Converted Document

Unit 1

What is the Internet?

The Internet is a worldwide telecommunications system that provides connectivity for millions of other, smaller networks; therefore, the Internet is often referred to as a network of networks. It allows computer users to communicate with each other across distance and computer platforms.

The Internet began in 1969 as the U.S. Department of Defense's Advanced Research Project Agency (ARPA) to provide immediate communication within the Department in case of war. Computers were then installed at U.S. universities with defense related projects. As scholars began to go online, this network changed from military use to scientific use. As ARPAnet grew, administration of the system became distributed to a number of organizations, including the National Science Foundation (NSF). This shift of responsibility began the transformation of the science oriented ARPAnet into the commercially minded and funded Internet used by millions today.

The Internet acts as a pipeline to transport electronic messages from one network to another network. At the heart of most networks is a server, a fast computer with large amounts of memory and storage space. The server controls the communication of information between the devices attached to a network, such as computers, printers, or other servers.

An Internet Service Provider (ISP) allows the user access to the Internet through their server. Many teachers use a connection through a local university as their ISP because it is free. Other ISPs, such as America Online, telephone companies, or cable companies provide Internet access for their members.

You can connect to the Internet through telephone lines, cable modems, cellphones and other mobile devices.

What makes up the World Wide Web?

The Internet is often confused with the World Wide Web. The misperception is that these two terms are synonymous. The Internet is the collection of the many different systems and protocols. The World Wide Web, developed in 1989, is actually one of those different protocols. As the name implies, it allows resources to be linked with great ease in an almost seamless fashion.

The World Wide Web contains a vast collection of linked multimedia pages that is ever-changing. However, there are several basic components of the Web that allow users to communicate with each other. Below you will find selected components and their descriptions.

TCP/IP protocols

In order for a computer to communicate on the Internet, a set of rules or protocols computers must follow to exchange messages was developed. The two most important protocols allowing computers to transmit data on the Internet are Transmission Control Protocol (TCP) and Internet Protocol (IP). With these protocols, virtually all computers can communicate with each other. For instance, if a user is running Windows on a PC, he or she can communicate with iPhones.

Domain name system

An Internet address has four fields with numbers that are separated by periods or dots. This type of address is known as an IP address. Rather than have the user remember long strings of numbers, the Domain Name System (DNS) was developed to translate the numerical addresses into words. For example, the address fcit.usf.edu is really 131.247.120.10.

URLs

Addresses for web sites are called URLs (Uniform Resource Locators). Most of them begin with http (HyperText Transfer Protocol), followed by a colon and two slashes. For example, the URL for the Florida Center for Instructional Technology is .

Some of the URL addresses include a directory path and a file name. Consequently, the addresses can become quite long. For example, the URL of a web page may be:

. In this example, "default.htm" is the name of the file which is in a directory named "holocaust" on the FCIT server at the University of South Florida.

Top-level domain

Each part of a domain name contains certain information. The first field is the host name, identifying a single computer or organization. The last field is the top-level domain, describing the type of organization and occasionally country of origin associated with the address.

Top-level domain names include:

Domain name country codes include, but are not limited to:

Why do I need a browser?

Once you have an account with an Internet service provider, you can access the Web through a browser, such as Safari or Microsoft Internet Explorer. The browser is the application responsible for allowing a user's computer to read and display web documents.

Hypertext Markup Language (HTML) is the language used to write web pages. A browser takes the HTML and translates it into the content you see on the screen. You will note your cursor turns into a pointing finger

over some images or text on the page. This indicates a link to additional information and it can be either a link to additional web pages, email, newsgroups, audio, video, or any number of other exciting files.

Types of Internet Connection

An internet connection is a means by which individual devices or local networks are linked to the global internet, allowing them to communicate and exchange data. There are many connections that can be used for internet access. All the connections have their own speed range that can be used for different purposes like for home, or for personal use. In this article, we will discuss different types of internet connections.

What is the Internet?

The Internet is a global network of computers connected. It allows people all over the world to communicate, share information, and access a huge amount of data. You can use the Internet to send emails, browse websites, watch videos, play games, and much more. It?s like a huge library and playground that you can access from your computer, phone, or other devices anytime and anywhere.

Types of Internet Connection

1. Dial-Up Connection

A is established between your computer and the ISP server using a modem. A dial-up Connection is a cheap and traditional connection that is not preferred these days as this type of connection is very slow.

To access the internet connection in the dial-up connection we need to dial a phone number on the computer and that?s why it requires a telephone connection. It requires a modem to set up a dial-up connection, which works as interference between your computer and the telephone line. In this connection, we can use either an internet connection or a telephone at a time.

Dial Up Connection

2. Broadband Connection

Broadband refers to high-speed internet access that is faster than traditional dial-up access. It is provided through either cable or telephone composition. It does not require any telephone connection that?s why here we can use telephone and internet connection simultaneously. In this connection, more than one person can access the internet connection simultaneously.

It is a wide bandwidth data transmission that transports several signals and traffic types. In this connection, the medium used is , , radio, or .

Broadband-Connection

3. DSL (Digital Subscriber Line)

DSL stands for . It provides an internet connection through the telephone line(network). DSL is a form of broadband communication that is always on, there is no need to dial a phone number to connect. DSL connection uses a router to transport data and the speed of this connection range between 128k to 8Mbps depending on the service offered. A DSL connection can translate data at 5 million bytes per second, or 5mbps.

DSL service can be delivered simultaneously with wired telephone service on the same telephone line due to high-frequency bands for data.

DSL

4. Cable

It is a form of broadband access cable that can provide extremely fast access to the internet. The speed of this connection varies which can be different for uploading data transmission or downloading.

It uses a cable modem to provide an internet connection and operates over cable TV lines. The speed of cable connection ranges from 512k to 20Mbps.h

Cable

5. Satellite Connection

This type of connection is provided mainly in rural areas where a broadband connection is not yet offered. It accesses the internet via a satellite that is in Earth?s orbit.

