**JavaScript**

What is Scripting ?

>A script is program code that doesn’t need pre-processing (e.g. compiling) before being run. In

the context of a Web browser, scripting usually refers to program code written in JavaScript that

is executed by the browser when a page is loaded, or in response to an event triggered by the

user.

Client-side script:

>These scripts are getting executed within the web browser(client).

>Here we don't need any software.

>these scripts are used for client-side validatinos(data verification & data validations)

Ex: JavaScript, VBScript, typescript etc..

**Server-side script:**

>A script which executes in server machine with support of the web-server/app-server software’s like IIS(Internet information services), Tomcat, JBOSS, etc.

>These scripts are used for server-side validations (authentication & authorization).

Ex: php, jsp, asp.net, VueScript, Express Script, nodeJS, cgi, perl etc…

**JavaScript Introduction**

>In 1995 javaScript was created by a Netscape(Mozilla) developer named "Brendan Eich".

Mocha(1995) --> LiveScript -->JavaScript(1997-dec)

>Netscape first introduced a JavaScript interpreter in Navigator 2.

**Why is it called JavaScript ?**

--> WhenJavaScript 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.

JS ES

Later JavaScript became an ECMA (European Computer Manufacturers Association

Script) standard in 1997. ECMAScript is the official name of the language.

✔JavaScript is implementation of ES; ES is the specification of JavaScript.

RBI SBI, HDFC, ICICI customer

ES JS Programmer

--->JavaScript is a Speed, light weight, Interoperability, Extended Functionality, dynamic, loosely typed, cross platform, free ware and open-source.

Speed->ja applications runs faster than...

light weight->less code more operations

Interoperability-> JavaScript they have the capability to work within other web technologies

Extending Functionality->lib ->live

Dynamic typed->with out declaring vars we use can directly

loosely typed->defined any var in JS, that allows to store any type value

cross platform->cross platform compatible

>Its single threaded program

>JavaScript is an object-based or prototype-based programming.-Oops

>JavaScript is client-side(browser-side) programming. That means its executes on the browser.

>It can also be used in server-side by using Node, ASP, PHP

>JavaScript is a case sensitive program(mixed case)

>To work with JavaScript, we don't need to install any software.

>JavaScript is "Interpreter-based" programming, means the code will be converted into machine language line-by-line. JavaScript interpreter is already embedded in Browsers.

Parser:

JS code(high) <->JS parser <-> machine code

Js Engine:

V8-Chrome,Edge and Opera

spiderMonkey->Fire

**How many ways to imp js?**

>inline scripting

onclick, on submit,

<tag event="js code" event="js" event="js">

>internal scripting

<head>/<body>

<script type="text/javaScript">

js code

</script>

</head>/<body>

>external scripting

<script src="filename.js"></script>

**function definition syn:**

function fun-name()

{

Statements }

calling syn:

fun-name(); <-html inline or internal script

>JS->lib->collecection of predefined programs

reserved words,operations,functions,methods,classes and objects(implicit)

>object is an instance(dynamic memory block for the specified class) of class( copy of class)

>object is a collection of properties & methods

"window" is the base object for all JS objects

"window" object used to interacting with browser windows to perform some operations.

>it is implicit object

"document" is the sub object of window.

"document" object used for interacting with web page/web documents to perform some operations.

syn: window.document or documents

"console" is the sub object of window.

"console" object used for interacting with browsers console to perform some operations.

syn: window.console or console

press F12 key

**JavaScript Printing methods:**

write() method:The write() method writes HTML expressions or javaScript code to a document without line breaking.

syn: window.document.write(val1,val2,val3......);

writeln() method:The write() method writes HTML expressions or javaScript code to a document with line breaking.

syn: window.document.writeln(val1,val2,val3......);

log() method:The log() method writes HTML expressions or javaScript code on browser's console(press F12 key)with line break.

syn: window.console.log(val1,val2,val3...);

**some important points on javaScript**

>we can use all above methods without "window".

>there is no need of semicolon at the end of every statement because semicolon is optional.

>js is case sensitive.

>javaScript is always shows error but error shows in the console

>if error is there then it skips from error line up to script tag(type error)

>in some cases the total script tag is going to thrown out(Syntax error).

>you can't use direct html tags in javaScript but you can use with the help of literal

**literals**

#5 type of literals are in javaScript

->string literal("" and '')

-string literal is used to represent string/text.

-js1

-SL supports one line of text.

-not supports interpolation.

-"" or '' both are same in javaScript.

->template literal(``---backtick)

-string literal is used to represent string/text.

-since js6 (ES).

-TL supports N lines.

