Unit-8

JAVASCRIPT

Basics of Javascript: Client Side Scripting with JS

- JavaScript is one of the most used languages when it comes to building web applications as it allows developers to wrap HTML and CSS code in it to make web apps interactive.
- It enables the interaction of users with the web application. It is also used for making animations on websites
- JavaScript was initially created to **"make web pages alive".**
- The programs in this language are called scripts. They can be written right in a web page's HTML and run automatically as the page loads.
- Scripts are provided and executed as plain text. They don't need special preparation or compilation to run.
- It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.
- It is an interpreted programming language with object-oriented capabilities.
- It is used both client-side and server-side to make web apps interactive.
- Uses built-in browser DOM.
- Extension of JavaScript file is **.js**

Why is it called JavaScript?

When JavaScript was created, it initially had another name: "**LiveScript**". But Java was very popular at that time, so it was decided that positioning a new language as a "younger brother" of Java would help.

But as it evolved, JavaScript became a fully independent language with its own specification called **ECMAScript**, and now it has no relation to Java at all.

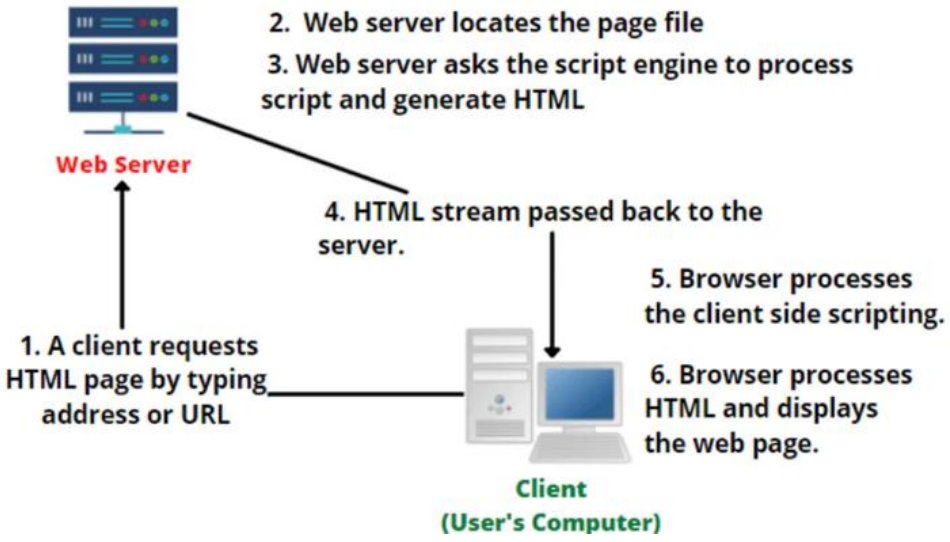
ECMAScript, also known as JavaScript, is a programming language adopted by the European Computer Manufacturer's Association as a standard for performing computations in Web applications.

History of JavaScript

In 1993, **Mosaic**, the first popular web browser, came into existence. In the **year 1994**, **Netscape** was founded by **Marc Andreessen**. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTML to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited **Brendan Eich** intending to implement and embed Scheme programming language to the browser. But, before Brendan could start, the company merged with **Sun Microsystems** for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'Javascript'. Finally, in May 1995, Marc Andreessen coined the first code of Javascript named '**Mocha**'. Later, the marketing team replaced the name with '**LiveScript**'. But, due to trademark reasons and certain other reasons, in December 1995, the language was finally renamed to 'JavaScript'. From then, JavaScript came into existence.

Client-side scripting

- A client-side script is a tiny program (or collection of instructions) that is put into a web page. It is handled by the client browser rather than the web server.
- In other words, client-side scripting is a method for browsers to run scripts without having to connect to a server.
- The code runs on the client's computer's browser either while the web page is loading or after it has finished loading.
- Client-side scripting is mostly used for dynamic user interface components including pull-down menus, navigation tools, animation buttons, and data validation.
- The client refers to the script that runs on the user's computer system. It can either be integrated (or injected) into the HTML content or stored in a separate file (known as an external script).
- When the script files are requested, they are transmitted from the web server (or servers) to the client system. The script is run by the client's web browser, which subsequently displays the web page, including any visible script output.
- look at the diagram below.



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.

Advantages of JavaScript

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

- We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:
- Client-side JavaScript does not allow the reading or writing of files. It has been kept for the security reason.
- JavaScript could not used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities

Syntax of JavaScript

- JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.
- You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.
- The script tag takes two important attributes –
  - **Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
  - **Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".
- The <script> tag alerts the browser program to start interpreting all the text between these tags as a script.

```
<script language = "javascript" type = "text/javascrit">
JavaScript code
</script>
```

Internal JavaScript

JavaScript can be added directly to the HTML file by writing the code inside the <script> tag. We can place the <script> tag either inside <head> or the <body> tag according to the need.

