JAVA SCRIPT

JavaScript: Why we need JavaScript, What is JavaScript, Environment Setup, Working with Identifiers, Type of Identifiers, Primitive and Non Primitive Data Types, Operators and Types of Operators, Types of Statements, Non - Conditional Statements, Types of Conditional Statements, If and Switch Statements, Types of Loops, Types of Functions, Declaring and Invoking Function, Arrow Function, Function Parameters, Nested Function, Built-in Functions, Variable Scope in Functions, Working With Classes, Creating and Inheriting Classes, In-built Events and Handlers, Working with Objects, Types of Objects, Creating Objects, Combining and cloning Objects using Spread operator, Destructuring Objects, Browser and Document Object Model, Creating Arrays, Destructuring Arrays, Accessing Arrays, Array Methods, Introduction to Asynchronous Programming, Callbacks, Promises, Async and Await, Executing Network Requests using Fetch API, Creating and consuming Modules.

1. Why We Need JavaScript:

JavaScript is an essential language for modern web development because it brings websites to life. Here's why:

- **Interactivity:** JavaScript allows you to create dynamic and engaging user experiences.
    - Imagine a website where you can click buttons, fill out forms, and see immediate responses.
    - JavaScript makes this possible, creating a more interactive and enjoyable experience for users.
- **Dynamic Content:** JavaScript can manipulate the content of a web page after it has been loaded.
    - This means you can update parts of a page without needing to reload the entire thing.
    - For example, displaying search results, changing content based on user actions, or creating animations.
- **Client-Side Scripting:** JavaScript runs directly in the user's web browser, reducing the load on the server.
    - This makes websites faster and more responsive.

- **Versatility:** JavaScript is not limited to just web browsers.
    - It can also be used to build server-side applications (with Node.js), mobile apps, and even desktop applications.

Here are some specific examples of what JavaScript can do:

- **Create interactive forms:** Validate user input, provide real-time feedback, and submit data.
- **Build dynamic user interfaces:** Create dropdown menus, modal windows, and other interactive elements.
- **Handle events:** Respond to user actions like mouse clicks, key presses, and page scrolling.
- **Make websites more visually appealing:** Create animations, transitions, and other visual effects.
- **Develop complex web applications:** Build single-page applications (SPAs) and other advanced web applications.

2 What is JavaScript?

JavaScript is a programming language that developers use to make interactive webpages. From refreshing social media feeds to displaying animations and interactive maps, JavaScript functions can improve a website's user experience. As a client-side scripting language, it is one of the core technologies of the World Wide Web. For example, when browsing the internet, anytime you see an image Scrolling, a click-to-show dropdown menu, or dynamically changing element colors on a webpage, you see the effects of JavaScript.

2.1 How does Java Script

All programming languages work by translating English-like syntax into machine code, which the operating system then runs. JavaScript is broadly categorized as a scripting language, or an interpreted language. JavaScript code is interpreted—that is, directly translated into underlying machine language code by a JavaScript engine.

2.2 Java script Engine

A JavaScript engine is a computer program that runs JavaScript code. The first JavaScript engines were mere interpreters, but all modern engines use just-in-time or runtime compilation to improve performance.

2.3 Client-side Java Script

Client-side JavaScript refers to the way JavaScript works in your browser. In this case, the JavaScript engine is inside the browser code. All major web browsers come with their own built-in JavaScript engines.

Web application developers write JavaScript code with different functions associated with various events, such as a mouse click or mouse hover. These functions make changes to the HTML and CSS.

Here is an overview of how client-side JavaScript works:

1. The browser loads a webpage when you visit it.

2. During loading, the browser converts the page and all its elements, such as buttons, labels, and dropdown boxes, into a data structure called the Document Object Model (DOM).

3. The browser's JavaScript engine converts the JavaScript code into bytecode. This code is an intermediary between the JavaScript syntax and the machine.

4. Different events, such as a mouse click on a button, trigger the execution of the associated JavaScript code block. The engine then interprets the bytecode and makes changes to the DOM.

5. The browser displays the new DOM.

2.4 Server-side JavaScript

Server-side JavaScript refers to the use of the coding language in back-end server logic. In this case, the JavaScript engine sits directly on the server. A server-side JavaScript function can access the database, perform different logical operations, and respond to various events triggered by the server's operating system. The primary advantage of server-side scripting is that you can highly customize the website response based on your requirements, your access rights, and the information requests from the website.

3 Java Script Environment Setup

1. Install Node.js and npm

- **Node.js:** This is the runtime environment that allows you to execute JavaScript code outside of a web browser. It also comes bundled with:
    - **npm (Node Package Manager):** [1] This is a package manager for JavaScript. It allows you to easily install, update, and manage third-party libraries and tools for your projects.

