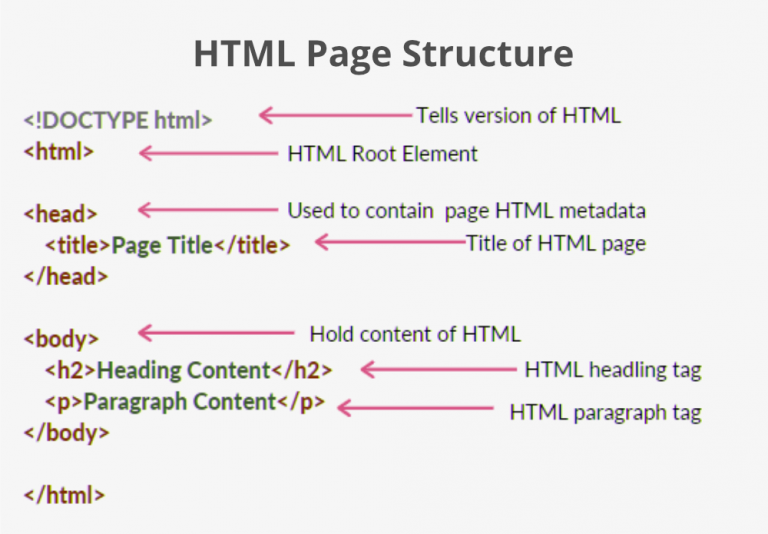
**HTML5** is the latest version. It is faster, cleaner, and supports modern features like video, audio, animations, and more — **no extra plugins** needed.

## ****Daily Life Example****

Imagine creating a **poster** for an event:

* **HTML** is like arranging the **text and pictures** on the poster.
* **CSS** would be the **colors, fonts, and layout styles**.
* **JavaScript** would make the poster **interactive** (e.g., blinking text or pop-ups).

**Basic Structure of an HTML5 Page**



|  |  |  |
| --- | --- | --- |
| Part | What It Means | Example / Analogy |
| <!DOCTYPE html> | Tells the browser: “This is an HTML5 document” | Like saying “This is an English letter” |
| <html> | The main container for the entire web page | Like the envelope holding everything |
| <head> | Contains info about the page (not visible on the page) | Like the behind-the-scenes setup: title, styles |
| <title> | The title shown in the browser tab | Like the title of a book |
| <body> | The visible content on the page | Like the actual content inside the book |
| <h1> to <h6> | Headings (from big to small) | Like section titles in a report |
| <p> | Paragraph of text | Like normal written content |

**HTML Tags and Attributes**

**What Are HTML Tags?**

HTML (HyperText Markup Language) uses tags to define the structure and content of web pages. Tags are written using angle brackets like <p>, <h1>, etc.

Tags usually come in pairs:

* Opening tag: <tagname>
* Closing tag: </tagname>

Some tags do not need a closing tag. These are called empty tags, such as <br> and <img>.

**Types of HTML Tags**

1. **Block-level Tags**

* Begin on a new line.
* Occupy the full width available.
* Can contain both inline and block elements.
* Examples: <div>, <p>, <h1> to <h6>, <section>, <article>, <form>, <header>, <footer>, <ul>, <ol>, <li>

1. **Inline Tags**

* Do not start on a new line.
* Take up only as much width as necessary.
* Usually used inside block-level elements.
* Examples: <a>, <span>, <img>, <strong>, <em>, <label>, <abbr>, <input>

1. **Empty Tags**

* Do not wrap any content and do not require a closing tag.
* Examples: <br>, <hr>, <img>, <input>, <meta>, <link>

**HTML Attributes**

Attributes provide additional information about elements. They are written inside the opening tag using the format:

Attribute="value"

* Multiple attributes are separated by spaces.
* Quotation marks are required around attribute values.
* Common attributes include: id, class, style, title, href, src, alt, target

**Using Common HTML Tags**

**1. Headings (<h1> to <h6>)**

* Define headings or titles.
* <h1> is the most important; <h6> is the least.
* Only one <h1> should be used per page for SEO and semantic clarity.

**2. Paragraph (<p>)**

* Represents a block of text.
* Automatically adds space before and after the paragraph.

**3. Image (<img>)**

* Used to insert images on a web page.
* It is an empty tag and does not have a closing tag.

Common attributes for <img>:

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Required |
| src | Specifies the path or URL of the image | Yes |
| alt | Alternative text for accessibility and SEO | Yes |
| width | Width of the image in pixels or percent | No |
| height | Height of the image in pixels or percent | No |
| title | Tooltip shown on hover | No |

**4. Picture (<picture>)**

* A container element used to display different images based on screen size or resolution.
* Contains multiple <source> elements with media conditions and one fallback <img>.

|  |  |
| --- | --- |
| Element | Purpose |
| <picture> | Wraps the image sources |
| <source> | Specifies media conditions and image files |
| <img> | Fallback image shown if no media condition is met |

**5. Link (<a>)**

* Used to create hyperlinks to other web pages, files, or email addresses.

Common attributes for <a>:

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Required |
| href | Specifies the link's destination | Yes |
| target | Defines where to open the link (\_self, \_blank, etc.) | No |
| title | Text shown as tooltip on hover | No |
| download | Prompts the user to download the linked file | No |

**Code Execution Flow**

The browser reads and executes HTML code **from top to bottom**.

