# **JAVASCRIPT**

JavaScript is a programming language. When we build a webpage using HTML and CSS, these webpages will be considered as **Static webpages**, where there is no changes in the webpage. To overcome it we have a language which makes the webpages live i.e. **Dynamic webpages** by using the language JavaScript.

JavaScript is a programming language which makes our webpages **interactive** by adding some **live effects** to it.

JavaScript is a programming language which a browser can understand and makes webpages Dynamic. JavaScript adds functionality and behaviour to the webpage.

**History of JS:**

It was initially named as Live Script, later it got changed to JavaScript because of popularity of java.

**Note**: The organisation which maintains JS is ECMA (European Computer Manufacture Association).

The one who gives standard and current release of JS is ECMAScript (ES). Current release of ECMAScript is (ES - 14), Version learnt is (ES – 6).

JS can be executed in two ways:

* Inside the browser (JS Engine)
* Outside the browser (Node JS)

We can add JavaScript to the HTML file in two ways:

* Internal Scripting
* External Scripting

Both can be done with the help of Script tag, it is a paired tag.

1. **Internal Scripting:**

JS instruction will be added within the script tag.

Ex: <!-- Internal Scripting -->

        <script>

            console.log("hello world!!!")

        </script>

1. **External Scripting:**

To create a JS file we use an extension called filename.js.

Ex:

Filename.js

-

-

JS Instructions

-

Before closing of body tag in HTML document i.e. after loading HTML document add,

**<script src:”Filename.js”></script>**

**Features of JS:**

* High level language (human understandable)
* Loosely typed language
* Dynamically typed language
* Portable
* Free & Open source
* Platform independent language
* OOPs
* **Synchronous single threaded language**
* Backend development
* Asynchronous processing
* Light weight scripting language
* Scripting language (uses interpreter)
* Interpreted language (executes line by line)
* Various frameworks & libraries

1. **High level language :**

High level language means human understandable language

1. **Interpreted language :**

It executes the code line by line.

1. **Scripting language :**

The language which uses interpreter for its execution is known as scripting language. The most popular scripting languages are python and JavaScript.

**Note:** not all programming languages are scripting languages but all scripting languages are known as programming languages.

1. **Dynamically typed language :**

* Variable: variables are the containers which is used to store some data. In JS to declare a variable we use three keywords:-

1. var
2. let
3. const

**Note :** there is no datatype declaration in dynamically typed language.

**Tokens:**

Token is the smallest unit in programming language. Tokens are classified into three types:

1. Identifier
2. Keywords
3. Literals

**Ex:** var x = 10;

where var is keyword

x is identifier

10 is literal

1. **Identifier:**

The name given a variable is known as identifier. It should not start with the number, no special characters are allowed except \_ and $.

Keywords cannot be used as an identifier.

1. **Keywords:**

Pre-defined words or reserved words which JS Engine can understand are called as keywords. Keywords are always written in lowercase.

**Ex**: var/let/const/function/null/typeof/switch/if/else/do/while/return/case/default/this/true/false/break/ continue…. Etc

1. **Literals:**

Literals are values or data which are passed to an variables / identifier.

**Data Types**

Data types are categorized into two types:

* Primitive datatype (single valued)
* Non-primitive datatype (multi valued)

1. **Primitive datatype:**

It stores single values. They are classified into:

* Number
* String
* Boolean
* Undefined
* Null

1. **Non-primitive datatype**

It stores multiple values. They are classified into:

* Array
* Object
* Function

1. **Number:**

Any mathematical number is considered as number datatype.

**Ex:** 1, 1.134, 100, -1, 9867669679

var x = 10;

Console.log(typeof 10) o/p: number

Console.log(typeof x) o/p: number

1. **String:**

Set of characters combined together is called as String. Characters must be enclosed using string datatype . characters can be enclosed in single quotes ‘ ‘ double quotes “ “ or backtick ` `

**Ex:** var msg = “Rahul”

Console.log(msg); o/p: Rahul

**Advantages of backtick:**

We can use single or double quotes inside the backtick. It will accept more than one line of string (template string or template literal). We can evaluate JS expressions using ${} in the backtick (String intercolation).

1. **Boolean:**

true or false keywords are considered as Boolean type.

**Note:** true -> 1

false -> 0

1. **Undefined:**

If the variable is declared without initialization, it is declared as undefined by JS Engine.