Example:

```
<html>
  <head>
    <script language = "javascript" type = "text/javascript">
        document.write("Hello World!")
        console.log(“Hello!”)
    </script>
  </head>
</html>
```

External JavaScript

The other way is to write JavaScript code in another file having a .js extension and then link the file inside the **<head> or <body>** tag of the HTML file in which we want to add this code.

Example:

j1.html

```
<html>
  <head>
    <script src="j1.js"></script>
  </head>
</html>
```

j1.js

```
document.write("This is written in external file")
```

JavaScript in <body> and <head> sections together: Always execute script in <head> first then in body

```
<!DOCTYPE html>
<head>
   <script type="text/javascript" language="javascript">
      function sayHello(){
                  alert("Hello World")
         }
document.write("Hello World")
   </script>
</head>
<body>
   <script type="text/javascript" language="javascript">
         document.write("How r u?")
   </script>
   <input type="button" onClick="sayHello()" value="Greetings"/>
</body>
</html>
```

Output:

Variables

- **Variables are Containers for Storing Data**
- **Variables** are the building blocks of any programming language.
- In JavaScript, variables can be used to store reusable values.
- The values of the variables are allocated using the assignment operator("=").

JavaScript Identifiers

JavaScript variables must have unique names. These names are called **Identifiers**.
There are some basic rules to declare a variable in JavaScript:

- Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
- After first letter we can use digits (0 to 9), for example value1.
- JavaScript variables are case sensitive, for example x and X are different variables.
- A variable name cannot be a reserved keyword.

JavaScript is a dynamically typed language so the type of variables is decided at runtime. Therefore, there is no need to explicitly define the type of a variable. We can declare variables in JavaScript in four ways:

1. Automatically (Implicit)
2. var
3. let
4. const

All three keywords do the basic task of declaring a variable but with some differences Initially, **all the variables in JavaScript were written using the var keyword** but **in ES6 the keywords let and const were introduced.**

JavaScript local variable

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

```
<script>
function abc(){
var x=10;//local variable
document.write(x)
}
abc();//calling JavaScript function
</script>
```

Output:

10

JavaScript global variable

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

```
<script>
var d=200;//global variable
function a(){
document.writeln(d);
}
document.writeln(d);
a();//calling JavaScript function
</script>
```

Output:

200 200

1. Undeclared

In this first example, x, y, and z are undeclared variables. They are automatically declared when first used:

```
x = 5;
y = 6;
z = x + y;
```

2. Var Method

The **var** is the oldest keyword to declare a variable in JavaScript.

It has the Global scoped or function scoped which means variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function.

The below code example explains the use of the var keyword to declare the variables in JavaScript.

```
var a = 10
function f1() {
var b = 20
console.log(a, b)
}
f1();
console.log(a);
```

Output:
10 20
10

The below example explains the behavior of var variables when declared inside a function and accessed outside of it.

```
function f1() {
// It can be accessible anywhere within this function
var a = 10;
console.log(a)
}
f1();

// A cannot be accessible outside of function
console.log(a);
```

Output:

10
ReferenceError: a is not defined

The below code re-declare a variable with same name in the same scope using the var keyword, which gives no error in the case of var keyword.

```
var a = 10
// User can re-declare variable using var
var a = 8
// User can update var variable
a = 7
console.log(a);
```

Output:

7

The below code explains the initialized the variables after accessing.

```
console.log(a);
var a = 10; //initialized the variable after accessing
```

Output:

Undefined

3. Let Method

The let keyword is an improved version of the var keyword.
It is introduced in the ES6 or EcmaScript 2015.
These variables has the block scope.
It can't be accessible outside the particular code block ({block}).

The below code declares the variable using the let keyword.

```
let a = 10;
function f() {
let b = 9
console.log(b);
console.log(a);
}
f();
console.log(b);

Output:
9
10
Uncaught ReferenceError: b is not defined
```

The below code explains the behavior of let variables when they are re-declared in the same scope.

```
let a = 10
// It is not allowed
let a = 10
// It is allowed
a = 10

Output:
Uncaught SyntaxError:Identifier 'a' has already been declared
```

The below code explains the behavior of let variable when it isre-declared in the different scopes.

```
let a = 10
function f1() {
let a = 9
console.log(a) // It prints 9
}
f1()
console.log(a) // It prints 10
Output:
9
10
```

The below code explains the initialized the variables after accessing.

```
console.log(a);
let a = 10; //initialized the variable after accessing
Output:

Uncaught ReferenceError: Cannot access 'a' before initialization
```

4. Const Method

The const keyword has all the properties that are the same as the let keyword, except the user cannot update it and have to assign it with a value at the time of declaration.
These variables also have the block scope.
It is mainly used to create constant variables whose values can not be changed once they are initialized with a value.
This code tries to change the value of the const variable.

```
const a = 10;
function f() {
a = 9
console.log(a)
}
f();
```

Output:

TypeError:**Assignment to constant variable.**

No reassignment is happening—two independent const variables (a) exist in different scopes.

```
const a = 10;
function f() {
   const a = 9
   console.log(a)
}
f();
console.log(a);
```

Output:
9
10

This works without error because the const a = 9; inside the function creates a new variable in the function's local scope**, shadowing the global a.**
Note:

The variables declared with var and let are mutable that is their value can be changed but variables declared using const are immutable.