1. It starts with the <! DOCTYPE html> declaration to understand the HTML version.
2. It then processes the <head> section first, including Meta tags, title, CSS links, and scripts marked as defer.
3. The <body> content is parsed and rendered next.
4. Scripts without defer or sync are executed immediately, which can block rendering.
5. **HTML Tables**
6. Tables are used in HTML to display data in a structured format using rows and columns. Tables can also be used to design layouts where precise control of alignment is needed, especially in older or restricted environments like email clients.

**Basic Structure of a Table**

A table is defined using the <table> element. It is made up of several child elements:

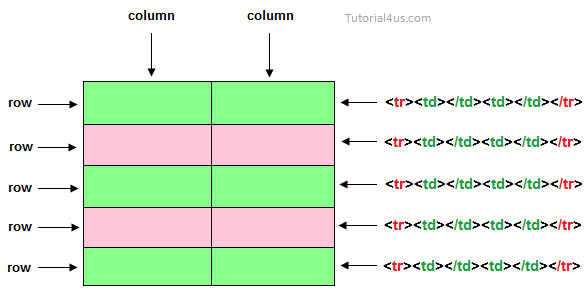
|  |  |
| --- | --- |
| Tag | Purpose |
| <table> | Starts the table |
| <tr> | Table row – contains one or more cells |
| <td> | Table data – defines a standard cell |
| <th> | Table header – bold and centered by default |
| <thead> | Groups the header content (optional) |
| <tbody> | Groups the body content (optional) |
| <tfoot> | Groups the footer content (optional) |
| <caption> | Adds a title or description for the table |

**Attributes Commonly Used in Tables**

|  |  |  |
| --- | --- | --- |
| Attribute | Element | Description |
| border | <table> | Sets the border width of the table |
| cellpadding | <table> | Adds space inside each cell |
| cellspacing | <table> | Adds space between cells |
| colspan | <td>, <th> | Merges columns across multiple cells |
| rowspan | <td>, <th> | Merges rows across multiple cells |
| width | All elements | Sets width (can be in % or px) |
| align | <table>, <td> | Aligns table or content (left, center, right) |
| valign | <td> | Vertical alignment (top, middle, bottom) |
| style | Any | Allows inline CSS for better control |

Note: cellpadding, cellspacing, align, and valign are outdated in modern HTML but still widely used in **email layouts** for compatibility.

**Table Example Structure**



**When to Use <th> Instead of <td>**

Use <th> for table headings. It helps with **semantics** and **accessibility** and also improves SEO for data tables on web pages.

**1. Colspan (Column Span)**

* Used in <td> or <th>.
* Merges **multiple columns** into a single cell.
* Accepts a numeric value (e.g., colspan="2").

**Usage Example**  
if a single heading needs to span across 2 columns:

< th colspan="2">Student Info</th>

**2. Rowspan (Row Span)**

* Used in <td> or <th>.
* Merges **multiple rows** into one cell vertically.
* Accepts a numeric value (e.g., rowspan="3").

**Usage Example**  
If one cell should span 3 rows:

<td rowspan="3">Science</td>

**Using Tables for Layout (Email Context)**

In email design, modern layout techniques like CSS Grid and Flexbox are not well supported across email clients. Therefore, developers use <table> to design fixed layouts with multiple sections (e.g., header, content, footer).

Each row and cell acts as a container for images, text, and buttons. Inline CSS is also used heavily in this case for styling.

This approach ensures that the design renders **consistently across all major email platforms** like Gmail, Yahoo, and Outlook.

**CSS – Cascading Style Sheets**

**What is CSS?**  
CSS stands for *Cascading Style Sheets*. It is used to control the **appearance and layout** of HTML elements. HTML handles the **structure**, while CSS handles the **style** (color, size, spacing, fonts, layout, etc.).

**Ways to Include CSS in an HTML Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Description | Where It’s Written | Use Case |
| Inline | CSS is written directly in the HTML tag using the style attribute. | Inside an HTML element (e.g., <p>) | Quick styling, small changes |
| Internal | CSS is written inside a <style> tag in the <head> section. | <head> of the HTML document | Styling a single HTML page |
| External | CSS is written in a separate .css file and linked to the HTML file. | In an external file, linked using <link> | Best practice for large websites |

**Example of Link Tag for External CSS**

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

**Note:**

* **External CSS** is preferred for real-world projects because it promotes **reusability** and **clean separation** of content and style.
* **Inline CSS** is discouraged for large-scale use as it makes code harder to maintain.

# Lab 2

**HTML Lists**

Lists are used in HTML to display grouped items. HTML supports three types of lists:

1. **Unordered List** (<ul>)
2. **Ordered List** (<ol>)
3. **Description List** (<dl>)

Each list type is used for a different purpose, and each supports various attributes and formatting.

### ****1. Unordered List (****<ul>****)****

* Displays list items with **bullet points**.
* Used when **order does not matter**.
* Each item is written inside a <li> (list item) tag.

**Default Bullet Types (depending on browser and list level):**

* Disc (●)
* Circle (○)
* Square (■)

**Change Bullet Style with CSS:**

ul {

list-style-type: circle; /\* or square, disc, none \*/

}

**Common list-style-type values:**

|  |  |
| --- | --- |
| Value | Resulting Bullet |
| disc | ● (default) |
| circle | ○ |
| square | ■ |
| none | No bullet |

### ****2. Ordered List (****<ol>****)****

* Displays list items in a **specific order** (numbers or letters).
* Used when **sequence is important** (e.g., steps in a process).
* Items are written in <li> tags.

