JAVASCRIPT

(FOCUSED FOR REACT JS)

Prepared By: **Sagar Shrestha**
sagar.stha.brt@gmail.com

Introduction

- JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages.
- It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.
- It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser.
- Since then, it has been adopted by all other graphical web browsers.
- With JavaScript, users can build modern web applications to interact directly without reloading the page every time.
- The traditional website uses js to provide several forms of interactivity and simplicity.
- Although, JavaScript has no connectivity with Java programming language.
- The name was suggested and provided in the times when Java was gaining popularity in the market.
- In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

Introduction

- **Features of JavaScript**
- There are following features of JavaScript:
- All popular web browsers support JavaScript as they provide built-in execution environments.
- JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
- JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
- JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
- It is a light-weighted and interpreted language.
- It is a case-sensitive language.
- JavaScript is supportable in several operating systems including, Windows, macOS, etc.
- It provides good control to the users over the web browsers.

Introduction

- **Application of JavaScript**
- JavaScript is used to create interactive websites. It is mainly used for:
- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

- **JavaScript Example**

```
<body>
    <!-- Content here -->
    <script>
        document.write("Hello World by JavaScript.");
    </script>
</body>
```

Places to put JavaScript Code

- Javascript example is easy to code.
- The script tag specifies that we are using JavaScript.
- The text/javascript is the content type that provides information to the browser about the data.
- JavaScript provides 3 places to put the JavaScript code:
  i. Between the body tag of html
  ii. Between the head tag of html
  iii. In .js file (external javaScript).

i. JavaScript Example : code between the body tag

- Let's see the simple example of JavaScript that displays alert dialog box.

```
<body>
    <!-- Content here -->
    <script type="text/javascript">
        alert("Hello World!");
    </script>
</body>
```

Places to put JavaScript Code

ii. JavaScript Example : code between the head tag

- Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.
- In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.
- To call function, you need to work on event. Here we are using onclick event to call msg() function.

```
<!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 type="text/javascript">
        function showAlert(){
            alert("Hello World!");
        }
    </script>
</head>
<body>
    <p>Welcome to JavaScript</p>
    <button onclick="showAlert()">Show Alert</button>
</body>
</html>
```

Places to put JavaScript Code

iii. External JavaScript file

- We can create external JavaScript file and embed it in many html page.
- It provides code re-usability because single JavaScript file can be used in several html pages.
- An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.
- Let's create an external JavaScript file that prints *Hello World!* in a alert dialog box.

message.js

```
function showAlert(){
    alert("Hello World!");
}
```

index.html

```
<!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 type="text/javascript" src="message.js"></script>
</head>
<body>
    <p>Welcome to JavaScript (Using External JS)</p>
    <button onclick="showAlert()">Show Alert</button>
</body>
</html>
```

JavaScript Comment

- The JavaScript comments are meaningful way to deliver message.
- It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.
- The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.
- There are two types of comments in JavaScript.
    i. Single-line Comment
    ii. Multi-line Comment

i. Single-line Comment

```
<script>
    // This is single-line comment
</script>
```

ii. Multi-line Comment

```
<script>
    /*This is
    multi-line
    comment*/
</script>
```

JavaScript Variables

- In JavaScript, variables are used to store data values that can be referenced and manipulated throughout your code.

❑ Variable Scope

i. **Global Scope:** Variables declared outside of any function or block are globally scoped and accessible from anywhere in the code.

ii. **Function Scope:** Variables declared with var inside a function are accessible only within that function.

iii. **Block Scope:** Variables declared with let and const inside a block (e.g., within {}) are only accessible within that block.

❑ Variable Naming Rules

- Names can include letters, digits, underscores, and dollar signs.
- Names must start with a letter, underscore, or dollar sign.
- JavaScript is case-sensitive, so name, Name, and NAME are different variables.
- Names cannot be JavaScript reserved keywords (like for, while, if, etc.).

JavaScript Variables

❑ Variable Types

➢ JavaScript is a loosely typed language, meaning variables can hold different types of values, including:

i. **Primitive Types:** number, string, boolean, null, undefined, symbol, and bigint.

```
let num = 42; // number
let str = 'Hello'; // string
let isTrue = true; // boolean
let nothing = null; // null
let notDefined; // undefined
```

ii. **Object Types:** object, array, function, and date.

```
let person = { name: 'Alice', age: 30 }; // object
let numbers = [1, 2, 3]; // array
function greet() { console.log('Hello!'); } // function
let today = new Date(); // date
```

JavaScript Variables

❑ Declaring Variables

➢ You can declare variables using three keywords: var, let, and const.