Download and Install:

- Visit the official Node.js website (nodejs.org) and download the installer for your operating system (Windows, macOS, or Linux).
- Follow the on-screen instructions to install Node.js. This will typically also install npm along with it.

Verify Installation:

- Open your terminal or command prompt.
- Type node -v and press Enter. You should see the installed Node.js version.
- Type npm -v and press Enter. You should see the installed npm version.

3.1 Java script frameworks

Front-End Frameworks:

- **React:** A component-based library for building user interfaces, known for its flexibility and large ecosystem.
- **Angular:** A comprehensive framework for building dynamic web applications, favored for its robust features and enterprise-level support.
- **Vue.js:** A progressive framework that balances flexibility and opinionated structure, making it easy to learn and integrate.

Back-End Frameworks:

- **Node.js:** A runtime environment that allows you to execute JavaScript on the server-side, enabling the creation of scalable and high-performance web servers.
- **Express.js:** A minimal and flexible Node.js framework for building web applications and APIs.
- **NestJS:** A progressive Node.js framework for building efficient, scalable, and enterprise-grade server-side applications

4 Identifiers in JavaScript

In JavaScript, identifiers are used to name variables, functions, and other entities within your code. Here's a breakdown of the rules and best practices:

Rules for Valid Identifiers

- **Case-Sensitive:** JavaScript is case-sensitive. myVariable and myvariable are considered different identifiers.
- **Start with a Letter, Underscore, or Dollar Sign:**
    - myVariable, _myVariable, $myVariable are valid.
    - 123variable is invalid.
- **Consist of Letters, Digits, Underscores, and Dollar Signs:**
    - myVariable123, _myVariable, $variable_1 are valid.
    - my-variable (hyphen) is invalid.
- **Reserved Keywords:** You cannot use JavaScript keywords as identifiers (e.g., if, else, for, var, let, const).

Examples

- **Valid Identifiers:**
    - firstName
    - _lastName
    - $amount
    - customerAge
    - isCustomerActive
- **Invalid Identifiers:**
    - 1stName (starts with a number)
    - my-variable (contains a hyphen)
    - function (reserved keyword)

By following these guidelines, you can write more readable, maintainable, and less error-prone JavaScript code.

Types of Identifiers

In JavaScript, identifiers are names used to identify variables, functions, classes, and other objects. Here's a breakdown of the types of identifiers:

1. Variable Identifiers:

- These are the names given to variables that store data.
- Example: let age = 25; (Here, age is the variable identifier)

2. Function Identifiers:

- These are the names given to functions, which are reusable blocks of code.
- Example: function calculateArea(width, height) { ... } (Here, calculateArea is the function identifier)

3. Class Identifiers:

- These are the names given to classes, which are blueprints for creating objects.
- Example: class Person { ... } (Here, Person is the class identifier)

4. Object Property Identifiers:

- These are the names used to identify properties within an object.
- Example: const person = { name: "Alice", age: 30 }; (Here, name and age are object property identifiers)

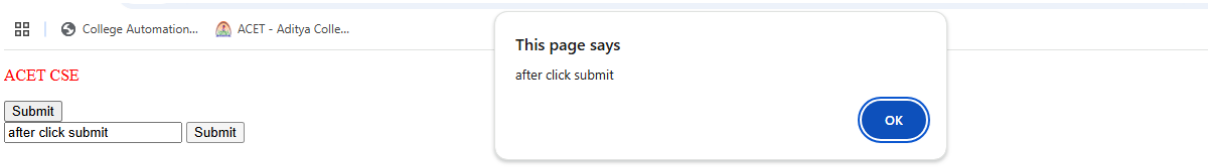
Example Program of JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
        function change_color()
        {
    var element = document.getElementById("ptag");
    element.style.color="red";
        }

       function show_userinput()
       {
            var user_name = document.getElementById("txtusername").value;
            alert(user_name);
       }
    </script>
</head>
<body>
    <p id="ptag"> ACET CSE  </p>
    <input type="submit" onclick="change_color();"/>
```

```
    </br>
    <input type="text" id="txtusername"/>
    <input type="submit" onclick="show_userinput();"/>