**Ex:** var str;

Console.log(str); // o/p: undefined

1. **Null:**

The return type of null is empty object.

**Ex:**  var obj = null;

Console.log(obj); //o/p: null

Console.log(typeof obj) // o/p: empty object

**Operators**

Operators are special keywords in JS which are used to perform operations between two operands. In JS operators are classified into four types :

* Arithmetic operators ( +, -, \*, /, %, \*\* )
* Comparison operators ( ==, >=, <=, <, >, !=, === )
* Assignment operators ( =, +=, -=, /=, %= )
* Logical operators ( &&, ||, ! )

**Conditional Statements**

* If
* If-else
* If-else if- else
* Nested if
* Switch

**Loops**

In looping we must have three things **initialization, condition** and **updation.**

* for
* while
* do-while

**Date() Methods**

The return type of Date() is string, hence it is immutable datatype before Date() we use a keyword “new” to make it mutable. “new” keyword represents an object.

**Ex:** typeof Date() // o/p: string (no modification)

typeof new Date() // o/p: object (can be modified)

**Syntax:** var/let/const varName = new Date();

**Methods:**

* toLocaleString()
* toLocaleDateString()
* toLocaleTimeString()
* getDay()
* getDate()
* getFullYear()
* getHours()
* getMinutes()
* getSeconds() and many more……

**Math Methods**

* PI
* SQRT\_2
* Max
* Min
* Random
* Round
* Floor
* Ceil
* Sqrt
* Trunc

**String Methods**

* length
* toLowerCase()
* toUpperCase()
* slice()
* indexOf()
* concat()
* includes()
* replace()
* replaceAll()
* padStart()
* padEnd()
* charAt()
* charCodeAt()
* codePointAt()
* at()
* trim()
* trimRight()
* trimLeft()
* match()
* matchAll()

**Dialogue boxes or Pop-up boxes:**

There are three in-built methods of browser, they are **prompt(), confirm(), and alert().**

1. **Prompt():** it is used for collecting data from end user. Return type is String.
2. **Confirm():** it is used to ask the confirmation from the end user. Return type is Boolean.
3. **Alert():** it is used to give a pop up with a msg. return type is undefined.

**Execution Process in JavaScript**

JS Engine is the translator for JavaScript where it converts js instructions(high level language) into machine understandable language. Every js has js engine by default.

Translator

Machine understandable language

High level language

Browser

JS

Engine

-

-

JS Instructions

-

-

The js engine which is used in google is V8. Browser itself acts as translator in web-development.

JavaScript is a synchronous, single threaded language, the meaning of synchronous is it can perform **only one task at a time.** The place where JavaScript code gets executes is known as **Call Stack**.

**Note:** We can achieve asynchronous JavaScript.

**Window Object**

When we execute js code inside the browser, js engine will create a brand-new global window object under the scope. The word window represents an object and it has a global scope.

In js, browser itself is the biggest object which gives all the super powers which is needed to a web development. browser can also be known as window. Its an browser object model (BOM).

Window is nothing but BOM which has all the super powers of browser such as location, built-in methods, storage etc… the variable declared by using var keyword will be implicitly added to global window object.

**Execution of JS instructions**

When we execute a JavaScript code JS engine will create a global execution context in a call stack. It consists of two phases:

1. Variable space
2. Execution area
3. **Variable space**

In this phase all the variable will be picked from top to bottom order and initialized by the value undefined to a variable.

1. **Execution area**

In this phase instructions will be executed line by line. Execution happens in top to bottom order. If there is any initialization it will remove undefined and it will initialize with the value to a variable in the variable space. If there is no initialization it remains undefined itself.

**Note:** JS codes are executed twice.

|  |  |
| --- | --- |
| **Variable Space** | **Execution Area** |
| a : ~~undefined~~  a : 10  b : ~~undefined~~  b : 20 | Start  a initialization  10  b initialization  20  End |

**Ex:**

Console.log(“start”);

Var a = 10;

Console.log(a);

Var b = 20;

Console.log(b);

Console.log(“end”);

**Steps to debug:**

1. Go to sources in dev tools
2. Go to pages you will get files
3. Open JS file and add a breakpoint by tapping the number
4. Refresh the browser
5. You will see Global inside the scope and Window object on the right side of global
6. You will also see anonymous inside the callstack
7. Check the phase, variable space in the scope
8. Use down arrow to execute and move to next line

**Note:** once all lines are executed remove the breakpoint and stop the debugger.

**Non-primitive datatypes**

We can store more than one value/data. It is classified into three types:

1. Array (symbol – [ ] )
2. Object ( symbol – { } )
3. Function (symbol – ( ) )
4. **Array**

Collection of values ais known as array.

