**JAVA SCRIPT**

**JavaScript: -**

* Java script is a programming language and it is used to develop dynamic (changes in webpage) UI components.
* Using Java script we creating the UI components on demand.
* We can create Frontend, backend, and database applications by using java script.
* We can develop mobile app, web app, and desktop app using java script.

**Program: -**

It is a file it contains set of instructions.

**Programming: -**

It is the process of giving instructions to machine.

**Execution: -**

we translate the source code into machine code and then machine can read it and do something with it.

Or

It is the process of following the instructions to do some action.

**Variables: -**

Variables are Containers for Storing Data.

JavaScript Variables can be declared in 4 ways:

1. Automatically
2. Using var
3. Using let
4. Using const

**Syntax: -**

Syntax is a Grammer in programming language it contains set of rules.

**Keyword: -**

Keyword is reserved word in programming language.

**Datatypes: -**

JavaScript includes primitive and non-primitive data types. The primitive data types in JavaScript include string, number, boolean, undefined, null, and symbol. The non-primitive data type includes the object. A variable of primitive data type can contain only a single value. Non-primitive data type can store multiple data at a time.

**Primitive: -**

* It can store single data at a time.
* Primitive data types are the most basic data types. They are immutable, meaning their values cannot be changed once created.
* simple (easy to understand what we have to store in datatypes).
* no methods.

and

**Non-primitive: -**

* It can store multiple data at a time.
* Non-primitive data types are more complex data structures. They can hold collections of values and more complex entities. They are mutable, meaning their contents can be changed.
* Complex.
* It has methods.

**Function: -**

Function is a block where we can add instructions or statements or code to perform some actions.

Syntax: -

Function <FunctionName>(){

};

**Parameters: -**

* + Parameters are simple variables defined while the function defined inside the parenthesis () without scope statement.
  + Parameters are used hold or store the input data.
  + Parameters are simple variables.
  + While defining the function we passed the parameters.

Syntax: -

Function <FunctionName>(**parameters**){

}

**Arguments: -**

* + While calling the function we passing some values it is called arguments.
  + Arguments are actual data.
  + While calling the function we passed the arguments.

Syntax: -

Function <FunctionName>(parameters) {

}

<FunctionName>(Arguments);

**Return statement: -**

* + Using return statement, we can return the data from the function.
  + Return statement is the last statement inside the function because it terminates the execution of the function.

Or

In JavaScript, the "return" statement is used to specify the value that a function should output or "return" back to the caller. When a return statement is encountered within a function, the function will immediately stop executing, and the specified value will be sent back to the code that called the function.

Syntax: -

function <functionName>(){

return <variable>;

}

**Undefined: -**

* + Undefined is a special value.
  + Data type of this value also named as undefined.
  + Whenever function does not have any return statement, in that case it will always return undefined value (that means by default it prints undefined value).

**Types Of Functions: -**

**Anonymous Function: -**

* + - Function does not have name anonymous function.
    - Directly we cannot use or call anonymous function.

Syntax: -

function(){

}

**Function Expression**:

Function is assigned to variable is known as Function expression.

Syntax: -

Var value = Function(){

Console.log(“welcome”);

}

Value();

Or

var fn = function () {

  console.log("Function Expression");

};

var f1 = function add(p1) {

  console.log("add is called", p1);

};

fn();

f1(100);

**Arrow Function: -**

Function without function keyword and name is called arrow function.

Syntax: -

var myArrow = () => {

  console.log("Arrow Function");

};

myArrow();

**Callback Function: -**

Any function which is passed as an argument is called callback function.

Syntax: -

function fnCall(x){

    //console.log(x)

x();

}

fnCall (function (){

    console.log("call back function");//CALL BACK FUNCTION IN YELLOW COLOUR

});

**Higher Order Function: -**

Function which takes another function as an argument is called higher order function.

Syntax: -

function fnCall(x){

    //console.log(x)

x();

}

fnCall (function (){ //HIGHER ORDER FUNCTION IN YELLOW COLOUR

    console.log("call back function");

});

Or

Syntax: -

function f1(x) {}

function f2(x) {}

function f3(x, y, z) {}

f1(100);

f2(function () {});

f3(100, 200, function () {});

//f2,f3---Higher order function

**Object: -**

* + Object is a data structure (data structure is a technique it is used manage and store the data)
  + Object is a pattern or a technique is used to store and manage the data.
  + We use key: value pair is used to store and manage the data.
  + Object is a collection of properties.
  + Data type of object is object.
  + It is a non-primitive data type.