Example on Update and Redeclare of var/let/const in same scope.

```
<script>
// Redeclaring with var
var g = 5;
var g = 10;  // Redeclaring g is allowed with var
document.write("Redeclaring with var"+g); // 10
```

```
// Updating with var
var h = 5;
h = 10;  // Updating the value of h
document.write("<br>Updating with var"+h); // 10

// Redeclaring with let
let i = 5;
let i = 10;  // Error: Identifier 'i' has already been declared
document.write("<br>Redeclaring with let"+i)

// Updating with let
let j = 5;
j = 10;  // Updating the value of j
document.write("<br>Updating with let"+j); // 10

// Redeclaring with const
const k = 5;
const k = 10;  // Error: Identifier 'k' has already been declared
document.write("<br>Redeclaring with const"+k)

// Updating with const
const l = 5;
l = 10;  // Error: Assignment to constant variable
document.write(l)
</script>
```

var	let	const
The scope of a *var* variable is functional or global scope.	The scope of a *let* variable is block scope.	The scope of a *const* variable is block scope.
It can be updated and re-declared in the same scope.	It can be updated but cannot be re-declared in the same scope.	It can neither be updated or re-declared in any scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
It can be accessed without initialization as its default value is “undefined”.	It cannot be accessed without initialization otherwise it will give ‘referenceError’.	It cannot be accessed without initialization, as it cannot be declared without initialization.

Data Types

- JavaScript provides different data types to hold different types of values.
- There are two types of data types in JavaScript.
  1. **Primitive data type**
  2. **Non-primitive (reference) data type**
- JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine.
- You need to use var, let or const here to specify the data type.
- It can hold any type of values such as numbers, strings etc.

For example:

```
var a=20;//holding number

var b="LJU";//holding string
```

Primitive data types

Primitive data types are also known as in-built data types.primitive data types are a set of basic data types from which all other data types are constructed.
Below are the primitive data types

1. **String** represents sequence of characters e.g. "hello"
   x = "LJU"; //x is a String

2. **Number** represents numeric values e.g. 100

   // With decimals:
   let x1 = 24.00;
   // Without decimals:
   let x2 = 24;

3. **Boolean** represents Boolean value either **false** or **true**

   let x = 5;
   let y = 5;
   let z = 6;
   (x == y) // Returns true
   (x == z) // Returns false

   == is the comparison operator. It will only return true if both values are equivalent after coercing their types to the same type.
   === is a more strict comparison operator often called the identity operator. It will only return true if both the type and value of the operands are the same.

4. **Undefined** represents undefined value. A variable has been declared but not assigned a value.
   let name;   // Value is undefined, type is undefined

5. **Null** represents null i.e. no value at all
   let name = null; // value is null

All primitive types, except null, can be tested by the typeof operator.
typeof null returns "object", so one has to use === null to test for null.

Non-primitive data types

Non-primitive data types are called reference types because they refer to objects.

The main difference between primitive and non-primitive data types are: Primitive types are predefined (already defined) in Java.

Non-primitive types are created by the programmer and is not defined by Java (except for String).

The non-primitive data types are as follows:

1) **Object**
2) **Array**

User Defined Objects

It represents instance through which we can access members

- JavaScript objects are written with curly braces {}.
- Object properties are written as name:value pairs, separated by commas.

Example: const person = {firstName:"ABC", lastName:"DEF", age:30};

```
<script>
  const person = {firstName:"Priyen", lastName:"Patel"};
  document.write(person.firstName);
</script>
```

Output:
Priyen

JavaScript objects are best explained by thinking of a real-world object. Take a car for example. Cars come in all shapes and sizes - different colors, different makes and models, different weight, etc. Every car has these properties, but the values are different from one car to another. A red Ford Focus and a blue Honda Civic are both “cars”, but they are different makes, models, and colors.

JavaScript objects are variables that contain multiple data values. The values within a JS object are known as properties. Objects use keys to name values, much like how is done with variables.

1. By Object Literal

Syntax

```
object = {property1 : value1, property2 : value2,.. propertyN : valueN}
```

Example

```
<html>
<body>
<script type="text/javascript" language="javascript">
var emp ={id:101, name:"xyz", salary:60000};
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```

2. By creating instance of object (*Reference)

Synatx

```
var objectname = new Object();
```

Example

```
<html>
<body>
<script type="text/javascript" language="javascript">
var emp = new Object();
emp.id = 101;
emp.name = "xyz";
emp.salary = 60000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```

3. By using an object constructor (*Reference)

JavaScript is a **prototype-based object-oriented language** as it has no classes like other object-oriented languages.

Constructors are methods that are automatically executed every time you create an object. The purpose of a constructor is to construct an object and assign values to the object's members. A constructor takes the same name as the class to which it belongs, and does not return any values.

