Mohsin Iban Hossain
AIUB, Web Tech Notes

WEB TECHNOLOGIES

Table of Contents

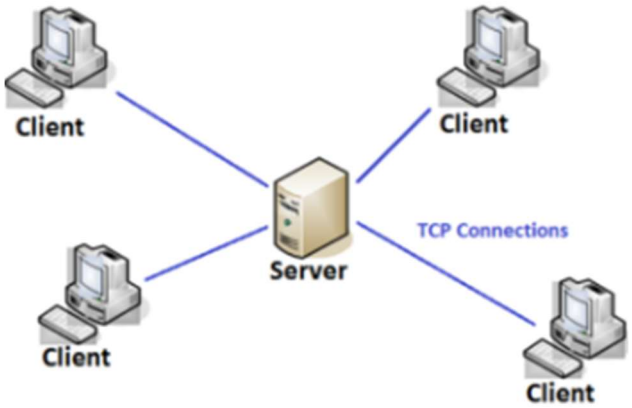
INTRODUCTION TO WEB TECHNOLOGIES

Client/Server model

⇨ A model where tasks are divided between clients and servers.

- **Client:**
  - Requests services or resources.
  - Examples: Web browsers, mobile apps, email clients.

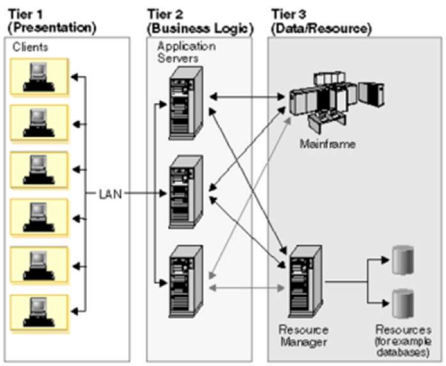
- **Server:**
  - Provides services or resources.
  - Examples: Web server, database server, file server.



Three-tier Client Server Architecture

- **Two-Tier Architecture:**
  - **Client Level:** User interface and request handling
  - **Server Level:** Processes requests and manages data

- **Three-Tier Architecture:**
  - **Client:** Interacts with the user
  - **Application Server:** Contains business logic
  - **Resource Manager:** Stores and manages data

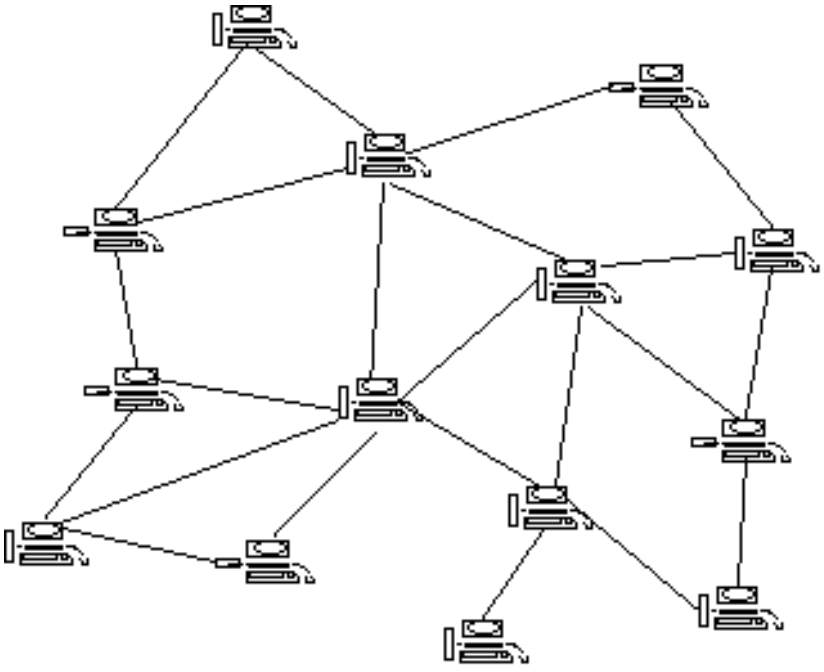


Types of Servers

Peer to Peer (P2P)

⇨ Distributed architecture that divides tasks among peers.

- **Peers:** Equally privileged and capable participants.
- **Network:** Forms a peer-to-peer (P2P) network of nodes.



Internet

- Global network of interconnected devices
- Uses **TCP/IP** protocols for communication

Types of Networks (by Size & Coverage):

- **PAN (Personal Area Network):**
  - Short-range (≤10 meters)
  - Example: Bluetooth between phone and headset
- **LAN (Local Area Network):**
  - Covers small areas (home, office, building)
- **MAN (Metropolitan Area Network):**
  - Covers a city or large campus
- **WAN (Wide Area Network):**
  - Covers large areas (country/world)

Network Address Translation (NAT)

⇨ Modifies IP address info in packet headers via a router or firewall.

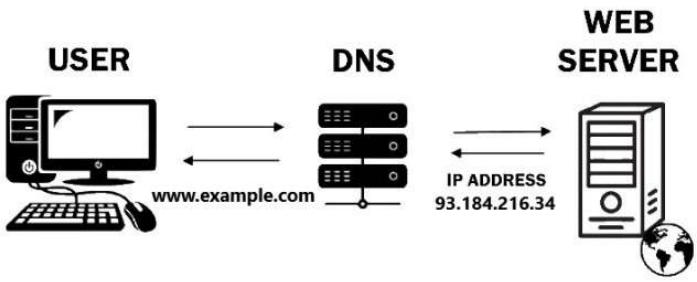
✓ **Purpose:**
- Share a **single public IP** among multiple devices
- **Conserves IP addresses**
- Enhances **network security**

✓ **IP Address Types:**

- **Private IP:**
  - Used within local networks
  - Not routable on the internet
  - Examples: 192.168.x.x, 10.x.x.x, 172.16.x.x
- **Public IP:**
  - Assigned by ISPs
  - Used for internet communication

Domain Name System

- **Function:** Translates domain names into IP addresses
- **Purpose:** Allows access to websites using names (e.g., *www.google.com*) instead of IPs
- **Analogy:** Acts like the **phone book** of the internet, mapping names to addresses



World Wide Web (WWW)

⇨ System of interlinked hypertext documents and multimedia accessed via the Internet

- **Inventor:** Tim Berners-Lee (1989)
- **Importance:** Most widely used service on the Internet

Decade	Key Developments
1960s	Development of **ARPANET**
1980s	Proposal of the World Wide Web by Tim Berners-Lee
1990s	Launch of the first web browser and server
2000s	Dot-com boom, emergence of social media
2010s	Rise of mobile technology, advent of IoT
2020s	Continued innovation and evolution

WWW Standards

- **W3C (World Wide Web Consortium):**
  - Develops standards like **HTML**, **XHTML**, **DOM**
  - Defines structure & interpretation of web documents
- **IETF & Others:**
  - Contribute to internet standards
- **ECMAScript:**
  - Standard for **JavaScript**, by **Ecma International**
- **DOM (Document Object Model):**
  - From **W3C**, defines how web documents are structured and accessed
- **HTTP (Hyper Text Transfer Protocol):**
  - Protocol for communication & authentication between browser and server
- **URI (Uniform Resource Identifier):**
  - Identifies resources (e.g., documents, images) on the Internet

Uniform Resource Identifier (Example)

http://en.wikipedia.org:80/wiki/URI?page=2&frame=1#Examples_of_URI_references

- **Scheme:** `http` — protocol used
- **Authority:** `en.wikipedia.org` — domain name or server
- **Port:** `80` — communication port
- **Path:** `/wiki/URI` — specific resource location
- **Query:** `page=2&frame=1` — parameters for resource
- **Fragment:** `#Examples_of_URI_references` — section anchor within the page

URI, URL, & URN

- **URI (Uniform Resource Identifier):**
  - Identifies a resource on the Internet
  - Broad term covering URLs and URNs
  - Example:`https://www.example.com/path/to/resource?query=value#section`
- **URL (Uniform Resource Locator):**
  - Specifies resource location and access protocol
  - Example: `https://www.example.com/index.html`
- **URN (Uniform Resource Name):**
  - Provides a unique, persistent name without location or access info
  - Example: `urn:isbn:0451450523`

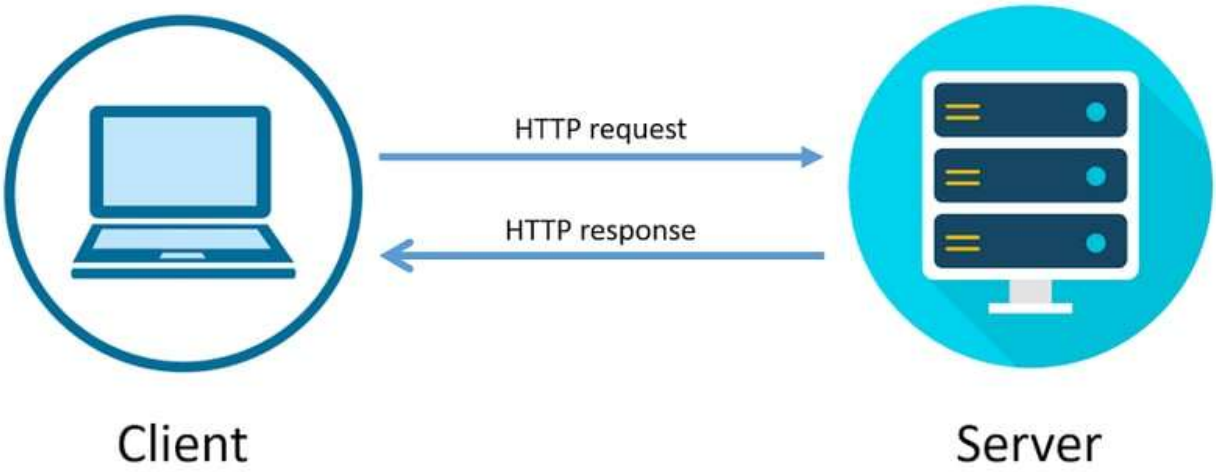
Browsers

⇨ **Definition:** Software application to retrieve, display, and navigate web content

- **Content:** Accesses resources identified by a URI (web pages, images, videos, etc.)
- **Examples:** Internet Explorer, Netscape, Firefox, Opera, Chrome, Safari

HTTP protocol

- **Client:** Sends an **HTTP request** to the server
- **Server:** Returns an **HTTP response** to the client



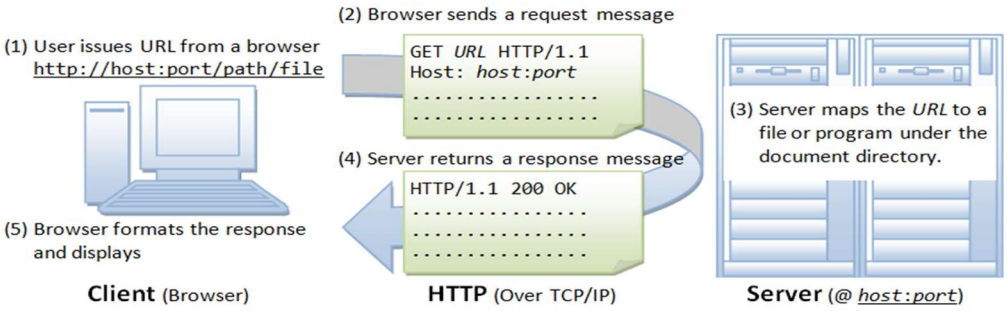
HTTP (HyperText Transfer Protocol:

- **Default Port:** 80
- **Communication:** Via HTTP **Requests** and **Responses**
- **Connectionless:** No memory of previous connections after closing
- **Stateless:** Client and server do not retain session information

HTTP Steps

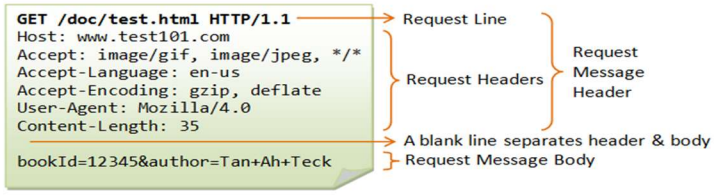
URL to HTTP Request:

- When a **URL** (e.g., `http://www.abc.com/index.html`) is entered in a browser:
  - The browser **converts it into an HTTP request**
  - Sends the request to the **HTTP server** to fetch the resource



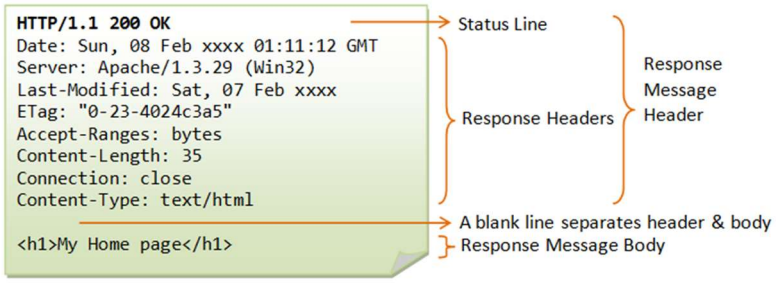
HTTP Server & URL:

- **HTTP Server:**
  - Interprets the request
  - Sends a **response** (requested resource or error)
- **URL (Uniform Resource Locator):**
  - Uniquely identifies a web resource
  - Syntax: `protocol://hostname:port/path-and-file-name`
- **Browser Action:**
  - Converts URL into a **request message**
  - Sends it to the server using the specified **protocol**



How Server Handles HTTP Requests:

- **Request Mapping:**
  - Maps the request to a **file** in the server's document directory
  - Or maps to a **program**, executes it, and returns the output
- **Response:**
  - Returns the requested **file** or **program output**
  - If not successful, returns an **error message**



HTTP Response Message

Response Types:

- **1xx:** Informational
- **2xx:** Success
- **3xx:** Redirection
- **4xx:** Client Error
- **5xx:** Server Error

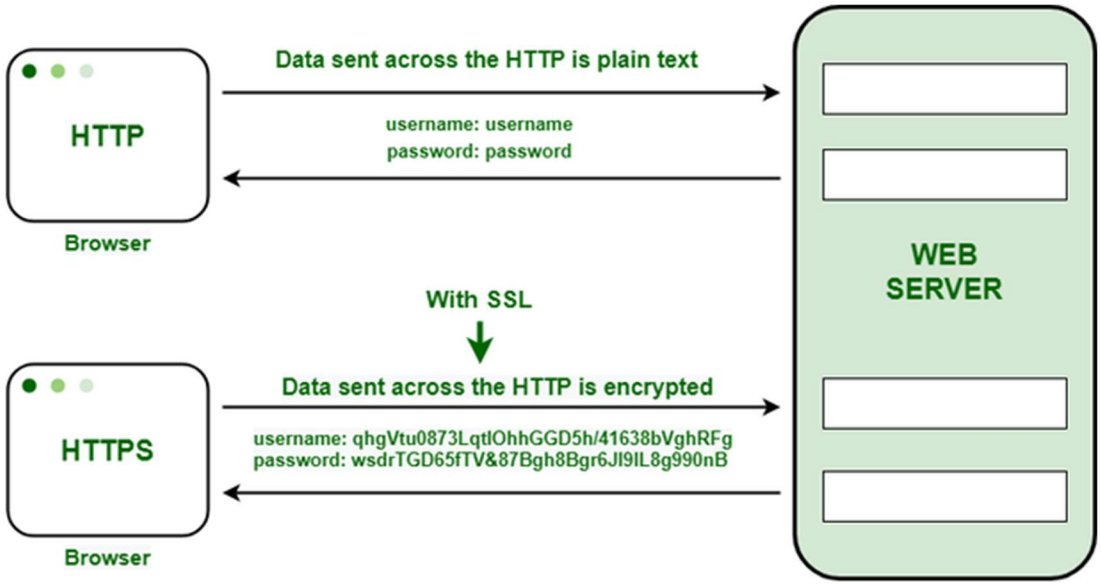
Common Status Codes:

Status Code	Status Text	Meaning
1xx	Informational	Request received, continuing process
200	OK	Request successful; data returned
201	Created	Request successful; new resource created
400	Bad Request	Invalid syntax in request
401	Unauthorized	Authentication required
403	Forbidden	Server understood request but refuses to authorize
404	Not Found	Requested resource not found
500	Internal Server Error	Server encountered an unexpected error

HTTP Request Methods

Method	Purpose
GET	Retrieves a resource from the server
POST	Sends data to the server (e.g., via HTML forms)
PUT	Asks the server to store or update data
DELETE	Requests the server to delete specified data

HTTP Vs HTTPs



HTML

What is HTML?

HTML (HyperText Markup Language):

- Describes the **structure** of a web page
- Made up of **elements** represented by **tags**
- Tags define content types like **heading**, **paragraph**, **table**, etc.
- **Browsers use tags** to render content but do not display the tags themselves
- **Invented by Tim Berners-Lee** in **1993**

A Simple HTML Document

Example:

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

Explanation:

HTML Elements Explanation:

- `<!DOCTYPE html>:` Declares the document as **HTML5**
- `<html>:` Root element of the HTML page
- `<head>:` Contains **meta information** about the page
- `<title>:` Sets the **page title** shown in browser tab or title bar
- `<body>:` Contains all **visible content** (headings, paragraphs, images, etc.)
- `<h1>:` Defines a **large heading**
- `<p>:` Defines a **paragraph**

What is an HTML Element?

- An HTML element is a building block of an HTML page, consisting of a **start tag**, **content**, and an **end tag**. It defines the structure and content of parts of a webpage, such as headings, paragraphs, links, images, and more.

Example:

```
<h1>My First Heading</h1>
<p>My first paragraph.</p>
```

Start Tag	Element Content	End Tag
<h1>	My First Heading	</h1>
	My first paragraph.	