- **var:** This keyword declares a variable that is function-scoped or globally-scoped. Variables declared with var can be re-assigned and re-declared in the same scope.

```
var name = 'Alice';
name = 'Bob'; // Re-assigning
```

- **let:** Introduced in ES6 (ECMAScript 2015), let declares a block-scoped variable, meaning it is limited to the block in which it is defined. It can be re-assigned but not re-declared within the same scope.

```
let age = 30;
age = 31; // Re-assigning
```

- **const:** Also introduced in ES6, const declares a block-scoped, read-only constant. Variables declared with const cannot be re-assigned or re-declared.

```
const pi = 3.14;
// pi = 3.1415; // This will cause an error
```

JavaScript Variables

❑ Hoisting in JavaScript

- ➢ Hoisting is a JavaScript mechanism where variable and function declarations are moved to the top of their containing scope during the compilation phase, before the code is actually executed.
- ➢ This means you can use variables and functions before they are declared in the code.

- ▪ **How Hoisting Works**

i. **Variable Declarations:** Only the declaration part of a variable is hoisted, not the initialization. This means the variable is created and initialized with undefined until the actual assignment is encountered in the code.

ii. **Function Declarations:** Entire function declarations are hoisted, so you can call a function before it is declared in the code.

- ▪ **Example variable hoisting with var:**

```
console.log(x); // undefined, not ReferenceError
var x = 5;
console.log(x); // 5
```

JavaScript Variables

- **Explanation:** During hoisting, the JavaScript engine moves the declaration var x; to the top of the code, so the code effectively looks like this:

```
var x; // Declaration hoisted
console.log(x); // undefined
x = 5; // Initialization
console.log(x); // 5
```

- The variable x is hoisted with an initial value of undefined until it is explicitly assigned 5.
- **Example Function Hoisting:**

```
greet(); // "Hello!"
function greet() {
  console.log('Hello!');
}
```

- **Explanation:** The entire function greet is hoisted, so you can call it before its declaration.

JavaScript Operators

- JavaScript operators are symbols that are used to perform operations on operands.
- There are following types of operators in JavaScript.
  i. Arithmetic Operators
  ii. Comparison (Relational) Operators
  iii. Bitwise Operators
  iv. Logical Operators
  v. Assignment Operators
  vi. Special Operators

JavaScript 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 Operators

- **Comparison Operators**
  - The JavaScript comparison operator compares the two operands.

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 Operators

- **Bitwise Operators**
  - The bitwise operators perform bitwise operations on operands.

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 Operators

- **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 Operators

- **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 Operators

- **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 If-Else Statement

- 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.

```
if(expression){
//content to be evaluated
}
```

JavaScript If-Else Statement

- **JavaScript If...else Statement**
  - It evaluates the content whether condition is true of false.

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

JavaScript If-Else Statement

- **JavaScript If...else if Statement**
  - It evaluates the content only if expression is true from several expressions.

```
if(expression1){
//content to be evaluated if expression1 is true
}
else if(expression2){
//content to be evaluated if expression2 is true
}
else if(expression3){
//content to be evaluated if expression3 is true
}
else{
//content to be evaluated if no expression is true
}
```

JavaScript Switch Statement

- 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.

```
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;
}
```

JavaScript Loops

- In JavaScript, loops are used to execute a block of code repeatedly until a specified condition is met.
- **Types of loops in JavaScript:**

i. **for loop:** Used when you know how many times to iterate.

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}
```

ii. **while loop:** Executes as long as the condition is true.

```
let i = 0;
while (i < 5) {
    console.log(i);
    i++;
}
```

JavaScript Loops

iii. **do...while loop:** Similar to while, but ensures the code runs at least once.

```
let i = 0;
do {
    console.log(i);
    i++;
} while (i < 5);
```

iv. **for...of loop:** Iterates over iterable objects like arrays.

```
const arr = [10, 20, 30];
for (const num of arr) {
    console.log(num);
}
```

v. **for...in loop:** Iterates over object properties.

