

Introduction to JavaScript

4.1 Concept of Script

In computing, a **script** is a sequence of instructions written in a programming or scripting language designed to automate tasks or execute a series of commands. Scripts are typically used to simplify repetitive tasks, configure systems, process data, or perform complex operations with minimal manual intervention. Here's a detailed look at the concept of scripts:

Types of Scripts

1.Shell Scripts: Written for command-line interfaces (CLI) such as Bash (Bourne Again Shell) in Unix/Linux or PowerShell in Windows. Used for automating system tasks, managing files, and executing commands.

Example: A Bash script to back up files might look like this:
bashCopy code
#!/bin/bash tar -czf backup.tar.gz /path/to/directory

2.Python Scripts: Python is a versatile language used for web development, data analysis, automation, and more. Python scripts are used for a wide range of applications due to its readability and extensive libraries.

Example: A Python script to print a message:
pythonCopy code
print("Hello, World!")

3.JavaScript Scripts: Used primarily in web development to create interactive and dynamic web pages. Runs in the browser and can also be used on the server side with Node.js.

Example: A simple JavaScript code to display an alert:
javascriptCopy code
alert("Hello, World!");

Introduction to JavaScript

1. What is JavaScript?

JavaScript is a high-level, interpreted scripting language that runs in web browsers, as well as on servers and other environments via platforms like Node.js. It was originally developed by Netscape in the mid-1990s and has since become a fundamental technology of the web, alongside HTML and CSS.

Key Features

Interpreted Language: JavaScript code is executed line-by-line by the browser's JavaScript engine without the need for prior compilation.

Client-Side Scripting: Primarily used to create interactive and dynamic web pages by running directly in the user's browser.

Event-Driven: Supports event-driven programming, allowing code to respond to user interactions (like clicks, keyboard input, etc.).

Dynamic Typing: Variables in JavaScript do not have a fixed type and can change type during runtime.

Object-Oriented: Supports object-oriented programming principles, allowing the creation of objects and the use of prototypes.

4.2 JavaScript Variables

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

- 1.Name must start with a letter (a to z or A to Z), underscore(_), or dollar(\$) sign.
- 2.After first letter we can use digits (0 to 9), for example value1.
- 3.JavaScript variables are case sensitive, for example x and X are different variables.

Ex:

```
var x = 10;  
var _value="sonoo";
```

Ex:

```
<script>  
var x = 10;  
var y = 20;  
var z=x+y;  
document.write(z);  
</script>
```

1.JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>  
function abc(){  
var x=10;//local variable  
}  
</script>
```

II. A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

<script>

```
var data=200;//gloabal variable
```

```
function a(){
```

```
document.writeln(data);
```

```
}
```

```
function b(){
```

```
document.writeln(data);
```

```
}
```

```
a();//calling JavaScript function
```

```
b();
```

</script>

A **JavaScript global variable** is declared outside the function or declared with window object. It can be accessed from any function.

```
function m(){  
  window.value=100;//declaring global variable  
  by window object  
}  
function n(){  
  alert(window.value);//accessing global variable  
  from other function  
}
```

JavaScript Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

JavaScript is a **dynamic type language**, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine.

You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc. For example:

1. `var a=40;`//holding number
2. `var b="Rahul";`//holding string
3. `var isActive = true; let isOver18 = false;`//boolean
4. `var undefinedVariable;`//Undefined

1. JavaScript primitive data types

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

2.JavaScript non-primitive data types

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

1.Object

Used to store collections of data and more complex entities.

An object is a collection of key-value pairs.

Example:

```
Var person = { name: "Alice", age: 30, isStudent: false };
```

2.Array

A special type of object that is used to store ordered collections of data.

Arrays are zero-indexed, meaning the first element is at index 0.

Example:

```
Var fruits = ["apple", "banana", "cherry"];
```

3.Function

A special type of object that is executable.

Functions can be assigned to variables, passed as arguments, and returned from other functions.

Example:

```
function greet() { console.log("Hello, world!"); }  
var greetFunction = greet; // Assigning function to a variable
```

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

```
var sum=10+20;
```

Here, + is the arithmetic operator and = is the assignment operator. There 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
7. JavaScript Arithmetic Operators

Arithmetic operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10===20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
	Bitwise OR	(10==20 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
	Logical OR	(10==20 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
=	Multiply and assign	var a=10; a=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

JavaScript Control Structure

1. JavaScript If-else

The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.

1. If Statement
2. If else statement
3. if else if statement

JavaScript If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

1. if(expression){
2. //content to be evaluated
3. }

Ex:

```
<script>
var a=20;
if(a>10){
document.write("value of a is greater than 10");
}
</script>
```

JavaScript If...else Statement

It evaluates the content whether condition is true or false. The syntax of JavaScript if-else statement is given below.