HTML Page Structure

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

The <!DOCTYPE> Declaration

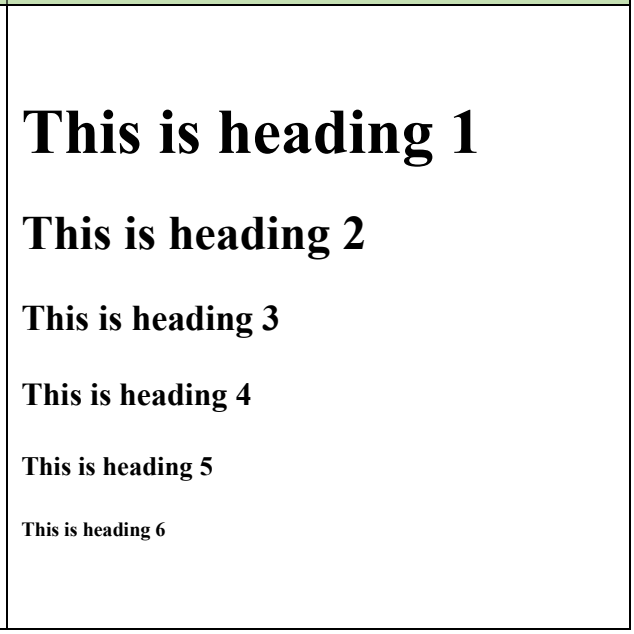
- Declares the **HTML version** (e.g., HTML5)
- Helps browsers to **render** the page correctly
- Must be the **first line** in the HTML file
- Example: `<!DOCTYPE html>`
- **Not** an HTML tag and has **no closing tag**

HTML Headings

⇨ HTML headings are used to define **titles or subtitles** on a webpage. They help organize content into sections, making it easier for users and search engines to understand the structure of the page.

- There are **6 levels** of headings: from `<h1>` (highest/most important) to `<h6>` (lowest).
- `<h1>` is typically used for the **main title**, while `<h2>` to `<h6>` are used for **subsections**.
- Headings are **bold** by default and decrease in size from `<h1>` to `<h6>.`
- Proper use of headings improves **readability** and **SEO**.
- Headings should follow a **hierarchical order** for better structure.

Example:	Output:

<pre><!DOCTYPE html>
<html>
  <head>
 	
  </head>
  <body>
   This is heading 1
   This is heading 2
   This is heading 3
   This is heading 4
   This is heading 5
   This is heading 6
  </body>
</html></pre>
	

HTML Paragraphs

⇨ In HTML, a **paragraph** is a block of text defined using the tag. It is used to group related sentences and improve the structure and readability of content on a webpage.

- Written using the ... tag.
- Automatically starts on a **new line**.
- Browsers add **space before and after** each paragraph by default.

Example:	Output:
`<p>This is a paragraph.</p>` `<p>This is another paragraph.</p>`	This is a paragraph. This is another paragraph.

HTML Text Formatting Tag

⇨ **Text formatting tags** in HTML are used to style or emphasize text in various ways, such as making it **bold, italic, underlined, or marked**. They help enhance readability and meaning.

Common Formatting Tags & Uses:

Tag	Purpose	Example
	Bold (non-semantic)	Bold Text
	Strong importance (semantic)	Important
	Italic (non-semantic)	Italic Text
	Emphasis (semantic italic)	Emphasized
<u>	Underlined text	<u>Underlined</u>
<mark>	Highlighted/marked text	<mark>Marked</mark>
<small>	Smaller-sized text	Small Text
<del>	Deleted/struck-through text	~~Old Price~~
<ins>	Inserted/underlined text	<ins>New Text</ins>
<sub>	Subscript (e.g., H_2O)	H₂O
<sup>	Superscript (e.g., x^2)	x²

Example:	Output:
This isboldandimportanttext. This is italic and emphasized text. This is <u>underlined</u> and <mark>highlighted</mark>. Water formula: H₂O Math: x² + y²	This is **bold** and **important** text. This is *italic* and *emphasized* text. This is underlined and highlighted. Water formula: H_2O Math: $x^2 + y^2$

HTML Links

⇨ **HTML links** allow users to **navigate** between different web pages or sections of a page. They are created using the `<a>` (anchor) tag and are one of the most essential elements of the web.

Basic Syntax:

```
<a href="URL">Link Text</a>
```

- **`href` attribute** specifies the destination URL.
- **Clickable text** is placed between the opening and closing `<a>` tags.

Types of Links:

Type	Example
External link	`<a href="https://www.google.com">Google</a>`
Internal link	`<a href="about.html">About Page</a>`
Email link	`<a href="mailto:someone@example.com">Send Email</a>`
Phone link	`<a href="tel:+880123456789">Call Now</a>`
Anchor link	`<a href="#section1">Go to Section 1</a>`

Open link in new tab:

Use `target="_blank"`

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

HTML Images

⇨ HTML images are used to embed pictures or graphics into a web page using the `<img>` tag. Images help make web content more engaging and informative.

- The `<img>` tag is **self-closing** (void element); it does **not** have an end tag.
- **Essential attributes:**
  - `src` — specifies the **path or URL** of the image.
  - `alt` — provides **alternative text** shown if the image can't be displayed; important for accessibility.
- Optional attributes:
  - `width` and `height` — control the image size (in pixels or percentage).
- Images can be stored locally (relative path) or hosted online (absolute URL).

Basic Syntax:

```
<img src="path/to/image.jpg" alt="Description of image">
```

Local Image Syntax:

```
<img src="images/photo.jpg" alt="Description of local image">
```

Online Image Syntax:

```
<img src="https://www.example.com/images/photo.jpg" alt="Image Description">
```

HTML Lists

⇨ HTML lists **organize content** into **ordered or unordered groups**, making information easier to read and understand.

Types of Lists:

Type	Tag	Description
Ordered List	`<ol>`	Displays items in a numbered sequence
Unordered List	`<ul>`	Displays items with bullet points
List Item	`<li>`	Defines each item inside `<ol>` or `<ul>`

Basic Syntax:

Ordered List:

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

Unordered List:

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

Example:

```
<h3>Shopping List</h3>
<ul>
  <li>Milk</li>
  <li>Bread</li>
  <li>Eggs</li>
</ul>

<h3>Steps to Make Tea</h3>
<ol>
  <li>Boil water</li>
  <li>Add tea leaves</li>
  <li>Pour into cup</li>
</ol>
```

Output:

Shopping List

- Milk
- Bread
- Eggs

Steps to Make Tea

1. Boil water
2. Add tea leaves
3. Pour into cup

HTML Tables

⇨ HTML tables organize data into rows and columns, providing a structured way to display tabular information on a webpage.

Key Tags:

Tag	Description
`<table>`	Defines the table container
`<tr>`	Table row
`<th>`	Table header cell (bold and centered by default)
`<td>`	Table data cell (regular cell)

Example:

```
<table border="1">
  <tr>
    <th>Name</th>
    <th>Age</th>
    <th>City</th>
  </tr>
  <tr>
    <td>John</td>
    <td>25</td>
    <td>New York</td>
  </tr>
  <tr>
    <td>Mary</td>
    <td>30</td>
    <td>London</td>
  </tr>
</table>
```

Output:

Name	Age	City
John	25	New York
Mary	30	London

Notes:

- The `border` attribute adds a border around the
- Tables can include captions using `<caption>` tag (optional).
- You can merge cells using `colspan` and `rowspan` attributes.

HTML Forms

⇨ HTML forms collect user input and send it to a server for processing. Forms are essential for **interaction, such as logging in, signing up, or submitting data**.

Key Tags and Attributes:

Tag	Purpose
`<form>`	Container for form elements
`<input>`	Various input fields (text, password, etc.)
`<textarea>`	Multi-line text input
`<label>`	Defines labels for input elements
`<button>`	Clickable button
`<select>`	Drop-down list
`<option>`	Options inside `<select>`

Important <form> Attributes:

Attribute	Description
`action`	URL where form data is sent
`method`	HTTP method for submission (`GET` or `POST`)

Example:	Output:

```
<form action="/submit-form" method="post">
  <label for="email">Email:</label>
  <input type="email" id="email" name="email"><br><br>

  <label for="pwd">Password:</label>
  <input type="password" id="pwd" name="password"><br><br>

  <button type="submit">Login</button>
</form>
```

HTML <input> Tag Types

- The `<input>` element is used to create **interactive controls** in a web form. It has **many types** depending on the kind of data you want to collect.

Common <input> Types and Their Uses:

Type	Purpose	Example
`text`	Single-line text input	`<input type="text" name="username">`
`password`	Password input (masked)	`<input type="password" name="pwd">`
`email`	Email address input with validation	`<input type="email" name="email">`
`number`	Numeric input	`<input type="number" name="age">`
`tel`	Telephone number input	`<input type="tel" name="phone">`
`url`	URL input	`<input type="url" name="website">`
`checkbox`	Checkbox for multiple selections	`<input type="checkbox" name="subscribe">`
`radio`	Radio buttons for single choice selection	`<input type="radio" name="gender" value="male">`
`submit`	Submit button to send form data	`<input type="submit" value="Send">`
`reset`	Reset button to clear form inputs	`<input type="reset" value="Clear">`
`button`	General clickable button	`<input type="button" value="Click Me">`
`file`	File selector for uploading files	`<input type="file" name="upload">`
`hidden`	Hidden input not visible to user	`<input type="hidden" name="userid" value="123">`
`date`	Date selector	`<input type="date" name="birthdate">`
`color`	Color picker	`<input type="color" name="favcolor">`
`range`	Slider control for numeric range	`<input type="range" name="volume" min="0" max="10">`
`datetime-local`	Date and time selector (local time)	`<input type="datetime-local" name="appt">`
`month`	Month and year selector	`<input type="month" name="monthyear">`
`time`	Time selector	`<input type="time" name="alarm">`
`week`	Week selector	`<input type="week" name="weekyear">`

Example:	Output:
<form> <input type="text" name="username" placeholder="Username"> <input type="password" name="pwd" placeholder="Password"> <input type="email" name="email" placeholder="Email"> <input type="checkbox" name="subscribe" value="newsletter"> Subscribe to newsletter <input type="radio" name="gender" value="male"> Male <input type="radio" name="gender" value="female"> Female <input type="submit" value="Submit"> </form>	

HTML Non-SemanticElements

➔ **Non-semantic HTML elements** do **not describe** their content's meaning or role in the webpage. They are used mainly for **layout and styling purposes** but offer **no meaning to browsers or search engines**. Non-semantic elements are **generic containers**.

Common Non-Semantic Elements:

Tag	Purpose
	Generic block-level container
	Generic inline container

Usage:

- ** ** is commonly used to group content into sections (e.g., header, content, sidebar).
- **** is used to apply styles or manipulate **inline content**.

Example:

```
<div>
  <h2>This is a div container</h2>
  <p>This is some content inside a div.</p>
</div>

<p>This is a <span>highlighted</span> word using span.</p>
```

HTML Semantic Elements

- **Semantic elements** in HTML clearly describe their **meaning** in both the **browser** and the **developer's perspective**. Unlike non-semantic elements (like `<div>` or `<span>`), semantic elements define the **structure** and **purpose** of the content inside them.

    - Improves **code readability**
    - Enhances **SEO (Search Engine Optimization)**
    - Boosts **accessibility** for screen readers
    - Makes layout and content **easier to maintain.**

Common Semantic Elements:

Tag	Purpose
`<header>`	Defines the header of a section or page
`<nav>`	Defines a navigation bar or links
`<main>`	Represents the main content of the document
`<section>`	Defines a generic section of content
`<article>`	Self-contained, independent content
`<aside>`	Sidebar content, related but separate
`<footer>`	Defines the footer of a section or page
`<figure>`	Used to wrap media (image, chart, etc.)
`<figcaption>`	Caption for the `<figure>` content
`<details>`	Used for collapsible content
`<summary>`	Visible heading for `<details>` section
`<mark>`	Highlights text
`<time>`	Represents a time or date

Example:

```
<main>
  <header>
    <h1>Welcome to My Website</h1>
  </header>

  <nav>
    <a href="#home">Home</a> |
    <a href="#about">About</a>
  </nav>

  <section>
    <article>
      <h2>Blog Post</h2>
      <p>This is a sample blog post using semantic tags.</p>
    </article>
  </section>

  <aside>
    <p>Related Links or Ads</p>
  </aside>

  <footer>
    <p>Copyright © 2025</p>
  </footer>
</main>
```

Output:

HTML Comments

⇨ **HTML comments** are notes written in the code that are **ignored by the browser**. They are useful for explaining code, reminding yourself or others about functionality, or temporarily disabling parts of the code

Basic Syntax:

```
<!-- This is a comment -->
```

Note:

- Anything between <!-- and --> is treated as a comment.
- Comments can span single or multiple lines.

Example:

```
<!-- This is the main header section -->
<h1>Welcome to My Website</h1>

<!-- Temporarily hiding this paragraph
<p>This paragraph is under construction.</p>
-->

<!-- Remember to update the link below -->
<a href="about.html">About Us</a>
```

Key Uses:

- Add **notes or reminders** for developers.
- Explain sections of code for better **readability**.
- Temporarily **disable code** without deleting it.
- Helps in **team collaboration**.

HTML Attributes

⇨ **HTML attributes** provide **additional information** about HTML elements. They are always specified in the **start tag** and usually come in **name-value pairs**.

Common HTML Attributes:

Attribute	Description	Used With
`href`	Specifies the URL of a link	`<a>`
`src`	Specifies the path of an image	`<img>`
`alt`	Alternative text for an image	`<img>`
`title`	Tooltip text shown on hover	All elements
`style`	Inline CSS to style an element	All elements
`id`	Unique identifier for an element	All elements
`class`	Class name for grouping elements (used with CSS/JS)	All elements
`width/height`	Specifies size of image, table, etc.	`<img>`, `<table>`, etc.
`type`	Specifies the type of input or button	`<input>`,`<button>`
`value`	Specifies the default value	`<input>`, `<button>`
`name`	Name of the input element (important in forms)	`<input>`, `<form>`

Example:

```
<img src="logo.png" alt="Company Logo" width="200" height="100">

<a href="https://google.com" title="Go to Google" target="_blank">Google</a>

<p id="intro" class="highlight" style="color:blue;">Welcome!</p>
```

Notes:

- Attributes modify the **behavior**, **appearance**, or **function** of elements.
- Some elements require specific attributes to work correctly (e.g., `src` in `<img>`, `href` in `<a>`).
- **Multiple attributes** can be used in one tag.

HTML Multimedia

⇨ **HTML Multimedia** refers to the use of **audio** and **video** content on web pages. HTML5 introduced built-in support for multimedia using `<audio>` and `<video>` tags without needing external plugins.

Key Tags:

Tag	Purpose
`<audio>`	Embeds sound/audio content
`<video>`	Embeds video content
`<source>`	Specifies multiple media file sources
`controls`	Adds play, pause, and volume controls
`autoplay`	Automatically starts playing
`loop`	Repeats media after it ends
`muted`	Starts media in a muted state

Audio Syntax:

```
<audio controls>
  <source src="audiofile.mp3" type="audio/mpeg">
  Your browser does not support the audio tag.
</audio>
```

Video Syntax:

```
<video width="320" height="240" controls>
  <source src="video.mp4" type="video/mp4">
  Your browser does not support the video tag.
</video>
```

What is a Favicon?

⇨ A **favicon** is the small icon shown in the **browser tab**, **bookmark bar**, and **history** next to the page title. It helps users recognize your website.

Syntax to Add a Favicon:

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

Notes:

- `href`: Path to your icon file (usually `.ico`, `.png`, or `.svg`)
- The icon file (`favicon.ico`) should be in the same folder or correctly linked
- `type` attribute is optional but recommended

Example:

```
<head>
  <title>My Website</title>
  <link rel="icon" href="images/favicon.png" type="image/png">
</head>
```

Recommended Sizes:

- `.ico` file: 16x16, 32x32, or 48x48
- `.png` file: 32x32 or 64x64

 Tag in HTML

⇨ The `<br>` tag in HTML stands for **"line break"**. It is used to insert a **line break** in text — just like pressing **Enter** on your keyboard.

Example:	Output:
`<p>` `  Hello there! ` `  This is line 2. ` `  And this is line 3.` `</p>`	Hello there! This is line 2. And this is line 3.


CSS

What is CSS?

⇨ **CSS** stands for **Cascading Style Sheets**. It is a **stylesheet language** used to **describe the presentation** (layout and appearance) of HTML documents.

Purpose of CSS:

- Separates **content (HTML)** from **design (CSS)**
- Controls how HTML elements are displayed
- Helps create **consistent**, **responsive**, and **visually appealing** websites

Why Use CSS?

Without CSS	With CSS
Plain, unstyled content	Visually styled, structured, attractive
Hard to maintain styles	Styles are reusable and easier to manage
Style mixed with content	Style kept separate for cleaner code
	

CSS is Used For:

- Setting colors, fonts, margins, padding, borders
- Positioning elements
- Animations and transitions
- Making layouts responsive

Example:

HTML:

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

CSS:

```
<style>
  p {
    color: blue;
    font-size: 18px;
  }
</style>
```

Ways to Apply CSS

⇨ CSS can be applied in **three different ways** based on how and where you define the styles.

- **Inline CSS** (highest priority)
- **Internal CSS** (medium priority)
- **External CSS** (lowest priority)

1. Inline CSS:

- Applied **directly inside** an HTML element using the `style` attribute.
- Affects **only that specific element**.

Example:	Output:
`<p style="color: red; font-size: 18px;">This is inline CSS</p>`	This is inline CSS

2. Internal CSS:

- Written **inside the `<style>` tag** within the `<head>` section of an HTML document.
- Affects **only the current HTML page**.

Example:

```
<head>
  <style>
    h1 {
      color: green;
      text-align: center;
    }
    p {
      font-size: 16px;
      color: blue;
    }
  </style>
</head>
<body>
  <h1>Welcome to My Web Page</h1>
  <p>This is a simple HTML.</p>
</body>
```

Output:

Welcome to My Web Page

This is a simple HTML.

3. External CSS:

- CSS is written in a **separate file** with `.css` extension (e.g., `style.css`).
- Linked using `<link>` tag inside the `<head>` of the HTML file.
- Used for **styling multiple web pages consistently**.