There are 3 ways to create object.

1. Literal notation
2. Class constructor(function) and new operator
3. Object constructor(function).

**Literal notation: -**

Var objName = {

}

Ex: - Var user={

Name : “manoj”,

Id : 12345

}

How to access specific property value in the object?

Var varName= objectName.keyName; //dot notation

Or

Var varName= objectName[“keyName”]; //bracket notaion

How to insert or add new property value in the object?

objectName.newKeyName = newValue;

Or

objectName[“newKeyName”]=newValue;

**Comments: -**

* + Comments are instruction but it is not a part of execution.
  + Comments are used to provide info. About the code to developer.

CRUD: - C-Create-Insert

R-Read-Access

U-Update-Modify

D-Delete-Remove

How to update property value?

objectName.keyName = “updated value”;

Or

objectName[“keyName”] = “updated value”;

How to delete the property of the value?

delete <objectName>.keyname;

**Note: -**

We can assign any type of value to the property.

Ex: -We can assign number, string, Boolean, and a function.

Var obj ={

Pid : function(){}

Pname : function myfunction(){}

}

**Console.log();** var console={ //console is an object

log: function(){----------} //log is a function

}

**Nested Object: -**

Object inside object.

Ex: - Var obj = {

name : {

firstName : manoj,

lastName : jami

}

}

**Array: -**

* + It is a collection of heterogenous data types.
  + Array is a Data Structure, used to store same or different types of data.
  + What we have store inside the array is called elements.
  + Array follows index values , index values always start with 0.
  + Represented in [].
  + Comma is used to separate the elements.
  + Data type of array is object.

How to create array?

Syntax: -

Var arr = [];

JavaScript is providing pre-defining functions to perform some actions?

1. **Push()**

The new element is added at the last position.

Syntax : -

<arrayName>.push(what we add data at the last position we passed in this);

1. **Unshift()**

It is used to add the new element at the initial position.

Syntax : -

<arrayName>.unshift(what we add data at the first position we passed in this);

1. **Pop()**

It is used to delete or remove the element at the last position.

Syntax : -

<arrayName>.pop();

1. **Shift()**

It is used to remove the first element in the array.

Syntax: -

<arrayName>.shift();

Slice: - it is used to access elements in the array.

Splice: - it is used to add or delete the element at specific position in the array.

1. **Includes()**

* It is used to check the element is present in the array or not.
* If the element is present in the array, it will print TRUE or else it will print FALSE.

Syntex:

Var value = <arrayname>.include(give the value);

1. **IndexOf()**

It is Used check the element position (element index value) in the array and also when the element is present or not in the array, if it is not presents it prints -1.

Syntex :

Var value = <arrayName>.indexOf(value)

1. **Slice: -**

It is used to access part of an array.

Syntax : - 1 + required value index+1

Var slice = <arrayName>.slice(from one index value to up to what index value we want);

EX:

Var array = [10,20,30,40,50];

Var value = array.slice(1, 4);

Console.log(value);

o/p: - [20,30,40]

1. **Splice : -**

It is used to add or remove any specific element from the array.

Syntax: -

<arrayName>.splice(starting no. pass where we want to remove, no. of elements remove,pass element what we add);

EX:

0 1 2 3 4------------------🡪index values

var arr = [10, 20, 30, 40, 50];

// arr.splice(2, 2); o/p:[10,20,50]

// arr.splice(2); o/p:[10,20]

//arr.splice(1,3); o/p:[10,50]

arr.splice(2, 0, "sagar", "Aditya");

console.log(arr);

o/p: [10,20,” sagar”,” aditya”,30,40,50]

How to access a specific value from the array?

By using index value

Syntax:

var element=<arrayName>[indexvalue];

1. **forEach: -**

It is used to access each and every element in the array.

Syntax : -

<arrayName>.forEach(function(element,index){

});

Ex: -

var users = ["yash", "siddharath", "yukti", "sai", "ramya", "shyam"];

users.forEach(function (element, index) {

  console.log(element + " reddy"); //yash, siddharath

});

1. **Map: -**

It is used to access each element of the array and do some actions on it.

It is used to return the result of the action in a new array.

Syntax : -

Var newarray=<arrayName>.map(function (element, index){

return

});

Ex: -

var users = ["yash", "siddharath", "yukti", "sai", "ramya", "shyam"];

var newArray = users.map(function (element, index) {

  return element + " Reddy";

});

console.log(newArray);

1. **lastIndexOf: -**

If there are duplicate elements in an array, In that array it will print the last element from the duplicates.