**Syntax:** var/let/const arrName = [value1, value2, value2 , …….., value n]

In JS collection can be either homogeneous or heterogeneous. Homogeneous contains same datatype where as heterogeneous contains combination of datatype

**Note:** type of array is an object. Array is an mutable DT.

1. To update: arrName[index]
2. To add/update: arrName[index] = value
3. To delete delete arrName[index]

**Note:** index always starts with 0, length starts with 1. When we try to add the value more than the length it will create undefined empty holes. When we delete a value, it initializes undefined to it.

**Array methods:**

* Length
* Shift()
* Unshift()
* Push()
* Pop()
* Reverse()
* Indexof()
* Concat()
* toString()

**note:** concat(), join(), tostring() methods will not modify the original array. Tostring and join methods will convert array to string.

**Ex:**

Var str = “Rahul”

Console.log(str.split(“”).reverse().join(“”)) //o/p: luhar

1. **object**

objects are mutable data in JS hence we can perform, update, add, modify and delete operations. In object retrieving, updation and deletion can be performed using the keys of an object. Accessing keys can be done in two ways

1) dot notation . objName.key

2) square bracket notation [ ] objName[“key”]

**Note:** when we use [ ] notation, key must be enclosed using string.

**Object methods:**

* Object.keys( )
* Object.values( )
* Object.entries( )

**Type Casting**

Conversion of one datatype into another datatype is called as type casting. It is also called as type coercion. It has two types:

1. Implicit type casting (done by JS)
2. Explicit type casting (done by in-built methods)
3. **Explicit type casting**

In this type casting conversion of datatype Is done by the help of in-built methods.

**Ex:** split(), toString(), join() ….etc

**Number Method:** This is an in-built method which converts a value into a number Datatype.

**Ex:**

Number(true) //1

Number(false) //0

Number(“7”) //7

Number(“Rahul”) //NaN

1. **Implicit type casting**

In this type casting conversion of Dt is done by JS engine.

**Type casting with respect to + operator**

+ operator can be used for addition and concatenation. If both operands are number, it will perform addition, but if anu one of the operands is string it will perform concatenation.

**Note:** if one of the operands is string the other operand will implicitly get converted into String.

**Ex:** 2+3 //5 1 + true //2

“2”+”3” //23 “true” + 1 //true1

**Type casting with respect to all other operators**

In this string datatype will be converted implicitly to number datatype. If a converted string is not a valid number it will return a special number known as **“NaN” (Not a Number).**

**Ex:**  7 – “3” //4 1 – “true” //NaN

“7”\*”3” //21 true – 1 //0

**Note:**

* Two NaN can never be same.
* Any arithmetic operations performed with NaN or undefined it returns NaN.

**Boolean type casting with respect to logical operator**

True can be considered as 1 and false can be considered as 0. In JS false, 0, undefined, NaN, “” (empty string) are considered as falsy values. The values except above are considered as true values.

**Note:** a non empty string is considered as true value in JS.

**AND operator: (&&)**

Case 1: If LHS value is false it returns LHS

Case 2: if LHS value is true it returns RHS

**OR operator: ( | | )**

Case 1: If LHS value is false it returns RHS

Case 2: if RHS value is true it returns LHS

**NOT operator: ( ! )**

Inverts one to another

true => false false => true

**=**

**==**

**===**

It is an assignment operator which is used to assigning a value to a variable.

**Ex:**  var x = 10

It is a comparison operator which is used to compare two operands. (values)

It will compare only values not datatypes

**Ex:** true == “true” (true)

It is same as == but it is known as strict operator. (Datatype)

It will compare values and datatype.

