JAVASCRIPT

JavaScript is client-side scripting language. JavaScript is used mainly for enhancing the interaction of a user with the webpage i.e. webpages are made more lively and interactive. JavaScript is also being used widely in game development and Mobile application development.

It is interpreted programming language. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

JavaScript was developed by Brendan Eich in 1995, which appeared in Netscape, a popular browser of that time.

The language was initially called LiveScript and was later renamed JavaScript.

Executing JavaScript code

Being a scripting language, JavaScript cannot run on its own. In fact, the browser is responsible for running JavaScript code. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it is up to the browser to execute it. The main advantage of JavaScript is that all modern web browsers support JavaScript.

Applications of Javascript Programming

- Client side validation - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validting those inputs at front-end itself.
- Manipulating HTML Pages - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.
- User Notifications - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
- Back-end Data Loading - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
- Presentations - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
- Server Applications - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows −

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

JavaScript in <head>...</head> section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows −

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function sayHello() {
          alert("Hello World")
        }
      //-->
    </script>
  </head>

  <body>
    <input type = "button" onclick = "sayHello()" value = "Say Hello" />
  </body>
</html>
```

JavaScript in <body>...</body> section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Example :

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

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

    <p>This is web page body </p>
  </body>
</html>
```

This code will produce the following results −

JavaScript in <body> and <head> Sections

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function sayHello() {
          alert("Hello World")
        }
      //-->
    </script>
  </head>

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

    <input type = "button" onclick = "sayHello()" value = "Say Hello" />
  </body>
</html>
```

JavaScript in External File

Example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute.

```
<html>
  <head>
    <script type = "text/javascript" src = "filename.js" ></script>
  </head>

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

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in filename.js file and then you can use sayHello function in your HTML file after including the filename.js file.

```
function sayHello() {
  alert("Hello World")
}
```

Tools You Need

To start with, you need a text editor to write your code and a browser to display the web pages you develop.

You should place all your JavaScript code within <script> tags (<script> and </script>) if you are keeping your JavaScript code within the HTML document itself. This helps your browser distinguish your JavaScript code from the rest of the code. As there are other client-side scripting languages (Example: VBScript), it is highly recommended that you specify the scripting language you use. You have to use the type attribute within the <script> tag and set its value to text/javascript like this:

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

<html>

<head>

        <title>My First JavaScript code!!!</title>

        <script type="text/javascript">

                alert("Hello World!");

        </script>

</head>

<body>

</body>

</html>
```

Note: type="text/javascript" is not necessary in HTML5.

JavaScript Datatypes

JavaScript allows you to work with three primitive data types −

- Numbers, eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object.

Note − JavaScript does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values.

JavaScript Variables

Variables are declared with the var keyword as follows.

```
<script type = "text/javascript">
  <!--
    var money;
    var name;
  //-->
</script>
```

You can also declare multiple variables with the same var keyword as follows −

```
<script type = "text/javascript">
  <!--
    var money, name;
  //-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type = "text/javascript">
  <!--
    var name = "Ali";
    var money;
    money = 2000.50;
  //-->
</script>
```

Note − Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- Global Variables − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- Local Variables − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<html>
  <body onload = checkscope();>
    <script type = "text/javascript">
      <!--
        var myVar = "global";      // Declare a global variable
        function checkscope( ) {
          var myVar = "local";    // Declare a local variable
          document.write(myVar);
        }
      //-->
    </script>
  </body>
</html>
```

This produces the following result −

```
local
```

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one.
- JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

JavaScript Reserved Words

Reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

Abstract	Else	instanceof	switch

Boolean	Enum	int	synchronized
Break	Export	interface	this
Byte	Extends	long	throw
Case	False	native	throws
Catch	Final	new	transient
Char	Finally	null	true
Class	Float	package	try
Const	For	private	typeof
Continue	Function	protected	var
Debugger	Goto	public	void
Default	If	return	volatile
Delete	implements	short	while
Do	Import	static	with
double	In	super	

Operators in JAVASCRIPT

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

Assume variable A holds 10 and variable B holds 20, then −

Operator & Description
+ (Addition) Adds two operands **Ex:** A + B will give 30
- (Subtraction) Subtracts the second operand from the first **Ex:** A - B will give -10
*** (Multiplication)** Multiply both operands **Ex:** A * B will give 200
/ (Division) Divide the numerator by the denominator **Ex:** B / A will give 2
% (Modulus) Outputs the remainder of an integer division **Ex:** B % A will give 0
++ (Increment) Increases an integer value by one **Ex:** A++ will give 11

-- (Decrement)

Decreases an integer value by one

Ex: A-- will give 9

Note − Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example

```
<html>
   <body>

      <script type = "text/javascript">
         <!--
            var a = 33;
            var b = 10;
            var c = "Test";
            var linebreak = "<br />";

            document.write("a + b = ");
            result = a + b;
            document.write(result);
            document.write(linebreak);

            document.write("a - b = ");
            result = a - b;
            document.write(result);
            document.write(linebreak);

            document.write("a / b = ");
            result = a / b;
            document.write(result);
            document.write(linebreak);

            document.write("a % b = ");
            result = a % b;
            document.write(result);
            document.write(linebreak);

            document.write("a + b + c = ");
            result = a + b + c;
            document.write(result);
            document.write(linebreak);

            a = ++a;
            document.write("++a = ");
            result = ++a;
            document.write(result);
            document.write(linebreak);
            b = --b;
            document.write("--b = ");
            result = --b;
            document.write(result);
            document.write(linebreak);
         //-->
      </script>

      Set the variables to different values and then try...
   </body>
</html>
```

