# BCA First year
# Unit 1

1. **Define the Internet and explain any four applications of the Internet.**

**Ans :** The Internet is a global system that connects computers and devices worldwide, enabling communication, information sharing, and access to digital services.

- **Connects people and devices globally**
- **Enables communication, learning, and business**
- **Supports digital services and cloud platforms**
- **Operates using standard communication protocols**
- **Forms the backbone of modern digital life.**

### History of the Internet

The Internet originated in the 1960s with the development of ARPANET (Advanced Research Projects Agency Network). It was designed to allow multiple computers to communicate within a single network an achievement that was revolutionary at the time.

### How Does the Internet Work?

The Internet operates using a client–server model:

- **Client:** A device such as a laptop or smartphone connected to the Internet
- **Server:** Powerful computers that store websites and data

### Step-by-step working:

- The client sends a request through a browser
- The server identifies the website using its IP address
- DNS converts the domain name into an IP address
- The request is routed to the correct server
- The server processes the request and returns the webpage

The Internet carries a vast range of informa on resources and services, such as the inter-linked hypertext documents of the World Wide Web (WWW) and the infrastructure tosupport electronic mail.

**Applications :**

1**. Search engine**: It can be used to search anything and everything. Most popular search engines are google and yahoo searches.

**2. Shopping:** Shopping has become easier with the advent of internet. You can buy or sell online

3. **Communication** : This is a major role of the internet. It helps people to communicate either with the use of social networking websites or through e mails. Even cha ng is a major use of the internet.

4**. Job search:** Nowadays, many people search for their jobs online as it is quicker and there is a larger variety of job vacancies present.

5. **Hobbies:** Those who are having certain hobbies can try to improve on it by reading up on many aspects of their hobby.

**6. Research**: Research papers are present online which helps in the researcher doing a literature review.

**7. Studying**: Now right from kinder garden children are exposed to internet and computers. They find many useful things to learn on the internet (though with supervision). Upto doctorate level education, people rely on internet for their education. Online educational books have even reduced the need for a library.

**Q.2 What is the World Wide Web (WWW)? Explain its relationship with the Internet.**

**World Wide Web (WWW)**

The World Wide Web is a collection of web pages and documents accessed through URLs (Uniform Resource Locators) on the Internet. For example, **www.geeksforgeeks**.org is a URL, and its content is part of the WWW. It serves as an information retrieval system where documents are linked using hypertext or hypermedia. Hyperlinks allow users to click on words or phrases to easily access related information.

The World Wide Web (WWW), often called the Web, is a system of interconnected webpages and information that you can access using the Internet. It was created to help people share and find information easily, using links that connect different pages together.

- The Web allows us to browse websites, watch videos, shop online, and connect with others around the world through our computers and phones.
- All public websites or web pages that people may access on their local computers and other devices through the internet are collectively known as the World Wide Web or W3.
- Users can get further information by navigating to the links interconnecting these pages and documents.
- This data may be presented in text, picture, audio, or video formats on the internet.

**Relationship with the Internet**

- **Infrastructure vs. Service:** The Internet is the underlying physical infrastructure (a global network of computers, cables, and servers), while the Web is a service/application running on top of it.
- **Subset Concept:** The Web is a subset of the Internet; other services like email (SMTP), FTP, and online gaming also run on the internet.
- **Analogy:** The Internet is the "bookstore," while the World Wide Web is the "collection of books" within it.

**Difference Between World Wide Web and the Internet**

- The key difference between the World Wide Web and the Internet are:

| World Wide Web | Internet |
|---|---|
| All the web pages and web documents are stored there on the | The Internet is a global network of computers that is accessed by the World wide web. |

| World Wide Web | Internet |
|---|---|
| World wide web and to find all that stuff you will have a specific URL for each website. | |
| The world wide web is a service. | The Internet is an infrastructure. |
| The world wide web is a subset of the Internet. | The Internet is the superset of the world wide web. |
| The world wide web is software-oriented. | The Internet is hardware-oriented. |
| The world wide web uses HTTP. | The Internet uses IP Addresses. |
| The world wide web can be considered as a book from the different topics inside a Library. | The Internet can be considered a Library. |
| Examples: Websites, e-commerce, blogs | Example: The network connecting all online services |

**Q-3 Differentiate between Static and Dynamic websites with suitable examples.**

There are two basic methods of web design: static and dynamic web pages. Users access static web pages, which present the same content every time they are viewed. On the other hand, dynamic webpages create content instantly in response to user input and present customized or updated information. Let's learn about this in detail in this article.

| Feature | Static Website | Dynamic Website |
|---|---|---|
| **Content** | Fixed; every user sees the same content. | Varies based on user input, location, time, etc.. |

| | | |
|---|---|---|
| **Content Generation** | Pre-built files delivered as-is (e.g., HTML, CSS, JavaScript). | Generated on the fly using server-side processing and databases. |
| **Languages Used** | Primarily client-side languages like HTML, CSS, and basic JavaScript. | Uses server-side scripting languages (PHP, Python, Node.js, ASP.NET) and databases (MySQL, MongoDB). |
| **Interactivity** | Limited to basic client-side interactions (e.g., animations, clickable links). | Highly interactive; supports user accounts, comments, search functions, and e-commerce. |
| **Updates** | Requires manual editing and re-uploading of individual files. | Managed easily through a Content Management System (CMS), with changes applying across the site. |
| **Performance** | Generally loads faster due to pre-rendered pages and fewer server-side processes. | Can have slightly slower load times due to server-side processing and database queries. |
| **Security** | Generally more secure due to fewer attack vectors (no database vulnerabilities). | More vulnerable to security risks like SQL injection, requiring more security measures. |
| **Cost** | Less expensive to develop and host due to simpler requirements. | More expensive to develop and host due to complexity and resource needs. |

Suitable Examples

**Static Website Examples**

These sites are ideal for information that rarely changes and doesn't require user interaction:

- **Personal Portfolios/Resumes:** A simple online resume or portfolio that showcases an individual's work, which is updated infrequently.
- **Small Business Brochure Sites:** Websites for local businesses (like a restaurant menu or a gym) that display basic, fixed information like contact details, hours, and a list of services.
- **Landing Pages:** Specific promotional pages for a marketing campaign where the content remains consistent for all visitors.
- **Documentation or Manuals:** Technical documentation that needs to be fast-loading and easily indexed by search engines.

**Dynamic Website Examples**

These sites are ideal for content that updates frequently, allows user interaction, and needs personalization:

- **E-commerce Stores:** Websites like Etsy or Amazon, where product listings, prices, inventory, and shopping carts rely on real-time data from a database.
- **Social Media Platforms:** Sites like Facebook or X (formerly Twitter) that feature user-generated content, personalized feeds, and user accounts that change constantly.
- **News and Blogging Websites:** Platforms where new articles are published frequently and content is managed through a CMS, often with features like comment sections and search functions (e.g., a blog built with WordPress).
- **Streaming Services:** Services such as Netflix which provide personalized recommendations and "continue watching" features based on user viewing history and preferences.

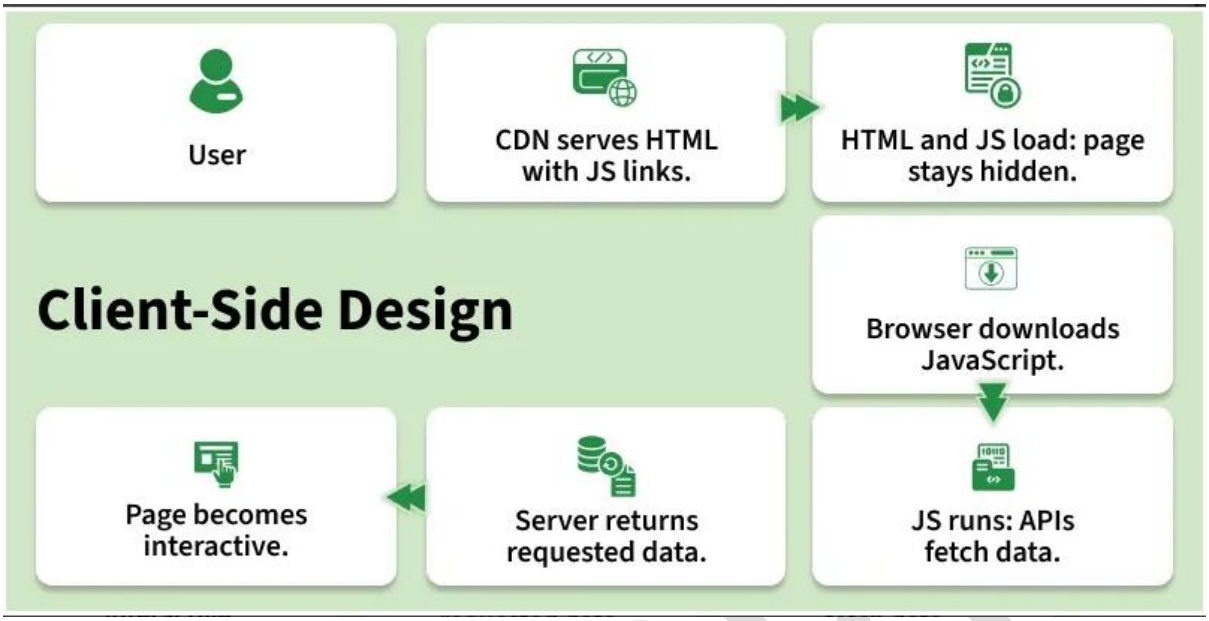
**Q 4 - Explain the Client-Server Architecture in the context of the World Wide Web.**

Client–server architecture is a widely used system design where multiple clients request services or resources from a central server. The server handles processing, data storage, and resource management, while clients focus on user interaction, enabling efficient, scalable, and organized distributed systems.

- Clients send requests and receive responses using protocols like HTTP/HTTPS or SQL.
- The server centralizes data management, complex processing, and storage.
- The architecture is scalable by upgrading servers or adding multiple servers.
- Commonly used in web applications, databases, and email systems.

**Key Aspects of Web Client-Server Architecture:**

- **The Client (Front-end):** Devices or applications (e.g., Chrome, Safari, mobile apps) that request, display, and manage user interactions.
- **The Server (Back-end):** A powerful computer that processes requests, manages databases, and stores data (HTML, images).
- **Request-Response Cycle:** The fundamental communication flow:
    - **Client Request:** You enter a URL in your browser, which sends an HTTP request over the internet to a server.
        - **Server Processing:** The server receives the request, runs any necessary backend code, and retrieves data from a database.
        - **Server Response:** The server sends the requested data (HTML, JSON, images) back to the client.
- **Protocols:** Standardized languages like **HTTP** (HyperText Transfer Protocol) or **HTTPS** (secure version) are used for communication.
- **Key Components:** In addition to the client and server, this model often involves intermediaries like **DNS servers** to find the server's IP address




**Real-World Examples of Client-Server Architecture:**

Some are some real-world examples of Client Server Architecture are:

1. **Banking Systems:** Online banking applications use client-server architecture to enable customers (clients) to interact securely with bank servers. Clients can perform transactions, check balances, and manage accounts through web or mobile applications.
2. **Enterprise Applications:** Large organizations use client-server architecture for various internal systems such as ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), and document management systems. Clients (employees or users) access centralized servers to retrieve and update data related to business operations.
3. **Telecommunications:** Telecom networks rely heavily on client-server architecture for services like voice-over-IP (VoIP), video conferencing, and messaging applications. Clients (users' devices or applications) communicate through servers that manage call routing, signaling, and media streaming.
4. **Internet of Things (IoT):** IoT ecosystems often employ client-server architecture for managing devices and collecting sensor data. Devices act as clients that send data to servers for processing, storage, and analysis. Examples include smart home systems and industrial IoT applications.
5. **Healthcare Systems:** Electronic Health Record (EHR) systems and telemedicine platforms utilize client-server architecture to securely manage patient records, facilitate remote consultations, and exchange medical data between healthcare providers and patients.

**Q. 5 Compare the HTTP and HTTPS protocols, highlighting why security is essential for modern web applications.**

HTTP (Hypertext Transfer Protocol) transfers data in plain text, making it vulnerable to interception, while HTTPS (Secure) uses TLS/SSL to encrypt communications. HTTPS uses port 443, ensures data integrity, and authenticates servers. Security is essential for modern applications to protect user data, ensure privacy, and maintain trust.

**Key Differences Between HTTP and HTTPS**

- **Security Mechanism:** HTTP offers no encryption, meaning data is sent in plain text, which can be read by anyone. HTTPS encrypts data, protecting sensitive information like passwords, credit card numbers, and personal data from eavesdropping.
- **Data Integrity:** HTTPS prevents data from being altered or corrupted during transmission. If a hacker intercepts HTTP traffic, they can modify it; with HTTPS, this is not possible.
- **Authentication:** HTTPS provides validation that you are communicating with the intended website (via SSL certificate), protecting users from fake or phishing websites.
- **URL & Port:** HTTP URLs begin with http:// and use port 80. HTTPS URLs begin with https:// and use port 443.
- **User Trust & SEO:** Browsers flag HTTP sites as "Not Secure". HTTPS websites are marked with a padlock icon, increasing user trust, and they often receive better rankings from search engines.

### Why Security (HTTPS) is Essential for Modern Web Applications

- **Protection of Sensitive Data:** Modern applications frequently handle login credentials, personal data, and financial transactions, requiring strong encryption to comply with regulations like GDPR and PCI DSS.
- **Prevention of Attacks:** HTTPS protects against Man-in-the-Middle (MITM) attacks, where a third party tries to intercept or alter communication between the client and server.
- **User Trust and Experience:** Users are more likely to trust and interact with a website that shows a secure connection, directly impacting user engagement.
- **Improved Search Engine Optimization (SEO):** Search engines, such as Google, prioritize HTTPS sites in search results, improving visibility.
- **Faster Performance:** While encryption once slowed down sites, modern HTTPS implementations often load faster than insecure HTTP alternatives.

Q-6 Define **HTML** and **CSS**, explaining their specific roles in building the structure and style of a webpage.

**HTML** (HyperText Markup Language) and **CSS** (Cascading Style Sheets) are the foundational technologies used to build web pages, working together to create structured, visually appealing, and functional websites. HTML provides the content and structure, while CSS manages the visual presentation and layout.

**HTML** stands for HyperText Markup Language. It is the standard language used to create and design web pages on the internet. It was introduced by Tim Berners-Lee in 1991 at CERN as a simple markup language HTML provides the essential structure and content of a webpage. It uses a system of elements and tags to define the different parts of a page

HTML is a markup language that serves as the **structural backbone** or "skeleton" of a webpage. It uses a system of tags and elements to organize content and give it semantic meaning.