**Types of Numbering (using type attribute):**

|  |  |
| --- | --- |
| Type Value | Output Format |
| 1 | 1, 2, 3, 4 |
| A | A, B, C, D |
| a | a, b, c, d |
| I | I, II, III, IV |
| i | i, ii, iii, iv |

**Start Attribute:**

* Defines the starting number or letter.

<ol type="A" start="3">

### ****3. Description List (****<dl>****)****

* Used for **term-definition pairs** (like glossaries or FAQs).
* Contains:
  + <dt> – Definition Term
  + <dd> – Definition Description

**Example:**

<dl>

<dt>HTML</dt>

<dd>HyperText Markup Language</dd>

</dl>

### ****Nesting Lists****

You can place one list inside another to create sub lists.

<ul>

<li>Main Item

<ul>

<li>Sub Item</li>

</ul>

</li>

</ul>

**HTML Form – Working**

An HTML form is used to **collect user input** and **send it to a server** for processing. The process involves interaction between the **browser**, the **form**, and the **server**.

### ****Form Submission Workflow****

1. **User Fills the Form**
   * The user enters data into various fields (e.g., name, email, message).
2. **Form is Submitted**
   * When the user clicks the submit button, the form sends the data to the server.
3. **Action Attribute**
   * The action attribute in the <form> tag specifies the **server endpoint (URL)** where the form data will be sent.

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

1. **Method Attribute**
   * The method attribute defines **how** the data is sent:
     + GET: Appends form data in the URL (visible, less secure).
     + POST: Sends data in the body (not visible, more secure).
2. **Form Data is Packaged**
   * Each field has a name attribute. Data is sent in **key-value pairs**, like:

name=Ali&email=ali@example.com

1. **Data is Sent to Server**
   * The browser sends the packaged data using the specified method (GET or POST) to the server URL defined in action.
2. **Server Receives and Processes**
   * The server reads the data, processes it (e.g., saves to database, sends email), and returns a response.
3. **User Sees Response**
   * The server may return a new page, a success message, or redirect the user to another URL.

### ****Client-Side vs. Server-Side****

|  |  |  |
| --- | --- | --- |
| Process | Happens Where | Purpose |
| Form Display | Browser | Show form to user |
| Data Collection | Browser | Collect user input |
| Data Submission | Browser | Send data to server |
| Data Processing | Server | Validate, store, respond |

**Note:**

* Without a backend or proper action, the form won’t actually store or process data.
* method="post" is preferred for sensitive data (e.g., passwords, emails).

## ****HTML Forms – Complete Overview****

HTML forms are used to **collect user input** and send it to a server for processing. Forms support various **input types**, **attributes**, and **basic validation mechanisms** to ensure user-friendly and secure data collection.

### ****1. Key**** <form> ****Tag Attributes****

|  |  |
| --- | --- |
| Attribute | Description |
| action | URL or script where form data will be sent |
| method | HTTP method for data submission (get or post) |
| autocomplete | Enables/disables browser autofill (on / off) |
| target | Specifies where to display the response (\_blank, \_self, etc.) |
| novalidate | Disables default HTML5 validation if present |

### ****2. GET vs POST****

|  |  |  |  |
| --- | --- | --- | --- |
| Method | Description | Visible in URL? | Use Case |
| GET | Sends data in the URL | Yes | Search queries, filters |
| POST | Sends data in the request body | No | Login forms, registration, secure data |

### ****3. Basic Form Controls****

* <input> – For text fields, buttons, checkboxes, etc.
* <textarea> – For multi-line text.
* <select> – For dropdown lists.
* <option> – Inside <select>, defines each option.
* <label> – Provides a clickable caption for form elements.
* <button> – Triggers actions like submission.

### ****Input Types (With Attributes & Description)****

|  |  |  |  |
| --- | --- | --- | --- |
| Input Type | Purpose | Common Attributes | Validations/Notes |
| text | Single-line text input | name, value, placeholder, maxlength, required | Used for names, cities |
| password | Hides characters (●●●) | name, maxlength, required | Used for passwords |
| email | Validates email format | name, required, placeholder | type="email" does format check |
| number | Numeric input | min, max, step, required | Only numbers allowed |
| tel | Telephone number input | pattern, placeholder | Allows pattern-based format |
| url | URL input | required | Checks for valid URL format |
| checkbox | Multiple options (on/off) | checked, name, value | Can select multiple |
| radio | One option from a group | name, checked, value | One per group (same name) |
| file | File upload | accept, multiple | Used in contact/job forms |
| date | Date picker | min, max, value, required | Native browser support |
| range | Slider | min, max, step | Used for volume, price range |
| color | Color picker | value | Returns hex code |
| submit | Submit the form | value | Triggers form submission |
| reset | Clears all inputs | - | Resets form fields |
| button | General button | onclick | Requires JS or type setting |
| hidden | Hidden from user | name, value | Used to send hidden data |

### ****5. Select Dropdown (Single and Multiple Options)****

* <select> contains multiple <option> tags.
* Use multiple attribute to allow multi-selection.