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...
```

Comparison Operators

Assume variable A holds 10 and variable B holds 20, then −

Operator & Description
= = (Equal) Checks if the value of two operands are equal or not, if yes, then the condition becomes true. **Ex:** (A == B) is not true.
!= (Not Equal) Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. **Ex:** (A != B) is true.
> (Greater than) Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A > B) is not true.
< (Less than)

Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. **Ex:** (A < B) is true.
>= (Greater than or Equal to) Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A >= B) is not true.
<= (Less than or Equal to) Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. **Ex:** (A <= B) is true.

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 10;
            var b = 20;
            var linebreak = "<br />";

            document.write("(a == b) => ");
            result = (a == b);
            document.write(result);
            document.write(linebreak);

            document.write("(a < b) => ");
            result = (a < b);
            document.write(result);
            document.write(linebreak);

            document.write("(a > b) => ");
            result = (a > b);
            document.write(result);
            document.write(linebreak);

            document.write("(a != b) => ");
            result = (a != b);
            document.write(result);
            document.write(linebreak);

            document.write("(a >= b) => ");
            result = (a >= b);
            document.write(result);
            document.write(linebreak);

            document.write("(a <= b) => ");
            result = (a <= b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      Set the variables to different values and different
operators and then try...
   </body>
</html>
```

Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true
Set the variables to different values and different operators and
then try...
```

Logical Operators

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	**&& (Logical AND)** If both the operands are non-zero, then the condition becomes true. **Ex:** (A && B) is true.
2	**|| (Logical OR)** If any of the two operands are non-zero, then the condition becomes true. **Ex:** (A || B) is true.
3	**! (Logical NOT)** Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. **Ex:** ! (A && B) is false.

Example

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = true;
            var b = false;
            var linebreak = "<br />";

            document.write("(a && b) => ");
            result = (a && b);
            document.write(result);
            document.write(linebreak);

            document.write("(a || b) => ");
            result = (a || b);
            document.write(result);
            document.write(linebreak);

            document.write("!(a && b) => ");
            result = (!(a && b));
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Output

```
(a && b) => false
(a || b) => true
!(a && b) => true
Set the variables to different values and different operators and
then try...
```

Bitwise Operators

Assume variable A holds 2 and variable B holds 3, then −

Operator & Description
& (Bitwise AND) It performs a Boolean AND operation on each bit of its integer arguments. **Ex:** (A & B) is 2.

| (BitWise OR)

It performs a Boolean OR operation on each bit of its integer arguments.

Ex: (A | B) is 3.

^ (Bitwise XOR)

It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.

Ex: (A ^ B) is 1.

~ (Bitwise Not)

It is a unary operator and operates by reversing all the bits in the operand.

Ex: (~B) is -4.

<< (Left Shift)

It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.

Ex: (A << 1) is 4.

>> (Right Shift)

Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.

Ex: (A >> 1) is 1.

>>> (Right shift with Zero)

This operator is just like the >> operator, except that the bits shifted in on the left are always zero.

Ex: (A >>> 1) is 1.

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 2; // Bit presentation 10
            var b = 3; // Bit presentation 11
            var linebreak = "<br />";
            document.write("(a & b) => ");
            result = (a & b);
            document.write(result);
            document.write(linebreak);

            document.write("(a | b) => ");
            result = (a | b);
            document.write(result);
            document.write(linebreak);

            document.write("(a ^ b) => ");
            result = (a ^ b);
            document.write(result);
            document.write(linebreak);

            document.write("(~b) => ");
            result = (~b);
            document.write(result);
            document.write(linebreak);

            document.write("(a << b) => ");
            result = (a << b);
            document.write(result);
            document.write(linebreak);

            document.write("(a >> b) => ");
            result = (a >> b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

```
(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0
Set the variables to different values and different operators and
then try...
```

Assignment Operators

JavaScript supports the following assignment operators −

Sr.No.	**Operator & Description**

1	**= (Simple Assignment)** Assigns values from the right side operand to the left side operand **Ex:** C = A + B will assign the value of A + B into C
2	**+= (Add and Assignment)** It adds the right operand to the left operand and assigns the result to the left operand. **Ex:** C += A is equivalent to C = C + A
3	**−= (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **Ex:** C -= A is equivalent to C = C - A
4	***= (Multiply and Assignment)** It multiplies the right operand with the left operand and assigns the result to the left operand. **Ex:** C *= A is equivalent to C = C * A
5	**/= (Divide and Assignment)** It divides the left operand with the right operand and assigns the result to the left operand. **Ex:** C /= A is equivalent to C = C / A
6	**%= (Modules and Assignment)** It takes modulus using two operands and assigns the result to the left operand. **Ex:** C %= A is equivalent to C = C % A

Note − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 33;
            var b = 10;
            var linebreak = "<br />";

            document.write("Value of a => (a = b) => ");
            result = (a = b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a += b) => ");
            result = (a += b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a -= b) => ");
            result = (a -= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a *= b) => ");
            result = (a *= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a /= b) => ");
            result = (a /= b);
            document.write(result);
            document.write(linebreak);

            document.write("Value of a => (a %= b) => ");
            result = (a %= b);
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Output

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
Set the variables to different values and different operators and
then try...
```

Other Operator

Conditional operator (? :) and the **typeof operator**.

