VJTECH ACADEMY

Client Side Scripting (JavaScript)

Author: "Prof. Vishal L. Jadhav"

Notes Creator: Ghansham Irashetti

[BE in Computer Engineering and Having 10 years of IT industry experience (Worked in Capgemini, Amdocs & currently working in TIBCO Software)]

Basics of JavaScript Programming

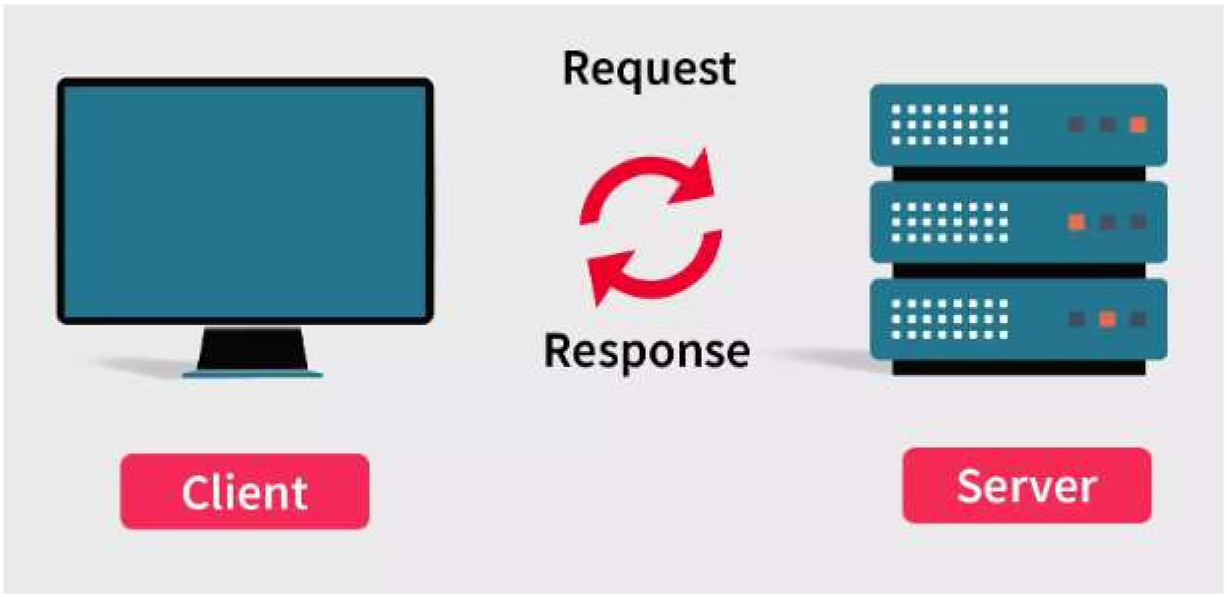
1.1 Features of JavaScript
1.2 Object Name, Property, Method, Dot Syntax, Main Event.
1.3 Values And Variables
1.4 Operators and Expressions- Primary Expressions, Object and Array initializers, function definition expressions, invocation expressions.
1.5 If Statement, if…else, if…else if, nested if statement.
1.6 Switch… case statement
1.7 Loop statement - for loop, for….in loop, while loop, do….while loop, continue statement.
1.8 Querying and setting properties and deleting properties, property getters and setters.

❖ Introduction to JavaScript :

JavaScript is used to create “client-side dynamic” pages. JavaScript is an object-based scripting language which is lightweight and cross-platform. JavaScript is not a compiled language, but it is a translated language. The JavaScript translator is responsible for the JavaScript code for the web browser.

❖ What is JavaScript :

- JavaScript is an open source & most popular client side scripting language supported by all browsers.
- JavaScript is a widely used programming language primarily known for its role in web development.
- It allows you to add interactivity and dynamic behavior to websites and web applications.
- JavaScript is a versatile language that can be used both on the client side (in web browsers) and on the server side (with technologies like Node.js).



- Actually there are so many languages that we can use for client-side programming but some popular ones are – [HTML, CSS, JAVASCRIPT]

Client-Side Scripting	Server-side Scripting
It is executed on the client side i.e. Front-end.	It is executed on the server side i.e. Back-end
It is visible to the user	It is not visible to the user
Useful in various frontend Operations	Useful in various backend operations
It can be used to collect the input given by the user	It can be used to process the input given by the user
It is not preferred for performing complex computations and transactions	It is preferred for performing complex computations and transactions
It is generally less secure	It is generally more secure
HTML, CSS and JavaScript are used for Client-side programming	Node.js, PHP, Python and Java are used for Server-side programming

❖ JavaScript History:-

- **Birth of JavaScript (1995):** JavaScript was created by Brendan Eich, a programmer at Netscape Communications Corporation.
- The language was originally named "Mocha," which was later changed to "LiveScript" and finally to "JavaScript" to capitalize on the popularity of Java, another programming language at the time.
- It was first introduced in the Netscape Navigator 2.0 browser.
- **Everywhere Today:** JavaScript is a must-know for web development, and its use has expanded to servers, mobile apps, and more.

❖ Features of JavaScript:-

- **Interactivity:** JavaScript enables dynamic and interactive elements on websites, responding to user actions in real time.
- **Client-Side Scripting:** It runs directly in web browsers, allowing manipulation of web page content without needing server interaction.
- **Event Handling:** JavaScript can respond to various user events like clicks, inputs, and scrolls, enhancing user experience.
- **DOM Manipulation:** It interacts with the Document Object Model (DOM) to modify and update web page content and structure.
- **Cross-Platform:** Works on different operating systems and is supported by all modern web browsers.
- **Open Source Community:** A large developer community contributes libraries, frameworks, and tools to enhance JavaScript's capabilities.
- **Server-Side Development:** With technologies like Node.js, JavaScript can be used to build server-side applications.
- **Regular Expressions:** Supports pattern matching for text searching and manipulation.
- Most of the JavaScript control statements syntax is same as syntax of control statements in C language.