The signal travels from a long distance that is from earth to satellite and back again which provides a delayed connection. Satellite connection speeds range from 512k to 2.0Mbps.

Satellite Connection

6. Wireless Connection

As the name suggestsdoes not use telephone lines or cables to connect to the internet. The wireless connection uses a radio frequency band to connect to the internet. It is also an always-on connection and this connection can be accessed from anywhere and speed may vary for different locations. It ranges from 5Mbps to 20Mbps.

Wireless Connection

7. Cellular

provides wireless Internet access through cell phones. Speed may vary depending on the service provider. The most common are 3G and 4G which means from 3rd generation and 4th generation respectively. The speed of the 3G cellular network is around 2.0Mbps and the 4G cellular network is around 21Mbps the goal

of the 4G network is to achieve peak mobile speeds of 100Mbps but the current speed of the 4G network is about 21Mbps.

Cellular

8. ISDN (Integrated Service Digital Network)

ISDN stands for and it is a telephone network system, but it also provides access to networks that transmits both voice and data over a digital line. It provides a packet-switched connection for data in increments of 64 kilobit/s.

ISDN connection provides better speeds and higher quality than traditional connections. It provided a maximum of 128kbit/s bandwidth in both upstream and downstream directions.

ISDN

Components Required For Internet Connecton

Modem: A device that modulates and demodulates signals for and digital data transmitted over a telephone line or cable system.

Router: A device that routes data from a local network to the internet and vice versa, often includes capabilities.

ISP (Internet Service Provider): The company that provides internet access to customers.

Establishing connectivity on the internet

Establishing connectivity on the internet involves enabling communication between devices (such as computers, servers, routers, and other networked devices) across the global network of networks known as the internet. This process involves several key components and protocols working together to facilitate data transmission. Let?s explore the steps involved in establishing connectivity on the internet in detail:

1. Network Infrastructure:

The internet is a massive network infrastructure comprised of interconnected networks, including local area networks (LANs), wide area networks (WANs), and autonomous systems (ASes) operated by internet service providers (ISPs).

Network infrastructure components such as routers, switches, modems, and cables form the physical and logical pathways through which data travels across the internet.

2. IP Addressing:

Every device connected to the internet is assigned a unique numerical label called an IP (Internet Protocol) address, which serves as its identifier on the network.

IP addresses are either IPv4 (e.g., 192.0.2.1) or IPv6 (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334) and are used for routing data packets between devices.

3. Domain Name System (DNS):

The Domain Name System translates human-readable domain names (e.g., example.com) into IP addresses that computers use to locate each other on the internet.

DNS servers maintain databases of domain name records and facilitate the resolution of domain names to IP addresses.

4. Internet Protocols:

Internet protocols, such as TCP/IP (Transmission Control Protocol/Internet Protocol), govern the rules and procedures for transmitting data packets across the internet.

TCP/IP provides reliable, connection-oriented communication between devices, while other protocols like UDP (User Datagram Protocol) offer connectionless, unreliable communication.

5. Routing and Packet Switching:

Routing protocols determine the best path for data packets to travel from the source device to the destination device across multiple networks.

Packet switching technology breaks data into smaller packets and routes them individually through the most efficient paths to their destination, where they are reassembled.

6. Internet Service Providers (ISPs):

ISPs are organizations that provide internet connectivity services to individuals, businesses, and other entities.

ISPs maintain network infrastructure, including servers, routers, and cables, to facilitate data transmission between their customers and the rest of the internet.

7. Establishing Connections:

To establish connectivity on the internet, devices must establish connections with each other using protocols such as TCP/IP.

Devices initiate communication by sending packets to the destination device?s IP address using the appropriate protocol and port number.

Handshake protocols, such as the TCP three-way handshake, are used to establish and manage connections between devices.

8. Security and Encryption:

Security measures, such as firewalls, encryption protocols (e.g., SSL/TLS), and virtual private networks (VPNs), are implemented to protect data and ensure privacy during transmission over the internet.

Secure protocols and cryptographic techniques help prevent unauthorized access, data interception, and tampering.

By integrating these components and protocols, connectivity on the internet is established, enabling seamless communication and data exchange between devices worldwide. Continuous advancements in networking technologies and protocols further enhance the efficiency, reliability, and security of internet connectivity.

HTML: File Creation:

To create an HTML file, you can follow these steps:

Open a text editor: You can use any text editor, like Notepad (Windows), TextEdit (Mac), or a code editor like Visual Studio Code, Sublime Text, or Atom.

Write the HTML code: Below is a simple HTML structure.

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

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

Save the file:

In your text editor, click File > Save As.

Save the file with the .html extension, e.g., index.html.

Make sure to choose All Files in the "Save as type" field (if needed) to ensure it's saved as an HTML file and not as a .txt file.

Open the file in a browser:

Double-click on the saved HTML file, and it should open in your default web browser.

Client-Server Model

The Client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and delivers the data packets requested back to the client. Clients do not share any of their resources. Examples of the Client-Server Model are Email, World Wide Web, etc.

How Does the Client-Server Model Work?

In this article, we are going to take a dive into the Client-Server model and have a look at how the Internet works via, web browsers. This article will help us have a solid WEB foundation and help us easily work with.

Client: When we say the word Client, it means to talk of a person or an organization using a particular service. Similarly in the digital world, a Client is a computer (Host) i.e. capable of receiving information or using a particular service from the service providers (Servers).

Servers: Similarly, when we talk about the word Servers, It means a person or medium that serves something. Similarly in this digital world, a Server is a remote computer that provides information (data) or access to particular services.

So, it is the Client requesting something and the Server serving it as long as it is in the database.

Client Server Model

How the Browser Interacts With the Servers?

There are a few steps to follow to interacts with the servers of a client.

User enters the URL(Uniform Resource Locator) of the website or file. The Browser then requests the Server.