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Operator and Description
? : (Conditional) If Condition is true? Then value X : Otherwise value Y

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 10;
            var b = 20;
            var linebreak = "<br />";

            document.write ("((a > b) ? 100 : 200) => ");
            result = (a > b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);

            document.write ("((a < b) ? 100 : 200) => ");
            result = (a < b) ? 100 : 200;
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

Output

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100
Set the variables to different values and different operators and
then try...
```

typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var a = 10;
            var b = "String";
            var linebreak = "<br />";

            result = (typeof b == "string" ? "B is String" : "B
is Numeric");
            document.write("Result => ");
            document.write(result);
            document.write(linebreak);

            result = (typeof a == "string" ? "A is String" : "A
is Numeric");
```

```
            document.write("Result => ");
            document.write(result);
            document.write(linebreak);
         //-->
      </script>
      <p>Set the variables to different values and different
operators and then try...</p>
   </body>
</html>
```

<u>Output</u>

```
Result => B is String
Result => A is Numeric
Set the variables to different values and different operators and
then try...
```

Conditional Statements and Decision Making

JavaScript supports the following forms of **if..else** statement −

- if statement
- if...else statement
- if...else if... statement.

if statement

Syntax

The syntax for a basic if statement is as follows −

```
if (expression) {
   Statement(s) to be executed if expression is true
}
```

Example

Try the following example to understand how the **if** statement works.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var age = 20;

            if( age > 18 ) {
               document.write("<b>Qualifies for driving</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Qualifies for driving
Set the variable to different value and then try...
```

if...else statement

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

```
if (expression) {
   Statement(s) to be executed if expression is true
} else {
   Statement(s) to be executed if expression is false
}
```

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var age = 15;

            if( age > 18 ) {
               document.write("<b>Qualifies for driving</b>");
            } else {
               document.write("<b>Does not qualify for
driving</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Does not qualify for driving
Set the variable to different value and then try...
```

if...else if... statement

The **if...else if...** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows −

```
if (expression 1) {
   Statement(s) to be executed if expression 1 is true
} else if (expression 2) {
   Statement(s) to be executed if expression 2 is true
} else if (expression 3) {
   Statement(s) to be executed if expression 3 is true
} else {
   Statement(s) to be executed if no expression is true
}
```

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var book = "maths";
            if( book == "history" ) {
               document.write("<b>History Book</b>");
            } else if( book == "maths" ) {
               document.write("<b>Maths Book</b>");
            } else if( book == "economics" ) {
               document.write("<b>Economics Book</b>");
            } else {
               document.write("<b>Unknown Book</b>");
            }
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
<html>
```

SWITCH STATEMENT

```
switch (expression) {
   case condition 1: statement(s)
   break;

   case condition 2: statement(s)
   break;
   ...

   case condition n: statement(s)
   break;

   default: statement(s)
}
```

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var grade = 'A';
            document.write("Entering switch block<br />");
            switch (grade) {
               case 'A': document.write("Good job<br />");
               break;

               case 'B': document.write("Pretty good<br />");
               break;

               case 'C': document.write("Passed<br />");
               break;

               case 'D': document.write("Not so good<br />");
               break;

               case 'F': document.write("Failed<br />");
               break;

               default:  document.write("Unknown grade<br />")
            }
            document.write("Exiting switch block");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Entering switch block
Good job
Exiting switch block
Set the variable to different value and then try...
```

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var grade = 'A';
            document.write("Entering switch block<br />");
            switch (grade) {
               case 'A': document.write("Good job<br />");
               case 'B': document.write("Pretty good<br />");
               case 'C': document.write("Passed<br />");
               case 'D': document.write("Not so good<br />");
               case 'F': document.write("Failed<br />");
               default: document.write("Unknown grade<br />")
            }
            document.write("Exiting switch block");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
Set the variable to different value and then try...
```

WHILE LOOP

Syntax

```
while (expression) {
   Statement(s) to be executed if expression is true
}
```

Example

```
<html>
   <body>

      <script type = "text/javascript">
         <!--
            var count = 0;
            document.write("Starting Loop ");

            while (count < 10) {
               document.write("Current Count : " + count + "<br
/>");
               count++;
            }

            document.write("Loop stopped!");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
Set the variable to different value and then try...
```

The do...while Loop

Syntax

The syntax for **do-while** loop in JavaScript is as follows −

```
do {
   Statement(s) to be executed;
} while (expression);
```

Note − Don’t miss the semicolon used at the end of the **do...while** loop.

Example

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var count = 0;
            
            document.write("Starting Loop" + "<br />");
            do {
               document.write("Current Count : " + count + "<br
/>");
               count++;
            }
            
            while (count < 5);
            document.write ("Loop stopped!");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then try...
```

FOR....IN

The **for...in** loop is used to loop through an object's properties

Syntax

The syntax of 'for..in' loop is −

```
for (variablename in object) {
   statement or block to execute
}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

Example

Try the following example to implement 'for-in' loop. It prints the web browser's **Navigator** object.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var aProperty;
            document.write("Navigator Object Properties<br /> ");
            for (aProperty in navigator) {
               document.write(aProperty);
               document.write("<br />");
            }
            document.write ("Exiting from the loop!");
         //-->
      </script>
      <p>Set the variable to different object and then try...</p>
   </body>
</html>
```

Output

```
Navigator Object Properties
serviceWorker
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack
onLine
languages
language
userAgent
product
platform
appVersion
appName
appCodeName
hardwareConcurrency
maxTouchPoints
vendorSub
vendor
productSub
cookieEnabled
mimeTypes
plugins
javaEnabled
getStorageUpdates
getGamepads
webkitGetUserMedia
vibrate
getBattery
sendBeacon
registerProtocolHandler
unregisterProtocolHandler
Exiting from the loop!
Set the variable to different object and then try...
```

FUNCTIONS

- Predefined & User Defined Functions

User Defined Functions

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
<script type = "text/javascript">
   <!--
      function functionname(parameter-list) {
         statements
      }
   //-->
</script>
```

Example

```
<script type = "text/javascript">
   <!--
      function sayHello() {
         alert("Hello there");
      }
   //-->
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello() {
            document.write ("Hello there!");
         }
      </script>

   </head>
   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "sayHello()" value =
"Say Hello">
      </form>
      <p>Use different text in write method and then try...</p>
   </body>
</html>
```

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello(name, age) {
            document.write (name + " is " + age + " years old.");
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "sayHello('Zara', 7)"
value = "Say Hello">
      </form>
      <p>Use different parameters inside the function and then
try...</p>
   </body>
</html>
```