</body>
</html>
```

Output :

Before click submit button



5 Primitive and Non Primitive types

Primitive Data Types

- **Definition:** These are the basic building blocks of data in programming languages. They represent single values of a specific type.

Non-Primitive Data Types (Reference Types)

- **Definition:** These are more complex data structures that can hold multiple values or collections of values.

Primitive Data Types:

- **Number:** Represents numerical values (e.g., 10, 3.14, -5, Infinity, NaN).
- **String:** Represents a sequence of characters (e.g., "Hello", 'world', ``).
- **Boolean:** Represents a logical entity (either true or false).
- **Null:** Represents the intentional absence of any value (e.g., null).

- **Undefined:** Represents a variable that has been declared but has not been assigned a value (e.g., let myVar;).
- **Symbol:** Represents a unique and immutable value (introduced in ES6).

Non-Primitive Data Types (Objects):

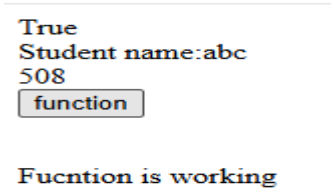
- **Object:** The most fundamental building block for creating more complex data structures. Can hold a collection of key-value pairs.
- **Array:** An ordered collection of values (can be of different data types).
- **Function:** A block of code designed to perform a specific task.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script>
   //
    let a=10;
    let b="acet";
    let d=true;
    if(d)
    {
        document.writeln("True");
    }
   else
   {
    document.writeln("False");
   }
   //Objects
   const Student = {number:508,name:"abc",dept:"cse"};
   document.writeln( "</br> Student name:" +  Student.name)
   // Array
   let student = [508,"abc","cse"];
   document.writeln("</br>" + student[0]);
   // function
   function test_function()
   {
    document.getElementById("demo").innerHTML="</br>" + "Fucntion is working";
   }
</script>
</head>
```

```
<body>
</br>
   <input type="submit" onclick="test_function();" value="function" />
    <p id="demo">
    </p>
</body>
</html>
```

Output



6 Operators

In programming, operators are symbols or keywords that perform specific operations on one or more values (called operands). They are essential for manipulating data and controlling the flow of a program.

Types of Operators

Here are some common types of operators:

1. **Arithmetic Operators:**
   - Perform basic mathematical operations.
   - Examples: + (addition), - (subtraction), * (multiplication), / (division), % (modulus - remainder after division)
2. **Relational Operators:**
   - Compare values and return a boolean result (true or false).
   - Examples: == (equal to), != (not equal to), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to)
3. **Logical Operators:**
   - Combine boolean expressions.
   - Examples: && (AND), || (OR), ! (NOT)
4. **Bitwise Operators:**
   - Operate on individual bits of data.
   - Examples: & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), << (left shift), >> (right shift)
5. **Assignment Operators:**
   - Assign values to variables.
   - Examples: = (simple assignment), += (add and assign), -= (subtract and assign), *= (multiply and assign), /= (divide and assign), %= (modulus and assign)
6. **Increment/Decrement Operators:**

- Increase or decrease the value of a variable by 1.
- Examples: ++ (increment), -- (decrement)

7. **Conditional (Ternary) Operator:**
   - A shorthand way to write an if-else statement.
   - Example: condition ? value_if_true : value_if_false

Example Program:

```
x = 10
y = 5

# Arithmetic operators
sum = x + y  # sum will be 15
difference = x - y  # difference will be 5
product = x * y  # product will be 50
quotient = x / y  # quotient will be 2.0
remainder = x % y  # remainder will be 0

# Relational operators
is_equal = (x == y)  # is_equal will be False
is_not_equal = (x != y)  # is_not_equal will be True
is_greater = (x > y)  # is_greater will be True

# Logical operators
both_true = (x > 0 and y > 0)  # both_true will be True
either_true = (x > 0 or y < 0)  # either_true will be True
not_true = not (x == y)  # not_true will be True

# Assignment operators
x += 5  # x will now be 15
y -= 2  # y will now be 3

# Increment/decrement operators
x++  # x will now be 16
y--  # y will now be 2

# Conditional operator
result = "x is greater" if x > y else "y is greater"
```

7 Conditional Statements

Conditional statements in JavaScript allow you to control the flow of your program based on specific conditions. They enable you to execute different blocks of code depending on whether a certain condition is true or false. Here are the primary conditional statements

If : Executes a block of code only if a specific condition is true

```
if (condition) {
  // Code to be executed if the condition is true
}
```

Example on If :

```
let age = 25;
if (age >= 18) {
  console.log("You are an adult.");
}
```

If ……. Else : Executes one block of code if the condition is true, and another block if it's false.

```
if (condition) {
  // Code to be executed if the condition is true
} else {
  // Code to be executed if the condition is false
}
```

If….else Example

```
let isRaining = true;

if (isRaining) {
  console.log("Take an umbrella.");
} else {
  console.log("Enjoy the sunshine!");}
```

`If...else if...else` statement: Handles multiple conditions sequentially. If the first condition is true, its block executes. If not, the next else if condition is checked, and so on. If none of the conditions are true, the else block (if present) executes.

```
if (condition1) {
  // Code to be executed if condition1 is true
} else if (condition2) {
  // Code to be executed if condition1 is false and condition2 is true
} else {
  // Code to be executed if neither condition1 nor condition2 is true
}
```