❖ Advantages of JavaScript:-

1. **Reduce server interaction :** you can validate user input before sending the web page to the server. This saves server traffic, which means less loads on your server.
2. **Immediate feedback to the visitors :** they don't have to wait for a page reload to see if they have forgotten to enter something.
3. **Increased interactivity :** you can create interfaces that more interactive.
4. **Richer interfaces :** you can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

❖ Limitations of JavaScript:-

- **Client-Side Execution:** JavaScript runs on the client-side, which can limit its access to system resources.
- **Limited File Access:** Due to security concerns, JavaScript has limited access to the local file system.
- **Single-Threaded:** JavaScript is single-threaded, which can result in performance bottlenecks for resource-intensive tasks.
- **No Multithreading:** Lack of true multithreading support can affect heavy computation tasks.
- **Code Maintainability:** Large codebases might become harder to manage due to JavaScript's dynamic nature.
- **Limited Offline Functionality:** It requires an internet connection to access external resources.
- **Code Visibility:** Client-side code is visible to users, potentially exposing sensitive logic.
- **Lack of Compile-Time Checking:** Errors might only be caught during runtime, leading to unexpected behavior.

❖ JavaScript Vs Java :

Aspect	JavaScript	Java
Type	Interpreted scripting language.	Compiled programming language.
Usage	Primarily used for front-end web development. Can also be used on the server side (Node.js).	Used for building desktop, web, mobile, and enterprise applications.
Platform	Web browsers and Node.js for server-side.	Desktop applications, servers, mobile devices, and embedded systems.
Syntax	C-like syntax with dynamic typing.	C/C++-inspired syntax with strong static typing.
Data Types	Primitive and reference types.	Primitive and reference types.
Object Model	Prototype-based. Objects are instances of classes.	Class-based. Objects are instances of classes.
Frameworks	Offers frameworks like React, Angular, and Vue.js for front-end.	Offers various frameworks for different purposes (Spring, JavaFX, etc.).
Mobile Apps	Used with frameworks like React Native for mobile development.	Used with Android development (Java-based) or cross-platform solutions.
Compilation	No compilation, code is executed directly.	Code is compiled into bytecode and executed on the Java Virtual Machine (JVM).
Popularity	Widely used for web development and scripting.	Widely used for enterprise applications, Android apps, and more.

❖ JavaScript Tag :

1. What is the '<script>' Tag?

The **'<script>'** tag is used in HTML to include JavaScript code directly within a web page. It allows you to add interactivity and dynamic behavior to your web content.

2. The script tag takes two important attributes :

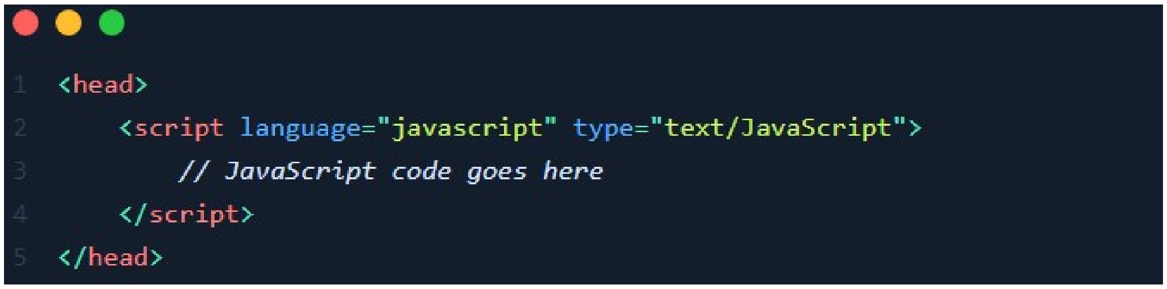
1. Language − this attribute specifies what scripting language you are using. Typically, its value will be JavaScript

2. Type − this attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

3. Where to Place the '<script>' Tag?

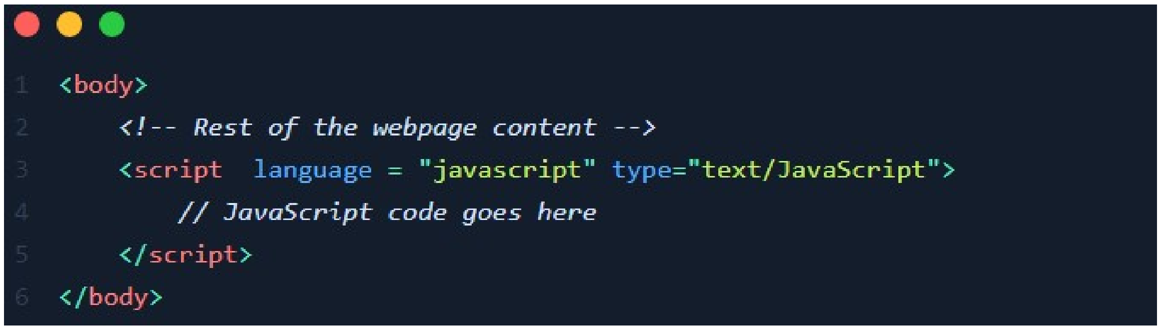
You can place the **'<script>'** tag in two main locations within an HTML document:

a. Inside the **'<head>'** section [example 1]:



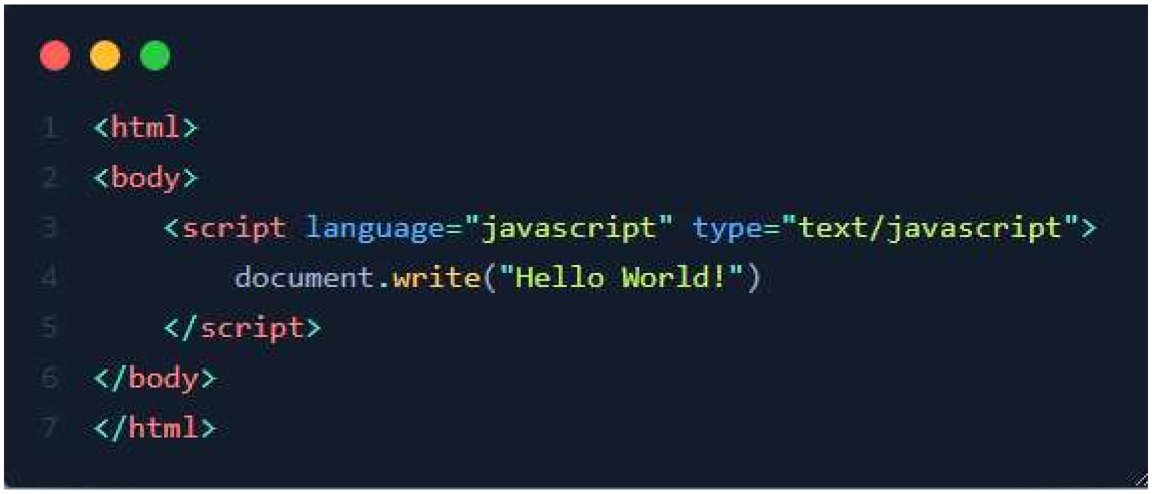
```
<head>
    <script language="javascript" type="text/JavaScript">
        // JavaScript code goes here
    </script>
</head>
```

b. At the end of the ' **<body>'** section [example 2] :



```
<body>
    <!-- Rest of the webpage content -->
    <script  language = "javascript" type="text/JavaScript">
        // JavaScript code goes here
    </script>
</body>
```

4. First JavaScript Code :



```
<html>
<body>
    <script language="javascript" type="text/javascript">
        document.write("Hello World!")
    </script>
</body>
</html>
```

❖ Terms of JavaScript :

1. Object :

a. An object in JavaScript is like a container that holds related information and actions together.

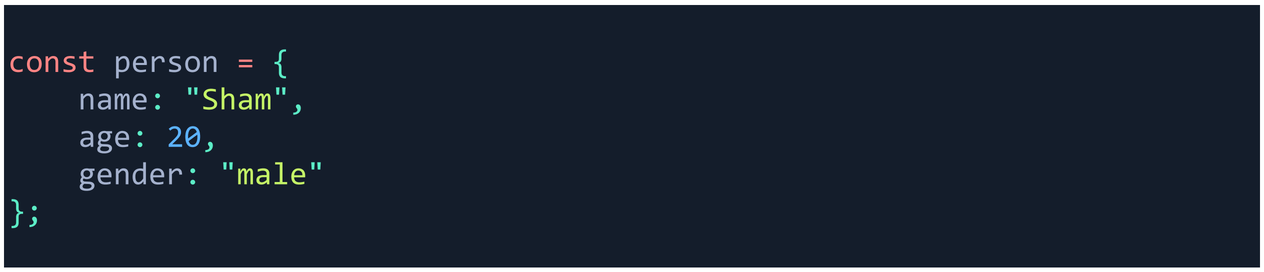
b. It organizes information neatly, grouping properties (attributes) and methods (functions) that belong together.

c. Objects act as blueprints for creating multiple instances of the same type, allowing you to reuse code efficiently.

d. Example :

Imagine a "cat" object. It has properties like "name" (which stores the cat's name), "color" (storing the fur color), and "age" (for the cat's age). It also has methods like "meow" (to make the cat meow) and "eat" (for feeding the cat).

2. **Property**
    a. Attribute: A property is like an attribute that provides information about an object.
    b. Data Holder: It holds data that describes the characteristics of an object.
    c. Accessed by Dot: Properties are accessed using dot syntax, like "object.property".
    d. For example, consider a "person" object :



```
const person = {
    name: "Sham",
    age: 20,
    gender: "male"
};
```

- In this object, "name," "age," and "gender" are properties that store specific information about the person.

3. **Method :**
    a. It's a function defined within an object, allowing the object to perform specific actions.
    b. It operates on the object's data, like actions connected to an object's properties.
    c. Methods enable objects to interact with and modify their own properties or perform related tasks.
    d. Methods group related functionalities within an object, improving code structure.
    e. Example: Consider a "car" object with a "start" method. The "start" method could make the car's engine roar to life, changing its internal state from "off" to "on."

4. **Dot Syntax :**
    a. Dot syntax in JavaScript is a way to access properties and methods of objects using a dot (.) between the object name and the property/method name.

    b. Dot syntax allows you to reach into an object and retrieve its properties or invoke its methods.
    c. The dot indicates that you're working within the context of a specific object, letting you interact with its data and behaviors.
    d. Dot syntax makes code easy to read, resembling natural language, and helps developers quickly understand how an object is used.
    e. It provides a consistent structure for interacting with objects, reducing ambiguity and making code more maintainable.
    f. Example: **document.write("Hello World");** // Accessing method **"write"** from document object.
5. **Main Event :**
    a. Events are actions or occurrences that happen in the browser, like a user clicking a button, a page finishing loading, or an element being hovered over.
    b. JavaScript can "listen" for these events and respond by executing specific code or functions. This allows websites to be interactive and responsive.
    c. There are various types of events, including user-generated events (click, input, mouseover), document-related events (DOMContentLoaded, load), and more.
    d. [we will see event handling part in upcoming unit 3 in detail]

❖ Values and Variables In JavaScript :

1. **Variables:**

- Variables are like containers that hold values.
- They give names to values, making it easier to refer to them in your code.
- You can think of variables as labeled boxes where you can store different things.
- Declaring Variables:-

  To declare a variable, use the **'var'**, **'let'**, or **'const'** keyword followed by a variable name.