The return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

```
<html>
   <head>
      <script type = "text/javascript">
         function concatenate(first, last) {
            var full;
            full = first + last;
            return full;
         }
         function secondFunction() {
            var result;
            result = concatenate('Zara', 'Ali');
            document.write (result );
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the function</p>
      <form>
         <input type = "button" onclick = "secondFunction()"
value = "Call Function">
      </form>
      <p>Use different parameters inside the function and then
try...</p>
  </body>
</html>
```

Event

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

onclick Event Type

Example

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function sayHello() {
               alert("Hello World")
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following button and see result</p>
      <form>
         <input type = "button" onclick = "sayHello()" value = "Say 
Hello" />
      </form>
   </body>
</html>
```

onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

```
<html>
```

```
   <head>
      <script type = "text/javascript">
         <!--
            function validation() {
               all validation goes here
               .........
               return either true or false
            }
         //-->
      </script>
   </head>

   <body>
      <form method = "POST" action = "t.cgi" onsubmit = "return
validate()">
         .......
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function over() {
               document.write ("Mouse Over");
            }
            function out() {
               document.write ("Mouse Out");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Bring your mouse inside the division to see the result:</p>
      <div onmouseover = "over()" onmouseout = "out()">
         <h2> This is inside the division </h2>
      </div>
   </body>
</html>
```

HTML 5 Standard Events

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
onafterprint	script	Triggers after the document is printed
onbeforeonload	script	Triggers before the document loads
onbeforeprint	script	Triggers before the document is printed
onblur	script	Triggers when the window loses focus
oncanplay	script	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
onchange	script	Triggers when an element changes
onclick	script	Triggers on a mouse click
oncontextmenu	script	Triggers when a context menu is triggered
ondblclick	script	Triggers on a mouse double-click
ondrag	script	Triggers when an element is dragged

ondragend	script	Triggers at the end of a drag operation
ondragenter	script	Triggers when an element has been dragged to a valid drop target
ondragleave	script	Triggers when an element is being dragged over a valid drop target
ondragover	script	Triggers at the start of a drag operation
ondragstart	script	Triggers at the start of a drag operation
ondrop	script	Triggers when dragged element is being dropped
ondurationchange	script	Triggers when the length of the media is changed
onemptied	script	Triggers when a media resource element suddenly becomes empty.
onended	script	Triggers when media has reach the end
onerror	script	Triggers when an error occur
onfocus	script	Triggers when the window gets focus
onformchange	script	Triggers when a form changes
onforminput	script	Triggers when a form gets user input
onhaschange	script	Triggers when the document has change
oninput	script	Triggers when an element gets user input
oninvalid	script	Triggers when an element is invalid

onkeydown	script	Triggers when a key is pressed
onkeypress	script	Triggers when a key is pressed and released
onkeyup	script	Triggers when a key is released
onload	script	Triggers when the document loads
onloadeddata	script	Triggers when media data is loaded
onloadedmetadata	script	Triggers when the duration and other media data of a media element is loaded
onloadstart	script	Triggers when the browser starts to load the media data
onmessage	script	Triggers when the message is triggered
onmousedown	script	Triggers when a mouse button is pressed
onmousemove	script	Triggers when the mouse pointer moves
onmouseout	script	Triggers when the mouse pointer moves out of an element
onmouseover	script	Triggers when the mouse pointer moves over an element
onmouseup	script	Triggers when a mouse button is released
onmousewheel	script	Triggers when the mouse wheel is being rotated
onoffline	script	Triggers when the document goes offline
onoine	script	Triggers when the document comes online

ononline	script	Triggers when the document comes online
onpagehide	script	Triggers when the window is hidden
onpageshow	script	Triggers when the window becomes visible
onpause	script	Triggers when media data is paused
onplay	script	Triggers when media data is going to start playing
onplaying	script	Triggers when media data has start playing
onpopstate	script	Triggers when the window's history changes
onprogress	script	Triggers when the browser is fetching the media data
onratechange	script	Triggers when the media data's playing rate has changed
onreadystatechange	script	Triggers when the ready-state changes
onredo	script	Triggers when the document performs a redo
onresize	script	Triggers when the window is resized
onscroll	script	Triggers when an element's scrollbar is being scrolled
onseeked	script	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
onseeking	script	Triggers when a media element's seeking attribute is true, and the seeking has begun

onselect	script	Triggers when an element is selected
onstalled	script	Triggers when there is an error in fetching media data
onstorage	script	Triggers when a document loads
onsubmit	script	Triggers when a form is submitted
onsuspend	script	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	script	Triggers when media changes its playing position
onundo	script	Triggers when a document performs an undo
onunload	script	Triggers when the user leaves the document
onvolumechange	script	Triggers when media changes the volume, also when volume is set to "mute"
onwaiting	script	Triggers when media has stopped playing, but is expected to resume

Alert Dialog Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Example

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function Warn() {
               alert ("This is a warning message!");
               document.write ("This is a warning message!");
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following button to see the result: </p>
      <form>
         <input type = "button" value = "Click Me" onclick =
"Warn();" />
      </form>
   </body>
</html>
```

Confirmation Dialog Box

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: **OK** and **Cancel**.

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.

Example

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function getConfirmation() {
               var retVal = confirm("Do you want to continue ?");
               if( retVal == true ) {
                  document.write ("User wants to continue!");
                  return true;
               } else {
                  document.write ("User does not want to
continue!");
                  return false;
               }
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following button to see the result: </p>
      <form>
         <input type = "button" value = "Click Me" onclick =
"getConfirmation();" />
      </form>
   </body>
</html>
```

Prompt Dialog Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called **prompt()** which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: **OK** and **Cancel**. 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**.

Example

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function getValue() {
               var retVal = prompt("Enter your name : ", "your name
here");
               document.write("You have entered : " + retVal);
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following button to see the result: </p>
      <form>
         <input type = "button" value = "Click Me" onclick =
"getValue();" />
      </form>
   </body>
</html>
```

A button with click me text is displayed as output.

