JavaScript Basics

Introduction to JavaScript

Scripting Languages are high-level programming languages that are used to automate tasks, control the behaviour of software applications, or manipulate data in various environments. These languages are typically interpreted, meaning that their code is executed line-by-line by an interpreter rather than being compiled into machine code before execution.

Key Characteristics of Scripting Languages:

1. **Interpreted**: Scripting languages are generally interpreted, not compiled. This means that the source code is executed directly by an interpreter without a separate compilation step.
2. **Ease of Use**: Scripting languages often have simple syntax and are designed to be easy to write and understand.
3. **Dynamic Typing**: Many scripting languages do not require explicit declaration of data types, allowing for more flexibility during coding.
4. **Automation**: They are used primarily to automate repetitive tasks, simplifying workflows in different computing environments.
5. **Integration**: Scripting languages are often used to integrate and communicate between different software applications and systems, providing interoperability.

Where?

Scripting languages are widely used in various domains:

1. **Web Development**:

   Client-Side Scripting: JavaScript is commonly used in web development for creating interactive and dynamic web pages. It allows the browser to respond to user actions like clicks, form submissions, and animations without reloading the page. (Examples: JavaScript, VBScript, J Script etc…)

   Server-Side Scripting: Languages like PHP, Python, ASP, JSP, Servlets, Ruby are often used on the server-side to process requests, manage databases, and generate dynamic web content.
2. **Automation**: Scripting languages are commonly used in automation tasks, including system administration, file management, and even network monitoring. Bash scripting (on Unix-based systems) or PowerShell (on Windows) are commonly used to automate system tasks.

3. **Game Development**: Many game engines use scripting languages like Lua or Python to allow game developers to control game behavior, events, and interactions without needing to modify the core engine code.
4. **Data Analysis**: Scripting languages such as Python and R are used extensively in data analysis, machine learning, and artificial intelligence (AI) due to their rich libraries and frameworks that enable complex data manipulation, visualization, and algorithm implementation.
5. **Software Prototyping**: Due to their simplicity and rapid development cycles, scripting languages are often used to create quick prototypes of software, allowing developers to test concepts or demonstrate functionality without needing to write full-fledged, compiled programs.
6. **Web Scraping**: Scripting languages, particularly Python, are used in web scraping to extract data from websites automatically for analysis, research, or data aggregation.

Why?

1. **Quick Development**: Scripting languages are generally more flexible and faster for development because they do not require explicit compilation steps, and their syntax is often simpler than that of traditional programming languages. This makes them ideal for rapid development of applications.
2. **Interactivity and Dynamic Behavior**: In web development, scripting languages like JavaScript enable the creation of interactive web pages, enhancing user experience by making content dynamic (e.g., form validation, animations, etc.).
3. **Ease of Debugging and Testing**: Because scripting languages are interpreted, developers can test and debug them interactively, which helps in identifying and fixing errors quickly. This is in contrast to compiled languages, where testing often requires recompiling the code after every change.
4. **Cross-Platform Compatibility**: Most scripting languages are platform-independent, meaning that they can run on different operating systems with minimal changes. This is especially useful for web development, where scripts written in JavaScript run on all modern browsers, regardless of the underlying OS.
5. **Automation of Repetitive Tasks**: Scripting languages are ideal for automating repetitive or time-consuming tasks. For instance, a Python script can automate data entry, system backups, or even web scraping.

6. **Integration with Other Systems**: Scripting languages are often used to glue together various components of a system. For example, a Python script can interact with databases, file systems, APIs, and other software tools to create more sophisticated applications without needing to write a full-fledged application in a compiled language.

Scripting languages play a crucial role in modern software development and system administration. Their ability to automate tasks, interact with other systems, and quickly prototype applications has made them indispensable across various domains such as web development, data analysis, game development, and system automation. Despite their simplicity, they offer powerful capabilities that streamline workflows, boost productivity, and improve the overall user experience.

JavaScript:

☑ JavaScript is a scripting language.

☑ JavaScript is a lightweight programming language and that is interpreted by the browser engine when the web page is loaded.

Features:

All popular web browsers support JavaScript as they provide built-in execution environments.

1. It is a light-weighted and interpreted language.
2. It is a case-sensitive language.
3. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
4. Validate web form data
5. Make web pages responsive

Advantages:

1. Client-side validation.
2. Dynamic drop-down menus
3. Displaying date and time.
4. Displaying pop-up windows and dialog boxes.

Comparison of Java and JavaScript:

Aspect	Java	JavaScript
Definition	General-purpose, object-oriented programming language used for building complex applications.	High-level, interpreted scripting language primarily used for web development.
Type of Language	Compiled, strongly typed, object-oriented language.	Interpreted, dynamically typed, multi-paradigm language (supports functional, object-oriented programming).

Aspect	Java	JavaScript
Platform	Platform-independent, runs on Java Virtual Machine (JVM).	Platform-independent, runs on browsers and Node.js (server-side).
Compilation	Code is compiled into bytecode and executed by the JVM.	Code is interpreted by browsers or server-side engines (Node.js).
Execution Speed	Faster execution due to pre-compilation (JVM).	Slower execution due to runtime interpretation (though JIT compilation in modern browsers has improved speed).
Syntax	Similar to C and C++, with strict syntax rules.	Similar to C, but more flexible with syntax (dynamic typing, no strict rules).
Typing	Statically typed: Variables must be declared with specific types.	Dynamically typed: Variable types are determined at runtime.
Memory Management	Manual memory management (via garbage collection) within JVM.	Automatic memory management with garbage collection in JavaScript engines.
Object-Oriented Support	Fully object-oriented: Everything is part of a class (except primitive types).	Supports object-oriented programming, but is more flexible with prototypes rather than classes.
Concurrency	Uses multithreading for concurrency with threads.	Uses event-driven, non-blocking asynchronous programming (with callbacks, promises, async/await).
Platform Independence	"Write once, run anywhere" (WORA). Java code runs on any platform that has JVM installed.	Code runs in any modern web browser and on servers with Node.js.
Usage	Used for large-scale applications, enterprise systems, Android development, backend development.	Primarily used for frontend web development (interactive elements), and backend with Node.js.
Popular Frameworks	Spring, Hibernate, JavaFX, Android SDK.	React, Angular, Vue.js, Node.js, Express.js.
Execution Environment	Requires JVM (Java Virtual Machine).	Requires web browser or Node.js environment.
File Extension	.java for source code; .class for bytecode.	.js for JavaScript files.
Error Handling	Uses try-catch-finally blocks for exception handling.	Uses try-catch for handling errors in JavaScript.
Inheritance	Class-based inheritance model (extends classes).	Prototype-based inheritance model (inherits from other objects).

Aspect	Java	JavaScript
Access Modifiers	Supports private, protected, and public access modifiers for classes and methods.	No explicit access modifiers (public by default). Can use closures to simulate encapsulation.
Libraries/Packages	Extensive standard libraries (Java API) and third-party libraries.	Extensive standard libraries (JavaScript API) and packages via npm (Node Package Manager).
Support for GUI	Java provides rich GUI libraries (Swing, JavaFX).	JavaScript relies on HTML/CSS for the user interface, with frameworks like React.js or Vue.js for building complex UIs.
Data Types	Supports primitive data types (int, float, char, etc.) and complex types (objects, arrays).	Supports primitive types (number, string, boolean) and dynamic objects. Arrays and functions are objects in JavaScript.
Security	Strong security model with built-in security mechanisms (sandboxing, bytecode verification).	Less robust security features; security is often handled at the browser level.
Popular IDEs	Eclipse, IntelliJ IDEA, NetBeans.	Visual Studio Code, Sublime Text, Atom, WebStorm.
Deployment	Deployed as .jar files (Java ARchive) or web applications (Java EE).	Deployed as .js files in web applications or Node.js applications.

What type of language is JavaScript? Compiled or interpreted?

JavaScript is an **interpreted language**, not a compiled language.

Compiled vs. Interpreted Languages

1. **Compiled Languages**:

   In compiled languages, the source code is first translated (compiled) into machine code or an intermediate bytecode by a **compiler** before execution. This compilation process occurs before runtime, and the output is a standalone executable file.

   Example: C, C++, Java (which compiles into bytecode and is then interpreted by the JVM).

2. **Interpreted Languages**:

   In interpreted languages, the source code is directly executed line-by-line by an **interpreter** at runtime, without needing a prior compilation step. The interpreter translates and executes the code dynamically, which may result in slower execution compared to compiled languages.