**Role in structure**: HTML defines the different parts of a webpage, telling the browser what each piece of content represents. Key structural elements include:

- **Headings** (<h1> to <h6>): Define titles and subtitles, establishing a content hierarchy.
- **Paragraphs** (<p>): Organize text into readable blocks.
- **Lists** (<ul>, <ol>, <li>): Structure items in bullet points or numbered orders.
- **Links** (<a>): Create hyperlinks to connect different pages and resources on the web.
- **Media** (<img>, <video>, <audio>): Embed images, audio, and video content.
- **Semantic elements** (<header>, <footer>, <article>, <section>): Provide additional meaning to content blocks, which improves accessibility and search engine optimization (SEO)

```
<html>
<head>
  <title>My First Webpage</title>
</head>
<body>
  <h1>Welcome to My Webpage</h1>
  <p>This is my first paragraph of text!</p>
</body>
</html>
```

**CSS (Cascading Style Sheets)**

**CSS** is a style sheet language responsible for the **visual presentation** and "design" of the structured HTML content. It determines how the HTML elements should be displayed on various screen sizes and devices, turning a plain document into a visually engaging experience.

- **Role in style**: CSS controls the aesthetic aspects and layout of the webpage, allowing developers to customize:
  - **Colors and fonts**: Define text colors, font families, sizes, and styles.
  - **Layout and positioning**: Arrange elements using properties like display, position, flexbox, and grid to create complex, responsive layouts.
  - **Spacing and dimensions**: Control the margins, padding, and borders of elements to manage the space between them (the CSS box model).
  - **Visual effects**: Add animations, transitions, and hover effects to enhance user interaction.
  - **Responsiveness**: Use media queries to adapt the design and layout of the site for different devices and screen sizes.

```
<!DOCTYPE html>
<html>
```

```
<body>
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">
This is CSS Example</p>
</body>
</html>
```

**HTML** provides the content and its underlying order, while CSS provides the visual instructions on how that order should look, enabling a clean separation of content from design.

**Q-7 What is web hosting? Explain any two types of hosting (Shared, VPS, Cloud, Localhost).**

Web hosting is a service that provides server space and technologies to make websites accessible on the internet. It stores website files (HTML, images, database) on a specialized computer (server) that remains connected to the web. The host ensures users can view the site by typing the domain name.

Here are two common types of web hosting:

1. **Shared Hosting**
   - **Definition:** Multiple websites reside on a single physical server, sharing resources like RAM, CPU, and bandwidth.
   - **Pros/Cons:** It is the most affordable and beginner-friendly option. However, high traffic on one site can potentially slow down other sites on the same server.
   - **Best For:** Small blogs, portfolios, and new websites with low traffic.
2. **Virtual Private Server (VPS) Hosting**
   - **Definition:** A step up from shared hosting, where a single physical server is divided into virtual, isolated compartments.
   - **Pros/Cons**: Each user gets dedicated, guaranteed resources within their virtual environment, offering better performance, security, and control than shared hosting. It requires more technical knowledge to manage.
   - **Best For:** Medium-sized businesses, e-commerce sites, and rapidly growing websites.

Other types include **Cloud hosting** (using a network of servers for high reliability) and **Localhost** (a local computer server used for development and testing).

**Q-8 what is text editors?**

A text editor is a software application designed for creating, editing, and managing plain text files. Unlike word processors (like Microsoft Word), they do not add formatting like bold, italics, or varying fonts to the file itself, making them essential for writing computer code where "extra" formatting would break the program

**Popular Text Editors**

Text editors are applications where developers write and manage their source code.

- **Visual Studio Code (VS Code):** Developed by Microsoft, it is currently the most popular, free, and open-source editor. It functions like a "workbench" with a massive ecosystem of extensions for debugging, Git integration, and AI-assisted coding. While feature-rich, it can be heavier on system memory than simpler editors.
- **Sublime Text:** A lightweight, high-performance editor known for its extreme speed and "distraction-free" interface. It excels at handling very large files and offers powerful features like "Goto Anything" and multiple cursors. While it has a free trial, continued use requires a paid license.

**UNIT-2**

**Foundation of web development**

**1. What is HTML?**

**Answer:**
HTML (Hypertext Markup Language) is used to create and structure web pages. It uses tags to display content in a browser.

- HTML stands for **Hypertext Markup Language**
- Used to create web pages
- It is not a programming language, but a **markup language**
- Works with browsers like Google Chrome, Mozilla Firefox

**2. Write the basic structure of an HTML document.**

```
<!DOCTYPE html>
<html>
<head>
  <title>My Page</title>
</head>
<body>
  <h1>Hello World</h1>
</body>
</html>
```

- <!DOCTYPE html> → HTML5 declaration
- <html> → root element
- <head> → metadata
- <body> → visible content

**3. What are tags and attributes?**

**Answer:**

HTML tags are the fundamental structural building blocks (e.g., <p>, <h1>, <a>) that define elements in a document, usually written in pairs, such as <tag>content</tag>. Attributes are special modifiers added within an opening tag to provide additional information, behavior, or properties, usually in name="value" pairs (e.g., href in <a href="...">)

- **Tags**: Define the start and end of an element. They define the type of content (text, images, links).
- **Attributes:** Describe the characteristics of an element and are always placed inside the opening tag

.

**Usage Examples:**

- **Tag Example ( ): This is a paragraph.  (Creates a paragraph).**
- **Tag Example (<a>): <a>Click here</a> (Creates a link).**
- **Attribute Example (href): <a href="https://example.com"> (Defines the destination).**
- **Attribute Example (src & alt): <img src="image.jpg" alt="Description"> (Defines image source and description).**
- **Attribute Example (style):  (Applies CSS styling)**

**4. What are headings and paragraphs?**

Headings and paragraphs are the basic building blocks used to structure text on a webpage. In HTML, these are defined by specific tags that tell the browser how to display and organize content.

**HTML Headings**

Headings are used to define the titles and subtitles of a page, creating a visual and structural hierarchy.

- **Six Levels:** HTML provides six heading levels, from <h1> to <h6>.
- **Hierarchy:** <h1> is the most important (the main title), while <h6> is the least important.
- **Visual Style:** By default, higher-level headings (like <h1>) appear larger and bolder than lower-level ones.

**HTML Paragraphs**

Paragraphs are used to group related sentences into distinct blocks of text.

- **The  Tag:** A paragraph is defined by wrapping text inside  and  tags.
- **Automatic Spacing:** Browsers automatically add a small amount of white space (margins) before and after a  element to separate it from other content.
- **Line Breaks:** Paragraphs always start on a new line. If you need a line break *within* a paragraph without starting a new one, use the empty <br> tag.
- **Formatting:** Browsers ignore extra spaces or line breaks created by pressing "Enter" inside your code; you must use  or <br> tags to control the layout.

**5. What is an anchor tag?**

The <a> tag defines a hyperlink that connects one page or resource to another. Its key attribute, href, specifies the destination URL where users are directed upon clicking.

- Can link to web pages, email addresses, phone numbers, or sections within the same page.
- Supports attributes like target, rel, and download for enhanced functionality.

**Syntax**

```
<a href = "link"> Link Name </a>
<a href="https://www.example.com">Visit Example Website</a>
```

- **<a>**: The opening tag that starts the link.
- **href:** Short for "hypertext reference," this attribute specifies the destination URL.
- **Anchor Text:** The clickable content (text or images) located between the tags.
- **</a>:** The closing tag that ends the link.

## 6. What is the image tag?

The HTML image tag is <img>, an empty, self-closing element used to embed images into a webpage. It requires the src attribute to specify the image path and the alt attribute for descriptive text. It is an inline element that supports attributes for sizing (width, height) and styling.

**Key Aspects of the <img> Tag:**

- **Syntax:** <img src="image.jpg" alt="Description">.
- **Essential Attributes:**
  - **src (Source):** Defines the URL or path to the image.
  - **alt (Alternative Text):** Provides text if the image cannot be displayed, crucial for accessibility (SEO).
- **Optional Attributes:** width, height, style, title.
- **Self-Closing:** It does not require a </img> closing tag.

**7. What are lists in HTML?**

In HTML, lists are semantic elements used to organize related information in a structured, easy-to-read format. They are commonly used for general content, navigation menus, and grouping related data.

**There are three primary types of lists in HTML:**

- **Unordered List (<ul>):** Used for items where the specific order does not matter (e.g., a grocery list). By default, items are displayed with bullet points (discs).
  - Tag: <ul> wraps the entire list.
  - Item Tag: <li> defines each individual list item.
- **Ordered List (<ol>):** Used for items that follow a specific sequence or ranking (e.g., a recipe or a top-10 list). By default, items are numbered numerically.
  - Tag: <ol> wraps the entire list.
  - Customization: The type attribute can change numbering to letters or Roman numerals, and the start attribute can change the beginning value.
- **Description List (<dl>):** Formerly known as a definition list, it is used for name-value pairs, such as terms and their corresponding definitions or glossaries.
  - Tag: <dl> defines the start of the list.
  - Term Tag: <dt> defines the specific term being described.
  - Description Tag: <dd> defines the explanation or definition for that term.

Key Features of HTML Lists

- Nesting: You can create "nested lists" by placing one list (ordered or unordered) inside a <li> element of another list.
- Semantics and Accessibility: Using list tags instead of plain text helps Screen Readers communicate the structure to users with visual impairments.

- Styling: While HTML provides basic markers, developers typically use CSS list-style-type to customize markers with different shapes, images, or to remove them entirely (often done for navigation bars).

**9. What are tables in HTML?**

HTML tables are structured elements used to organize and display tabular data—information that is best presented in a two-dimensional grid of rows and columns. While they were once used for overall website layouts, modern web development uses CSS for layout and reserves tables strictly for data such as schedules, pricing lists, or financial reports.

**Core Structural Elements**

An HTML table is built using several nested tags that define its framework:

- <table>: The top-level container that marks the start and end of the table.
- <tr> (Table Row): Defines a single horizontal row of cells.
- <th> (Table Header): Defines a header cell, typically used for column or row titles. By default, the text is bold and centered.
- <td> (Table Data): Defines a standard cell containing the actual data (text, images, or links).

**Advanced Semantic Tags**

For complex data, additional tags improve accessibility and organization:

- <thead>, <tbody>, and <tfoot>: Used to group the header, main body, and footer sections of the table respectively.
- <caption>: Provides a title or description for the entire table.
- <colgroup> and <col>: Allow you to apply styles to entire columns rather than individual cells**.**

**Key Attributes**

- colspan: Allows a single cell to span across multiple columns.
- rowspan: Allows a single cell to span across multiple rows.
- border: While a border attribute exists for the <table> tag, it is now standard practice to manage all borders and spacing using the CSS border property.

**10. What are forms in HTML?**

In HTML, forms are specialized sections of a document used to collect and submit user input to a server for processing. They serve as the primary point of interaction on the web, enabling tasks such as registration, logging in, and submitting search queries**.**

**Core Components**

**An HTML form is defined by the <form> tag, which acts as a container for various interactive controls.**

- The <form> Element: Wraps all other form components. It typically includes the action attribute (specifies the URL where data is sent) and the method attribute (defines the HTTP method, usually GET or POST).

- Form Controls: These are the interactive elements users fill out:
    - **<input>:** The most versatile element, modified by its type attribute to create text fields, passwords, checkboxes, radio buttons, and file uploads.
    - **<textarea>:** Used for multi-line text input, such as comments or descriptions.
    - **<select> and <option>:** Creates dropdown menus for choosing from predefined lists.
    - **<button>:** Defines clickable buttons, most commonly a "Submit" button to send data.
- **Descriptive Elements:**
    - **<label>:** Provides captions for controls and is essential for accessibility and usability.
    - **<fieldset> and <legend>:** Used to group related controls visually and semantically.

**Key Attributes**

- **name:** Essential for every input; it acts as a key for the data sent to the server **(e.g., username="Mona").**
- **value:** Defines the initial value of an input or the value sent if the control is selected **(like in radio buttons).**
- **Validation Attributes:** HTML5 includes built-in attributes like required, pattern, and maxlength to ensure users provide the correct information before submission**.**

**11. What are different input types?**

The HTML <input> element uses the type attribute to determine what kind of data the user can enter. Below is a categorized list of all standard HTML input types.

**Common HTML Input Types**

- **Text (type="text")**: The default type; creates a single-line text field for general data like names or usernames.
- **Password (type="password")**: A single-line text field that masks characters (typically with dots or asterisks) to hide sensitive information.
- **Checkbox (type="checkbox")**: A box that can be toggled on or off, allowing users to select multiple options from a set.
- **Radio (type="radio")**: Used in groups where a user can select only one option from a set of mutually exclusive choices.
- **Submit (type="submit")**: A button that, when clicked, sends the form's data to a server.
- **Email (type="email")**: A field that automatically validates whether the input is a correctly formatted email address.
- **Number (type="number")**: A field specifically for numeric values, often featuring "spinners" (up/down arrows) to change the value.
- **File (type="file")**: Provides a "Browse" button for users to select one or more files to upload from their device.

**12. What is a select tag?**

The <select> tag in HTML is used to create a drop-down list of options within a web page, typically used in forms to collect user input. It allows users to choose one or more predefined choices from a menu.

**Key Components**

- <select>: Acts as the container for the entire drop-down menu.
- <option>: Nested inside the select tag, these define the individual choices available to the user.
- <optgroup>: An optional tag used to group related <option> elements into categories for better organization.

**Common Attributes**

- name: Defines the name of the control, which is used to identify the data when the form is submitted to a server.
- multiple: A boolean attribute that allows users to select more than one option at a time (typically by holding the Ctrl or Cmd key).
- size: Specifies the number of visible options shown at once. By default, it is usually 1, but setting a higher number turns the drop-down into a scrolling list.
- required: Ensures the user must select a value before they can submit the form.
- disabled: Makes the entire drop-down unclickable and unusable.

**Basic Example**

```
<label for="colors">Choose a color:</label>
<select name="colors" id="colors">
  <option value="red">Red</option>
  <option value="green">Green</option>
  <option value="blue">Blue</option>
</select>
```

**13. What are semantic tags?**

**Semantic tags** are HTML elements that clearly describe their meaning and purpose to both the browser and the developer. Unlike non-semantic tags like  and , which act as generic containers with no inherent meaning, semantic tags explicitly define what kind of content is contained within them.

- **Accessibility:** Screen readers and other assistive technologies use these tags to navigate pages more effectively, helping users with disabilities understand the site structure.
- **SEO (Search Engine Optimization):** Search engines like Google use semantic cues to better index content and identify the most important parts of a page.
- **Code Readability:** Meaningful names make it easier for developers to read, maintain, and collaborate on code without needing excessive comments.
- **Future-Proofing:** They follow web standards that ensure better compatibility with future technologies.

**Common Semantic Tags and Their Uses**

- **<header>**: Represents introductory content at the top of a page or section, often containing logos and navigation.
- **<nav>**: Defines a block of major navigation links, such as site menus or tables of contents.
- **<main>**: Specifies the primary, unique content of a document; it should appear only once per page.
- **<section>**: Groups related content together, typically sharing a common theme or heading.
- **<article>**: Defines self-contained, independent content that could be reused elsewhere, such as a blog post or news story.
- **<aside>**: Contains content tangentially related to the main content, like sidebars, advertisements, or pull quotes.
- **<footer>**: Indicates the bottom section of a page or section, typically holding copyright information and contact details.
- **<figure> and <figcaption>**: Used together to embed media (like images or charts) with a specific caption.
- **<time>**: Represents a specific date, time, or duration in a machine-readable format.
- **<mark>**: Used to highlight or mark text that is relevant in a particular context.