DNS Server lookup for the address of the WEB Server.

The DNS Server responds with the IP address of the WEB Server.

The Browser sends over an HTTP/HTTPS request to the WEB Server?s IP (provided by the DNS server).

The Server sends over the necessary files for the website.

The Browser then renders the files and the website is displayed. This rendering is done with the help of DOM (Document Object Model) interpreter, CSS interpreter, and JS Engine collectively known as the JIT or (Just in Time) Compilers.

Client Server Request and Response

Advantages of Client-Server Model

Centralized system with all data in a single place.

Cost efficient requires less maintenance cost and Data recovery is possible.

The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server Model

Clients are prone to viruses, Trojans, and worms if present in the Server or uploaded into the Server.

Servers are prone to attacks.

Data packets may be spoofed or modified during transmission.

Phishing or capturing login credentials or other useful information of the user are common and attacks are common.

Difference between Web Browser and Web Server

For International Network communication, we require a web browser and web servers. Web browsers and servers play an important role in establishing the connection. The client sends requests for web documents or services. The message that goes from the web browser to the web server is known as an HTTP request. When the web server receives the request, it searches its stores to find the appropriate page. If the web server can locate the page, it parcels up to the HTML contained within (using some transport layer protocol), addresses these parcels to the browser (using HTTP), and transmits them back across the network. If the web server is unable to find the requested page, it sends a page containing an error message (i.e. Error 404 ? page not found) and it parcels up to the dispatches that page to the browser. This message received by the web browser by the server is called the HTTP response. The main differences between the Web browser and web servers are:

Web Server

Definition: A web server is a system (software or hardware) that stores web pages and serves them to clients (browsers) over the internet using the HTTP or HTTPS protocols.

Function:

Receives requests from a client (usually a web browser).

Processes the request and sends the requested web pages or files back to the client.

Examples of Web Servers:

Apache: Open-source web server widely used on Linux.

Nginx: A lightweight, high-performance web server.

Microsoft IIS: A web server developed by Microsoft for Windows.

Request-Response Model:

The client sends an HTTP request (e.g., asking for index.html).

The web server processes the request and sends an HTTP response with the requested content.

Web Client

Definition: A web client is any device or software that makes requests to the web server to access web pages or data. The most common web client is a web browser.

Examples:

Browsers: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, Opera.

Function:

Sends HTTP requests to a web server.

Displays the response from the server (usually a webpage) in the browser window.

Handles resources like images, videos, and JavaScript.

Introduction to HTML

HTML: HyperText Markup Language is the standard language for creating and structuring web content. It is used to define the structure of a webpage using a system of nested elements.

Purpose: HTML is designed to create the skeleton of a webpage, which includes headings, paragraphs, links, images, tables, forms, and more.

HTML Elements: Each HTML element is wrapped in tags, which are typically made up of an opening tag and a closing tag. For example:

This is a paragraph

Common Tags:

<html>, <head>, <body>, <title>, <h1>, , <a>, etc.

5. HTML Tags

Definition: Tags are the core building blocks of an HTML document. They define the structure and content of the webpage.

Common Tags:

Converted Document

<html>: Defines the document as HTML.

<head>: Contains metadata about the document, like title and linked files (CSS, JavaScript).

<title>: Specifies the title of the webpage (appears in the browser?s title bar).

<body>: Contains the content of the page (headings, paragraphs, images, etc.).

<h1>, <h2>, <h3>, etc.: Heading tags for defining headings (h1 is the highest, h6 the lowest).

: Paragraph tag used to define text paragraphs.

<a>: Anchor tag used for creating hyperlinks.

Self-closing Tags:

<img>: Defines an image.

: Creates a line break.

<hr>: Creates a horizontal rule (line).

Empty Tags: Tags that do not have a closing tag.

6. Structure of HTML Programs

Basic HTML Document: Every HTML document follows a basic structure with a <!DOCTYPE>, <html>, <head>, and <body> tags.

Detailed Structure:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Title of the Page</title>

</head>

<body>

<h1>This is a heading</h1>

This is a paragraph of text.

</body>

</html>

Explanation:

<!DOCTYPE html>: Tells the browser that the document is HTML5.

<html>: The root element.

<head>: Contains metadata like title, links to external CSS files, or scripts.

<body>: Contains the content of the webpage, such as text, images, links, and other elements.

7. Heading, Titles, and Footers

Headings: HTML headings define the importance of different sections of the page. There are six levels:

<h1>: Most important heading (largest).

<h2>, <h3>, etc., up to <h6>.

Titles: The <title> tag defines the title of the webpage, which appears in the browser?s title bar or tab.

<title>Page Title</title>

Footers: A footer typically contains information like copyright notice, contact info, or terms and conditions.

<footer>© 2025 My Website</footer>

8. Text Formatting

HTML provides several tags to format text:

Bold: <b> or

Bold Text

Italic: <i> or

Italic Text

Underline: <u>

<u>Underlined Text</u>

Strikethrough: <strike> or <del>

~~Strikethrough Text~~

9. Text Styles

Font Size: You can use the style attribute or CSS to change the font size:

Large Text

Font Family:

Text with Arial Font

Text Color:

Red Text

Text Alignment:

Centered Text

10. Text Effects

Text Shadow: Add shadows to text using CSS:

Text with Shadow

Letter Spacing:

Text with Spaced Letters

11. Color and Backgrounds

Text Color:

Blue Text

Background Color:

<body style="background-color: lightblue;">

Page with Background Color

</body>

Background Image:

<body style="background-image: url('background.jpg');">

12. Lists

Unordered List (bullets):

Item 1

Item 2

Ordered List (numbers):

First item

Second item

Definition List:

<dl>

<dt>Term 1</dt>

<dd>Description of Term 1</dd>

</dl>

13. Adding Graphics

Images: The <img> tag is used to embed images in HTML.