JAVASCRIPT OBJECTS

JavaScript is an Object Oriented Programming (OOP) language. The essential properties supported are:

- **Encapsulation** − the capability to store related information, data or methods, together in an object.
- **Aggregation** − the capability to store one object inside another object.
- **Inheritance** − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** − the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. Attributes define properties of an object.

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is −

```
objectName.objectProperty = propertyValue;
```

For example − The following code gets the document title using the **"title"** property of the **document** object.

```
var str = document.title;
```

Methods are the functionionality supported by an object

Example − **write()** method of document object to write any content on the document.

```
document.write("This is test");
```

User-Defined Objects

The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

The Object() Constructor

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.

Example to create an Object.

```
<html>
  <head>
    <title>User-defined objects</title>
    <script type = "text/javascript">
      var book = new Object();   // Create the object
      book.subject = "Perl";     // Assign properties to the object
      book.author  = "Mohtashim";
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      document.write("Book name is : " + book.subject + "<br>");
      document.write("Book author is : " + book.author + "<br>");
    </script>
  </body>
</html>
```

Output

```
Book name is : Perl
Book author is : Mohtashim
```

Example

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

Live Demo

```
<html>
  <head>
  <title>User-defined objects</title>
    <script type = "text/javascript">
      function book(title, author) {
        this.title = title;
        this.author  = author;
      }
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      var myBook = new book("Perl", "Mohtashim");
      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
    </script>
  </body>
</html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
```

Defining Methods for an Object

Example

```
<html>

  <head>
  <title>User-defined objects</title>
    <script type = "text/javascript">
      // Define a function which will work as a method
      function addPrice(amount) {
        this.price = amount;
      }

      function book(title, author) {
        this.title = title;
        this.author  = author;
        this.addPrice = addPrice;  // Assign that method as property.
      }
    </script>
  </head>

  <body>
    <script type = "text/javascript">
      var myBook = new book("Perl", "Mohtashim");
      myBook.addPrice(100);

      document.write("Book title is : " + myBook.title + "<br>");
      document.write("Book author is : " + myBook.author + "<br>");
      document.write("Book price is : " + myBook.price + "<br>");
    </script>
  </body>
</html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

JavaScript Native Objects

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

ARRAY Object

An array is a collection of variables of the same type. The **Array** object allows storing multiple values in a single variable.

Example array

var fruits = new Array("apple", "orange", "mango");

Another way to create array by simply assigning values as follows −

```
var fruits = [ "apple", "orange", "mango" ];
```

To access and to set values inside an array :

fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element

Array Properties

Properties of the Array object along with their description.

Sr.No.	Property & Description
1	constructor Returns a reference to the array function that created the object.
2	**Index** The property represents the zero-based index of the match in the string
3	**Input** This property is only present in arrays created by regular expression matches.
4	length Reflects the number of elements in an array.
5	prototype The prototype property allows you to add properties and methods to an object.

JavaScript array **constructor** property returns a reference to the array function that created the instance's prototype.

Syntax

array.constructor

Return Value **Returns the function that created this object's instance.**

Example

```
<html>
  <head>
    <title>JavaScript Array constructor Property</title>
  </head>

  <body>
```

```
    <script type = "text/javascript">
      var arr = new Array( 10, 20, 30 );
      document.write("arr.constructor is:" + arr.constructor);
    </script>

  </body>
</html>
```

Output

```
arr.constructor is: function Array() { [native code] }
```

- JavaScript array **length** property returns an unsigned, 32-bit integer that specifies the number of elements in an array.

  Syntax: array.length

  Return Value: Returns the length of the array.

  Example

```
<html>
  <head>
    <title>JavaScript Array length Property</title>
  </head>

  <body>
    <script type = "text/javascript">
      var arr = new Array( 10, 20, 30 );
      document.write("arr.length is : " + arr.length);
    </script>
  </body>
</html>
```

Output

```
arr.length is : 3
```

Array Methods

Sr.No.	Method & Description
1	concat() Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	every() Returns true if every element in this array satisfies the provided testing function.
3	filter()

	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	forEach() Calls a function for each element in the array.
5	indexOf() Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	join() Joins all elements of an array into a string.
7	lastIndexOf() Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	map() Creates a new array with the results of calling a provided function on every element in this array.
9	pop() Removes the last element from an array and returns that element.
10	push() Adds one or more elements to the end of an array and returns the new length of the array.
11	reduce() Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	reduceRight() Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13	reverse() Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	shift() Removes the first element from an array and returns that element.
15	slice() Extracts a section of an array and returns a new array.
16	some() Returns true if at least one element in this array satisfies the provided testing function.
17	toSource()

	Represents the source code of an object
18	sort() Sorts the elements of an array
19	splice() Adds and/or removes elements from an array.
20	toString() Returns a string representing the array and its elements.
21	unshift() Adds one or more elements to the front of an array and returns the new length of the array.

Javascript array **push()** method appends the given element(s) in the last of the array and returns the length of the new array.

Syntax

Its syntax is as follows −

array.push(element1, ..., elementN);

Return Value

Returns the length of the new array.

Example

```
<html>
  <head>
    <title>JavaScript Array push Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var numbers = new Array(1, 4, 9);

      var length = numbers.push(10);
      document.write("new numbers is : " + numbers );

      length = numbers.push(20);
      document.write("<br />new numbers is : " + numbers );
    </script>
  </body>
</html>
```

Output