  After declaring a variable, you can assign a value to it using the assignment operator " = ".

- Example :

```
var age;
let userName;
const pi = 3.14159;
```

The general rules for constructing names for variables:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names.

Types Of Variables:

- Global Variables
- Local Variables
- Block Variables

Local Variables :-

- A JavaScript local variable is declared inside block or function. It is accessiblewithin the function or block only.

```
function calculateSum() {

    var num1 = 5; // num1 is a local variable
    var num2 = 10;
    var sum = num1 + num2;
    return sum;
}
// we can’t access ‘num1’,’num2’,and ‘sum’ outside the function
```

Global Variables :-

- A global variable has global scope which means it can be defined / accessed anywhere in your JavaScript code.

```
var globalVar = 20; // globalVar is a global variable
function accessGlobal() {
    console.log(globalVar); // Accessing the global variable
}
```

Block Variables (ES6 and Later):-

- A Block variables are declared using let or const within a code block, like a loop or an if statement.
- They are confined to the block they're declared in and are not accessible outside of that block.

```
if (true) {
    let blockVar = 'Block variable'; // blockVar is a block variable
    console.log(blockVar);
}// blockVar is not accessible here
```

2. Values :

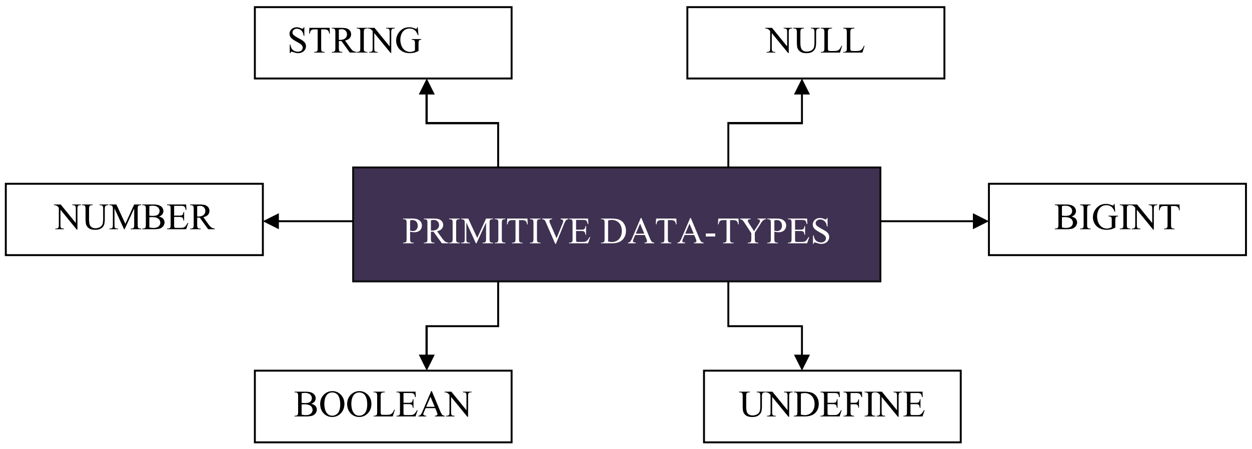
a. Values are the fundamental building blocks of any programming language, including JavaScript.
b. In JavaScript, values can be numbers (e.g., 42, 3.14), strings (e.g., "Hello, world!"), booleans (true or false), objects (complex data structures), arrays (ordered lists), functions (blocks of code), and more.

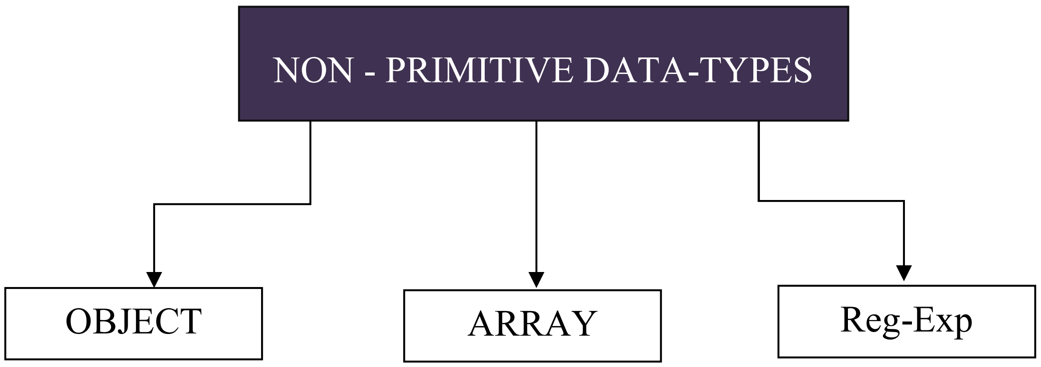
Difference between “var”, “let” and “const”

Feature	var	let	const
Scope	Function scope	Block scope	Block scope
Re-declaration	Allowed within the same scope	Not allowed within the same block	Not allowed within the same block
Value Change	Can be re-assigned	Can be re-assigned	Cannot be re-assigned
Initial Value	Initialized with 'undefined'	Not initialized at declaration	Must be initialized at declaration
Use Case	Legacy usage; Avoid in modern code	Use when re-assignment is needed	Use when the value won't change

[Note : Remember, the choice between ‘var’, ‘let’, and ‘const’ depends on the specific needs of your code. for our syllabus we use only ‘var’ keyword for declaration of variables .]