<img src="image.jpg" alt="Description of the image" width="500" height="600">

Attributes:

src: Path to the image file.

alt: Description of the image (shown if the image doesn't load).

width, height: Resize the image.

14. Tables

Tables are defined using <table>, <tr> (table row), <th> (table header), and <td> (table data).

<table>

<tr>

<th>Header 1</th>

<th>Header 2</th>

</tr>

<tr>

<td>Data 1</td>

<td>Data 2</td>

</tr>

</table>

15. Linking Documents

Anchor Tag (<a>): Used to create hyperlinks to other pages or websites.

Visit Example

Attributes:

href: URL of the destination.

target="_blank": Opens the link in a new tab.

16. Frames

Frames: The <iframe> tag is used to embed another HTML document within the current page.

<iframe src="page.html" width="500" height="300"></iframe>

What Is HTML?

HTML, which stands for Hypertext Markup Language, is the language used to describe structured documents and the language used to create web pages on the internet.

You can use any text editor to write the HTML code, such as Notepad (PC) or TextEdit (Mac).

Save the file with the .html extension, and open it using a web browser of your choice.

What Is an HTML Tag?

HTML tags are the keywords on a web page that define how your web browser must format and display your web page.

Almost all tags contain two parts, an opening, and a closing tag. For example, <html> is the opening tag and </html> is the closing tag. Note that the closing tag has the same text as the opening tag, but has an additional forward-slash (/) character.

There are a total of 100 HTML tags. We will divide them into categories and discuss the important ones in this article.

Basic HTML Tags

Head Tag

The head tag <head> contains all the elements describing the document.

Title Tag

The title tag <title> specifies the HTML page title, which is shown in the browser?s title bar.

Body Tag

The body tag <body> is where you insert your web page?s content.

Paragraph Tag

A paragraph tag <p> is used to define a paragraph on a web page.

Heading Tag

The HTML heading tag is used to define the heading of the HTML document. The <h1> tag defines the most important tag, and <h6> defines the least.

Let?s practice using these tags and create a web page with them:

Formatting Tags

Emphasis tag

The HTML tag is used to emphasize the particular text in a paragraph.

Bold Tag

The <b> tag is used to make the text bold.

Italic Tag

The <i> tag is used to make the text italic.

Underline Tag

The <u> tag is used to set the text underline.

Link Tag

The <a> tag links one page to another page. The href attribute is used to define

List Tag

The <li> tag is used if you want to enter the contents in the listed order. There are two types of lists.

Ordered list

Unordered list

Image Tag

The <img> tag is used to in an HTML document. You need to specify the source of the image inside the tag.

The center tag will center your content.

The Table Tag

The <table> tag is used to in the HTML document.

The table row (<tr>) tag is used to make the rows in the table, and the table data (<td>) tag is used to enter the data in the table.

The style (<style>) tag is used to add methods to the content by typing the code in the HTML file itself.

How to Check Your Website?s HTML Tags?

Checking your website's HTML tags is crucial for ensuring proper structure, semantics, and accessibility.

Here are some methods to do so:

View Page Source: Right-click on a webpage and select "View Page Source" or "View Source" from the browser menu. This will display the webpage's HTML code in a new tab or window. Scan through the code to inspect the HTML tags used.

Developer Tools: Most modern web browsers have built-in developer tools that allow you to inspect HTML elements. You can access these tools by pressing F12 or right-clicking on a webpage and selecting "Inspect" or "Inspect Element." Developer tools provide a comprehensive view of the , including elements' hierarchy and associated styles and attributes.

Online Validators: Various online tools and validators are available that can analyze your website's HTML code for errors, warnings, and best practices compliance. These validators can identify missing tags, incorrect nesting, deprecated elements, and other issues affecting your website's performance and .

Accessibility Checkers: Accessibility checkers help ensure that your HTML tags are semantically correct and comply with accessibility standards such as WCAG (Web Content Accessibility Guidelines). These tools can flag potential accessibility issues related to HTML structure, form labels, alt text for images, and more.

Regularly checking your website's HTML tags, you can maintain a well-structured, accessible, and search engine-friendly website.

How Do Web Pages Read HTML Tags?

Web browsers render and display web pages based on the HTML tags in their code. Here's how web pages read HTML tags:

Parsing: When a web browser loads a webpage, it starts by parsing the HTML code. Parsing analyzes the HTML document and creates a structured representation of its contents, known as the Document Object Model (DOM).

DOM Tree: The DOM tree represents the hierarchical structure of HTML elements, where each element is represented as a node with properties and attributes. The browser constructs the DOM tree by interpreting the HTML tags and their relationships, such as parent-child and sibling elements.

Rendering: Once the DOM tree is constructed, the browser uses it to render the webpage on the screen. Rendering involves applying CSS styles, calculating layout, and painting pixels based on the content and properties of HTML elements.

Event Handling: Web browsers also handle user interactions and dynamic content updates based on JavaScript code associated with HTML elements. Event listeners attached to HTML tags respond to user

actions such as clicks, mouse movements, and keyboard inputs.

In summary, web pages read HTML tags by parsing them into a structured DOM tree, which is then used to render the visual representation of the webpage and handle user interactions.

HTML Tags and Attributes Difference

HTML tags and attributes are essential components of HTML elements, but they serve different purposes:

HTML Tags: HTML tags are used to define the structure and content of an HTML document. They enclose content and determine the type of element represented. For example, <p> tags indicate a paragraph element, tags define a division or container, and <a> tags create hyperlinks.

HTML Attributes: HTML attributes provide additional information or properties to HTML elements. They modify the behavior or appearance of elements and are specified within the opening tag of an element. Attributes can define characteristics such as the source of an image (src), the URL of a hyperlink (href), the style of an element (style), or the alternative text for an image (alt).

While HTML tags define the type and structure of elements, attributes provide specific details and configurations to customize their behavior or appearance. Both tags and attributes work together to create meaningful and interactive web content.