```
const obj = { a: 1, b: 2 };
for (const key in obj) {
    console.log(key, obj[key]);
}
```

JavaScript Functions

- In JavaScript, functions are reusable blocks of code that perform a specific task. A function can be defined once and called multiple times, making code modular and efficient.

i. **Function Declaration:** Declares a named function using the function keyword.

```
function greet() {
    console.log("Hello, world!");
}
greet();  // Calls the function
```

ii. **Function Expression:** Defines a function inside an expression and stores it in a variable.

```
const greet = function() {
    console.log("Hello, world!");
};
greet();  // Calls the function
```

JavaScript Functions

iii. **Arrow Function:** A more concise syntax, especially useful for anonymous functions.

```
const greet = () => {
    console.log("Hello, world!");
};
greet();
```

iv. **Parameters and Arguments:** Functions can take inputs (parameters) and return outputs.

```
function add(a, b) {
    return a + b;
}
console.log(add(5, 3));  // Outputs 8
```

v. **Return Value:** A function can return a value using the return keyword.

```
function square(x) {
    return x * x;
}
console.log(square(4));  // Outputs 16
```

JavaScript Objects

- In JavaScript, objects are one of the most important data types and serve as the building blocks for modern JavaScript applications.
- Unlike primitive data types (Number, String, Boolean, null, undefined, Symbol), which store a single value, objects can store multiple values as properties.

- **Key Points About Objects**

- Objects are collections of related data stored in key-value pairs.
- They can contain primitive values (like numbers, strings) and reference values (like arrays or other objects).
- Objects are reference types, meaning variables store a reference (memory address) to the object instead of the actual value.
- Object properties are written as "key: value" pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

JavaScript Objects

- **Creating an Object:**

➢ JavaScript objects can be created using curly brackets {} with a list of properties inside. A property is a "key: value" pair, where a key is a string (also called a " property name"), and value can be anything.

➢ Example:

```
<script>
    //Creating an object:
    var college = {
        faculty: "BSc. CSIT",
        semester: "5th",
        year: "2025",
        subject: "Web Technology"
    }
</script>
```

JavaScript Objects

- **Accessing Object Properties:**

➢ We can access an object's properties using two methods:

i. **Dot notation:** objectName.propertyName

ii. **Bracket notation:** objectName["propertyName"]

➢ Examples:

Using Dot Notation:

```
// Create an object
var college = {
    faculty: "BSc. CSIT",
    year: 2017,
    subject: "Web Development"
};

// Accessing properties using dot notation
document.write("The college has " + college.faculty);
```

Using Bracket Notation

```
// Create an object
var college = {
    faculty: "BSc. CSIT",
    year: 2017,
    subject: "Web Development"
};

// Accessing properties using bracket notation
document.write("The college has " + college["faculty"]);
```

JavaScript Objects

- **Modifying Object Properties:**

➢ We can update or add properties to an exiting object by simply giving a value.

➢ Example:

```
//Modifying Object Properties
college.semester = "fifth"; //Updating existing value
college.university = "TU"; //Adding new property
```

- **Deleting Object Property:**

➢ The delete keyword is used to remove a property from an object. Once deleted, attempting to access the property will return undefined.

```
// Deleting Object Property
delete college.semester;
```

JavaScript Objects

- **User-Defined Objects**
  - All user-defined objects and built-in objects are descendants of an object called Object.
  - Here are some the common ways of creating user-defined objects:

i. By Creating Instance of Object (new Object()) :

- A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called Object() to build the object. The return value of the Object constructor is assigned to a variable.

```
<script type=text/javascript>
    // Create an object using Object() constructor
    var student = new Object();

    // Assign properties to the object
    student.name = "Gunjan Yadav";
    student.rollNo = "1";
    student.address = "Biratnagar";

    // Display peroperties
    document.write("Student name: " + student.name + "<br>");
    document.write("Roll no.: " + student.rollNo + "<br>");
    document.write("Address: " + student.address + "<br>");
</script>
```

JavaScript Objects

ii. Using Constructor Functions :

- A constructor function initializes multiple objects with the same structure.