<select name="country">

<option value="pk">Pakistan</option>

<option value="us">USA</option>

</select>

### ****6. Radio Buttons vs Checkboxes****

|  |  |  |
| --- | --- | --- |
| Feature | Radio (<input type="radio">) | Checkbox (<input type="checkbox">) |
| Selection | Only one in a group | Multiple allowed |
| Grouping | Must have the same name | Can have different or same name |
| Use Case | Gender, Payment method | Hobbies, Newsletter opt-in |

### ****7. Basic Inline Validation (No JavaScript)****

HTML5 provides built-in form validation:

|  |  |
| --- | --- |
| Attribute | Purpose |
| required | Field must be filled before submission |
| maxlength | Limits maximum characters |
| min/max | Defines range for numbers or dates |
| pattern | Regex pattern validation |
| type | Some input types (e.g., email) have built-in validation |

**Example:**

<input type="email" required placeholder="Enter your email">

## ****CSS Selectors****

CSS selectors are used to **target specific HTML elements** for styling. They define **which elements** the CSS rules apply to.

### ****1. Tag (Element) Selector****

* Selects all elements of a specific HTML tag.

p {

color: black;

}

**Applies to:** Every <p> element on the page.

### ****2. Class Selector (****.class****)****

* Selects all elements with a specific class attribute.
* Use . followed by the class name.

.highlight {

background-color: yellow;

}

**Usage in HTML:**

<p class="highlight">This is highlighted text. </p>

**Multiple elements can share the same class.**

### ****3. ID Selector (****#id****)****

* Selects a **single unique element** with a specific ID.
* Use # followed by the ID name.

#header {

font-size: 20px;

}

**Usage in HTML:**

<h1 id="header">Welcome</h1>

**ID must be unique per page** (Important for SEO and accessibility).

### ****4. Group Selector****

* Apply the same styles to multiple elements separated by commas.

h1, h2, p {

font-family: Times New Roman;

}

### ****5. Descendant Selector****

* Targets elements **inside** another element.

div p {

color: gray;

}

**Applies to:** All <p> inside any <div>.

### ****6. Child Selector (****>****)****

* Selects **direct children** only.

ul > li {

list-style-type: square;

}

### ****7. Universal Selector (****\*****)****

* Selects **all elements** on the page.

\* {

margin: 0;

padding: 0;

}

### ****8. Attribute Selector****

* Targets elements based on the **presence or value** of an attribute.

input[type="text"] {

border: 1px solid black;

}

### ****9. Pseudo-class Selector (Intro Only)****

* Targets elements based on their **state**.

a:hover {

color: red;

}

### ****Selector Summary Table****

|  |  |  |  |
| --- | --- | --- | --- |
| Selector Type | Syntax | Targets | Use Case Example |
| Tag Selector | p | All <p> tags | Basic element styling |
| Class Selector | .className | Elements with that class | Styling reusable blocks |
| ID Selector | #idName | One unique element | Styling a single header or block |
| Group Selector | h1, p, div | Multiple elements | Common styling |
| Descendant | div p | <p> inside <div> | Nested styling |
| Child | ul > li | Direct child <li> only | Precision list styling |
| Universal | \* | All elements | Reset CSS |
| Attribute | input[type="email"] | Inputs of specific type | Form field customization |

## ****CSS Float****

The float property in CSS is used to **position elements side-by-side** (typically left or right), allowing text and other elements to wrap around them. It was widely used for layout creation before the advent of Flexbox and Grid.

### ****1. Syntax of**** float

selector {

float: left | right | none | inherit;

}

|  |  |
| --- | --- |
| Value | Description |
| left | Floats the element to the **left** |
| right | Floats the element to the **right** |
| none | Default. Element does not float |
| inherit | Inherits float value from parent |

### ****2. Behavior of Floated Elements****

* A floated element is **taken out of the normal document flow**.
* Block elements like <div> shrink to fit the floated content.
* Following content may **wrap around** the floated element.

### ****3.**** Clear ****Property****

The clear property prevents elements from **appearing beside floated elements**.

clear: left | right | both | none;

|  |  |
| --- | --- |
| Value | Prevents Floating Next To |
| left | Left-floated elements |
| right | Right-floated elements |
| both | Both left and right floats |
| none | No clearing (default) |

**Common Use Case:**  
To stop text or other elements from wrapping below floated columns.

### ****Layout Creation with Float****

Floats are commonly used for **2-column** or **3-column layouts**.

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

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

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

<div class="clear"></div>

.header {

background: #ccc;

padding: 10px;

}

.sidebar {

float: left;

width: 30%;

background: #f2f2f2;

}

.content {

float: right;

width: 68%;

background: #e0e0e0;

}

.clear {

clear: both;

}

* float: left is used for the sidebar.
* float: right is used for the main content.
* A clear: both is applied after floating elements to maintain layout integrity.

### ****6. Limitations of Float Layouts****

* Floats can collapse parent containers if not cleared properly.
* Not flexible for complex responsive designs.
* Replaced in modern layouts by **Flexbox** and **CSS Grid**, but still valuable for foundational understanding.

# ****Lab 3****

## ****CSS Flexbox****

**Flexbox (Flexible Box Layout)** is a CSS layout model that allows **responsive alignment and distribution** of elements in a container — even when their size is unknown or dynamic.