Here's a comparison table highlighting the differences between HTML tags and attributes:

The Most Common HTML Tags

Here's a table listing some of the most common HTML tags, along with their descriptions:

These tags are fundamental building blocks for creating structured and interactive web pages.

JavaScript in Web Pages

What is JavaScript?

JavaScript is a high-level, interpreted scripting language used primarily for creating dynamic and interactive content in web pages. Unlike HTML and CSS, which define the structure and style of a webpage, JavaScript allows developers to add interactivity, control browser behavior, and dynamically update content.

How JavaScript Works in a Web Page:

Client-Side Language: JavaScript runs on the client-side (in the browser), meaning it doesn?t require constant server communication.

Integration into HTML: JavaScript can be added directly in an HTML document using the <script> tag or through external files.

Example of Including JavaScript in HTML:

Inline JavaScript: Directly inside an HTML tag event handler like onclick.

html

Copy

```
<button onclick="alert('Hello!')">Click Me</button>
```

Internal JavaScript: Inside a <script> tag in the HTML file.

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<h1>Welcome!</h1>
<script>
console.log("JavaScript works!");
</script>
</body>
</html>
```

External JavaScript: Linking an external .js file.

```
<script src="script.js"></script>
```

2. Advantages of JavaScript

Interactivity: JavaScript is widely used to add interactivity to websites, such as form validation, image sliders, and interactive maps.

Asynchronous Communication: JavaScript, combined with AJAX, allows for asynchronous communication, meaning data can be sent and received without refreshing the page (e.g., loading new content).

Cross-Browser Compatibility: JavaScript is supported by all modern web browsers, making it highly versatile.

Lightweight: It runs on the client-side, reducing the server load and providing faster response times.

Widely Supported: It?s an essential part of web development and is supported by all major browsers.

Versatility: JavaScript can be used for both front-end (with libraries and frameworks like React, Angular, Vue)

and back-end development (using Node.js).

3. Writing JavaScript into HTML

Syntax Rules:

Statements end with a semicolon ;.

Case Sensitivity: JavaScript is case-sensitive.

Comments:

Single-line: // This is a comment

Multi-line: /* This is a comment */

Embedding JavaScript:

You can embed JavaScript code in HTML in the <script> tag or external files. JavaScript interacts with HTML elements using the DOM (Document Object Model).

4. Programming Concepts in JavaScript

Data Types and Literals

JavaScript supports several types of data that can be categorized into primitive and non-primitive data types:

Primitive Data Types:

Number: Represents numeric values.

let age = 30;

String: Represents a sequence of characters.

let name = "John";

Boolean: Represents a true or false value.

let isStudent = true;

Null: Represents the intentional absence of any value.

let value = null;

Undefined: A variable declared without a value will be undefined.

let x;

console.log(x); // undefined

Symbol: Represents a unique value (used for creating unique identifiers).

BigInt: For representing integers larger than 2^53 - 1.

let bigNum = 1234567890123456789012345678901234567890n;

Type Casting

Implicit Type Conversion: JavaScript automatically converts data types when necessary.

let str = "5";

let num = 10;

let result = str + num; // "510" (string concatenation)

Explicit Type Conversion: You can manually convert between types using functions like Number(), String(), and Boolean().

let str = "100";

let num = Number(str); // Converts string to number

Variables

Variables are used to store data.

var: Declares a variable with function scope (older, not recommended for modern use).

let: Declares a block-scoped variable.

const: Declares a block-scoped variable that cannot be reassigned.

let x = 10; // Can be reassigned

const y = 20; // Cannot be reassigned

Arrays

In JavaScript, arrays are used to store multiple values in a single variable. An array can store a collection of values, which can be of any data type, such as strings, numbers, or even other arrays.

1. Creating an Array:

You can create an array using either the Array constructor or the array literal syntax.

Array Literal Syntax:

let fruits = ['apple', 'banana', 'orange'];

console.log(fruits); // ['apple', 'banana', 'orange']

Array Constructor Syntax:

let fruits = new Array('apple', 'banana', 'orange');

console.log(fruits); // ['apple', 'banana', 'orange']

2. Accessing Array Elements:

You can access elements of an array using the index. JavaScript arrays are zero-indexed, meaning the first element is at index 0.

```
let fruits = ['apple', 'banana', 'orange'];

console.log(fruits[0]); // 'apple'

console.log(fruits[1]); // 'banana'

console.log(fruits[2]); // 'orange'
```

3. Modifying Array Elements:

You can modify an array element by assigning a new value to the specified index.

```
let fruits = ['apple', 'banana', 'orange'];

fruits[1] = 'grape';  // Changing 'banana' to 'grape'

console.log(fruits);   // ['apple', 'grape', 'orange']
```

4. Array Length:

To find the length of an array (i.e., how many elements it contains), use the length property.

```
let fruits = ['apple', 'banana', 'orange'];

console.log(fruits.length);  // 3
```

5. Adding and Removing Elements from Arrays:

push(): Adds one or more elements to the end of the array.

pop(): Removes the last element of the array.

shift(): Removes the first element of the array.

unshift(): Adds one or more elements to the beginning of the array.

