UNIT-4
Java Script

❖ Introduction to javascript:

- JavaScript is a lightweight because it has low CPU usage, is easy to implement, cross-platform, single-threaded, and interpreted compiled programming language.
- It is also known as the scripting language for webpages. It is well-known for the development of web pages, and many non-browser environments also use it.
- JavaScript can be used for Client-side developments as well as Server-side developments.
- JavaScript contains a standard library of objects, like Array, Date, and Math, and a core set of language elements like operators, control structures, and statements.
- JavaScript is a high-level, dynamic programming language used to make web pages interactive.
- It is a core technology of the World Wide Web, alongside HTML and CSS.
- JavaScript is a versatile programming language primarily used for web development.
- It adds interactivity, dynamic content, and enhanced user experiences to websites. With JavaScript, you can create features like sliders, interactive forms, animations, and more.
- It's also widely used in building web applications, backend services (using Node.js), game development, and even mobile apps.
- JavaScript is a case-sensitive language.
- JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
- JavaScript is supportable in several operating systems including, Windows, macOS, etc.

History of JavaScript:

- It was created by Brendan Eich in 1995. He was an engineer at Netscape.
- It was originally going to be named LiveScript but was renamed.
- Unlike most programming languages, JavaScript language has no concept of input or output.

- It is designed to run as a scripting language in a host environment, and it is up to the host environment to provide mechanisms for communicating with the outside world.
- The most common host environment is the browser.
- JavaScript is the most popular language on earth.

Applications of JavaScript:

- Web Development, Web Applications, Server Applications, Games, Smartwatches, Art, Machine Learning, Mobile Applications, 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.

Advantages of Javascript:

- **Javascript can react to events-** A javascript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.
- **Javascript can be used to validate data -** A javascript can be used to validate form data before it is submitted to server.
- **Javascript can be used to detect the visitor's browser –** A javascript can be used to detect the visitor's browser and depending on the browser – load another page specifically designed for that browser.
- **Javascript can be used to create cookies –** A javascript can be used to store and retrieve information on the visitor's computer where to insert javascript.

Elements of Javascript:

- A javascript is a series of statements embedded in an HTML file, in between <script> and </script> tag.
- Several javascript statements grouped together in a statement block.
- Comments are useful for a good javascript program.
- Comments can be added to explain the javascript, or to make it more readable.
- 1. Javascript statements

- 2. Javascript code
- 3. Statements blocks
- 4. Comments.

1. Javascript statements:

- A javascript statement is a command to a browser. The purpose of the command is to tell the browser what to do.
- This javascript statement tells the browser to write "welcome javascript" to the web page.
- **Example:**
- Document.write("welcome javascript");
- It is normal to add a semicolon at the end of each executable statement.
- Most people think this is a good programming practice, and most often you will see this in javascript above example on the web.
- The semicolon is optional in javascript, and the browser is supposed to interpret the end of the line as the end of the statement.
- Because of this you will often see examples without the semicolon at the end.

2. Javascript code:

- Javascript code(or just javascript) is a sequence of javascript statements.
- Each statement is executed by the browser in the sequence they are written.

- **Example:**

```
<script type="text/javascript">
        document.write("welcome");
        document.write("<h1>this is heading1</h1>");
        document.write("<b>this is bold text</b>");
        document.write("<h2>this is heading2</h2>");
</script>
```

3. Javascript Blocks:

- Javascript statements can be grouped together in blocks.
- Blocks start with a left curly bracket {, and end with a right curly bracket }.
- The purpose of a block is to make the sequence of statements execute together.

- **Example:**

```
<script type="text/javascript">
{
        document.write("welcome");
        document.write("<h1>this is heading1</h1>");
        document.write("<p>this is paragraph</p>");
        document.write("<h2>this is heading2</h2>");
}
</script>
```

4. Javascript comments:

- JavaScript comments can be used to explain JavaScript code, and to make it more readable.
- JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

- Single line comments start with //.
- Any text between // and the end of the line will be ignored by JavaScript (will not be executed).
- **Example:**
- //this is single line comment

Multi-line Comments

- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored by JavaScript.
- **Example:**
- /* this is
- multi line comment */
- It is most common to use single line comments. Block comments are often used for formal documentation.

❖ How to Link JavaScript File in HTML ?

- JavaScript can be included in an HTML document in three ways:
- **Inline**, by placing it directly within HTML elements.
- **Internal**, by placing it within a <script> tag in the HTML document.
- **External**, by linking to a separate .js file using the <script src=”filename.js”></script> tag.

1) **Inline JS:** Inline JavaScript refers to JavaScript code embedded directly within HTML elements using the onclick, onmouseover, or other event attributes. This allows you to execute JavaScript code in response to user interactions or specific events without needing a separate JavaScript file or script block.

Syntax:

- <html elemment><script> //javascript code</script></ html elemment >

Example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>inline javascript</title>
    </head>
    <body>
        <p><script>document.write("welcome");</script></p>
    </body>
</html>
```

2) **Internal JS:** We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement.

1. **Script in Head Section:**
   Script to be executed when the page loads is written in the head section.

Syntax:

```
<head>
    <script>
        // JavaScript Code
    </script>
</head>
```

Example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>internal javascript</title>
        <script>
```

```
                document.write("welcome, this is head section");
            </script>
        </head>
        <body>
        </body>
    </html>
```

2. **Script in Body Section:**
   Scripts to be executed when they are called, or when an event is triggered, is written in the body section.

Syntax:

```
<body>
    <script>
        // JavaScript Code
    </script>
</body>
```

Example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>internal javascript</title>
    </head>
    <body>
        <script>
            document.write("welcome this is body section ");
        </script>
    </body>
</html>
```

3. **Scripts in Both the Body and The Head Section:**
   You can place an unlimited number of scripts in your document; you can insert scripts in the body and head section together.

Syntax:

```
<html>
```

```
<head>
    <script>
        // JavaScript Code
    </script>
</head>
<body>
    <script>
        // JavaScript Code
    </script>
</body>
</html>
```

Example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>internal javascript</title>
        <script type="text/javascript">
            document.write("welcome ");
        </script>
    </head>
    <body>
        <script type="text/javascript">
            document.write("javascript");
        </script>
    </body>
</html>
```

3) **External JS:** We can write JavaScript code in another files having an extension.js and then link this file inside the <head> tag of the HTML file in which we want to add this code.

- Sometimes you might want to run the same javascript on several pages, you can do it by writing a javascript in an external file.
- It must be saved with .js file extension.

- The external script cannot contain the <script> tag.
- To use the external script, point to the.js file in the "src" attribute of <script> tag (in (head or body) html file).

Syntax:

- 1. Create a javascript file with .js extension
- 2. Create an html file with .html extension and add script src.

```
<html>
        <head>
                <title>javascript</title>
                <script src="script.js">  </script>
        </head>
        <body>
        </body>
</html>
```

Example:

1. **Script.js**

```
document.write("this is javascript file");
```

2. **File.html**

```
<!DOCTYPE html>
   <html>
   <head>
          <title>javascript</title>
          <script src="script.js">
          </script>
   </head>
   <body>
          <p>welcome</p>
   </body>
   </html>
```

❖ Variables:

- Variables are used to store data in JavaScript.
- Variables are used to store reusable values.
- The values of the variables are allocated using the assignment operator("=").
- Variables are Containers for Storing Data or values.

- **Basic rules to declare a variable in JavaScript:**
- These are case-sensitive. For example, 'A' and 'a' are different variables.
- Can only begin with a letter(a to z or A to Z), underscore("_") or "$" (dollar) symbol.
- It can contain letters, numbers, underscore, or "$" symbol
- A variable name cannot be a reserved keyword.

Variable Declaration:

```
var x=10;  //valid
var _value="abcd";  //valid
var value123=15; //valid
var x=10; //valid
var $=20; //valid
var $a=20; //valid
var 12=20; //invalid
var #x=25; //invalid
var *y=10; //invalid
```

- **JavaScript Variables Can Be Declared In 4 Ways:**
- 1.Automatically
- 2. Using var
- 3. Using let
- 4. Using const

- In this first example, x, y, and z are undeclared variables.
- They are automatically declared when first used:
- y = "hello"
- let $ = "Welcome"
- const _example = "abc"

- **Example**
- x = 5;
- y = 6;
- z = x + y;
- x stores the value 5
- y stores the value 6
- z stores the value 11

- The var keyword was used in all JavaScript code from 1995 to 2015.
- The var keyword should only be used in code written for older browsers.
- The let and const keywords were added to JavaScript in 2015.