**14. What is multimedia in HTML?**

Multimedia in HTML refers to the integration of different media formats—such as sound, music, videos, animations, and graphics—directly into web pages to create a more engaging and interactive user experience.

With the advent of HTML5, developers can embed these elements natively using specific tags without requiring external plugins like Flash.

**Key Multimedia Elements and Tags**

- <video>: Used to embed video content or movie clips. It supports attributes like controls (to show play/pause/volume), width, and height.
- <audio>: Used to embed sound, music, or voice recordings. Like the video tag, it often includes a controls attribute for user interaction.
- <img>: The standard tag for displaying static images.
- <source>: A nested tag used inside <audio> and <video> to specify multiple media files. This allows the browser to choose the first format it supports.
- <iframe>: Frequently used to embed external multimedia content, such as a YouTube video player.
- <embed> and <object>: Older tags used to define containers for external applications or multimedia files like PDFs and Flash.

**Supported Formats in HTML5**

While there are many media formats, the HTML5 standard officially supports a specific set for maximum browser compatibility:

- Video: MP4 (.mp4), WebM (.webm), and Ogg (.ogg). MP4 is the most recommended for wide support.
- Audio: MP3 (.mp3), WAV (.wav), and Ogg (.ogg). MP3 is the most common format for web audio.

**Common Attributes**

Multimedia tags use various attributes to control playback behavior:

- controls: Displays the standard browser media player UI.
- autoplay: Starts the media automatically when the page loads (often requires the muted attribute to work in modern browsers).
- loop: Causes the media to restart automatically once it finished.
- muted: Specifies that the audio output should be silent by default.
- poster: Specifies an image to be shown while the video is downloading or until the user hits the play button.

**15. What is iframe?**

An **<iframe>** (short for **Inline Frame**) is an HTML element used to embed another HTML document or external content directly within the current webpage. It essentially acts as a "window" that allows visitors to view and interact with content from another source—such as a video, map, or advertisement—without leaving the parent page.

**Key Features and Functions**

- **Nested Browsing Context:** It creates a separate browsing environment that can have its own CSS, JavaScript, and session history independent of the main page.
- **Multimedia Integration:** Widely used to embed YouTube videos, Vimeo clips, or Google Maps.
- **Third-Party Widgets:** Often used for payment gateways, social media feeds, or interactive forms and customer service chat boxes.

**Common Attributes**

| Attribute | Description |
|---|---|
| **src** | Specifies the URL of the document or webpage to be embedded. |
| **height / width** | Defines the size of the iframe (defaults to 150px height and 300px width). |
| **title** | Provides a description for screen readers, which is essential for web accessibility. |
| **sandbox** | Enables security restrictions to prevent malicious actions like script execution or pop-ups from the embedded content. |

| | |
|---|---|
| **loading** | Can be set to lazy to improve performance by loading the iframe only when it is near the user's viewport. |

**Basic Syntax Example**

<iframe src="https://www.example.com" width="600" height="400" title="Description of content"></iframe>

**16. What is canvas and SVG?**

Canvas and SVG are HTML5 technologies for rendering 2D graphics in browsers. **Canvas** (raster/pixel-based) uses JavaScript to draw graphics dynamically, ideal for games and high-frequency animations. **SVG** (vector-based) uses XML to define shapes in the DOM, making it perfect for logos, icons, and scalable graphics.

**Key Differences Between Canvas and SVG:**

- **Technology:** Canvas is pixel-based (raster), while SVG is shape-based (vector).
- **Scalability:** SVG scales infinitely without quality loss; Canvas pixelates when scaled.
- **DOM Integration:** SVG elements are part of the DOM, allowing CSS styling and event handlers for each shape. Canvas is a single DOM node.
- **Performance:** Canvas performs better for large numbers of objects or high-interaction, real-time pixel updates (e.g., gaming). SVG excels with fewer objects and large surface areas.
- **Best Use Cases:**
    - **Canvas:** Pixel manipulation, complex games, high-frequency animations.
    - **SVG:** High-resolution icons, diagrams, responsive graphics, interactive illustrations.

**Example Definitions:**

- **Canvas:** A blank, scriptable canvas <canvas> element requiring JavaScript to render graphics.
- **SVG:** XML-based code (e.g., <svg><circle.../></svg>) defining shapes, which are then rendered by the browser.

**17. What are meta tags?**

Meta tags are HTML elements placed in the <head> section of a webpage that provide structured metadata about a page's content, such as its description, character set, and viewport settings. They are invisible to users on the page itself but crucial for search engines (SEO) and browsers to understand, index, and display the site correctly.

Key details about meta tags include:

- **Purpose:** They improve SEO, control how the page looks on social media (via Open Graph), and ensure responsiveness on mobile devices.
- **Placement:** They are always placed within the <head> section of HTML code.
- **Essential Types:**

- **Meta Description:** A brief summary of the page content, often displayed in search results.
- **Title Tag:** While technically a <title> tag, it functions as the main meta title in search results.
- **Viewport:** Configures the page to scale correctly on mobile devices.
- **Charset:** Defines character encoding (e.g., UTF-8).
- **Open Graph / Twitter Cards:** Controls how links look when shared on social media.

Meta tags are crucial for SEO and digital marketing because they help search engines understand the page and improve click-through rates.

**18. What is character encoding?**

Character encoding in HTML is a method used to convert a sequence of bytes into readable characters. It acts as a "key" that tells a web browser how to interpret the numeric data stored in a file so it can correctly display text, symbols, and emojis.

**Why Character Encoding Matters**

- Correct Rendering: Without a specified encoding, browsers may guess incorrectly, resulting in "gibberish" text (like `` or Ã©) instead of the intended characters.
- Multilingual Support: It allows websites to display characters from diverse languages (e.g., Chinese, Arabic, or Cyrillic) and special symbols like currency signs.
- Security: Proper encoding helps prevent security vulnerabilities such as Cross-Site Scripting (XSS) by ensuring that user input is treated as plain text rather than executable HTML code.

**Common Encoding Standards**

- UTF-8: The current universal standard and default for HTML5. It is highly efficient because it uses variable-length encoding (1 to 4 bytes) and covers almost every character and symbol in the world.
- ASCII: One of the earliest standards, limited to 128 characters (primarily English letters, numbers, and basic symbols).
- ISO-8859-1 (Latin-1): The default for HTML 4.01, supporting Western European languages but lacking support for many other international scripts.

**9. What are HTML5 updates?**

HTML5 updates represent a major evolution of the Hypertext Markup Language, finalized in 2014 and maintained as a "living standard" to support modern web applications, rich multimedia, and mobile devices without requiring third-party plugins like Flash. It introduced new semantic tags, better form controls, and APIs that improve web structure, accessibility, and performance.

Here are the key updates and features introduced by HTML5:

**1. New Semantic Elements (Structure)**

HTML5 replaced generic  tags with semantic elements that clearly define the purpose of page content, improving readability, SEO, and accessibility:

- **<header> & <footer>:** Mark the top and bottom sections.
- **<nav>:** Defines a section of navigation links.
- **<article>:** Defines self-contained content.
- **<section>:** Groups related content.
- **<aside>:** Defines content aside from the main content.
- **<figure> & <figcaption>:** Groups media (like images) with a caption.

**2. Native Multimedia Support**

HTML5 removed the need for Flash plugins by introducing native support for audio and video:

- **<audio>:** Embeds sound content.
- **<video>:** Embeds video content.
- **<source> & <track>:** Enables multiple media sources and text tracks (subtitles).

**3. Graphics and Visual Effects**

- **<canvas>:** Allows JavaScript to draw 2D graphics, animations, and games directly in the browser.
- **SVG Integration:** Provides native support for Scalable Vector Graphics, ideal for logos and icons.

**4. Enhanced Web Forms**

New input types and attributes improved validation and usability, particularly on mobile devices:

- **Input Types:** email, date, time, url, number, range, color, search.
- **Attributes:** placeholder (short hint), required (validation), autofocus.
- **Elements:** <datalist> (autocomplete), <progress> (task progress), <meter> (data measurement).

**5. New APIs (Application Programming Interfaces)**

HTML5 introduced several APIs that make web apps function more like native applications:

- **Web Storage (localStorage & sessionStorage):** Stores large amounts of data securely in the client's browser, replacing cookies.
- **Geolocation API:** Allows websites to access the user's location (with permission).
- **Offline Web Applications (Application Cache):** Enables web apps to work without an internet connection.
- **Web Workers:** Allows JavaScript to run in the background without affecting the responsiveness of the main page.
- **WebSockets:** Enables two-way, real-time communication between the client and server.

**6. Improved Mobile-Friendly Design**

- **Responsive Viewport:** HTML5 helps websites adapt to different screen sizes and devices, supporting responsive web design.
- **Touch Events:** Native support for touch-screen interactions.

**7. Clean Syntax and Removed Elements**

- **Simple Doctype:** The declaration was simplified to <!DOCTYPE html>.
- **Deprecated Elements:** Outdated presentational tags such as <font>, , and <frame> were removed in favor of CSS.

**20. What are HTML best practices?**

Following best practices in HTML ensures your website is accessible, SEO-friendly, and easy to maintain. Key standards include:

- Always Declare a Doctype: Start every document with <!DOCTYPE html> to ensure the browser renders the page in "standards mode".
- Use Semantic HTML: Prefer meaningful elements like <header>, <nav>, <main>, <article>, and <footer> over generic  tags to improve SEO and accessibility.
- Maintain a Proper Heading Hierarchy: Use only one <h1> per page for the main topic and follow a logical order (<h2> to <h6>) without skipping levels.
- Always Include Image Alt Text: Every <img> tag must have an alt attribute. Use descriptive text for content images or an empty alt="" for decorative ones.
- Keep Markup in Lowercase: Use lowercase for all element names and attributes to maintain industry standards and improve code readability.
- Separate Content from Presentation: Do not use inline styles or presentational tags like <b> or <i>. Use external CSS files for styling and semantic tags like  or  for meaning.
- Close All Tags: Even if a browser can guess a missing tag, explicitly closing tags prevents layout glitches and ensures your code is standards-compliant.
- Use Meaningful Titles and Meta Tags: Define a unique, descriptive <title> (under 60 characters) and a <meta name="description"> to help search engines index your site correctly.
- Format and Indent Consistently: Use consistent indentation (e.g., 2 spaces) to visually represent the document structure and make the code easier for others to read.
- Validate Your Code: Use tools like the W3C Markup Validation Service to find and fix syntax errors or non-compliant markup.
- Optimize Script Loading: Place <script> tags at the bottom of the <body> or use defer/async attributes to prevent blocking the initial page render.
- Label Web Forms Correctly: Always associate <label> tags with their corresponding <input> using the for attribute to ensure they are accessible to screen readers.

# Subject: Web Design
# Unit -4 JavaScript Basic

---

**Section A – Very Short Answer (1 Mark Each) (Attempt all questions)**

1. What is JavaScript?

Ans JavaScript is a **programming language used to create interactive and dynamic web pages**.

2. Write the correct syntax to declare a variable using let.

Ans let variableName;

Example:

let age = 20;

3. Name any two data types in JavaScript.

Ans  Number

 String

4. What is the use of const keyword?

Ans const is used to **declare a constant variable whose value cannot be changed** after assignment.

5. Which operator is used for comparison?

Ans **Comparison operators** such as ==, ===, >, <, !=.

6. Write the syntax of if statement.

Ans if (condition) {
// code to be executed
}

7. Which loop is used when number of iterations is known?

Ans : for loop

8. What is DOM?

Ans DOM (Document Object Model) is a **programming interface that allows JavaScript to access and modify HTML elements**.

9. Write the method used to select element by ID

Ans : document.getElementById("idName");

. 10. What is the use of alert()?

Ans : alert() is used to **display a popup message box in the browser**.

---

**Section B – Short Answer Questions (2–3 Marks Each)**

(Attempt any 8)

1. Explain different ways of placing JavaScript in HTML.

2. Differentiate between var, let, and const.
3. Explain primitive and non-primitive data types.
4. Write a program to add two numbers using JavaScript.
5. Explain arithmetic and logical operators with examples.
6. Difference between == and ===.
7. Explain switch statement with example.
8. Write a program using for loop to print numbers from 1 to 10.
9. What is a function? Explain function declaration and function expression.
10. Explain getElementById() and querySelector().
11. What is form validation? Why is it important?
12. Write short note on JavaScript events.

---

**Section C – Long Answer Questions (5–7 Marks Each)**

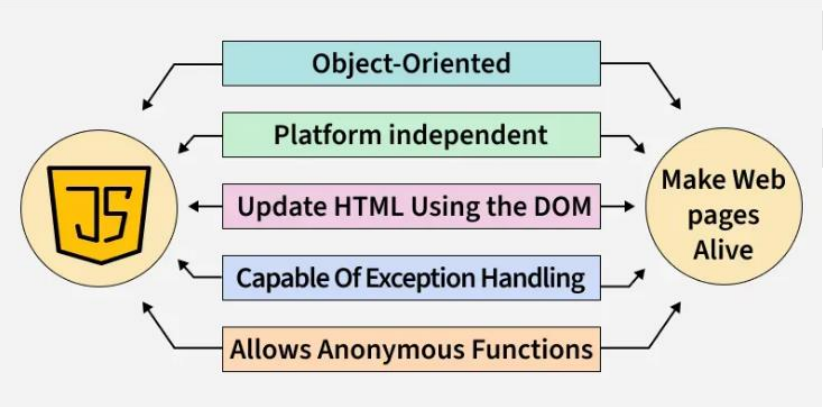
(Attempt any 5)

1. Explain JavaScript data types in detail with examples.
2. Explain all types of operators in JavaScript with suitable examples.
3. Describe conditional statements (if, else if, switch) with examples.
4. Explain different types of loops (for, while, do-while) with programs.
5. Explain different types of functions in JavaScript (declaration, expression, arrow function).
6. What is DOM Manipulation? Explain with examples of:
    - getElementById()
    - querySelector()
    - innerHTML
    - Changing CSS using style
7. Explain JavaScript events: click, submit, keypress, mouseover with examples.
8. Write a JavaScript program for simple form validation (e.g., empty field check).
9. Explain alert(), confirm(), and prompt() with examples.
10. Write short note on introduction to JavaScript libraries and basics of jQuery.

# Unit IV

## Q-1 JavaScript introduction and placement

JavaScript is a versatile, dynamically typed programming language that brings life to web pages by making them interactive. It is used for building interactive web applications, supports both client-side and server-side development, and integrates seamlessly with HTML, CSS, and a rich standard library.




**Introduction to JavaScript**

- **Purpose**: JavaScript allows developers to implement complex features on web pages, such as interactive maps, animated graphics, and real-time content updates. Before its advent, web pages were largely static.
- **Functionality**:
    - **Client-side**: It runs directly in the user's web browser, allowing for faster response times and reduced server load by handling user events and manipulating the page's structure (Document Object Model or DOM) and style (CSS).
    - **Server-side**: With runtime environments like Node.js, JavaScript can also be used on the server to handle databases, file manipulations, and security features, enabling full-stack development.