   Example: Python, Ruby, JavaScript.

JavaScript: Interpreted Language

JavaScript is classified as an interpreted language for the following reasons:

☑ **Line-by-Line Execution:**

- JavaScript is executed line by line by the browser's JavaScript engine.
- When a webpage is loaded in a browser, the browser's JavaScript engine interprets the JavaScript code directly, without converting it into machine code beforehand.

```
console.log("Hello, World!");
```

This JavaScript code is executed directly by the browser's JavaScript engine when the page is loaded. There is no need for a pre-compilation step.

☑ **Dynamic Execution:**

- JavaScript code is executed at runtime, which means it can be modified or interacted with dynamically. This gives developers flexibility to change or update code on the fly.

```
let userAge = prompt("Enter your age:");
alert("You are " + userAge + " years old.");
```

In this example, the prompt appears during runtime, and the user's input is processed immediately without any compilation step.

☑ **No Pre-compilation:**

- Unlike compiled languages like C or C++, JavaScript doesn't require a separate compilation step before execution. When a web page is loaded, the browser directly interprets and runs the JavaScript code.
- Example: You don't need to compile a JavaScript file before including it in your HTML. You can include the script directly in the HTML file or link to an external .js file, and the browser will automatically interpret the script as it loads the page.

☑ **Cross-platform Interpretation:**

- JavaScript is platform-independent in terms of execution. The JavaScript engine within browsers interprets the JavaScript code regardless of the underlying operating system, making it cross-platform.
- This is why JavaScript can run on any device or operating system that has a modern browser, making it suitable for web applications.

JavaScript Syntax and Basics:

Basic syntax of JavaScript program with respect to HTML:

```
<html>
    <body>
        <script type="text/javascript" language=”javascript”>
            Statement1;
            Statement2;
            Statement3;
        </script>
    </body>
</html>
```

```
<html>
    <body>
        <script type="text/javascript">
            document.write("JavaScript is a simple language for
                learners");
        </script>
    </body>
</html>
```

How the JavaScript program would be compiled?

Unlike languages like C or Java, JavaScript is interpreted and not compiled in the traditional sense. Here's how JavaScript runs:

☑ **Interpretation by Browser**

- JavaScript is typically run directly in a web browser.
- The browser has a built-in JavaScript engine (e.g., V8 in Google Chrome) that interprets the JavaScript code.
- When a web page containing JavaScript is loaded, the browser reads the HTML file, identifies the JavaScript code embedded or linked within it, and interprets and executes it line by line.

☑ **Execution Flow**

- **Synchronous execution:** JavaScript code is executed from top to bottom (line by line).
- **Asynchronous execution:** JavaScript can handle asynchronous events (e.g., callbacks, promises) that allow non-blocking operations (like fetching data from a server).

☑ **No Compilation Step**

- JavaScript doesn't need to be explicitly compiled like C or Java. It is executed directly by the JavaScript engine in the browser (or Node.js environment).
- If you're using Node.js (for server-side JavaScript), the JavaScript code is interpreted by the V8 engine, which directly executes the code.

Few scripting languages:

☑ **JavaScript** – Primarily used for web development to create interactive effects within web browsers.

- ☑ **Python** – Known for its simplicity and readability, used for web development, data analysis, artificial intelligence, and automation.
- ☑ **Ruby** – Often used in web development, particularly with the Ruby on Rails framework.
- ☑ **PHP** – Widely used for server-side web development, especially for creating dynamic web pages.
- ☑ **Perl** – Known for its text-processing capabilities, used in system administration, web development, and network programming.
- ☑ **Bash** – A Unix shell and command language, commonly used for scripting in Linux and macOS.
- ☑ **PowerShell** – A task automation and configuration management framework from Microsoft, used for system administration.
- ☑ **Lua** – Lightweight scripting language, often embedded in game engines and applications.
- ☑ **TypeScript** – A superset of JavaScript that adds static typing, widely used for web development.
- ☑ **ActionScript** – Used for developing web applications with Adobe Flash.

Advantages of Submitting a Form Using JavaScript:

1. **No Page Reload (AJAX Submission):** Using JavaScript for form submission, especially through AJAX, allows data to be sent to the server asynchronously without reloading the page. This creates a smoother user experience since the page does not need to refresh after form submission.
2. **Improved User Experience:** JavaScript allows for more dynamic interactions. You can validate the form data before submission, show real-time error messages, or show loading indicators without a page refresh.
3. **Form Validation Before Submission:** JavaScript can validate the form data before it is sent to the server. This reduces the chances of invalid data being sent and enhances the overall user experience.

   Example: You can ensure that fields like email or phone number are properly formatted before submitting the form.
4. **Control Over the Submission Process:** With JavaScript, you have full control over when and how the form is submitted. You can cancel the submission based on custom logic (e.g., if the form contains errors or if additional information is required from the user).

5. **Conditional Form Submission:** JavaScript allows for conditional form submission based on user inputs, improving the flexibility of how forms behave. For example, you could choose to send data to different servers based on user choices or conditions.
6. **Enhanced Security:** You can use JavaScript to add security measures, such as ensuring data is properly sanitized before sending it to the server, preventing certain attacks (e.g., XSS).
7. **Better Feedback and Real-Time Updates:** You can instantly provide feedback to the user after form submission (e.g., success messages, error alerts, or even partial form updates), improving interaction without requiring a reload.
8. **Use of JSON for Data Handling:** JavaScript makes it easier to submit form data as JSON, which can be useful if you're working with modern web frameworks or APIs that prefer JSON over traditional form submission methods.
9. **Integrating with APIs:** JavaScript-based form submission can easily integrate with third-party APIs, allowing for tasks like form submission, email sending, or even handling payments asynchronously.
10. **Speed and Efficiency:** Since there's no need for the entire page to reload, the submission process can be faster and more efficient, leading to improved overall site performance.

In summary, using JavaScript for form submission enhances user experience, provides flexibility, and allows asynchronous data submission without the need for full page reloads.

We can place JavaScript code in 3 ways in HTML code:

1. **Inline JavaScript (within HTML attributes)**

   JavaScript can be directly written inside HTML attributes, such as onclick, onsubmit, onmouseover, etc.

   The code is executed when the event associated with the attribute occurs.

   Example:

2. **Internal JavaScript (within <script> tags)**

   JavaScript can be written directly within <script> tags in the <head> or <body> of an HTML document.

   This is useful for more complex logic than what you would use in an inline attribute.

   Example:

3. **External JavaScript (with src attribute in <script> tag)**

   JavaScript can be stored in an external .js file and linked to the HTML document using the src attribute in the <script> tag.

This method is preferred for keeping the JavaScript code separate from the HTML content, making the code easier to manage and maintain.

Example:

Note: *JavaScript can be placed between the body tags of html or between the head tags of html.*

Pop Up Boxes:

A pop-up box in JavaScript refers to a small window or dialog that appears on the screen to interact with the user, typically for showing information or requesting input. It temporarily "pops up" over the main content of a web page. In the context of the examples you asked for earlier, pop-up boxes are methods provided by JavaScript to engage users, either to display messages or collect user input.

The Three Common Types of Pop-Up Boxes:

1. **alert()**:
   - A simple pop-up box that shows a message to the user and provides an OK button to dismiss it.
   - Example: It can display a notification or warning.
   - **Usage**: Primarily used for giving information or alerts.

   Example:
2. **confirm()**:
   - A pop-up box that asks the user to confirm or cancel an action. It typically contains two buttons: OK (True) and Cancel (False).
   - Example: It is used when you want the user to make a decision, like confirming whether they want to proceed with an action.
   - **Usage**: Useful for asking "yes/no" or "proceed/cancel" type questions.

   Example:
3. **prompt()**:
   - A pop-up box that asks the user to input some text (e.g., a name or value). It displays a text field along with OK and Cancel buttons.
   - Example: It is used when you need input from the user (such as asking for their name).
   - **Usage**: Ideal for gathering input from users in the form of text.

   Example:

Each of these methods allows for interaction with the user through a modal dialog that blocks interaction with the rest of the page until the user responds.