```
    if(expression){  
//content to be evaluated if condition is true  
} else{  
//content to be evaluated if condition is false  
}
```

Ex:

```
<script>  
var a=20;  
if(a%2==0){  
document.write("a is even number");  
}  
else{  
document.write("a is odd number");  
}  
</script>
```

JavaScript If...else if statement

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

```
if(expression1){  
  //content to be evaluated if e  
  xpression1 is true  
}  
else if(expression2){  
  //content to be evaluated if e  
  xpression2 is true  
}  
else if(expression3){  
  //content to be evaluated if e  
  xpression3 is true  
}  
else{  
  //content to be evaluated if n  
  o expression is true  
}
```

Ex:

```
<script>  
var a=20;  
if(a==10){  
  document.write("a is equal to 10");  
}  
else if(a==15){  
  document.write("a is equal to 15");  
}  
else if(a==20){  
  document.write("a is equal to 20");  
}  
else{  
  document.write("a is not equal to 10, 15 o  
r 20");  
}  
</script>
```


JavaScript Switch

The JavaScript switch statement is used to execute one code from multiple expressions. It is just like else if statement that we have learned in previous page. But it is convenient than if..else..if because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

```
switch(expression){  
  case value1:  
    code to be executed;  
    break;  
  case value2:  
    code to be executed;  
    break;  
  .....  
  
  default:  
    code to be executed if above values are not matched;  
}
```

Ex:

```
<script>  
var grade='B';  
var result;  
switch(grade){  
  case 'A':  
    result="A Grade";  
    break;  
  case 'B':  
    result="B Grade";  
    break;  
  case 'C':  
    result="C Grade";  
    break;  
  default:  
    result="No Grade";  
}  
document.write(result);  
</script>
```

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) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*

. It should be used if number of iteration is known. The syntax of for loop is given below.

```
for (initialization; condition; increment)
{
    code to be executed
}
```

Ex:

```
<script>
for (i=1; i<=5; i++)
{
    document.write(i + "<br/>")
}
</script>
```

2) JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times.*

It should be used if number of iteration is not known. The syntax of while loop is given below.

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

Ex:

```
<script>
var i=11;
while (i<=15)
{
    document.write(i + "<br/>");
    i++;
}
</script>
```

3) JavaScript 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.

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

Ex:

```
<script>
var i=21;
do{
    document.write(i + "<br/>");
    i++;
}while (i<=25);
</script>
```

4) JavaScript for in loop

The JavaScript for in loop is used *to iterate the properties of an object*.

The for...in loop in JavaScript is used to iterate over the enumerable properties of an object. It allows you to loop through the keys (or property names) of an object, giving you access to each key in succession.

Syntax of for...in Loop
javascriptCopy codefor (let key in object) { // code to execute for each key }

Ex:

```
const person = { firstName: "John", lastName: "Doe", age: 30 };  
for (let key in person) { console.log(key); // Outputs: firstName, lastName, age  
  console.log(person[key]); // Outputs: John, Doe, 30 }
```

4.3 Examples on JavaScript Operators

// Arithmetic Operators

<html>

<body>

<h2>JavaScript Arithmetic</h2>

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

<script>

let x = 100 + 50;

document.getElementById("demo").innerHT

ML = x;

</script>

</body>

</html>

4.4 JavaScript Functions

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

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

Code reusability: We can call a function several times so it save coding.

Less coding: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

JavaScript Function Syntax

The syntax of declaring function is given below.

```
function functionName([arg1, arg2,  
...argN]){  
  //code to be executed  
}
```

Ex:

```
<script>  
function msg(){  
  alert("hello! this is message");  
}  
</script>  
<input type="button" onclick="msg()" value="call function"/>
```

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

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1.Code reusability: We can call a function several times so it save 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.

example of function in JavaScript that does not has arguments.

```
<html><body><script>
function msg(){
alert("hello! this is message");
} </script>
<input type="button" onclick="msg()" value="call function"/>
</body>
</html>
```

JavaScript Function Arguments

We can call function by passing arguments. Let's see the example of function that has one argument.

Ex:

```
<script>
```

```
function getcube(number){  
  alert(number*number*number);  
}
```

```
</script>
```

```
<form>
```

```
<input type="button" value="click"  
onclick="getcube(4)"/>
```

```
</form>
```

Function with Return Value

We can call function that returns a value and use it in our program. Let's see the example of function that returns value.

Ex:

```
<script>
```

```
function getInfo(){  
  return "hello javatpoint! How r u?";  
}
```

```
</script>
```

```
<script>
```

```
document.write(getInfo());
```

```
</script>
```


JavaScript Function Object

In JavaScript, the purpose of Function constructor is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

Syntax