It is more efficient and powerful than traditional float or inline-block methods for designing modern web layouts.

## ****2. Flexbox Structure****

* **Main Axis**: The primary direction (row or column)
* **Cross Axis**: Perpendicular to main axis

## ****3. Flex Container Properties (Parent Only)****

### ****A.**** display: flex

Turns an element into a flex container. All direct children become flex items.

### ****B.**** flex-direction

Defines the **direction** of the main axis.

|  |  |
| --- | --- |
| Value | Description |
| row (default) | Left to right (main axis horizontal) |
| row-reverse | Right to left |
| column | Top to bottom (main axis vertical) |
| column-reverse | Bottom to top |

### ****C.**** flex-wrap

Allows items to wrap to the next line if there's not enough space.

|  |  |
| --- | --- |
| Value | Description |
| nowrap (default) | All items on one line |
| wrap | Items wrap onto multiple lines |
| wrap-reverse | Wraps in reverse order |

### ****D.**** justify-content

Aligns **items along the main axis** (left–right in row, top–bottom in column).

|  |  |
| --- | --- |
| Value | Description |
| flex-start | Items align to start of main axis |
| flex-end | Items align to end of main axis |
| center | Items centered on main axis |
| space-between | First item at start, last at end, rest distributed **with space in between** |
| space-around | Equal space **around** each item |
| space-evenly | Equal space **between all items including edges** |
| left (deprecated) | Align items to the left (non-standard) |
| right (deprecated) | Align items to the right (non-standard) |

### ****E.**** align-items

Aligns **items along the cross axis** (vertical in row, horizontal in column).

|  |  |
| --- | --- |
| Value | Description |
| stretch (default) | Items stretch to fill cross-axis |
| flex-start | Items align to **start of cross axis** |
| flex-end | Items align to **end of cross axis** |
| center | Items **vertically centered** (if row) |
| baseline | Aligns items along their **text baseline** |
| top / bottom | Not valid for align-items — handled using align-self or margins |

### ****F.**** align-content

Aligns **multiple rows** (when flex-wrap is enabled) **along the cross axis**.

Only applies when **more than one row or column** exists.

|  |  |
| --- | --- |
| Value | Description |
| flex-start | Rows aligned to start of cross axis |
| flex-end | Rows aligned to end of cross axis |
| center | Rows centered |
| space-between | Equal space **between rows** |
| space-around | Equal space **around each row** |
| space-evenly | Equal space **between and around rows** |
| stretch (default) | Rows stretch to fill container height/width |

### ****G.**** gap

Sets spacing between items **without using margins**.

|  |  |
| --- | --- |
| Property | Description |
| gap: 10px | Applies both row and column gap |
| row-gap | Sets vertical spacing |
| column-gap | Sets horizontal spacing |

## ****4. Flex Item Properties (Children Only)****

### ****A.**** flex-grow

Defines how much a flex item **can grow** relative to others.

|  |  |
| --- | --- |
| Value | Description |
| 0 (default) | Item won’t grow |
| 1 | Item can grow (relative) |

Example:

* flex-grow: 2 grows **twice** as much space as an item with flex-grow: 1.

### ****B.**** flex-shrink

Defines how much a flex item **can shrink** if there's not enough space.

|  |  |
| --- | --- |
| Value | Description |
| 1 (default) | Item can shrink if needed |
| 0 | Item will not shrink |

### ****C.**** flex-basis

Defines the **initial size** of an item **before growing or shrinking**.

|  |  |
| --- | --- |
| Value | Description |
| auto (default) | Uses width/height or content size |
| e.g. 200px | Starts with 200px width (before flexing) |

### ****D.**** flex ****(Shorthand)****

Shorthand for:

flex: <flex-grow> <flex-shrink> <flex-basis>;

Examples:

* flex: 1 → 1 1 0% (grow, shrink, no base size)
* flex: 0 0 100px → fixed size (won’t grow or shrink)

### ****E.**** align-self

Overrides align-items for **one specific item**.

|  |  |
| --- | --- |
| Value | Description |
| auto (default) | Inherits from align-items |
| flex-start | Align item to start of cross axis |
| flex-end | Align item to end of cross axis |
| center | Center the item on cross axis |
| baseline | Align item to baseline |
| stretch | Stretch to fill (if no set height or width) |