Comparison of alert(), confirm(), and prompt():

Method	Description	Returns	Use Case
alert()	Displays a simple message box with an OK button.	Returns undefined.	Used for showing information or warnings to the user.
confirm()	Displays a dialog box with OK and Cancel buttons.	Returns true if OK is clicked, false if Cancel is clicked.	Used when you need the user to confirm or deny an action (Yes/No).
prompt()	Displays a dialog box asking the user for input (e.g., a name).	Returns the input text as a string if OK is clicked, null if Cancel is clicked.	Used when you need to get input from the user (e.g., text input).

Operators:

What is an operator?

An Operator is a symbol which will perform certain operation on the corresponding operands. These are following types of operators in JavaScript.

1. **Arithmetic Operators**
2. **Comparison (Relational) Operators**
3. **Bitwise Operators**
4. **Logical Operators**
5. **Assignment Operators**
6. **Special Operators**

Explanation:

1. Arithmetic Operators

JavaScript supports the following arithmetic operators:

+ (Addition), - (Subtraction), * (Multiplication), / (Division), % (Modulus), ++ (Increment), -- (Decrement)

Note − Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example:

2. Comparison (Relational) Operators

JavaScript supports the following comparison operators:

= = (Equal), != (Not Equal), > (Greater than), < (Less than), >= (Greater than or Equal to), <= (Less than or Equal to)

Example:

3. Bitwise Operators

JavaScript supports the following bitwise operators:

& (Bitwise AND), | (Bitwise OR), ^ (Bitwise XOR), ~ (Bitwise Not), << (Left Shift), >> (Right Shift), >>> (Right shift with Zero)

Example:

4. Logical Operators

JavaScript supports the following logical operators: && (Logical AND), || (Logical OR), ! (Logical NOT)

Example:

5. Assignment Operators

JavaScript supports the following assignment operators:

= (Simple Assignment), += (Add and Assignment), −= (Subtract and Assignment), *= (Multiply and Assignment), /= (Divide and Assignment), %= (Modules and Assignment)

Example:

6. Special Operators

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Var var_name=condition? True value: false value ;

var big=x>y ? x:y;

Example:

typeof Operator

In JavaScript, the typeof operator returns the data type of its operand in the form of a string. The operand can be any object, function, or variable.

Syntax:

typeof operand

or

typeof (operand)

Example:

Difference between == and === in JavaScript:

In JavaScript, == and === are comparison operators, but they differ in how they compare values.

1. == (Loose Equality / Abstract Equality)

- The == operator compares two values for equality after performing type coercion. This means that if the operands are of different types, JavaScript will attempt to convert them to a common type before making the comparison.
- Type Coercion: When using ==, JavaScript tries to automatically convert the operands to the same type for comparison.

2. **=== (Strict Equality)**
   - The === operator compares both value and type of the operands. No type coercion is performed, meaning the operands must be of the same type to be considered equal.
   - No Type Coercion: The operands are compared as-is without any conversion.

Why prefer === over ==?

- Predictability: Since === doesn't perform type coercion, it is more predictable. It ensures that both value and type must be identical.
- Avoiding bugs: Type coercion can sometimes lead to unexpected results (e.g., 0 == false is true), so using === can help avoid subtle bugs in code.

Key Differences:

Feature	== (Loose Equality)	=== (Strict Equality)
Type Coercion	Yes, performs type conversion if types differ.	No, both value and type must be the same.
Comparing Different Types	Converts one or both values to a common type before comparing.	Compares both type and value exactly, without conversion.
Common Use Cases	Used when you expect automatic type conversion.	Used when you want to ensure both value and type are the same.
Example	`5 == "5"` (true)	`5 === "5"` (false)

Variables and Data Types:

1. Variables in JavaScript

A variable in JavaScript is a container that holds a value, and you can assign new values to it over time. Variables are declared using three keywords: var, let, and const.

a. Declaring Variables:

- **var**: Older keyword for declaring variables, function-scoped.
- **let**: A newer keyword, block-scoped, and allows reassignment.
- **const**: Block-scoped, but the value cannot be reassigned after initialization.

Example:

```
var name = "John";     // Declaring a variable with 'var'
let age = 30;          // Declaring a variable with 'let'
const city = "New York"; // Declaring a variable with 'const'
```

- **var**: Variables declared with var are function-scoped, meaning they are only accessible within the function where they are declared.
- **let**: Variables declared with let are block-scoped, meaning they are accessible within the nearest block (i.e., between curly braces {}).
- **const**: Constants declared with const are also block-scoped, but they cannot be reassigned after their initial assignment.

Example-1:

Example-2:

Data Types in JavaScript

JavaScript supports several data types that can be broadly classified into two categories:

- **Primitive Data Types**: These are basic data types that hold a single value and are immutable.
- **Non-Primitive (Reference) Data Types**: These are more complex and can store multiple values or reference other objects.
- **Primitive Data Types**:

1. **String**: Represents a sequence of characters. Strings are enclosed in single quotes (') or double quotes (").

```
let greeting = "Hello, World!";  // String
let name = 'John';               // String
```

2. **Number**: Represents both integers and floating-point numbers.

```
let age = 30;        // Integer
let height = 5.9;    // Float
```

3. **Boolean**: Represents a logical value: true or false.

```
let isMarried = false; // Boolean
let isAdult = true;    // Boolean
```

4. **Undefined**: A variable that has been declared but not assigned a value is undefined.

```
let x;  // Undefined
console.log(x);  // Output: undefined
```

5. **Null**: Represents the intentional absence of any object value.

```
let user = null; // Null
```

6. **Symbol:** Represents a unique and immutable value that is often used for object property keys.

```
let mySymbol = Symbol('description'); // Create a Symbol with an optional description
```

7. **BigInt:** Allows working with large integers that exceed the limit of the Number type.

```
let largeNumber = 1234567890123456789012345678901234567890n; // BigInt example
```

Example:

- **Non-Primitive (Reference) Data Types**:

1. **Object**: A collection of key-value pairs. Objects are used to store multiple values and are the most complex data type in JavaScript.

```
let obj = { key1: value1, key2: value2, ... };
```

2. **Array**: A special type of object used to store ordered collections of values (elements). Arrays are indexed and can hold different types of data.

```
let arr = [element1, element2, element3, ...];
```

3. **RegExp:** A regular expression (RegExp) is an object used for pattern matching within strings, enabling powerful text search and manipulation.

```
let regex = /pattern/flags;
```

```
let regex = new RegExp('pattern', 'flags');
```

4. **Date:** The Date object is used to work with dates and times, allowing you to create, manipulate, and format date objects in JavaScript.

```
let date = new Date(); // Current date and time
let specificDate = new Date(year, month, day, hours, minutes, seconds, milliseconds);
```

Example:

JavaScript Conditional Statements:

JavaScript supports conditional statements which are used to perform different actions based on different conditions.

JavaScript supports the following forms of if-else statements:

- **if statement**

  The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

  Syntax:

```
if (expression)
{
    Statement(s) to be executed if expression is true
}
```

  Example:

- **if...else statement**

  The 'if...else' statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

  Syntax:

```
if (expression)
{
   Statement(s) to be executed if expression is true
}
else
{
   Statement(s) to be executed if expression is false
}
```

  Example:

- **if...else if... statement**

  The if...else if... statement is an advanced form of if…else that allows JavaScript to make a correct decision out of several conditions.

  Syntax:

```
if (expression 1)
{
   Statement(s) to be executed if expression 1 is true
}
else if (expression 2)
{
   Statement(s) to be executed if expression 2 is true
}
else if (expression 3)
{
   Statement(s) to be executed if expression 3 is true
}
else
{
   Statement(s) to be executed if no expression is true
}
```

  Example:

- **Switch:**

The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax:

```
switch (expression)
{
   case condition 1: statement(s)
   break;
   case condition 2: statement(s)
   break;   ...
   case condition n: statement(s)
   break;
   default: statement(s)
}
```

Example:

JavaScript Loops

The JavaScript loops are used to iterate the piece of code using for, while, do while or for-in -loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. **for loop**

   The 'for' loop is the most compact form of looping. It includes the following three important parts −

   - The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
   - The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
   - The iteration statement where you can increase or decrease your counter.

   Syntax:

```
for (initialization; test condition; iteration statement)
{
   Statement(s) to be executed if test condition is true
}
```

   Example:

2. **while loop**

   The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

Syntax:

```
while (expression)
{
   Statement(s) to be executed if expression is true
}
```

Example:

3. **do-while loop**

The JavaScript do while loop iterates the elements for the infinite number of times like while loop. But, code is executed at least once whether condition is true or false. The syntax of do while loop is given below.