```
<script type=text/javascript>
    // Create and object using Constructor Functions
    function Student(name, rollNo, address){
        this.name = name;
        this.rollNo = rollNo;
        this.address = address;
    }

    // Creating object instances
    var gunjan = new Student("Gunjan Yadav", "1", "Biratnagar-001");
    var pawan = new Student("Pawan Dahal", "2", "Biratnagar-002");

    document.write("Hello, " + gunjan.name + ". Are you from " + gunjan.address + "? <br>");
    document.write("Hello, " + pawan.name + ". Are you from " + pawan.address + "? <br>");
</script>
```

JavaScript Objects

- **Defining method in JavaScript Object**
- We can define method in JavaScript object. But before defining method, we need to add property in the function with same name as method.
- The example of defining method in object is given below:

```
<script>
  function emp(id, name, salary) {
    this.id = id;
    this.name = name;
    this.salary = salary;

    this.changeSalary = changeSalary;
    function changeSalary(otherSalary) {
      this.salary = otherSalary;
    }
  }
  e = new emp(103, "Sonoo Jaiswal", 30000);
  document.write(e.id + " " + e.name + " " + e.salary);
  e.changeSalary(45000);
  document.write("<br>" + e.id + " " + e.name + " " + e.salary);
</script>
```

JavaScript Array

- The Array object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Array in JavaScript are indexed from 0.
- **Array Declaration:**
- An array can be declared using any one of the following techniques:

```
var array1 = new Array(); // Using the Array constructor
var array2 = []; // Using array literal (preferred)
```

- Example: **Creating and Accessing an Array**

```
<script type="text/javascript">
    // Creating and Accessing an Array
    var colleges = ["BKC", "HDC", "MMAMC", "NCMIT"];
    console.log(colleges[0]); //BKC
    console.log(colleges[1]); //HDC
    console.log(colleges[2]); //MMAMC
</script>
```

JavaScript Array

- **JavaScript Array Constructor (new keyword)**

➢ Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

```
<script>
  var i;
  var emp = new Array();
  emp[0] = "Ajay";
  emp[1] = "Abin";
  emp[2] = "Aashutosh";

  for (i = 0; i < emp.length; i++) {
    document.write(emp[i] + "<br>");
  }
</script>
```

JavaScript Array

JavaScript Array Methods

- **.push():** Adds one or more elements to the end of an array.
- **.pop():** Removes the last element from an array.
- **.shift():** Removes the first element from an array.
- **.unshift():** Adds one or more elements to the beginning of an array.
- **.map():** Creates a new array by applying a function to each element of the array.
- **.filter():** Creates a new array with all elements that pass a test.
- **.reduce():** Reduces an array to a single value by executing a function for each element.
- **.forEach():** Executes a provided function once for each array element.
- **.find():** Returns the first element that passes a test.
- **.findIndex():** Returns the index of the first element that passes a test.
- **.slice():** Returns a shallow copy of a portion of an array into a new array.
- **.splice():** Adds or removes elements from an array.
- **.concat():** Merges two or more arrays.
- **.includes():** Checks if an array contains a specified element.
- **.some():** Tests whether at least one element in the array passes a test.
- **.every():** Tests whether all elements in the array pass a test.
- **.indexOf():** Returns the first index of a specified element in the array.
- **.join():** Joins all elements of an array into a string.

JavaScript Array

❑ Array Methods

i. **concat():** Combines two or more arrays into one new array.

```
var alpha = ["a", "b", "c"];
var numeric = [1, 2, 3];
var alphaNumeric = alpha.concat(numeric);
document.write(alphaNumeric); // a,b,c,1,2,3
```

ii. **length:** Returns the Number of Elements Finds how many elements are in an array.

```
var subjects = ["Web", "DBMS", "C Programming"];
document.write(subjects.length); // 3
```

iii. **join():** Returns a string by joining all elements with a separator.

```
var arr = ["First", "Second", "Third"];
document.write(arr.join()); // First,Second,Third
document.write("<br>" + arr.join(", ")); // First, Second, Third
document.write("<br>" + arr.join(" + ")); // First + Second + Third
```

JavaScript Array

iv. **indexOf():** Finds Index of an Element Returns the first occurrence index of an element; returns -1 if not found.

```
var subjects = ["Web", "DBMS", "Web", "Microprocessor"];
document.write(subjects.indexOf("Web")); // 0
document.write("<br>" + subjects.indexOf("Web", 2)); // 2 (search starts from index 2)
```

v. **lastIndexOf():** Returns the last occurrence index of an element.