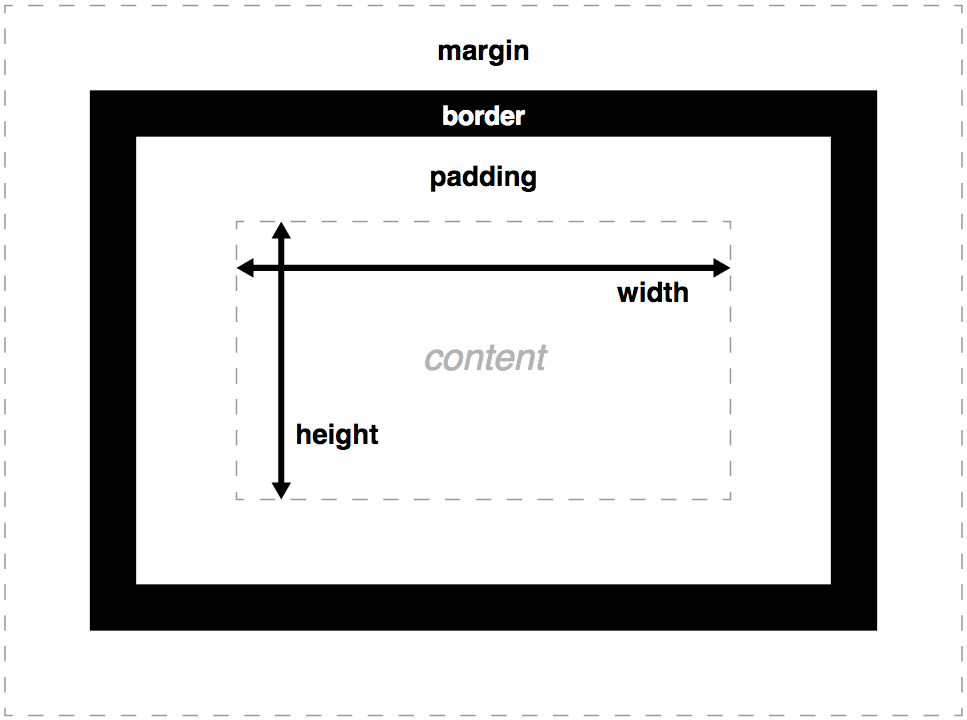
## ****CSS Box Model****

### ****Definition****

The **CSS Box Model** describes how every HTML element is treated as a **rectangular box** consisting of the following parts:

|  |  |
| --- | --- |
| Part | Description |
| **Content** | Actual text or image inside the element |
| **Padding** | Space between the content and the border |
| **Border** | The line surrounding the padding (optional; can be styled, sized, or removed) |
| **Margin** | Space between the element’s border and surrounding elements (external gap) |

### ****Structure (Inside to Outside)****



### ****Width and Height Calculation****

By default, width and height in CSS only apply to the **content box**.

**Total Width** =  
margin-left + border-left + padding-left + content-width + padding-right + border-right + margin-right

**Total Height** =  
margin-top + border-top + padding-top + content-height + padding-bottom + border-bottom + margin-bottom

### ****Box-Sizing Property****

|  |  |
| --- | --- |
| Property Value | Effect |
| content-box (default) | Width/height includes **only content**; padding and border are added outside |
| border-box | Width/height includes **content + padding + border** (easier layout control) |

Use box-sizing: border-box; in modern layouts to prevent unexpected overflows.

### ****Example Comparison****

|  |  |  |
| --- | --- | --- |
| Property | content-box Result | border-box Result |
| width: 100px; padding: 10px; border: 5px; | Final box = 130px | Final box = 100px |

## ****HTML Size Units****

HTML and CSS use different **units to define size**, such as width, height, margins, font-size, etc.

### ****1. Absolute Units****

Fixed size regardless of screen or resolution.

|  |  |  |
| --- | --- | --- |
| Unit | Description | Example |
| px | Pixels | font-size: 16px; |
| pt | Points (1pt = 1/72 inch) | Rarely used in web |
| in | Inches | width: 1in; |
| cm | Centimeters | height: 2cm; |
| mm | Millimeters | margin: 5mm; |

### ****2. Relative Units****

Adjust based on other elements or screen.

|  |  |  |
| --- | --- | --- |
| Unit | Description | Example |
| % | Percentage of parent element | width: 50%; |
| em | Relative to parent’s font-size | font-size: 1.5em; |
| rem | Relative to root (html) font-size | font-size: 2rem; |
| vw | % of viewport width | width: 50vw; |
| vh | % of viewport height | height: 100vh; |
| vmin | Smaller of vw or vh |  |
| vmax | Larger of vw or vh |  |

For responsive layouts, use relative units like %, rem, vw, vh.

## ****Microsoft Clarity****

**Microsoft Clarity** is a **free analytics tool** that shows how users interact with your site using:

* **Heatmaps** (where users click)
* **Session recordings**
* **User behavior tracking**

### ****Why it's useful****

* Detects UX issues like rage clicks or dead zones
* No performance impact
* GDPR compliant

### ****How to Use****

1. Sign up at [clarity.microsoft.com](https://clarity.microsoft.com)
2. Add the tracking code to your HTML <head> section
3. Data appears in your dashboard within minutes

## ****Google Analytics****

Google Analytics tracks **website traffic and user behavior**.

### ****What it Does****

* Tracks number of users, sessions, bounce rate
* Shows traffic sources (search engine, direct, social)
* Offers reports on pages viewed, session duration, devices

### ****How to Set Up****

1. Sign up at [analytics.google.com](https://analytics.google.com)
2. Create a property for your website
3. Copy tracking script
4. Paste in <head> of your HTML

<! -- Google Analytics Code (example) -->

<script async src="https://www.googletagmanager.com/gtag/js?id=UA-XXXXXXX-X"></script>

<script>

window.dataLayer = window.dataLayer || [];

function gtag(){dataLayer.push(arguments);}

gtag('js', new Date());

gtag('config', 'UA-XXXXXXX-X');

</script>

## ****Overlay Gradient****

An **overlay gradient** is used to place a semi-transparent color **on top of a background image**, often for better text readability.