Syntax:

```
do
{
   code to be executed
}while (condition);
```

Example:

4. **for/in loop**

The for/in loop in JavaScript is used to iterate over the enumerable properties of an object (including its prototype chain). It's particularly useful when you want to access the keys or property.

Syntax:

```
for (key in object) {
    // code to be executed
}
```

- **key:** The variable that represents the property name in each iteration.
- **object:** The object whose properties you want to loop through.

Example:

Functions in JavaScript:

- A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again.
- The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

```
<script type="text/javascript">
    function functionname(parameter-list)
    {
        Statements;
    }
</script>
```

Examples:

Example 1: Calling a Function Directly in the Script

Example 2: Calling a Function on Button Click

Example 3: Function to Calculate Cube of a Number (No Return Value)

Example 4: Function with Parameters and Return Value

Example 5: Using Prompt and Alert to Calculate Sum

Example 6: Button to Trigger a Prompt and Alert

Example 7: Function with Form Input for Calculation

Objects in JavaScript

1. **Array**
2. **String**
3. **Date**
4. **Window**
5. **Navigator**
6. **Document**

1. Array:

In JavaScript, an array is a special type of object used for storing multiple values in a single variable. Arrays can hold elements of any type, including numbers, strings, objects, other arrays, or even mixed data types.

- Array Indexing: Arrays are zero-indexed, meaning the first element is at index 0.
- Array Methods: JavaScript arrays come with various built-in methods for manipulating and interacting with the data, like .push(), .pop(), .shift(), .unshift(), .map(), .filter(), and more.

Two Ways to Construct Arrays in JavaScript

Let's now look at two common ways to create arrays in JavaScript: using array literal syntax and using the new Array() constructor.

1. Using Array Literal Syntax

This is the most common and concise way to create arrays in JavaScript. It involves placing elements between square brackets [].

Syntax:

```
let arrayName = [element1, element2, element3, ...];
```

Example:

2. Using the new Array() Constructor

Another way to create an array is by using the new Array() constructor. This can be used in three ways:

- Without arguments: Creates an empty array.
- With one argument (length): Creates an array with a specified length, but the elements are empty (i.e., undefined).
- With multiple arguments (values): Creates an array with the specified values.

Syntax:

```
let arrayName = new Array(); // Empty array
let arrayName = new Array(length); // Array with the specified length
let arrayName = new Array(element1, element2, ...); // Array with values
```

Example:

Points to remember about Arrays:

- An array is a special type of variable that stores multiple values using a special syntax.
- An array can be created using array literal or Array constructor syntax.
- Array literal syntax: var stringArray = ["one", "two", "three"];
- Array constructor syntax:var numericArray = new Array(3);
- A single array can store values of different data types.
- An array element (values) can be accessed using zero based index (key). e.g. array[0].
- An array index must be numeric.
- Array includes length property and various methods to operate on array objects.

Methods in JavaScript:

Category	Method Name	Description	Example
Array Methods	push()	Adds one or more elements to the end of an array and returns the new length of the array.	let arr = [1, 2, 3]; arr.push(4); // Adds 4 to the end
	pop()	Removes the last element from an array and returns that element.	let arr = [1, 2, 3, 4]; let removedElement = arr.pop(); // Removes 4

Category	Method Name	Description	Example
	indexOf()	It searches the specified element in the given array and returns the index of the first match.	let fruits = ["Apple", "Banana", "Cherry", "Banana"]; // Find the first index of "Banana" let index = fruits.indexOf("Banana");
	lastIndexOf()	It searches the specified element in the given array and returns the index of the last match.	let fruits = ["Apple", "Banana", "Cherry", "Banana"]; // Find the last index of "Banana" let lastIndex = fruits.lastIndexOf("Banana");
	reverse()	It reverses the elements of given array.	let numbers = [1, 2, 3, 4, 5]; // Reverse the array numbers.reverse();
	sort()	It returns the element of the given array in a sorted order.	let fruits = ["Apple", "Banana", "Cherry", "Date"]; // Sort the array alphabetically (default behavior) fruits.sort();
	concat()	Merges two or more arrays and returns a new array.	let arr1 = [1, 2]; let arr2 = [3, 4]; let combined = arr1.concat(arr2);
	find()	Returns the first element in the array that satisfies the provided testing function.	let arr = [5, 10, 15]; let found = arr.find(element => element > 10);
	findIndex()	Returns the index of the first element in the array that satisfies the provided testing function.	let arr = [5, 10, 15]; let index = arr.findIndex(element => element > 10);
	shift()	Removes the first element from an array and returns that element.	let arr = [1, 2, 3]; let removedElement = arr.shift();
	unshift()	Adds one or more elements to the beginning of an array and returns the new length of the array.	let arr = [1, 2, 3]; arr.unshift(0); // Adds 0 to the beginning
	join()	Joins all elements of an array into a single string.	let arr = ["apple", "banana", "cherry"]; let joined = arr.join(", ");
	slice()	Returns a shallow copy of a portion of an array into a new array object.	let arr = [1, 2, 3, 4]; let sliced = arr.slice(1, 3); // Extracts elements from index 1 to 2

Category	Method Name	Description	Example
	splice()	Changes the contents of an array by removing or replacing existing elements and/or adding new elements.	let arr = [1, 2, 3, 4]; arr.splice(1, 2, 5, 6); // Removes 2 elements from index 1 and adds 5, 6
	forEach()	Executes a provided function once for each array element.	let arr = [1, 2, 3]; arr.forEach(element => console.log(element));
	map()	Creates a new array populated with the results of calling a provided function on every element in the array.	let arr = [1, 2, 3]; let doubled = arr.map(element => element * 2);
	filter()	Creates a new array with all elements that pass the test implemented by the provided function.	let arr = [1, 2, 3, 4]; let evenNumbers = arr.filter(element => element % 2 === 0);
	reduce()	Applies a function against an accumulator and each element in the array (from left to right) to reduce it to a single value.	let arr = [1, 2, 3]; let sum = arr.reduce((accumulator, currentValue) => accumulator + currentValue, 0);
	reduceRight()	Applies a function against an accumulator and each element in the array (from right to left) to reduce it to a single value.	let arr = [1, 2, 3]; let sumRight = arr.reduceRight((accumulator, currentValue) => accumulator + currentValue, 0);
	some()	Tests whether at least one element in the array passes the test implemented by the provided function.	let arr = [1, 2, 3]; let hasEven = arr.some(element => element % 2 === 0);
	every()	Tests whether all elements in the array pass the test implemented by the provided function.	let arr = [2, 4, 6]; let allEven = arr.every(element => element % 2 === 0);

Array Methods Example:

All Array Methods Example:

Example: Search Number in Array

Program-1

Program-2

2. Strings

- A string in JavaScript is a primitive data type used for representing textual data. Strings can include letters, digits, special characters, or any other characters.
- Strings are immutable in JavaScript, meaning once a string is created, it cannot be modified directly. However, you can create a new string based on the manipulation of an existing string.

String Literal Syntax:

This is the most common and concise way to create strings. You can use single quotes ('), double quotes (""), or backticks (``) for template literals.

Syntax:

```
let str1 = "Hello, World!";  // Using double quotes
let str2 = 'Hello, World!';  // Using single quotes
let str3 = `Hello, World!`;  // Using backticks (Template literals)
```

Example:

String Object (Using new Keyword):

A string can also be created using the String constructor. This is less common and generally not recommended because string literals are more efficient and simpler to use.