```
new Function ([arg1[, arg2[,  
....argn]],] functionBody)
```

Parameter

arg1, arg2, , argn - It represents the argument used by function.

functionBody - It represents the function definition.

Ex:

```
<script>
```

```
var add=new Function("num1","num2","return num1+num2");
```

```
document.writeln(add(2,5));
```

```
</script>
```

Method	Description
apply()	It is used to call a function contains this value and a single array of arguments.
bind()	It is used to create a new function.
call()	It is used to call a function contains this value and an argument list.
toString()	It returns the result

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

1. Writing into an HTML element, using innerHTML.
2. Writing into the HTML output using document.write().
3. Writing into an alert box, using window.alert().
4. Writing into the browser console, using console.log().

Ex:

```
<html>
<body>
<h2>My First Web Page</h2>
<p>My First Paragraph.</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
document.write(11 + 6);
window.alert(5 + 6);
alert(5 + 6);
</script>
<button type="button" onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

Types of Functions in JavaScript

There are two types of functions in JavaScript like any other programming language such as C, C++, and Java.

1. Predefined functions

2. User-defined functions

In this tutorial, we will concentrate on user-defined functions.

Predefined Functions in JavaScript

There are hundreds of predefined functions built into JavaScript to perform a variety of tasks. Some of the important predefined functions are as follows:

1. `alert()`: This function displays an alert dialog box on the browser.
2. `confirm()`: This function displays a confirmation dialog box and asks the user to choose one from two options.
3. `prompt()`: The `prompt()` function displays a prompt dialog box on the browser and prompts the user to enter input.
4. `write()`: The `write()` function used to write something on the document.
5. `Date()`: This function used to get the current date and time.
6. `select()`: The `select()` function used to select the pointed object.
7. `parseInt(numString)`: This function converts a string into an integer.

8. `parseFloat(numString, radix)`: This function converts a string into floating point number.

9. `sqrt(number)`: This function used to get the square root of any number.

10. `pow(number)`: It used to get the power of an integer.

User-defined Functions in JavaScript

Like any other robust programming language, JavaScript also allows to create your own user-defined function. With the help of user-defined function, we can organize JavaScript code into discrete and reusable chunks.

The syntax to define a user-defined function has shown above section.

JavaScript use-defined function does not execute when the web page loads. So it should place in the head section of an HTML document.

[adinsertter block="2"]

1. alert(): This function displays an alert dialog box on the browser.

Ex:

```
<!DOCTYPE html>
```

```
<html><head>
```

```
<title>Alert Example</title></head>
```

```
<body>
```

```
<button onclick="showAlert()">Click Me!</button>
```

```
<script>
```

```
function showAlert()
```

```
{
```

```
  alert("Hello, this is an alert message!");
```

```
} </script></body></html>
```

2. confirm(): This function displays a confirmation dialog box and asks the user to choose one from two options.

Ex: <html> <head>

<title>Confirm Example</title> </head>

<body> <button onclick="confirmAction()">Delete
Item</button> <script>

function confirmAction()

{ var userConfirmed = confirm("Are you sure you want to
delete this item?");

if (userConfirmed) { // User clicked "OK" alert("Item
deleted!"); }

else { // User clicked "Cancel" alert("Action cancelled."); }
}

</script> </body> </html>

3. prompt(): The prompt() function displays a prompt dialog box on the browser and prompts the user to enter input.

```
<html>
<head>
<script type = "text/javascript">
function fun() {
prompt ("This is a prompt box", "Hello world");
}
</script>
</head>
<body>
<p> Click the following button to see the effect </p>
<form>
<input type = "button" value = "Click me" onclick = "fun();" />
</form>
</body>
</html>
```

4.Date():

Ex:

```
<script>
const d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

5.Select()

```
<html><body>
Name: <input type="text" id="myText" value="hfgyttu">
<p>Click the button to select the content of the text field.</p>
<button type="button" onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  document.getElementById("myText").select();
}</script></body></html>
```


6.JavaScript Number parseInt() method

The JavaScript number parseInt() method parses a string argument and converts it into an integer value. With string argument, we can also provide radix argument to specify the type of numeral system to be used.

Syntax

The parseInt() method is represented by the following syntax:

`Number.parseInt(string, radix)`

`string` - It represents the string to be parsed.

`radix` - It is optional. An integer between 2 and 36 that represents the numeral system to be used.

ex:

ex:

```
<html>
```

```
<body>
```

```
<script>
```

```
var a="50";
```

```
var b="50.25"
```

```
var e="50.25String"
```

```
document.writeln(Number.parseInt(a)+"<br>");
```

```
document.writeln(Number.parseInt(b)+"<br>");
```

```
document.writeln(Number.parseInt(e));
```

```
</script>
```

```
</body>
```

```
</html>
```

4.5 JavaScript Events


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

Common events include mouse clicks, keyboard presses, page loads, and form submissions. Event handlers are JavaScript functions that respond to these events, allowing developers to create interactive web applications.

Syntax:

<HTML-element Event-Type = "Action to be performed">

Common JavaScript Events Table

Event Attribute	Description	Trigger
<code>`onclick`</code>	Triggered when an element is clicked.	Click on the element.
<code>`onmouseover`</code>	Fired when the mouse pointer moves over an element.	Mouse pointer enters the element.
<code>`onmouseout`</code>	Occurs when the mouse pointer leaves an element.	Mouse pointer leaves the element.
<code>`onkeydown`</code>	Fired when a key is pressed down.	Key press event.
<code>`onkeyup`</code>	Fired when a key is released.	Key release event.
<code>`onchange`</code>	Triggered when the value of an input element changes.	Input value changes (e.g., text input, select box).
<code>`onload`</code>	Occurs when a page has finished loading.	Page load completion.
<code>`onsubmit`</code>	Fired when a form is submitted.	Form submission event.
<code>`onfocus`</code>	Occurs when an element gets focus.	Element gains focus (e.g., when a user clicks on an input field).
<code>`onblur`</code>	Fired when an element loses focus 	Element loses focus (e.g., when a user clicks away from an input field).

1. JavaScript Events Examples

Example 1: onClick()

Here, we will display a message in the alert box when the button is clicked using onClick() event. This HTML document features a button styled to appear in the middle of the page. When clicked, the button triggers the `abc()` JavaScript function, which displays an alert box with the message “Hi there!”.

Ex:

```
<html>
```

```
<head>
```

```
<script> function abc()
```

```
{ alert('Hi there!'); }
```

```
</script> </head> <body>
```

```
<button type="button" onclick="abc()"
  style="margin-left: 50%;"> Click me event
</button> </body> </html>
```

2. **onkeyup()** ,**onKeyDown()**

```
<html>
```

```
<head><title>onkeyup Event Attribute </title>
```

```
<style> h2 {
```

```
    text-align: center;
```

```
  }input[type=text] {
```

```
    width: 50%;padding: 12px 20px;
```

```
    margin: 8px 0; box-sizing: border-box;
```

```
    font-size: 24px;color: white;}
```

```
  p { font-size: 20px;
```

```
  }</style></head><body>
```

```
<p>
```

```
  Release the key to set a green background color.
```

```
</p> <input type="text" id="demo"
```

```
  onkeydown="this.style.backgroundColor = 'blue';"
```

```
  onkeyup="this.style.backgroundColor = 'green';">
```

```
</body>
```

```
</html>
```

4.6 Concept of array , Types of arrays

A [JavaScript array](#) is a collection of multiple values at different memory blocks but with the same name. The values stored in an array can be accessed by specifying the indexes inside the square brackets starting from **0** and going to the **array length – 1** ([0]...[n-1]).

A JavaScript array can be classified into multiple types:

Table of Content

1. [Numeric Array](#)
2. [String Array](#)
3. [Array of Arrays \(2D Arrays\)](#)
4. [Array of Objects](#)
5. [Mixed Arrays](#)
6. [Array with empty values](#)

1.Numeric Array

A numeric array is an array that contains only the numeric values as elements that are stored at different memory locations but in a consecutive manner.

Example: The below code example is an implementation of defining and using a numeric array.

```
const numArr = [1, 2, 3, 4, 5];  
console.log("Numeric Array: ", numArr);  
console.log("First element: ", numArr[0]);
```

Output Numeric Array: [1, 2, 3, 4, 5] First element: 1

2.String Array

A string array contains only the string elements which means it has all the memory blocks filled with the string values that can be accessed using the indexing syntax.

Example: The below code explains the string array implementation.

```
const strArr = ['GeeksforGeeks', 'JavaScript',  
  'TypeScript'];  
console.log("String Array: ", strArr);  
console.log("First element: ", strArr[0]);
```

Output

```
String Array: [ 'GeeksforGeeks', 'JavaScript', 'TypeScript'  
  ]
```

```
First element: GeeksforGeeks
```

3.Array of Arrays (2D Arrays)

An array of arrays or the 2D array contains the multiple arrays as its elements that are stored at different memory locations. It can be used to create the nested arrays.

Example: The below example is the practical implemntation of the 2D arrays in JavaScript.