Example:

HTML:

```
<head>
<link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Welcome to My Web Page</h1>
  <p>This is a simple HTML.</p>
</body>
```

CSS:

```
h1 {
  color: blueviolet;
  text-align: center;
}
p {
  font-size: 16px;
  color: blue;
}
```

Output:

Welcome to My Web Page

This is a simple HTML.

Notes:

- Best practice for all modern web development
- Cleaner and more maintainable

CSS Syntax

⇨ CSS syntax is the way CSS rules are written to define how HTML elements should be styled. Each rule consists of a **selector**, a **property**, and a **value**.

Basic Structure:

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

- **Selector**: Identifies the HTML element(s) you want to style
- **Property**: The style attribute you want to change (e.g., `color`, `font-size`)
- **Value**: The setting you want to apply to the property (e.g., `red`, `16px`)

Example:

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

- `h1` → selector (targets all `<h1>` elements)
- `color` → property
- `blue` → value
- `font-size: 24px;` → another property-value pair

Multiple Properties:

⇨ You can define multiple properties inside one selector block by separating them with semicolons.

Example:

```
p {
  color: black;
  line-height: 1.6;
  text-align: justify;
}
```

Notes:

- Always end each property with a `;` (semicolon)
- Enclose the property block in `{}` curly braces
- Use consistent spacing and indentation for readability

CSS Selectors

⇨ CSS **selectors** are patterns used to **select** and **style HTML elements**. They tell the browser **which element(s)** to apply the styles to.

Basic Types of CSS Selectors:

Selector	Description	Example
* (Universal)	Selects **all** elements	* {margin: 0;}
element	Selects all elements of a specific type	p {color: blue;}
.class	Selects all elements with a specific class	.intro {font-size: 20px;}
#id	Selects a specific element with an ID	#header {background: gray;}
element1, element2	Selects multiple elements	h1, h2 {color: navy;}

Combinator Selectors:

Selector	Meaning	Example
A B	Selects all B inside A (descendant)	div p {color: red;}
A > B	Selects all B that are **direct children** of A	ul > li {list-style: none;}
A + B	Selects B **immediately after** A	h1 + p {margin-top: 0;}
A ~ B	Selects all B **after** A (siblings)	h1 ~ p {color: green;}

Pseudo-element Selectors:

Selector	Purpose	Example
::before	Inserts content **before** the element content	p::before {content: "Note: ";}
::after	Inserts content **after** the element content	p::after {content: "✓";}
::first-line	Styles only the first line of text in an element	p::first-line {font-weight: bold;}

CSS Colors and Units

⇨ CSS supports different ways to define colors. You can use:

1. Color Names: Predefined color names

```
h1 {
  color: red;
}
```

2. HEX Values: Begins with #, followed by 3 or 6 hexadecimal digits

```
p {
  color: #00ff00; /* Bright green */
}
```

3. RGB Values: Defines colors using Red, Green, Blue. Range: 0 to 255

```
div {
  color: rgb(255, 0, 0); /* Red */
}
```

4. RGBA Values: Same as RGB but includes Alpha (opacity). Range of Alpha: 0 to 1

```
span {
  color: rgba(0, 0, 255, 0.5);
}
```

CSS Units

Absolute Units

Unit	Description
px	Pixels (most common)
cm	Centimeters
mm	Millimeters
in	Inches
pt	Points (1 pt = 1/72 inch)
pc	Picas (1 pc = 12 pt)

Relative Units

Unit	Description
%	Relative to parent element
em	Relative to the font-size of the element
rem	Relative to the font-size of the root element (<html>)
vw	Relative to 1% of the **viewport width**
vh	Relative to 1% of the **viewport height**
vmin	Smaller of vw or vh
vmax	Larger of vw or vh

Example:

```
body {
  background-color: #f0f0f0;
  font-size: 1.2rem;
}

.container {
  width: 80%;
  height: 50vh;
  padding: 20px;
}
```

CSS Text Styling Properties

⇨ These properties control the **appearance** and **alignment** of text in HTML elements.

1. color:

➢ Sets the text color.

Example:

HTML:

```
<head>
<link rel="stylesheet" href="style.css">
</head>
<body>
  <p>This is a simple HTML.</p>
</body>
```

CSS:

```
p {
  color: blue;
}
```

Output:

This is a simple HTML.

2. text-align:

➢ Aligns the text inside an element.

Value	Meaning
`left`	Aligns text to the left
`right`	Aligns text to the right
`center`	Centers the text
`justify`	Aligns text evenly

Example:

```
h1 {
  text-align: center;
}
```

Output:

This is heading 1

3. text-decoration:

- Adds decoration like underline, overline, or removes it.

Value	Meaning
`none`	No decoration
`underline`	Underlines text
`line-through`	Strikes text
`overline`	Adds line above

Example:	Output:
`a { text-decoration: none; }`	Google.com

4. text-transform:

- Controls letter casing.

Value	Meaning
`uppercase`	Converts to UPPERCASE
`lowercase`	Converts to lowercase
`capitalize`	Capitalizes each word

Example:	Output:
`h2 { text-transform: uppercase; }`	**THIS IS HEADING 2**

5. letter-spacing & word-spacing:

- Adds space between letters or words.

Example:	Output:
`p { letter-spacing: 2px; word-spacing: 5px; }`	This is a simple HTML.

6. line-height:

- Controls the space between lines of text.

Example:	Output:
`p { line-height: 1.6; }`	This is a simple HTML.

CSS Font Styling Properties

1. font-family: Sets the font of the text. Always provide **fallback fonts**.

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

2. font-size: Defines the size of the text.

```
h1 {
  font-size: 32px;
}
```

3. font-style: Sets the style of the font.

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

Value	Description
normal	Default
italic	Italicized text
oblique	Slanted text

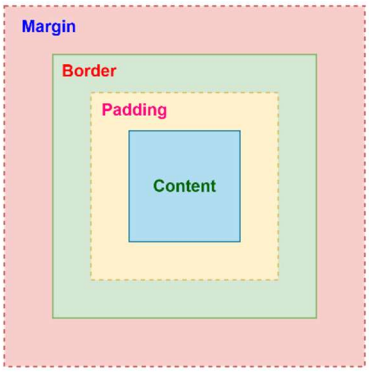
4. font-weight: Defines the thickness of the text.

```
span {
  color: rgba(0, 0, 255, 0.5);
}
```

Value	Description
normal	Regular
bold	Bold
100–900	Numeric weight scale

CSS Box Model

- The **CSS Box Model** is a fundamental concept that describes how elements are **displayed and spaced** on a web page.
- Every HTML element is considered a **box** composed of:



Parts of the Box Model:

Part	Description
Content	Actual text/image inside the box
Padding	Space between content and border
Border	Wraps the padding and content; thickness/color customizable
Margin	Space outside the border (separates element from others)

Example:	**Output:**
`div { width: 200px; padding: 20px; border: 5px solid blue; margin: 30px; }`	Box Model

$$\textbf{Total width} = width + left/right\ padding + left/right\ border + left/right\ margin$$
$$= 200 + 40 + 10 + 60 = 310px$$

Box-sizing Property:

⇨ Controls how the total size of the box is calculated.

Value	Description
`content-box`	Default. Width/height apply **only** to content; padding and border **added**.
`border-box`	Width/height include **content + padding + border** (recommended for layout).

Example:	Output:
`.card { width: 300px; padding: 10px; border: 1px solid #ccc; margin: 20px; box-sizing: border-box; }`	Sample Image **Card Title** This is a sample card component with an image, title, and description.

CSS Positioning and Layout

⇨ CSS **positioning** is used to place HTML elements **precisely** on the webpage. It defines **how elements are laid out**, stacked, and behave in the document flow.

`position` Property Values:

Value	Description
`static`	Default. Elements follow the normal document flow.
`relative`	Moves element **relative to its normal position**.
`absolute`	Positions element **relative to the nearest positioned ancestor** (not static).
`fixed`	Positions element **relative to the browser window** (doesn't scroll).
`sticky`	Switches between `relative` and `fixed` based on scroll position.

Visual Representation of CSS `position` Property Values

Default Position: static

The default position for an element is `position: static`, which doesn't have any effect on the element.

Consider the figure where all the cats are sitting together:

A Relatively Positioned Element:

An element with `position: relative` stays in its place but can be moved using `top, bottom, left, or right`.

In the picture, a naughty cat has `moved down` without permission.

An Absolutely Positioned Element:

An element with `position: absolute` moves based on its nearest positioned parent, not its original place. You can use `top, bottom, left, or right` to move it.

In the picture, a naughty cat ran away to the `top-left` corner without permission.

A Fixed Positioned Element:

An element with `position: fixed` stays in the same place on the screen, even when you scroll. It moves using `top, bottom, left, or right,` based on the viewport (the screen). But if its parent has `transform, filter, or perspective`, that parent becomes the reference.

In the picture, the cat has escaped to the `top-left` corner of the viewport. This is where `position: fixed` become handy.

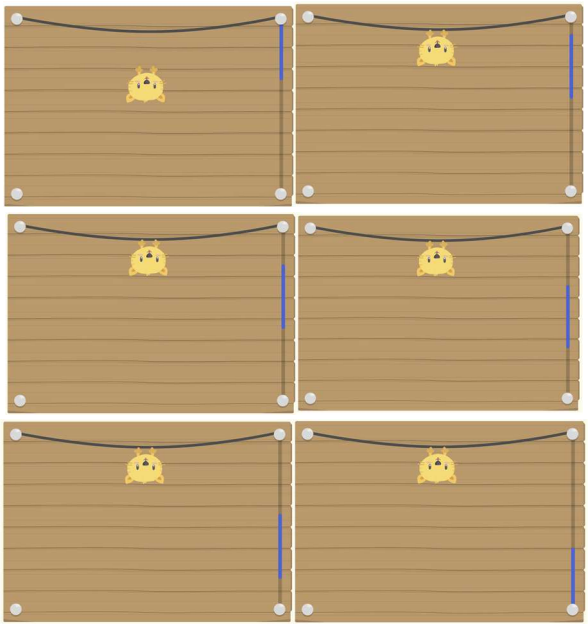
Sticky Positioning

The food is in the center of the wooden box, and the cat wants to stick to the rope at the top. This is where `position: sticky` helps.

With `position: sticky`, the cat stays in place at the top center while you scroll inside the wooden box. Try scrolling to see how it sticks!

```
.wood-wrapper {
  position: relative;
}

.the-cat {
  position: sticky;
  top: 0;
}
```



Top, Right, Bottom, Left:

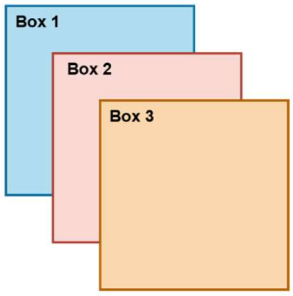
⇨ These properties adjust the element's position **when** `position` is not `static`.

Example:

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

z-index:

- Controls the **stacking order** of elements (which appears in front).
- Higher value = more front.

Example:	Output:
`.box1 { z-index: 1; } .box2 { z-index: 2; } .box3 { z-index: 3; }`	

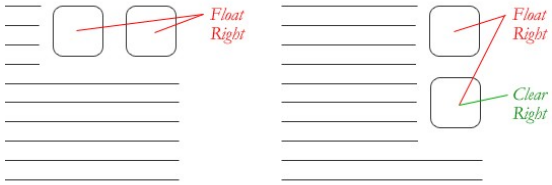
CSS Layout Techniques

1. `display` Property:

Value	Effect
`block`	Element takes full width
`inline`	Element takes only the space it needs
`inline-block`	Like inline but allows box styling
`none`	Hides the element
`flex`	Enables Flexbox layout
`grid`	Enables CSS Grid layout

2. `float` and `clear`:

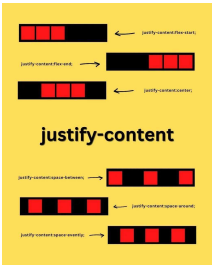
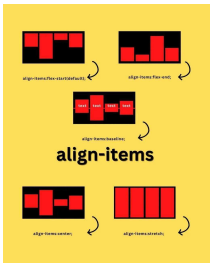
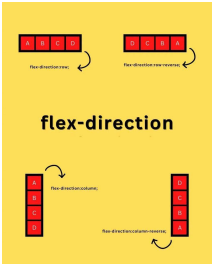
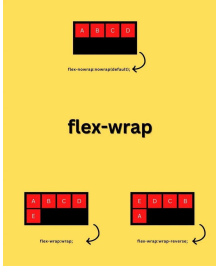
➔ Used to wrap content around elements (e.g., images).

Example:	Output:
`box { float: right; clear: right; } p { clear: left; }`	

3. Flexbox:

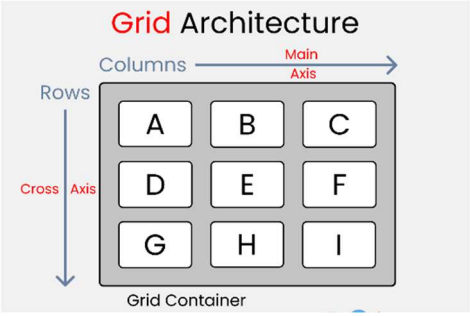
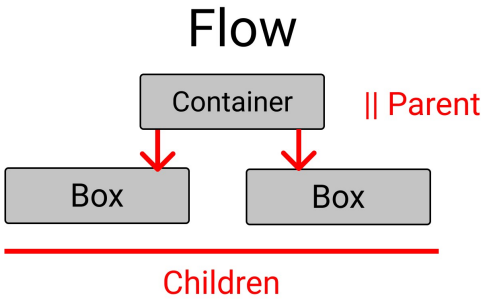
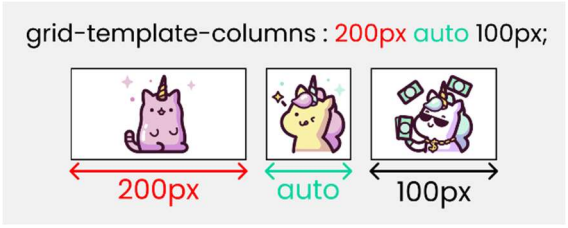
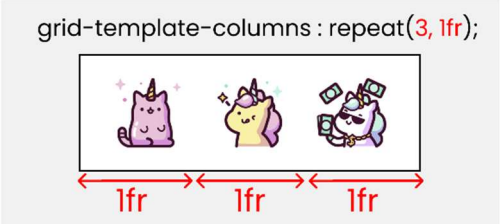
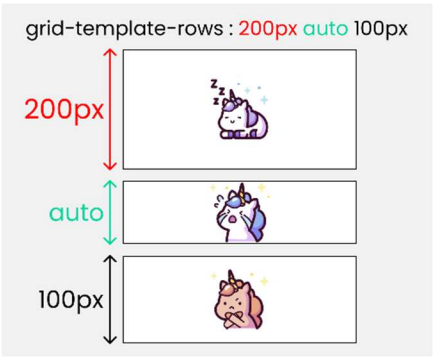
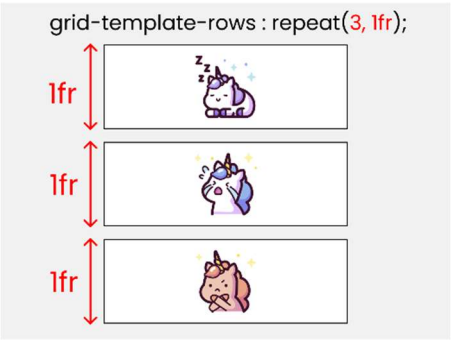
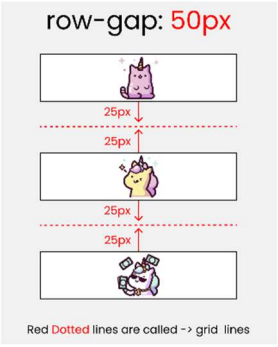
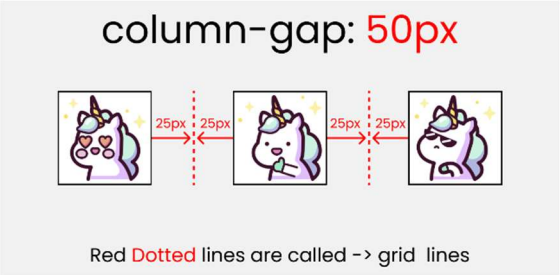
- Powerful 1D layout system.
- Key properties:
  - `display: flex`
  - `justify-content`
  - `align-items`
  - `flex-direction`

Example:	Output:
`.container { display: flex; justify-content: space-between; align-items: center; }`	

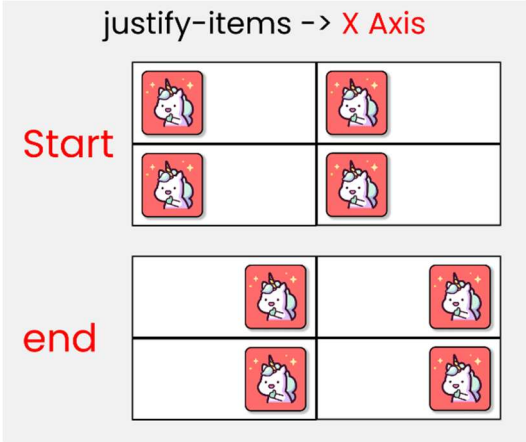
`justify-content`	`align-items`
	
`flex-direction`	`flex-wrap`
	

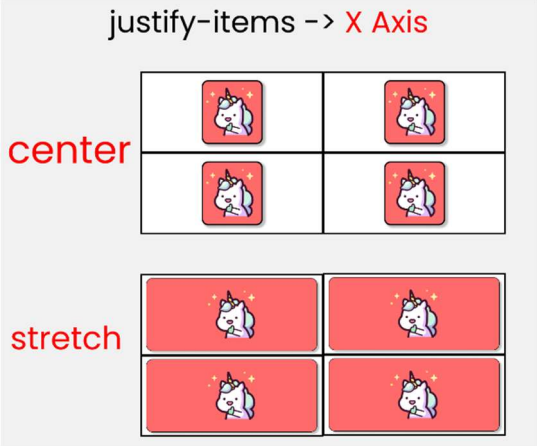
4. CSS Grid :