Syntax:

```
let strObj = new String("Hello, World!");
```

Example:

String Methods	charAt(position)	Returns the character at the specified position (in Number).	let text = "Hello, World!"; // Get the character at index 4 (5th position) let char = text.charAt(4);
	concat([string,,])	Joins specified string literal values (specify multiple strings separated by comma) and returns a new string.	let str1 = "Hello"; let str2 = "World"; // Concatenate strings let combined = str1.concat(", ", str2, "!");

	indexOf(SearchString, Position)	Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found.	let text = "Hello, World!"; // Find the index of the first occurrence of "World" let index = text.indexOf("World");
	lastIndexOf(SearchString, Position)	Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found.	let text = "Hello, World! World!"; // Find the last occurrence of "World" let lastIndex = text.lastIndexOf("World");
	replace(searchValue, replaceValue)	Search specified string value and replace with specified replace Value string and return new string. Regular expression can also be used as searchValue.	let text = "Hello, World!"; // Replace the first occurrence of "World" with "Universe" let newText = text.replace("World", "Universe");
	substr(start, length)	Returns the characters in a string from specified starting position through the specified number of characters (length).	let text = "Hello, World!"; // Get a substring starting from index 7 and of length 5 let substring = text.substr(7, 5);
	substring(start, end)	Returns the characters in a string between start and end indexes.	let text = "Hello, World!"; // Get a substring from index 7 to index 12 (not including 12) let substring = text.substring(7, 12);
	toLowerCase()	Returns lower case string value.	let text = "Hello, World!"; // Convert the string to lowercase let lowerCaseText = text.toLowerCase();
	toUpperCase()	Returns upper case string value.	let text = "Hello, World!"; // Convert the string to uppercase let upperCaseText = text.toUpperCase();
	slice()	Extracts a section of a string and returns a new string.	let text = "Hello, World!"; // Extract substring from index 7 to the end of the string let slicedText = text.slice(7);
	charCodeAt(index)	Returns the Unicode value (UTF-16 code unit) of the character at the specified position (index) in a string.	let text = "Hello, World!"; // Get the Unicode value of the character at index 4 let charCode = text.charCodeAt(4);

			console.log(charCode); // Output: 111 (Unicode of 'o')
	fromCharCode(num1, num2, ...)	Converts one or more Unicode values into a string and returns it.	// Convert Unicode values to a string let text = String.fromCharCode(72, 101, 108, 108, 111); console.log(text); // Output: "Hello"
	split()	Splits a string into an array of substrings based on a separator.	let text = "apple,banana,cherry"; // Split the string into an array using comma as delimiter let fruits = text.split(",");
	includes()	Checks if a string contains a specified substring.	let text = "Hello, World!"; // Check if the string contains "World" let containsWorld = text.includes("World");
	trim()	Removes whitespace from both ends of a string.	let text = " Hello, World! "; // Remove leading and trailing spaces let trimmedText = text.trim();

String Methods Example:

Difference between charCodeAt() and fromCharCode() methods in JavaScript:

Feature	charCodeAt()	fromCharCode()
Operation	Converts a character to its Unicode value	Converts Unicode values to characters
Input	String and an index	Unicode values (numbers)
Output	Unicode value (number)	String of characters
Usage Direction	Works from string → Unicode	Works from Unicode → string

3. Date

The Date object in JavaScript is used to work with dates and times, including operations for getting and setting various date and time components like days, months, years, hours, minutes, seconds, and milliseconds.

Syntax:

- **Create a Date Object with the Current Date and Time**

```
var dt = new Date();
```

new Date() – Creates a Date object representing the current date and time.

Example-1:

Example-2:

Example-3:

- **Create a Date Object Using Milliseconds (Timestamp)**

```
var dt = new Date(milliseconds);
```

new Date(milliseconds) – Creates a Date object from the number of milliseconds since January 1, 1970.

Example-4:

- **Create a Date Object Using a Date String**

```
var dt = new Date('date string');
```

new Date('date string') – Creates a Date object from a string in a recognized date format (ISO 8601, RFC 2822, etc.).

Example-5:

- **Create a Date Object Using Specific Date and Time Components**

```
var dt = new Date(year, month, date, hour, minute, second, millisecond);
```

new Date(year, month, date, hour, minute, second, millisecond) – Creates a Date object using specific components of the date and time. The month is 0-indexed.

Example-6:

Here's a tabular representation of some commonly used JavaScript `Date` methods, with brief descriptions and small examples:

Date Methods	Method	Description	Example
	getDate()	Returns the day of the month (1-31) for the specified date.	let date = new Date(); document.write(date.getDate()); // Example: 15 (if today is the 15th)
	getDay()	Returns the day of the week (0-6) for the specified date (0 = Sunday).	let date = new Date(); document.write(date.getDay()); // Example: 2 (if today is Tuesday)
	getFullYear()	Returns the 4-digit year of the specified date.	let date = new Date(); document.write(date.getFullYear()); // Example: 2024
	getHours()	Returns the hour (0-23) for the specified date.	let date = new Date(); document.write(date.getHours()); // Example: 14 (if it's 2 PM)
	getMilliseconds()	Returns the milliseconds (0-999) for the specified date.	let date = new Date(); document.write(date.getMilliseconds()); // Example: 150
	getMinutes()	Returns the minutes (0-59) for the specified date.	let date = new Date(); document.write(date.getMinutes()); // Example: 45 (if it's 45 minutes past the hour)
	getMonth()	Returns the month (0-11) for the specified date.	let date = new Date(); document.write(date.getMonth()); // Example: 5 (if it's June, as months are 0-indexed)

Date Methods	Method	Description	Example
	getSeconds()	Returns the seconds (0-59) for the specified date.	let date = new Date(); document.write(date.getSeconds()); // Example: 30 (if it's 30 seconds past the minute)
	getTime()	Returns the number of milliseconds since January 1, 1970.	let date = new Date(); document.write(date.getTime()); // Example: 1627605793581 (timestamp)
	setDate()	Sets the day of the month (1-31) for a specified date.	let date = new Date(); date.setDate(25); document.write(date); // Sets the day to 25th of current month
	setFullYear()	Sets the full year (e.g., 2024) for a specified date.	let date = new Date(); date.setFullYear(2025); document.write(date); // Sets year to 2025
	setHours()	Sets the hour (0-23) for a specified date.	let date = new Date(); date.setHours(18); document.write(date); // Sets hour to 6 PM
	setMilliseconds()	Sets the milliseconds (0-999) for a specified date.	let date = new Date(); date.setMilliseconds(500); document.write(date); // Sets milliseconds to 500
	setMinutes()	Sets the minutes (0-59) for a specified date.	let date = new Date(); date.setMinutes(45); document.write(date); // Sets minutes to 45
	setMonth()	Sets the month (0-11) for a specified date.	let date = new Date(); date.setMonth(6); document.write(date); // Sets month to July (7th month)
	setSeconds()	Sets the seconds (0-59) for a specified date.	let date = new Date(); date.setSeconds(30); document.write(date); // Sets seconds to 30
	setTime()	Sets the time (in milliseconds since 1970) for a specified date.	let date = new Date(); date.setTime(1627605793581); document.write(date); // Sets the exact time using timestamp
	toDateString()	Returns the date portion of a Date object as a string.	let date = new Date(); document.write(date.toDateString()); // Example: Thu Dec 13 2024
	toLocaleDateString()	Returns a string representing the date portion of the Date object, formatted according to the locale.	let date = new Date(); document.write(date.toLocaleDateString()); // Example: 12/13/2024 (in US locale)
	toLocaleTimeString()	Returns a string representing the time portion of the Date object, formatted according to the locale.	let date = new Date(); document.write(date.toLocaleTimeString()); // Example: 10:45:30 AM

Date Methods	Method	Description	Example
	toISOString()	Returns the Date object as an ISO-formatted string (e.g., YYYY-MM-DDTHH:MM:SSZ).	let date = new Date(); document.write(date.toISOString()); // Example: 2024-12-13T10:45:30.000Z
	toString()	Returns the full string representation of the Date object.	let date = new Date(); document.write(date.toString()); // Example: Thu Dec 13 2024 10:45:30 GMT+0000 (UTC)
	toUTCString()	Returns the Date object as a UTC string.	let date = new Date(); document.write(date.toUTCString()); // Example: Thu, 13 Dec 2024 10:45:30 GMT
	valueOf()	Returns the primitive value of the Date object (same as getTime()).	let date = new Date(); document.write(date.valueOf()); // Example: 1627605793581 (timestamp)

Example:

4. Window

The window object is a global object in JavaScript that represents the browser window or tab. It provides properties and methods to interact with the browser environment, such as controlling the window, accessing the document, handling events, and more. The window object is automatically created by the browser, and in most browsers, it acts as the global object in a browser environment, meaning variables and functions defined in the global scope are properties of the window object.

Syntax:

```
window.property    // Accessing properties
window.method()    // Calling methods
```

Alternatively, you can omit the window part as it's the default object:

```
property           // Same as window.property
method()           // Same as window.method()
```

Common Properties and Methods of the `window` Object

Window Method	Property/Method	Description	Example
	window.alert()	Displays an alert box with a message and an OK button.	window.alert("Hello, World!"); // Displays an alert box with the message.
	window.confirm()	Displays a dialog box with a message and OK/Cancel buttons. Returns true/false.	let result = window.confirm("Are you sure?"); // Shows a confirmation box and returns true/false.