```
new numbers is : 1,4,9,10
new numbers is : 1,4,9,10,20
```

STRING object

Syntax

var val = new String(string);

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

String Properties

Here is a list of the properties of String object and their description.

Sr.No.	Property & Description
1	constructor Returns a reference to the String function that created the object.
2	length Returns the length of the string.
3	prototype The prototype property allows you to add properties and methods to an object.

String Methods

Sr.No.	Method & Description
1	charAt() Returns the character at the specified index.
2	charCodeAt() Returns a number indicating the Unicode value of the character at the given index.
3	concat() Combines the text of two strings and returns a new string.
4	indexOf() Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	lastIndexOf() Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	localeCompare() Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	match()

	Used to match a regular expression against a string.
8	replace() Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
9	search() Executes the search for a match between a regular expression and a specified string.
10	slice() Extracts a section of a string and returns a new string.
11	split() Splits a String object into an array of strings by separating the string into substrings.
12	substr() Returns the characters in a string beginning at the specified location through the specified number of characters.
13	substring() Returns the characters in a string between two indexes into the string.
14	toLocaleLowerCase() The characters within a string are converted to lower case while respecting the current locale.
15	toLocaleUpperCase() The characters within a string are converted to upper case while respecting the current locale.
16	toLowerCase() Returns the calling string value converted to lower case.
17	toString() Returns a string representing the specified object.
18	toUpperCase() Returns the calling string value converted to uppercase.
19	valueOf() Returns the primitive value of the specified object.

String HTML Wrappers

Here is a list of the methods that return a copy of the string wrapped inside an appropriate HTML tag.

Sr.No.	**Method & Description**

1	anchor() Creates an HTML anchor that is used as a hypertext target.
2	big() Creates a string to be displayed in a big font as if it were in a <big> tag.
3	blink() Creates a string to blink as if it were in a <blink> tag.
4	bold() Creates a string to be displayed as bold as if it were in a <b> tag.
5	fixed() Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag
6	fontcolor() Causes a string to be displayed in the specified color as if it were in a <font color="color"> tag.
7	fontsize() Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag.
8	italics() Causes a string to be italic, as if it were in an <i> tag.
9	link() Creates an HTML hypertext link that requests another URL.
10	small() Causes a string to be displayed in a small font, as if it were in a <small> tag.
11	strike() Causes a string to be displayed as struck-out text, as if it were in a <strike> tag.
12	sub() Causes a string to be displayed as a subscript, as if it were in a <sub> tag
13	sup() Causes a string to be displayed as a superscript, as if it were in a <sup> tag

Search method

This method executes the search for a match between a regular expression and this String object and search function

Syntax : **string.search(regexp);**

Argument Details

regexp − A regular expression object. If a non-RegExp object **obj** is passed, it is implicitly converted to a RegExp by using **new RegExp(obj)**.

Return Value

If successful, the search returns the index of the regular expression inside the string. Otherwise, it returns -1.

Example

```
<html>
  <head>
    <title>JavaScript String search() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var re = /apples/gi;
      var str = "Apples are round, and apples are juicy.";

      if ( str.search(re) == -1 ) {
        document.write("Does not contain Apples" );
      } else {
        document.write("Contains Apples" );
      }
    </script>
  </body>
</html>
```

Output

```
Contains Apples
```

Date Object

The Date object is a datatype built into the JavaScript language.

Date objects are created with the **new Date( )** :

new Date( )
new Date(milliseconds)
new Date(datestring)
***new Date(year,month,date[,hour,minute,second,millisecond])

Description of the parameters −

- **No Argument** − With no arguments, the Date() constructor creates a Date object set to the current date and time.
- **milliseconds** − When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
- **datestring** − When one string argument is passed, it is a string representation of a date, in the format accepted by the **Date.parse()** method.
    - **7 agruments** −****
    - **year** − Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
    - **month** − Integer value representing the month, beginning with 0 for January to 11 for December.
    - **date** − Integer value representing the day of the month.
    - **hour** − Integer value representing the hour of the day (24-hour scale).
    - **minute** − Integer value representing the minute segment of a time reading.
    - **second** − Integer value representing the second segment of a time reading.
    - **millisecond** − Integer value representing the millisecond segment of a time reading.

Date Properties

Here is a list of the properties of the Date object along with their description.

Sr.No.	**Property & Description**
1	constructor Specifies the function that creates an object's prototype.
2	prototype The prototype property allows you to add properties and methods to an object

Date Methods

List of the methods used with **Date**

Sr.No.	Method & Description
1	Date() Returns today's date and time
2	getDate() Returns the day of the month for the specified date according to local time.
3	getDay() Returns the day of the week for the specified date according to local time.
4	getFullYear() Returns the year of the specified date according to local time.
5	getHours() Returns the hour in the specified date according to local time.
6	getMilliseconds() Returns the milliseconds in the specified date according to local time.
7	getMinutes() Returns the minutes in the specified date according to local time.
8	getMonth() Returns the month in the specified date according to local time.
9	getSeconds() Returns the seconds in the specified date according to local time.
10	getTime() Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
11	getTimezoneOffset() Returns the time-zone offset in minutes for the current locale.
12	getUTCDate() Returns the day (date) of the month in the specified date according to universal time.
13	getUTCDay() Returns the day of the week in the specified date according to universal time.
14	getUTCFullYear() Returns the year in the specified date according to universal time.
15	getUTCHours() Returns the hours in the specified date according to universal time.

16	getUTCMilliseconds() Returns the milliseconds in the specified date according to universal time.
17	getUTCMinutes() Returns the minutes in the specified date according to universal time.
18	getUTCMonth() Returns the month in the specified date according to universal time.
19	getUTCSeconds() Returns the seconds in the specified date according to universal time.
20	getYear() **Deprecated** - Returns the year in the specified date according to local time. Use getFullYear instead.
21	setDate() Sets the day of the month for a specified date according to local time.
22	setFullYear() Sets the full year for a specified date according to local time.
23	setHours() Sets the hours for a specified date according to local time.
24	setMilliseconds() Sets the milliseconds for a specified date according to local time.
25	setMinutes() Sets the minutes for a specified date according to local time.
26	setMonth() Sets the month for a specified date according to local time.
27	setSeconds() Sets the seconds for a specified date according to local time.
28	setTime() Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
29	setUTCDate() Sets the day of the month for a specified date according to universal time.
30	setUTCFullYear() Sets the full year for a specified date according to universal time.
31	setUTCHours() Sets the hour for a specified date according to universal time.

32	setUTCMilliseconds() Sets the milliseconds for a specified date according to universal time.
33	setUTCMinutes() Sets the minutes for a specified date according to universal time.
34	setUTCMonth() Sets the month for a specified date according to universal time.
35	setUTCSeconds() Sets the seconds for a specified date according to universal time.
36	setYear() **Deprecated -** Sets the year for a specified date according to local time. Use setFullYear instead.
37	toDateString() Returns the "date" portion of the Date as a human-readable string.
38	toGMTString() **Deprecated -** Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
39	toLocaleDateString() Returns the "date" portion of the Date as a string, using the current locale's conventions.
40	toLocaleFormat() Converts a date to a string, using a format string.
41	toLocaleString() Converts a date to a string, using the current locale's conventions.
42	toLocaleTimeString() Returns the "time" portion of the Date as a string, using the current locale's conventions.
43	toSource() Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
44	toString() Returns a string representing the specified Date object.
45	toTimeString() Returns the "time" portion of the Date as a human-readable string.
46	toUTCString() Converts a date to a string, using the universal time convention.

47	valueOf() Returns the primitive value of a Date object.

EXAMPLE

Javascript **Date()** method returns today's date and time and does not need any object to be called.

Syntax-Date()

Return Value- Returns today's date and time.

Example