**Ex:**  true === “true” (false)

**WAP to check given string is palindrome or not. Collect the input from user. Without using loops**

var str = prompt("Enter a string: ").toLowerCase();

var rev = str.split("").reverse().join("");

if(str===rev)

{

    console.log("It is palindrome");

}

else

{

    console.log("It is not a palindrome");

}

**WAP to check the no. of words given in a string, if the input is “my name is Rahul I am aspiring to become a javafullstack developer”. Without using loops**

var str1 = "my name is Rahul i am aspiring to become a javafullstack developer";

console.log(str1.split(" ").length);

**Difference B/W var vs let vs const**

|  |  |  |  |
| --- | --- | --- | --- |
| **Properties** | **Var** | **let** | **Const** |
| Scope { } | global | script | Script |
| Example | {  Var a = 10;  c.log(a); //10  }  c.log(a); //10 | {  let a = 10;  c.log(a); //10  }  c.log(a); //reference error | {  const a = 10;  c.log(a); //10  }  c.log(a); //reference error |
|  | When we create a scope for var keyword it has global scope it can be accessed within the scope or outside the scope | It has block scope. The variable declared can be accessed only within the block. | It has block scope. The variable declared can be accessed only within the block. |
|  | because global scope has the limit to access until the execution gets completed. | Once the execution of the block code is completed it will be automatically be removed from the scope. | Once the execution of the block code is completed it will be automatically be removed from the scope. |
| Declaration without initialization | Var a;  c.log(a); //undefined  a=10;  c.log(a); // 10 | let a;  a=10;  c.log(a); // 10 | const a; //syntax error  a=10;  c.log(a); |
|  | We can declare a variable without initialization | We can declare a variable without initialization | We cannot declare a variable without initialization. Declaration and initialization must be done in the same line. |
| Value modification | Var a = 30;  a=10;  c.log(a); // 10 | let a = 30;  a=10;  c.log(a); // 10 | const a = 30;  a=10; //type error  c.log(a); |
| Re-declaration of a variable | Var allows re-declaration of the same variable name. the most recently created variable will be used. | Let will not allow re-declaration and does not allow duplicates. | const will not allow re-declaration and does not allow duplicates. |
| Window | Var a = 10;  c.log(a); //10  c.log(window.a) ; //10 | let a = 10;  c.log(a); //10  c.log(window.a);//undefined | const a = 10;  c.log(a); //10  c.log(window.a);//undefined |
|  | Variable var belongs to window object. It has global scope so we can use window keyword to access the variable which is declared using var keyword. | We cannot access window because it has script scope. | We cannot access window because it has script scope. |
| Hoisting | Variable var is hoisted and it can be accessed before the code execution | Let keyword is also hoisted but cannot be accessed before because it belongs to temporal dead zone | const keyword is also hoisted but cannot be accessed before because it belongs to temporal dead zone | |
|  | c.log(a) //undefined  var a = 10; | c.log(b) //error  let b = 10; | c.log(c) //error  const c = 10; |

|  |  |  |  |
| --- | --- | --- | --- |
| **properties** | **var** | **let** | **const** |
| **Scope** | Global | block | Block |
| **Declare with initialization** | ✔️ | ✔️ | ❌ |
| **Value modification** | ✔️ | ✔️ | ❌ |
| **Redeclaration** | ✔️ | ❌ | ❌ |
| **Window** | ✔️ | ❌ | ❌ |
| **Hoisting** | ✔️ | ✔️ | ✔️ |

**Functions**

Functions are set of instruction which is used to perform some specific tasks.

**Advantages:**

* Code reusability, means we can create a function once and call it multiple times.

**Note:** function gets executed only when it is called.

We can create a function in two ways:

1. Function declaration statement.
2. Function expression.
3. **Function declaration statement**

**Syntax:**

Function funName (parameters)

{

//instructions

}

funName(arguments) //calling a function

when we declare a function we pass parameters, when we call a function we pass arguments.

**Note:**

* When a parameter is declared without an argument the result will be undefined.
* When there is no. of arguments is more than the no. of parameters it ignores the rest of the values.

1. **Function Expression**

Storing the whole function inside a variable is known as function expression. In this function variable name acts as a function call, there is no need of function name in this function.

**Syntax:**

Var/Let/const varName = function funName(parameters)

{

c.log(“I am a function expression”);

}

varName(arguments)

**Function execution context**

In an function execution context, when a function is invoked a new global execution system will be created in execution area.