Window Method	Property/Method	Description	Example
	window.prompt()	Displays a dialog box with a message and a text input field.	let name = window.prompt("Enter your name:"); // Displays a prompt box where user enters their name.
	window.setTimeout()	Executes a function after a specified delay (in milliseconds).	setTimeout(function() { alert("Hello after 3 seconds!"); }, 3000); // Shows an alert after 3 seconds.
	window.open()	Opens a new browser window or tab with a specified URL, name, and features.	let newWindow = window.open("https://www.example.com", "newWindow", "width=400,height=400"); // Opens a new window with specified URL and size.
	window.close()	Closes the current browser window (works only for windows opened via script).	window.close(); // Closes the current window (if it's opened by a script).
	window.document	Refers to the document object, which represents the web page loaded in the window.	console.log(window.document.title); // Prints the title of the document.
	window.innerWidth	Returns the width of the browser window's content area (excluding scrollbars).	console.log(window.innerWidth); // Logs the window's content width in pixels.
	window.innerHeight	Returns the height of the browser window's content area (excluding scrollbars).	console.log(window.innerHeight); // Logs the window's content height in pixels.
	window.location	Represents the current URL of the window.	console.log(window.location.href); // Logs the full URL of the current page.
	window.setInterval()	Repeatedly executes a function at specified intervals (in milliseconds).	setInterval(function() { alert("This will appear every 2 seconds"); }, 2000); // Repeats every 2 seconds.
	window.localStorage	Provides access to local storage, where you can store data on the client side.	window.localStorage.setItem("username", "John"); // Stores data in localStorage.
	window.sessionStorage	Provides access to session storage, which is limited to the current session.	window.sessionStorage.setItem("sessionKey", "value"); // Stores session data that lasts only during the session.
	window.console	Provides access to the browser's console for logging and debugging.	window.console.log("This is a log message"); // Logs a message to the console.

Example:

5. Navigator

The JavaScript Navigator Object provides information about the web browser being used by the client. It's commonly used for detecting browser details such as the name, version, user agent, and whether certain browser features are supported.

The navigator object is the window property, so it can be accessed by:

Syntax:

window.navigator (OR) navigator

Properties of JavaScript navigator object:

There are many properties of navigator object that returns information of the browser.

Properties	Description
appcodename	specifies the code name of the browser (*experimental property - can return incorrect value*)
Appname	specifies the name of the browser (*experimental property - can return incorrect value*)
Appversion	specifies the version of browser being used (*experimental property - can return incorrect value*)
cookieEnabled	specifies whether cookies are enabled or not in the browser
Platform	contains a string indicating the machine type for which the browser was compiled.
Useragent	contains a string representing the value of **user-agent header** sent by the client to server in HTTP protocol
Geolocation	returns object of GeoLocation which can be used to get the location information of the device.
Online	specifies whether the browser is online or not.
language	returns a string with the preferred browser language, for example, **en-US** for English.
languages	returns a string with all the languages supported by the browser in order of user preference.

Example of Properties:

Here is a comprehensive table of all the methods available in the `navigator` object, including a brief description of each method:

Navigator Methods	Method	Description	Example
	navigator.geolocation.getCurrentPosition()	Retrieves the geographical position of the device.	navigator.geolocation.getCurrentPosition(successCallback, errorCallback)
	navigator.geolocation.watchPosition()	Watches the position of the device and updates the position as it changes.	navigator.geolocation.watchPosition(successCallback, errorCallback)
	navigator.geolocation.clearWatch()	Stops the continuous monitoring of the user's location initiated by watchPosition().	navigator.geolocation.clearWatch(watchId)
	navigator.registerProtocolHandler()	Registers a web app to handle a specific protocol (e.g., mailto, tel). *Deprecated in most browsers.*	navigator.registerProtocolHandler("mailto", "mailto:%s", "My Email App")
	navigator.requestMIDIAccess()	Provides access to the Web MIDI API, allowing interaction with MIDI devices.	navigator.requestMIDIAccess().then(successCallback).catch(errorCallback)

Navigator Methods	Method	Description	Example
	navigator.vibrate()	Makes the device vibrate for a specified duration.	navigator.vibrate(1000) or navigator.vibrate([1000, 500, 1000])
	navigator.getBattery()	Returns a BatteryManager object that provides battery information such as level and charging state. *Deprecated in some browsers.*	navigator.getBattery().then(function(battery) {console.log(battery.level);})
	navigator.connection	Provides information about the network connection (such as type and speed). *Network Information API.*	navigator.connection.effectiveType or navigator.connection.downlink
	navigator.onLine	Returns whether the browser is online or offline.	navigator.onLine
	navigator.mediaDevices.getUserMedia()	Requests access to the user's camera and microphone.	navigator.mediaDevices.getUserMedia({ video: true, audio: true }).then(successCallback).catch(errorCallback)
	navigator.getGamepads()	Returns the list of gamepads connected to the device.	navigator.getGamepads()

Example of methods:

6. Document

What is the Document Object?

- ☑ The **document object** is part of the **Browser Object Model (BOM)** and the **DOM (Document Object Model)**.
- ☑ It represents the entire HTML document (the web page) loaded into the browser.
- ☑ The **document object** allows access, manipulation, and modification of the structure and content of an HTML page.

Relationship to the Window Object:

- ☑ The **document object** is a **child object** of the window object.
- ☑ You can access it as:

```
window.document // Explicit
document        // Shorthand
```

Hierarchy:

The document object contains the **HTML document structure** as a tree of elements:

- <html> (root element)
- <head> (head section)
- <body> (body section)
- Child tags like <p>, , <h1>, etc.

Document Object Purpose:

It provides methods and properties to access and manipulate the HTML document.

You can:

- Retrieve elements (getElementById, querySelector).
- Modify content (innerHTML, textContent).
- Add or remove elements.
- Respond to events.

Ways to Access the Document Object

Using document Directly:

This is the most common way

```
console.log(document.title);  // Outputs the document's title
```

Using window.document:

This is equivalent but explicit:

```
console.log(window.document.title);  // Same result
```

Example:

Innerhtmlexample:

Properties	Description
Cookie	returns a report that contains all the visible and unexpired cookies associated with the document
Domain	returns the domain name of the server from which the document has originated
lastModified	returns the date on which document was last modified
documentMode	returns the mode used by the browser to process the document
readyState	returns the loading status of the document.
Referrer	returns the URL of the documents referred to in an HTML document
Title	returns the name of the HTML document defined between the starting and ending tags of the TITLE element
URL	returns the full URL of the HTML document.

Document Properties Example:

Methods of the document Object

The document object represents the web page's content and provides methods for interacting with the DOM (Document Object Model).

Method Name	Description	Example
getElementById()	Returns the element with the specified id attribute.	let el = document.getElementById('myId'); → Returns the element with id="myId"

Method Name	Description	Example
getElementsByName()	Returns a **NodeList** of all elements with the specified name attribute.	let elements = document.getElementsByName('username'); // → Returns a NodeList of all elements with the name attribute 'username'
getElementsByClassName()	Returns a collection of elements with the specified class name.	let elements = document.getElementsByClassName('myClass'); → Returns a live HTMLCollection of elements
getElementsByTagName()	Returns a collection of elements with the specified tag name.	let divs = document.getElementsByTagName('div'); → Returns a live HTMLCollection of elements
querySelector()	Returns the first element that matches a CSS selector.	let el = document.querySelector('.myClass'); → Returns the first element with class myClass
querySelectorAll()	Returns a static NodeList of all elements that match a CSS selector.	let els = document.querySelectorAll('div.myClass'); → Returns a NodeList of all with class myClass
write("string")	Writes a string of text directly to the document at the time the script is executed.	document.write(" Hello, World! "); // → Dynamically inserts " Hello, World! " into the document
writeln("string")	Similar to write() but appends a newline (\n) character to the written string (visible in the HTML source code).	document.writeln(" Hello, World! "); // → Adds " Hello, World! " with a newline in the source code
createElement()	Creates a new element node with the specified tag name.	let div = document.createElement('div'); → Creates a new element
createTextNode()	Creates a new text node containing the specified text.	let textNode = document.createTextNode('Hello, world!'); → Creates a text node with the specified content
appendChild()	Adds a new child node to the end of the list of children of a specified parent node.	document.body.appendChild(div); → Appends the newly created to the body
removeChild()	Removes a child node from a specified parent node.	document.body.removeChild(div); → Removes the from the body
replaceChild()	Replaces a child node with a new node.	document.body.replaceChild(newDiv, oldDiv); → Replaces oldDiv with newDiv in the body
insertBefore()	Inserts a new child node before an existing child node within a parent node.	parent.insertBefore(newNode, referenceNode); → Inserts newNode before referenceNode in parent node
setAttribute()	Sets a specified attribute to a specified value for the element.	el.setAttribute('class', 'myClass'); → Sets the class attribute to myClass for the element el
getAttribute()	Gets the value of a specified attribute on the element.	el.getAttribute('class'); → Gets the value of the class attribute of element el