Example Automatic Declaration:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>variable keywords javascript</title>
            <script>
                x=40;
                var x=50;
                y=60;
                y= 10;
                document.write(x);
                document.write(y);
                /* not allowed
                    z=20;
                    let z=80;
                    const z=50;
                    document.write(z);     */
            </script>
        </head>
        <body>
        </body>
    </html>
```

Example: (Using let)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript</title>
            <script>
                let x=30;
                x=10;
                document.write(x);
                /* not allowed
                let y=30;
                const y=30;
```

```
                var y=20;
                let y=50;
                document.write(y);  */
            </script>
        </head>
        <body>
        </body>
    </html>
```

Example: (Using var)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript</title>
            <script>
                var x=30;
                x=10;
                var x=50;
                document.write(x);
                /* not allowed
                var y=30;
                const y=30;
                let y=20;
                document.write(y); */
            </script>
        </head>
        <body>
            <script>
            </script>
        </body>
    </html>
```

Example: (Using const)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>variable keywords javascript</title>
            <script>
                const x=40;
                document.write(x);
```

```
                    /* not allowed
                     const x=40;
                     x=80;
                     const x=10;
                     let x=50;
                     var x=60;
                    document.write(x); */
                </script>
            </head>
            <body>
            </body>
        </html>
```

Mixed Example

```
<!DOCTYPE html>
    <html>
        <body>
            <h1>JavaScript Variables</h1>
            <p>In this example, price1, price2, and total are variables.</p>
            <script>
                const price1 = 5;
                const price2 = 6;
                let total = price1 + price2;
                document.write("price1 is  ",price1+"<br>");
                document.write("price2 is  ",price2+"<br>");
                document.write("The total is:   ",total);
            </script>
        </body>
    </html>
```

Description:

- The two variables price1 and price2 are declared with the const keyword.
- These are constant values and cannot be changed.
- The variable total is declared with the let keyword.
- The value total can be changed.

When to Use var, let, or const?

- Always declare variables

- Always use const if the value should not be changed.(Used to declare a fixed value that cannot be changed.)
- Always use const if the type should not be changed (Arrays and Objects)
- Only use let if you can't use const
- Only use var if you MUST support old browsers. (Used to initialize to value, redeclared and its value can be reassigned.)

Types of Variable in Javascript:

- 1. Global variable
- 2. Local variable

1. **Global variable:**

- A global variable is accessible from any function. A variable declared outside of the function or declared with window object is known as global variable.

2. **Local variable:**

- A variable which is declared inside block or function is called local variable. It is accessible with in the function or block only.

Example: (Local & Global Variable)

```
<!DOCTYPE html>
	<html>
		<head>
			<title>local global variable</title>
			<script>
				var a=10; //global variable
				function fun() {
					var b = 20 //local variable
					document.write(a,b);  }
				fun();
				document.write(a);
				document.write(b);
			</script>
		</head>
	<body>
	</body>
</html>
```

❖ Data Types:

- Every variable has a data type.
- Data types indicate what kind of data the variable holds.
- In javascript, the type of data variable hold can be grouped into 2 categories.
  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.
- **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

- **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

❖ JavaScript Operators:

- JavaScript Operators are symbols used to perform specific mathematical, comparison, assignment, and logical computations on operands.
- They are fundamental elements in JavaScript programming, allowing developers to manipulate data and control program flow efficiently.
- There are various operators supported by JavaScript.
- Operators and Operands
- The numbers (in an arithmetic operation) are called operands.

- The operation (to be performed between the two operands) is defined by an operator.

Operand	Operator	Operand
100	+	50

1) JavaScript Arithmetic Operators:

- Arithmetic operators perform arithmetic on numbers
- JavaScript Arithmetic Operators perform arithmetic operations: addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponentiation (**).

No.	Operator	Name & Description	Syntax	Example

1.	+	**Addition** '+' operator performs addition on two operands. This '+' operator can also be used to concatenate (add) strings.	10+20 = 30
			Y = "wel" + "co" + "me" gives Y = "welcome"
				// Addition Operator
				let a = 1 + 2
				console.log(a);

2.	-	**Subtraction** '-' operator performs subtraction on two operands.	Y = 5 – 3 gives Y = 2	// Subtraction Operator
				let a = 5- 2
				console.log(a);

3.	*	**Multiplication** '*' operator performs multiplication on two operands.	Y = 5 * 5 gives Y = 25	// Multiplication Operator
				let a = 5*2
				console.log(a);

4.	/	**Division** '/' operator performs division on two operands	Y = 5 / 5 gives Y = 1	// Division Operator
				let d = 10 / 2
				console.log(d);
				//document.write ("sum is",sum);

5.	%	**Modulus (Remainder)** '%' operator gives a remainder of an **integer** division.	A % B means remainder (A/B) Y = 5 % 4 gives Y = 1	// Modulas Operator let e = 5 % 2 console.log(e)

6.	++	**Increment** '++' operator increases an integer value by one.	let A = 10 and Y = A + + then A = 11, Y=10	// Increment Operator
				let g = 2;
				g1 = g++;
				console.log(g)

7.	--	**Decrement** '--' operator decreases an integer value by one.	let A = 10 and Y = A – – then A = 9, Y=10	// Decrement Operator
				let g = 2;
				g1 = g--;
				console.log(g)

8.	+(Unary)	**Unary** '+' is the fastest and preferred way of converting something into a number	+a means a is a positive number	// Unary plus Operator
				let i = 3;
				i1 = +i;
				console.log(i1)

9.	-(Negation)	**Negation** '-' operator gives the negation of an operand.	-a means a is a negative number	// Negation Operator
				let j = 3;
				j1 = -j;
				console.log(j1)

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript operators</title>
            <script>
                let x=6;
                let y=5;
                let z;
                document.write("frist value is"+" ",x +"<br>");
                document.write("second value is"+" ",y +"<br>");

                //arithmetic operators
                z=x+y; // addition
                document.write("sum is  ",z +"<br>");
                z=x-y; //subtraction
                document.write("minus is  ",z +"<br>");
                z=x*y; //multiplication
```

```
                document.write("multiplication is  ",z +"<br>");
                z=x/y; //division
                document.write("division is  ",z +"<br>");
                z=x%y; //modulas(remainder)
                document.write("modulas is  ",z +"<br>");
                x1=x++; //increment
                document.write("increment is  ",x+"<br>");
                let a=5;
                a1=a--; //Decrement
                document.write("decrement is  ",a+"<br>");
                let i = 4;
                let i1 = +i; //unary +
                document.write("positive number is  ",i1+"<br>");
                let i2 = -i; //Negation -
                document.write("negative number is  ",i2+"<br>");
            </script>
        </head>
        <body>
        </body>
</html>
```

2) JavaScript Assignment Operators:

- Assignment operators assign values to JavaScript variables.
- The assignment operation evaluates the assigned value.

No.	Operator	Name & Description	syntax	Example

1.	=	The Simple **Assignment** Operator assigns a value to a variable. Assign operator assigns the right operand value to the left operand.	If A = 10 and Y = A then Y = 10	// Assignment Operator
				let a = 2;
				console.log(a);

2.	+=	**Add and assign**
		The Addition Assignment Operator adds a value to a variable.
		Sums up left and right operand values and then assigns the result to the left operand.
			var a=10;
			a+=20; Now a = 30
			or
			Y += 1 gives Y = Y + 1
				let x = 10;
				x += 5;
				console.log(x);
				or
				let text = "Hello";
				text += " World";
				console.log(text);

3.	-=	**Subtract and assign**
		The Subtraction Assignment Operator subtracts a value from a variable.
		It subtracts the right side value from the left side value and then assigns the result to the left operand.
			var a=20; a-=10; Now a = 10
			Or
			Y -= 1 gives Y = Y – 1
				// Subtract Assignment Operator
				let x = 10;
				x -= 5;
				console.log(x);

4.	*=	**Multiply and assign**
		The Multiplication Assignment Operator multiplies a variable.
		It multiplies a variable by the value of the right operand and assigns the result to the variable.
			var a=10; a*=20; Now a = 200
			Or
			Y *= A is equivalent to Y = Y * A
				// Multiply Assignment Operator
				let x = 10;
				x *= 5;
				console.log(x);

5.	/=	**Divide and assign**
		The Division Assignment Operator divides a variable.
		It divides a variable by the value of the right operand and assigns the result to the variable.
			var a=10; a/=2; Now a = 5
			Or
			Y /= A is equivalent to Y = Y / A
				// Divide Assignment Operator
				let x = 10;
				x /= 5;
				console.log(x);

6.	%=	**Modulus/Remainder and assign**
		The Remainder Assignment Operator assigns a remainder to a variable.
		It divides a variable by the value of the right operand and assigns the remainder to the variable.
			var a=10; a%=2; Now a = 0
			Or
			Y %= A is equivalent to Y = Y % A
				// Modulus Assignment Operator
				let x = 10;
				x %= 4;
				console.log(x);

7.	**=	**Exponentiation Assignment**
		The Exponentiation Assignment Operator raises a variable to the power of the operand.
		This raises the value of a variable to the power of the right operand.
			Y **= A is equivalent to Y=Y ** A	// Exponention Assignment Operator
				let x = 10;
				x **= 5;
				console.log(x);

Example: (Assignment Operator)