```
const arrOfArr =  
[  
  [1, 2, 3],  
  ['GeeksforGeeks', 'JavaScript']  
];  
console.log("Array of Arrays: ", arrOfArr);  
console.log("First Array: ", arrOfArr[0]);  
console.log("Second Array: ", arrOfArr[1]);
```

Output

```
Array of Arrays: [ [ 1, 2, 3 ], [ 'GeeksforGeeks', 'JavaScript' ] ]  
First Array: [ 1, 2, 3 ]  
Second Array: [ 'GeeksforGeeks', 'JavaScript' ]
```

4.Array of Objects

An array of objects is a array that contains the [JavaScript Objects](#) with different properties as elements stored at the different memory locations.

Example: The below code implements the Array of Objects in JavaScript.

Javascript

```
const objectsArray =  
[  
  { type: "Company",  
    name: "GeeksforGeeks"},  
  { type: "Cricketer",  
    name: "Virat Kohli"  
  }  
];  
  
console.log("Array of Objects: ", objectsArray);  
console.log("First Object: ", objectsArray[0]);  
console.log("Second Object: ", objectsArray[1]);  
  
Output:Array of Objects: [ { type: 'Company', name:  
  'GeeksforGeeks' }, { type: 'Cricketer', name: 'Virat Kohli'  
  } ] First Object: { type: 'Company', name:  
  'GeeksforGeeks' } Second Object: { type: 'Cri...
```

5.Mixed Arrays

The mixed arrays in JavaScript can contain elements of multiple data types stored at contiguous memory location with same name that can be accessed using the indexing syntax.

Example: The below code is the practical implementation of the mixed arrays in JavaScript.

Javascript

```
const mixedArr =  
[  
  [1, 2, 3],  
  "GeeksforGeeks",  
  23,  
  {  
    cricketer: "Virat Kohli"  
  }  
];
```

```
console.log("Array element: ", mixedArr[0]);  
console.log("String element: ", mixedArr[1]);  
console.log("Number element: ", mixedArr[2]);  
console.log("Object element: ", mixedArr[3]);
```

OutputArray element: [1, 2, 3] String element: GeeksforGeeks Number element: 23 Object element: { cricketer: 'Virat Kohli' }

4.7 Math function

<code>Math.abs()</code>	Returns the absolute value of a number.	<code>Math.abs(x)</code>	<code>Math.abs(-10)</code> returns <code>10</code>
<code>Math.ceil()</code>	Rounds a number UP to the nearest integer.	<code>Math.ceil(x)</code>	<code>Math.ceil(4.3)</code> returns <code>5</code>
<code>Math.floor()</code>	Rounds a number DOWN to the nearest integer.	<code>Math.floor(x)</code>	<code>Math.floor(4.7)</code> returns <code>4</code>
<code>Math.round()</code>	Rounds a number to the nearest integer.	<code>Math.round(x)</code>	<code>Math.round(4.5)</code> returns <code>5</code>
<code>Math.max()</code>	Returns the largest of zero or more numbers.	<code>Math.max(x1, x2, ..., xn)</code>	<code>Math.max(1, 5, 3)</code> returns <code>5</code>
<code>Math.min()</code>	Returns the smallest of zero or more numbers.	<code>Math.min(x1, x2, ..., xn)</code>	<code>Math.min(1, 5, 3)</code> returns <code>1</code>
<code>Math.pow()</code>	Returns the base raised to the exponent power.	<code>Math.pow(base, exponent)</code>	<code>Math.pow(2, 3)</code> returns <code>8</code>
<code>Math.sqrt()</code>	Returns the square root of a number.	<code>Math.sqrt(x)</code>	<code>Math.sqrt(25)</code> returns <code>5</code>
<code>Math.random()</code>	Returns a pseudo-random number between 0.5 and 1.	<code>Math.random()</code>	<code>Math.random()</code> might return 0.755

Date and Math Object with example

Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers. The Math object has no constructor. The Math object is static.

All methods and properties can be used without creating a Math object first.

Math Properties (Constants)

The syntax for any Math property is : *Math.property*.

JavaScript provides 8 mathematical constants that can be accessed as Math properties:

Example

Math.E // returns Euler's number

Math.PI // returns PI

Math.SQRT2 // returns the square root of 2

Math.SQRT1_2 // returns the square root of 1/2

Math.LN2 // returns the natural logarithm of 2

Math.LN10 // returns the natural logarithm of 10

Math.LOG2E // returns base 2 logarithm of E

Math.LOG10E // returns base 10 logarithm of E

Example of math constant