Method Name	Description	Example
removeAttribute()	Removes a specified attribute from an element.	el.removeAttribute('class'); → Removes the class attribute from element el
open()	Opens a new document stream (used in document.write() method).	document.open(); → Opens a document stream for further writing
close()	Closes the document stream and renders the content to the browser.	document.close(); → Closes the document stream and renders the content
getElementsByName()	Returns a collection of elements with the specified name attribute.	let inputs = document.getElementsByName('username'); → Returns a collection of elements with name username
focus()	Focuses on an element, making it active.	let el = document.getElementById('inputField'); el.focus(); → Focuses on the element with id="inputField"

Example of document object methods:

Conversion of value k = "2" to integer 2 in JavaScript:

Program:

Explanation:

1. **HTML Structure**:
   - The page contains a heading (<h2>) to introduce the topic.
   - A paragraph (<p>) with the id="result" is where the results of the conversions will be displayed.
   - A button (<button>) triggers the demonstrateConversion() JavaScript function to perform all the conversions and display the results.
2. **JavaScript Code**:
   - The string k = "2" is declared initially to simulate the conversion of a string to an integer.
   - Four methods are demonstrated for converting the string to an integer:

i) Using parseInt():

```
let num1 = parseInt(k);
let result1 = "Using parseInt: " + num1;
```

parseInt(k) converts the string "2" to the integer 2. It stops parsing as soon as it encounters a non-numeric character, making it suitable for extracting integers from strings.

ii) Using Unary Plus (+):

```
let num2 = +k;
let result2 = "Using Unary Plus: " + num2;
```

The unary plus (+k) is a concise way to convert a string to a number. It automatically coerces the string into a number (integer or float, depending on the value).

iii) Using Number():

```
let num3 = Number(k);
let result3 = "Using Number(): " + num3;
```

Number(k) converts the string "2" into the number 2. This method works for both integers and floating-point numbers and is more explicit.

iv) Using Math.floor(parseFloat()):

```
let kFloat = "2.5";
let num4 = Math.floor(parseFloat(kFloat));
let result4 = "Using Math.floor(parseFloat()): " + num4;
```

This example demonstrates converting a float string ("2.5") to an integer using Math.floor(), which rounds the number down to the nearest integer.

3. **Displaying Results**:

The document.getElementById("result").innerHTML updates the contents of the paragraph with the id="result" to display the results of all four conversions.

User Defined Functions in JavaScript:

In JavaScript, user-defined functions are functions that you create to perform specific tasks. You define a function using the function keyword, followed by the function name, parentheses for parameters, and curly braces {} for the function body containing the code to be executed.

Steps to Define and Call a User-Defined Function:

- Define the function using the function keyword.
- Call the function by using its name and passing any required arguments (parameters).

Illustration of JavaScript code to test whether a character accepted is a vowel or not?

Explanation:

Explanation of the Code:

7. HTML Structure:

☑ **Input field:**

- The <input> element with the id="charInput" allows the user to enter a single character.
- The maxlength="1" attribute ensures that the user can only enter one character.

☑ **Button:**

- The <button> element triggers the checkVowel() function when clicked.

☑ **Result paragraph:**

- The <p id="result"> element is used to display the result, whether the character is a vowel or not.

8. JavaScript Code:

checkVowel() function: This function is called when the user clicks the "Check if Vowel" button.

Step 1: Get the input character:

```
let char = document.getElementById("charInput").value;
```

We use document.getElementById("charInput").value to get the value entered by the user in the input field and store it in the variable char.

Step 2: Validate if the input is a single character:

We check if the input contains exactly one character using char.length. If the length is not 1, an error message is displayed, and the function exits using return.

```
if (char.length !== 1) {
    document.getElementById("result").textContent = "Please enter exactly one character.";
 return;
}
```

Step 3: Convert the character to lowercase:

The toLowerCase() method converts the entered character to lowercase. This ensures the comparison is case-insensitive (i.e., it works for both uppercase and lowercase vowels).

```
char = char.toLowerCase();
```

Step 4: Check if the character is a vowel:

```
if (char === 'a' || char === 'e' || char === 'i' || char === 'o' || char === 'u') {
    document.getElementById("result").textContent = char + " is a vowel.";
} else {
    document.getElementById("result").textContent = char + " is not a vowel.";
}
```

The program checks if the lowercase version of char is one of the vowels ('a', 'e', 'i', 'o', or 'u') using a series of if conditions.

If the character is a vowel, it displays a success message (char is a vowel).

If the character is not a vowel, it displays a failure message (char is not a vowel).

JavaScript Form Validations:

JavaScript validation form to raise the following conditions:

I) Length of the username must be 7

II) Password and confirm password must be same

III) Phone number must be a number? (Both HTML and JavaScript)

Program

Explanation:

HTML Structure:

- Username: A text input for the username with a placeholder that specifies it should be 7 characters.
- Password & Confirm Password: Password fields to ensure that both passwords match.
- Phone Number: A text input for the phone number, which will be validated to ensure it's a number.
- Submit Button: A button that triggers the form validation.

JavaScript Validation:

- Username Validation: Checks if the username is exactly 7 characters long.
- Password and Confirm Password Validation: Ensures that the password and confirm password fields match.
- Phone Number Validation: Ensures that the phone number input is a valid number using isNaN() to check for non-numeric input.

Error Messages:

- The error messages are displayed in the <p> tag with the ID errorMessages. If any validation fails, it will show the specific message.
- If all validations pass, a success message is displayed using alert().

Validation for the following conditions in JavaScript:

i) At least 1 special character should be there.

ii) 1st letter must be capital?

Explanation:

At least 1 Special Character Should Be There

Special characters can include characters like !, @, #, $, %, ^, etc. To check if a string contains at least one special character, we can use a regular expression like this:

```
/[!@#$%^&*(),.?":{}|<>]/
```

This regular expression checks if any of the characters inside the square brackets are present in the string.

The First Letter Must Be Capitalized

To ensure that the first letter of a string is capitalized, we can use the following condition:

```
/^[A-Z]/
```

This checks if the first character of the string is an uppercase letter (from A to Z).

Program

Explanation of the Code:

1. **HTML:**
   - Input field: The user can input a string that will be validated.
   - Button: When clicked, it triggers the validateString() function to perform the validation.
2. **JavaScript:**
   - hasSpecialChar: This variable checks if the string contains at least one special character using the regex /[!@#$%^&*(),.?":{}|<>]/.
   - startsWithCapital: This variable checks if the first character of the string is uppercase using the regex /^[A-Z]/.
   - validateString(): This function performs both checks and generates a result message, which is displayed to the user.
3. **Result:**
   - If both conditions are met (special character and capitalized first letter), the message will say "valid." If either condition is not met, an appropriate message will be displayed explaining what is missing.

Methods in the history object in JavaScript:

The history object in JavaScript is part of the Window interface and allows manipulation of the browser's session history. Below is a summary of the most commonly used methods in the history object:

☑ **history.back()**

Description: Navigates to the previous page in the session history (equivalent to clicking the back button in the browser).

Example:

```
history.back();
```

☑ **history.forward()**

Description: Navigates to the next page in the session history (equivalent to clicking the forward button in the browser).

Example:

```
history.forward();
```

☑ **history.go()**

Description: Navigates to a specific page in the session history, relative to the current page.

Pass a positive number to move forward.

Pass a negative number to move backward.

Pass 0 to reload the current page.

Example:

```
history.go(-1); // Go back one page
history.go(1);  // Go forward one page
```

☑ **history.pushState()**

Description: Adds a new entry to the session history stack without reloading the page.