EX:

Var array = [12,34,67,98,54,67,**67**,78];

Var indexvalue = Array.lastIndexOf(67);

Console.log(indexvalue);

o/p—6

1. **filter: -**

* It is used to filter the array, based on some conditions.
* In this we return the Boolean values, if we give true it prints the elements, in case we return the false it will not print anything.
* In this we use comparison operators (<,>) in that case it prints true or false.

Syntax: -

Var new = arrayName.filter(function(){

return true or false;

});

Console.log(new);

1. **Array Length: -**

It is used to find the number of elements in the array.

Syntax: -

Var numberOfElements = <arrayName>.length;

**Operators: -**

1. Arithmetic
2. Logical
3. Comparison or Relational
4. Bit wise
5. Assignment
6. Special operators
7. typeof--------------🡪it tells the type of operator.
8. **Arithmetic Operator: -**

+ 🡪add

- 🡪subtraction

\* 🡪multiplication

% 🡪modules (division)-----it prints remainder as a result

++ 🡪increment

1. Pre-increment (++x)
2. Post-increment (x++)

-- 🡪decrement

1) Pre-decrement (--x)

2) Post-decrement (x--)

\*\* 🡪Exponential

**2) Comparison or Relational: -**

It is used to compare two values.

**< > <= >= != == ===**

Statement with comparison operator’s it will return Boolean values.

EX: -

var x = 10;

var y = 20;

var result = x > y;

console.log(result);

o/p: - false.

**3) Logical Operator’s: -**

It is used to compare the Boolean values.

AND &&

OR ||

NOT !

**AND && : -**

true && true 🡺true

false && true 🡺 false

true && false 🡺 false

false && false 🡺 false

**OR || : -**

true || true 🡺true

false || true 🡺 true

true || false 🡺 true

false || false 🡺 false

**NOT ! : -**

!true 🡺false

!false 🡺true

🡪What is Document Object Model (DOM)?

**It is a Structure of HTML documents where all the HTML Elements are converted into objects.**

HTML Documents (Elements or Tags)🡪JavaScript Objects🡪DOM Node or DOM Element.

🡪DOM Node or DOM Element: It is an object which is created based on HTML Elements.

🡪All The DOM Elements which are arranged in Sequentially in a tree kind of structures and that structure is called DOCUMENT OBJECT MODEL(DOM).

Why DOM is created?

To create dynamic user interface.

🡪User interface which can be created on-demand.

🡪Style of the UI which can be changed on-demand.

🡪Content of UI can be modified on-demand.

🡪We can ADD or REMOVE the elements from the web page.

What is DOM Manipulation?

It is a process to change the content or style of the element or add new element or remove existing element or modify the element on the web page.

Or

It is a process of modifying the DOM.

**Destructuring :-**

**Object: -**

It is a technique unpack the properties from the object.

It does not follow the sequence of key names.

**Syntax: -**

Var {keyname1,keyname2,keyname3} = <object-name>

**Array:** -

It is a technique unpack the elements from the array.

Sequence matter here.

**Syntax: -**

Const [\_,$, ,x,y] = <array-name>

**Promises: -**

Promises is a special object in the java script where we can store the information / Data.

Or

Promises is a special data structure where we can store and manage the data.

Data of promise is categorized into two types

1. Success data.
2. Failure / Error data.

**How to create a promise object?**

To create a promise, we use

1. new Operator
2. Promise() ---->pre-defined constructor

**Promise object syntax: -**

Var promiseObj = new Promise(function(resolve, reject){});

Console.log(promiseObj);

**How to insert or store data in the promise object?**

Using two pre-defined function called resolve() and reject(), we can store data in the promise object.

We can access resolve() and reject() using two parameters which are defined in the callback function of Promise() constructor.

Promise(function(resolve, reject){});

**How to add or store data in the promise object using resolve()?**

Var promiseObj = new Promise(function(resolve, reject){});

resolve(“manoj”)

Console.log(promiseObj);

**Note: -** whatever data passed as an argument to the resolve(), that data will be stored in the promise object.

We can pass any type of data in the promise object.

whatever data we store using resolve() in the promise object, that data is called Success data.