```
<!DOCTYPE html>
    <html>
        <head>
        <title>javascript operators</title>
        <script>
            //assignment operators
            let x=6; //assignment operator
            let y=5;
            document.write("frist value is"+" ",x +"<br>");
            document.write("second value is"+" ",y +"<br>");
            x+=20;  //Add and assign
            document.write("addition value is"+" ",x +"<br>");
            x=6;
            x-=2;  //Subtract and assign
            document.write("minus is  ",x +"<br>");
            x=4;
            x*=5;  //Multiply and assign
            document.write("multiplication is  ",x +"<br>");
            x=10;
            x/=2;  //Divide and assign
            document.write("Division is  ",x +"<br>");
            x=5;
            x%=2; // Modulus/Remainder and assign
            document.write("Reminder is  ",x + "<br>");
            x=2;
            y=4;
            x**=y; //Exponentiation Assignment
            document.write("Exponention  is  ",x + "<br>");
        </script>
    </head>
    <body>
    </body>
</html>
```

3) JavaScript Comparison Operators:

- Comparison operators are mainly used to perform the logical operations and used in logical statements to determine the equality or difference between the values.
- Comparison operators are used to test for true or false.
- Relational operators (also called comparison operators) compare two expressions.
- The JavaScript comparison operator compares the two operands.
- The comparison operators are as follows:

No.	Operator	Name & Description	Syntax	Example

1.	==	**Is equal to**
		Compares the equality of two operands. If equal then the condition is true otherwise false.
			Y = 5 and X = 6 Y = = X is false.
			OR
			10==20 = false
				// Assigning values
				let val1 = 5;
				let val2 = 5;
				// Equality Operator
				console.log(val1 == val2);

2.	===	**Identical (equal value and equal type) OR Strict equality Compares** the equality of two operands with type. If both value and type are equal then the condition is true otherwise false.	10==20 = false
			OR
			Y = 5 and X = ‘5’ Y = = = X is false.
				// Assigning values
				let val1 = 5;
				let val2 = '5';
				// Strict equality Operator
				console.log(val1 === val2);

3.	!=	**Not equal to OR Inequality**
		Compares inequality of two operands. True if operands are not equal.
			let X = 10 then X ! = 11 is true.
			OR
			10!=20 = true
				// Assigning values
				let val1 = 5;
				let val2 = 15;
				
				// Inequality Operator
				console.log(val1 != val2);

4.	!==	**Not IdenticalOR Strict inequality OR not equal value or not equal type**
		Compares the inequality of two operands with type. If both value
			let X = 10 then X ! == ’10’ is true.
			OR
			20!==20 =
				// Assigning values
				let val1 = 5;
				let val2 = "5";
				
				// Strict Inequality

		and type are equal then the condition is true otherwise false.	false	Operator console.log(val1 !== val2);

5.	>	**Greater than**
		This operator checks whether the left side value is greater than the right side value. If yes then it returns true otherwise it returns false.
			let X = 10
			then X > 11
			is false.
			OR
			20>10 = true
				// Assigning values
				let val1 = 15;
				let val2 = 5;
				// Greater than Operator
				console.log(val1 > val2);

6.	>=	**Greater than or equal to**
		This operator checks whether the left side operand is greater than or equal to the right side operand. If yes then it returns true otherwise it returns false.
			let X = 10
			then X > =
			11 is false.
			OR
			20>=10 = true
				// Assigning values
				let val1 = 15;
				let val2 = 5;
				// Greater than or equal Operator
				console.log(val1 >= val2);

7.	<	**Less than**
		This operator checks whether the left side value is less than the right side value. If yes then it returns true otherwise it returns false.
			let X = 10
			then X < 11
			is true.
			OR
			20<10 = false
				// Assigning values
				let val1 = 15;
				let val2 = 5;
				
				// Greater than or equal Operator
				console.log(val1 < val2);

8.	<=	**Less than or equal to**
		This operator checks whether the left side operand value is less than or equal to the right side operand value. If yes then it returns true otherwise it returns false.
			let X = 10
			then X < =
			10 is true.
			OR
			
			20<=10 = false
				// Assigning values
				let val1 = 15;
				let val2 = 5;
				
				// Greater than or equal Operator
				console.log(val1 <= val2);

Example: (Comparison Operators)

```
<!Doctype html>
    <html>
        <head>
```

```
<title>comparision operators</title>
<script>
    //comparision operators
    let x = 5; //assignment
    let y = 6; //assignment
    document.write("first value ",x +"<br>");
    document.write("second value ",y +"<br>");
    document.write("Is equal to  ", x==y); //Is equal to
    document.write("<br>");
    document.write("Strict equality Compares ",x===y);
    //Strict equality Compares
    document.write("Not equal to  ",x!=y);
    document.write("not equal value or not equal type
    ",x!==y); //not equal value or not equal type
    document.write("Greater than ",x>y); // Greater than
    document.write("Greater than or equal to ",x>=y); //
    Greater than or equal to
    document.write("Less than  ",x<y); // Less than
    document.write("Less than or equal to ",x<=y); // Less
    than or equal to
    let a=5; //assignment
    let b='5'; //assignment
    document.write("equality Compares ",a==b); // equality
    Compares
    document.write("Strictequality Compares ",a===b);
    //Strict equality Compares
</script>
</head>
<body>
</body>
</html>
```

4) JavaScript Logical Operators:

- Logical operators are used to determine the logic between variables or values.

No.	Operator	Name & Description	Syntax	Example
1.	&&	**Logical AND** Evaluates operands and	(10==20 && 20==33) = false	(10==20 && 20==33) =

		return true only if all are true	Or x = 6, y = 3 (x < 10 && y > 1) is true	false
2.	||	**Logical OR** Returns true even if one of the multiple operands is true	(10==20 || 20==33) = false Or x = 6, y = 3 (x == 5 || y == 5) is false	(10==20 || 20==33) = false
3.	!	**Logical Not** Converts operator to boolean and returns flipped value	!(10==20) = true Or x = 6, y = 3 !(x == y) is true	!(10==20) = true

Example: (Logical Operators)

```
<!Doctype html>
    <html>
        <head>
            <title>javascript operators</title>
            <script>
                //Logical operator
                let x=6;
                let y=5;
                document.write("first value ",x +"<br>");
                document.write("second value ",y +"<br>");
                document.write("logical and -",x < 10 && y > 1);
                document.write("<br>");
                document.write("logical or -",x < 4 || y > 7);
                document.write("<br>");
                document.write("logical not -",x!=y);
            </script>
        </head>
        <body>
        </body>
    </html>
```


5) Ternary (Conditional) Operators

- The ternary operator has three operands. It is the simplified operator of if/else.

No.	Operator	Name & Description	Syntax	Example
1.	?:	**Ternary Operator** Conditional Operator returns value based on the condition. It is like the short form of the if-else condition.	Y = ? A : B If the condition is true then Y = A otherwise Y = B	let result = (marks > 39) ? "Pass" : "Fail"; console.log(result);

6) Comma Operators:

- Comma Operator (,) mainly evaluates its operands from left to right sequentially and returns the value of the rightmost operand.

No.	Operator	Name & Description	Syntax	Example

1	,	comma operator
		When a comma operator is placed in an expression, it executes each expression and returns the right most expression.
			Expression1, Expression2, Expression3, …so on	let x = 1;
				let y = 2;
				let z = 3;
				document.write(x,y,z);
				console.log(x,y,z);

7) JavaScript String Operators

- JavaScript String Operators include concatenation (+) and concatenation assignment (+=), used to join strings or combine strings with other data types.

No.	Operator	Name & Description	Syntax	Example

1	+	concatenation operator
		It concatenates two string values together, returning another string that is the union of the two operand strings.
			str1 + str2	let str1 = "Geeks";
				let str2 = "forGeeks";
				let result = (str1 + str2);
				document.write(result);
				console.log(result);

Example: (Ternary (Conditional) Operator, Comma Operator and String Operator)

```
<!Doctype html>
```

```
<html>
    <head>
        <title>javascript operators</title>
        <script>
            //Ternary operator
            marks=40;
            let result = (marks > 39) ? "Pass" : "Fail";
            document.write(result);
            document.write("<br>");
            //Comma operator
            let x = 1;
            let y = 2;
            let z = 3;
            let b=(1,2,3);
            document.write(x,y,z);
            document.write("<br>");
            //String operator
            let str1="welcome ";
            let str2 = "javascript";
            let string = (str1 + str2);
            document.write(string);
        </script>
    </head>
    <body>
    </body>
</html>
```

❖ Conditional (Control) Statements:

- JavaScript conditional statements allow you to execute specific blocks of code based on conditions.
- If the condition is met, a particular block of code will run; otherwise, another block of code will execute based on the condition.
- There are several methods that can be used to perform Conditional Statements in JavaScript.

Conditional Statement	Description
if statement	Use if to specify a block of code to be executed, if a specified condition is true Executes a block of code if a specified condition is true.
else statement	Use else to specify a block of code to be executed, if the same condition is false Executes a block of code if the same condition of the preceding if statement is false.
else if statement	Use else if to specify a new condition to test, if the first condition is false Adds more conditions to the if statement, allowing for multiple alternative conditions to be tested.
switch statement	Use switch to specify many alternative blocks of code to be executed Evaluates an expression, then executes the case statement that matches the expression's value.
ternary operator	Provides a concise way to write if-else statements in a single line.
Nested if else statement	Allows for multiple conditions to be checked in a hierarchical manner.

1. Using if Statement

- The if statement is used to evaluate a particular condition. If the condition holds true, the associated code block is executed.
- Use this statement to execute some code only if a specified condition is true.

Syntax:

```
if ( condition ) {
  // If the condition is met,
  //code  will get executed.
}
```

Example: (If)

```
<!Doctype html>
    <html>
        <head>
            <title>javascript If Condition</title>
            <script>
```

```
                x=20;
                y=50;
                if(x<y)
                {  document.write("x is less than y");  }
            </script>
        </head>
        <body>
        </body>
    </html>
```

2. Using If-else Statement:

- The if-else statement will perform some action for a specific condition.
- Here use the else statement in which the else statement is written after the if statement and it has no condition in their code block.
- Use the if…else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax:

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

Example: (If-else)

```
<!Doctype html>
    <html>
        <head>
            <title>javascript If Else Condition</title>
            <script>
                x=20;
                y=50;
                if(x>y)
                { document.write("x is greater than y");  }
                else
                { document.write("x is not greater than y");}
            </script>
        </head>
        <body>
        </body>
    </html>
```

3. Else if Statement

- The else if statement in JavaScript allows handling multiple possible conditions and outputs, evaluating more than two options based on whether the conditions are true or false.
- Use else if to specify a new condition to test, if the first condition is false
- Use the if…elseif…else statement to select one of several blocks of code to be executed.

Syntax:

```
if (1st condition) {
   // Code for 1st condition }
else if (2nd condition) {
   // ode for 2nd condition }
else if (3rd condition) {
   // Code for 3rd condition }
else {
   //  ode that will execute if all
   // above conditions are false  }
```

Example: (If…Else if)

```
<!Doctype html>
    <html>
        <head>
            <title>javascript If…Elseif Condition</title>
            <script>
                x=50;
                y=50;
                if(x>y)
                {  document.write("x is greater than y");}
                else if(x<y)
                {  document.write("x is less than y");}
                else
                {  document.write("x is equal to y");}
            </script>
        </head>
        <body>
        </body>
    </html>
```

4. Using Switch Statement (JavaScript Switch Case):

- As the number of conditions increases, you can use multiple else-if statements in JavaScript. but when we dealing with many conditions, the switch statement may be a more preferred option.
- Conditional statements are used to perform different actions based on different conditions.
- Use the switch statement to select one of many blocks of code to be executed.
- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Syntax:

```
switch (expression) {
  case value1:
    statement1;
    break;
  case value2:
    statement2;
    break;
  . . .
  case valueN:
    statementN;
    break;
  default:
    statementDefault;
};
```

Example: (Switch)

```
<!Doctype html>
    <html>
        <head>
        <title>javascript Switch Case Example</title>
        <script>
              let day = 2;
              let dayName;
              switch(day) {
                    case 1:
                          dayName = "Monday";
```

```
                    break;
                case 2:
                    dayName = "Tuesday";
                    break;
                case 3:
                    dayName = "Wednesday";
                    break;
                default:
                    dayName = "Invalid day";
                }
                document.write(dayName);
        </script>
        </head>
        <body>
        </body>
    </html>
```

5. Using Ternary Operator (?:)

- The conditional operator, also referred to as the ternary operator (?:), is a shortcut for expressing conditional statements in JavaScript.

Syntax:

```
condition ? value if true : value if false
```

Example: (Ternary)

```
<!Doctype html>
    <html>
        <head>
            <title>javascript Ternary operators</title>
            <script>
                marks=40;
                let result = (marks > 39) ? "Pass" : "Fail";
                document.write(result);
            </script>
        </head>
        <body>
        </body>
    </html>
```

6. Nested If…else

- Nested if…else statements in JavaScript allow us to create complex conditional logic by checking multiple conditions in a hierarchical manner.

Each if statement can have an associated else block, and within each if or else block, you can nest another if…else statement.

- This nesting can continue to multiple levels, but it's important to maintain readability and avoid excessive complexity.

Syntax:

```
if (condition1)
{       // Code block 1
        if (condition2) {// Code block 2 }
        else { // Code block 3 }
}
else {
   // Code block 4
}
```

Example: (Nested if…else)

```
<!DOCTYPE html>
        <html>
                <head>
                        <title>javascript Nested If Example</title>
                </head>
                <body>
                        <script>
                                a=20;
                                if (a >= 10)
                                {   document.write("Num is more than 10.");
                                  if (a > 15)
                                   {document.write("Num is also more than 15.");  }
                                }
                                else
                                {  document.write("Num is less than 10.");  }
                        </script>
                </body>
        </html>
```

❖ JavaScript Loops, Break and Continue Statements:

- Loops can execute a block of code a number of times.
- JavaScript loops are essential for efficiently handling repetitive tasks. They execute a block of code repeatedly as long as a specified condition remains true.

- When the condition becomes false the loop stops and control is passes to the subsequent statement of the program.
- 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.
- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **for/of** - loops through the values of an iterable object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

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.
- The for loop is used when you know in advance how many times statements run.

Syntax

```
for (initialization; testing condition; increment/decrement) {
        statement(s)
        code to be executed
}
```

- For example, suppose we want to print “welcome” 5 times. This can be done using JS Loop easily. In Loop, the statement needs to be written only once and the loop will be executed 5 times as shown below.

Example: (For Loop)

```
<!Doctype html>
    <html>
        <head>
            <title>javascript For Loop Example</title>
            <script>
                for (x = 1; x <= 5; x++)
                {  document.write( x+"<br>");  }
            </script>
        </head>
        <body>
```

```
        </body>
    </html>
```

2. JavaScript while Loop:

- The while loop loops through a block of code as long as a specified condition is true.
- Loops executed a block of codes a specified number of times, or while a specified condition is true.
- The while loop loops through a block of code while a specified condition is true.
- A while loop is an example of an entry control loop.

Syntax

```
while (boolean condition) {
   loop statements...
}
```

Example: (While Loop)

```
<!Doctype html>
    <html>
        <head>
            <title>javascript While Loop Example</title>
            <script>
                let val = 1;
                while (val < 6) {
                    document.write(val);
                    document.write("<br>");
                    val=val+1;
                }
            </script>
        </head>
        <body>
        </body>
    </html>
```

3. JavaScript do-while Loop:

- The do while loop is a variant of the while loop.
- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- A do-while loop is an example of an exit control loop.

Syntax

```
do {
   Statements...
}
while (condition);
```

Example: (Do…While Loop)

```
<!Doctype html>
    <html>
        <head>
            <title>javascript Do...While Loop Example</title>
            <script>
                let test = 1;
                do {
                    document.write(test);
                    document.write("<br>");
                    test++;
                }
                while(test <= 5)
            </script>
        </head>
        <body>
        </body>
    </html>
```

4. JavaScript For In Statement Loop:

- The JavaScript for in statement loops through the properties of an Object
- The JavaScript for-in loop iterates over the enumerable properties of an object, allowing you to access each key or property name in turn.
- It's commonly used to traverse object properties but can also be used with arrays.
- Use the for-in loop to iterate over non-array objects. Even though we can use a for-in loop for an array, it is generally not recommended.

Syntax

```
for (variable_name in object_name) {
  // code block to be executed
}
```

Example: (For In Statement Loop)

```
<!Doctype html>
```

```
<html>
    <head>
        <title>javascript FOR IN Loop Example</title>
        <script>
            const person = {fname:"John", lname:"Doe", age:25};
            let text = "";
            for (let x in person)
            {
                text = person[x];
                document.write(text+"<br>");
            }
        </script>
    </head>
    <body>
    </body>
</html>
```

5. JavaScript For Of Statement Loop:

- The JavaScript for of statement loops through the values of an iterable object.
- It lets you loop over iterable data structures such as Arrays, Strings and more.

Syntax

```
for (variable of variable_name of  object_name) {
  // code block to be executed
}
for (variable of iterable) {
  // code block to be executed
}
```

Example: (For Of Statement Loop)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript FOR OF Loop Example</title>
            <script>
                const subject = ["c", "c++", "html"];
                let text = "";
```

```
                for (let x of subject) {
                        text = x;
                        document.write(text+"<br>");  }
                </script>
        </head>
        <body>
        </body>
</html>
```

❖ JavaScript Break and Continue:

- The break statement "jumps out" of a loop.
- The continue statement "jumps over" one iteration in the loop.
- The break statement is used to jump out of a loop.
- It breaks the loop and continues executing the code after the loop.

❖ The Break Statement:

- JS break statement is used to terminate the execution of the loop or switch statement when the condition is true.

Syntax

- break;

Example: (Break)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript Break Example</title>
            <script>
                let text="";
                let i;
                for (i=1;i<7;i++)
                {
                    if(i==4)
                    {
                        break;
                    }
                    text=i;
                    document.write(text);
                }
```

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

❖ The Continue Statement

- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.
- The continue statement "jumps over" one iteration in the loop. It breaks iteration in the loop and continues executing the next iteration in the loop.
- The major difference between the continue and break statement is that the break statement breaks out of the loop completely while continue is used to break one statement and iterate to the next statement.

Syntax

- Continue;

Example: (Continue)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript Continue Example</title>
            <script>
                let text="";
                let i;
                for (i=1;i<7;i++)
                {
                    if(i==4)
                    {
                        continue;
                    }
                    text=i;
                    document.write(text);
                }
            </script>
        </head>
    <body>
    </body>
```

```
</html>
```

❖ JavaScript Dialogue Boxes:

- Dialogue boxes are a kind of popup notification, this kind of informative functionality is used to show success, failure, or any particular/important notification to the user.
- JavaScript uses 3 kinds of dialog (popup) boxes:
- 1. Alert
- 2. Prompt
- 3. Confirm
- These dialog boxes can be of very much help in making our website look more attractive.
- **1. Alert Box:** An alert box is used on the website to show a warning message to the user that they have entered the wrong value other than what is required to fill in that position.
- An alert box can still be used for friendlier messages.
- The alert box gives only one button "OK" to select and proceed.