A constructor is a special function that creates and initializes an object instance of a class**. In JavaScript, a constructor gets called when an object is created using the new keyword. The purpose of a constructor is to create a new object and set values for any existing object properties.**

Example

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

```
<script type="text/javascript" language="javascript">
function emp(id, name, salary)
{
this.id = id;
this.name = name;
this.salary = salary;
}
let employee = new emp(101, "xyz", 60000);
document.write(employee.id+" "+ employee.name+" "+ employee.salary);
</script>
</body>
</html>
```

Explanation:

In the above example, the 'new' keyword creates an empty object.

Here,emp() includes three properties 'id, 'name', and 'salary' that are declared with 'this' keyword. In a constructor function this does not have a value. It is a substitute for the new object. The value of this will become the new object when a new object is created.

So, a new empty object will now include all these properties and the newly created objects are returned as employee().

Array

It represents group of similar values

- Arrays are a special kind of objects, with numbered indexes.
- An array can hold many values under a single name, and you can access the values by referring to an index number.
- It is a common practice to declare arrays with the const keyword.
- JavaScript arrays are written with square brackets.
- Array items are separated by commas.
- The following code declares (creates) an array called fruits, containing three items (car names):

Example: const fruits = ["grapes", "mango", "watermelon"];

- There are two main ways to create arrays in JavaScript:
  - you can use a literal, or
  - you can use a constructor. (*Reference)
- The **literal** looks like this and can be defined with or without any array elements. The one above doesn't have any elements yet.

```
const myArray = [];
```

- The **constructor** looks like this and uses the new keyword on the Array object. **(*Reference)**

const myArray = new Array();
In JavaScript, a constructor gets called when an object is created using the new keyword. The purpose of a constructor is to create a new object and set values for any existing object properties.

The length Property

The length property of an array returns the length of an array (the number of array elements).

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

The length property is always one more than the highest array index.

Accessing the First Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0]
```

Accessing the Last Array Element

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

Example: Display all elements of an array in new line.

```
<script type="text/javascript" language="javascript">
    var emp =["abc", "pqr", "xyz"];
    for(i=0; i<emp.length; i++) {
        document.write(emp[i]+"<br>");
    }
</script>
```

Output:
abc
pqr
xyz

Example: Multiply each elements of an array by 2 and display in new line.

```
  <script>
    var num = [2,4,3,1,5];
    for(i=0; i<num.length; i++) {
      let mul = num[i]*2;
    document.write("multiplication is: "+ mul+"<br>");
    }
    </script>
```

Output:
multiplication is: 4
multiplication is: 8
multiplication is: 6
multiplication is: 2
multiplication is: 10

When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

Example to find datatype using "typeof()"

```
var a = undefined;
var b = "test";
var c=true;
var d=5;
let e=null;
document.write(typeof(a)+"<br>")
document.write(typeof(b)+"<br>")
document.write(typeof(c)+"<br>")
document.write(typeof(d)+"<br>")
document.write(typeof(e)+"<br>")
```

Output:
undefined
string
boolean
number
object

Example to show way of reading variables.

```
<html>
<body>
<script type="text/javascript" language="javascript">
var x = 16 + 4 +"xyz";  //Treats 16 and 4 as numbers, until it reaches "xyz".
var y = "xyz" + 16 + 4;  // the first operand is a string, all operands are treated as strings.
document.write(x+"<br>");
document.write(y);
</script>
```

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

Output:
20xyz
xyz164

Javascript Array constructor (*reference)

Example

```
<html>
<body>
<script type="text/javascript" language="javascript">
var i;
var emp = new Array("abc", "xyz", "pqr");
for(i=0; i<emp.length; i++)
{
document.write(emp[i]+"<br>");

}
console.log(typeof emp); // "object"
console.log(Array.isArray(emp)); // true
</script>
</body>
</html>
```

Output:
abc
xyz
pqr

Functions

Functions: Syntax, Calling function on some event

- A function is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing the same code again and again.
- Functions allow a programmer to divide a big program into a number of small and manageable functions.
- Before we use a function, we need to define it.
- 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: **{}**

```
function Name(Parameter1, Parameter2, ...)
{
   // Function body
}
```

- A function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output.
- To use a function, you must define it somewhere in the scope from which you wish to call it.

Function Invocation:

- Triggered by an event (e.g., a button click by a user).
- When explicitly called from JavaScript code.
- Automatically executed, such as in self-invoking functions.

Example of function without passing parameter

```
<head>
    <script>
        function fun() {
              document.write ("Hello! How are you?");
         }
     </script>
 </head>
 <body> <input type = "button" onclick = "fun()" value = "Click"></body>
```

Example of function with passing parameter

```
<head>
    <script>
        function fun(f_name,l_name) {
              document.write ("Hello " + f_name  + " " + l_name + " !");
         }
     </script>
 </head>
 <body> <input type = "button" onclick = "fun('ABC','XYZ')" value = "Click"></body>
```

JavaScript return statement

- It is used to return a particular value from the function. The function will stop the execution when the return statement is called and return a specific value.
- The return statement should be the last statement of the function because the code after the return statement won't be accessible.
- We can return any value i.e. Primitive value (Boolean, number and string, etc) or object type value (function, object, array, etc) by using the return statement.