```
<html>
<body>

<h2>JavaScript Math Constants</h2>

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

<script>
document.getElementById("demo").innerHTML =
"<p><b>Math.E:</b> " + Math.E + "</p>" +
"<p><b>Math.PI:</b> " + Math.PI + "</p>" +
"<p><b>Math.SQRT2:</b> " + Math.SQRT2 + "</p>" +
"<p><b>Math.SQRT1_2:</b> " + Math.SQRT1_2 + "</p>" +
"<p><b>Math.LN2:</b> " + Math.LN2 + "</p>" +
"<p><b>Math.LN10:</b> " + Math.LN10 + "</p>" +
"<p><b>Math.LOG2E:</b> " + Math.LOG2E + "</p>" +
"<p><b>Math.Log10E:</b> " + Math.LOG10E + "</p>";
</script>

</body>
</html>
```

JavaScript Math Constants
Math.E: 2.718281828459045
Math.PI: 3.141592653589793
Math.SQRT2: 1.4142135623730951
Math.SQRT1_2: 0.7071067811865476
Math.LN2: 0.6931471805599453
Math.LN10: 2.302585092994046
Math.LOG2E: 1.4426950408889634
Math.Log10E: 0.4342944819032518

Date Object

- The **JavaScript Date** object represents a single moment in time in a platform-independent format, encapsulating milliseconds since January 1, 1970, 00:00:00 UTC. It is fundamental for managing date and time in applications, providing methods for date arithmetic, formatting, and manipulation, essential for handling temporal data in web development.
- **Understanding the Date Object**
- The time value in a JavaScript Date object is measured in milliseconds since January 1, 1970, 00:00:00 UTC. The new Date() constructor initializes it, supporting parameters to specify year, month, day, hour, minute, second, and milliseconds.

Creating a Date Object

Creating a Date object involves invoking the new Date() constructor, which initializes the object with the current date and time based on the system's local time zone. The Date constructor supports various parameter options to specify a specific date and time, including year, month, day, hour, minute, second, and milliseconds.

You can create a Date object in several ways:

Syntax

```
new Date();  
new Date(value);  
new Date(dateString);  
new Date(year, month, day, hours, minutes, seconds,  
milliseconds);
```

1. **getFullYear():** Returns the year as a four-digit number (e.g., 2021).
2. **getMonth():** Returns the month as a number (0-11), where 0 is January and 11 is December.
3. **getDate():** Returns the day of the month (1-31).
4. **getDay():** Returns the day of the week as a number (0-6), where 0 is Sunday.
5. **getHours():** Returns the hour (0-23).
6. **getMinutes():** Returns the minutes (0-59).
7. **getSeconds():** Returns the seconds (0-59).
8. **getMilliseconds():** Returns the milliseconds (0-999).
9. **getTime():** Returns the number of milliseconds since January 1, 1970 (the Unix epoch).

Working with UTC Methods

UTC (Universal Time Coordinated) is the time set by the World Time Standard.

Example: Get UTC Date Components

```
<html>
<body><h2>Using new Date()</h2>
<script>
  var utcDate = new Date();
  var utcYear = utcDate.getUTCFullYear();
  var utcMonth = utcDate.getUTCMonth() + 1; // Adding 1 for human-readable month
  var utcDay = utcDate.getUTCDate();
  document.write(`UTC Year: ${utcYear}, UTC Month: ${utcMonth}, UTC Day: ${utcDay}`);</script>
</body>
</html>
```

JavaScript Set Date Methods

Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object.

```
<html>
<body>
<p>The setDate() method sets the day of a date object:</p>
<p id="demo"></p>
<script>
const d = new Date();
d.setDate(15);
//document.getElementById("demo").innerHTML = d;
document.write(d);
</script>
</body>
</html>
```

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

4.8 JavaScript string object

The **String** object in JavaScript lets you work with a series of characters; it wraps JavaScript's string primitive data type with a number of helper methods. The string is a sequence of characters containing 0 or more characters. For example, 'Hello' is a string.

Syntax

JavaScript strings can be created as objects using the String() constructor or as primitives using string literals.

Use the following syntax to create a String object –

```
var val = new String(value);
```

The String parameter, value is a series of characters that has been properly encoded.

We can create string primitives using string literals and the String() function as follows –