**Key Features**:

- **Lightweight and Interpreted**: Code is executed line by line by a JavaScript engine within the browser or server environment, without requiring a separate compilation step beforehand.

- **Cross-Platform**: It runs across all modern web browsers and various other devices.
- **Object-Oriented**: It supports object-oriented programming concepts through a prototype mechanism.
- **Rich Ecosystem**: A vast collection of libraries and frameworks (React, Angular, Vue.js, etc.) enhance its capabilities and speed up development.
- **जावास्क्रिप्ट** एक बहुउपयोगी (Versatile) और डायनेमिकली टाइप्ड प्रोग्रामिंग भाषा है, जो वेब पेजों को इंटरैक्टिव बनाकर उन्हें जीवंत बनाती है। इसका उपयोग इंटरैक्टिव वेब एप्लिकेशन बनाने के लिए किया जाता है। यह क्लाइंट-साइड और सर्वर-साइड दोनों प्रकार के विकास का समर्थन करती है तथा HTML, CSS और एक समृद्ध मानक लाइब्रेरी के साथ आसानी से एकीकृत हो जाती है।
- जावास्क्रिप्ट एक **सिंगल-थ्रेडेड भाषा** है, जो एक समय में एक ही कार्य को निष्पादित करती है। यह एक **इंटरप्रेटेड भाषा** है, अर्थात यह कोड को पंक्ति-दर-पंक्ति चलाती है। जावास्क्रिप्ट में वेरिएबल का डेटा टाइप रन-टाइम पर निर्धारित होता है, इसलिए इसे **डायनेमिकली टाइप्ड भाषा** कहा जाता है।
- ---
- ## जावास्क्रिप्ट की मुख्य विशेषताएँ
- **1. क्लाइंट-साइड स्क्रिप्टिंग:**
  जावास्क्रिप्ट उपयोगकर्ता के ब्राउज़र पर चलती है, जिससे सर्वर से बार-बार संपर्क किए बिना तेज़ प्रतिक्रिया मिलती है।
- **2. बहुउपयोगी (Versatile):**
  इसका उपयोग साधारण गणनाओं से लेकर जटिल सर्वर-साइड एप्लिकेशन तक के विकास में किया जा सकता है।
- **3. इवेंट-ड्रिवन (Event-Driven):**
  यह उपयोगकर्ता की क्रियाओं जैसे क्लिक, की-प्रेस आदि पर तुरंत प्रतिक्रिया देती है।
- **4. असिंक्रोनस (Asynchronous):**
  यह सर्वर से डेटा प्राप्त करने जैसे कार्यों को यूज़र इंटरफेस को फ्रीज़ किए बिना संभाल सकती है।
- **5. समृद्ध इकोसिस्टम:**
  जावास्क्रिप्ट पर आधारित अनेक लाइब्रेरी और फ्रेमवर्क उपलब्ध हैं, जैसे **React**, **Angular**, और **Vue.js**, जो विकास प्रक्रिया को तेज़ और अधिक प्रभावी बनाते हैं।
- 

## JavaScript Placement in HTML

JavaScript code can be included in an HTML document in two primary ways: **internally or externally.**

- **Internal JavaScript:** Code is written directly within <script>...</script> tags inside the HTML file.
  - In the <head> section: Often used for functions that are called by events (like a button click) to ensure the script is loaded before the user interacts with the page element.

    - In the <body> section: Typically placed at the bottom of the <body> tag to ensure that the HTML elements are fully loaded and parsed into the DOM before the script attempts to interact with them.

- **External JavaScript:** The script is written in a separate file with a .js extension (e.g., script.js) and then linked to the HTML document using the src attribute within a <script> tag.

  ```
  <script src="path/to/your/script.js"></script>
  ```

  This method separates HTML structure from code logic, making the code cleaner, more maintainable, and allowing the script to be reused across multiple pages. External scripts can be placed in either the <head> or <body> sections, and attributes like async or defer can be used to control when they load to optimize performance.

  ### JavaScript variables

  JavaScript variables are named containers for storing data values that can be used and changed throughout a program's execution. They are declared using the keywords let, const, or the older var

## Q-2 Difference between var, let and const keywords in JavaScript

JavaScript provides three ways to declare variables: var, let, and const, but they differ in scope, hoisting behaviour, and re-assignment rules.

- **var:** Declares variables with function or global scope and allows re-declaration and updates within the same scope.
- **let:** Declares variables with block scope, allowing updates but not re-declaration within the same block.
- **const:** Declares block-scoped variables that cannot be reassigned after their initial assignment.

```
// var example
var x = 10;
var x = 20;   // Re-declaration allowed
x = 30;       // Update allowed
console.log(x); // Output: 30
// let example
let y = 10;
// let y = 20; // Re-declaration NOT allowed
y = 25;       // Update allowed
console.log(y); // Output: 25
// const example
const z = 10;
// z = 20;     // Re-assignment NOT allowed
console.log(z); // Output: 10
```

example : let x = 5;
let y = 6;
let z = x + y;

```
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, x, y, and z are variables.</p>
<p id="demo"></p>

<script>
let x = 5;
let y = 6;
let z = x + y;
document.getElementById("demo").innerHTML = "The value of z is " + z;
</script>
</body>
</html>
```

**Example using const:**

```
const x = 5;
const y = 6;
const z = x + y;
<html>
<body>
<h1>JavaScript Variables</h1>
<p>In this example, x, y, and z are variables.</p>
<p id="demo"></p>
<script>
const x = 5;
const y = 6;
const z = x + y;
document.getElementById("demo").innerHTML = "The value of z is " + z;
</script>
</body>
</html>
```

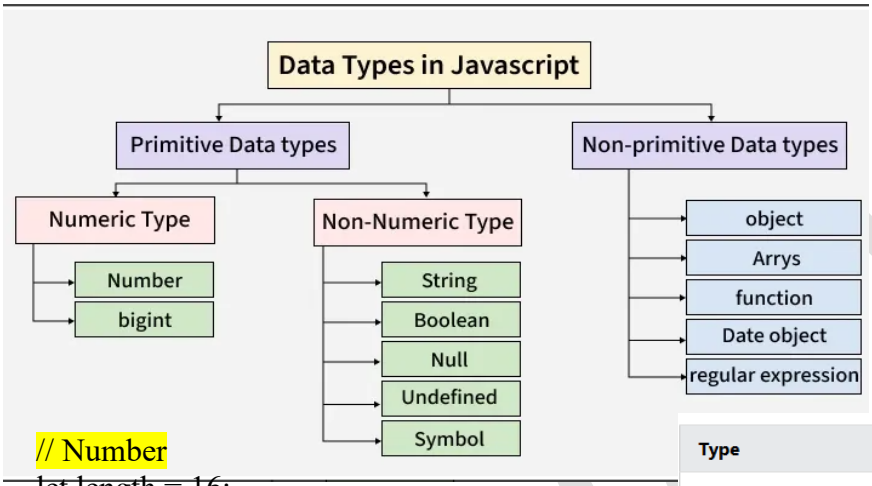
# Q-3 what is data type in java script?

JavaScript has eight data types, which are categorized into two groups: primitive (representing single, immutable values) and non-primitive or reference (representing collections of data and more complex entities).

A **JavaScript variable** can hold **8 types** of data.

**7 Primitive** Data Types and **1 Object** Data Type.

The Object data type can hold many different object types




| Type | Description |
|---|---|
| Number | A number representing a numeric value |
| Bigint | A number representing a large integer |
| String | A text of characters enclosed in quotes |
| Boolean | A data type representing true or false |
| Undefined | A variable with no assigned value |
| Null | A value representing object absence |
| Symbol | A unique primitive identifier |
| Object | A collection of key-value pairs of data |

```
// Number
let length = 16;
let weight = 7.5;

// BigInt
  let x = 1234567890123456789012345n;
  let y = BigInt(1234567890123456789012345)

//Strings
let color = "Yellow";
let lastName = "Johnson";

// Boolean
let x = true;
let y = false;

// Undefined
let x;
let y;

// Null
let x = null;
let y = null;

// Symbol
const x = Symbol();
const y = Symbol();

// Object
const person = {firstName:"John", lastName:"Doe"};

// Array Object
const cars = ["Saab", "Volvo", "BMW"];
```

```
// Date Object
const date = new Date("2022-03-25");
```

## Non-Primitive Data Type

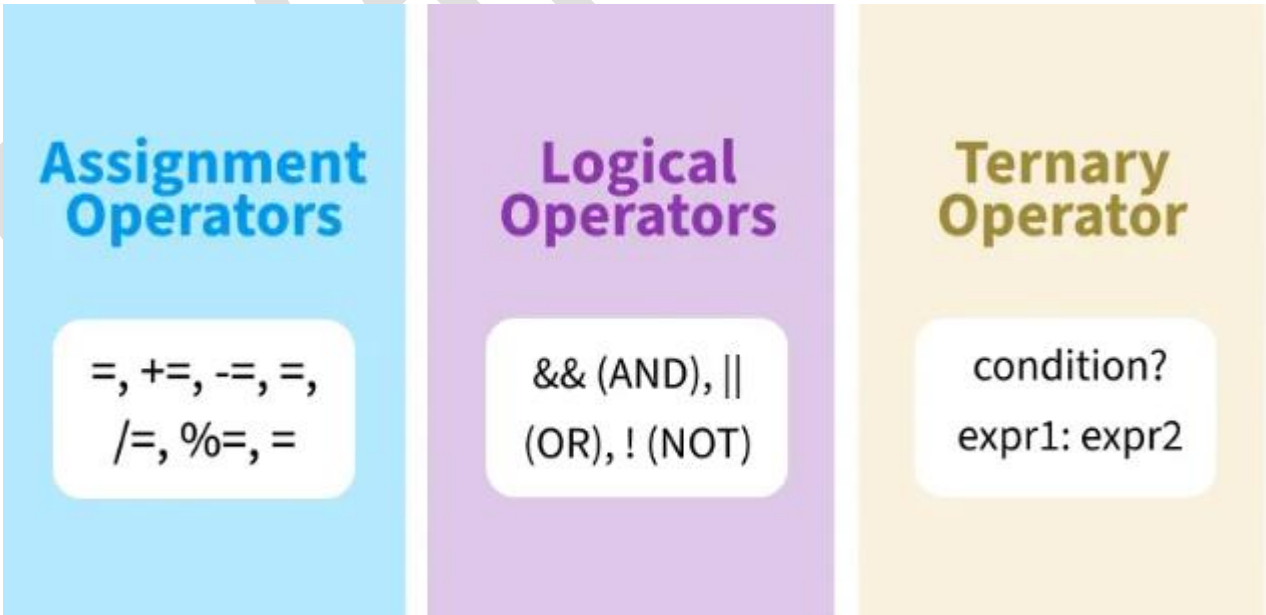
Non-primitive types, or reference types, are used to store collections of data and more complex entities. They are mutable and are assigned by reference, not by value.

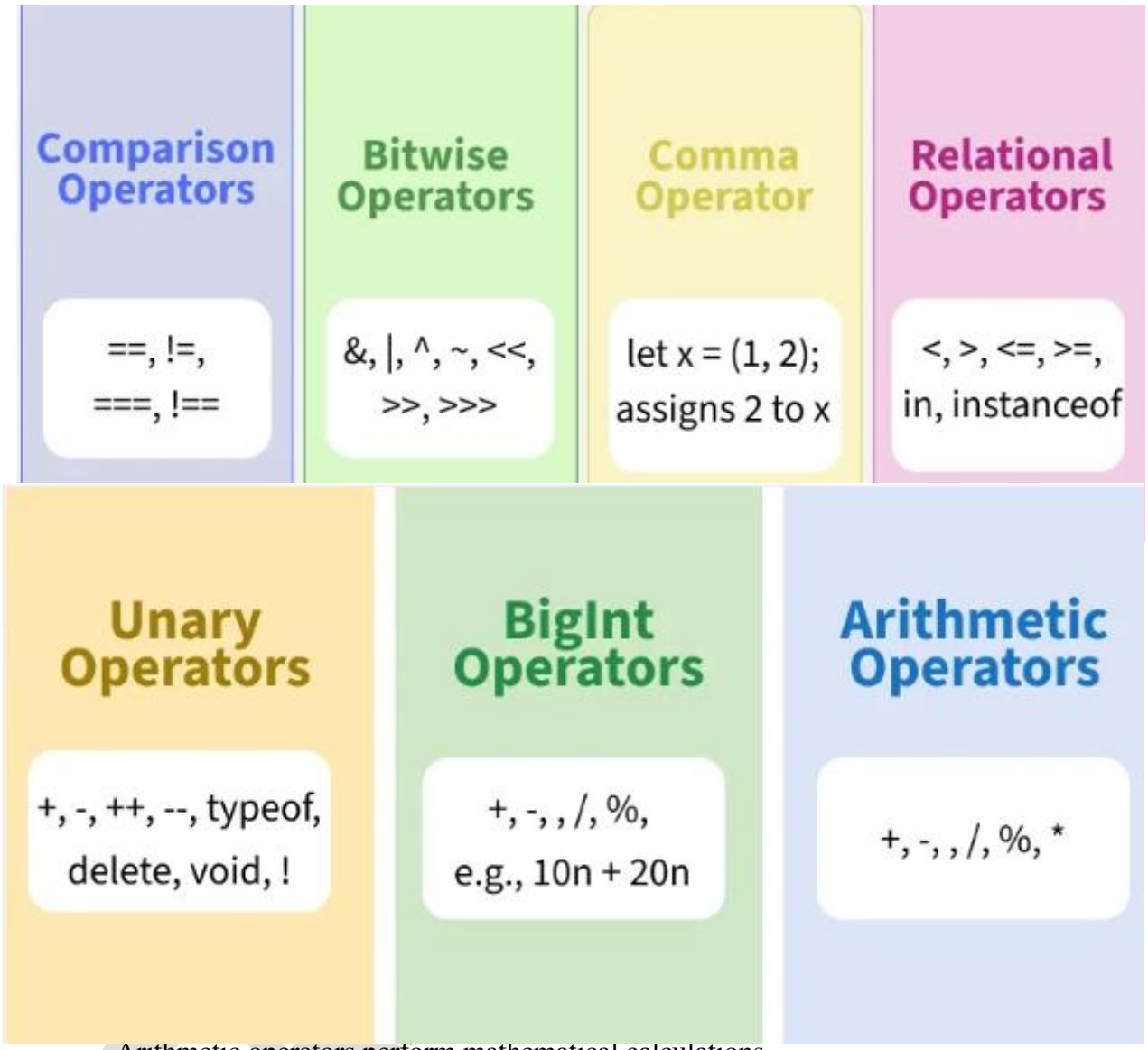
**Object**: The primary non-primitive data type. It is a collection of key-value pairs.

- Other complex data types like **Arrays** ([]), **Functions** (which are callable objects), **Dates**, and **Regular Expressions** are all types of objects.
- *Example*: let person = { name: "Alice", age: 30 };
- JavaScript Objects represent **complex data** structures and functionalities beyond the primitive data types (string, number, boolean, null, undefined, symbol, bigint).
- JavaScript objects are written with curly braces { }.
- JavaScript objects contains a collection of different **properties**.
- Object properties are written as **name:value** pairs, separated by commas.