➔ 2D layout system (rows + columns)

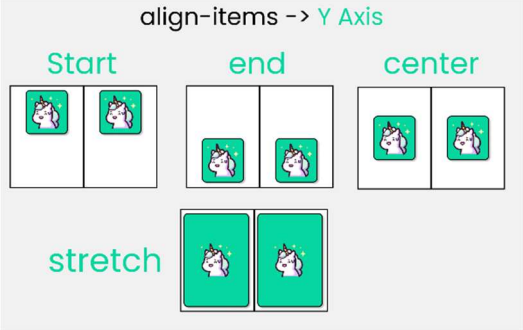
<table>
<tr><th>CSS Grid Architecture:</th><th>Relationship between parent and child classes:</th></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><th colspan="2">The grid-template-columns property:</th></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><th colspan="2">The grid-template-row property:</th></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><th>The row-gap property:</th><th>The column-gap property:</th></tr>
<tr><td> 
</td><td> 
</td></tr>
</table>

The justify-items property:

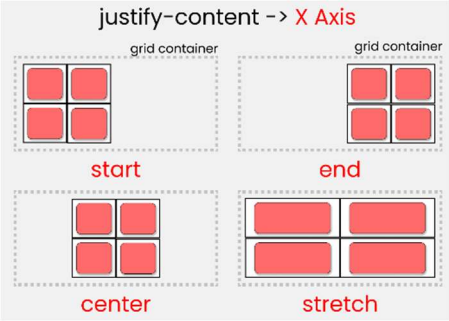


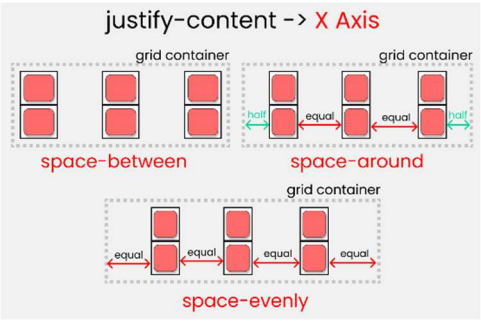


The align-items property:

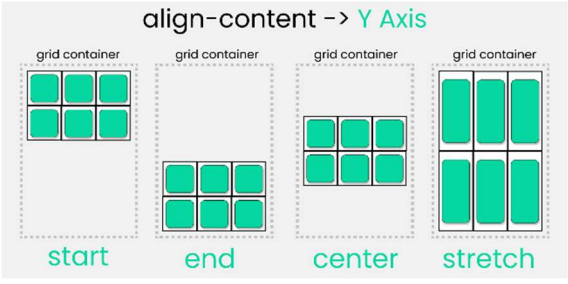


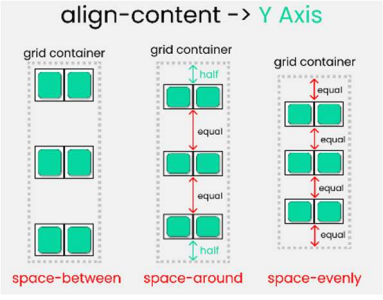
The justify-content property:





The align-content property:





The justify-self property:

The align-self property:

Shorthand for CSS Grid Properties:

place-content:

place-content : align-content / justify-content

↑ Y-axis ↑ X-axis

place-items:

place-items : align-items / justify-items

↑ Y-axis ↑ X-axis

place-self:

place-self : align-self / justify-self

↑ Y-axis ↑ X-axis

grid-template:

grid-template : grid-template-rows / grid-template-columns

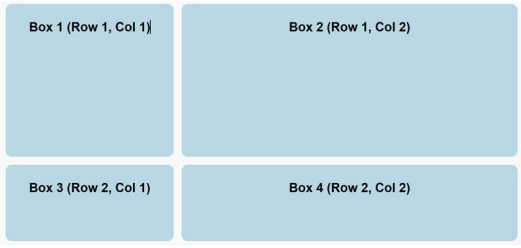
gap/grid-gap:

gap : row-gap column-gap;

Example:

```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr;
  grid-template-rows: 2fr 1fr;
  gap: 10px;
}
```

Output:



CSS Layout Techniques

➔ CSS Background Properties:

⇨ Used to style the **background** of an element (color, image, position, size, etc.).

1. background-color:

⇨ Sets the background color.

Example:	Output:
`div { background-color: lightblue; }`	Box 1 Box 2 Box 3

2. background-image:

⇨ Sets an image as the background.

Example:	Output:
`div { background-image: url("background.jpg"); }`	Box 1 Box 2 Box 3

3. background- repeat:

⇨ Controls how the background image repeats.

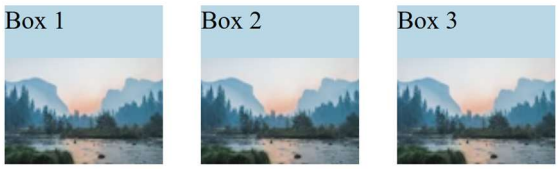
Value	Description
repeat	Repeats both horizontally and vertically (default)
no-repeat	Image appears once only
repeat-x	Repeats horizontally only
repeat-y	Repeats vertically only

Example:	Output:
`div { background-repeat: no-repeat; }`	Box 1 Box 2 Box 3

4. background-position:

⇨ Defines the starting position of the background image.

Example:	Output:

`div {`
`  background-position: center bottom;`
`}`
	

5. background-size:

⇨ Specifies the size of the background image.

Value	Description
auto	Default size
cover	Scales image to cover element
contain	Scales image to fit element

Example:	Output:

`div {`
`  background-size: cover;`
`}`
	

background (shorthand):

Example:	Output:

`div {`
`  background: url("bg.jpg") no-repeat center center / cover;`
`}`
	

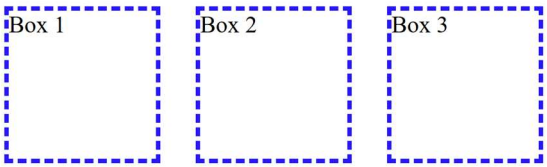
CSS Borders

⇨ Used to add borders around elements.

1. `border-width, border-style, border-color:`

Example:	Output:

```
div {
  border-width: 3px;
  border-style: dashed;
  border-color: blue;
}
```

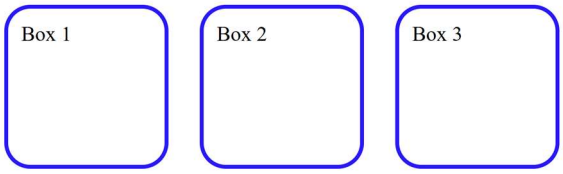


2. `border-radius:`

⇨ Rounds the corners of the element.

Example:	Output:

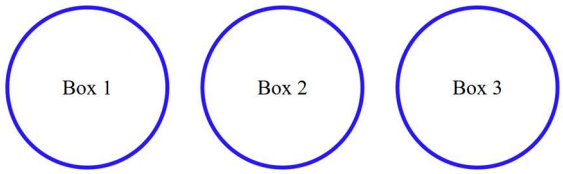
```
div {
  border-radius: 20px;
}
```



⇨ We can create a circle by using `border-radius: 50%.`

Example:	Output:

```
div {
  border-radius: 50%;
}
```



2. `border- shadow:`

⇨ Applies shadow around an element's box.

Example:	Output:

```
div {
  box-shadow: 0 4px 8px red;
}
```

CSS Transitions and Animations

CSS Transitions

⇨ Transitions make CSS property changes happen **smoothly over time**, instead of instantly.

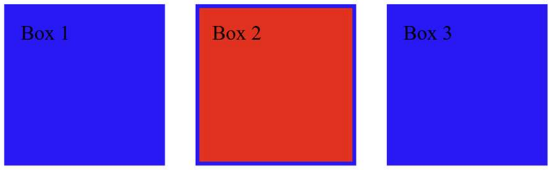
1. `transition` (shorthand property) :

Example:

```css
div {
  background-color: blue;
  transition: background-color 0.5s ease;
}

div:hover {
  background-color: red;
}
```

Output:



⇨ Breakdown of Syntax:

```
transition: property duration timing-function delay;
```

Part	Example	Description
`property`	`background-color`	CSS property to animate
`duration`	`0.5s`	Time it takes (in seconds or milliseconds)
`timing-function`	`ease`	Controls the speed curve
`delay`	`0s`	Delay before animation starts

⇨ Common `timing-function` values:

- `linear` – same speed from start to end
- `ease` – slow start, fast, then slow end
- `ease-in` – slow start
- `ease-out` – slow end
- `ease-in-out` – slow start and end

Note: For a better understanding of transitions, **visit**: https://web.dev/learn/css/transitions

Want cool transition effects? **Visit this site**: https://www.transition.style/#in:circle:hesitate

CSS Animations

⇨ Animations define **keyframes** to create more complex movements or effects.

1. `@keyframes` Rule

```
@keyframes slideRight {
  from {
    transform: translateX(0px);
  }
  to {
    transform: translateX(200px);
  }
}
```

2. Applying an Animation

```
div {
  background: green;
  animation: slideRight 2s ease-in-out
infinite;
}
```

Animation Properties:

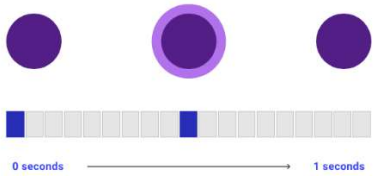
Property	Description
`animation-name`	Name of the keyframes rule
`animation-duration`	How long the animation runs
`animation-timing-function`	Speed curve
`animation-delay`	Delay before it starts
`animation-iteration-count`	Number of times it repeats (`infinite` or number)
`animation-direction`	`normal`, `reverse`, `alternate`, etc.

Example:

```
@keyframes pulse {
  0%   { transform: scale(1); }
  50%  { transform: scale(1.1); }
  100% { transform: scale(1); }
}

div {
  animation: pulse 1s infinite;
}
```

Output:



Note:

⇨ For a better understanding of Animations, **visit**: https://prismic.io/blog/css-animation-examples

⇨ Want cool Animations effects? **Visit this site**: https://animate.style/

Responsive CSS

⇨ **Responsive CSS** ensures that a website **adapts** its layout and appearance across different screen sizes and devices—like mobiles, tablets, laptops, and desktops.

Why Use Responsive Design?

- Improves **user experience**
- Ensures **accessibility** on all devices
- Makes the layout **fluid and flexible**
- Reduces the need for a separate mobile site

Key Concepts in Responsive CSS

1. Viewport Meta Tag (HTML)

➔ Use this in the `<head>` of your HTML to control layout scaling:

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

2. Media Queries

➔ Allows you to **apply CSS rules** only when **certain conditions** (like screen width) are met.

```
@media (max-width: 768px) {
  body {
    background-color: lightgray;
  }
}
```

Common Breakpoints	Device Type
`max-width: 480px`	Mobile (portrait)
`max-width: 768px`	Tablets & small screens
`max-width: 1024px`	Desktops & laptops

3. Relative Units

➔ Use **flexible units** for layout and font sizes:

- `%, em, rem, vw, vh`
- Avoid using fixed `px` for responsive elements

```
.container {
  width: 80%;
  padding: 2vw;
}
```

4. Flexible Layouts with Flexbox/Grid

- `flex-wrap: wrap;` helps in making flex containers responsive
- CSS Grid allows auto-fitting with `minmax()` and `auto-fit`

```
.container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
}
```

5. Responsive Images

- `max-width: 100%` → Prevents the image from overflowing its container.
- `height: auto` → Keeps the image's aspect ratio.

```
<img src="image.jpg" style="width: 100%; height: auto;">
```

Best Practices

- Always test on multiple devices/screen sizes
- Start with **mobile-first** design using `min-width` media queries
- Keep layout **fluid** using `%` **or** `vw/vh` instead of fixed sizes

Common CSS Examples

1. Center Text

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

2. Center a Div Horizontally

```css
.center-div {
  width: 300px;
  margin: 0 auto;
}
```

3. Responsive Image

```css
img {
  width: 100%;
  height: auto;
}
```

4. Rounded Corners and Shadow

```css
.card {
  border-radius: 10px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
}
```

5. Button Hover Effect

```css
button {
  background-color: blue;
  color: white;
  padding: 10px 20px;
  border: none;
  transition: 0.3s;
}

button:hover {
  background-color: darkblue;
}
```

6. Sticky Navbar

```css
.navbar {
  position: sticky;
  top: 0;
  background-color: #fff;
  padding: 10px;
  z-index: 999;
}
```

7. Text with Ellipsis (when too long)

```css
.text-ellipsis {
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

8. Responsive Grid Layout

```css
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 20px;
}
```

8. Flexbox Centering

```css
.flex-center {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}
```

10. Fade In Animation

```css
@keyframes fadeIn {
  from { opacity: 0; }
  to { opacity: 1; }
}
.fade-in {
  animation: fadeIn 1s ease-in;}
```

What is Git?

- Git is a **distributed version control system**.
- Helps track **changes in code**, collaborate with others, and revert back if needed.
- Developed by **Linus Torvalds** in 2005.

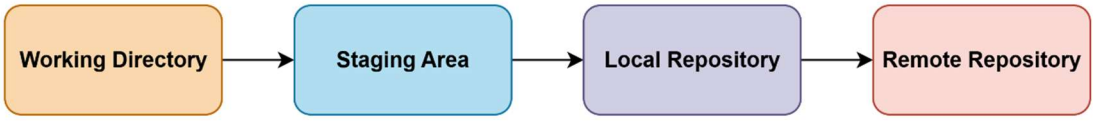
What is GitHub?

- GitHub is a **cloud-based hosting service** for Git repositories.
- Offers GUI tools, issue tracking, pull requests, and team collaboration.

Basic Git Terminology

Term	Description
Repository	A directory where your project and its version history live
Commit	A snapshot of your code at a given time
Branch	A separate line of development
Merge	Combines changes from one branch into another
Clone	Downloads a remote repository to your local machine
Push	Upload local changes to the remote repository
Pull	Download latest changes from the remote repo
Staging Area	Where changes are prepared before committing
HEAD	Refers to the current branch or commit you're working on

Git Workflow



Essential Git Commands

Command	Description
`git init`	Initialize a new Git repository
`git clone <url>`	Clone a repo from GitHub
`git status`	Show changes in working directory
`git add <file>`	Stage a file for commit
`git commit -m "message"`	Commit changes with a message
`git log`	View commit history
`git branch`	List branches
`git branch <branch-name>`	Create a new branch
`git checkout <branch-name>`	Switch to another branch
`git merge <branch-name>`	Merge a branch into current branch
`git pull`	Fetch and merge changes from remote
`git push`	Upload local changes to GitHub

Useful GitHub Features

Feature	Purpose
Pull Request	Request to merge code into another branch
Fork	Copy someone else's repo to your account
Stars	Like or bookmark a project
Issues	Report bugs or suggest enhancements
Actions	Automate workflows (CI/CD, testing, etc.)

GitHub Workflow

1. Create a **repository** on GitHub
2. **Clone** it to your local system
3. Make changes in code
4. Use `git add`, `git commit`, `git push`
5. Changes appear on GitHub

GitHub
GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

`git config --global user.name "[firstname lastname]"`
set a name that is identifiable for credit when review version history
`git config --global user.email "[valid-email]"`
set an email address that will be associated with each history marker
`git config --global color.ui auto`
set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`
initialize an existing directory as a Git repository
`git clone [url]`
retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`
show modified files in working directory, staged for your next commit
`git add [file]`
add a file as it looks now to your next commit (stage)
`git reset [file]`
unstage a file while retaining the changes in working directory
`git diff`
diff of what is changed but not staged
`git diff --staged`
diff of what is staged but not yet committed
`git commit -m "[descriptive message]"`
commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`
list your branches. a * will appear next to the currently active branch
`git branch [branch-name]`
create a new branch at the current commit
`git checkout`
switch to another branch and check it out into your working directory
`git merge [branch]`
merge the specified branch's history into the current one
`git log`
show all commits in the current branch's history

INSPECT & COMPARE

Examining logs, diffs and object information

`git log`	show the commit history for the currently active branch
`git log branchB..branchA`	show the commits on branchA that are not on branchB
`git log --follow [file]`	show the commits that changed file, even across renames
`git diff branchB...branchA`	show the diff of what is in branchA that is not in branchB
`git show [SHA]`	show any object in Git in human-readable format

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

`git remote add [alias] [url]`	add a git URL as an alias
`git fetch [alias]`	fetch down all the branches from that Git remote
`git merge [alias]/[branch]`	merge a remote branch into your current branch to bring it up to date
`git push [alias] [branch]`	Transmit local branch commits to the remote repository branch
`git pull`	fetch and merge any commits from the tracking remote branch

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`	delete the file from project and stage the removal for commit
`git mv [existing-path] [new-path]`	change an existing file path and stage the move
`git log --stat -M`	show all commit logs with indication of any paths that moved

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`	apply any commits of current branch ahead of specified one
`git reset --hard [commit]`	clear staging area, rewrite working tree from specified commit

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

`git config --global core.excludesfile [file]`	system wide ignore pattern for all local repositories

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`	Save modified and staged changes
`git stash list`	list stack-order of stashed file changes
`git stash pop`	write working from top of stash stack
`git stash drop`	discard the changes from top of stash stack

Mohsin Iban Hossain
AIUB, JavaScript Notes

Table of Contents

INTRODUCTION TO JAVASCRIPT

Introduction

➢ What is JavaScript?

⇨ **JavaScript (JS)** is a high-level, dynamic, interpreted programming language.

⇨ Primarily used to make web pages **interactive** and **dynamic**.

⇨ Runs in the browser but can also run on servers using **Node.js**.

➢ Features of JavaScript?

- **Interpreted**: No need for compilation, runs directly in the browser.
- **Dynamic Typing**: Data types are determined at runtime.
- **Event-Driven**: Can respond to user actions (clicks, keystrokes, etc.).
- **Prototype-Based**: Uses prototypes for inheritance instead of classical classes.
- **Cross-Platform**: Works on all major browsers and devices.

Difference Between JavaScript, Java, and ECMAScript

Feature	JavaScript	Java	ECMAScript
Type	Scripting language	Object-oriented programming	Standard specification for JS
Execution	Runs in browsers & Node.js	Runs in JVM	Not a language, but a standard
Syntax	Loosely typed	Strongly typed	Defines syntax & rules of JavaScript
Usage	Web interactivity	Enterprise applications	Acts as blueprint for JavaScript

➢ History

⇨ Created in **1995** by **Brendan Eich** at Netscape.

⇨ Originally called **Mocha**, then **LiveScript**, and finally **JavaScript**.

⇨ Standardized as **ECMAScript** in 1997 by **ECMA International**.

➢ Why Learn JavaScript?

- **Essential for Web Development**: Alongside HTML & CSS, it's one of the three core web technologies.
- **Versatile**: Works for frontend, backend, mobile apps, desktop apps, and even IoT.
- **Huge Community & Libraries**: jQuery, React, Vue, Angular, Node.js, Express.js.

Variables

⇨ **Variable** is a name used to **store data** in memory. It acts as a container for values.

➢ **Declaring Variables?**

Keyword	Scope	Reassignment	Hoisting	Notes
`var`	Function	Yes	Yes	Old way, avoid in modern JS
`let`	Block	Yes	No	Preferred for variables that change
`const`	Block	No	No	Preferred for constants

➢ **How to Declare a Variable?**

Syntax:	**Example Code:**

var x = 10;
let y = 20;
const z = 30;
	`var x = 10;  // function-scoped`
	`let y = 20;  // block-scoped`
	`const z = 30;  // cannot be reassigned`

Rules for Naming Variables:	
• Must start with a letter or underscore (_) • Cannot start with a digit • Can contain letters, digits, and underscores • Case-sensitive (`age` and `Age` are different)	

Data Types

⇨ JavaScript has **primitive types** and **reference types**.

Primitive Types:		
Type	**Example**	**Description**
string	`"Hello"`	Text data
number	`42, 3.14`	Numbers (integer & float)
boolean	`true, false`	Logical values
null	`null`	Intentional empty value
undefined	`let x;`	Declared but not assigned
symbol	`Symbol('id')`	Unique identifiers
bigint	`123n`	Large integers

Example Code:

```
let name = "Alice";    // string
let age = 20;          // number
let isStudent = true; // boolean
let car = null;        // null
let city;              // undefined
```

Output Methods

Method	Where It Shows	Example
`console.log()`	Browser console	`console.log("Hello")`
`alert()`	Popup box	`alert("Warning!")`
`document.write()`	Directly to web page	`document.write("Hi")`

Example Code: `console.log()`

```
let name_ = "Mohsin";
console.log("Name:", name_)
```

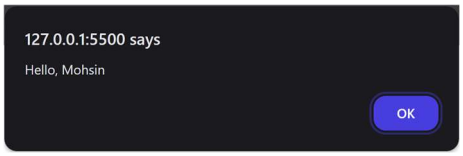
Output:

Mohsin

Example Code: `alert()`

```
let name_ = "Mohsin";
alert("Hello, " + name_);
```

Output:



Example Code: `document.write()`

```
let name_ = "Mohsin";
const country = "Bangladesh";
let age = 21;
document.write(`You are from ${country} and
${age} years old.`);
```

Output:

You are from Bangladesh and 21 years old.

How to Run JavaScript

In HTML:

```
    <!-- Inline -->
<script>
  console.log("Hello World");
</script>

    <!-- External File -->
<script src="script.js"></script>
```

Tip: Place scripts **before** `</body>` so the page loads faster.

Type Conversion

➔ **Type Conversion** means changing the data type of a value or variable.

Conversion	Example	Output
String → Number	`Number("10")`	`10`
Number → String	`String(10)`	`"10"`
Boolean → Number	`Number(true)`	`1`
Automatic (coercion)	`"5" * 2`	`10`

Comments

⇨ **Comments** are notes in the code that are ignored by the browser, used to explain or temporarily disable code.

Type	Syntax	Example
Single-line	`//`	`// This is a comment`
Multi-line	`/* ... */`	`/* This is a multi-line comment */`

Operators

⇨ Operators are **special symbols** used to perform operations on **values or variables**.

Example Code:	Output:

`let a = 5;`
`let b = 2;`
`let sum = a + b; // + is the operator`
`console.log(sum); // 7`
	7

Types of Operators

a) Arithmetic Operators:

⇨ Used for **mathematical calculations**.

Operator	Description	Example	Result
+	Addition	5 + 2	7
-	Subtraction	5 - 2	3
*	Multiplication	5 * 2	10
/	Division	5 / 2	2.5
%	Modulus (remainder)	5 % 2	1
**	Exponentiation	5 ** 2	25
++	Increment (+1)	a++	6
--	Decrement (-1)	b--	1

b) Assignment Operators:

⇨ Used to **assign values** to variables.

Operator	Example	Meaning
=	x = 5	Assign 5 to x
+=	x += 2	x = x + 2
-=	x -= 2	x = x - 2
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2

c) Comparison Operators:

⇨ Used to **compare two values**. Result is **true** or **false**.

Operator	Example	Meaning
`==`	`5 == '5'`	Equal (value only)
`===`	`5 === '5'`	Equal (value & type)
`!=`	`5 != 2`	Not equal (value)
`!==`	`5 !== '5'`	Not equal (value & type)
`>`	`5 > 2`	Greater than
`<`	`5 < 2`	Less than
`>=`	`5 >= 5`	Greater or equal
`<=`	`5 <= 2`	Less or equal

d) Logical Operators:

⇨ Used for **Boolean logic**.

Operator	Example	Result	Meaning
`&&`	`true && false`	`false`	AND — true only if **both** are true
`||`	`true || false`	`true`	OR — true if **at least one** is true
`!`	`!true`	`false`	NOT — flips the Boolean value

In JavaScript:

- `&&` stops checking if the first value is **false** (short-circuiting).
- `||` stops checking if the first value is **true**.
- `!` works on truth/falsie values, not just `true`/`false`.

e) String Operators:

⇨ Mainly **concatenation**.

Example Code:	Output:

`let firstName = "Mohsin";`
`let lastName = "ELI";`
`let fullName = firstName + " " + lastName;`
`console.log(fullName);`
	`Mohsin ELI`

How to See the Data Type

⇨ `typeof` → returns **data type**.

Example Code:	Output:

`let x = 10;`
`let str = "Hello, World!";`
 	
`console.log(typeof x); // number`
`console.log(typeof str); // string`
	`number`
	`string`

Keywords

➔ **Keywords** are reserved words in JavaScript. We cannot use them as variable names or identifiers.

List of Common JavaScript Keywords:

break	case	catch	class	const	continue
debugger	default	delete	do	else	export
extends	finally	for	function	if	import
in	instanceof	new	return	super	switch
this	throw	try	typeof	var	void
while	with	yield	let	enum	await

- **Problem1:** Write a JavaScript program that assign values to variables num1, num2 and perform addition operations. Also print the results.

Code:	Output:

``
let num1 = 10;
let num2 = 5;
let sum = num1 + num2;
 	
console.log("Sum:", sum);
``
	Sum: 15

- **Problem2:** Write a JavaScript program that assign values to variables num1, num2 and perform subtraction operations. Also print the results.

Code:	Output:

``
let num1 = 10;
let num2 = 5;
let sub = num1 - num2;
 	
console.log("Sub:", sub);
``
	Sub: 5

- **Problem3:** Write a JavaScript program that assign values to variables num1, num2 and perform multiplication & division operations. Also print the results.

Code:	Output:

``
let num1 = 10;
let num2 = 5;
 	
let mult = num1 * num2;
let div = num1 / num2;
 	
console.log("Mult:", mult);
console.log("Div:", div);
``
	Mult: 50
	Div: 2.0

Input in JavaScript

➔ The `prompt()` shows a pop-up box that asks the user for input.

Syntax:

```
let variable = prompt("Enter something: ")
```

Example Code:

Example code:

```
let name = prompt("Enter your name:");
console.log("Hello, " + name);
```

Output:

```
Enter your name: Mohsin Ibna Hossain
Hello, Mohsin Ibna Hossain
```

Convert input to other types:

```
let input = prompt("Enter a number:");
console.log("Type:", typeof input);

let num = Number(input);
console.log("Number:", num);
console.log("Type:", typeof num);
```

Output:

```
Enter a number: 23
Type: string
Number: 23
Type: number
```

➔ **Note:** In JavaScript, when you take user input (usually with `prompt()` in the browser), it comes in as a **string**, so you need to convert it to other types like **number** or **boolean** if needed.

Conversion Type	Method	Example	Result
String → Number (integer)	`parseInt()`	`parseInt("42")`	`42`
String → Number (float)	`parseFloat()`	`parseFloat("3.14")`	`3.14`
String → Number (auto)	`Number()`	`Number("42.5")`	`42.5`
Number → String	`.toString()` or `String()`	`(42).toString()`	`"42"`
String → Boolean	`Boolean()`	`Boolean("Hello")`	`true`
Boolean → Number	`Number(true)`	`Number(true)`	`1`

Practice Problem

- **Problem1:** Write a Program to input 2 numbers & print their sum.

Code:	Output:

`let num1 = Number(prompt("Enter number 1:"));`
`let num2 = Number(prompt("Enter number 2:"));`
`let sum = num1 + num2;`
 	
`console.log("Sum:", sum);`
	`Enter number 1: 10`
	`Enter number 2: 20`
	`Sum: 30`

- **Problem2:** Write a JavaScript program to take five float variables as input from the user. Find out the summation and average of five numbers

Code:	Output:

`let num1 = parseFloat(prompt("Enter number 1:"));`
`let num2 = parseFloat(prompt("Enter number 2:"));`
`let num3 = parseFloat(prompt("Enter number 3:"));`
`let num4 = parseFloat(prompt("Enter number 4:"));`
`let num5 = parseFloat(prompt("Enter number 5:"));`
 	
`let summ = num1 + num2 + num3 + num4 + num5;`
`let average = summ / 5;`
 	
`console.log("Summation:", summ);`
`console.log("Average:", average);`
	`Enter number 1: 10`
	`Enter number 2: 3`
	`Enter number 3: 7`
	`Enter number 4: 6`
	`Enter number 5: 8`
	`Summation: 34`
	`Average: 6.8`

CONDITIONAL STATEMENTS

Conditional Statements

➔ **Conditional statements** let you execute different blocks of code based on conditions.

Common String Operations:

`if` Statement:

```
let age = 20;
if (age >= 18) {
    console.log("You are an adult");
}
```

`if...else` Statement:

```
let age = 16;
if (age >= 18) {
    console.log("You are an adult");
} else {
    console.log("You are a minor");
}
```

`if...else if...else` Statement:

```
let score = 85;
if (score >= 90) {
    console.log("Grade: A");
} else if (score >= 80) {
    console.log("Grade: B");
} else {
    console.log("Grade: C or below");
}
```

Notes:

- Conditions use **comparison operators** (==, !=, >, <, >=, <=)

Nested Conditional Statements

➔ A **nested conditional** means using one `if` or `else` block **inside another**.
It allows for **more complex decision making**.

Nested `if`:

```
let age = 25;
let hasID = true;

if (age >= 18) {
    if (hasID) {
        console.log("You can enter");
    } else {
        console.log("ID required");
    }
} else {
    console.log("You are too young");
}
```

Output:

```
You can enter
```

Practice Problem

Problem1: Write a JavaScript program to check whether a number entered by the user is **odd** or **even**, and print the result.

Code:	Output:

`let num = Number(prompt("Enter a number:"));`
 	
`if (num % 2 === 0) {`
`  console.log("Even");`
`} else {`
`  console.log("Odd");`
`}`
	`Enter a number: 10`
	`Even`
	
	`Enter a number: 5`
	`Odd`

Problem2: Write a JavaScript program to **grade students based on their marks** using the following criteria:

- If marks are **90 or above**, grade = "A"
- If marks are **80 to 89**, grade = "B"
- If marks are **70 to 79**, grade = "C"
- If marks are **below 70**, grade = "D"

Then print the grade.

Code:	Output:

`let marks = Number(prompt("Enter student's marks:"));`
`let grade;`
 	
`if (marks >= 90) {`
`  grade = "A";`
`} else if (marks >= 80) {`
`  grade = "B";`
`} else if (marks >= 70) {`
`  grade = "C";`
`} else {`
`  grade = "D";`
`}`
`console.log("Grade:", grade);`
	`Enter student's marks: 73`
	`Grade: C`

- **Problem3:** Write a JavaScript program to input three numbers from the user and print the **greatest** among them.

Code:	Output:

``let a = Number(prompt("Enter first number:"));
let b = Number(prompt("Enter second number:"));
let c = Number(prompt("Enter third number:"));
 	
let greatest;
 	
if (a >= b && a >= c) {
    greatest = a;
} else if (b >= a && b >= c) {
    greatest = b;
} else {
    greatest = c;
}
 	
console.log("The greatest number is:", greatest);```
	Enter first number: 10
	Enter second number: 20
	Enter third number: 30
	The greatest number is: 30
	
	Enter first number: 15
	Enter second number: 10
	Enter third number: 5
	The greatest number is: 15
	
	Enter first number: 7
	Enter second number: 21
	Enter third number: 14
	The greatest number is: 21

- **Problem6:** Write a JavaScript program to check whether a number is divisible by 2, 5, and 11. Also, **demonstrate the use of both OR and AND operators** in the program.

Code:	Output:

``let num = Number(prompt("Enter a number:"));
 	
if (num % 2 === 0 && num % 5 === 0 && num % 11 === 0) {
    console.log("Number is divisible by 2, 5, and 11.");
} else {
    console.log("Number is NOT divisible by 2, 5, and 11.");
}
 	
if (num % 2 === 0 || num % 5 === 0 || num % 11 === 0) {
    console.log("Number is divisible by at least one of 2, 5, or 11.");
} else {
    console.log("Number is not divisible by 2, 5, or 11.");```
	Enter a number: 110
	number is divisible by 2, 5, and 11.
	number is divisible by at least one of 2, 5, or 11.
	
	Enter a number: 37
	number is NOT divisible by 2, 5, and 11.
	number is not divisible by 2, 5, or 11.

LOOPS

Loops

➔ **Loops** are used to execute a block of code repeatedly.

Types of Loops:

`for` Loop: iterate over a sequence

```
for (let i = 1; i <= 5; i++) {
    console.log(i);
}
```

`while` Loop: runs while a condition is `True`

```
let i = 1;
while (i <= 5) {
    console.log(i);
    i++;
}
```

`do...while` Loop: Runs the code **once** before checking the condition.

```
let i = 1;
do {
    console.log(i);
    i++;
} while (i <= 5);
```

`for...of` Loop: Loops through **iterable objects** like arrays or strings.

```
let colors = ["red", "green", "blue"];
for (let color of colors) {
    console.log(color);
}
```

`for...in` Loop: Loops through **object properties**.

```
let person = {name: "John", age: 30};
for (let key in person) {
    console.log(key + ": " + person[key]);
}
```

`break` Statement: Stops the loop immediately.

```
for (let i = 1; i <= 10; i++) {
  if (i === 5) {
    break;
  }
  console.log(i);
}
```

`continue` Statement: Skips the current iteration and moves to the next.

```
for (let i = 1; i <= 5; i++) {
    if (i === 3){
      continue;
    }
    console.log(i);
}
```

Practice Problem

- **Problem1:** Write a JavaScript program to **print numbers from 1 to 5 using a `while` loop**.

Code:	Output:

``let i = 1;
while (i <= 5) {
    console.log(i);
    i++;
}```
	1
	2
	3
	4
	5

- **Problem2:** Write a JavaScript program to **print numbers from 5 to 1** using a while loop.

Code:	Output:

``let i = 5;
while (i >= 1) {
    console.log(i);
    i--;
}```
	5
	4
	3
	2
	1

- **Problem3:** Write a JavaScript program to input a number n and print its **multiplication table** using a while loop.

Code:	Output:

``let n = parseInt(prompt("Enter a number:"));
 	
let i = 1;
while (i <= 10) {
    console.log(n + " x " + i + " = " + (n * i));
    i++;
}```
	Enter a number: 11
	11 x 1 = 11
	11 x 2 = 22
	11 x 3 = 33
	11 x 4 = 44
	11 x 5 = 55
	11 x 6 = 66
	11 x 7 = 77
	11 x 8 = 88
	11 x 9 = 99
	11 x 10 = 110

- **Problem4:** Write a JavaScript program to **input a number n** and print the **sum of all numbers from 1 to n** using a `for` loop.

Code:	Output:

``
let n = parseInt(prompt("Enter a number:"));
let sum = 0;
 	
for (let i = 1; i <= n; i++) {
    sum += i;
}
console.log("The sum of 1 to " + n + " is: " + sum);
``
	Enter a number: 10
	The sum of 1 to 10 is: 55

- **Problem4:** Write a JavaScript program that takes numbers as input from the user using a `while` loop. The loop should **stop if the user enters** `-1`. Print the **sum** of all entered numbers (excluding `-1`).

Code:	Output:

``
let total = 0;
 	
while (true) {
    let num = parseInt(prompt("Enter a number (-1 to stop):"));
    if (num === -1) {
        break;
    }
    total += num;
}
console.log("Total sum is: " + total);
``
	Enter a number (-1 to stop): 10
	Enter a number (-1 to stop): 5
	Enter a number (-1 to stop): 3
	Enter a number (-1 to stop): 2
	Enter a number (-1 to stop): 1
	Enter a number (-1 to stop): -1
	Total sum is: 21

- **Problem5:** Write a JavaScript program to print all numbers from **1 to 10 in a line**, except for number **5**, using a `for` loop and the `continue` statement.

Code:	Output:

``
for (let i = 1; i <= 10; i++) {
    if (i === 5) {
        continue;
    }
    process.stdout.write(i + " ");
}
``
	1 2 3 4 6 7 8 9 10


STRING

Strings

➔ A **string** is a sequence of characters inside **single quotes (')**, **double quotes ("")**, or **backticks (``)**.

Example Code:

```
let str1 = "Hello";
let str2 = 'World';
let str3 = `Hello World`; // template literal
```

String Creation:

```
let single = 'Hello';
let double = "Hello";
let backtick = `Hello`; // allows variables & expressions
```

Template Literals: Allow variables and expressions inside strings using `${}`.