```
str1 = 'Hello World!'; // using single quote
```

```
str2 = "Hello World!"; // using double quote
```

```
str3 = 'Hello World'; // using back ticks
```

```
str4 = String('Hello World!'); // using String() function
```

String Object Methods

Sr.No.	Method	Description
1	at()	Returns the character from the specified index.
2	charAt()	Returns the character at the specified index.
3	charCodeAt()	Returns a number indicating the Unicode value of the character at the given index.
4	codePointAt()	Returns a number indicating the Unicode value of the character at the given index.
5	concat()	Combines the text of two strings and returns a new string.
6	endsWith()	Checks whether the string ends with a specific character or substring.
7	includes()	To check whether one string exists in another string.
8	indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
9	lastIndexOf()	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.

localeCompare()	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
match()	Used to match a regular expression against a string.
matchAll()	Used to match all occurrences of regular expression patterns in the string.
normalize()	To get the Unicode normalization of the string.
padEnd()	To add padding to the current string with different strings at the end.
padStart()	To add padding to the current string with different strings at the start.
raw()	Returns a raw string form of a given template literal.
repeat()	To get a new string containing the N number of copies of the current string.
replace()	Used to find a match between a regular expression and a string and replace the matched substring with a new one.
replaceAll()	Used to find a match between a regular expression and a string and replace all the

search()	Executes the search for a match between a regular expression and a specified string.
slice()	Extracts a section of a string and returns a new string.
split()	Splits a String object into an array of strings by separating the string into substrings.
substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.
substring()	Returns the characters in a string between two indexes into the string.
toLocaleLowerCase()	The characters within a string are converted to lowercase while respecting the current locale.
toLocaleUpperCase()	The characters within a string are converted to the upper case while respecting the current locale.
toLowerCase()	Returns the calling string value converted to lowercase.
toString()	Returns a string representing the specified object.

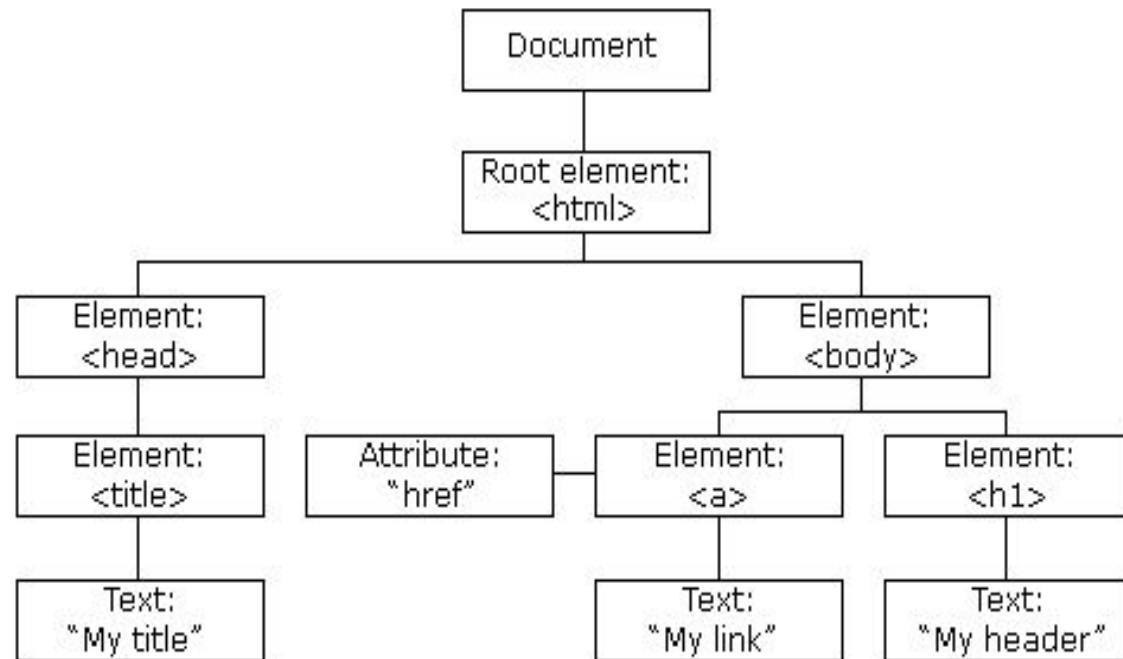
Example:

```
<html>
<body>
<h1>JavaScript Strings</h1>
<script>
var text = "AVWXYZ";
var str1="welcome java";
var str2="hello";
document.write(text.length+"<br>");
document.write("first char="+text.charAt(0)+"<br>");
document.write("char="+text.charAt(text.length-3)+"<br>");
document.write("char="+text.charCodeAt(text.length-2)+"<br>");
document.write(str1.concat(str2)+"<br>");
document.write(str1.replace("java","os")+"<br>");
document.write(str1.substring(3,6));
</script>
</body>
</html>
```


4.9 The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects:



- In this DOM tree, the document is the root node. The root node has one child node which is the <html> element. The <html> element is called the *document element*.
- Each document can have only one document element. In an HTML document, the document element is the <html> element. Each markup can be represented by a node in the tree. The document object represents the whole html document.
- When html document is loaded in the browser, it becomes a document object. It is the root element that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.
- Properties of document object
- Let's see the properties of document object that can be accessed and modified by the document object.

Methods of document object

Method	Description
<code>write("string")</code>	writes the given string on the document.
<code>writeln("string")</code>	Same as <code>write()</code> , but adds a newline character after each statement.
<code>getElementById()</code>	returns the element having the given id value.
<code>getElementsByName()</code>	returns all the elements having the given name value.
<code>getElementsByTagName()</code>	returns all the elements having the given tag name.
<code>getElementsByClassName()</code>	returns all the elements having the given class name.

document

If `document.body` is used before the `<body>` tag (e.g. inside the `<head>`), it will return [null](#) instead of the body element. Because the point at which the script is executed, the `<body>` tag was not parsed by the browser, so `document.body` is truly null at that point.

ex:

```
<html lang="en">
<head> <meta charset="utf-8">
<title>JS Document.body Demo</title>
<script> alert("From HEAD: " + document.body); // Outputs:
    null (since <body> is not parsed yet) </script>
</head> <body>
<script> alert("From BODY: " + document.body); // Outputs:
    HTMLBodyElement </script> </body> </html>
```

document.getElementById()

The document.getElementById() method returns the element of specified id.

document.getElementById() method to get value of the input text.

But we need to define id for the input field.

Let's see the simple example of document.getElementById() method that prints cube of the given number.

Ex:

```
<script type="text/javascript">
```

```
function getcube(){
```

```
var number=document.getElementById("number").value;
```

```
alert(number*number*number);
```

```
}
```

```
</script>
```

```
<form>
```

```
Enter No:<input type="text" id="number" name="number"/><br/>
```

```
<input type="button" value="cube" onclick="getcube()"/>
```

```
</form>
```

document.getElementsByName()

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the `getElementsByName()` method is given below:

```
document.getElementsByName("name")
```

Ex: **<script** type="text/javascript">

```
function totalelements()
```

```
{
```

```
var allgenders=document.getElementsByName("gender");
```

```
alert("Total Genders:"+allgenders.length);
```

```
}
```

```
</script>
```

```
<form>
```

```
Male:<input type="radio" name="gender" value="male">
```

```
Female:<input type="radio" name="gender" value="female">
```

```
<input type="button" onclick="totalelements()" value="Total Genders">
```

```
</form>
```

document.getElementsByTagName()

The **document.getElementsByTagName()** method returns all the element of specified tag name.

The syntax of the `getElementsByTagName()` method is given below:

```
document.getElementsByTagName("name")
```

Ex:

```
<script type="text/javascript">
```

```
function countpara(){
```

```
var totalpara=document.getElementsByTagName("p");
```

```
alert("total p tags are: "+totalpara.length);
```

```
}
```

```
</script>
```

```
<p>This is a paragraph</p>
```

```
<p>Here we are going to count total number of paragraphs by getEleme  
ntByTagName() method.</p>
```

```
<p>Let's see the simple example</p>
```

```
<button onclick="countpara()">count paragraph</button>
```

getElementsByClassName()

The `getElementsByClassName()` method returns a collection of elements with a specified class name(s).

Ex:

```
<html>
```

```
<body>
```

```
<h1>The Document Object</h1>
```

```
<h2>The getElementsByClassName() Method</h2>
```

```
<p>Change the text of the first element with class="example":</p>
```

```
<div class="example">Element1</div>
```

```
<div class="example">Element2</div>
```

```
<script>
```

```
const collection = document.getElementsByClassName("example");
```

```
collection[0].innerHTML = "Hello World!";
```

```
</script>
```

```
</body>
```

```
</html>
```


4.10 Validation in JavaScript

- JavaScript Form Validation is a way to ensure that the data users enter into a form is correct before it gets submitted. This helps ensure that things like emails, passwords, and other important details are entered properly, making the user experience smoother and the data more accurate.
- **Steps for Form Validation in JavaScript**
- When we validate a form in JavaScript, we typically follow these steps:
- **Data Retrieval:**
 - The first step is to get the user's values entered into the form fields (like name, email, password, etc.). This is done using `document.forms.RegForm`, which refers to the form with the name "RegForm".
- **Data Validation:**
 - Name Validation: We check to make sure the name field isn't empty and doesn't contain any numbers.
 - Address Validation: We check that the address field isn't empty.
 - Email Validation: We make sure that the email field isn't empty and that it includes the "@" symbol.
 - Password Validation: We ensure that the password field isn't empty and that the password is at least 6 characters long.
 - Course Selection Validation: We check that a course has been selected from a dropdown list.
- **Error Handling:**
 - If any of the checks fail, an alert message is shown to the user using `window.alert`, telling them what's wrong.
 - The form focuses on the field that needs attention, helping the user easily fix the error.
- **Submission Control:**
 - If all the validation checks pass, the function returns true, meaning the form can be submitted. If not, it returns false, stopping the form from being submitted.
- **Focus Adjustment:**
 - The form automatically focuses on the first field that has an error, guiding the user to fix it.