-it supports interpolation.

**Working with varibales**

>variable is a reference name fo a memory block.

>variables are created or stored in RAM (stack area).

>variables are used to store/to hold a value for reuse purpose and automatically substritute values in steps.

RAM

JS Runtime Area

>Stack Area (variables, params, Current Run code )

>Heap Area (array, objects, funs)

>Class/Method Area( class, constructor, static items )

>Register Area (sceduling the programs is happen in this area)

How to declare a variable ?

> we can define vars in js Three ways, those are:

>by using "var"

Syn: var varname; <-declaration

or

var varname=value; <-initialization

>by using "let"(since js6)

Syn: let varname;

or

let varname=value;

>by using "const"(since js6)

Syn: const varname=value; <-initialization

Where do we declare variables >

>We can declare varibles in open script tag(global), within function(local) or within block (block level)

**Rules for variables naming**

>Name should start with an alphabet (a to z or A to Z), underscrore(\_), or dollar( $) sign.

>After the first claracter we can use digits(0 to 9)

>Variables are case sensitive. For example , a and A are different variables.

>Space is not allowed, means name should be a single word.

>Special chars( symbol ) are not allowed in name, except \_and $.

**Rules for variable in javaScript**

>var let support modification const not support modification and you have to initialize at the time of declaration.

>var support redefinition but not let.

>var support hoisting but not let.

var let const

declaration Y Y N

assigning Y Y N

initialization Y Y Y

modification Y Y N

version Js1 ES(6) ES(6)

**difference b/w var and let**

**var let**

* >global/external Js >local, block level
* >support re-definition >not support
* >support var hoisting >not support
* >since js1 >since js6
* >sir suggest to use let

**declaration-allocating memory**

initialization-memory allocation and storing data at same time called initialization.

assigning- storing the data for already declared variable.

re-definition -declaration two

hoisting- use before declare after

loosely typed- if we declared any variable in js we can store any type of data.

**dynamically typed**

>if we don't declaration any variable we can use the variable

>it stores that variable in window object.

>by default it take the variable as "var" because it is present from first version of JavaScript.

**JavaScript Data type:**

In JavaScript data types are classified into the following two cat.

1.Primitive data type

2.Non-primitive data type

**Primitive data type Not- Primitive data type**

>PDT type allow storing data directly >reference/address

>PDT allow us to store only 1 value at >N values

a time

>stack area >Heap Area

>Not shareable >Shareable

>These are popularly known as non-reference >reference data type

>predefined data types >used defined data

>no new key word >new keyword is available(or [] ,{}) any of the three

**Primitive data types**

>>string,number,boolean,undefined,null(object)

**Non-Primitive data types**

>>Array,class&oject,functions

**Type of**

#by the use of typeof keyword you can know which type of data it is.

syn: typeof var-name

typeof value

const d = new Date();

>date class object created and initialize with default value and stores the address in d reference value.

**JavaScript Operators**

>Operator is a symbol (special char) and it is used to perform certain operation (task).

>Every operator is a symbol, but every symbol is not an operator.

>Every operator requires some values, those are called operands.

Ex: a+b

**Operators in JavaScript**

There are 3 types of operator in JavaScript

1.Unary -> it requires one operands

>increment ++

>decrement --

1.Binary -> it requires Two operands

>Arithmetic -> +,-,\*,/,%,\*\*

>Relational -> >,<,>=,<=,==,!= ,===,!==

>Logical -> &&,||,!

>Assignment -> =

>short hand -> +=,-=,\*=,/=,%=,\*\*=

>Concatenation -> +

1.Ternary -> it requires Three operands

>Conditional ? :

**MAIN Operator Precedence**

1) () -parenthesis

2) . -dot

3) [] -square bracket

4) pre ++ ,pre --

5) \*\* -power

6) \*,/,% -arithmetic

7) +,- -arithmetic

8) >,<, >= , <= - relational

9) ==, !=, === , !== - relational

10) = - assignment

11) post ++, post --

**parsing**

Changing the data mode from string format to number format

>types of parsing

1.Auto parsing(implicit)

All type subtraction, all type of **products**, all type of **division**, increment

2.Manual parsing(explicit)

a. using functions

i)parseInt();

predefined function of window,used string based int converts into integer format.

syn: window.parseInt("value")

"100" -> 100

"10.79"-> 10

"rama" -> NaN(Not a Number)

ii)parseFloat();

predefined function of window,used string based int converts into floating format.

syn: window.parseFloat("value")