### ****Usage Example (CSS)****

css

CopyEdit

background:

linear-gradient(rgba(0,0,0,0.5), rgba(0,0,0,0.5)),

url('image.jpg');

background-size: cover;

### ****Explanation****

* linear-gradient(...) adds a semi-transparent dark layer
* rgba(0,0,0,0.5) is a black color at 50% transparency
* The image remains visible but is darkened for clarity

# Lab 4

## ****What is CSS Grid?****

CSS Grid is a **2-dimensional layout system** for arranging items in rows and columns. Unlike Flexbox (which is one-directional), Grid allows for full control over **both horizontal and vertical placement**.

It is ideal for creating **complex web layouts**, **dashboard UIs**, **magazine-style pages**, and **asymmetric designs**.

## ****Basic Terminology****

|  |  |
| --- | --- |
| Term | Description |
| Grid Container | The parent element with display: grid |
| Grid Items | Direct children of the container |
| Grid Lines | The dividing lines between rows and columns |
| Grid Tracks | The rows and columns themselves |
| Grid Cell | The space between two adjacent row and column lines |
| Grid Area | A rectangular space covering one or more grid cells |

## ****Creating a Grid Container****

.container {

display: grid;

}

## ****1. Grid Template: Rows and Columns****

Used to define how many columns or rows the grid should have and their size.

.container {

display: grid;

grid-template-columns: 200px 1fr 2fr;

grid-template-rows: auto 100px;

}

|  |  |
| --- | --- |
| Unit | Description |
| px, em | Fixed size |
| % | Relative to container |
| fr | Fraction of available space (most used) |
| auto | Size based on content |

fr is **unique to grid** and allows flexible layout.

## ****2. Grid Gaps****

Use gap, row-gap, and column-gap to create space between grid items.

grid-gap: 20px; /\* shorthand \*/

row-gap: 10px;

column-gap: 30px;

## ****3. Naming Grid Lines****

You can assign **names to grid lines** for easier reference.

grid-template-columns: [start] 1fr [middle] 2fr [end];

## ****4. Placing Items with Line Numbers****

You can manually place items using line numbers.

.item {

grid-column: 1 / 3; /\* starts at line 1, ends before line 3 \*/

grid-row: 2 / 4;

}

## ****5. Placing Items with**** grid-column-start****,**** grid-column-end****, etc.****

You can also write them separately:

.item {

grid-column-start: 1;

grid-column-end: 3;

grid-row-start: 2;

grid-row-end: 4;

}

## ****6. Spanning Multiple Rows or Columns****

|  |  |
| --- | --- |
| Syntax | Meaning |
| grid-column: 1 / span 2 | Start at line 1, span 2 columns |
| grid-row: 2 / span 3 | Start at row line 2, span 3 rows |

The span keyword is useful for adaptive layouts.

## ****7. Grid Auto Flow****

Controls how auto-placed items fill the grid.

grid-auto-flow: row | column | row dense | column dense;

|  |  |
| --- | --- |
| Value | Description |
| row | Default: fills rows first |
| column | Fills columns first |
| row dense | Tries to backfill empty gaps |
| column dense | Same for columns |

dense is useful for compact layouts, like image galleries.

## ****8. Auto Rows and Auto Columns****

Used for defining size of implicit rows/columns (those not explicitly defined).

grid-auto-rows: 100px;

grid-auto-columns: 1fr;

## ****10.**** justify-items ****and**** align-items

Control alignment of items **within their cells**.

|  |  |  |
| --- | --- | --- |
| Property | Applies To | Values |
| justify-items | horizontal axis | start, end, center, stretch |
| align-items | vertical axis | start, end, center, stretch |

## ****11.**** justify-content ****and**** align-content

Aligns **entire grid** inside container.

|  |  |  |
| --- | --- | --- |
| Property | Axis | Values |
| justify-content | Main (row) | start, end, center, space-between, space-around, space-evenly |
| align-content | Cross (column) | Same |

## ****12.**** justify-self ****and**** align-self

Align a **single item** inside its cell.

.item {

justify-self: center;

align-self: end;

}

## ****Comparison of Table, Float, Flexbox, and Grid Layouts****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | **Table** | **Float** | **Flexbox** | **Grid** |
| **Introduced In** | HTML 3.2 (1997) | CSS1 (1996) | CSS3 | CSS3 |
| **Layout Type** | Fixed tabular layout (2D by default) | Originally for inline content wrap, later used for layout hacks | One-dimensional (row **or** column) | Two-dimensional (rows **and** columns) |
| **Direction Control** | Row & Column inherently | Left or right (float only) | Row or column using flex-direction | Both directions using grid-template-\* |
| **Responsiveness** | Poor | Poor to moderate (requires clearfix, media queries) | Good — designed for responsive layouts | Excellent — best for complex responsive designs |
| **Alignment Support** | Limited (via cell attributes) | Requires manual margin management | Rich support with justify-\*, align-\* | Richest support — per-item and container-wide |
| **Content Reordering** | Not supported | Not supported | Supported with order | Fully supported with order, line placements, areas |
| **Overlapping Elements** | Not possible | Not intended | Not ideal | Possible with grid-area, z-index, etc. |
| **Explicit Placement** | Yes (row/col numbers) | No | Partially (via order) | Full — via line numbers, names, or area mapping |
| **Main Use Case** | Tabular data (e.g., financial reports, schedules) | Wrapping text around images, old layout tricks | Toolbars, navigation bars, cards in a row/column | Full-page layouts, dashboards, asymmetric designs |
| **Important Attribute** | <table>, <tr>, <td> structure | float: left/right | display: flex and flex-direction | grid-template-rows, grid-template-columns, areas |
| **Drawbacks** | Not semantic for layout; poor accessibility | Needs clearfix, breaks flow | Harder to manage complex 2D layouts | Slightly more complex syntax; newer browser support |
| **Semantic Use Today** | Yes, but **only for tabular data** | No (outdated for layout purposes) | Yes | Yes |

## ****1. CSS Specificity & Precedence****

When multiple CSS rules apply to the same element, **specificity** and **order of appearance** determine which style is applied.