## Explain JavaScript operators

JavaScript operators are symbols used to perform operations on variables and values, such as arithmetic, comparison, logical, and assignment operations. They are fundamental building blocks of JavaScript expressions and conditional statements.






Arithmetic operators perform mathematical calculations.

- + (Addition): Adds two values or concatenates strings.
- - (Subtraction): Subtracts one value from another.
- * (Multiplication): Multiplies values.
- / (Division): Divides one value by another.
- % (Modulus): Returns the division remainder.
- ** (Exponentiation): Raises a base to an exponent power.
- ++ (Increment): Increases a value by one.
- -- (Decrement): Decreases a value by one

Example const sum = 5 + 3; // Addition

const diff = 10 - 2; // Subtraction

const p = 4 * 2; // Multiplication

const q = 8 / 2; // Division

console.log(sum, diff, p, q);

### 2. Assignment Operators

Assignment operators assign values to variables.

- = (Assignment): Assigns the value on the right to the variable on the left.
- += (Addition assignment): x += y is the same as x = x + y.
- -= (Subtraction assignment): x -= y is the same as x = x - y.
- *= (Multiplication assignment): x *= y is the same as x = x * y.
- /= (Division assignment): x /= y is the same as x = x / y.
- %= (Remainder assignment): x %= y is the same as x = x % y.
- **= (Exponentiation assignment): x **= y is the same as x = x ** y.

Example- let n = 10;

n += 5;

n *= 2;

console.log(n);

### 3. Comparison Operators

Comparison operators compare two values and return a boolean (true or false) result.

- == (Equal to): Compares value only (performs type coercion).
- === (Strict equal to): Compares both value and type.
- != (Not equal to): Compares value only.
- !== (Strict not equal to): Compares both value and type.
- > (Greater than)
- < (Less than)
- >= (Greater than or equal to)
- <= (Less than or equal to)

Example console.log(10 > 5);

console.log(10 === "10");

### 4. Logical Operators

Logical operators are used to determine the logic between variables or values.

- && (Logical AND): Returns true if both operands are true.
- || (Logical OR): Returns true if at least one operand is true.
- ! (Logical NOT): Reverses the boolean result of an operand.

Example const a = true, b = false;

console.log(a && b); // Logical AND

console.log(a || b); // Logical OR

### 5. Other Operators

- **Conditional (Ternary) Operator (? :)**: A shorthand for an if...else statement, taking three operands (condition ? expression1 : expression2).
- **typeof Operator**: Returns a string indicating the data type of the operand.
- **Nullish Coalescing Operator (??)**: Returns the right-hand side operand if the left-hand side is null or undefined; otherwise, it returns the left-hand side value.
- **Optional Chaining (?.)**: Provides a safe way to access nested object properties without throwing an error if a property doesn't exist

Example **Ternary**

```
(1) const age = 18;
const status = age >= 18 ? "Adult" : "Minor";
console.log(status);
```

**Bitwise**

```
(2) const res = 5 & 1; // Bitwise AND
console.log(res);
```

**Comma Operator**

```
(3) let n1, n2
const res = (n1 = 1, n2 = 2, n1 + n2);
console.log(res);
```

**Unary Operators (4)**

```
let x = 5;
console.log(++x); // Pre-increment
console.log(x--);
```

**String Operators (5)**

```
const s = "Hello" + " " + "World";
console.log(s);
```

what is Conditional Statements?

**Ans : Conditional Statements** allow us to perform **different actions** for **different conditions**.

Conditional statements **run different code** depending on **true** or **false** conditions.

Q- **When to use Conditionals Statements**

- Use if to specify a **code block** to be executed, if a specified condition is true
- Use else to specify a **code block** to be executed, if the same condition is false
- Use else if to specify a **new condition** to test, if the first condition is false
- Use switch to specify many **alternative code blocks** to be executed
- Use (? :) (ternary) as a **shorthand** for if...else
- **Types of Conditional Statements**

**1. if Statement**

The if statement checks a condition written inside parentheses. If the condition evaluates to true, the code inside {} is executed; otherwise, it is skipped.

Syntax

```
if (condition) {
  // code to execute if the condition is true
}
```

```
let x = 20;

if (x % 2 === 0) {
   console.log("Even");
}

if (x % 2 !== 0) {
   console.log("Odd");
};
```

**2. if-else Statement**

The if-else statement executes one block of code if a condition is true and another block if it is false. It ensures that exactly one of the two code blocks runs.

Syntax

- ```
  if (condition) {
    // code to execute if the condition is true
  } else {
    // code to execute if the condition is false
  }
  ```
- Used when there are two possible outcomes.
- The else block runs when the if condition is not satisfied.

```
let age = 25;

if (age >= 18)

{
   console.log("Adult")
} else {
   console.log("Not an Adult")
};
```

3.**The else if Statement**

Use else if to specify a **new condition** to test, if the first condition is false.

Syntax

```
if (condition1) {
  // code to execute if condition1 is true
} else if (condition2) {
  // code to execute if the condition1 is false and condition2 is true
} else {
  // code to execute if the condition1 is false and condition2 is false
}
```

- Allows checking more than two conditions.
```

- Evaluated from top to bottom until a true condition is found.

Loading Playground...

```
const x = 0;
if (x > 0) {
   console.log("Positive.");
} else if (x < 0) {
   console.log("Negative.");
} else {
   console.log("Zero.");
};
```

**4. Using Switch Statement (JavaScript Switch Case)**

The switch statement evaluates an expression and executes the matching case block based on its value. It provides a clean and readable way to handle multiple conditions for a single variable.

- Used when one variable needs to be compared against multiple fixed values.
- Improves readability compared to long if...else if chains.

```
const marks = 85;


let Branch;
switch (true) {
   case marks >= 90:
      Branch = "Computer science engineering";
      break;
   case marks >= 80:
      Branch = "Mechanical engineering";
      break;
   case marks >= 70:
      Branch = "Chemical engineering";
      break;
   case marks >= 60:
      Branch = "Electronics and communication";
      break;
   case marks >= 50:
      Branch = "Civil engineering";
      break;
   default:
```

```
        Branch = "Bio technology";
        break;
    }
    console.log(`Student Branch name is : ${Branch}`);
```

## Loops in javascript

Loops in JavaScript allow a block of code to run multiple times as long as a given condition is satisfied. They help reduce repetition and make programs more efficient and organized.

- Loops continue running until the condition becomes false.
- They are useful for iterating over arrays, strings, and ranges of values.

In JavaScript, there are three types of Loops :

### 1. for Loop

The for loop repeats a block of code a specific number of times. It contains initialization, condition, and increment/decrement in one line.

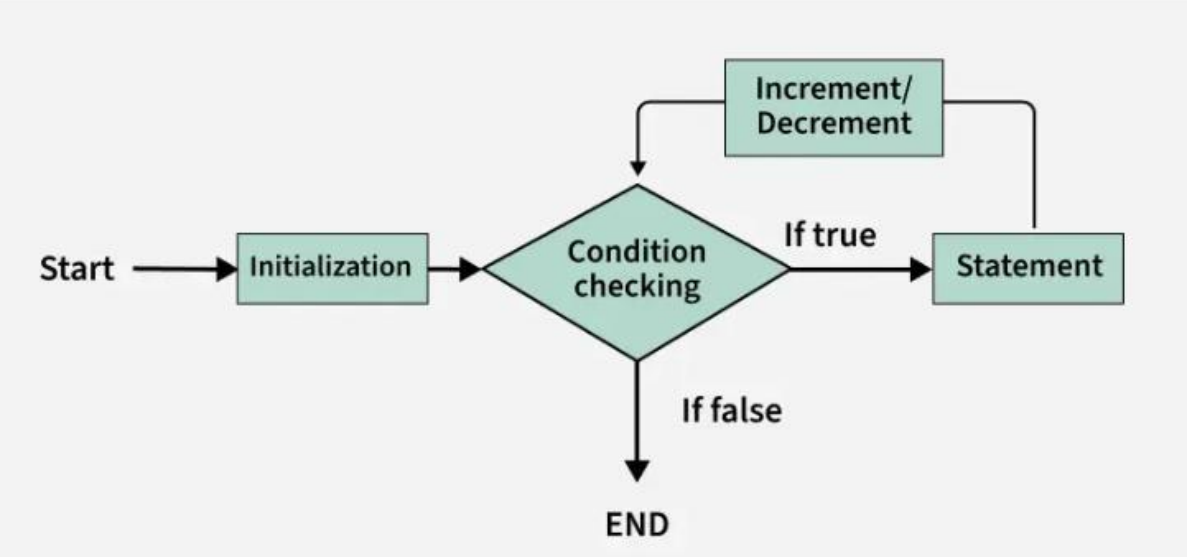
**Syntax**

```
for (initialization; condition; increment/decrement)
 {       // Code to execute}
```

**Example**: The below JavaScript program for loop runs from i = 1 to i = 10, incrementing i by 1 each time, and prints "Count:" followed by the current value of i.

```
for (let i = 1; i <= 10; i++) {
   console.log("Count:", i);
}
```

### The image below demonstrates the flow chart of a for loop:



- **Initialization condition**: Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- **Testing Condition:** It is used for testing the exit condition for a loop. It must return a boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.
- **Statement execution**: Once the condition is evaluated to true, the statements in the loop body are executed.
- **Increment/ Decrement**: It is used for updating the variable for next iteration.
- **Loop termination**:When the condition becomes false, the loop terminates marking the end of its life cycle.

## 2. while Loop

The while loop executes as long as the condition is true. It can be thought of as a repeating if statement.

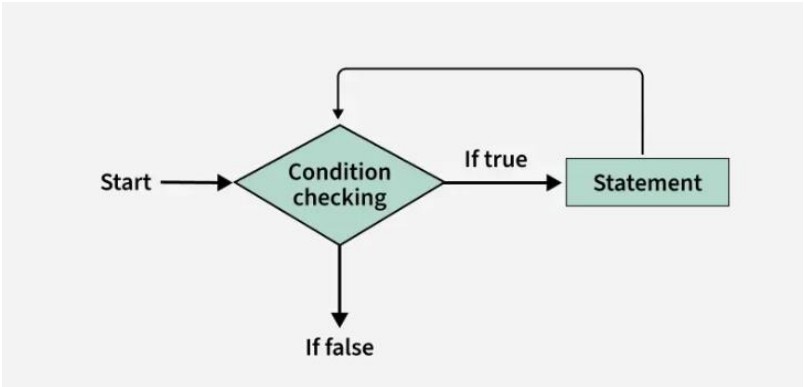
### Syntax

```
while (condition) {
   // Code to execute
}
```

**Example**: The below JavaScript program while loop prints "Number:" followed by i repeatedly while i is less than 3, incrementing i by 1 each time.

```
let i = 0;
while (i < 3) {
   console.log("Number:", i);
   i++;
}
```

The image below demonstrates the flow chart of a while loop:




- While loop starts with the checking of Boolean condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called Entry control loop
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.

- When the condition becomes false, the loop terminates which marks the end of its life cycle.

**3. do-while Loop**

The do-while loop is similar to while loop except it executes the code block at least once before checking the condition.

**Syntax**

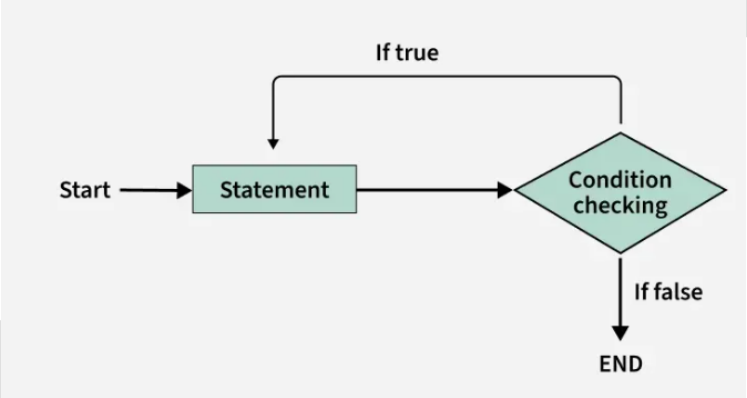
```
do {
   // Code to execute
} while (condition);
```

**Example**: The below JavaScript program do-while loop prints "Iteration:" followed by i, increments i by 1, and repeats the process while i is less than 3, ensuring the block runs at least once.

```
let i = 0;

do {
   console.log("Iteration:", i);
   i++;
} while (i < 3);
```

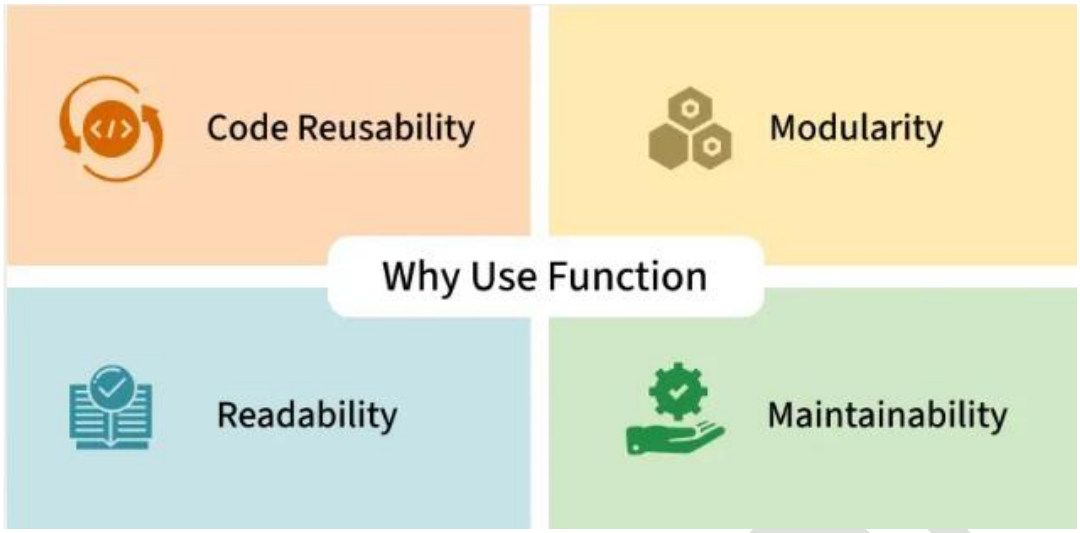
The image below demonstrates the flow chart of a do-while loop:

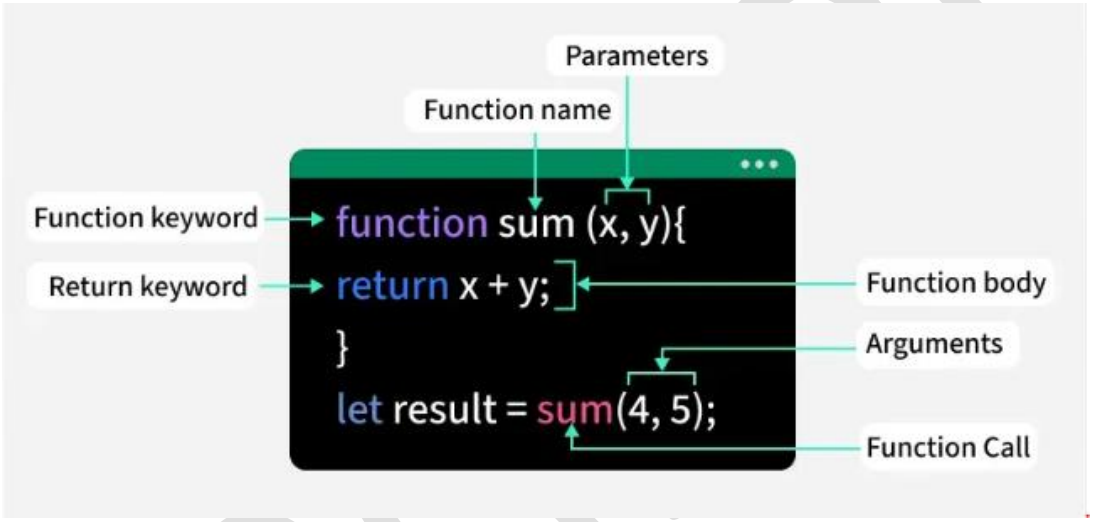



- do while loop starts with the execution of the statement. There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements a tleast once before any condition is checked, and therefore is an example of exit control loop.

## Functions in JavaScript

Functions in JavaScript are reusable blocks of code designed to perform specific tasks. They allow you to organize, reuse, and modularize code. It can take inputs, perform actions, and return outputs.




**Function Declarations :**

**Function declaration is one specific way to define a function.**

**Examples of function definitions include:**

- Function declarations
- Function expressions
- Arrow functions