Syntax

```
window.alert("sometext");
```

- The window.alert() method can be written without the window prefix.
- alert("I am an alert box!");

Example: (Alert Box)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript Alert Box Example</title>
            <script>
                alert("This is an alert bo")
            </script>
        </head>
        <body>
        </body>
    </html>
```

- **2. Confirm box:** A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

- If the user clicks on the OK button, the window method confirm() will return true.
- If the user clicks on the Cancel button, then confirm() returns false and will show null.

Syntax

- window.confirm("sometext");
- The window.confirm() method can be written without the window prefix.

Example: (Confirm Box)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript Confirm Box Example</title>
            <script>
                confirm("This is a Confirm Box");
            </script>
        </head>
        <body>
        </body>
    </html>
```

- **3. Prompt Box:** A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks the OK button, the window method prompt() will return the entered value from the text box.
- If the user clicks the Cancel button, the window method prompt() returns null.

Syntax

- window.prompt("sometext","defaultText");
- The window.prompt() method can be written without the window prefix.

Example: (Prompt Box)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript Prompt Box Example</title>
            <script>
                prompt("This is Prompt box");
```

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

❖ JavaScript Arrays:

- An array is a special variable, which can hold more than one value:
- An array used to store multiple values in a single variable. It can hold various data types.Elements are accessed by their index, starting from 0.
- Arrays use numbers to access its "elements".
- Many programming languages support arrays with named indexes.
- JavaScript does not support arrays with named indexes.
- In JavaScript, arrays always use numbered indexes.

Creating an Array

Syntax:

- Var array_name=[item1,item2,…]

Example:

```
Var fruits=[“Banana”,”Apple”,”Orange”,”Grapes”];
Var fruits=new Array(“Banana”,”Apple”,”Orange”,”Grapes”);
const person = {firstName:"AAA", lastName:"BBB", age:46};
```

Access the element of an Array

Syntax:

- Var variable_name=array_name[index];
- Document.write(variable_name);

Example:

- Var fruit1=fruits[0];
  Document.write(fruit1);
- const person = {firstName:"AAA", lastName:"BBB", age:46};
  Document.write(person[“lastname”]);

Adding Element Into Array

Example:

```
Var fruits=[“Banana”,”Apple”,”Orange”,”Grapes”];
Fruits.[fruits.length]=”pineapple”;
Document.write(fruits);
```

Delete Element From Array

Example:

```
Var fruits=[“Banana”,”Apple”,”Orange”,”Grapes”,”pineapple”];
Delete fruits[2];
Documents.write(fruits);
```

Joining Two Arrays

Example:

```
Var fruits=[“Banana”,”Apple”,”Orange”,”Grapes”];
Var colors=[“blue”,”yellow”,”white”,”black”];
Var mix=fruits.concat(colors);
Document.write(mix);
```

Example: (Array)

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript Array Example</title>
            <script>
                var fruits=["Banana","Apple","Orange","Grapes"];
                document.write(fruits);
                fruits[fruits.length]="pineapple";
                document.write(fruits);
                delete fruits[2];
                document.write(fruits);
                var colors=["blue","yellow","white","black"];
                var mix=fruits.concat(colors);
                document.write(mix);
            </script>
        </head>
        <body>
        </body>
    </html>
```

❖ JavaScript User Define Function

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" calls it.
- In javascript, we can define our own functions, or we can use built-in functions already defined in the javascript language.

- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: {}

- **1. Function With Parameters:**
- When you call a function, you can pass along some values to it, these values are called parameters or arguments.
- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Syntax:

```
Function function_name(parameter1,parameter2,…)
{
    Some code to be executed
}
```

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>JavaScript Functions With Parameters</title>
        </head>
        <body>
            <script>
                function calculation(p1, p2)
                {
                    addition=p1+p2;
                    document.write(addition);
                }
                calculation(4,3);
            </script>
        </body>
```

```
</html>
```

- **2. Function With A Return Values:**
- Sometimes you want your function to return a value back to where the call was made.
- This is possible by using the return statement.
- When using the return statements, the function will stop executing and return the specified value.

Syntax:

```
Function function_name(parameter1,parameter2,…)
{
    Some code to be executed
     return value;
}
```

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>javascript Function Return Value Example</title>
            <script>
                function fun(n1,n2,n3)
                {
                    avg=(n1+n2+n3)/3;
                    return avg;

                }
                fun(10,20,30);
                document.write(avg);
            </script>
        </head>
        <body>
        </body>
    </html>
```

❖ Built in Function: string, Maths, Array, Date:

1. String Functions:

- JavaScript strings are the sequence of characters.

- JavaScript provides a bunch of string methods for representing and manipulating strings.
- JavaScript provides several built-in methods and properties for working with strings.

1) charAt()

- The charAt() method returns the character at a specified index (position) in a string:

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javascript String Function</title>
            <script>
                var str = "COMPUTER";
                document.write(str.charAt(2));
            </script>
        </head>
        <body>
        </body>
    </html>
```

Output: M

2) concat()

- concat() joins two or more strings:
- The concat() method can be used instead of the plus operator.

Example:

```
<script>
    let str1 = "Learn";
    let str2 = "Javascript";
    let str3 = str1.concat(" ", str2);
    document.write(str3);
</script>
```

Output: Learn Javascript

3) indexOf()

- The indexOf() method returns the index (position) of the first occurrence of a string in a string, or it returns -1 if the string is not found:
- JavaScript counts positions from zero.

Example:

```
<script>
    var str = "Computers Mobiles";
    var index = str.indexOf("s");
    document.write(index);
</script>
```

Output: 8

4) lastIndexOf()

- The lastIndexOf() method returns the index of the last occurrence of a specified text in a string:
- lastIndexOf() return -1 if the text is not found

Example:

```
<script>
    var str = "Computers Mobiles";
    var index = str.lastIndexOf("s");
    document.write(index);
</script>
```

Output: 16

5) replace():

- The replace() method replaces a specified value with another value in a string.
- The replace() method returns a new string.
- The replace() method replaces only the first match

Example:

```
<script>
    var str = "Computers Mobiles";
    var newstr = str.replace("Mobiles", "Tablets");
    document.write(newstr);
</script>
```

Output: Computers Tablets

6) search()

- The search() method searches a string for a string (or a regular expression) and returns the position of the match.

Example:

```
<script>
    var str = "Computers Mobiles";
    var str2 = str.search("o");
    document.write(str2);
</script>
```

Output: 1

7) substr()

- substr() extracts a part of a string and returns the extracted part in a new string.
- Beginning at a specified start position, through thespecified number of character(second argument is for length).

Example:

```
<script>
    var str = "Javascript";
    var str2 = str.substr(4,6);
    document.write(str2);
</script>
```

Output: script

8) substring()

- The substring() method extract a part of a string and returns the extracted parts in a new string:
- Extract the characters from a string, between two specified indices (till second option's character).

Example:

```
<script>
    var str = "Javascript";
    var str2 = str.substring(4,6);
    document.write(str2);
</script>
```

Output: sc

9) toLowerCase()

- A string is converted to lower case with toLowerCase():

Example:

```
<script>
    var str = "COMPUter";
    var str2 = str.toLowerCase();
    document.write(str2);
</script>
```

Output: computer

10) toUpperCase()

- A string is converted to upper case with toUpperCase():

Example:

```
<script>
    var str = "computer";
    var str2 = str.toUpperCase();
    document.write(str2);
</script>
```

Output: COMPUTER

2. Maths Functions:

- The JavaScript Math object allows you to perform mathematical tasks on numbers.
- The syntax for any Math property is : Math.property.

1) abs()

- The Math.abs() method returns the absolute value of a number (positive value).

Example:

```
<script>
    var x=-7.25
    document.write(Math.abs(x));
</script>
```

Output: 7.25

2) ceil()

- The Math.ceil() method rounds a number rounded Upwards to the nearest integer.

Example:

```
<script>
```

```
        var x=1.4;
        document.write(Math.ceil(x));
    </script>
```

Output: 2

3) floor()

- The Math.floor() method rounds a number DOWN to the nearest integer.

Example:

```
    <script>
        var x=5.6;
        document.write(Math.floor(x));
    </script>
```

Output: 5

4) pow()

- The Math.pow() method returns the value of x to the power of y (xy).

Example:

```
    <script>
        var x=2;
        var y=4;
        document.write(Math.pow(x,y));
    </script>
```

Output: 16

5) random()

- The Math.random() method returns a random floating point number between 0 and 1.
- **For example:**
- let x = Math.random() * 10;
- Return a random number between 0 (inclusive) and 10 (exclusive):