```
let name = "John";
let age = 25;
console.log(`My name is ${name} and I am ${age} years old.`);
```

Common String Methods:

Method	Description	Example
`toUpperCase()`	Converts to uppercase	`"hello".toUpperCase() → "HELLO"`
`toLowerCase()`	Converts to lowercase	`"HELLO".toLowerCase() → "hello"`
`charAt(index)`	Returns character at index	`"Hello".charAt(1) → "e"`
`indexOf(value)`	Finds position of first match	`"Hello".indexOf("l") → 2`
`lastIndexOf(value)`	Finds last match	`"Hello".lastIndexOf("l") → 3`
`includes(value)`	Checks if contains substring	`"Hello".includes("lo") → true`
`startsWith(value)`	Checks if starts with substring	`"Hello".startsWith("He") → true`
`endsWith(value)`	Checks if ends with substring	`"Hello".endsWith("lo") → true`

`slice(start, end)`	Extracts part of a string	`"Hello".slice(1, 4) → "ell"`
`substring(start, end)`	Similar to slice	`"Hello".substring(1, 4) → "ell"`
`substr(start, length)`	Extracts part by length	`"Hello".substr(1, 3) → "ell"`
`replace(old, new)`	Replaces text	`"Hello".replace("He", "Ye") → "Yello"`
`replaceAll(old, new)`	Replaces all matches	`"hi hi".replaceAll("hi", "hey") → "hey hey"`
`trim()`	Removes spaces from both ends	`" hello ".trim() → "hello"`
`repeat(n)`	Repeats string n times	`"Hi".repeat(3) → "HiHiHi"`
`split(separator)`	Converts to array	`"a,b,c".split(",") →["a", "b", "c"]`

Common String Properties:

Property	Description	Example
`.length`	Returns string length	`"Hello".length // 5`

Escape Characters

Code	Meaning
`\'`	Single quote
`\"`	Double quote
`\\`	Backslash
`\n`	New line
`\t`	Tab

Example code:	Output:
`console.log("Hello\nWorld");`	**Hello** **World**
Example code:	**Output:**

`let a = "Hello";`
`let b = "World";`
`console.log(a + " " + b);`
`console.log(`${a} ${b}`);`
	Hello World
	Hello World

Practice Problem

- **Problem1:** Write a JavaScript to input user's first name & print its length.

Code:	Output:

`let first_name = prompt("Enter first name:");`
`let length = first_name.length;`
`console.log("Length of first name is: " + length);`
	`Enter first name: Mohsin`
	`Length of first name is: 6`

- **Problem2:** Write a JavaScript program to initialize a string and find how many times the character '$' appears in it. text = "He bought a pen for $5 and a book for $10, total $15."

Code:	Output:

`let text = "He bought a pen for $5 and a book for $10, total $15.";`
`let count = 0;`
 	
`for (let i = 0; i < text.length; i++) {`
`  if (text[i] === '$') {`
`  count++;`
`  }`
`}`
 	
`console.log("The character '$' appears " + count + " time(s).");`
	`The character '$' appears 3 time(s).`

- **Problem3:** Prompt the user to enter their full name. Generate a username for them based on the input. Start username with @, followed by their full name and ending with the fullname length. eg: user name = "mohsineli" , username should be "@mohsineli9"

Code:	Output:

`const fullName = prompt("full name:");`
`const username = `@${fullName}${fullName.length}`;`
`console.log(`Your username: ${username}`);`
`alert(`Your username is: ${username}`);`
	`Your username: @mohsineli9`

Arrays

⇨ An **array** is a special variable that can hold **multiple values** in a single name.

- Array elements can be of **different data types**.
- Indexing starts at **0**.

Syntax:	Example:
`let llistVariableName = ["item1", "item2"]`	`let fruits = ["Apple", "Banana", "Mango"];`

Creating Arrays:

```
let arr1 = ["Apple", "Banana", "Mango"]; // literal
let arr2 = new Array("Apple", "Banana"); // constructor
let arr3 = []; // empty array
```

Accessing Array Elements:

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(fruits[0]); // Apple
console.log(fruits[fruits.length - 1]); // Mango
```

Changing Array Elements:

```
fruits[1] = "Orange"; // changes Banana to Orange
```

Common Array Properties:

Property	Description	Example
`.length`	Number of elements	`fruits.length` ⟶ `3`

Example code:	Output:

`let fruits = ["Apple", "Banana", "Mango"];`
`let length = fruits.length;`
 	
`console.log("Number of fruits:", length);`
	`Number of fruits: 3`

Common Array Methods:

Method	Description	Example
`push(item)`	Adds to end	`arr.push("Grapes")`
`pop()`	Removes last item	`arr.pop()`
`unshift(item)`	Adds to start	`arr.unshift("Lemon")`
`shift()`	Removes first item	`arr.shift()`
`concat(arr2)`	Joins arrays	`arr.concat(arr2)`
`join(separator)`	Array → string	`arr.join(", ")`
`slice(start, end)`	Copies part of array	`arr.slice(1, 3)`
`splice(start, deleteCount, ...items)`	Adds/removes items	`arr.splice(1, 1, "Kiwi")`
`indexOf(value)`	First index of value	`arr.indexOf("Apple")`
`lastIndexOf(value)`	Last index of value	`arr.lastIndexOf("Apple")`
`includes(value)`	Checks if exists	`arr.includes("Banana")`
`reverse()`	Reverses array	`arr.reverse()`
`sort()`	Sorts alphabetically	`arr.sort()`
`toString()`	Array → string	`arr.toString()`
`flat(depth)`	Flattens nested arrays	`[1, [2, 3]].flat()`

Multidimensional Arrays

Example code:	Output:

`let matrix = [`
`  [1, 2, 3],`
`  [4, 5, 6]`
`];`
`console.log(matrix[1][2]);`
	6

Looping Through Arrays:

Example code:	Output:

`let fruits = ["Apple", "Banana", "Mango"];`
 	
`// for loop`
`for (let i = 0; i < fruits.length; i++) {`
`  console.log(fruits[i]);`
`}`
	Apple
	Banana
	Mango

Practice Problem

- **Problem1:** For a given array with marks of students → [85, 97, 44, 37, 76, 60] Find the average marks of the entire class.

Code:	Output:

`let marks = [85, 97, 44, 37, 76, 60];`
 	
`let sum = 0;`
`for (let i = 0; i < marks.length; i++) {`
`  sum += marks[i];`
`}`
 	
`let average = sum / marks.length;`
`console.log("Average Marks:", average);`
	`Average Marks: 66.5`

- **Problem2:** For a given array with prices of 5 items → [250, 645, 300, 900, 50] All items have an offer of 10% OFF on them. Change the array to store final price after applying offer.

Code:	Output:

`let prices = [250, 645, 300, 900, 50];`
 	
`for (let i = 0; i < prices.length; i++) {`
`  let discount = prices[i] * 0.10;`
`  prices[i] = prices[i] - discount;`
`}`
 	
`console.log("Final Prices after 10% OFF:", prices);`
	`Final Prices after 10% OFF:`
	`(5) [225, 580.5, 270, 810, 45]`

- **Problem3:** Create an array to store companies → “Bloomberg”, “Microsoft”, “Uber”, “Google”, “IBM”, “Netflix”

  a. Remove the first company from the array

  b. Remove Uber & Add Ola in its place

  c. Add Amazon at the end

Code:	Output:

``
let companies = ["Bloomberg", "Microsoft", "Uber", "Google", "IBM", "Netflix"];
 	
// a. Remove the first company
companies.shift();
 	
// b. Remove Uber & Add Ola in its place
let uberIndex = companies.indexOf("Uber");
if (uberIndex !== -1) {
    companies.splice(uberIndex, 1, "Ola");
}
 	
// c. Add Amazon at the end
companies.push("Amazon");
 	
console.log(companies);
``
	(6) ['Microsoft', 'Ola', 'Google', 'IBM', 'Netflix', 'Amazon']

Functions

➔ A **function** is a block of reusable code that performs a specific task.

Define a Function:	**Call a Function:**

`function function_name(){`
`  console.log("Hello World!");`
`}`
	`function_name();`

Example code:	**Output:**

`function greet(name) {`
`  console.log("Hello " + name);`
`}`
 	
`greet("Mohsin");`
	Hello, Mohsin

Notes: Use `function` to define a function.	

Function Expression

⇨ A function can be stored in a variable.

Example code:	**Output:**

`const greet = function(name) {`
`  console.log("Hello " + name);`
`};`
 	
`greet("Alif");`
	Hello Alif

Notes: Functions help make code **modular** and **reusable**.	

Arrow Functions

Example code:	**Output:**

`const greet = (name) => {`
`  console.log("Hello " + name);`
`};`
 	
`greet("Jam");`
	Hello Jam

Function with Return Value

Example code:	Output:

`function add(a, b) {`
`  return a + b;`
`}`
`let sum = add(5, 3);`
`console.log(sum); // 8`
	8

Default Parameters

Example code:	Output:

`function greet(name = "Guest") {`
`  console.log("Hello " + name);`
`}`
`greet(); // Hello Guest`
`greet("Nihan"); // Hello Nihan`
	Hello Guest
	Hello Nihan

Rest Parameters

- Allows a function to accept **any number of arguments**.

Example code:	Output:

`function sum(...numbers) {`
`  let total = 0;`
`  for (let num of numbers) {`
`  total += num;`
`  }`
`  return total;`
`}`
`console.log(sum(1, 2, 3, 4)); // 10`
	10

Anonymous Functions

- Functions without a name, usually used as arguments.

Example code:	Output:

`setTimeout(function() {`
`  console.log("Hello after 2 seconds");`
`}, 2000);`
	Hello after 2 seconds

IIFE (Immediately Invoked Function Expression)

- A function that runs **immediately** after it's defined.

Example code:	Output:

`(function() {`
`  console.log("IIFE runs immediately");`
`})();`
	IIFE runs immediately

Scope in Functions

- Variables inside a function are **local**.
- Variables outside are **global**.

Example code:	Output:

`let globalVar = 10;`
`function test() {`
`  let localVar = 5;`
`  console.log(globalVar); // 10`
`  console.log(localVar); // 5`
`}`
 	
`test();`
`console.log(localVar); // ❌ Error`
	10
	5

Practice Problem

- **Problem1:** Create a function using the "function" keyword that takes a String as an argument & returns the number of vowels in the string.

Code:	Output:

```
function countVowels(str) {
    let count = 0;
    let vowels = "aeiouAEIOU";

    for (let i = 0; i < str.length; i++) {
        if (vowels.includes(str[i])) {
            count++;
        }
    }

    return count;
}

console.log(countVowels("Hello World"));
```

3

- **Problem2:** Create an arrow function to perform the same task

Code:	Output:

```
const countVowels = (str) => {
    let count = 0;
    let vowels = "aeiouAEIOU";

    for (let i = 0; i < str.length; i++) {
        if (vowels.includes(str[i])) {
            count++;
        }
    }

    return count;
};

console.log(countVowels("Hello World"));
```

3

Problem3: Create an arrow function to perform the same task

Code:	Output:

`let numbers = [2, 4, 6, 8, 10];`
 	
`numbers.forEach(function(num) {`
`  console.log(num * num);`
`});`
	4
	16
	36
	64
	100

Problem4: We are given array of marks of students. Filter out of the marks of students that scored 90+.

a) Take a number n as input from user. Create an array of numbers from 1 to n.
b) Use the reduce method to calculate sum of all numbers in the array.
c) Use the reduce method to calculate product of all numbers in the array.

Code: Filter students with marks 90+	Output:

`var marks = [85, 97, 44, 93, 76, 99, 91];`
`var highScorers = marks.filter(function(mark) {`
`  return mark > 90;`
`});`
`console.log("Marks 90+ :", highScorers);`
	Marks 90+ : (4) [97, 93, 99, 91]

Code: Create array from 1 to n	**Output:**

`var n = parseInt(prompt("Enter a number:"));`
`var arr = [];`
`for (var i = 1; i <= n; i++) {`
`  arr.push(i);`
`}`
`console.log("Array from 1 to n:", arr);`
	Array from 1 to n: (5) [1, 2, 3, 4, 5]

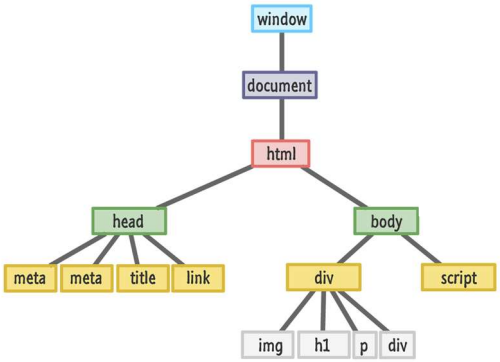
Code: Calculate product using reduce	**Output:**

`var arr = [1,2,3,4,5];`
`var product = arr.reduce(function(accumulator, currentValue) {`
`  return accumulator * currentValue;`
`}, 1);`
`console.log("Product of numbers:", product);`
	Product of numbers: 120

DOM MANIPULATION

DOM Manipulation

- **DOM** (Document Object Model) represents the **HTML structure** of a webpage.
- JavaScript can **access, change, add, or remove HTML elements** using the DOM.



Accessing Elements

Method	Description	Example
`getElementById()`	Select element by `id`	`document.getElementById("myDiv")`
`getElementsByClassName()`	Select elements by `class`	`document.getElementsByClassName("myClass")`
`getElementsByTagName()`	Select elements by tag	`document.getElementsByTagName("p")`
`querySelector()`	Select first element matching CSS selector	`document.querySelector(".myClass")`
`querySelectorAll()`	Select all elements matching CSS selector	`document.querySelectorAll("p")`

Changing Content

Property / Method	Description	Example
`innerHTML`	Change HTML content	`element.innerHTML = "<div>Hello</div>"`
`textContent`	Change text content only	`element.textContent = "Hello"`
`innerText`	Change visible text	`element.innerText = "Hello"`

Changing Styles using DOM

Example code:

```
let box = document.getElementById("Box");
box.style.backgroundColor = "red";
box.style.width = "200px";
box.style.border = "2px solid black";
```

Output:

Before JS	After JS
	

Changing Attributes

Method	Description	Example
`getAttribute()`	Get attribute value	`element.getAttribute("src")`
`setAttribute()`	Set attribute value	`element.setAttribute("src", "image.jpg")`
`removeAttribute()`	Remove an attribute	`element.removeAttribute("disabled")`

Example code:

```
<div id="Box" title="Original Title">Box</div>
<script>
    // Select the element
    let box = document.getElementById("Box");

    // getAttribute()
    console.log("Title before:", box.getAttribute("title"));

    // setAttribute()
    box.setAttribute("title", "Updated Title");
    console.log("Title After:", box.getAttribute("title"));

    // removeAttribute()
    box.removeAttribute("title");
    console.log("Title After remove:", box.getAttribute("title"));
</script>
```

Output:

```
Title before: Original Title
Title After: Updated Title
Title After remove: null
```

Adding / Removing Elements

Example code:

```
    <div id="Box" title="Original Title">Box</div>
    <div id="Box2"></div>
    <script>
        // Create a new element
        let p = document.createElement("p");
        p.textContent = "Hello World";

        let div2 = document.getElementById("Box2");

        // Append to body or another element
        div2.appendChild(p);

        // Remove an element
        let div = document.getElementById("Box");
        div.remove();

    </script>
```

Output:

```
<body>

    <div id="Box2">
        <p>Hello World</p>
    </div>
    <script>…</script>


</body>
```

Event Handling

- Respond to **user actions** like **clicks, mouse movements, keyboard input**.

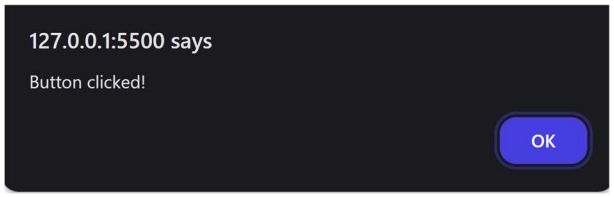
Example code:

```
<button id="myBtn">Click me</button>

<script>
    let button = document.getElementById("myBtn");

     button.addEventListener("click", function () {
        alert("Button clicked!");
    });
</script>
```

Output:



- **Common Events:**

Event Name	Trigger Description
`click`	Mouse click
`mouseover`	Mouse moves over the element
`mouseout`	Mouse leaves the element
`keydown`	A key is pressed on the keyboard
`keyup`	A key is released on the keyboard
`submit`	A form is submitted

Traversing the DOM

Property	Description	Example
`parentNode`	Access parent element	`element.parentNode`
`children`	Access child elements	`element.children`
`firstChild`	First child node	`element.firstChild`
`lastChild`	Last child node	`element.lastChild`
`nextSibling`	Next sibling node	`element.nextSibling`
`previousSibling`	Previous sibling node	`element.previousSibling`

Example code: Traversing the DOM

```
<div id="parent">
    <p id="first">First paragraph</p>
    <p id="second">Second paragraph</p>
    <p id="third">Third paragraph</p>
</div>

<script>
    let secondPara = document.getElementById("second");

    // 1. parentNode
    console.log("Parent Node of second:", secondPara.parentNode);

    // 2. children
    console.log("Children of parent div:", secondPara.parentNode.children);

    // 3. firstChild
    console.log("First Child of parent div:", secondPara.parentNode.firstChild);

    // 4. lastChild
    console.log("Last Child of parent div:", secondPara.parentNode.lastChild);

    // 5. nextSibling
    console.log("Next Sibling of second paragraph:", secondPara.nextSibling);

    // 6. previousSibling
    console.log("Previous Sibling of second paragraph:", secondPara.previousSibling);
</script>
```

Output:

```
parentNode: <div id="parent">…</div>
children: HTMLCollection(3) [<p id="first">…</p>, <p id="second">…</p>, <p id="third">…</p>]
firstChild: <p id="first">First paragraph</p>
lastChild: <p id="third">Third paragraph</p>
nextSibling: <p id="third">Third paragraph</p>
previousSibling: <p id="first">First paragraph</p>
```

Example code: Event Handling

```
<button id="btnClick">Click Me</button><br><br>

<div id="hoverBox">Hover Here</div><br>

<input id="keyInput" type="text" placeholder="Type something..."><br><br>

<form id="myForm">
    <input type="text" placeholder="Your name">
    <button type="submit">Submit</button>
</form>

<script>
    // Elements
    const btnClick = document.getElementById("btnClick");
    const hoverBox = document.getElementById("hoverBox");
    const keyInput = document.getElementById("keyInput");
    const myForm = document.getElementById("myForm");

    // Actions
    const handleClick = () => {
        console.log("Button clicked!");
    };

    const handleMouseOver = () => {
        console.log("Mouse over the box!");
        hoverBox.style.backgroundColor = "lightgreen";
    };

    const handleMouseOut = () => {
        console.log("Mouse out of the box!");
        hoverBox.style.backgroundColor = "lightblue";
    };

    const handleKeyDown = (event) => {
        console.log("Key down:", event.key);
    };

    const handleKeyUp = (event) => {
        console.log("Key up:", event.key);
    };
```

```
    const handleSubmit = (event) => {
        event.preventDefault();
        console.log("Form submitted!");
    };

    // Event Listeners
    btnClick.addEventListener("click", handleClick);
    hoverBox.addEventListener("mouseover", handleMouseOver);
    hoverBox.addEventListener("mouseout", handleMouseOut);
    keyInput.addEventListener("keydown", handleKeyDown);
    keyInput.addEventListener("keyup", handleKeyUp);
    myForm.addEventListener("submit", handleSubmit);
</script>
```

Output:

Button clicked!