❖ Data Types In JavaScript :





PRIMITIVE DATA-TYPES

Data Type	Description	Example
undefined	Represents an undefined value.	let x; // x is undefined
null	Represents an null value.	let y = null; // y is null
boolean	Represents a boolean value, which can be true or false	let a = true; // a is true
number	Represents a numeric value, either integer or floating point.	let b = 5; // b is 5
string	Represents a sequence of characters, enclosed in quotes.	let c = "hello"; // c is "hello"
symbol	Represents a unique identifier, used primarily for object properties.	let d = Symbol("description"); // d is a unique symbol
BigInt	Represents integers with arbitrary precision.	let e = 9007199254740992n; // e is a BigInt

NONT-PRIMITIVE DATA-TYPES

Data Type	Description	Example
object	Represents a collection of key-value pairs, or properties, where each key is a string and each value can be any data type, including other objects.	let obj = {name: "John", age: 30}; // obj is an objectd
array	Represents a collection of elements, where each element can be any data type, including other arrays and objects. Arrays are also objects in JavaScript.	let arr = [1, 2, "three"]; // arr is an array
RegExp	Represents a regular expression, which is a pattern used to match character combinations in strings.	let pattern = /[a-z]+/; // pattern is a regular expression

❖ Comments In JavaScript :

- JavaScript supports both C-style and C++-style comments.
- In JavaScript, comments are text annotations you add to your code that are ignored by the browser's JavaScript engine.
- They are meant for developers to provide explanations, notes, or reminders within the code.
- Comments are not executed as part of the program and don't affect its behavior. There are two main types of comments in JavaScript:

1. **Single-line Comments:** These comments are used for adding notes on a single line of code.

```
// This is a single-line comment
var age = 25; // This comment explains the purpose of the variable
```

2. **Multi-line Comments (Block Comments):** These comments can span multiple lines and are often used for longer explanations or temporarily "commenting out" sections of code.

```
/* This is a
   multi-line comment
*/

/*
var x = 10;
var y = 20;
*/
```

❖ Operators and Expressions :

1. Operator :

- Operators are symbols or keywords used to perform operations on values or variables.
- They allow you to manipulate and combine data in various ways.
- JavaScript has several types of operators:
    1. Arithmetic Operators
    2. Comparison Operators
    3. Logical Operators
    4. Bitwise Operators
    5. Assignment Operators
    6. Increment/Decrement Operators
    7. Conditional (Ternary) Operator

2. Expressions :

- An expression is a combination of values, variables, and operators that can be evaluated to produce a result.
- Think of it as a formula in math. Expressions can be simple or complex

Let's explore operators in details –

1. Arithmetic Operator

- Arithmetic operators in JavaScript are used to perform mathematical calculations on numerical values. Here are the primary arithmetic operators:

- **Addition (+)**	: Adds two values together.
- **Subtraction (-)**	: Subtracts the second value from the first.
- **Multiplication (*)**	: Multiplies two values.
- **Division (/)**	: Divides the first value by the second.
- **Modulus (%)**	: Returns the remainder of the division.
- **Exponentiation (**)**	: Raises the first value to the power of the second.

- For example, if ‘a’ is ‘10’ and ‘b’ is ‘3’, the table would show the following results:

Operator	Operation	Example	Result
+	Addition	10 + 3	13
-	Subtraction	10 - 3	7
*	Multiplication	10 * 3	30
/	Division	10 / 3	3.333...
%	Modulus (Remainder)	10 % 3	1
**	Exponentiation	10 ** 3	1000

- Example :

```
<html>
<head>
   <title>Javascript Arithmetic Operator</title>
</head>
<body>
   <script language="javascript" type="text/javascript">
      // Arithmetic Operators
      var num1 = 10;
      var num2 = 5;
      // Addition
      var sum = num1 + num2;
      document.write("Sum:", sum);
      document.write("<br>")

      // Subtraction
      var difference = num1 - num2;
      document.write("Difference:", difference);
      document.write("<br>")

      // Multiplication
      var Multiplication = num1 * num2;
      document.write("Multiplication", Multiplication);
      document.write("<br>")

      // Division
      var quotient = num1 / num2;
      document.write("Quotient:", quotient);
      document.write("<br>")

      // Modulus (Remainder)
      var remainder = num1 % num2;
      document.write("Remainder:", remainder);
      document.write("<br>")

      // Exponentiation
      var exponentiation = num1 ** num2;
      document.write("Exponentiation:", exponentiation);
      document.write("<br>")
   </script>
</body>
</html>
```

2. Comparison Operator

- Comparison operators in JavaScript are used to compare two values and return a Boolean result (true or false) based on the comparison.
- Here's a table illustrating common comparison operators and their meanings:

Comparison Operator	**Literal Meaning**	**Use case**
>	greater than	It returns true if the operand on the left of it is greater than the operand on the right of it.
<	less than	It returns true if the operand on the left of it is less than the operand on the right of it.
>=	greater than or equal to	It returns true if the operand on the left of it is greater than or equal to the operand on the right of it.
<=	less than or equal to	It returns true if the operand on the left of it is less than or equal to the operand on the right of it.
==	equal to	It returns true if the operand on the left of it is (loosely)equal to the operand on the right of it.
===	strict equal to	It returns true if the operand on the left of it is (strictly)equal to the operand on the right of it.