```
let fruits = ['apple', 'banana', 'orange'];

fruits.push('grape');  // Adds 'grape' to the end

console.log(fruits);    // ['apple', 'banana', 'orange', 'grape']

fruits.pop();           // Removes the last element ('grape')

console.log(fruits);    // ['apple', 'banana', 'orange']

fruits.shift();         // Removes the first element ('apple')

console.log(fruits);    // ['banana', 'orange']
```

```
fruits.unshift('kiwi'); // Adds 'kiwi' to the beginning
console.log(fruits);    // ['kiwi', 'banana', 'orange']
```

6. Looping Through an Array:

You can use different loop methods to iterate through an array:

For Loop:

```
let fruits = ['apple', 'banana', 'orange'];
for (let i = 0; i < fruits.length; i++) {
console.log(fruits[i]);
}
// Output:
// apple
// banana
// orange
```

forEach Method:

```
let fruits = ['apple', 'banana', 'orange'];
fruits.forEach(function(fruit) {
console.log(fruit);
});
// Output:
// apple
// banana
// orange
```

7. Array Methods:

JavaScript arrays come with many built-in methods:

concat(): Combines two or more arrays.

join(): Joins all elements of an array into a string.

slice(): Returns a shallow copy of a portion of the array.

splice(): Adds or removes elements from an array.

map(): Creates a new array by applying a function to each element.

filter(): Creates a new array with elements that pass a test.

Example of using map() and filter():

```
let numbers = [1, 2, 3, 4, 5];

// Using map() to square each number

let squaredNumbers = numbers.map(num => num * num);

console.log(squaredNumbers);  // [1, 4, 9, 16, 25]

// Using filter() to get even numbers

let evenNumbers = numbers.filter(num => num % 2 === 0);

console.log(evenNumbers);  // [2, 4]
```

8. Multi-Dimensional Arrays:

JavaScript arrays can contain other arrays, creating multi-dimensional arrays.

```
let matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

console.log(matrix[0][0]);  // 1

console.log(matrix[1][2]);  // 6

console.log(matrix[2][1]);  // 8
```

9. Array Destructuring:

You can use destructuring to assign array elements to variables.

```
let fruits = ['apple', 'banana', 'orange'];

let [first, second, third] = fruits;

console.log(first);  // 'apple'

console.log(second); // 'banana'

console.log(third);  // 'orange'
```

Summary of JavaScript Array Features:

Creation: Arrays can be created using the array literal ([]) or new Array().

Accessing: Access elements by index.

Modifying: You can change array elements by index.

Adding/Removing: Methods like push(), pop(), shift(), and unshift() modify arrays.

Iteration: Use loops like for, forEach(), etc., to iterate over arrays.

Methods: Built-in methods like map(), filter(), concat(), join(), and slice() help in manipulation.

Multi-Dimensional: JavaScript arrays can be nested to create multi-dimensional arrays.

Destructuring: Use destructuring to extract values from arrays into variables.

In JavaScript, operators are symbols used to perform operations on variables and values, and expressions are combinations of variables, values, and operators that produce a result.

1. Types of Operators in JavaScript

A. Arithmetic Operators

These operators perform arithmetic operations such as addition, subtraction, multiplication, division, etc.

B. Assignment Operators

Assignment operators are used to assign values to variables.

C. Comparison Operators

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

D. Logical Operators

Logical operators are used to perform logical operations, usually on boolean values.

E. Unary Operators

Unary operators work with a single operand.

F. Ternary (Conditional) Operator

The ternary operator is a shorthand for if-else statements. It evaluates a condition and returns one of two values based on whether the condition is true or false.

javascript

Copy

condition ? expressionIfTrue : expressionIfFalse;

G. Bitwise Operators

Bitwise operators operate on binary representations of numbers.

H. String Operators

String operators are used for concatenation.

I. Type Operators

These operators check and manipulate the type of a value.

2. Expressions in JavaScript

An expression in JavaScript is any valid unit of code that resolves to a value. Expressions can include variables, literals, operators, function calls, and more.

A. Simple Expression:

A simple expression could be a variable or a constant.

```
let x = 5;    // Here, `5` is an expression
```

B. Arithmetic Expression:

Combining operands and arithmetic operators to produce a result.

```
let sum = 10 + 5;  // `10 + 5` is an arithmetic expression
```

C. Logical Expression:

Combining operands with logical operators.

```
let isAdult = age >= 18 && hasID;  // `age >= 18 && hasID` is a logical expression
```

D. Conditional Expression (Ternary Operator):

Using the ternary operator to create a condition.

```
let result = (a > b) ? 'a is greater' : 'b is greater';  // Ternary expression
```

E. Function Expressions:

Functions themselves can be expressions. You can assign them to variables.

```
let greet = function(name) { return 'Hello ' + name; };
```

F. Array Expressions:

Creating arrays is also an expression.

```
let numbers = [1, 2, 3, 4];  // Array expression
```

G. Object Expressions:

Defining objects with properties is an expression.

```
let person = { name: 'John', age: 30 };  // Object expression
```

3. Operator Precedence

JavaScript follows a precedence order when evaluating expressions. Operators with higher precedence are evaluated before operators with lower precedence.

Example Precedence (highest to lowest):

Parentheses ()

Unary operators ++, --, typeof

Arithmetic operators *, /, %

Relational operators <, >, <=, >=

Logical operators &&, ||

Example:

```
let x = 5 + 3 * 2;  // 5 + (3 * 2) ? 11
```

In this example, multiplication (*) has higher precedence than addition (+), so 3 * 2 is evaluated first.

Summary of Key JavaScript Operators:

Arithmetic Operators: +, -, *, /, %, ++, --

Assignment Operators: =, +=, -=, *=, /=, %=

Comparison Operators: ==, ===, !=, !==, >, <, >=, <=

Logical Operators: &&, ||, !

Unary Operators: +, -, ++, --, typeof, delete

Ternary Operator: ? :

Bitwise Operators: &, |, ^, ~, <<, >>, >>>

String Operators: +, +=

Type Operators: typeof, instanceof

These operators and expressions are the building blocks for performing computations, comparisons, and logical operations in JavaScript.

JavaScript ? Conditional Statements

JavaScript conditional statements allow you to execute specific blocks of code based on conditions. If the condition is met, a particular block of code will run; otherwise, another block of code will execute based on the condition.

1. Using if Statement

The if statement is used to evaluate a particular condition. If the condition holds true, the associated code block is executed.

```
let x = 20;
if (x % 2 === 0) {
console.log("Even");
}
if (x % 2 !== 0) {
console.log("Odd");
};
```

2. Using if-else Statement

The if-else statement will perform some action for a specific condition. Here we are using the else statement in which the else statement is written after the if statement and it has no condition in their code block.