```
return value;
```

Example:

```
function Product(a, b) {
// Return the product of a and b
return a * b;
};
console.log(Product(6, 10));
```

Assigned function to the variable

```
    var a = function ( x, y ) {
      return x + y;
    }
    let result = a(3, 5);
document.write(result);
```

or

```
var res = fun(12, 30);
function fun(x,y)
{
return x * y;
}
document.write(res);
```

Inbuilt Objects

JavaScript provides various built-in objects. Each object contains properties and methods. Some of the built-in objects in JavaScript are:

- Array
- Date
- Math
- String
- Number

Math object

The Math object provides a number of methods to work with Number values.

Methods available on *Math* object are:

Method	Description
max(a,b)	Returns largest of a and b
min(a,b)	Returns least of a and b
round(a)	Returns nearest integer
ceil(a)	Rounds up. Returns the smallest integer greater than or equal to a
floor(a)	Rounds down. Returns the largest integer smaller than or equal to a
pow(a,b)	Returns a^b
abs(a)	Returns absolute value of a
random()	Returns a pseudo random number between 0 and 1
sqrt(a)	Returns square root of a

```
<script type="text/javascript" language="javascript">
document.write("MATH FUNCTIONS");
document.write("<br/>Square root : "+Math.sqrt(9));
document.write("<br/>Absolute : "+Math.abs(-1));
document.write("<br/>Ceil : "+Math.ceil(1.4));
document.write("<br/>Negative Ceil : "+Math.ceil(-1.4));
document.write("<br/>Floor : "+Math.floor(1.7));
document.write("<br/>Negative Floor : "+Math.floor(-1.7));
document.write("<br/>Power: "+Math.pow(3,2));
document.write("<br/>Maximum: "+Math.max(3,2,2.4,3.5));
document.write("<br/>Minimum: "+Math.min(5,5.5));
</script>
```

Output:

MATH FUNCTIONS
Square root : 3
Absolute : 1
Ceil : 2
Negative Ceil : -1
Floor : 1
Negative Floor : -2
Power: 9
Maximum: 3.5
Minimum: 5

Date object

At times you there will be a need to access the current date and time and also past and future date and times. JavaScript provides support for working with dates and time through the ***Date*** object.

Date objected is created using the ***new* operator** and one of the *Date's* constructors. Current date and time can be retrieved as shown below:

```
var today = new Date( );
```

Methods available on *Date* object are:

Method	Description
new Date()	Creates a *Date* object with the current date and time of the browser's PC
Date("Month,dd, yyyy hh:mm:ss")	This creates a *Date* object with the specified string
Date("Month dd, yyyy")	This creates a *Date* object with the specified string
Date(yy,mm,dd,hh,mm,ss)	This creates a *Date* object with the specified string
Date(yy,mm,dd)	This creates a *Date* object with the specified date
Date(milliseconds)	Creates a *Date* object with the date value represented by the number of milliseconds since midnight on Jan 1, 1970
getFullYear()	Returns the 4-digit year component of the date.

getMonth()	Returns the month component of the date. Returns the month (0 to 11) of a date. January =0, February = 1, ... upto December=11
getDate()	Returns the date component of the date.
getDay()	Returns the day component of the date. ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday", "Saturday"] **(0-6)** Sunday-0 to Saturday- 6
getTime()	returns the number of milliseconds since January 1, 1970 00:00:00.
getHours()	Returns the hours component of the date
getMinutes()	Retrieves the minutes component of the date
getSeconds()	Retrieves the seconds component of the date
getMilliseconds()	Returns the milliseconds component of the date
setFullYear()	Sets the year component of the date using a 4-digit number
setMonth()	Sets the month component of the date. January =0, February = 1, ... upto December=11
setTime()	The setTime() method sets a date and time by adding or subtracting a specified number of milliseconds to/from midnight January 1, 1970.
setDate()	Sets the day-of-month component of the date
setHours()	Set the hours component of the date
setMinutes	Set the minutes component of the date
setSeconds()	Sets the seconds component of the date
setMilliseconds()	Sets the milliseconds component of the date

Example:

```
<body>
<script type="text/javascript" language="javascript">
var d1 = new Date();
document.write(d1+"<br>");
document.write(d1.getDay()+"<br>");
document.write(d1.getMonth()+"<br>");
document.write(d1.getDate()+"<br>");
document.write(d1.getFullYear()+"<br>");
document.write(d1.getHours()+"<br>");
document.write(d1.getMinutes()+"<br>");
document.write(d1.getSeconds()+"<br>");
</script> </body>
```

Output:

```
Wed Jan 31 2024 11:40:16 GMT-0800 (Pacific Standard Time)
3
0
31
2024
11
40
16
```

Example: Write a JavaScript that uses function to calculate how many days are left in your upcoming birthday from current date in same year.