- Example :

```
<html>
<head>
   <title>JavaScript - Comparison Operator</title>
</head>
<body>
   <script language="javascript" type="text/javascript">
      // Comparison Operator Examples
      var num1 = 10;
      var num2 = 5;
      var text1 = 'hello';
      var text2 = 'world';

      document.write("num1:", num1);
      document.write("num2:", num2);

      document.write("Equal to (==):", num1 == num2);                  // false
      document.write("<br>")
      document.write("Not equal to (!=):", num1 != num2);              // true
      document.write("<br>")
      document.write("Strict equal to (===):", num1 === '10');         // false
      document.write("<br>")
      document.write("Strict not equal to (!==):", num1 !== '10');     // true
      document.write("<br>")
      document.write("Greater than (>):", num1 > num2);                // true
      document.write("<br>")
      document.write("Less than (<):", num1 < num2);                   // false
      document.write("<br>")
      document.write("Greater than or equal to (>=):", num1 >= 10);    // true
      document.write("<br>")
      document.write("Less than or equal to (<=):", num2 <= 5);        // true
      document.write("<br>")
   </script>
</body>
</html>
```

3. Logical Operators

- Logical operators in JavaScript are used to combine or manipulate boolean values (true or false).
- These operators, including AND (&&), OR (||), and NOT (!), are commonly used in decision-making and looping scenarios.

1. Logical AND (&&) operator :

- he logical "AND" operator in JavaScript, represented by "&&", evaluates two operands. If both operands are true, it returns true. Otherwise, it returns false.
- Please refer to the truth table given below:

Operand 1	Operand 2	One && Two
True	False	False
False	True	False
False	False	False
True	True	True

- Example :

```
<body>
   <script language="javascript" type="text/javascript">
      document.write(true && true); // true
      document.write("<br>");
      document.write(false && true); // false
      document.write("<br>");
      document.write(10 > 0 && 10 < 50); // true
      document.write("<br>");
   </script>
</body>
```

2. Logical OR (||) operator :

- The || operator in JavaScript returns the first truthy operand encountered, or the value of the last falsy operand if all values are falsy.
- Please refer to the truth table given below:

Operand 1	Operand 2	One **||** Two
True	False	True
False	True	True
True	True	True
False	False	False

- Example :

```
<body>
  <script language="javascript" type="text/javascript">
     // logical or operator examples

     document.write(true || true); // true
     document.write("<br>");
     document.write(false || true); // true
     document.write("<br>");
     document.write(10 <  0 || 10 >50); // false
     document.write("<br>");
  </script>
</body>
```

3. Logical OR (||) operator :

- Whenever there is a need to inverse the boolean value, the logical NOT (!) operator is used.
- This logical operator in JavaScript is also called logical complement as it converts the truthy value to the falsy value and the falsy value to the truthy value.

Operand 1	!Operand 1
True	False
False	True

4. Assignment Operators

Operator	Example	Description
=	x = 5;	Assigns the value on the right to the variable on the left.
+=	x += 3;	Adds the right value to the variable's current value and assigns the result.
-=	x -= 2;	Subtracts the right value from the variable's current value and assigns the result.
*=	x *= 4;	Multiplies the variable's current value by the right value and assigns the result.

- Example :

```
<body>
   <script language="javascript" type="text/javascript">
       var num1 = 10;
       var num2 = 5;

       document.write("num1:", num1);
       document.write("num2:", num2);
       document.write("<br>")

       num1 += num2;   // 10 + 5 = 15
       document.write("Add and assign (+=):", num1);
       document.write("<br>")

       num1 -= num2;   // 15 - 5 = 10
       document.write("Subtract and assign (-=):", num1);
       document.write("<br>")

       num1 *= num2;   // 10 * 5 = 50
       document.write("Multiply and assign (*=):", num1);
       document.write("<br>")

       num1 /= num2;   // 50 / 5 = 10
       document.write("Divide and assign (/=):", num1);
       document.write("<br>")

       num1 %= num2;   // 10 % 5 = 0
       document.write("Modulus and assign (%=):", num1);
       document.write("<br>")

       num1 **= num2;  // 0 ** 5 = 0
       document.write("Exponentiation and assign (**=):", num1);
       document.write("<br>")

   </script>
</body>
```

5. Conditional (Ternary) Operator:

- The conditional operator, often called the ternary operator, is a shorthand way to write simple if-else statements. It allows you to make a quick decision between two values based on a condition.
- Syntax : [condition ? value_if_true : value_if_false;]

- Explanation:
  - The condition is evaluated first.
  - If the condition is true, the expression returns **value_if_true**.
  - If the condition is false, the expression returns **value_if_false**.

- Example

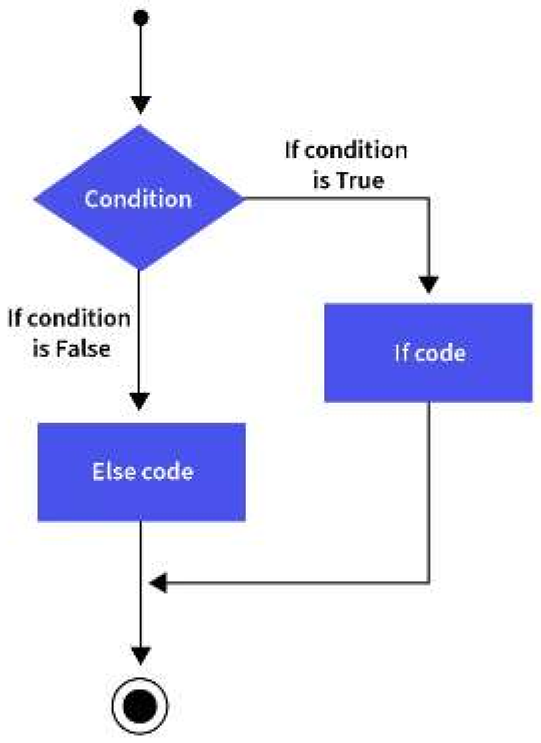
```
<script language="javascript" type="text/javascript">
    var age = 18;
    var message = (age >= 18) ? "You are an adult" : "You are not an adult";
    console.log(message);
</script>
```