```
    <script>
        document.write(Math.random());
    </script>
```

Output: 0.7534718835172003

6) round()

- The Math.round() method rounds a number to the nearest integer.
- 2.49 will be rounded down (2), and 2.5 will be rounded up (3).

Example:

```
<script>
    var x=10.5;
    document.write(Math.round(x));
</script>
```

Output: 11

7) max()

- The Math.max() method returns the number with the highest value.

Example:

```
<script>
    var x=Math.max(0, 150, 30, 20, 38, 10.5);
    document.write(x);
</script>
```

Output: 150

8) min()

- The Math.min() method returns the number with the lowest value.

Example:

```
<script>
    var x=Math.min(0, 150, 30, 20, 38, 10.5);
    document.write(x);
</script>
```

Output: 0

9) sqrt()

- The Math.sqrt() method returns the square root of a number.

Example:

```
<script>
    var x=Math.sqrt(25);
    document.write(x);
</script>
```

Output: 5

10) cbrt()

- The Math.cbrt() method returns the cubic root of a number.

Example:

```
<script>
    var x=Math.cbrt(125);
```

```
        document.write(x);
    </script>
```

Output: 5

3. Array Functions:

- JavaScript array methods are built-in functions that allow efficient manipulation of arrays.
- They provide essential functionalities like adding, removing, and transforming elements, as well as searching and sorting array elements.

1) length()

- The length property returns the length (size) of an array.

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple",
    "Grapes"];
    let size = fruits.length;
    document.write(size);
</script>
```

Output: 4

2) sort()

- The sort() method sorts an array alphabetically:

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple",
    "Grapes"];
    sorting=fruits.sort();
    document.write(sorting);
</script>
```

Output: Apple,Banana,Grapes,Orange

3) reverse()

- The reverse() method reverses the elements in an array:

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple",
    "Grapes"];
    fruits.reverse();
```

```
        document.write(fruits);
    </script>
```

Output: Grapes,Apple,Orange,Banana

4) pop():

- The pop() method removes the last element from an array:

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple",
    "Grapes"];
    fruits.pop();
    document.write(fruits);
</script>
```

Output: Banana,Orange,Apple

5) push()

- The push() method adds a new element to an array (at the end):

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple",
    "Grapes"];
    fruits.push("Kiwi");
    document.write(fruits);
</script>
```

Output: Banana,Orange,Apple,Grapes,Kiwi

6) shift()

- The shift() method removes the first array element and "shifts" all other elements to a lower index.
- The shift() method removes the first element of an array (and "shifts" the other elements to the left).

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple",
    "Grapes"];
    fruits.shift();
    document.write(fruits);
</script>
```

Output: **Orange,Apple,Grapes**

7) unshift()

- The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple", "Grapes"];
    fruits.unshift("Lemon");
    document.write(fruits);
</script>
```

Output: **Lemon,Banana,Orange,Apple,Grapes**

8) join():

- The join() method joins all array elements into a string and it returns string, but in addition you can specify the separator.

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple", "Grapes"];
    mix=fruits.join(" -- ");
    document.write(mix);
</script>
```

Output: **Banana -- Orange -- Apple – Grapes**

9) at():

- Get the third element of fruits using at().
- The at() method returns an indexed element from an array.
- The at() method returns the same as [].

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple", "Grapes"];
    val=fruits.at(2);
    document.write(val);
</script>
```

Output: **Apple**

10) indexOf()

- The indexOf() method searches an array for an element value and returns its position.
- The first item has position 0, the second item has position 1, and so on.

Example:

```
<script>
    const fruits = ["Banana", "Orange", "Apple", "Grapes"];
    var position = fruits.indexOf("Apple");
    document.write(position);
</script>
```

Output: **2**

4. Date Functions:

- It is used for managing date and time in applications, providing methods for date arithmetic, formatting, and manipulation.
- The time value in a JavaScript Date object is measured in milliseconds since January 1, 1970, 00:00:00.

1) new Date()

- In JavaScript, date objects are created with new Date().
- new Date() returns a date object with the current date and time.

Example:

```
<script>
    const d = new Date();
    document.write(d);
</script>
```

Output: **Thu Sep 26 2024 19:13:54 GMT+0530 (India Standard Time)**

2) getDate()

- The getDate() method returns the day of a date as a number (1-31):

Example:

```
<script>
    const d = new Date("2021-10-25");
    n=d.getDate();
    document.write(n);
</script>
```

Output: **25**

3) getDay()

- The getDay() method returns the weekday of a date as a number (0-6).
- In JavaScript, the first day of the week (day 0) is Sunday.

Example:

```
<script>
    const d = new Date("2021-10-25");
    n=d.getDay();
    document.write(n);
</script>
```

Output: 1

4) getMonth()

- The getMonth() method returns the month of a date as a number (0-11).
- In JavaScript, January is month number 0, February is number 1, ...
- Finally, December is month number 11.

Example:

```
<script>
    const d = new Date("2021-11-25");
    n=d.getMonth();
    document.write(n);
</script>
```

Output: 10

5) getFullYear()

- The getFullYear() method returns the year of a date as a four digit number.

Example:

```
<script>
    const d = new Date("2021-11-25");
    n=d.getFullYear();
    document.write(n);
</script>
```

Output: 2021

6) getHours()

- The getHours() method returns the hours of a date as a number (0-23).

Examples:

```
<script>
```

```
        const d = new Date();
        n=d.getHours();
        document.write(n);
    </script>
```

Output: 19

7) getMinutes()

- The getMinutes() method returns the minutes of a date as a number (0-59).

Example:

```
<script>
        const d = new Date();
        n=d.getMinutes();
        document.write(n);
</script>
```

Output: 29

8) getSeconds()

- The getSeconds() method returns the seconds of a date as a number (0-59).

Example:

```
<script>
        const d = new Date();
        n=d.getSeconds();
        document.write(n);
</script>
```

Output: 37

9) getMilliseconds()

- The getMilliseconds() method returns the milliseconds of a date as a number (0-999).

Example:

```
<script>
        const d = new Date();
        n=d.getMilliseconds();
        document.write(n);
</script>
```

Output: 433

10) setDate()

- Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object.
- The setDate() method sets the day of a date object (1-31):

Example:

```
<script>
    const d = new Date();
    d.setDate(15);
    document.write(d);
</script>
```

Output: Sun Sep 15 2024 19:40:26 GMT+0530 (India Standard Time)

11) setMonth()

- The setMonth() method sets the month of a date object (0-11):

Example:

```
<script>
    const d = new Date();
    d.setMonth(11);
    document.write(d);
</script>
```

Output: Thu Dec 26 2024 19:45:08 GMT+0530 (India Standard Time)

12) setFullYear()

- The setFullYear() method sets the year of a date object. In this example to 2020:

Example:

```
<script>
    const d = new Date();
    d.setFullYear(2020);
    document.write(d);
</script>
```

Output: Sat Sep 26 2020 19:46:41 GMT+0530 (India Standard Time)

13) setHours()

- The setHours() method sets the hours of a date object (0-23).

Example:

```
<script>
    const d = new Date();
    d.setHours(22);
    document.write(d);
```

```
</script>
```

Output: **Thu Sep 26 2024 22:49:37 GMT+0530 (India Standard Time)**

14) setMinutes()

- The setMinutes() method sets the minutes of a date object (0-59).

Example:

```
<script>
    const d = new Date();
    d.setMinutes(40);
    document.write(d);
</script>
```

Output: **Thu Sep 26 2024 19:40:52 GMT+0530 (India Standard Time)**

15) setSeconds()

- The setSeconds() method sets the seconds of a date object (0-59).

Example:

```
<script>
    const d = new Date();
    d.setSeconds(50);
    document.write(d);
</script>
```

Output: **Thu Sep 26 2024 19:52:50 GMT+0530 (India Standard Time)**

❖ Events:

(onclick, ondblclick, onmouseover, onmouseout, onkeypress, onkeyup, onfocus, onblur, onload, onchange, onsubmit, onreset)

- HTML events are "things" that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can "react" on these events.
- An HTML event can be something the browser does, or something a user does.
- JavaScript Events are actions or occurrences that happen in the browser.
- **Here are some examples of HTML events:**
- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

- Common events include mouse clicks, keyboard presses, page loads, and form submissions. Event handlers are JavaScript functions that respond to these events.