```
<html>
<head>
<script>
function daysdifference(date1, date2)
{
   one_day =24*60*60*1000; // one day time in milliseconds (86400000)
   date1_ms= date1.getTime(); //get time in milliseconds ( 0 hours, 0 minutes, 0 seconds starting from January 1, 1970 to Sat Feb 10 2024)
   date2_ms= date2.getTime(); //get time in milliseconds ( 0 hours, 0 minutes, 0 seconds starting from January 1, 1970 to current date)
   difference_ms = Math.abs(date1_ms-date2_ms);
   return(Math.round(difference_ms/one_day));
}
var d1 = new Date();
d1.setDate(10); // set date
d1.setMonth(1); //set month
// now d1 = Sat Feb 10 2024
var d2 = new Date(); //current date
document.write("No of days left : "+ daysdifference(d1,d2));
</script>
</head>
</html>
```

String object

- ✓ A string is a collection of characters.
- ✓ JavaScript provides various properties and methods to work with String objects.
- ✓ One most frequently used property on String objects is *length*. The *length* property gives the number of characters in the given string. Consider the following example:

```
var str = “Hello World”;
var len = str.length;
```

The value stored in the *len* variable will be 11 which the number of characters in the string.

Methods available on *String* objects are:

Method	Description
charAt(index)	Returns the character at specified index
charCodeAt(index)	Returns the unicode value of the character at specified index. **(A-Z = 65-90) (a-z = 97-122)**
concat()	Combines two or more strings
indexOf(searchtext, index)	Searches for the specified string from the beginning of the string.
lastIndexOf(searchtext, index)	Searches for the specified string from the end of the string
replace(expr, new-string)	Replaces the old string with the specified new string
substr(start, length)	Returns a substring having length characters from a specified index **start** Required.The start position.First character is at index 0. If start is greater than the length, substr() returns "". If start is negative, substr() counts from the end of the string. **Length** Optional.The number of characters to extract. If omitted, it extracts the rest of the string
substring(start, end)	The substring() method extracts characters, between two indices (positions), from a string, and returns the substring. **start** Required.Start position.First character is at index 0. **end** Optional.End position (up to, but not including). If omitted: the rest of the string.

toUpperCase()	Returns the string after converting all characters to upper case
toLowerCase()	Returns the string after converting all characters to lower case
split(separator, limit)	Splits a string into an array of substring
trim()	The trim() method removes whitespace from both sides of a string.

Difference between substr() & substring()

Feature	substr()	substring()
Syntax	str.substr(start, length)	str.substring(start, end)
Parameters	start: Starting index	start: Starting index
Length	length: Number of characters to extract	end: Ending index (exclusive)

Example:

```
<html>
<body>
<script type="text/javascript" language="javascript">
document.write("<br/>STRING FUNCTIONS");
x="hello";
y=" World";
x1="this is
demo";
y1="is";
document.write("<br/>char at 0th position : "+x.charAt(0));
document.write("<br/>Unicode value at 0th position: "+x.charCodeAt(0));
document.write("<br/>concated string : "+x.concat(y));
document.write("<br/>first index occ: "+x1.indexOf(y1));
document.write("<br/>last index occ: "+x1.lastIndexOf(y1));
document.write("<br/>replace: "+x.replace("hell",y));
document.write("<br/>split: "+x.split("l"));
document.write("<br/>split: "+x1.split(" "));
document.write("<br/>sub str: "+x1.substr(2,5));
document.write("<br/>sub string: "+x1.substring(2,6));
document.write("<br/>sub string: "+x.toUpperCase());
document.write("<br/>sub string: "+y.toLowerCase());
</script>
</body>
</html>
```

Output:

STRING FUNCTIONS
char at 0th position : h
Unicode value at 0th position: 104
concated string : hello World
first index occ: 2
last index occ: 5
replace: Worldo
split: he,,o
split: this,is,demo
sub str: is is
sub string: is i
sub string: HELLO
sub string: world

Example 1:
Write a JS that find position of first occurrence of vowel 'a' and last occurrence of vowel 'a' in the given word "ajanta" also return the string between them.

```
<html>
<head>
<script type="text/javascript">
z="ajanta";
x=z.indexOf("a");
y=z.lastIndexOf("a");
document.write("<br/>first index occ: "+x);
document.write("<br/>last index occ: "+y);
document.write("<br/>sub string: "+z.substring(x+1,y));
</script>
</head>
<body>
</body>
</html>
```

Example 2:

Given digit is 23. divide it into two parts 2 and 3 and find 2^3