```
let age = 25;
if (age >= 18) {
console.log("Adult")
} else {
console.log("Not an Adult")
};
```

3. else if Statement

The else if statement in JavaScript allows handling multiple possible conditions and outputs, evaluating more than two options based on whether the conditions are true or false.

```
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)

As the number of conditions increases, you can use multiple else-if statements in JavaScript. but when we dealing with many conditions, the switch statement may be a more preferred option.

```
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}`);
```

Output

Student Branch name is : Mechanical engineering

5. Using Ternary Operator (?:)

The conditional operator, also referred to as the ternary operator (?:), is a shortcut for expressing conditional statements in JavaScript.

```
let age = 21;

const result =

(age >= 18) ? "You are eligible to vote."

: "You are not eligible to vote.";

console.log(result);
```

Output

You are eligible to vote.

6. Nested if?else

Nested if?else statements in JavaScript allow us to create complex conditional logic by checking multiple conditions in a hierarchical manner. Each if statement can have an associated else block, and within each if or else block, you can nest another if?else statement. This nesting can continue to multiple levels, but it?s important to maintain readability and avoid excessive complexity.

```
let weather = "sunny";

let temperature = 25;

if (weather === "sunny") {

if (temperature > 30) {

console.log("It's a hot day!");

} else if (temperature > 20) {

console.log("It's a warm day.");

} else {

console.log("It's a bit cool today.");

}

} else if (weather === "rainy") {

console.log("Don't forget your umbrella!");
```

} else {

console.log("Check the weather forecast!");

};

Output

It's a warm day.

JavaScript Loops

Loops in JavaScript are used to reduce repetitive tasks by repeatedly executing a block of code as long as a specified condition is true. This makes code more concise and efficient.

Suppose we want to print ?Hello World? five times. Instead of manually writing the print statement repeatedly, we can use a loop to automate the task and execute it based on the given condition.

for (let i = 0; i < 5; i++) {

console.log("Hello World!");

}

Output

Hello World!

Hello World!

Hello World!

Hello World!

Hello World!

Let?s now discuss the different types of loops available in JavaScript

1. JavaScript for Loop

The 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

}

for (let i = 1; i <= 3; i++) {

```
console.log("Count:", i);
}
```

Output

```
Count: 1
Count: 2
Count: 3
```

In this example

Initializes the counter variable (let i = 1).

Tests the condition (i <= 3); runs while true.

Executes the loop body and increments the counter (i++).

2. JavaScript while Loop

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

Syntax

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

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

Output

```
Number: 0
Number: 1
Number: 2
```

In this example

Initializes the variable (let i = 0).

Runs the loop body while the condition (i < 3) is true.

Increments the counter after each iteration (i++).

3. JavaScript do-while Loop

The 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);
```

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

Output

Iteration: 0

Iteration: 1

Iteration: 2

In this example:

Executes the code block first.

Checks the condition (i < 3) after each iteration.

4. JavaScript for-in Loop

The is used to iterate over the properties of an object. It only iterate over keys of an object which have their enumerable property set to ?true?.

Syntax

```
for (let key in object) {
    // Code to execute
}
```

```
const obj = { name: "Ashish", age: 25 };
for (let key in obj) {
console.log(key, ":", obj[key]);
```

}

Output

name : Ashish

age : 25

In this example:

Iterates over the keys of the person object.

Accesses both keys and values.

5. JavaScript for-of Loop

The is used to iterate over iterable objects like arrays, strings, or sets. It directly iterate the value and has more concise syntax than for loop.

Syntax

```
for (let value of iterable) {
    // Code to execute
}
```

```
let a = [1, 2, 3, 4, 5];
for (let val of a) {
console.log(val);
}
```

Output

1

2

3

4

5

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 sum(x, y) {
return x + y;
```

```
}

console.log(sum(6, 9));
```

Output

15

Function Syntax and Working

A function definition is sometimes also termed a function declaration or function statement. Below are the rules for creating a function in JavaScript:

Begin with the keyword function followed by,

A user-defined function name (In the above example, the name is sum)

A list of parameters enclosed within parentheses and separated by commas (In the above example, parameters are x and y)

A list of statements composing the body of the function enclosed within curly braces {} (In the above example, the statement is ?return x + y?).

Return Statement

In some situations, we want to return some values from a function after performing some operations. In such cases, we make use of the return. This is an optional statement. In the above function, ?sum()? returns the sum of two as a result.

Function Parameters

Parameters are input passed to a function. In the above example, sum() takes two parameters, x and y.

Calling Functions

After defining a function, the next step is to call them to make use of the function. We can call a function by using the function name separated by the value of parameters enclosed between the parenthesis.

```
// Function Definition
function welcomeMsg(name) {
return ("Hello " + name + " welcome to GeeksforGeeks");
}
let nameVal = "User";
// calling the function
console.log(welcomeMsg(nameVal));
```

Output

Hello User welcome to GeeksforGeeks

Why Functions?

Functions can be used multiple times, reducing redundancy.

Break down complex problems into manageable pieces.

Manage complexity by hiding implementation details.

Can call themselves to solve problems recursively.

Function Invocation

The function code you have written will be executed whenever it is called.

Triggered by an event (e.g., a button click by a user).

When explicitly called from JavaScript code.

Automatically executed, such as in .

Function Expression

It is similar to a function declaration without the function name. can be stored in a variable assignment.

Syntax:

```
let geeksforGeeks= function(paramA, paramB) {
    // Set of statements
}
```

```
const mul = function (x, y) {
return x * y;
};
console.log(mul(4, 5));
```

Output

20

Arrow Functions

are a concise syntax for writing functions, introduced in , and they do not bind their own this context.