```
<script type="text/javascript">
    var colleges = ["BKC", "HDC", "BKC", "MMAMC", "NCMIT", "BKC"];
    console.log(colleges.lastIndexOf("BKC")); //5
    console.log(colleges.lastIndexOf("BKC", 4)); //2 (search starts from index 4)
</script>
```

vi. **pop():** Removes the last element and returns it.

```
var numbers = [6, 2, 9, 10];
numbers.pop();
document.write(numbers); // 6,2,9
```

JavaScript Array

vii. **push():** Adds a new element at the end of the array.

```
var numbers = [6, 2, 9];
numbers.push(15);
document.write(numbers); // 6,2,9,15
```

viii. **reverse():** Reverses the order of elements in an array.

```
var numbers = [6, 2, 9, 10];
numbers.reverse();
document.write(numbers); // 10,9,2,6
```

ix. **shift():** Removes the first element and shifts remaining elements left.

```
var numbers = [6, 2, 9, 10];
numbers.shift();
document.write(numbers); // 2,9,10
```

JavaScript Array

x. **unshift():** Adds a new element at the beginning of an array.

```
var numbers = [6, 2, 9];
numbers.unshift(15);
document.write(numbers); // 15,6,2,9
```

xi. **sort():** Sorts elements in alphabetical order by default.

```
var fruits = ["banana", "mango", "orange", "apple"];
fruits.sort();
document.write(fruits); // apple,banana,mango,orange
```

xii. **slice():** Returns a new array with selected elements (without modifying the original array).

```
var arr = ["orange", "mango", "banana", "sugar"];
document.write(arr.slice(1, 3)); // mango,banana
```

xiii. **toString():** Returns a comma-separated string representation of an array.

```
var arr = ["orange", "mango", "banana"];
document.write(arr.toString()); // orange,mango,banana
```

JavaScript Array

xiv. **map():** Transforms every element in an array and returns a new array.

- Example: You have an array of numbers and want to multiply each number by 2.

```
const numbers = [1, 2, 3, 4];
const doubled = numbers.map((num) => num * 2);
```

xv. **filter():** Filters elements in an array based on a condition and returns a new array with only elements that meet the condition.

- Example: You have an array of ages and want to filter out people who are 18 or older.

```
const ages = [16, 18, 21, 15, 30];
const adults = ages.filter((age) => age >= 18);
```

xvi. **forEach():** Executes a provided function once for each array element. It doesn't return anything.

- Example: You have an array of names and want to log each name to the console.

```
const names = ["Ayush", "Ram", "Shyam"];
names.forEach((name) => document.write(name + " "));
```

JavaScript Math

- The JavaScript math object provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors.

- **Math.sqrt(n)**

- The JavaScript math.sqrt(n) method returns the square root of the given number.

```
Square Root of 17 is: <span id="p1"></span>
<script>
document.getElementById('p1').innerHTML=Math.sqrt(17);
</script>
```

- **Math.random()**

- The JavaScript math.random() method returns the random number between 0 to 1.

```
Random Number is: <span id="p2"></span>
<script>
document.getElementById('p2').innerHTML=Math.random();
</script>
```

JavaScript Math

- **Math.pow(m,n)**

➢ The JavaScript math.pow(m,n) method returns the m to the power of n that is mn.

```
3 to the power of 4 is: <span id="p3"></span>
<script>
document.getElementById('p3').innerHTML=Math.pow(3,4);
</script>
```

- **Math.floor(n)**

➢ The JavaScript Math.floor(n) method returns the lowest integer for the given number. For example 3 for 3.7, 5 for 5.9 etc.

```
Floor of 4.6 is: <span id="p4"></span>
<script>
document.getElementById('p4').innerHTML=Math.floor(4.6);
</script>
```

JavaScript Math

- **Math.ceil(n)**

➢ The JavaScript math.ceil(n) method returns the largest integer for the given number. For example 4 for 3.7, 6 for 5.9 etc.

```
Ceil of 4.6 is: <span id="p5"></span>
<script>
document.getElementById('p5').innerHTML=Math.ceil(4.6);
</script>
```

- **Math.round(n)**

➢ The JavaScript math.round(n) method returns the rounded integer nearest for the given number. If fractional part is equal or greater than 0.5, it goes to upper value 1 otherwise lower value 0. For example 4 for 3.7, 3 for 3.3, 6 for 5.9 etc.