```
<html>
<head>
   <script type="text/javascript">
      n=23;mul=0;
      m=n%10;
      n=parseInt(n/10);
       mul=Math.pow(n,m);
       document.write("answer is :" + mul);
    </script>
</head>
</html>
```

Output:

answer is :8

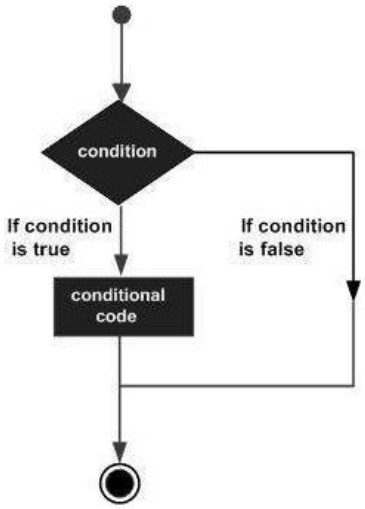
Conditions: If, If…Else

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false

If..Else Statement : The **JavaScript if-else statement** is used *to execute the code whether condition is true or false.*



Syntax:

The if Statement

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

The else Statement

```
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:

```
<html>
<body>
	<script type="text/javascript" language="javascript">
	var age= 20;
	if(age>=18) {
		document.write("You are eligible for voting");
	}
	else{
		document.write("You are not eligible for voting");
	}
</script>
</body>
</html>
```

Output:

You are eligible for voting

The If..Else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

Example:

```
<html><body>
<script type="text/javascript" language="javascript">
var sub="FSD-1";
if(sub=="FSD-1")
{
 document.write("PSP");
}
else if(sub=="FCSP-1")
{
 document.write("ABC");
}
```

```
else if(sub=="DE")
{
 document.write("XYZ");
}
else if(sub=="PS")
{
 document.write("MNO");
}
else
{
 document.write("Wrong Choice!!");
}
</script></body></html>
```

Output: PSP

Loops: for, while, do…while

Types of Loops

1. Entry Controlled (In this type of loop, the test condition is tested before entering the loop body) – while, for

2. Exit Controlled (In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at **least once**, irrespective of whether the test condition is true or false) **–** do… while

JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- 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

For Loop

It is used to iterate the elements for a fixed number of times. JavaScript for loop is used if the number of the iteration is known.

```
    Syntax:
for (initialization; test condition; iteration statement)
{
    Statement(s) to be executed if test condition is true
}
```

Example

```
<script type="text/javascript" language="javascript">
 for(count = 0; count < 10; count++)  {
                    document.write("Current Count : " + count );
                   document.write("<br >");
 }
</script>
```

Output:

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

Using **var** in a loop:

```
var i = 5;

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

Output:

0
1
2
3
4
5
6
7
8
9
Out: 10

Using **let** in a loop:

```
let i = 5;

for (let i = 0; i < 10; i++) {
  document.write (i+"<br>")
}
document.write ("Out: "+ i); // Here i is 5
```

Output:

0
1
2
3
4
5
6
7
8
9
Out: 5

- In the first example, using var, the variable declared in the loop redeclares the variable outside the loop.
- In the second example, using let, the variable declared in the loop does not redeclare the variable outside the loop. When let is used to declare the i variable in a loop, the i variable will only be visible within the loop.

While Loop

It is a control flow statement that allows the code to be executed repeatedly based on the given Boolean condition.

Syntax:

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

Example

```
<html><body>
<script type="text/javascript" language="javascript">
```

```
 var count = 0;
 while (count < 10)
 {                  document.write("Current Count : " + count + "<br>");
                   count++;
 }
</script></body></html>
```

Output:

Current Count : 0---------- Current Count : 9

Note: If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

Do..while Loop

It is a control flow statement that executes a block of code at least once, and then repeatedly executes the block or not depending on a given Boolean condition at the end of the block.

Synatax:

```
do
{
 Statement(s) to be executed if test condition is true
}
while(expression);
```

Example

```
<html><body>
<script type="text/javascript" language="javascript">
 var count = 0;
 do
 {
   document.write("Current Count : " + count+"<br>" );
       count++; // First print then count
 }
 while(count<10);
</script></body></html>
```

Output:

Current Count : 0---------- Current Count : 9

Break statement: The break statement is used to jump out of a loop. It can be used to "jump out" of a statement. It breaks the loop and continues executing the code after the loop.

Example

```
<html>
<body>
<script type="text/javascript" language="javascript">
 var x = 0;
 while (x < 10)
     {
                if (x == 5)
                {
                   break;
                 }
                x = x + 1;
                document.write( x + "<br>");
      }
</script></body></html>
```

Output:

1
2
3
4
5

Continue statement: 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.

Example

```
<html><head>
<script type="text/javascript" language="javascript">
var x = 0;
 while (x < 10)
 {
                x = x + 1;
                if (x == 5)
 {
                continue;
 }
 document.write( x + "<br />");
 }
</script></head></html>
```

Output:
1
2
3
4
6
7
8
9
10

Example: To find odd Number from (0 – 10)