**Ex:**

let square = function (num)

{

Let res = num\*num;

Return res;

}

Let square2 = square(2);

Let square4 = square(4);

**Scope of variables in functions**

The variable which is declared outside the scope will have global scope. The parameters and variable declared inside the function will have local scope.

**Hoisting**

It is a mechanism where variables and function declaration statements will be moved to the top of the scope before its code execution. There are two types:

* Variable hoisting
* Function hoisting

1. **Function hoisting**

Function declaration statements are hoisted whereas function expression are not. Hence declaration statements are hoisted so we can call the function before the code declaration.

1. **Variable hoisting**

In JS all the three variables are hoisted but let and const cannot be accessed before declaration because it belongs to temporal dead zone.

**Nested Functions**

A function within a function is called as nested function. Each and every time when a function is invoked a new execution context will be created in the execution area or function area.

**Types of functions:**

1. **Anonymous Function** : a function which is having no name is known as anonymous function. It must be created by using function expression.
2. **Immediate invoke function / self**-**invoking function :** calling the function without using function name is known as self-invoking function. In this function is invoked by itself and it can be called only once.
3. **Arrow Function :**

Creating function without using function keyword is known as arrow function. It can also be called as lambda function or fat arrow function. Arrow function will make our code more simple and more concise.

**Syntax:**

Var/let/const = ( ) => statement

Or

Var/let/const = ( ) => { statements }

**Note:** Arrow function can only be created by using function expression.

**Advantages:**

* For a single statement we don’t need of { } and return keyword
* Return is implicit here
* For a single parameter we don’t need a ( )

**Functional programming**

JS and python are the two-programming language which supports functional programming. If a function can accept another function as an argument and a function can be passed as an argument if this function types are supported it can be called as functional programming.

**Higher order functions**

A function which accepts another function as an argument is called as higher order function (HOF).

**Call back function**

A function which is passed as an argument to an higher order function is known as call back function (CBF).

**Syntax:**

HOF ( function CBF( ) { } )

**Timing functions in JS**

Timing functions are given by the window object.

* setTimeout ( ) : it executes the code only once, once the time is out.
* setInterval ( ) : it executes in a loop mode once after a given millisecond.
* clearInterval ( ) : it is used to stop setInterval ( ).

**Syntax:**

setTimeout( function( ) { } , milliseconds);

setInterval( function( ) { } , milliseconds);

clearInterval( setInterval variable name);

**Note:** setTimeout and setInterval are the HOFs. The function which is passed inside these functions are CBFs.

**Looping in JS**

1. **for in** : for in is used for iteration the index.
2. **for of** : for of is used for iterating the values.

**Note:** for in and for of can be used for both strings and arrays.

**Array iteration methods:**

It has 4 methods,

1. map( )
2. filter( )
3. reduce( )
4. sort( )
5. for each( )

the above 4 methods can be used only for array DT.

1. **foreach( )**

it is used to iterate each and every element in an index. The return type of foreach is undefined.

**Syntax:**  arrName.foreach(function (variable,index,orArr) {

} );

Instead of using function keyword we can use arrow functions which makes our code more simple and concise.

**Syntax:**  arrName.foreach( (variable,index,orArr) => {

} );

1. **map ( )**

same as foreach ( ), iterates each and every index but return type od map ( ) is new array.

**Syntax:**  arrName.map( (variable,index,orArr) => {

} );

1. **filter ( )**

it is used for filtering the values based on the given condition.

**Syntax:**  arrName.filter( (variable,index,orArr) => {

} );

1. **reduce( )**

Reduce method will returns a single value for the functions accumulated results. It iterates over each element in the array and each iteration return a single value which is called as accumulator

**Syntax:** arrayName.reduce((accumulator(inital val),item,index,array)=>{

},initalAccumulatorvalue(optional))

1. **sort ( )**

Sort methods sorts the elements as string in alphabetical and ascending order; It overrides the original array.

**Syntax:** sort( ) follows utf (ascii)

**Clousers**

Scope with respect to function. var/let/const variables declared before function;

-->GLOBAL SCOPE

parameters and variables declared inside function have local scope.