❖ JavaScript Statements:

1. Conditional Statement (If Statement).
2. Loop/Iteration Statement.
3. “with” Statement.

1. Conditional Statement :

- Conditional statements in JavaScript are used to make decisions in your code based on certain conditions.
- These statements allow your program to execute different blocks of code depending on whether a given condition is true or false.
- The most common type of conditional statement is the **‘if’** statement, and there are variations like **‘else if’** and **‘else’** to provide multiple decision paths. Here's how conditional statements work:
-



-

‘ if ’ Statement

- Once the condition is defined, place the condition inside **‘if()’** statement. The execution of **‘if()’** block will depend on whether the condition is true or false.
- Syntax :

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

‘ if….else ’ Statement

- **else statement** in javascript is used to execute a block when all if conditions fail. In order to write an else block, it is mandatory to declare an if statement, although the vice versa is not true.

- Syntax :

```
if (condition) {
   //code to be executed
} else {
   //code to be executed
}
```

➢ ' if….else if….else ' Statement

- The " **if else if** " condition is used. It is used where there are multiple conditions to be checked in order to proceed.
- Syntax :

```
if (condition1) {
   //code to be executed
} else if (condition2) {
   //code to be executed
} else if (condition3) {
   //code to be executed
} else if (condition4) {
   //code to be executed
} else if (condition5) {
   //code to be executed
}else{
   //code to be executed
}
```

- An else statement can also be added in the end of the if, and else if conditions as well. This else will only execute when all if conditions will fail.

- **Example 1 :**

```
var age = 18;
if (age >= 18) {
  document.write ("You are an adult");
} else {
  document.write ("You are not an adult");
}
```

- **Example 2 :**

```
var age = 18;
if (age >= 18) {
    document.write("You are an adult");
} else if (age < 18) {
   document.write ("You are not an adult");
} else {
   document.write ("You are not a human");
}
```

➢ **‘ switch ’ Case Statement**

- Switch statement is used to execute one of the statements from many blocks of statements.
- There are four keywords are used i.e switch, case, break, default.
- Switch case expression value compared with case value and if it is match then corresponding
- block will be executed.
- If none of case values are matched then default block is executed.

- Syntax :

```
switch (expression) {
    case caseValue1:
        //code a
        break;
    case caseValue2:
        //code b
        break;
    default:
        //code c
        break;
}
```

- **Syntax explanation**
- The **expression** is the condition or the value that needs to be evaluated. It is evaluated with respect to the first case, then the second case, and so on.
- The **case** is declared by using case keyword followed by the caseValue. This caseValue is evaluated with respect to expression.
- The **break** keyword breaks out of the switch block. It is used to stop the execution of the switch case. When the expression evaluates true with a case, it keeps executing all the statements until it encounters the break keyword. The break keyword is optional.
- The **default** statement is executed when the expression fails to evaluate true with respect to any case. The default statement is optional.

- **Example :**

```
var day = 2;
switch (day) {
    case 1:
        console.log("Monday");
        break;
    case 2:
        console.log("Tuesday");
        break;
    case 3:
        console.log("Wednesday");
        break;
    case 4:
        console.log("Thursday");
        break;
    case 5:
        console.log("Friday");
        break;
    case 6:
        console.log("Saturday");
        break;
    case 7:
        console.log("Sunday");
        break;
    default:
        console.log("Invalid day");
}
```

2. Loop/Iteration Statement :

- There are four types of loops in JavaScript.
- 1. for loop
- 2. while loop
- 3. do-while loop
- 4. for-in loop

➢ 'for' Loop :

- A for loop is an entry-controlled loop that repeats a block of code if the specified condition is met.
- A for loop in JavaScript is typically used to loop through a block of code multiple times.
- Syntax :

```
for (initialising statement; conditional statement; updation statement) {
  // code block to be executed
}
```

- The **initializing** statement initializes the concerned counter variable. Any order of complexity is accepted here.
- The **conditional statement** checks whether the required condition is satisfied or not at the beginning of each iteration. If the value of the conditional statement is true in the for loop, the statement executes. If the conditional statement is false, the for loop terminates. If there is no conditional statement provided, the for loop is executed anyway.
- The **update statement** increases or decreases the loop counter each time it is executed.

- **Example :**

```
 for (var i = 0; i < 10; i++) {
     document.write("i:", i);
     document.write("<br>")
 }
```

➢ **'while Loop :**

- A while loop executes the respective statements as long as the conditions are met.
- As soon as the condition becomes false, the loop is terminated, and the control is passed on to the statement outside the loop.
- It is important to note that **while** is an entry-controlled loop in JavaScript. This means that, like other entry-controlled loops, the conditional statement first needs to be evaluated.
- If the conditions are met, only then is the counter variable updated.
- Syntax :

```
while (condition) {
// code block
}
```

- **Example ;**

```
var i = 0;
while (i < 10) {
    document.write("i:", i);
    document.write("<br>")
    i++;
}
```

➢ ‘do…while’ Loop :

- A **do...while loop** is an exit-controlled loop.
- This means that the do...while loop executes the block of code once without checking the condition.
- After the code has been executed once, the condition is checked.
- If the condition is true, the code is executed further. If the condition is false, the loop is terminated.
- The do...while statement is used when you want to run a loop at least one time.