```
<html>
  <head>
    <title>JavaScript Date Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var dt = Date();
      document.write("Date and Time : " + dt );
    </script>
  </body>
</html>
```

Output

```
Date and Time : Tue Apr 14 2020 11:32:36 GMT+0530 (India Standard Time)
```

Date Static Methods

In addition to the many instance methods listed previously, the Date object also defines two static methods. These methods are invoked through the Date() constructor itself.

Sr.No.	Method & Description
1	Date.parse( ) Parses a string representation of a date and time and returns the internal millisecond representation of that date.
2	Date.UTC( ) Returns the millisecond representation of the specified UTC date and time.

Number Object

The **Number** object represents numerical date, either integers or floating-point numbers.

Syntax- var val = new Number(number);

Note: In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

Number Properties

Sr.No.	Property & Description
1	MAX_VALUE The largest possible value a number in JavaScript can have 1.7976931348623157E+308
2	MIN_VALUE The smallest possible value a number in JavaScript can have 5E-324
3	NaN Equal to a value that is not a number.
4	NEGATIVE_INFINITY A value that is less than MIN_VALUE.
5	POSITIVE_INFINITY A value that is greater than MAX_VALUE
6	prototype A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document
7	constructor Returns the function that created this object's instance. By default this is the Number object.

Number Methods

The Number object contains only the default methods that are a part of every object's definition.

Sr.No.	Method & Description
1	toExponential() Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
2	toFixed() Formats a number with a specific number of digits to the right of the decimal.

3	toLocaleString() Returns a string value version of the current number in a format that may vary according to a browser's local settings.
4	toPrecision() Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
5	toString() Returns the string representation of the number's value.
6	valueOf() Returns the number's value.

valueOf-This method returns the primitive value of the specified number object.

Syntax : number.valueOf()

Return Value- Returns the primitive value of the specified number object.

Example

```
<html>
  <head>
    <title>JavaScript valueOf() Method </title>
  </head>

  <body>
    <script type = "text/javascript">
       var num = new Number(15.11234);
       document.write("num.valueOf() is " + num.valueOf());
    </script>
  </body>
</html>
```

Output

```
num.valueOf() is 15.11234
```

Math object

The **math** object provides properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.

Note: The ***static*** *keyword defines a* ***static*** *method for a class.* ***Static*** *methods aren't called on instances of the class. Instead, they're called on the class itself. These are often utility functions.*

Syntax

```
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

Math Properties

Sr.No.	Property & Description
1	E \ Euler's constant and the base of natural logarithms, approximately 2.718.
2	LN2 Natural logarithm of 2, approximately 0.693.
3	LN10 Natural logarithm of 10, approximately 2.302.
4	LOG2E Base 2 logarithm of E, approximately 1.442.
5	LOG10E Base 10 logarithm of E, approximately 0.434.
6	PI Ratio of the circumference of a circle to its diameter, approximately 3.14159.
7	SQRT1_2 Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
8	SQRT2 Square root of 2, approximately 1.414.

Math Methods

Sr.No.	Method & Description
1	abs()

	Returns the absolute value of a number.
2	acos() Returns the arccosine (in radians) of a number.
3	asin() Returns the arcsine (in radians) of a number.
4	atan() Returns the arctangent (in radians) of a number.
5	atan2() Returns the arctangent of the quotient of its arguments.
6	ceil() Returns the smallest integer greater than or equal to a number.
7	cos() Returns the cosine of a number.
8	exp() Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	floor() Returns the largest integer less than or equal to a number.
10	log() Returns the natural logarithm (base E) of a number.
11	max() Returns the largest of zero or more numbers.
12	min() Returns the smallest of zero or more numbers.
13	pow() Returns base to the exponent power, that is, base exponent.
14	random() Returns a pseudo-random number between 0 and 1.
15	round() Returns the value of a number rounded to the nearest integer.
16	sin() Returns the sine of a number.
17	sqrt()

	Returns the square root of a number.
18	tan() Returns the tangent of a number.

Example

Math.max() function is used to return the largest of zero or more numbers.