- When events happen, you may want to do something.
- JavaScript lets you execute code when events are detected.
- HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

- With single quotes:
  <element event='some JavaScript'>
- With double quotes:
  <element event="some JavaScript">

1. Onclick:

- Triggered when an element is clicked
- This is the most frequently used event type which occurs when a user clicks the left button of the mouse.
- You can put your validation, warning etc. against this event type.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javacript Onclick Event</title>
            <script>
                function displayDate()
                {
                  document.write(Date());
                }
            </script>
        </head>
        <body>
            <button type="button" onclick="displayDate();"
            id="button_date">Today's Date</button>
        </body>
    </html>
```

2. Ondblclick:

- The ondblclick event occurs when the user double-clicks on an HTML element.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
        <title>Javacript OnDblclick Event</title>
        <script>
            function message() {
              document.write("This is a paragraph element");
            }
        </script>
        </head>
        <body>
            <p ondblclick="message()">Double-click this paragraph to trigger a function.</p>
        </body>
    </html>
```

3. Onmouseover:

- Fired when the mouse pointer moves over an element.
- The onmouseover event occurs when the mouse pointer enters an element.
- The onmouseover event is often used together with the onmouseout event, which occurs when the mouse pointer leaves the element.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
        <title>Javacript onmouseover Event</title>
        <script>
            function tooltip() {
              alert("This is a paragraph element");
            }
        </script>
        </head>
        <body>
            <p onmouseover="tooltip()">The event is triggered when the mouse pointer is moved out of the paragraph.</p>
```

```
        </body>
    </html>
```

4. Onmouseout:

- Occurs when the mouse pointer leaves an element.
- The onmouseout event occurs when the mouse pointer moves out of an element.
- The onmouseout event is often used together with the onmouseover event, which occurs when the pointer is moved over an element.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javacript Onmouseout Event</title>
            <script>
                function out() {
                  document.write("onmouse out event is occurred");
                }
            </script>
        </head>
        <body>
            <p onmouseout="out()" >The event is triggered when the mouse pointer is moved out of the paragraph.</p>
        </body>
    </html>
```

5. Onkeypress:

- The onkeypress event occurs when the user presses a key on the keyboard.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javacript Onkeypress Event</title>
            <script>
                function fun() {
                  alert("You pressed a key inside the input field");
```

```
                    }
                </script>
            </head>
            <body>
                    <input type="text" onkeypress="fun()">
            </body>
        </html>
```

6. Onkeyup:

- Fired when a key is released.
- The onkeyup event occurs when the user releases a key on the keyboard.

Example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Javacript Onkeyup Event</title>
        <script>
            function fun() {

              document.write("You pressed a key up");
            }
        </script>
    </head>
    <body>
        <p contentEditable=true onkeyup="fun()">Paragraph</p>
    </body>
</html>
```

7. Onfocus:

- Occurs when an element gets focus.
- The onfocus event occurs when an element gets focus.
- The onfocus event is often used on input fields.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javacript Onfocus Event</title>
        </head>
        <body>
```

```
                    Enter your First name: <input type="text" id="name1"
                    onfocus="name1.style.background='yellow'"
                    value="123">
                    Enter your Last Name: <input type="text" id="lastname"
                    onfocus="name1.style.background='blue'" value="123">
                </body>
            </html>
```

8. Onblur:

- Fired when an element loses focus.
- The onblur event occurs when an HTML element loses focus.
- The onblur event is often used on input fields.
- The onblur event is often used with form validation (when the user leaves a form field).

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javacript Onblur Event</title>
        </head>
        <body>
            Enter your First Name: <input type="text" id="name1"
            onblur="name1.style.color='white'" value="123"><br>
            Enter your Last Name: <input type="text" id="lastname"
            onfocus="name1.style.background='blue'" value="123">
        </body>
</html>
```

9. Onload:

- Occurs when a page has finished loading.
- The onload event occurs when an object has been loaded.
- onload is most often used within the <body> element to execute a script once a web page has completely loaded all content (including images, script files, CSS files, etc.).
- The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

Example:

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <title>Javacript Onload Event</title>
    </head>
    <body onload="load()">
        <script>
            function load() {
              alert("Page is loaded");
            }
        </script>
    </body>
</html>
```

10.Onchange:

- The onchange event occurs when the value of an HTML element(input element) is changed.
- This event occurs when the element loses focus, after the content has been changed.
- onchange event also works on <select> elements.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javacript OnChange Event</title>
            <script>
                function fun(val) {
                  confirm("Do You Want to Save the Changes " +
                val);
                }
            </script>
        </head>
        <body>
            <p>Modify the text in the input field, then click outside
            the field to fire the onchange event.</p>
            Enter some text: <input type="text" id="name1"
            name="text" value="Welcome"
            onchange="fun(name1.value)">
        </body>
```

```
</html>
```

11.Onsubmit:

- The onsubmit event occurs when a form is submitted.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javacript OnSubmit Event</title>
            <script>
                function fun_submit() {
                  alert("The form was submitted");
                }
            </script>
        </head>
        <body>
            <p>When you submit the form, a function is triggered
            which alerts some text.</p>
            <form action="" onsubmit="fun_submit()">
                Enter name: <input type="text" name="fname">
                 <input type="submit" value="Submit">
            </form>
        </body>
    </html>
```

12.Onreset:

- The onreset event occurs when a form is reset.

Example:

```
<!DOCTYPE html>
    <html>
        <head>
            <title>Javacript OnSubmit_Reset Event</title>
            <script>
                function fun_submit() {
                  alert("The form was submitted");
                }
                function fun_reset() {
                  alert("The form was reset");
                }
```

```
        </script>
    </head>
    <body>
        <p>When you submit the form, a function is triggered
        which alerts some text.</p>
        <p>When you reset the form, a function is triggered
        which alerts some text.</p>
        <form action="" onsubmit="fun_submit()"
        onreset="fun_reset()">
            Enter name: <input type="text" name="fname">
            <input type="submit" value="Submit">
            <input type="reset" value="Reset">
        </form>
    </body>
</html>
```

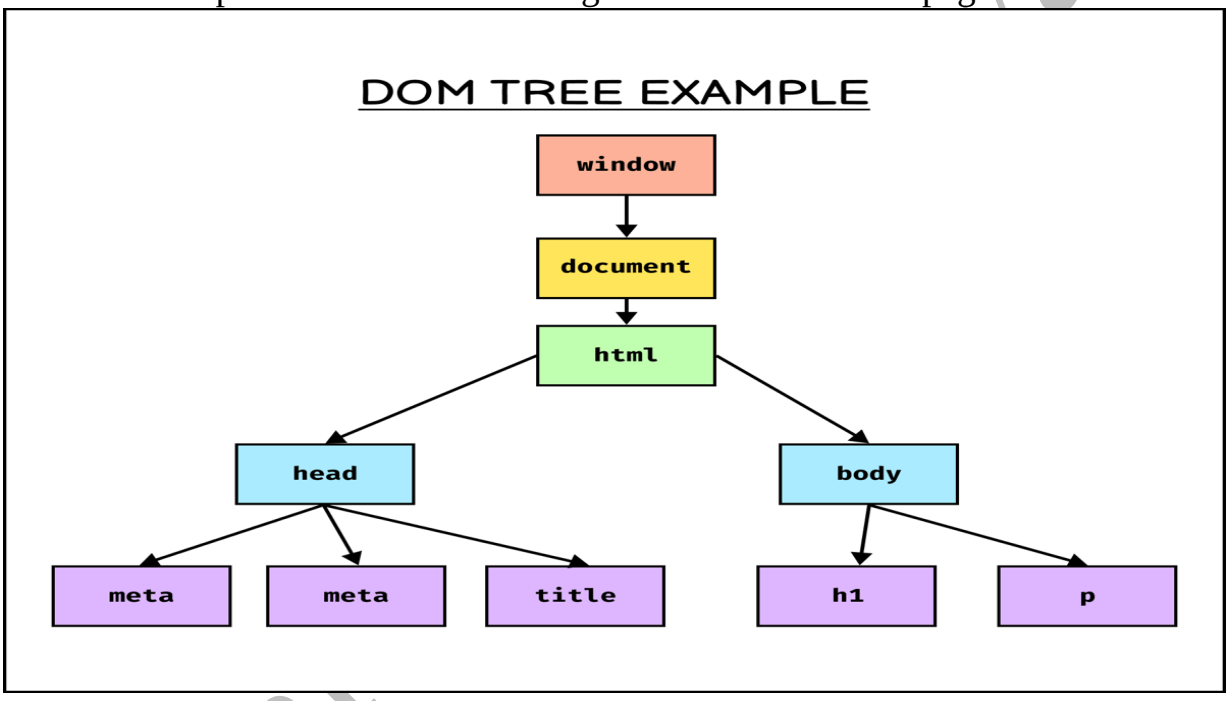
❖ DOM & History Object:

❖ 1. 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:
- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.
- The Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
- The DOM represents a document as a tree. The tree is made up of parent child relationship.
- A parent can have one or many children nodes.
- The DOM is a way to manipulate the structure and style of an HTML page. It represents the internal of the page as the browser sees it, and allows the developers to alter it with java script.
- The objects are organized in hierarchy.
- This hierarchical structure applies to the organization of objects in a web document.
- **Window Object:** Top of the hierarchy. It is the out most element of the object hierarchy.

- **Document Object:** Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.

- With the object model, JavaScript gets all the power it needs to create dynamic HTML:
- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page.
- JavaScript can react to all existing HTML events in the page.



DOM Methods for Finding HTML Elements:

No.	Method	Description
1.	document.getElementById(id)	Find an element by element id
2.	document.getElementsByTagName(name)	Find elements by tag name
3.	document.getElementsByClassName(name)	Find elements by class name
4.	document.write(text)	Write into the HTML output stream
5.	document.getElementsByName(name)	Find elements by name

1. **getElementById(id):**

- Get the element with the specified id

Example:

```
<!DOCTYPE html>
    <html>
        <body>
            <h1 id="demo">The Document Object</h1>
            <h2>The getElementById() Method</h2>
            <script>
            const heading = document.getElementById("demo");
            heading.style.color = "red";
            </script>
        </body>
    </html>
```

2. **getElementsByTagName(name):**

- Get all elements with the tag name "li" (use index value for specific number of tag).

Example:

```
<!DOCTYPE html>
<html>
    <body>
        <p>An unordered list:</p>
        <ul>
            <li>Coffee</li>
            <li>Tea</li>
            <li>Milk</li>
        </ul>
        <script>
            var li_tag = document.getElementsByTagName("li");
            li_tag[2].style.backgroundColor="yellow";
        </script>
    </body>
</html>
```

3. **getElementsByClassName(name):**

- Get all elements with class name.

Example:

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

```
        <head>
            <style>
                .example{border: 1px solid black;
                color:blue;
                background:yellow;
                }
            </style>
        </head>
        <body>
        <p class="example">Element1</p>
        <p class="example">Element2</p>
        <script>
            var para = document.getElementsByClassName("example");
            para[1].style.backgroundColor="green";
        </script>
    </body>
</html>
```

4. **write(text):**

- Write some text directly to the HTML output.

Example:

```
<!DOCTYPE html>
    <html>
        <body>
            <script>
                document.write("Javascript");
            </script>
        </body>
    </html>
```

5. **getElementsByName(name):**

- Get all elements with the name name.

Example:

```
<!DOCTYPE html>
<html>
    <body>
        <p>First Name: <input name="fname" type="text"
        value="Michael"></p>
```

```
        <p>First Name: <input name="fname" type="text"
        value="Doug"></p>
        <script>
            let elements = document.getElementsByName("fname");
            elements[1].value="john";
        </script>
    </body>
</html>
```

❖ History Object: (The Window History Object):

- The history object contains the URLs visited by the user (in the browser window).
- The history object is a property of the window object.
- The history object is accessed with window.history or just history.
- The JavaScript History object contains the browser's history.
- The window part can be removed from window.history just using the history object alone works fine.
- The JS history object contains an array of URLs visited by the user.
- By using the history object, you can load previous, forward, or any particular page using various methods.

History Object Properties and Methods:

No.	Property/Method	Description
1.	back()	Loads the previous URL (page) in the history list
2.	forward()	Loads the next URL (page) in the history list.
3.	go()	Loads a specific URL (page) from the history list
4.	length	Returns the number of URLs (pages) in the history list

1. Back():

- It loads the previous page. Provides the same effect as clicking forward in the browser.
- The history.back() method loads the previous URL (page) in the history list.
- The history.back() method only works if a previous page exists.

Example:

```
<button onclick="history.back()">Go Back</button>
```

2. Forward():

- The history.forward() method loads the next URL (page) in the history list.
- The history.forward() method only works if a next page exists.
- It loads the next page. Provides the same effect as clicking back in the browser.

Example:

```
<button onclick="history.forward()">Go Forward</button>
```

3. Go():

- The history.go() method loads a URL (page) from the history list.
- The history.go() method only works if the page exist in the history list.
- It loads the given page number in the browser. history.go(distance) function provides the same effect as pressing the back or forward button in your browser and specifying the page exactly that you want to load.

Example:

```
<button onclick="history.go(-2)">Go Back 2 Pages</button>
```

4. Length():

- It returns the length of the history URLs visited by the user in that session.
- The length property returns the number of URLs in the history list of the current browser window.
- The property returns at least 1, because the list includes the current page.
- This property is useful to find out how many pages the user has visited in the current browsing session.

Example:

```
<script>
    let length = history.length;
    document.write(length);
</script>
```

❖ The Window Navigator Object

- The navigator object contains information about the browser.
- The navigator object is a property of the window object.
- The navigator object is accessed with:
- window.navigator or just navigator:

Properties:

1) appName

- The appName property returns the browser name.
- The appName property is read-only.

- **Example:**

```
<script>
document.write(navigator.appName);
</script>
```

2) appVersion

- The appVersion property returns the browser version.
- The appVersion property is read-only.
- **Example:**

```
document.write(navigator.appVersion);
```

3) cookieEnabled

- The cookieEnabled property returns true if cookies are enabled in the browser
- **Example:**

```
document.write(navigator.cookieEnabled);
```

4) onLine

- The onLine property returns true if the browser is online, otherwise false.
- The onLine property is read-only.
- **Example:**

```
document.write(navigator.onLine);
```

5) platform

- The platform property returns for which platform the browser is compiled.
- The platform property is read-only.
- **Example:**

```
document.write(navigator.platform);
```

Method:
javaEnabled()

- The javaEnabled() method returns a Boolean value that specifies whether the browser has Java enabled.
- **Example:**

```
document.write(navigator.javaEnabled());
```

❖ Form Validation & E-mail Validation:

❖ Form Validation:

- HTML form validation can be done by JavaScript.
- Data validation is the process of ensuring that user input is clean, correct, and useful.

- It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.
- JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation.
- **Typical validation tasks are:**
- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?
- Most often, the purpose of data validation is to ensure correct user input.
- Validation can be defined by many different methods, and deployed in many different ways.
- If a form field (fname) is empty, the function alerts a message, and returns false, to prevent the form from being submitted.
- JavaScript is often used to validate numeric input.
- HTML form validation can be performed **automatically** by the browser:
- If a form field (fname) is empty, the **required** attribute prevents this form from being submitted.
- **Data Format Validation:**
- Secondly, the data that is entered must be checked for correct form and value.
- Your code must include appropriate logic to test correctness of data.
- **Error Handling:**
- If any of the checks fail, an alert message is shown to the user using window.alert, telling them what's wrong.
- **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.

isNaN() Function:

- The JavaScript isNaN() Function is used to check whether a given value is an illegal number or not. It returns true if the value is a NaN else returns false.
- In JavaScript NaN is short for "Not-a-Number".
- The isNaN() method returns true if a value is NaN.
- **Parameter Values:** This method accepts a single parameter as mentioned above and described below:

- **value:** It is a required value passed in the isNaN() function.
- **Return Value:** It returns a Boolean value for example, returns true if the value is NaN else returns false.

  Syntax:

  isNaN(value)

- **Form.html**

```
<!Doctype html>
<html>
    <head>
        <title>Form Validation Example </title>
        <script src="script.js"></script>
    </head>
    <body>
        <h1>REGISTRATION FORM</h1>
        <form name="Regform" onsubmit="return validateForm()"
        action="page2.html" method="get">
        <p> <label for="name">Name:</label>
        <input type="text" id="name" name="name" /> </p>
        <p><label for="number">Number:</label>
        <input type="text" id="number" name="number" /> </p>
        <p><label for="password">Password:</label>
        <input type="password" id="password"  name="Password" />
        </p>
        <p><label for="confirm_password">Confirm
        Password</label>
        <input type="password" id="confirm_password"
        name="confirm_password"> </p>
        <p><label for="address">Address</label>
        <textarea rows="2" cols="25" id="address"
        name="address"></textarea></p>
        <p><label for="emailid">Email-Id</label>
        <input type="text" id="emailid" name="emailid"/></p>
        <p>
        <input type="submit" value="Send" name="Submit" />
        <input type="reset" value="Reset" name="Reset" />
        </p>
    </form>
```

```
    </body>
</html>
```

- **Script.js**

```
function validateForm()
{
        const name = document.getElementById("name").value;
        const number = document.getElementById("number").value;
        const password = document.getElementById("password").value;
        const confirm_password =
        document.getElementById("confirm_password").value;
        const address = document.getElementById("address").value;
        const email = document.getElementById("emailid").value;
        email_pattern=
        /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
        //for name
        if(name=="" || name.length<4 || isNaN(name)==false)
        {
                alert("enter atleast 4 letter alphabetic name.");
                return false;
        }
        //for number
        if(isNaN(number)==true || number.length!=10)
        {
                alert("enter 10 digit number");
                return false;
        }
        //for password
        if(password=="" || password.length < 8)
        {
                alert("Password must be of 8 letters");
                return false;
        }
        //for confirm password
        if(confirm_password!==password)
        {
                alert("Enter A Same Password");
                return false;
```

```
        }
        //for address
        if(address=="")
        {
                alert("Enter Address");
                return false;
        }
        //for email
        if(email_pattern.test(email)==false)
        {
                alert("enter correct email id.");
                return false;
        }
}
```

❖ E-mail Validation:

- Validating an email address is a common task in web development.
- It involves ensuring that an email address entered by a user conforms to a valid format.
- Ensuring that a user inputs a valid email address can prevent errors, improve data quality, and enhance user experience.

Example:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Email Validation Example</title>
        <script>
        function validate()
        {
                const email = document.getElementById("emailid").value;
                var email_pattern=/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-
                zA-Z]{2,}$/;
                if(email=="" || email_pattern.test(email)==false)
                {
                        alert("enter correct email id.");
                        return false;
                }
```

```
                else
                {
                        alert("email is correct");
                        return true;
                }
        }
        </script>
    </head>
    <body>
        Email-Id<input type="text" id="emailid" name="emailid"/>
        <input type="submit" value="submit" onclick="validate()"/>
    </body>
</html>
```