Syntax:

```
let function_name = (argument1, argument2 ,..) => expression
```

```
const a = ["Hydrogen", "Helium", "Lithium", "Beryllium"];
const a2 = a.map(function (s) {
return s.length;
});
console.log("Normal way ", a2);
const a3 = a.map((s) => s.length);
console.log("Using Arrow Function ", a3);
```

Output

```
Normal way  [ 8, 6, 7, 9 ]
Using Arrow Function  [ 8, 6, 7, 9 ]
```

Immediately Invoked Function Expression (IIFE)

are executed immediately after their definition. They are often used to create isolated scopes.

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

Output

```
This runs immediately!
```

Callback Functions

A 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));
```

Output

```
10
```

Anonymous Functions

are functions without a name. They are often used as arguments to other functions.

```
setTimeout(function () {
console.log("Anonymous function executed!");
}, 1000);
```

Nested Functions

Functions defined within other functions are called . They have access to the variables of their parent function.

```
function outerFun(a) {
function innerFun(b) {
return a + b;
}
return innerFun;
}
const addTen = outerFun(10);
console.log(addTen(5));
```

Output

15

Pure Functions

return the same output for the same inputs and do not produce side effects. They do not modify state outside their scope, such as modifying global variables, changing the state of objects passed as arguments, or performing I/O operations.

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

Output

5

Key Characteristics of Functions

Parameters and Arguments: Functions can accept parameters (placeholders) and be called with arguments (values).

Return Values: Functions can return a value using the return keyword.

Default Parameters: Default values can be assigned to function parameters.

Advantages of Functions in JavaScript

Reusability: Write code once and use it multiple times.

Modularity: Break complex problems into smaller, manageable pieces.

Improved Readability: Functions make code easier to understand.

Maintainability: Changes can be made in one place without affecting the entire codebase.

JavaScript Dialogue Boxes

Dialogue boxes are a kind of popup notification, this kind of informative functionality is used to show success, failure, or any particular/important notification to the user.

JavaScript uses 3 kinds of dialog boxes:

Alert

Prompt

Confirm

These dialog boxes can be of very much help in making our website look more attractive.

Alert Box: An alert box is used on the website to show a warning message to the user that they have entered the wrong value other than what is required to fill in that position. Nonetheless, an alert box can still be used for friendlier messages. The alert box gives only one button ?OK? to select and proceed.

Example:

JavaScript

Output:

Confirm box: A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either ?OK? or ?Cancel? to proceed. If the user clicks on the OK button, the window method will return true. If the user clicks on the Cancel button, then confirm() returns false and will show null.

Example:

JavaScript

Output:

Prompt Box: A prompt box is often used if you want the user to input a value before entering a page. When a

prompt box pops up, the user will have to click either ?OK? or ?Cancel? to proceed after entering an input value. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method returns null.

Example:

JavaScript

Output:

Line Breaker: If you want a break in your dialogue box message, you can put a line breaker(?\n?) there.

Example:

JavaScript

Output:

JavaScript Cookies

cookies are small data stored on a user's device by a web browser. These cookies play a crucial role in web development, enabling websites to store and retrieve information about user preferences, , and other data. Cookies facilitate a more personalized browsing experience by allowing websites to remember user actions and preferences. They are widely used for user authentication, tracking, and maintaining session states.

Creating Cookie in JavaScript

Cookies are created by a web server and sent to the user's browser as part of the .

Creating cookies in JavaScript involves using the document.cookie object to set key-value pairs and additional parameters. To create a cookie, assign a string containing the desired cookie information to document.cookie. This string can include attributes like expiration date, domain, and path.

Example: To demonstrate creating a cookie in JavaScript using a document.cookie method.

document.cookie =

"courseName=webDeveopment bootcamp; expires =

Thu, 5 March 2030 12:00:00 UTC; path = /";

Output:

courseName= "webDeveopment bootcamp; expires = Thu, 5 March 2030 12:00:00 UTC; path = /"

Reading Cookie in JavaScript

JavaScript allows developers to read cookies using the document.cookie property, which stores all cookies as a string. To extract specific values, developers often create functions that parse this string. Security

considerations, like proper decoding and HttpOnly attributes, are crucial.

Example: To demonstrate reading an already created cookie in JavaScript.

```
function getCookie(cookieName) {
const cookies = document.cookie.split('; ');
for (const cookie of cookies) {
const [name, value] = cookie.split('=');
if (name === cookieName) {
return decodeURIComponent(value);
}
}
return null;
}
const courseName = getCookie('courseName');
console.log('Course Name:', courseName);
```

Output:

Course Name: webDeveopment bootcamp

Changing Cookie in JavaScript

JavaScript enables the modification of cookies by updating their values or attributes. Developers use the document.cookie property to both read and write cookies. When changing a cookie, it's crucial to consider parameters like expiration date, path, and security attributes.

Example: To demonstrate changing an already existing cookie in JavaScript.

```
document.cookie =
&amp;quot;courseName=WebDevelopmentBootcamp;
expires=Thu, 5 March 2030 12:00:00 UTC; path=/&amp;quot;;
function changeCookieValue(cookieName, newValue) {
document.cookie =
`${cookieName}=${newValue};
expires=Thu, 5 March 2030 12:00:00 UTC; path=/`;
```

}

changeCookieValue('courseName', 'AdvancedJavaScriptCourse');

Output:

Course Name: WebDevelopmentBootcamp

Course Name after Change: AdvancedJavaScriptCourse

Deleting Cookie in JavaScript

JavaScript provides a way to delete cookies by setting their expiration date in the past. When a cookie's expiration date is in the past, the browser automatically removes it. Developers use the document.cookie property to delete cookies, ensuring a clean and secure user experience.

Example: To demonstrate deleting a cookie in JavaScript using

document.cookie =

"courseName=WebDevelopmentBootcamp;

expires=Thu, 5 March 2030 12:00:00 UTC; path=/";

function deleteCookie(cookieName) {

document.cookie =

`${cookieName}=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;`;

}

deleteCookie('courseName');

Output:

Course Name: WebDevelopmentBootcamp

Cookie 'courseName' deleted.