**How to add or store data in the promise object using reject()?**

Var promiseObj = new Promise(function(resolve, reject){});

reject(“manoj”)

Console.log(promiseObj);

**Note: -** whatever data passed as an argument to the reject(), that data will be stored in the promise object.

We can pass any type of data in the promise object.

Data stored in the promise object using reject() is called Error / Failure Data.

**There are three states of promise**

1. Pending state🡪Promise object is empty
2. Fullfilled / resolved state🡪Promise object contains success data
3. Rejected state🡪Promise object contains failure / error data

**How to access data of the promise?**

There are two ways to access

1. Using then() and catch() pre-defined function
2. Using async await

**How to call then() and catch() on promise object?**

promiseObject.then().catch()

if promise object contains success data, in that case then() will be called

if promise object contains failure / error data, in that case catch() will be called

promiseObject.then(function(){}).catch(function(){})

promiseObject.then(function(successdata){}).catch(function(errordata){})

**syntax: -**

promiseObj

.then(function (successdata) {

console.log("then() executed");

console.log(successdata);

})

.catch(function (errordata) {

console.log("catch() executed");

console.log(errordata);

});

**Note :-**

The process of accessing data from the promise object is called Handling the promise.

**Scopes in JavaScript: -**

Generally, we have 3 scopes, 4th scope

1. Global scope
2. Function scope
3. Module scope
4. Block scope

**What is scope in JavaScript?**

Scope refers to the accessibility

Using scope, we can define till what extent we can access variables or function of JavaScript once they are defined.

1. **Global scope**

Any area or space not present inside any block comes under global scope in JavaScript File.

Any variable or function which is defined in the global scope or global area such variables and functions you can access any where inside that file only.

1. **Function scope**

Any area or space inside any JavaScript function is called function scope.

1. **Block scope**

Any area or space inside any JavaScript block is called Block scope.

Ex: - if, else, function, switch

1. **Module scope**

It is used to access variables and function outside the file (In some other file).

There are two patterns to change the scope of variable to module scope.

1. Common JS module pattern
2. ES6 module pattern

**Difference between the let, var and const**

Var is called function scope, that means any variable defined with var is called function scope variable.

Let and const are called block scope.

1st diff.

Inside the JavaScript function, we can define any variable with var at any place inside the function and we can access that variable anywhere inside that function only.

Variables (let, const) or functions which defined inside a block will be accessible only in that block only.

2nd diff.

Var and let variables support re-assignment.

Const don’t support re-assignment.

3rd diff.

With var and let we can do variable declaration.

With const we cannot do variable declaration.

4th diff.

Shadowing is possible only var.

Shadowing is not possible with let and const.

Shadowing is a technique of re-declaration of variable.

**Events: -**

Events are the name of the actions that happens on browser.

Ex:

clicking the button (ACTION)🡪CLICK ACTION OR CLICK EVENT

Loading the html file in the browser (ACTION)🡪LOAD EVENT

Type some content in the text box (ACTION)🡪INPUT EVENT

Scrolling The page🡪SCROLL

Selecting radio, check box, drop down🡪CHANGE EVENT.

click,

mouseleave,

mouseenter,

mousemove,

keypress,

keydown,

keyup,

load,

scroll…etc.

NOTE: Once browser actions take place after that we can only perform our actions. To perform our actions, we have to write some js code.

NOTE: If you want to execute any js code when ever some events occurs then we have to catch or listen those events.

How to listen the events?

By using Event Listener

**Event listener: -**

They are used listen our event or track our event.

Event listener name will start with the word “on” Event name.

Ex: - onclick, onchange, onmouseenter….etc

We assign event handler to the event listeners (nothing but java script function we can perform some actions)

Event Handler:

To execute js code whenever event is triggered for that we have to configure event handler to the event listener.

Or

We have to assign event handler (nothing but java script function) as a value to the event listener.

Event listener =” event handler(java script function)”

**Event bubbling: -**

Event bubbling is a mechanism it calls event handler of current element (child function) after that parent element and then parent element and then parent element this sequence will follow, in this event will travels inner most to the outer most.

**Event propagation: -**

It is a mechanism of registering the events and to check the events will travels inner most to the outer most.

**Event capturing: -**

Vice versa of event bubbling (parent to child).

**How to event listener dynamically?**

By using DomElement.addEventListener()

**Syntax: -**

DomElement.addEventListener(“event-name”,handler,true/false)

What is Event?

🡪It is an action.

What is event listener?

🡪It is used to listen the Action

What is event handler?

🡪It is used to perform our actions.

Note: We can configure any number of events to the element