Syntax:
Math.max(value1, value2, ...)

The result is “-Infinity” if no arguments are passed
The result is NaN if at least one of the arguments cannot be converted to a number.

I)

```
<script type="text/javascript">
   document.write("Output : " +
Math.max(10, 32, 2));
</script>
```

Output: 32

II)

```
<script type="text/javascript">
   document.write("Output : " +
Math.max());
</script>
```

Output:????

REGULAR EXPRESSIONS

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. These patterns are used with the exec() and test() methods of RegExp, and with the match(), matchAll(), replace(), search(), and split() methods of String. This chapter describes JavaScript regular expressions.

Creating a regular expression

You construct a regular expression in one of two ways:

- Using a regular expression literal, which consists of a pattern enclosed between slashes, as follows:

```
let re = /ab+c/;
```

Regular expression literals provide compilation of the regular expression when the script is loaded. If the regular expression remains constant, using this can improve performance.

- Or calling the constructor function of the RegExp object, as follows:

```
let re = new RegExp('ab+c');
```

Using the constructor function provides runtime compilation of the regular expression. Use the constructor function when you know the regular expression pattern will be changing, or you don't know the pattern and are getting it from another source, such as user input.

Writing a regular expression pattern

A regular expression pattern is composed of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab*c/ or /Chapter (\d+)\.\d*/. The last example includes parentheses, which are used as a memory device.

Using simple patterns

Simple patterns are constructed of characters for which you want to find a direct match. For example, the pattern /abc/ matches character combinations in strings only when the exact sequence "abc" occurs (all characters together and in that order). Such a match would succeed in the strings "Hi, do you know your abc's?" and "The latest airplane designs evolved from slabcraft." In both cases the match is with the substring "abc". There is no match in the string "Grab crab" because while it contains the substring "ab c", it does not contain the exact substring "abc".

Using special characters

When the search for a match requires something more than a direct match, such as finding one or more b's, or finding white space, you can include special characters in the pattern. For example, to match *a single "a" followed by zero or more "b"s followed by "c"*, you'd use the pattern /ab*c/: the * after "b" means "0 or more occurrences of the preceding item." In the string "cbbabbbbcdebc", this pattern will match the substring "abbbbc".

Boolean Object

The **Boolean** object represents two values, either "true" or "false". If *value* parameter is omitted or is 0, -0, null, false, **NaN,** undefined, or the empty string (""), the object has an initial value of false.

Syntax

var val = new Boolean(value);

Boolean Properties

Sr.No.	Property & Description
1	constructor Returns a reference to the Boolean function that created the object.
2	prototype The prototype property allows you to add properties and methods to an object.

Boolean Methods

Sr.No.	Method & Description
1	toSource() Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	toString() Returns a string of either "true" or "false" depending upon the value of the object.
3	valueOf() Returns the primitive value of the Boolean object.

```
<html>

<body>
   <script>
      var v1 = true;
      var v2 = new Boolean(true);
document.write('v1 = = v2 is ' + (v1 == v2));
      document.write("<br>"); document.write('v1 = = = v2 is ' +
(v1 = = = v2));
   </script>
</body>
</html>
```

Output:

```
v1 = = v2 is true

v1 = = = v2 is false
```

Note: v1 = = = v2 is not true as the type of v1 and v2 (object) is not same. (checks for equal value & equal type)

Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **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.

Example

```
<html>
  <head>
    <title>Form Validation</title>
    <script type = "text/javascript">
      <!--
        // Form validation code will come here.
      //-->
    </script>
  </head>

  <body>
    <form action = "/cgi-bin/test.cgi" name = "myForm" onsubmit = "return(validate());">
      <table cellspacing = "2" cellpadding = "2" border = "1">

        <tr>
          <td align = "right">Name</td>
          <td><input type = "text" name = "Name" /></td>
        </tr>

        <tr>
          <td align = "right">EMail</td>
          <td><input type = "text" name = "EMail" /></td>
        </tr>

        <tr>
          <td align = "right">Zip Code</td>
          <td><input type = "text" name = "Zip" /></td>
        </tr>

        <tr>
          <td align = "right">Country</td>
          <td>
            <select name = "Country">
              <option value = "-1" selected>[choose yours]</option>
              <option value = "1">USA</option>
              <option value = "2">UK</option>
              <option value = "3">INDIA</option>
            </select>
          </td>
        </tr>

        <tr>
          <td align = "right"></td>
          <td><input type = "submit" value = "Submit" /></td>
        </tr>

      </table>
    </form>
  </body>
</html>
```

Basic Form Validation

In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

```
<script type = "text/javascript">
  <!--
    // Form validation code will come here.
    function validate() {

      if( document.myForm.Name.value == "" ) {
        alert( "Please provide your name!" );
        document.myForm.Name.focus() ;
        return false;
      }
      if( document.myForm.EMail.value == "" ) {
        alert( "Please provide your Email!" );
        document.myForm.EMail.focus() ;
        return false;
      }
      if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||
        document.myForm.Zip.value.length != 5 ) {

        alert( "Please provide a zip in the format #####." );
        document.myForm.Zip.focus() ;
        return false;
      }
      if( document.myForm.Country.value == "-1" ) {
        alert( "Please provide your country!" );
        return false;
      }
      return( true );
    }
  //-->
</script>
```

Data Format Validation

Example - Validation of an entered email address.

An email address must contain at least a '@' sign and a dot (.).

Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

```
<script type = "text/javascript">
  <!--
    function validateEmail() {
      var emailID = document.myForm.EMail.value;
      atpos = emailID.indexOf("@");
      dotpos = emailID.lastIndexOf(".");

      if (atpos < 1 || ( dotpos - atpos < 2 )) {
        alert("Please enter correct email ID")
        document.myForm.EMail.focus() ;
        return false;
      }
      return( true );
    }
  //-->
</script>
```