Mouse over the box!

Mouse out of the box!

Key down: b

Key up: b

Form submitted!

Example code: Changing Content

```
<div id="demo1">Original <b>HTML</b> content</div>
<div id="demo2">Original <b>HTML</b> content</div>
<div id="demo3">Original <b>HTML</b> content</div>

<button id="btnHTML">Change innerHTML</button>
<button id="btnTextContent">Change textContent</button>
<button id="btnInnerText">Change innerText</button>

<script>
  const demo1 = document.getElementById("demo1");
  const demo2 = document.getElementById("demo2");
  const demo3 = document.getElementById("demo3");

  const btnHTML = document.getElementById("btnHTML");
  const btnTextContent = document.getElementById("btnTextContent");
  const btnInnerText = document.getElementById("btnInnerText");

  // innerHTML changes HTML structure
  btnHTML.onclick = () => {
    demo1.innerHTML = "<i>Hello</i> <b>World</b>";
  };

  // textContent changes only text (HTML tags shown as plain text)
  btnTextContent.onclick = () => {
    demo2.textContent = "<i>Hello</i> <b>World</b>";
  };

  // innerText changes visible text (similar to textContent but respects CSS styles)
  btnInnerText.onclick = () => {
    demo3.innerText = "<i>Hello</i> <b>World</b>";
  };
</script>
```

Output: Before

Original **HTML** content
Original **HTML** content
Original **HTML** content
Change innerHTML | Change textContent | Change innerText

Output: After

Hello **World**
Hello World
Hello World
Change innerHTML | Change textContent | Change innerText

Example code: Accessing Elements

```
<div id="myDiv">This is a div with id "myDiv"</div>

<p class="myClass">Paragraph 1 with class "myClass"</p>
<p class="myClass">Paragraph 2 with class "myClass"</p>
<p>Paragraph 3 with no class</p>

<button id="btn">Select Elements</button>

<script>
  const btn = document.getElementById("btn");

  btn.onclick = () => {
    // 1. getElementById
    const divById = document.getElementById("myDiv");
    console.log("getElementById:", divById);

    // 2. getElementsByClassName
    const elemsByClass = document.getElementsByClassName("myClass");
    console.log("getElementsByClassName:", elemsByClass);

    // 3. getElementsByTagName
    const elemsByTag = document.getElementsByTagName("p");
    console.log("getElementsByTagName:", elemsByTag);

    // 4. querySelector (first match)
    const firstByQuery = document.querySelector(".myClass");
    console.log("querySelectorClass:", firstByQuery);

    // 5. querySelector by id
    const idByQuery = document.querySelector("#myDiv");
    console.log("querySelectorId:", idByQuery);

    // 6. querySelectorAll (all matches)
    const allByQuery = document.querySelectorAll("p");
    console.log("querySelectorAll:", allByQuery);
  };
</script>
```

Output Tracing Practice

Problem1.

Write the output of the following code	Answer

<!DOCTYPE html>
<html lang="en">
<body>
<script>
let x = "1";
const y = 2;
let z = [1, 2, 3];
 	
console.log("1" + x - y);
console.log(parseInt(x));
let price = 9.99;
console.log(price.toString() + x);
 	
z[z.length - 3] = Number(x) ** y;
console.log(z[1]);
console.log(typeof z);
 	
document.write(`${typeof x}`);
</script>
</body>
</html>
	Write the console output
	
	9
	1
	9.991
	2
	object
	
	Write the browser (document) output
	
	string

Problem2.

Write the output of the following code	Answer

<!DOCTYPE html>
<html lang="en">
<body>
<script>
let a = 5;
let b = "10";
 	
console.log(a + b);
console.log(Number(b) - a);
 	
let arr = [a, b];
console.log(arr.length);
arr[2] = a * 2;
console.log(arr[2]);
 	
document.write(typeof b);
</script>
</body>
</html>
	Write the console output
	
	510
	5
	2
	10
	
	Write the browser (document) output
	
	string

Problem3.

Write the output of the following code	Answer

<!DOCTYPE html>
<html lang="en">
<body>
<script>
const str = "20";
let num = 4;
 	
console.log(str / num);
console.log(str + num);
console.log(parseInt(str) + num);
 	
let data = ["x", "y", "z"];
data[data.length - 1] = "last";
console.log(data);
 	
document.write(typeof data);
</script>
</body>
</html>
	Write the console output
	
	5
	204
	24
	["x", "y", "last"]
	
	Write the browser (document) output
	
	object

Problem4.

Write the output of the following code	Answer

<!DOCTYPE html>
<html lang="en">
<body>
<script>
let price = 15.5;
let quantity = "2";
 	
console.log(price * quantity);
console.log(price + quantity);
console.log(Number(quantity) ** 2);
 	
let items = [1, 2, 3];
items[1] = price;
console.log(items[1]);
 	
document.write(typeof price);
</script>
</body>
</html>
	Write the console output
	
	31
	15.52
	4
	15.5
	
	Write the browser (document) output
	
	number

Problem5.

Write the output of the following code	Answer

<!DOCTYPE html>
<html lang="en">
<body>
<script>
let x = "5";
let y = 3;
 	
console.log(x - y);
console.log(x + y);
console.log(parseFloat(x) + y);
 	
let nums = [10, 20];
nums[nums.length] = 30;
console.log(nums.length);
 	
document.write(typeof y);
</script>
</body>
</html>
	Write the console output
	
	2
	53
	8
	3
	
	Write the browser (document) output
	
	number

Problem6.

Write the output of the following code	Answer

<!DOCTYPE html>
<html lang="en">
<body>
<script>
let a = "10";
let b = 5;
 	
console.log(a - b);
console.log(a + b);
console.log(a * "2");
console.log("5" + 3 - 2);
 	
let arr = [1, 2, 3];
arr[arr.length] = arr.length - 1;
console.log(arr[arr[3]]);
 	
document.write(typeof (a + b - "2"));
</script>
</body>
</html>
	Write the console output
	
	5
	105
	20
	51
	3
	
	Write the browser (document) output
	
	number

Problem7.

Write the output of the following code	Answer

<!DOCTYPE html>
<html lang="en">
<body>
<script>
let x;
let y = null;
let z = "5";
 	
console.log(x + 1);
console.log(y + 1);
console.log(z - 2);
console.log(z + 2);
 	
let data = [, 10, 20];
console.log(data[0]);
document.write(x + "test");
data[0] = data[2] / data[1];
console.log(data[0]);
 	
document.write(typeof (x + "test"));
</script>
</body>
</html>
	Write the console output
	NaN
	1
	3
	52
	undefined
	2
	Write the browser (document) output
	undefinedteststring

Problem8.

Write the output of the following code	Answer

<!DOCTYPE html>
<html lang="en">
<body>
<script>
let a = "2";
let b = 2;
 	
console.log(a == b);
console.log(a === b);
console.log(a + b + "2");
console.log("2" + a * b);
console.log((a + b) * "2");
 	
let arr = [1, 2, 3];
arr.length = 5;
arr[4] = arr.length;
console.log(arr[arr[4] / 2]);
console.log(null + "2" - 1);
 	
document.write(typeof (null + "2" - 1));
</script>
</body>
</html>
	Write the console output
	true
	false
	222
	24
	44
	Undefined
	NaN
	Write the browser (document) output
	number

Problem9.

Consider the following HTML and JavaScript code:	Output of the browser is given:

<!DOCTYPE html>
<html>
<body>
 	
<button id="btn1">btn 1</button>
<button id="btn2">btn 2</button>
 	
<script>
const container = document.getElementById("container");
const btn1 = document.getElementById("btn1");
const btn2 = document.getElementById("btn2");
 	
btn1.addEventListener("click", () => {
const newPara = document.createElement("p");
newPara.textContent = "web";
container.appendChild(newPara);
});
 	
btn2.addEventListener("click", () => {
const paragraphs = container.querySelectorAll("p");
if (paragraphs.length > 0) {
const lastPara = paragraphs[paragraphs.length - 1];
container.removeChild(lastPara);
}
});
</script>
 	
</body>
</html>
	

What will be the output of **empty spaces under the buttons** after user perform following actions?	Write the output in the following sections
1) btn 1 is pressed 1 time	`web`
2) btn 2 is pressed 1 time and btn 1 is pressed 2 times	`Web` `web`
3) btn 2 is pressed 1 time	`web`
4) btn 1 is pressed 1 time and btn 2 is pressed 1 time	`web`
5) btn 1 is pressed 1 time	`web` `web`

Problem10.

Consider the following HTML and JavaScript code:	Output of the browser is given:

<!DOCTYPE html>
<html>
<body>
 	
<button id="add">Add</button>
<button id="remove">Remove</button>
 	
<script>
const container = document.getElementById("container");
const addBtn = document.getElementById("add");
const removeBtn = document.getElementById("remove");
 	
addBtn.addEventListener("click", () => {
const p = document.createElement("p");
p.textContent = "JS";
container.appendChild(p);
});
 	
removeBtn.addEventListener("click", () => {
const paras = container.querySelectorAll("p");
if (paras.length > 0) {
container.removeChild(paras[0]);
}
});
</script>
 	
</body>
</html>
	

What will be the output of **empty spaces under the buttons** after user perform following actions?	Write the output in the following sections
1) Add pressed 2 times	JS JS
2) Add pressed 1 times, Remove pressed 1 time	JS JS
3) Add pressed 1 time, Remove pressed 2 times	JS
4) Remove pressed 1 time, Add pressed 2 times	JS JS
5) Remove pressed 1 times	JS

Problem11.

Consider the following HTML and JavaScript code:	Output of the browser is given:

<!DOCTYPE html>
<html>
<body>
 	
<button id="btn1">Add HTML</button>
<button id="btn2">Clear</button>
 	
<script>
const container = document.getElementById("container");
const btn1 = document.getElementById("btn1");
const btn2 = document.getElementById("btn2");
 	
btn1.addEventListener("click", () => {
const p = document.createElement("p");
p.textContent = "HTML";
container.appendChild(p);
});
 	
btn2.addEventListener("click", () => {
const paras = container.querySelectorAll("p");
paras.forEach(p => container.removeChild(p));
});
</script>
 	
</body>
</html>
	

What will be the output of **empty spaces under the buttons** after user perform following actions?	Write the output in the following sections
1) Add HTML pressed 1 time	HTML
2) Add HTML pressed 3 times, Clear pressed 1 time	
3) Clear pressed 1 time, Add HTML pressed 2 time	HTML HTML
4) Add HTML pressed 2 times, Clear pressed 2 times	
5) Add HTML pressed 2 times, Clear pressed 1 times	

Problem12.

Consider the following HTML and JavaScript code:	Output of the browser is given:

<!DOCTYPE html>
<html>
<body>
 	
<button id="btn1">Add</button>
<button id="btn2">Remove Last</button>
 	
<script>
const container = document.getElementById("container");
const btn1 = document.getElementById("btn1");
const btn2 = document.getElementById("btn2");
 	
btn1.addEventListener("click", () => {
const p = document.createElement("p");
p.textContent = "Node";
container.appendChild(p);
});
 	
btn2.addEventListener("click", () => {
const paras = container.querySelectorAll("p");
if (paras.length > 0) {
const last = paras[paras.length - 1];
container.removeChild(last);
}
});
</script>
 	
</body>
</html>
	

What will be the output of **empty spaces under the buttons** after user perform following actions?	Write the output in the following sections
1) Add pressed 1 time	Node
2) Add pressed 2 times, Remove pressed 1 time	Node Node
3) Remove pressed 1 time, Add pressed 1 time	Node Node
4) Add pressed 3 times, Remove pressed 2 times	Node Node Node
5) Remove pressed 3 times	

Code Writing Practice

Problem1: You have given a task to create a product inventory management from for an e-commerce system that requires robust client-side validations. The form should allow staff to add new products. Write code for HTML form **(product.html)** with an external JavaScript file **(validations.js)** based on following guideline.

The form must collect and validate product details. 1) **Product Name** – Text Input, 2) **Price** – Number Input, 3) **Quantity in Stock** – Number Input, 4) **Category** – Dropdown Select,
5) **Expiry Date** – Date Input.

Your JavaScript validation should provide real-time feedback and display error messages using innerHTML based on following validation rule.

1. All fields are required
2. The price must be more then 0
3. Quantity in Stock must a number
4. Date must follow DD-MM-YYYY format
5. In Category, one options must be selected.

Make sure the form cannot be submitted without valid data

product.html

```
<!DOCTYPE html>
<body>
    <h2>Add New Product</h2>
    <form id="productForm">

        <label>Product Name:</label>
        <input type="text" id="productName">
        <span class="error"></span><br>

        <label>Price:</label>
        <input type="number" id="price">
        <span class="error"></span><br>

        <label>Quantity in Stock:</label>
        <input type="number" id="quantity">
        <span class="error"></span><br>

        <label>Category:</label>
        <select id="category">
            <option value="">--Select Category--</option>
            <option value="electronics">Electronics</option>
            <option value="clothing">Clothing</option>
        </select>
        <span class="error"></span><br>

        <label>Expiry Date:</label>
        <input type="date" id="expiryDate">
        <span class="error"></span><br>

        <br><br>
        <button type="submit">Add Product</button>
    </form>

    <script src="validations.js"></script>
</body>
</html>
```

validations.js

```
const form = document.getElementById("productForm");

const pName = document.getElementById("pName");
const price = document.getElementById("price");
const quantity = document.getElementById("quantity");
const category = document.getElementById("category");
const expiryDate = document.getElementById("expiryDate");

let error = document.querySelectorAll(".error");

function validateName() {
    if (pName.value.trim() === "") {
        error[0].innerHTML = "Product Name is required.";
        return false;
    }
    error[0].innerHTML = "";
    return true;
}

function validatePrice() {
    if (price.value === "") {
        error[1].innerHTML = "Product Price is required.";
        return false;
    } else if (Number(price.value) <= 0) {
        error[1].innerHTML = "Price must be more than 0.";
        return false;
    }
    error[1].innerHTML = "";
    return true;
}

function validateQuantity() {
    if (quantity.value === "") {
        error[2].innerHTML = "Quantity is required.";
        return false;
    } else if (isNaN(Number(quantity.value))) {
        error[2].innerHTML = "Quantity must be a number.";
        return false;
    }
    error[2].innerHTML = "";
    return true;
}

function validateCategory() {
    if (category.value === "") {
        error[3].innerHTML = "Please select a category.";
        return false;
    }
    error[3].innerHTML = "";
    return true;
}

function validateDate() {
    if (expiryDate.value === "") {
        error[4].innerHTML = "Expiry Date is required.";
        return false;
    }
    error[4].innerHTML = "";
    return true;
}

form.addEventListener("submit", function (event) {
    if (
        !validateName() ||
        !validatePrice() ||
        !validateQuantity() ||
        !validateCategory() ||
        !validateDate()
    ) {
        event.preventDefault();
    }
});
```

- **Problem2:** You are tasked with creating a **user registration form** for a website. Write code for HTML form **(registration.html)** with an external JavaScript file **(validations.js)** based on following guideline.

  The form must collect and validate details. 1) **Full Name** – Text Input, 2) **Email Address** – Email Input, 3) **Password** – Password input, 4) **Confirm Password** – Password input, 5) **Gender** – Radio buttons, 6) **Date of Birth** – Date input

  Your JavaScript validation should provide real-time feedback and display error messages using innerHTML based on following validation rule.