```
Round of 4.3 is: <span id="p6"></span><br>
Round of 4.7 is: <span id="p7"></span>
<script>
document.getElementById('p6').innerHTML=Math.round(4.3);
document.getElementById('p7').innerHTML=Math.round(4.7);
</script>
```

JavaScript Math

- **Math.abs(n)**

➢ The JavaScript Math.abs(n) method returns the absolute value for the given number. For example 4 for -4, 6.6 for -6.6 etc.

```
Absolute value of -4 is: <span id="p8"></span>
<script>
document.getElementById('p8').innerHTML=Math.abs(-4);
</script>
```

JavaScript Events

- The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser.
- When javascript code is included in HTML, js react over these events and allow the execution.
- This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

JavaScript Events

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

JavaScript Events

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

JavaScript Callback

- A callback function can be defined as a function passed into another function as a parameter.
- Don't relate the callback with the keyword, as the callback is just a name of an argument that is passed to a function.
- In other words, we can say that a function passed to another function as an argument is referred to as a callback function.
- The callback function runs after the completion of the outer function. It is useful to develop an asynchronous JavaScript code.
- In JavaScript, a callback is easier to create. That is, we simply have to pass the callback function as a parameter to another function and call it right after the completion of the task.
- Callbacks are mainly used to handle the asynchronous operations such as the registering event listeners, fetching or inserting some data into/from the file, and many more.

```
<script>
    window.onload = calculate(2, 2, findSquare);
    function calculate(a, b, callback){
        sum = a + b;
        callback(sum);
    }
    function findSquare(num){
        document.write(num * num);
    }
</script>
```

JavaScript Promises

- JavaScript Promises are a modern way to handle asynchronous operations and represent values that are not immediately available but will be resolved in the future. They provide a more readable and manageable approach to asynchronous programming compared to traditional callback-based methods.

- **Basic Concept**

- A Promise is an object representing the eventual completion or failure of an asynchronous operation. A Promise can be in one of three states:
  i. Pending: The initial state, neither fulfilled nor rejected.
  ii. Fulfilled: The operation completed successfully.
  iii. Rejected: The operation failed.

JavaScript Async/Await

- **async** and **await** are syntax features in JavaScript that simplify working with Promises, making asynchronous code easier to write and read.
- They allow you to write asynchronous code in a way that looks synchronous, avoiding the complexities of chaining .then() and .catch().

- **async Function**

- An async function is a function declared with the async keyword. It always returns a Promise. If the function returns a value, that value is automatically wrapped in a resolved Promise. If the function throws an error, it is automatically wrapped in a rejected Promise.

```
<script>
  async function example() {
    return "Hello, World!";
  }

  // Calling the async function
  example().then((result) => console.log(result)); // Output: "Hello, World!"
</script>
```

JavaScript Async/Await

- **await Expression**

➢ The await keyword can only be used inside async functions. It pauses the execution of the async function and waits for the Promise to resolve or reject. Once the Promise is resolved, await returns the resolved value. If the Promise is rejected, await throws the rejected value as an error.

```
<script>
  async function example() {
    let result = await new Promise((resolve, reject) => {
      setTimeout(() => resolve("Hello, World!"), 5000);
    });
    console.log(result); // Output: "Hello, World!"
  }

  example();
</script>
```

JavaScript try…catch Statement

- In JavaScript, the try...catch statement allows you to handle exceptions (errors) gracefully. This mechanism helps prevent the application from crashing and allows you to manage errors in a controlled manner.

i. **try Block:** Contains the code that might throw an exception. If an error occurs, control is transferred to the catch block.

ii. **catch Block:** Contains the code to handle the exception. The error object, which holds information about the error, is passed to this block as a parameter. You can use this object to get details about the error.

iii. **finally Block (Optional):** Contains code that will run regardless of whether an exception was thrown or not. It's useful for cleanup operations, such as closing files or releasing resources.

```
try {
    // Code that may throw an exception
} catch (error) {
    // Code to handle the exception
} finally {
    // Code that will always execute, regardless of whether an exception occurred
}
```

The End