- Syntax :

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

- **Example :**

```
var i = 0;
do {
   document.write("Current Value Of i:", i);
   document.write("<br>")
   i++;
} while (i < 10);
```

➢ 'for…in' Statement :

- The **'for...in'** loop is used to iterate over the properties (keys) of an object.
- It's particularly useful when you want to examine or manipulate the properties of an object.
- However, it's not recommended to use **'for...in'** with arrays, as it may not always work as expected due to its behavior with array indices.
- Syntax :

```
for (variable in object) {
   // Code to be executed for each property
}
```

- **variable:** A variable that will represent the current property (key) during each iteration.
- **object:** The object whose properties you want to iterate over.

- **Example :**

```
var person = {
   name: "John",
   age: 30,
   city: "New York"
};
for (var key in person) {
   document.write(key + ":" + person[key]);
   document.write("<br>")
}
```

3. Loop Control Statement :

- Loop control statements allow you to alter the flow of loop execution:

➢ **break Statement:**

- The ‘break’ statement terminates the loop prematurely when a certain condition is met.
- Example :

```
for (var i = 0; i < 5; i++) {
   if (i === 3) {
      break; // Loop terminates when i is 3
   }
}
```

➢ **continue Statement:**

- The ‘continue’ statement skips the current iteration and moves to the next iteration of the loop.
- Example :

```
for (let i = 0; i < 5; i++) {
   if (i === 3) {
      continue; // Skips iteration when i is 3
   }
}
```

4. “with” Statement

- The with statement in JavaScript was designed to simplify code by reducing the need to specify a repeated object prefix.
- However, its usage is strongly discouraged due to potential issues with scope and performance.
- As of last update in September 2021, the with statement is considered deprecated and is not recommended for use.
- .
- Syntax :

```
with (object) {
    // Code block where object properties are accessed directly
}
```

- **Usage :** The with statement allows you to access object properties directly, without repeating the object name:

- **Example :**

```
var person = {
    firstName: "John",
    lastName: "Doe"
};
with (person) {
    document.write(firstName);  // Accessing properties without specifying "person."
    document.write(lastName);
}
```

❖ Querying Object Properties:

- You can retrieve the value of an object's property using either dot notation or square bracket notation.
- Example :

```
<html>
<body>
  <script language="javascript" type="text/javascript">
    var person = {
      firstName: "John",
      lastName: "Doe",
      age: 30
    };
    document.write(person.firstName); // John (using dot notation)
    document.write(person['lastName']); // Doe (using square bracket notation)
  </script>
</body>
</html>
```

❖ Setting Object Properties:

- You can set or update the value of an object's property using the same notation.
- Example :

```
    person.age = 31; // Updating age using dot notation

    person.city = "New York"; // Adding a new property using dot notation
```

❖ Deleting Object Properties:

- You can delete a property from an object using the **'delete'** operator.
- Example :

```
delete person.age; // Deleting the 'age' property

// After deletion
document.write(person.age); // undefined
```

❖ Property Getter and Setter :

➢ Getter :

- A getter is a method used to retrieve the value of a property. When you access a property using a getter, the getter function is automatically invoked, and its return value is used as the property value.

➢ Setter :

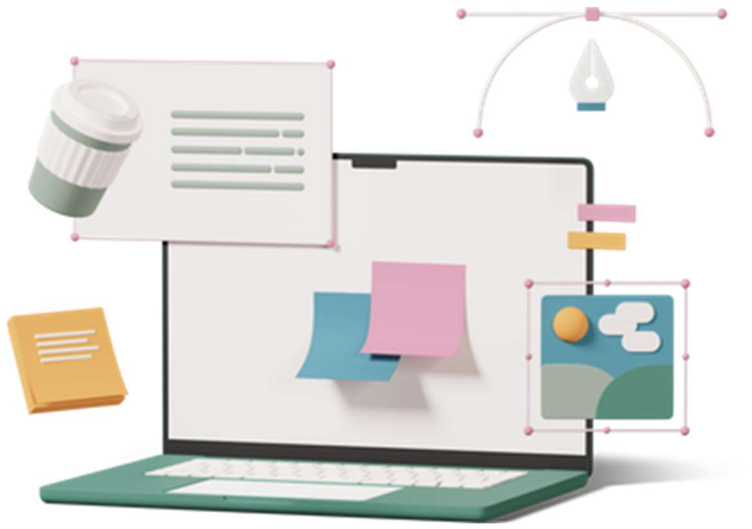
- A setter is a method used to set the value of a property. When you assign a value to a property using a setter, the setter function is automatically invoked with the assigned value.

➢ Defining Getters and Setters :

- You can define getters and setters using the 'get' and 'set' keywords within an object literal, a class definition, or an existing object prototype.

➢ Example : Defining Getters and Setters –

```
<html>
<body>
  <script language="javascript" type="text/javascript">
    const person = {
      firstName: "Vishal",
      lastName: "Jadhav",
      get fullName() {
        return this.firstName + " " + this.lastName;
      },
      set fullName(name) {
        const parts = name.split(" ");
        this.firstName = parts[0];
        this.lastName = parts[1];
      }
    };
    document.write(person.fullName); // Vishal Jadhav (using getter)
    document.write("<br>");
    person.fullName = "Ghanasham Irashetti"; // Using setter
    document.write(person.firstName); // Ghanasham
    document.write("<br>");
    document.write(person.lastName); // Irashetti
  </script>
</body>
</html>
```



VJTECH ACADEMY

CLASS NOTES

Author : Vishal Jadhav Sir | Contact : +91-9730087674
Edit By : Ghansham Irashetti