You can also pass state data and a URL to associate with the new entry.

Parameters:

state: A state object associated with the new history entry.

title: (optional) Title of the new history entry (often ignored).

url: (optional) The new URL to associate with the state.

Example:

```
history.pushState({ page: 1 }, "Title", "/page1");
```

☑ **history.replaceState()**

Description: Updates the current entry in the session history stack without creating a new entry.

Similar to pushState, but does not add a new history entry.

Parameters:

state: A state object associated with the history entry.

title: (optional) Title of the history entry (often ignored).

url: (optional) The new URL to associate with the state.

Example:

```
history.replaceState({ page: 2 }, "Title", "/page2");
```

☑ **history.length (Property, not a method)**

Description: Returns the number of entries in the browser's history stack.

Example:

```
console.log(history.length);
```

☑ **history.scrollRestoration**

Description: Controls whether the browser restores the scroll position when navigating through the history.

Values can be "auto" (default) or "manual".

Example:

```
history.scrollRestoration = "manual";
```

Example:

eval() method in JavaScript:

The eval() method in JavaScript is used to execute a string of JavaScript code. It parses the string as code and runs it. However, its use is discouraged because it can lead to security and performance issues. It is best avoided unless absolutely necessary and used in controlled environments.

Example:

Difference between floor() and ceil() methods in JavaScript:

The Math.floor() and Math.ceil() methods in JavaScript are used to round numbers, but they behave differently:

1. Math.floor()

Description: Rounds a number down to the nearest integer, toward negative infinity.

Example:

```
console.log(Math.floor(4.7));  // Output: 4
console.log(Math.floor(4.3));  // Output: 4
console.log(Math.floor(-4.7)); // Output: -5
console.log(Math.floor(-4.3)); // Output: -5
```

2. Math.ceil()

Description: Rounds a number up to the nearest integer, toward positive infinity.

Example:

```
console.log(Math.ceil(4.7));  // Output: 5
console.log(Math.ceil(4.3));  // Output: 5
console.log(Math.ceil(-4.7)); // Output: -4
console.log(Math.ceil(-4.3)); // Output: -4
```

Example:

Aspect	Math.floor()	Math.ceil()
Behavior	Rounds **down** toward negative infinity	Rounds **up** toward positive infinity

Aspect	Math.floor()	Math.ceil()
Positive Numbers	Returns the integer **below or equal** to the number	Returns the integer **above or equal** to the number
Negative Numbers	Moves farther away from zero	Moves closer to zero

Display Prime Numbers Up to a User-Entered Number Using JavaScript

Code Explanation:

Input Field and Button:

```
<label for="numberInput">Enter a number:</label>
<input type="number" id="numberInput" placeholder="Enter a number" min="1">
<button onclick="displayPrimes()">Submit</button>
```

☑ <label>: Associates the text with the input field for accessibility.

☑ <input type="number" id="numberInput" placeholder="Enter a number" min="1">:

- Creates a numeric input field.
- id="numberInput": Allows the field to be targeted by JavaScript.
- placeholder="Enter a number": Displays helper text inside the field.
- min="1": Restricts the input to numbers >= 1.

☑ <button onclick="displayPrimes()">Submit</button>:

- A button that triggers the displayPrimes() JavaScript function when clicked.

Placeholder for Results:

```
<h3>Prime Numbers:</h3>
<p id="result"></p>
```

☑ <h3>: Section heading for results.

☑ : Placeholder paragraph where the prime numbers will be displayed.

JavaScript Logic

Prime Number Check Function:

```
function isPrime(num) {
    if (num < 2) return false; // Numbers less than 2 are not prime
    for (let i = 2; i <= Math.sqrt(num); i++) {
        if (num % i === 0) return false; // Not prime if divisible by another number
    }
    return true; // Prime if no divisors are found
}
```

Purpose: Determines if a given number is prime.

Steps:

☑ Check if num < 2. If so, it's not a prime number.

☑ Loop from 2 to Math.sqrt(num):

- If num is divisible by any number in this range, it's not prime.
- The loop range is optimized to Math.sqrt(num) because divisors repeat after the square root.

☑ If no divisors are found, return true.

Display Primes Function:

```
// Function to display prime numbers up to the given number
function displayPrimes() {
    // Get the input value
    const inputNumber = parseInt(document.getElementById("numberInput").value);

    // Validate input
    if (isNaN(inputNumber) || inputNumber < 1) {
        document.getElementById("result").textContent = "Please enter a valid positive
            number.";
        return;
    }

    // Generate prime numbers up to the input value
    let primes = [];
    for (let i = 2; i <= inputNumber; i++) {
        if (isPrime(i)) {
            primes.push(i);
        }
    }

    // Display the result
    document.getElementById("result").textContent = primes.join(", ");
}
```

Steps:

☑ Get User Input:

- Retrieves the value of the input field using document.getElementById("numberInput").value.
- Converts it to an integer with parseInt().

☑ Input Validation:

- If the input is not a valid number or is less than 1, display an error message in the <p> element with id="result".

☑ Generate Prime Numbers:

- Loop from 2 to the user-provided number.
- Use the isPrime() function to check each number.
- Add prime numbers to an array (primes).

☑ Display Results:

- Convert the array of primes to a comma-separated string using primes.join(", ").
- Display the string in the <p> element with id="result".

Selecting the data from a text box in JavaScript when the text box is identified with the ID attribute

To select data from a text box in JavaScript when the text box is identified by the id attribute, you can use the following steps:

☑ Use the document.getElementById method to reference the text box element by its id.

☑ Access the value property of the text box to retrieve its content.

HTML events and their event handlers:

Mouse events:

Mouse events are triggered when the user interacts with an element using the mouse.

Event	Event Handler	Description
Click	onclick	Triggered when the user clicks on an element.
Mouseover	onmouseover	Triggered when the mouse cursor hovers over an element.
Mouseout	onmouseout	Triggered when the mouse cursor leaves an element.
Mousedown	onmousedown	Triggered when the mouse button is pressed.
Mouseup	onmouseup	Triggered when the mouse button is released.
Mousemove	onmousemove	Triggered when the mouse is moved over an element.

Keyboard events:

Keyboard events are triggered when the user interacts with the keyboard.

Event	Event Handler	Description
Keydown	onkeydown	Triggered when a key is pressed down.
Keyup	onkeyup	Triggered when a key is released.

Form events:

Form events are triggered when the user interacts with a form.

Event	Event Handler	Description
Focus	onfocus	Triggered when an input element gains focus.

Event	Event Handler	Description
Submit	onsubmit	Triggered when the form is submitted.
Blur	onblur	Triggered when an input element loses focus.
Change	onchange	Triggered when the value of a form element is changed.

Window/Document events:

These events are triggered at the window or document level.

Event	Event Handler	Description
Load	onload	Triggered when the page or resource finishes loading.
Unload	onunload	Triggered when the user leaves the page (unloads it).
Resize	onresize	Triggered when the browser window is resized.

Example:

LAB PROGRAMS

Week 5: JavaScript Functions

1. **Write a JavaScript program to demonstrate pop-up boxes (alert, prompt, confirm).**
2. **Write a JavaScript program to check if a given year is a leap year.**
3. **Write a JavaScript program to generate all prime numbers up to a given number (include a textbox to accept input and a submit button).**
4. **Write a JavaScript program to display a given number in reverse order using a function.**

   Program-1:

   Program-2:
5. **Write a JavaScript program to check whether a given number is an Armstrong number.**
6. **Write a JavaScript program to demonstrate linear search.**

   ☑ **Program-1**

   ☑ **Program-2**

Week 6: JavaScript Validations

1. **Write JavaScript to validate the following fields in a designed registration form:**
   - **Name (must contain alphabets and have a length of at least 7 characters).**
   - **Password (minimum length of 6 characters).**
   - **Password and confirm password must match.**
   - **Email ID (must follow the standard pattern `name@domain.com`).**
   - **Phone number (must be numeric and exactly 10 digits long).**
   - **Date of birth (must not exceed the current year).**
   - **Address (must not be empty).**

   Program
2. **Write JavaScript to validate email addresses with conditions:**
   - **At least one `@` symbol.**
   - **At least two dots (`.`).**
   - **No consecutive dots or `@`.**
   - **The first and last characters cannot be `.` or `@`.**

   Program