```
// Function Declaration
function myFunction(x, y) {
  return x * y;
}

// Function Expression (Named)
const myFunction = function name(x, y) {
  return x * y;
};

// Function Expression (Anonymous)
```

```
const myFunction = function (x, y) {
  return x * y;
};
```

**Function Declarations:** A function declaration uses the function keyword and a function name.

**Syntax**: function *functionName*(*parameters*) {
```
  // code to be executed
}
```

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Functions</h1>
<h2>The return Statement</h2>
<p>A function that performs a calculation and returns the result:</p>
<p id="demo"></p>
  <script>
      function myFunction(a, b) {return a * b}
       let x = myFunction(4, 3);
       document.getElementById("demo").innerHTML = "The product is " + x;
</script>
</body>
</html>
```

```
          function greet(name)
       {   // 'name' is a parameter
          console.log("Hello " + name);
       }
      greet("Alice");  // "Alice" is the argument
```

- Parameter: name (placeholder inside the function).
- Argument: "Alice" (real value given at call time).

**Default Parameters**

- Default parameters are used when no argument is provided during the function call.
- If no value is passed, the function automatically uses the default value.

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

```
greet("Aman");
```

**Return Statement**

- The return statement is used to send a result back from a function.
- When return executes, the function stops running at that point.
- The returned value can be stored in a variable or used directly.

## Type of Functions

Here are all the main types of functions in JavaScript:

1. **Named Function:** A function that has its own name when declared. It's easy to reuse and debug because the name shows up in error messages or stack traces.

```
function greet() {
  return "Hello!";
}
console.log(greet());
```

2. **Anonymous Function:** A function that does not have a name. It is usually assigned to a variable or used as a callback. Since it has no name, it cannot be called directly.

```
const greet = function() {
  return "Hi there!";
};
console.log(greet());
```

3 **Function Expression :** When you assign a function (can be named or anonymous) to a variable. The function can then be used by calling that variable.

```
const add = function(a, b) {
return a + b;
};
console.log(add(2, 3));
```

**4. Arrow Function (ES6) :** A new way to write functions using the => syntax. They are shorter and do not have their own this binding, which makes them useful in some cases.

```
const square = n => n * n;
console.log(square(4));
```

**5. Immediately Invoked Function Expression (IIFE):** IIFE functions are executed immediately after their definition. They are often used to create isolated scopes.

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

**6. Callback Functions**

A callback function is passed as an argument to another function and is executed after the completion of that function.

```
function num(n, callback) {
return callback(n);
   }

const double = (n) => n * 2;
console.log(num(5, double));
```

**7. Constructor Function**

A special type of function used to create multiple objects with the same structure. It's called with the new keyword.

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}
const user = new Person("Neha", 22);
console.log(user.name);
```

**JavaScript Events :** JavaScript Events are actions or occurrences that happen in the browser. They can be triggered by various user interactions or by the browser itself.

```
<html>
<script>
   function myFun() {
      document.getElementById(
         "gfg").innerHTML = "GeeksforGeeks";
   }
</script>
<body>
   <button onclick="myFun()">Click me</button>
   <p id="gfg"></p>
</body>
</html>
```

- The onclick attribute in the <button> calls the myFun() function when clicked.
- The myFun() function updates the <p> element with id="gfg" by setting its innerHTML to "Geetam".
- Initially, the <p> is empty, and its content changes dynamically on button click.

**Event Types**

JavaScript supports a variety of event types. Common categories include:

- **Mouse Events:** click, dblclick, mousemove, mouseover, mouseout

- **Keyboard Events:** keydown, keypress, keyup
- **Form Events:** submit, change, focus, blur
- **Window Events:** load, resize, scroll

**Common JavaScript Events**

| Event Attribute | Description |
|---|---|
| onclick | Triggered when an element is clicked. |
| onmouseover | Fired when the mouse pointer moves over an element. |
| onmouseout | Occurs when the mouse pointer leaves an element. |
| onkeydown | Fired when a key is pressed down. |
| onkeyup | Fired when a key is released. |
| onchange | Triggered when the value of an input element changes. |
| onload | Occurs when a page has finished loading. |
| onsubmit | Fired when a form is submitted. |
| onfocus | Occurs when an element gets focus. |
| onblur | Fired when an element loses focus. |

```
// Mouse Event
document.addEventListener("mousemove", (e) => {
    console.log(`Mouse moved to (${e.clientX}, ${e.clientY})`);
});
// Keyboard Event
document.addEventListener("keydown", (e) => {
    console.log(`Key pressed: ${e.key}`);
});
```

- The mousemove event tracks cursor movement.

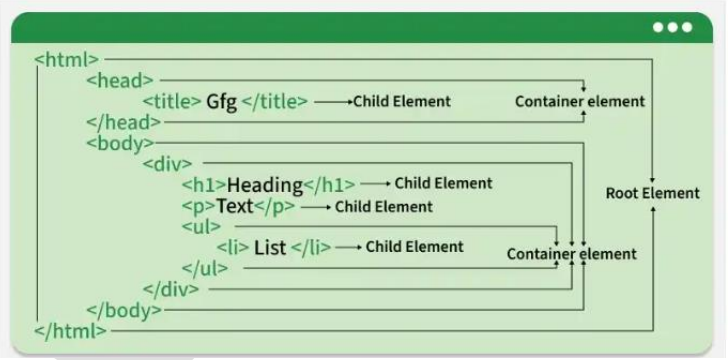
- The keydown event captures key presses.

**JavaScript Event Handlers**

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data

**DOM Manipulation in javascript** DOM manipulation in JavaScript is the process of using the Document Object Model (DOM) API to dynamically change the content,



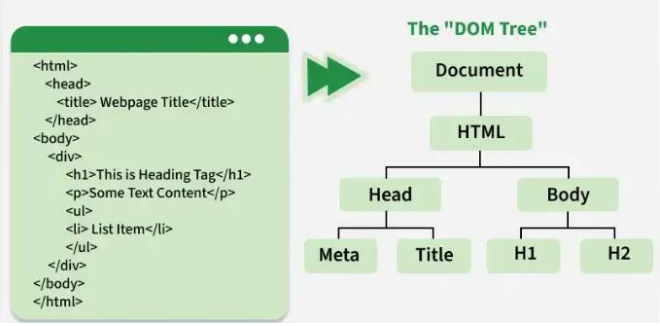

structure, and style of a webpage. This allows for interactive and dynamic web experiences without requiring a full page reload.

**Key DOM Manipulation Techniques**

To manipulate the DOM, you first need to select the element(s) you want to work with.

**1. Selecting Elements**

- document.getElementById("id"): Selects a single element by its unique ID.
- document.getElementsByClassName("class"): Returns a collection of all elements with a specified class name.
- document.getElementsByTagName("tag"): Returns a collection of all elements with a specified tag name (e.g., <p>, ).
- document.querySelector("selector"): Returns the first element that matches a specified CSS selector.



- document.querySelectorAll("selector"): Returns a NodeList of all elements that match a specified CSS selector.

**2. Modifying Content and Attributes**

- element.innerHTML = "new html": Gets or sets the HTML content inside an element, including HTML tags.
- element.textContent = "new text": Gets or sets the *textual* content of an element, ignoring any HTML tags within the new content.
- element.setAttribute("attribute", "value"): Sets the value of an element's attribute.
- element.getAttribute("attribute"): Retrieves the current value of an attribute.

**3. Changing Styles**

- element.style.propertyName = "value": Modifies individual CSS properties directly.
- element.classList.add("class"): Adds a class to an element (useful for applying complex styles defined in CSS).
- element.classList.remove("class"): Removes a class from an element.
- element.classList.toggle("class"): Toggles a class, adding or removing it as needed.

**4. Creating and Removing Elements**

- document.createElement("tag"): Creates a new HTML element node.
- parentNode.appendChild(childNode): Adds a new child node to a parent element, placing it as the last child.
- parentNode.removeChild(childNode): Removes a child node from its parent.
- element.remove(): A simpler method to remove an element from the DOM.

**5. Handling Events**

JavaScript can react to user actions (events) like clicks, keypresses, and form submissions using event listeners.