**Scope with respect to function**

The meaning of scope is , its a visibility area where we can access the members(variables). In function we have two scope

1. **Global scope** :The variable declared before the function will have global scope;
2. **Local Scope** : The variable declared inside the function and the parameters of the function will have local scope;

variable scoping : global/block/script;

function: global/local;

**De-structuring in JS**

De-structuring in JS is feature that allows to **unpack** the **values from arrays** or **properties from object** into distinct variables.

**Advantages:**

* It provides a concise way to extract the data from complex data.
* It makes the code more readable and maintainable.

**Note:** De-structuring does not modify the original array or objectit just copies the values into a new variable.

It is of two types,

1. Array de-structuring
2. Object de-structuring
3. **Array De-structuring**

Unpacking of values is done by the help of assigning an identifier to the particular index.

**Syntax:**  var/let/const [ ] = arrayName;

**Note:** to skip a value from an array we use comma ( , )

1. **Object De-structuring**

Unpacking is done by the help of keys of an object

**Syntax:** var/let/const { } = objectName;

**Note:**

* For an existing key we can’t modify the value
* If there is no index or key specify, we can assign a default value.

**REST and SPREAD operator**

The rest and spread operators in JS, both use same syntax ( … ) but they serve different purpose depending on the context in which they are used.

1. **Rest**

The rest operator is used to collect multiple elements and condense them into a single array or object. It is basically used in function parameter and de-structuring.

1. **Spread**

The spread operator is used to expand the elements of an iterable (like array or objects) into individual elements. It can be used for combining and copying arrays and objects.

**DOM (Document Object Model)**

DOM stands for document object Model. when we run the html code inside the browser the browser creates a DOM TREE. Javascript follows the features of REAL DOM. DOM is used for manipulating html elements. document is an object which is given by window object. document is nothing but the html file. Document has some in-built properties and methods. window.document===document //true(\*). In DOM the root node will be document.

**Dom methods:**

1. document.getElementById("IdName")
2. document.getElementsByClassName("ClassName")
3. document.getElementByTagName("tagName")
4. document.querySelector(".class/#id/tag")
5. document.querySelectorAll(".class/#id/tag")
6. **document.getElementById("IdName"):**

It manipulates the element by targeting its ID.

**Note** : if id is not present it returns null. it fetches only the first occurrence of an element.

1. **document.getElementsByClassName(“classname”)**

it manipulates the elements by targeting their class name. it fetches all the occurrence of the element. Return type is Array. Indexing is must.

1. **document.getElementByTagName(“tagName”)**

it manipulates the element by targeting its tag name it fetches all the occurrence of the element. Return type is array. Indexing is must.

1. **document.querySelector(".class/#id/tag")**

It can be manipulated by the help of tag, class or id. It fetches only the first occurrence of an element. If not present it returns null.

1. **Document.querySelectorAll(".class/#id/tag")**

Same as query selector, it fetches all the occurrences. return type is array. Indexing is must.

**Creating an element by dom:**

Once the element is created it must be appended by using a method appendChild(). Appending can be done to body or any html element. Properties to add a class is ClassName and for id is id.

variableName.ClassName="";

variableName.setAttribute("key","value")

**Events:**

In JS events can be added in 2 ways by using:

1. by using html attribute.
2. add eventListner.

click events, mouse events, keyboard events

**Mouse events:**

1. mouseover
2. mouseleave
3. mouseenter

**Keyboard events**:

1. key up
2. key down

**others**:

1. change
2. focus

when we use a html attribute as an event before event name on keyword is must.

addeventListner is an higher order function which accepts 2 arguments eventName and callback function.

**Syntax:** addEventListner("eventName",()=>{

log("hello")

})

**Asynchronous JS**

JS is synchronous single threaded language. Only one task can be done at a time. We can achieve asynchronous JS by the help of

1. callback
2. promise
3. async and await
4. **Callback**

A function which is passed as an argument to an higher order function is known as call back function.

There are some drawbacks of a callback function, they are

* **callback hell** : nested callback leads to callback hell. when we create a nested callback the codes gets increased in horizontal (pyramid of doom). this leads code difficult to debug, understand, maintain, read.
* **Inversion of control :** if any one nested callback fails the whole function will fails. We lose the control of the code this leads to inversion of control.

To overcome the drawback of callback function. They introduced promises.

1. **Promise**

Promise is an object which represents evential completion or failure of an asynchronous operation and its resulting value.

**Syntax:**

Let prom = new Promise((resolve, reject)=>{ });