### ****Precedence Order (Highest to Lowest)****

| **Source Type** | **Example** | **Priority** |
| --- | --- | --- |
| **Inline CSS** | <p style="color: red;"> | Highest |
| **Internal CSS** | <style>p { color: blue; }</style> | Medium |
| **External CSS** | Linked CSS file | Lowest |

If all styles have the same specificity, the **last defined one wins**.

### ****Specificity Rules (for selectors)****

Specificity is calculated based on the **type of selector**:

|  |  |  |
| --- | --- | --- |
| Selector Type | Specificity Weight | Example |
| **Inline Style** | 1000 | style="color: red" |
| **ID Selector** | 100 | #header {} |
| **Class / Attribute / Pseudo-class** | 10 | .menu, [type="text"], :hover |
| **Tag (Element) Selector** | 1 | div, h1, p |
| **Universal Selector** | 0 | \* {} |

Higher total specificity wins.  
!important can override specificity but should be avoided for maintainability.

## ****Image Attributes (Corrected and Complete)****

Including: alt, width, height, file path, object-fit, and object-position

### ****1.**** src ****(File Path)****

Tells the browser where to find the image.

|  |  |  |
| --- | --- | --- |
| Path Type | Example | Use Case |
| Relative Path | src="images/logo.png" | Image in local folder |
| Absolute Path | src="/assets/banner.jpg" | From root directory |
| Full URL | src="https://site.com/a.jpg" | External site image |

Best practice: use **relative path** within your project folder structure.

### ****2.**** alt ****(Alternative Text)****

Text shown when image doesn’t load or for screen readers.

* Required for **SEO** and **accessibility**
* Keep it meaningful and descriptive

**Always include** alt, even if it’s alt="" for decorative images.

### ****3.**** width ****and**** height

These control the display size.

|  |  |  |
| --- | --- | --- |
| Method | Example | Notes |
| HTML attribute | <img width="200" height="100"> | Sets static pixel size |
| CSS property | img { width: 100%; height: auto; } | Flexible, scalable sizing |

Always try to **preserve aspect ratio** (use auto or define only one).

### ****4.**** object-fit

Controls how image fits inside its container (used in CSS with img inside a div or layout box).

|  |  |
| --- | --- |
| Value | Description |
| fill | Stretches image to fit (may distort) |
| contain | Fits whole image inside (might leave blank space) |
| cover | Fills box completely (crops if needed to maintain ratio) |
| none | Image not resized |
| scale-down | Chooses none or contain — whichever results in smaller |

object-fit only works if both width and height of the container are defined.

### ****5.**** object-position

Used **with object-fit** to control **which part of the image is visible** when cropping happens (mainly with cover or contain).

|  |  |
| --- | --- |
| Value | Description |
| center center | Default: center horizontally and vertically |
| top left | Shows top-left part of the image |
| top right | Shows top-right part |
| bottom left | Shows bottom-left part |
| bottom right | Shows bottom-right part |
| 50% 75% | Custom positioning: horizontal %, vertical % |

img {

width: 100%;

height: 200px;

object-fit: cover;

object-position: top left;

}

**object-position only works with object-fit** — both must be used together.

# ****CSS Positioning****

CSS Positioning allows you to control **exactly where** elements appear on a web page. You can remove elements from the normal document flow and place them at specific locations using different position values.

## ****Types of Positioning in CSS****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Position Type** | **Definition** | **Offset Properties Used** | **Stacking Context** | **Removed from Normal Flow?** |
| static | Default position. Element stays in the normal flow of the document. | None (top, left, etc. don’t apply) | No | No |
| relative | Positions the element **relative to its original position** in normal flow. | top, right, bottom, left | No | No |
| absolute | Positions the element **relative to the nearest positioned ancestor** (non-static). If none, relative to <html>. | top, right, bottom, left | Yes | Yes |
| fixed | Positions the element **relative to the browser window**. Doesn’t move on scroll. | top, right, bottom, left | Yes | Yes |
| sticky | Element scrolls with the page until it reaches a defined offset, then “sticks”. | top, right, bottom, left | Partial | No |

## ****Offset Properties****

When using relative, absolute, fixed, or sticky positioning, you can use the following properties to place the element:

* **top** – Distance from the top edge of the containing block
* **right** – Distance from the right edge
* **bottom** – Distance from the bottom edge
* **left** – Distance from the left edge

## ****Stacking Elements:**** z-index

* z-index defines the stack order of overlapping positioned elements.
* Higher z-index means the element appears **on top**.

Only elements with a position value other than static can use z-index.