```
const button = document.querySelector("#myButton");
button.addEventListener("click", function() {
  alert("Button clicked!");
});
```

**Unit-5**

**Section A – Very Short Answer (1 Mark Each)**

*(Attempt all questions)*

1. **What is web development?**

**Ans.** Web development is the process of creating, building, and maintaining websites and web applications that run online, ranging from simple static pages to complex, dynamic web applications. It involves coding (HTML, CSS, JavaScript), server management, database configuration, and user experience design to ensure a functional, secure, and interactive digital experience.

2. **What is meant by project planning?**

Ans. Project planning is the process of defining goals, structure, and resources for a website. Project planning is the foundational phase of project management that involves defining the scope, goals, tasks,, schedules, and resources needed to achieve a specific objective. It creates a comprehensive roadmap—the project plan—detailing *what* needs to be done, *who* will do it, *when* it must be finished, and *how* risks will be handled.

3. **Define web hosting.**

**Ans : Web hosting is a service that stores website files on a server and makes them accessible on the internet** Web hosting is a service that makes websites accessible on the internet by storing website files (HTML, CSS, images, etc.) on specialized computers called servers. When users enter a domain name, the server delivers the site to their browser. It is essentially renting space on a secure, 24/7 connected server, essential for launching any online site.

4. What is a domain name?

Ans.: A domain name is the address of a website ( www.google.com).

5. What is the first step in web development?
6. What is testing in web development?
7. Name any one type of website.
8. What is deployment?
9. What is maintenance of a website?
10. What is a static website?

## Long Answer

1. **Explain the complete process of web development from planning to maintenance**.

The complete web development process follows a structured lifecycle involving several core phases Standard website development steps involve six key phases: **planning, design, development, testing, deployment, and ongoing maintenance**. Adhering to these steps helps ensure the final product is functional, user-friendly, and meets business objectives.

**1. Planning and Analysis**

This foundational phase involves defining the project's purpose, scope, and requirements. The planning phase lays the strategic groundwork for any development project. This web development step is focused on defining goals, priorities, requirements and information architecture:

- **Define business/user goals:** What purpose will this website serve? Key goals could include generating leads/sales, providing info, enabling bookings etc.
- **Target Audience**: Conduct user research to understand the needs, preferences, and behaviors of the target audience.
- **Requirement Gathering**: Document detailed functional and technical specifications, features, and integrations needed (e.g., payment systems, analytics).
- **Sitemap and Strategy**: Create a sitemap, which is a hierarchical list of all pages, and a content strategy to map out the website's structure and information flow.
- **Technology Selection**: Choose the appropriate technology stack, including programming languages, frameworks, and a hosting provider.

  Solid planning ensures your website will contain the right pages, features and messaging to support desired goals. It also helps shape realistic timelines and budget needs before the design and development begin.

**2. Design**

The design phase focuses on the website's visual aesthetics and user experience (UX/UI).

- **Wireframing**: Create low-fidelity sketches or blueprints to visualize the layout and structure of key page templates without focusing on visual details.
- **Mockups and Prototypes**: Develop high-fidelity visual designs (mockups) using color schemes, typography, and graphics, often creating interactive prototypes to simulate user flow and gather feedback.
- **Design core pages:** Homepage drafts, content page templates, graphics
- **User Experience (UX) Refinement**: Ensure ease of navigation and a smooth user journey, making sure the design is responsive and compatible across various devices.

This web design process shapes how visitors will experience and interact with all areas of your website. It leads directly into the development workflow for engineering the frontend to backend build.

**3. Development and Coding**

This is where the design is transformed into a functional website through coding.

- **Front-end Development**: Developers use languages like HTML, CSS, and JavaScript to build what users see and interact with in their web browsers.
- **Back-end Development**: The server-side logic, database structure, and application functionality are built using languages such as Python, PHP, or Ruby.
- **Content Integration**: Content (text, images, videos) is added and organized within the chosen Content Management System (CMS), such as WordPress or Drupal.

**4. Testing and Quality Assurance (QA)**

Rigorous testing is performed to identify and fix bugs and ensure a high-quality product.

- **Functional Testing**: Validate that all features, forms, links, and integrations work as intended.
- **Compatibility Testing**: Check performance across different browsers, devices, and operating systems.
- **Performance and Security Testing**: Assess loading speed, responsiveness under load, and security vulnerabilities.
- **User Acceptance Testing (UAT)**: The client or end-users test the website to ensure it meets their needs and expectations before launch.

**5. Deployment and Launch**

Once testing is complete and approved, the website is deployed to a live server and made publicly accessible.

- **Server Setup**: Configure the hosting environment and set up domain name systems (DNS).
- **Migration**: Transfer all files, databases, and content from the staging environment to the production server.
- **Final Checks**: Perform one last review on the live site to ensure everything is functioning correctly and analytics tools (like Google Analytics) are properly configured.
- Configuring SEO: Ensure metadata is optimised on all pages
- Enable security: Install firewalls, and malware protection
- Soft launch: Limited release to gather initial feedback
- Official launch: Broader release including marketing promotion
- Post-launch monitoring: Overseeing performance and monitoring for urgent issues

Carefully orchestrating both technical and marketing launch plans leads to an effective website release.

**6. Maintenance and Updates**

The process does not end at launch; ongoing maintenance is essential for sustained success.

- **Monitoring**: Continuously track the website's performance, security, and user metrics.
- **Bug Fixes and Updates**: Promptly address any new issues and install software or security updates.

- **Optimization and Enhancements**: Based on user feedback and analytics data, new features are added and existing ones optimized to improve performance and user experience over time.

**Q.2 What is wireframing? How is it different from mockups? Give examples of tools used.**

Wireframing is the process of creating a low-fidelity, skeletal blueprint of a digital interface (website or app) to define structure, layout, and information architecture. It acts as a foundational roadmap, focusing on functionality and user flow rather than aesthetics, allowing designers to map out where elements like navigation, content, and buttons go before adding design details.

**1. What is Wireframing?**

Wireframes are often referred to as the "skeleton" of a design. They are usually created early in the design process to align team members on the layout and user experience without getting distracted by colors, images, or typography**.**

**Key Characteristics of Wireframes:**

- Low-Fidelity (Lo-Fi): Usually black-and-white, gray-scale, or simple sketches.
- Placeholder Focused: Uses grey boxes, lines, and "lorem ipsum" text to represent content.
- Structure-Oriented: Focuses purely on where elements are placed and how they are structured.
- Purpose: Rapid ideation, layout exploration, and team alignment.

**. How Wireframes Differ from Mockups**

While wireframes are the skeletal plan, mockups are the detailed visual representations. Mockups build upon wireframes by adding high-fidelity design elements, such as color schemes, typography, and actual images, showing exactly what the final product will look like, but without interactivity.

| Feature | Wireframe | Mockup |
|---|---|---|
| **Fidelity** | **Low-Fidelity (Simple)** | **High-Fidelity (Detailed)** |
| **Visuals** | **Basic (Monochrome/Gray)** | **Detailed (Color, Imagery, Logos)** |
| **Purpose** | **Structure & Layout planning** | **Final look and feel, branding** |
| **Interactivity** | **Usually non-interactive** | **Non-interactive (Static)** |
| **Focus** | **How it works (User Flow)** | **What it looks like (Aesthetics)** |

**3. Examples of Tools Used**

**Wireframing tools range from quick sketching applications to comprehensive design platforms.**

- **Figma: A popular, collaborative cloud-based tool offering wireframe kits and real-time editing.**
- **Balsamiq: Famous for its "hand-drawn" style, designed specifically for low-fidelity wireframes to encourage fast iteration.**
- **MockFlow: A user-friendly tool that offers a robust editor and specialized components for early-stage planning.**
- **Miro: A digital whiteboard tool perfect for brainstorming, creating user flows, and low-fi wireframing.**
- **Adobe XD: A versatile platform for creating both wireframes and high-fidelity mockups.**
- **Wireframe.cc: A minimalist web-based tool designed specifically for fast, simple, and clean wireframing.**

## Website Structure and Navigation Planning

Effective website structure and navigation planning involve organizing content hierarchically (usually 2–8 main sections) to ensure users can find information within three clicks. Key steps include defining goals, mapping content with a sitemap, using intuitive navigation labels, and prioritizing mobile-first design for usability.

Website structure, or website architecture, is the organizational design of your site's pages. It involves categorizing content into a layout that's understandable, accessible, and predictable. Navigation should be intuitive so visitors can find what they're looking for quickly.

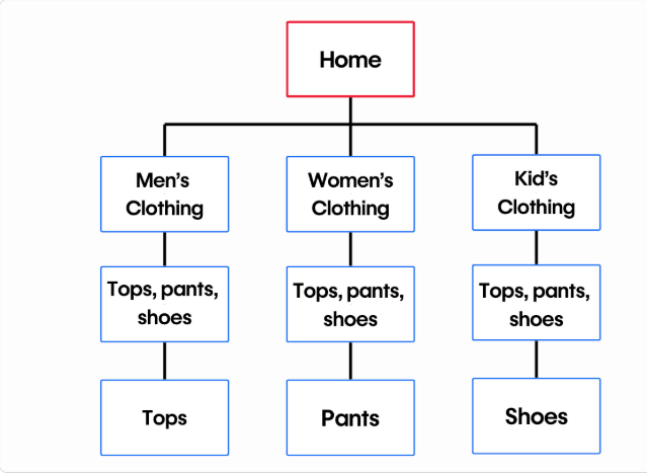
Imagine walking into a bookstore and seeing all the books piled into a corner—it's tempting to leave instead of sifting through titles. Your experience is smoother if they're organized alphabetically or by genre.

A site without a clear structure creates the same issue. Poor structure is the reason 34% of visitors leave a page, so set a clear layout to help people find what they need easily.

### Types of website structures: There are the three most common structures.

1. **Hierarchical Model:** The hierarchical structure, aka the tree model, takes a top-down approach to guide visitors from a general page, like a homepage, to more content. General pages are also called top pages, while pages with specific content are subpages or child pages.

   This model works best for content-heavy websites. The hierarchy for many ecommerce websites, for example, becomes more specific from the top page down. You'll find increasingly specific products as you move down the hierarchy. A clothing store, for example, likely has several categories, each with unique subcategories.




2. **Sequential model**

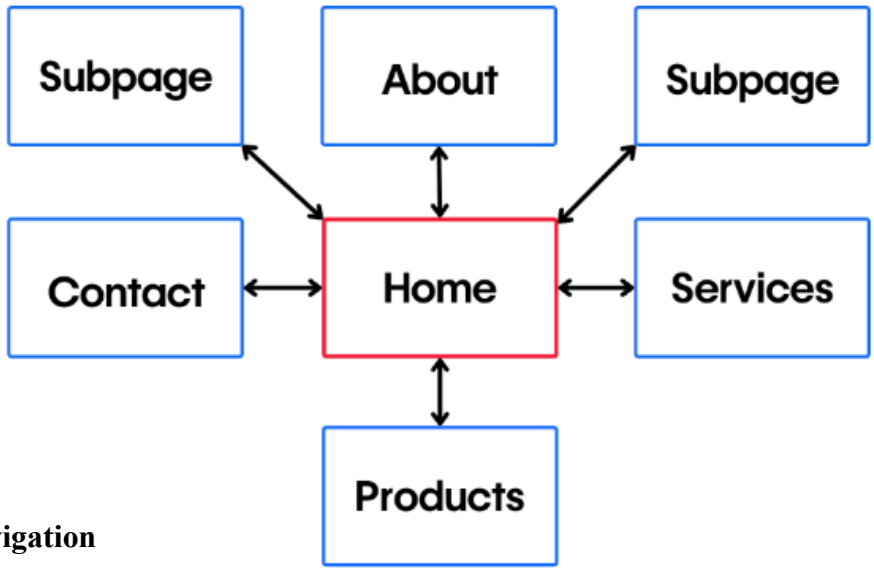
The sequential navigation structure (or linear model) places webpages in a single logical sequence, one page or step after the other. This model works well for creators who want to showcase a brand, product, or service with minimal content.

In a sequential structure, the site visitor can go directly to the next step when they've learned what they need to know from the previous one. Once they've chosen an offering from the home page they can learn more about it on a secondary page and then convert on the next. They only move on once they've found what they need.

3. **Webbed model**

The webbed model's name comes from its net-like structure. It connects the landing page or homepage to subpages through internal links. The difference is these links aren't hierarchical or nested — each internal link is available at all times. In the webbed model, site visitors can access any other page from the page they're currently on.

This model is best for smaller websites with fewer pages. A portfolio with a couple categories and a minimal navigation bar is a good example. With the navigation at the top of every page, visitors can move between information at any point.




## Planning Navigation

- **Main Menu :** Main menu is the most important navigation element and its mainly located in your website header.
- **Secondary Menu** : As the name implies this menu is secondary, which means it is for a subsection or a main section. It is generally located in the left navigation, often called as sidebar.
- **Header Navigation:** Includes essential links (About, Services, Blog) in the main navigation bar.
- **Footer Navigation:** Footer links are located in the footer of your website. They are another important navigational element, sometimes you give the most important links in your Main Menu in the Footer as well. This is to help users find the link to main pages of your website even when they scroll down to the bottom of your page.
- **Breadcrumbs:** Breadcrumbs are very important and yet, often ignored navigational element. Breadcrumbs explain to the user where they are located in the website right now, when they land on a particular webpage of your website from search engines like Google or other external websites. Breadcrumbs are extremely important web navigational element for any website as they explain to the users about the section of website they have landed.
- **Inline links , Related links & External Links** These are different links that you can have within the content area of your website. These links encourage more interaction from the users and makes the browsing experience of your website audience even better.

## Designing a multi-page website using HTML, CSS, and JavaScript:

Designing a multi-page website using HTML, CSS, and JavaScript involves creating multiple linked HTML files, using a shared CSS file for a consistent look, and incorporating JavaScript for enhanced interactivity.

**Core Concepts:**

- **HTML (Structure):** Used to define the content and structure of each page, including text, images, and links.
- **CSS (Styling): Manages the visual presentation** and layout, ensuring consistency across all pages and adapting to different screen sizes (responsive design).
- **JavaScript (Interactivity):** Adds dynamic functionality, such as interactive menus, form validations, or dynamic content loading.

**Step-by-Step Guide**

1. **Create Multiple HTML Files:**
    - Each page of your website needs its own HTML file (e.g., index.html, about.html, contact.html).
    - Use a basic HTML structure in each file.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Page Title</title>
  <link rel="stylesheet" href="styles.css"> <!-- Link to shared CSS file -->
</head>
<body>
  <!-- Page-specific content goes here -->
</body>
</html>
```

2. **Link Pages with HTML <a> Tags:** Use anchor (<a>) tags with the href attribute to link between pages. This is typically done within a navigation menu

```
<nav>
    <ul>
        <li><a href="index.html">Home</a></li>
        <li><a href="about.html">About</a></li>
        <li><a href="contact.html">Contact</a></li>
    </ul>
</nav>
```

3. **Style with a Shared CSS File:**
    1. Create a single styles.css file to manage the design of the entire website. This ensures a unified look and feel.
    2. Link this file in the <head> section of every HTML page as shown in Step 1.
    3. Use CSS for layout, colors, fonts, and responsive design (using @media queries).
4. **Add Interactivity with JavaScript:**
    4. Create a script.js file for dynamic features.
    5. Link the JavaScript file at the end of the <body> tag of each HTML file: <script src="script.js"></script>
    6. Example use cases could be for a responsive toggle menu or form validation.
5. **Test Your Website:**
    7. Open your main page (index.html) in a web browser.

8. Verify that all navigation links work correctly and that the design remains consistent across all pages.

**image optimization and accessibility**

**Image optimization and accessibility improve website speed and user experience for everyone. Key practices include resizing images to correct dimensions, using modern formats (WebP, JPEG), compressing files to under 100KB, and providing descriptive alt text for screen readers. Proper naming and lazy loading further boost both SEO and inclusivity..**

**Key Image Optimization Practices**

- **Compress and Resize: Use tools to reduce file sizes to under 100KB without sacrificing quality. Use properly scaled dimensions for mobile and desktop to reduce bandwidth.**
- **Use Proper Formats: Choose WebP for high-quality, lightweight images, JPEG for photos, and PNG for graphics/logos.**
- **Descriptive Filenames: Name images accurately, using hyphens to separate words (e.g., yellow-labrador-running.jpg instead of IMG001.jpg).**
- **Lazy Loading: Implement lazy loading so images load only as users scroll down, accelerating initial page load.**

**Image Accessibility Practices**

- **Descriptive Alt Text: Write alt text that describes the content and purpose of the image for users with visual impairments, ensuring it adds context rather than just stuffing keywords.**
- **Decorative Images: Use null alt text (alt="") for purely decorative images to help screen readers skip them.**
- **Functional Images: For images that are links or buttons, describe the functionality (e.g., "Search," "Buy Now") rather than the visual look.**
- **Contextual Captions: Use captions to provide context that enhances understanding for all users, not just those using assistive technology.**

**Benefits**

- **Enhanced SEO: Search engines understand images better with good alt text and file names, increasing chances of appearing in search results.**
- **Better Performance: Smaller files and lazy loading significantly improve page loading speeds, which reduces bounce rates.**
- **Inclusivity: Ensuring all users, regardless of ability, can understand the visual content of a website**

**version control git basics :**In software development, tracking changes, managing code versions, and collaborating smoothly are essential, and Version Control Systems (VCS) make this possible. Git is the most widely used VCS, helping developers work efficiently on personal projects or large team-based codebases.

- Tracks every change made to your project.
- Maintains multiple versions without manual file copies.
- Enables smooth collaboration across individuals and teams.

Version Control

Before starting to discuss Git, it is important to understand the concept of version control. In simple terms, version control is a system that tracks changes made to files over time. It allows developers to:

- Save and track changes: Every modification made to the codebase is recorded.
- Revert to previous versions: If something breaks or a feature doesn't work as expected, you can revert to a stable version.
- Collaborate: Multiple developers can work on the same project without overwriting each other's work.
- Branching and Merging: Developers can create branches for different features, work on them independently, and merge them back to the main codebase when ready.

## Introduction to Git

Git  GitHub is an online platform used to store and manage code.

GitHub  is a distributed version control system, meaning that it allows developers to work on their own local copies of a project, while still enabling them to push changes to a shared repository. Created by Linus Torvalds in 2005, Git has since become the standard for version control in the software development industry.

- Git manages and tracks code changes in a decentralized way.
- Every developer has a full copy of the project's complete history.
- This design makes Git fast, scalable, and resilient to server failures.

Capabilities of Git

The following are the key features of Git:

- Version Tracking: Git follows all adjustments done in one record, letting you revert to old releases without trouble.
- Collaboration: Different programmers can work on a similar task at the same time without clash.
- Branching: You have the option to create distinct branches for new attributes, bug repairs or tests.
- Distributed System: Every programmer has an entire version of the project implying that it is decentralized software.
- Log of Commits: With this feature, Git maintains an account of all commit actions (changes), which makes understanding how a project has evolved over time much easier.

Benefits of Using Git

In collaborative development, Git is a widely trusted tool that helps developers manage code changes smoothly. Since it is a distributed system, every contributor has a full copy of the project's history, enabling flexible work, even offline.

The Benefits of Git and a Distributed Version Control System:

- Distributed Nature: Each developer has the complete project history, allowing independent work without relying on a central server and enabling offline development.

- Smooth Collaboration: Branching and merging allow multiple developers to work on the same codebase without conflicts.
- Clear Version History: Every change is stored in an organized log, making it easy to track progress and troubleshoot issues.
- Easy Branching & Merging: Lightweight branches allow experimentation with new features before merging them into the main code.
- High Performance: Git handles large projects efficiently with fast operations and minimal storage overhead.

**GitHub repository** **A GitHub** repository (or "repo") is the central project hub on the GitHub platform where all of a project's files, source code, and each file's complete revision history are stored. It functions like a smart project folder in the cloud that uses the Git version control system to manage and track changes over time and enable collaboration among developers.