`If...else if...else` statement example:

```
let score = 85;
if (score >= 90) {
  console.log("Excellent!");
} else if (score >= 80) {
  console.log("Great job!");
} else {
  console.log("Good effort.");
}
```

Switch statement: Evaluates an expression and compares it to multiple cases. If a match is found, the corresponding block of code executes.

```
switch (expression) {
  case value1:
    // Code to execute if expression matches value1
    break;
  case value2:
    // Code to execute if expression matches value2
    break;
  // ... more cases
  default:
    // Code to execute if none of the cases match}
```

Switch Example:

```
let dayOfWeek = "Sunday";
switch (dayOfWeek) {
  case "Saturday":
  case "Sunday":
    console.log("Weekend!");
    break;
  case "Monday":
  case "Tuesday":
  case "Wednesday":
  case "Thursday":
  case "Friday":
    console.log("Weekday.");
    break;
  default:
    console.log("Invalid day.");
}
```

8 Types loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- for/of - loops through the values of an iterable object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true

For : The for statement creates a loop with 3 optional expressions:

1. **Expression 1** is executed (one time) before the execution of the code block.
2. **Expression 2** defines the condition for executing the code block.
3. **Expression 3** is executed (every time) after the code block has been executed.

Example program:

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

<h2>JavaScript Program For First 10 Numbers:</h2>

<p id="demo"></p>

 <script>
   let text=" ";
   for(let i=0; i<5; i++)
   {
    text+="The number is "+ i + "<br>";
   }
   document.getElementById("demo").innerHTML=text;

 </script>

</body>
</html>
```

Output:

JavaScript For Loop

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4

For-in: The JavaScript for in statement loops through the properties of an Object:

- The **for in** loop iterates over a **student** object
- Each iteration returns a **key** (x)
- The key is used to access the **value** of the key
- The value of the key is student**[x]**

```
const student = {name:"John", dept:"cse", age:25};
let obj = "";
for (let x in student) {
obj += student[x] + " " + "<br>" ;}
document.getElementById("object").innerHTML = obj;
```

oupt put:

For-of:

The JavaScript for of statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

```
let person = [101,"ravi",5000];
let arr="";
for ( let y of person)
{

  arr += y  + "<br>";

  document.getElementById("array").innerHTML = arr;
}
```

Out Put :

101
ravi
5000

9 Types of Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

Functions are useful in organizing the different parts of a script into the several tasks that must be completed. There are mainly two advantages of JavaScript functions.

1. **Code reusability:** We can call a function several times in a script to perform their tasks so it saves coding.
2. **Less coding:** It makes our program compact. We don't need to write many lines of code each time to perform a common task.

Types of Functions in JavaScript:

1. **Named Functions**
2. **Anonymous functions**
3. **Arrow functions**
4. **Generator functions**
5. **Async Functions**

1. Named Functions :
   - Function name is explicitly declared.
   - can be called before they are defined (hoisted).

Example :

```
  <!DOCTYPE html>
<html lang="en">
<head>
    <title>functions</title>
<script>
function myFunction(name)
{
   document.writeln( "Hello, " + name + "!");
}
</script>
</head>
<body>
<button onclick="myFunction('Harry Potter')">Try it</button>
</body>
</html>
```

2. **Anonymous Functions**

   An anonymous function is simply a function that does **not have a name**. Unlike named functions, which are declared with a name for easy reference, anonymous functions are usually created for specific tasks and are often assigned to variables or used as arguments for other functions.

   commonly used in various scenarios, such as callbacks, event handlers, and functional programming tasks.

   Example :

```
const greet = function () {
   console.log ("Welcome to GeeksforGeeks!");
};
```

Example of callbacks

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

3. **Arrow functions**

An **arrow function expression** is a compact alternative to a traditional function expression, with some semantic differences and deliberate limitations in usage:

Arrow functions without parameters

An arrow function without parameters is defined using empty parentheses (). This is useful when you need a function that doesn't require any arguments.

```
const arrow = ( ) => {
   console.log( "Arrow function" );
}

arrow();
```

Arrow function with single parameter:

If your arrow function has a single parameter, you can omit the parentheses around it.

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

Arrow function with multiple parameters:

Arrow functions with multiple parameters, like **(param1, param2) => { }**, simplify writing concise function expressions in JavaScript, useful for functions requiring more than one argument.

```
const arr = ( x, y, z ) => {
    console.log( x + y + z )
}

arr( 10, 20, 30 );
```

Arrow functions with array :

```
const subjects = ["MEANSTACK","ML","CNS"]

console.log(subjects.map((subject) => subject.length));
```

o/p: [9, 2, 3]