  1. All fields are **required**
  2. Email must be in valid email format
  3. Password must be at least **8 characters**
  4. Confirm Password must **match** the Password field
  5. One **Gender** option must be selected
  6. Date of Birth cannot be empty

  Make sure the form cannot be submitted without valid data

registration.html

```
<!DOCTYPE html>
<body>
    <h2>User Registration</h2>
    <form id="regForm">

        <label>Full Name:</label>
        <input type="text" class="inputs">
        <span class="error"></span><br>

        <label>Email:</label>
        <input type="email" class="inputs">
        <span class="error"></span><br>

        <label>Password:</label>
        <input type="password" class="inputs">
        <span class="error"></span><br>

        <label>Confirm Password:</label>
        <input type="password" class="inputs">
        <span class="error"></span><br>

        <label>Gender:</label>
        <input type="radio" value="Male" class="inputs">
        <input type="radio" value="Female" class="inputs">
        <span class="error"></span><br>

        <label>DOB:</label>
        <input type="date" class="inputs">
        <span class="error"></span><br>

        <button type="submit">Register</button>
    </form>
    <script src="validations.js"></script>
</body>
</html>
```

validations.js

```
const form = document.getElementById("regForm");
const inputs = document.querySelectorAll(".inputs");
let error = document.querySelectorAll(".error");

form.addEventListener("submit", function (event) {
    let ok = true;

    if (inputs[0].value === "") {
        error[0].innerHTML = "Name is required.";
        ok = false;
    } else {
        error[0].innerHTML = "";
    }

    if (inputs[1].value === "") {
        error[1].innerHTML = "Email required.";
        ok = false;
    } else {
        error[1].innerHTML = "";
    }

    if (inputs[2].value === "") {
        error[2].innerHTML = "Password is required.";
        ok = false;
    } else if (inputs[2].value.length < 8) {
        error[2].innerHTML = "Min 8 characters.";
        ok = false;
    } else {
        error[2].innerHTML = "";
    }

    if (inputs[3].value === "") {
        error[3].innerHTML = "Confirm Password is required.";
        ok = false;
    } else if (inputs[2].value !== inputs[3].value) {
        error[3].innerHTML = "Passwords do not match";
        ok = false;
    } else {
        error[3].innerHTML = "";
    }

    if (!inputs[4].checked && !inputs[5].checked) {
        error[4].innerHTML = "Please select a gender.";
        ok = false;
    } else {
        error[4].innerHTML = "";
    }

    if (inputs[6].value === "") {
        error[5].innerHTML = "DOB is required.";
        ok = false;
    } else {
        error[5].innerHTML = "";
    }

    if (!ok) {
        event.preventDefault();
    } else {
        alert("Reg Success");
    }});
```

Problem3: You are asked to create an **event booking form** for a conference website. Write code for HTML form **(booking.html)** with an external JavaScript file **(validations.js)** based on following guideline.

The form must collect and validate details. 1) **Full Name** – Text Input, 2) **Email Address** – Email Input, 3) **Number of Tickets** – Number input, 4) **Event Type** – Dropdown (Workshop, Seminar, Networking), 5) **Event Date** – Date input, 6) **Agree to Terms** – Checkbox.

Your JavaScript validation should provide real-time feedback and display error messages using innerHTML based on following validation rule.

1. All fields are **mandatory**
2. Email must be valid
3. Number of Tickets must be a **positive number**
4. Event Type must have **one option selected**
5. Event Date cannot be empty
6. Checkbox **must be checked** to allow submission

Make sure the form cannot be submitted without valid data

booking.html

```
<!DOCTYPE html>
<body>
    <h2>Event Booking Form</h2>
    <form id="regForm">

        <label>Full Name:</label>
        <input type="text" class="inputs">
        <span class="error"></span><br>

        <label>Email:</label>
        <input type="email" class="inputs">
        <span class="error"></span><br>

        <label>Number of Tickets:</label>
        <input type="number" class="inputs">
        <span class="error"></span><br>

        <label>Event Type:</label>
        <select class="inputs">
            <option value="Workshop">Workshop</option>
            <option value="Seminar">Seminar</option>
            <option value="Networking">Networking</option>
        </select>
        <span class="error"></span><br>

        <label>DOB:</label>
        <input type="date" class="inputs">
        <span class="error"></span><br>

        <label>Agree Terms & Condition:</label>
        <input type="checkbox" class="inputs">
        <span class="error"></span><br>

        <br><br>
        <button type="submit"> Book Now</button>
    </form>

    <script src="validations.js"></script>
</body>
</html>
```

validations.js

```
const form = document.getElementById("regForm");
const inputs = document.querySelectorAll(".inputs");
let error = document.querySelectorAll(".error");

form.addEventListener("submit", function (event) {
    let ok = true;

    if (inputs[0].value === "") {
        error[0].innerHTML = "Name is required.";
        ok = false;
    } else {
        error[0].innerHTML = "";
    }

    if (inputs[1].value === "") {
        error[1].innerHTML = "Email required.";
        ok = false;
    } else {
        error[1].innerHTML = "";
    }

    const ticketNum = Number(inputs[2].value);
    if (inputs[2].value === "") {
        error[2].innerHTML = "Tickets is required.";
        ok = false;
    } else if (isNaN(ticketNum) || ticketNum <= 0) {
        error[2].innerHTML = "Must be a positive number.";
        ok = false;
    } else {
        error[2].innerHTML = "";
    }

    if (inputs[3].value === "") {
        error[3].innerHTML = "Type required.";
        ok = false;
    } else {
        error[3].innerHTML = "";
    }

    if (inputs[4].value === "") {
        error[4].innerHTML = "DOB is required.";
        ok = false;
    } else {
        error[4].innerHTML = "";
    }

    if (!inputs[5].checked) {
        error[5].innerHTML = "Must agree to terms.";
        ok = false;
    } else {
        error[5].innerHTML = "";
    }

    if (!ok) {
        event.preventDefault();
    } else {
        alert("Booking successful");
    }
});
```

MCQ Practice

1. **Which HTTP request method is used to submit data to be processed to a server?**
   a) Get
   b) POST
   c) PUT
   d) DELETE
2. **What is XHTML?**
   a) A Stricker version of HTML
   b) An extension of HTML
   c) A document type delimitation
   d) An image formats
3. **Which of the following is not a feature of HTTP?**
   a) Connection oriented
   b) Stateless
   c) Request response protocol
   d) Connectionless
4. **The DOM is used to _____**
   a) Structure HTML documents only
   b) Dynamically access and update the content and style of a document
   c) Replace XML tags
   d) Encode URLs
5. **What does DHTML stand for?**
   a) Dynamic Hyper Text Markup Language
   b) Dynamic High Text Markup Language
   c) Direct Hyper Text Markup Language
   d) Document Hyper Text Markup Language
6. **Which protocol is primarily used for secure communications over the internet?**
   a) HTTP
   b) FTP
   c) HTTPS
   d) SMTP
7. **Which is the following XML naming rules**
   a) Names can start with a number
   b) Names can start with a punctuation character
   c) Names cannot contain space
   d) Names cannot use any alphabetic character
8. **In HTML which attribute is used to specify an image source?**
   a) link
   b) src
   c) href
   d) ref

9. **Which of the HTML elements is used to create a checkbox in forms?**
   a) <button type = 'checkbox'>
   b) <input type = 'button>
   c) <input type = 'checkbox'>
   d) <button type = 'check'>
10. **What does the get HTTP request method do?**
   a) Delete data from the server
   b) Retrieves data from the server
   c) Sends data to be processed to a server
   d) Updates existing server data
11. **Which organization is responsible for stabilizing web standards?**
   a) IETF
   b) W3C
   c) HTML5 Consortium
   d) ISO
12. **Network of networks that consists of millions of private, public, academic, business, and government networks is called web.**
   a) True b) false
13. **Which tag is used to create a hyperlink in HTML?**
   a) <a> b) <link> c) <url> d) <href>
14. **Which element in HTML is used to create an unordered list?**
   a) b) c) d) <list>
15. **In XHTML, all elements must be_____**
   a) uppercase b) lowercase c) bold d) italic
16. **The Document Object Model (DOM) represents HTML and XML documents as what type of structure?**
   a) Table b) Matrix c) Tree d) List
17. **What does XML stand for?**
   a) Extensive Markup Language
   b) Extra Markup Language
   c) Extensible Markup Language
   d) Extended Markup Language
18. **What is the main purpose of the HTTP protocol?**
   a) To transfer files between servers
   b) To authenticate servers
   c) To exchange hypertext information between client and server
   d) To format HTML documents
19. **Which HTML input type is used for a password field?**
   a) `<input>`
   b) `<input type="number">`
   c) `<input type="password">`
   d) `<input type="hidden">`

20. **In a URL, what does the 'path' specify?**
    a) The domain name
    b) The protocol
    c) The file location on the server
    d) The fragment identifier
21. **What feature makes DHTML different from HTML?**
    a) Static content
    b) Dynamic interactivity
    c) Hypertext links
    d) Structured formatting
22. **Which HTML element is used to represent a numbered list?**
    a) `<list>` b) `<ol>` c) `<ul>` d) `<nulls>`
23. **Which tag is used for inserting a line break in HTML?**
    a) `<lb>` b) `<line>` c) `<br>` d) `<newline>`
24. **What is the purpose of the `<form>` element in HTML?**
    a) Display images
    b) Define a form to collect user input
    c) Structure the layout
    d) Insert links
25. **In a client-server model, who initiates the request?**
    a) Server b) Client c) Peer d) Router
26. **URI stands for:**
    a) Uniform Resource Identifier
    b) Universal Resource Identifier
    c) Uniform Resource Interlink
    d) Unified Resource Integration
27. **The HTTP protocol is designed to permit intermediate network elements to improve or enable communications between clients and servers and it is a stateless protocol.**
    a) True
    b) False
28. **What is the correct HTML for creating a hyperlink?**
    a) `<a name="http://www.aiub.edu">aiub.edu</a>`
    b) `<a>http://www.aiub.edu</a>`
    c) `<a href="http://www.aiub.edu">AIUB</a>`
    d) `<a url="http://www.aiub.edu">AIUB</a>`
29. **Which HTTP request method is used to retrieve data from a server?**
    a) GET
    b) POST
    c) PUT
    d) DELETE

30. **The <head> sections of an HTML document is used to_____**
    a) Display content
    b) Define metadata
    c) Structure page layout
    d) Define DOM
31. **In HTML5, the <canvas> element is used for _____**
    a) Creating table
    b) Drawing graphics
    c) Embedding videos
    d) Defining forms
32. **Which attribute is necessary for file upload in HTML forms?**
    a) method="post"
    b) enctype="multipart/form-data"
    c) action="file_upload"
    d) name="file"
33. **Which of these elements are all table elements?**
    a) <table><head><tfoot>
    b) <table><tr><td>
    c) <table>
    d) <table><head><body>
34. **Which of the following is used to include the contents of one file into another?**
    a) include()
    b) header()
    c) append()
    d) load()
35. **In the URL: "https://www.example.com/page.html?id=123", what does "id=123" represent?**
    a) Anchor
    b) File path
    c) Query Parameter
    d) Domain Name
36. **Which one of the following statements is false about form handling?**
    a) Any amount of data can be supplied from a form using the POST method
    b) The HTTP GET method should never be used for sending passwords or other sensitive information
    c) The HTTP POST requests are never cached, and we cannot bookmark them
    d) The POST method is limited to 1024 bytes
37. **Which of the following is used to return the length of an array in JS?**
    a) .size
    b) .count
    c) .len
    d) .length

38. **Which of the following is true about the `id` attribute in HTML?**
    a) An element can have multiple `id` values
    b) The `id` attribute must be unique within a single HTML document
    c) The `id` attribute is only used for form elements
    d) The `id` attribute is optional and cannot be used for styling
39. **A router is configured to allow multiple devices in a home network to share a single public IP address. Which technology is being used here?**
    a) HTTP
    b) DNS (Domain Name System)
    c) NAT (Network Address Translation)
    d) TCP/IP
40. **What is the purpose of the `<head>` section in an HTML document?**
    a) To define the visible content of the body
    b) To store metadata and links to stylesheets or scripts
    c) To contain user input forms
    d) To display the page title on the body
41. **You are creating a webpage for a travel blog. On the blog, there are links to external websites like booking.com and tripadvisor.com. You want these external links to open in a new tab so that users don't lose their place on your blog. Which target attribute value should you use in the `<a>` tag to achieve this behavior?**
    a) `_blank` b) `parent` c) `self` d) `top`
42. **What is an attribute in HTML?**
    a) A function inside an HTML tag
    b) A property that provides additional information about an element
    c) A type of CSS style
    d) A separate HTML tag
43. **JavaScript was invented by**
    a) Brendon Eich, 1995
    b) Tim Berners-Lee, 1990
    c) Yukihiro Matsumoto, 1990
    d) Bjarne Stroustrup, 1979
44. **What is the primary role of a web HTTP server?**
    a) To act as a database server
    b) To serve static and dynamic web pages
    c) To manage version control for the project
    d) To provide a graphical user interface for the server
45. **https://example.com:7000/products?id=123. What is the port used in this URI?**
    a) https
    b) 8080
    c) 7000
    d) 80

46. **Which attribute is used to merge two or more columns in an HTML table?**
    a) rowspan
    b) colspan
    c) merge
    d) width

47. **You are styling a box, and you need the content to be spaced from the border by 15px on all sides. Which CSS property will you use?**
    a) margin
    b) padding
    c) border
    d) width

48. **In a form, you want to target only the input fields that are of type "text". Which selector will be the best choice?**
    a) input[type="text"]
    b) input.text
    c) input[type=text]
    d) input:checkbox

49. **You are designing a webpage and want to create space between two elements, so they don't appear too close to each other. However, you want to avoid changing the size or internal layout of either element. Which CSS property should you use to add space between the elements without affecting their internal layout or size?**
    a) padding b) margin c) border d) width

50. **To parse a string and return the first integer, we use the parseInt() method in JavaScript. The output of parseInt("20 min") will be:**
    a) 20
    b) NaN
    c) Undefined
    d) Type error

51. **Suppose you are required to create variables whose scope are defined within a block of code. JavaScript allows you to create block scopes using:**
    a) let
    b) const
    c) var
    d) Both let and const

52. **A developer is building a social media application. Which HTTP method should be used to create a new post?**
    a) GET b) POST c) PUT d) DELETE

53. **Which characteristic of HTTP makes possible to ensure that the server does not store any information about the client's previous requests. Which characteristic of HTTP makes this possible?**
    a) Connectionless
    b) Stateless
    c) Secure
    d) Secureless
54. **What is the output of const arr = [1,2]; arr.push(3)**
    a) [1,2]
    b) [3]
    c) [1,2,3]
    d) [0,1,2,3]
55. **To store and hold more than one item of data at the same time, an array could be a perfect arrangement. JavaScript requires the following syntax to declare an array:**
    a) var arrayname = new Array();
    b) var arrayname = new array();
    c) var myArray = [size];
    d) var myArray = new array[size];
56. **What is the output of the snippet below**
    var fruits = ["Apple", "Orange", "Apple", "Mango"]; console.log(typeof(fruits));
    a) Array b) String c) Number d) Object
57. **Which one is correct to convert the date to string using this standard, var d = new Date();**
    a) Date.now();
    b) d.toISOString();
    c) d.getTime();
    d) d.parseString();
58. **How do you select all elements with class "item" as a static NodeList?**
    a) document.querySelectorAll(".item")
    b) document.getElementsByClassName("item")
    c) document.getItemsByClass("item")
    d) document.find(".item")
59. **You're tasked with styling a webpage that should have a consistent look across multiple pages. Which CSS method would you choose to ensure styles are applied universally across the site?**
    a) Inline CSS
    b) Internal CSS
    c) External CSS
    d) Embedded CSS

60. **What is the correct syntax to add a click event to a button with ID "myBtn"?**
    a) myBtn.addEventListener("click", myFunction);
    b) document.getElementById("myBtn").onClick(myFunction);
    c) document.querySelector("myBtn").addEvent("click", myFunction);
    d) myBtn.click("myFunction");
61. **Which selector would you use to apply styles to an element with the id "main-header"?**
    a) #main-header {}
    b) .main-header {}
    c) main-header {}
    d) *main-header {}
62. **What is output of this code:**

```
let x = 5;
console.log(x);
```

    a) 5
    b) undefined
    c) ReferenceError
    d) null
63. **You want to ensure that an element takes up the entire width of its container and starts on a new line. What display property will you use?**
    a) inline-block
    b) inline
    c) block
    d) flex
64. **What is output of the following code:**

```
const arr = [1, 2];
arr[100] = 3;
console.log(arr.length);
```

    a) 2
    b) 3
    c) 100
    d) 101
65. **What is the correct way to display an image in HTML7?**
    a) <img src="pic.jpg">
    b)
    c) <picture src="pic.jpg">
    d) <pic src="pic.jpg">

66. **What is the purpose of the id attribute in HTML?**
    a) To define a heading
    b) To style an element using CSS
    c) To uniquely identify an element
    d) To create a hyperlink
67. **Which of the following is true about XML?**
    a) It is used only for web design
    b) It is used to style an element
    c) It is a programming language
    d) It is used to structure data
68. **What does DHTML stand for?**
    a) Dynamic Hyper Text Markup Language
    b) Direct Hyper Text Markup Language
    c) Dynamic High Text Markup Language
    d) Document Hyper Text Markup Language
69. **Which type of HTML element is used to create a checkbox in forms?**
    a) <input type="checkbox">
    b) <button type="check">
    c) <input type="button">
    d) <check>
70. **URI stands for:**
    a) Universal Resource Identify
    b) Uniform Resource Identifier
    c) Unified Resource Identify
    d) Universe Resource Identifier
71. **The HTTP protocol is designed to permit intermediate network elements to improve or enable communications between clients and servers and it is a stateless protocol.**
    a) True
    b) False
72. **What does Number("123abc") return?**
    a) 123 b) 0 c) NaN d) null
73. **Which of these elements are all <table> elements?**
    a) <table><tr><tt>
    b) <table><tr><td>
    c) <table><head><tfoot>
    d) <thead><body><tr>

74. **How do you access an element by its ID in JavaScript?**
    a) document.getElementId()
    b) document.getElementId()
    c) document.getElementById()
    d) document.ElementById()
75. **Which function is used to validate numeric input values in JavaScript?**
    a) isNaN()
    b) isNumber()
    c) validateNumber()
    d) checkNum()
76. **Which event is most likely to trigger form validation just before the form is submitted, ensuring all fields are correct in JavaScript?**
    a) onkeypress
    b) onaction
    c) onclick
    d) onsubmit
77. **What is the default value of an undeclared variable?**
    a) null
    b) undefined
    c) 0
    d) false
78. **What is the result of (0.1 + 0.2 === 0.3)?**
    a) true
    b) false
    c) undefined
    d) NaN