**Key Features and Concepts**

- **Version Control:** Repositories meticulously track every change made to files, allowing developers to revert to previous versions, see who made changes, and understand why.
- **Collaboration:** Repositories facilitate teamwork by allowing multiple people to work on the same codebase simultaneously without overwriting each other's work. Key collaboration tools include:
    - **Branches:** Developers create separate, parallel versions of the code (branches) to work on new features or bug fixes in isolation from the main project.
    - **Commits:** Changes made within a branch are saved as "commits," which are like snapshots of the project at a specific moment with a descriptive message.
    - **Pull Requests:** When changes on a branch are ready, a "pull request" is opened to propose merging them into the main codebase, allowing other team members to review, comment on, and approve the changes before they are integrated.
    - **Issues and Projects:** Repositories can include integrated tools for tracking bugs, organizing tasks with Kanban-style boards, and managing the overall project workflow.
- **Visibility:** Repositories can be public or private (and internal for enterprise users).
    - **Public repositories** are visible to everyone on the internet and are commonly used for open-source projects.
    - **Private repositories** are only accessible to you and specific people you share access with, making them ideal for proprietary or sensitive code.

**Github repositories, commits, pushing code, hosting:**

GitHub provides a platform for hosting code in repositories, where changes are tracked via commits and shared by pushing those commits from a local machine. It also offers a static site hosting service called GitHub Pages.

**Commits**

A commit is a snapshot of your project's changes at a specific point in time. Commits are initially a local operation, meaning they only exist on your local machine until they are shared. Each commit has a unique identifier and usually includes a descriptive message explaining the changes made.

- A **commit** means **saving changes** in a project.

- Each commit includes:
    - Changes made
    - Message (description)
    - Time and author

👉 Example: “Added homepage design”

✓ Commits help track progress and history of work.

**Pushing Code**

"Pushing" is the process of transferring your local commits to the remote repository hosted on GitHub. This action updates the central, online version of the project with your latest work, making it available to collaborators and backing up your code. The primary command used for this is git push origin main (or master, depending on your branch name).

- **Push** means uploading your local code to GitHub.
- It sends your commits from your computer to the online repository.

**Basic Commands:**

```
git add .
git commit -m "your message"
git push
```

👉 Use push to **save your work online and share with others**.

**Hosting**

GitHub offers a free static site hosting service called GitHub Pages. This service takes HTML, CSS, and JavaScript files directly from a repository and publishes a website at a public URL (typically your-username.github.io).

- **Hosting** means making your website **live on the internet**.
- GitHub provides free hosting using **GitHub Pages**.

**Features:**

- Free hosting
- Easy to use
- Best for static websites (HTML, CSS, JS)

👉 Example: Your project becomes live like
https://username.github.io/project-name

**General Workflow**

The standard command-line workflow for managing and hosting code is:

- **Initialize a local repository** within your project directory using git init.
- **Stage files** you want to include in the next commit using git add ..
- **Commit your changes** with a message using git commit -m "Your descriptive message".
- **Link to the remote repository** on GitHub using git remote add origin [repository URL].

- **Push the commits** to the remote repository using git push -u origin main

| Concept | Meaning |
|---|---|
| Repository | Project folder on GitHub |
| Commit | Save changes |
| Push | Upload code to GitHub |
| Hosting | Make website live |

**Hosting Gitllub Pages** **:** GitHub Pages is a free hosting service that allows you to publish static websites directly from a GitHub repository. It is commonly used for personal portfolios, project documentation, and landing pages.

**Core Requirements**

- Static Content Only: Supports only HTML, CSS, and JavaScript. It does not support server-side languages like PHP, Python, or Ruby (except via Jekyll).
- Entry Point: Your repository must contain an index.html, index.md, or README.md file at the root level.
- Visibility: For free accounts, repositories must be Public to use GitHub Pages.

**Step-by-Step Setup**

1. **Create a Repository:**
    1. User Site: Name it username.github.io (replace username with your exact GitHub username).
    2. Project Site: Name it anything you like; the URL will be username.github.io/repository-name.
2. **Upload Files:** Add your index.html and other assets to the repository via the web interface or by pushing from your local machine using Git.
3. **Enable Pages**:
    1. Navigate to your repository Settings.
    2. Select Pages from the left sidebar.
    3. Under "Build and deployment" > "Source," select Deploy from a branch.
    4. Choose your branch (usually main) and folder (/ (root)), then click Save.
4. **Visit Your Site:** Your site will be live at https://<username>.github.io/ within a few minutes.

**Netlify** **:**

o set up a custom domain on Netlify or Firebase, you must first register your domain with a third-party registrar (e.g., Namecheap, GoDaddy) and then configure the DNS settings to point to your chosen hosting provider.

**Netlify Custom Domain Setup**

You can either use Netlify DNS (managed by Netlify) or keep your current DNS provider.

- **Option 1: Using Netlify DNS (Recommended)**
    1. Add Domain: In the Netlify Dashboard, go to Site settings > Domain management, click Add custom domain, and enter your domain name.
    2. Delegate Nameservers: Netlify will provide four nameservers (e.g., dns1.p01.nsone.net). Copy these into your domain registrar's "Custom DNS" or "Nameservers" section.
    3. Automatic SSL: Once nameservers propagate, Netlify automatically provisions a Let's Encrypt SSL certificate.
- Option 2: External DNS Provider
    1. Apex Domain (example.com): Create an A record pointing to Netlify's IP: 75.2.60.5.
    2. Subdomain (www.example.com): Create a CNAME record pointing to your Netlify site URL (e.g., your-site.netlify.app).

**Firebase, custom domain setup:**

**Firebase Hosting Custom Domain Setup**

Firebase requires ownership verification through TXT records before your domain can be linked.

**Firebase**

1. **Start Wizard**: Go to the Firebase Console, select your project, navigate to Hosting, and click Add custom domain.
2. **Verify Ownership:** Firebase will provide a unique TXT record (e.g., google-site-verification=...). Add this to your domain registrar's DNS settings.
3. **Point DNS Records:**
    1. A Records: Once verified, Firebase will provide two IP addresses. Add two A records pointing your domain to these IPs.
4. **SSL Provisioning**: After DNS propagation (can take up to 24 hours), Firebase will automatically provision an SSL certificate for your domain.

**Comparison Table**

| Feature | Netlify | Firebase |
|---|---|---|
| Verification | Often instant via nameserver delegation | Requires TXT record verification |

| | | |
|---|---|---|
| DNS Management | Offers full DNS hosting (Netlify DNS) | Managed via external DNS |
| SSL | Free, automatic Let's Encrypt | Free, automatic global CDN SSL |
| Propagation | Usually 2–24 hours | Up to 24–48 hours |

## DNS overview

The Domain Name System (DNS) is the internet's "phonebook," translating human-readable domain names (e.g., example.com) into machine-readable IP addresses (e.g., 192.0.2.1). This hierarchical, decentralized system allows browsers to load internet resources without users needing to memorize complex, changing numerical addresses.

### Key Components of DNS

- **DNS Recursor:** A server, typically managed by an ISP, that receives queries from clients and acts as a middleman to find the IP address.
- **Root Nameserver:** The first step in translating human-readable names into IP addresses, directing queries to TLD servers
- **TLD Nameserver:** Manages information for top-level domains like .com or .net.
- **Authoritative Nameserver:** The final step in the query, providing the actual IP address for a requested domain.
- **Caching:** Temporarily storing DNS records locally or at the ISP level to speed up future lookups and reduce traffic.

### The DNS Lookup Process

1. **Request:** A user enters a URL in a browser.
2. **Query:** The computer checks its local cache and then queries a DNS recursor.
3. **Root Lookup:** The recursor queries a root server.
4. **TLD Lookup:** The root server directs the query to a TLD server.
5. **Authoritative Lookup:** The TLD server directs the query to the authoritative name server, which provides the IP address.
6. **Response:** The recursor returns the IP address to the browser, which then loads the website.

### Common DNS Records

**A Record:** Maps a domain to an IPv4 address.

**CNAME Record:** Points one domain name to another (alias).

**MX Record:** Directs email to a specific mail server.

**TXT Record:** Stores text information, often used for email verification.

## Security and Performance

**DNSSEC:** Adds security to DNS by enabling DNSSEC to verify the integrity of the data.

**DoT/DoH:** Encrypts DNS traffic for privacy, such as DNS over TLS (DoT).

**Performance:** Caching significantly improves speed, reducing the need for the full lookup process.

## Explain the process of hosting a website using GitHub Pages.

Hosting a website on **GitHub Pages** is a free service for publishing static content (HTML, CSS, and JavaScript) directly from a repository.

**Steps to Host Your Website**

1. **Create a New Repository**:
    - Log in to GitHub and click the **"+"** icon to select New repository.
    - **Repository Name**: To create a primary user site, name it <username>.github.io (e.g., octocat.github.io). For project sites, use any name.
    - **Visibility**: Must be set to **Public** for free accounts.
2. **Upload Your Files**:
    - Add your website files (HTML, CSS, JS) to the repository.
    - **Crucial**: Your main landing page must be named index.html and located in the root directory.
    - You can upload via the GitHub web interface (Add file -> Upload files) or by pushing from your local machine using Git.
3. **Configure GitHub Pages Settings**:
    - Navigate to your repository's **Settings** tab.
    - Select **Pages** from the left-hand sidebar.
    - Under **Build and deployment** > **Source**, ensure "Deploy from a branch" is selected.
    - Under **Branch**, select the branch you want to publish from (usually main) and the folder (usually / (root)). Click **Save**.
4. **View Your Site**:
    - GitHub will trigger an automatic build process. You can monitor this in the **Actions** tab.
    - Once complete, your site will be live at https://<username>.github.io/ (for user sites) or https://<username>.github.io/<repository-name>/ (for project sites).

**Key Considerations**

- **Static Only**: GitHub Pages does not support server-side languages like PHP, Ruby, or Python.

- **Custom Domains**: You can link a professional domain by adding it in the Pages settings and updating your registrar's DNS records.
- **Propagation Time**: It may take up to 10 minutes for your site to become accessible after the first deployment.

**What is SEO? Explain meta tags and keywords.**

Search Engine Optimization (SEO) is the practice of improving a website to increase its visibility in unpaid search results, driving more organic traffic. It involves optimizing content and technical elements, such as meta tags and keywords, to help search engines understand and rank the site higher.

**What is SEO?**

- **Purpose:** To increase website visibility on search engines like Google.
- **Method:** Involves improving on-page content, technical SEO, and off-page factors.
- **Goal:** To rank higher, resulting in more users visiting the site.

**Meta Tags Explained**
Meta tags are snippets of text added to the <head> section of a webpage's HTML code that describe the page's content to search engines. They are generally invisible to visitors but crucial for SEO.

- **Meta Title Tag:** Defines the title of the page, displayed in search engine result pages (SERPs) and at the top of the browser.
- **Meta Description:** A brief summary (about 150-160 characters) of the page's content that often appears under the title in search results, influencing click-through rates.
- **Meta Viewport:** Ensures the page is mobile-friendly, which is critical for Google's ranking.
- **Meta Robots:** Tells crawlers whether to index a page or follow links on it.

**Keywords Explained**
Keywords are specific words or phrases that users enter into search engines, which represent the topic of the webpage.

- **Usage:** They should be placed in the title, headers, and body text to match user intent.
- **Types:** Include short-tail (broad) and long-tail (specific) phrases.
- **Meta Keywords Tag:** Historically, this tag listed keywords in the HTML header. However, major search engines like Google no longer use this tag for ranking purposes.

**What is DNS? Explain custom domain setup.**

The Domain Name System (DNS) acts as the internet's phonebook, translating human-readable domain names (e.g., example.com) into numerical IP addresses (e.g., ) that computers use to identify each other. It enables users to access websites using memorable names instead of complex numbers

**What is DNS?**

DNS is a distributed database system that maps hostnames to IP addresses, ensuring that when you enter a URL, the browser can locate the correct server.

- **DNS Recursor:** The server that receives queries and searches for the IP address.
- **Root Name Server:** Directs queries to Top-Level Domain (TLD) servers (e.g., .com, .org).
- **Authoritative Name Server:** The final authority that holds the actual IP address mapping.

**Key DNS Records**

- **A Record:** Maps a domain directly to an IPv4 address.
- **CNAME Record:** Aliases one domain name to another (e.g., www.site.com to site.com).
- **MX Record:** Directs email to specific mail servers.
- **TXT Record:** Stores text information, often used for security verification.

**Custom Domain Setup**

Setting up a custom domain involves linking a purchased domain name to a specific website host, such as a blog or e-commerce platform.

1. **Buy a Domain:** Purchase a domain from a registrar (e.g., Namecheap, GoDaddy).
2. **Access DNS Management:** Log in to your registrar and find the DNS settings or zone editor.
3. **Add/Modify Records:**
    1. **A Record:** Point the root domain (@) to the web host's IP address.
    2. **CNAME Record:** Point the www subdomain to the web host's provided URL.
4. **Verify Ownership:** Use TXT records if the platform requires proof of domain ownership.

**Wait for Propagation** DNS changes can take up to 48 hours to update globally, though it is usually faster

**Key Terms**

- **DNS Provider:** Where you manage records, usually the registrar.
- **TTL (Time to Live):** Determines how long a DNS record is cached.

**Explain mobile responsiveness and testing & debugging.**

Mobile responsiveness is a design and development approach where a single website or application automatically adapts its layout, content, and interactive elements to fit the user's device. Unlike static designs, responsive sites are "reactive"—they detect the device's screen size and resolution (the "viewport") and adjust fluidly.

- **Core Technologies:**
    - Fluid Grids: Using percentage-based widths instead of fixed pixels so elements resize proportionally.

    - Flexible Images: Setting CSS properties (e.g., max-width: 100%) so media scales within its container.
    - Media Queries: CSS rules that apply different styles based on device characteristics like width, height, or orientation.
- **Key Design Elements:**
    - Breakpoints: Specific screen widths (e.g., 320px for mobile, 768px for tablets) where the layout shifts.
    - Touch Targets: Ensuring buttons and links are large enough (at least 44x44px) for fingers to tap accurately.
    - Readable Typography: Text must be legible without users needing to pinch or zoom.

**Testing & Debugging**

While often used interchangeably, testing is the process of *finding* defects, while debugging is the process of *fixing* them.

**1. Responsive Testing**

This validates that the application functions correctly and looks intended across all device types.

- Manual Testing: Testers interact with the site to catch visual or usability issues that automated tools might miss.
- Automated Testing: Tools like Selenium or Cypress run scripts to verify functionality across thousands of device combinations.
- **Testing Environments:**
    - Browser DevTools: Built-in tools like Chrome DevTools allow developers to simulate various viewports and orientations instantly.
    - Emulators & Simulators: Software that mimics the behavior of a mobile device on a computer.
    - Real Device Clouds: Platforms like BrowserStack or LambdaTest provide remote access to thousands of physical devices.

**2. Debugging Mobile Issues**

Once a bug is identified, developers use specific techniques to isolate and resolve it.

- Common Debugging Steps: Reproduce the error consistently, isolate the code responsible, and apply a fix.
- Remote Debugging: Connecting a physical smartphone to a computer via USB to inspect live code using the computer's Chrome or Safari developer tools.
- **Key Debugging Tools:**
    - Breakpoints: Pausing code execution at a specific line to examine current variables and program flow.

- Console Logs: Adding statements to print messages or data values to a log for tracking behavior.
- Network Throttling: Simulating slow 3G or 4G connections to find performance bottlenecks.