```
for (let i = 0; i < 10; i++) {
  if (i % 2 === 0) {
    continue;
  }
  console.log(i);
}
```

Output:
1
3
5
7
9

Pop up Boxes: Alert, Confirm, Prompt

Alert Box

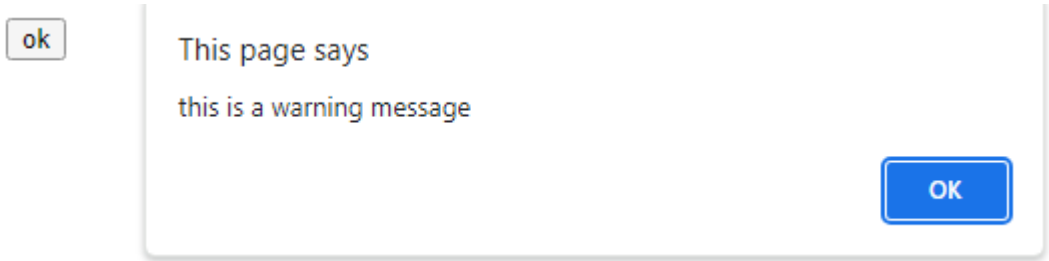
An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Example

```
<html>
<head>
<script>
function warn()
{
   alert("this is a warning message");
   document.write("Warning!!!!");    //It will be printed after closing alert box.//
}
</script>
</head>
<body>
   <input type="button" value="ok" onclick="warn()"/>
</body>
</html>
```

Output:



Confirmation Box

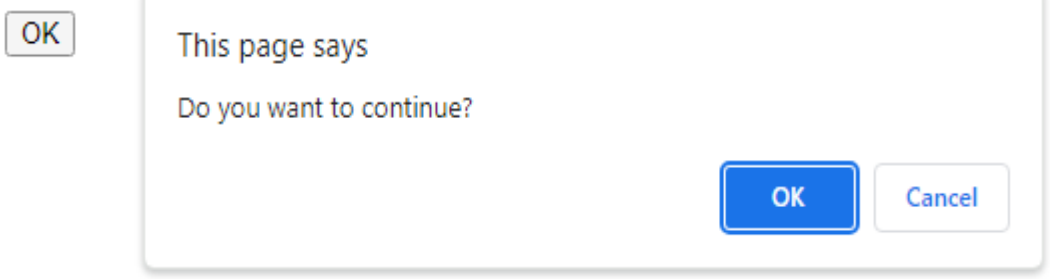
A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

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. You can use a confirmation dialog box as follows.

Example

```
<html>
<head>
<script type="text/javascript">
function conf()
{
ret = confirm("Do you want to continue?");          //It returns value as true or false//
if(ret==true)
{
   document.write("user wants");
   return true;
}
else
{
   document.write("user does not want");
   return false;
}
}
</script>
</head>
<body>
   <input type="button" onclick="conf()" value="OK"/>
</body>
</html>
```

Output:



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.

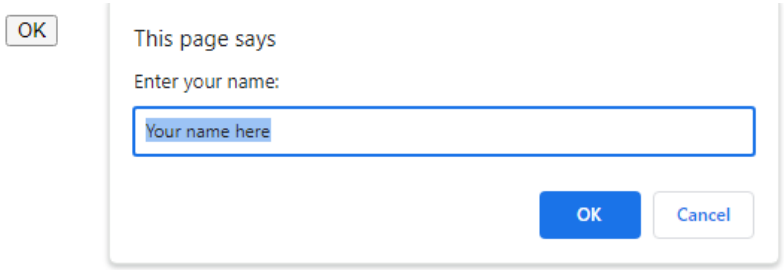
This dialog box is displayed using a method called **prompt()** which takes two parameters:

(i) a label which you want to display in the text box and
(ii) a default string to display in the text box.

Example:

```
<html>
<head>
<script type="text/javascript">
function getval()
{
   var ret = prompt("Enter your name: ", "Your name here");
   document.write("you entered "+ret);
}
</script>
</head>
<body>
<input type="button" onclick="getval()" value="OK"/>
</body>
</html>
```

Output:



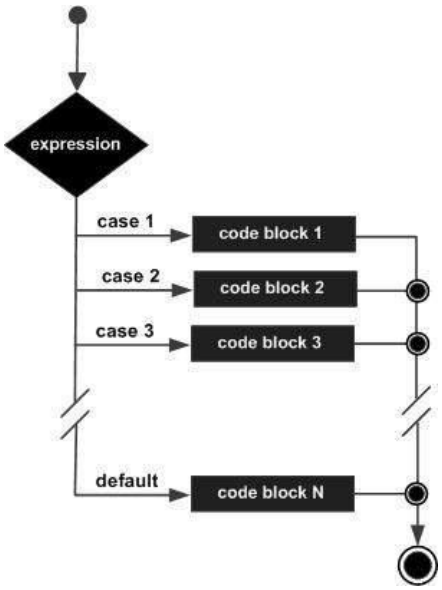
Switch (*Reference Only)

Multiple **if...else…if** statements, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

switch statement which handles exactly this situation, and it does so more efficiently than repeated **if...else if** statements.

Flow Chart

The following flow chart explains a switch-case statement works.



The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression.

The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

Syntax

```
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>
<head>
<script type="text/javascript" language="javascript">
var grade = 'A'; // for anything else from case trigger default condition
 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 />");
 }
</script></head></html>
```

Output:

Good job

Note: Break statements play a major role in switch-case statements. Try the above code that uses switch-case statement without any break statement.