"100" -> 100

"10.79"-> 10.79

"rama" -> NaN(Not a Numeric)

b. using parse operator

-> + is parse operator

-> Unary operator use only left side(prefix)

syn: +"value" or +variable

+"10" ->10

+"10.56" ->10.56

+"ram" ->NaN

**dialog boxes in JS**

>alert box -> used to display data to user

syn: window.alert(data)

>confirm box ->used to take the confirmation (cross check) from user.

syn:window.confirm(data)

>prompt box ->used to take the input from user .

syn: window.prompt(data)

**Events**

Onclick – on clicking something in webpage

oninput –at typing change will happen

**control statement**

Anonymous block

{

steps

}

if/else/for/while/do/addition

{

steps

}

if => true block

else =>false block

>if is decision making

>switch is selection

>switch support all the datatype. switch(day){}

>case can be variable also in javaScript.

**Loops**

-while loop (top testing/entry level)

-for loop

-do-while (bottom testing /exit level )

Initval - where the loop start/begins i=1

Condition - where the loop ends i<=10

Stepping - loop updater i++

**Unconditional statements**

>if condition is compulsory

break; stops the whole code (loops, switch )

continue; stops the current iteration (loops )

return; return from whole code

**String interpolation**

>string interpolation replaces the expressions in the string with actual values of the specified variables.

>substitute dynamically var/ expr/fun-call in between strings

>operator is ${}

>string should be enclosed with in " backtick"(``),but not"" and''.

syn: ${var} ${expr} ${fun call}

`text${var}text ${expr}text ${fun call}...`

use: while printing data & assigning values to variable.

**Arrays in JavaScript**

>array is collection of elements (values).

>storing group of value with same rename is called array.

>array allows similar type of values (homogeneous ) as well as different types of values, means one array can store group number , strings, Booleans etc ....

>we can create arrays in local scope or outer scope.

>arrays are belongs to reference/non-primitive data type.

>arrays are created dynamically, and arrays are created in heap area.

>arrays maintains data in sequence order

**adv:**

>arrays are simplifying coding when work with group of values.

>easy transporting data.

>also used for data maintenance in applications.

**array creation:**

Approach 1:

using array Literals[ ]

syn: let/var/const array=[]; //declaration

let/var/const array=[val1,val2,val3......]; //initialization

Approach 2:

using new keyword & constructor

syn: var/let/const array= new Array(); //declaration

var/let/const array=new Array(val1,val2,val3...); //initialization

accessing array:

array[index]

index is a slno of memory block, its start 0.

set value:

array[index]=value;

size of array:

array.length =>predefined property, its returns size of array

array.length=N; =>it reset size of array

**Error Handling in JavaScript**

> JavaScript supports the following list of error

syntax error

runtime error

logical error

**Exception handling in JavaScript**

to work with error/exception use try, catch, finally, throw keywords.

>try, catch and finally are blocks.

>throw statement.

>Try..catch statement: This statement allows you to test a block of code for errors. The try block contains the code to be run & the catch block contains the code to be executed if an error occurs.

Syntax:

try{

code to run[break;]

}

catch(ob){

code to run if an exception occurs[break;]

}

finally{

statements

}

**Possible combinations**

try-catch

try-finally

try-catch-finally these combinations are only possible

#List of Error:

Error name Description

EvalError An error has occurred in the eval() function(old var)->

RangeError A number "out of range" has occurred

ReferenceError An illegal reference has occurred

SyntaxError A syntax error has occurred <-

TypeError A type error has occurred

URIError An error in encodeURI("uri") has occurred

**Throw statement:**

This statement allows to you create an exception .if you use this statement together with try catch statement, you can control program low and generate accurate error message. The

>it is the custome error handling

Syn: throw excep-obj;

throw "text";

**Timer Functions**

setTimeout()

callback,timer,it exe task, only once after n mi.sec

var=window.setTimeout(task,"interval")

setInterval()

callback,timer,it exe task, repeatedly for every n mi.sec

var=window.setInterval(task,"interval")

clearTimeout()

it stops/clears the timeout function

var=window.clearTimeout(t-index)

clearInterval()

it stops/clears the interval

var=window.clearInerval(t-index)

**print()**

This method display print dialog box, which is used to print the current webpage/document through selected printer.

most of browser before printing page, they shows "Print preview"

Syn: window.print()

**Deployment**

These are some options where you can host you website for free.

* netlify
* w3 spaces
* early packages
* gitHub Pages

**Nested Array:**

* Arrays inside a array is called nested array.

const a1=[10,20,30];

const a2=[11,22,33];

cosnt a=[a1,a2];

document.write(`${a}<br>`);

// A simple nested array

let nestedArray = [

["apple", "banana", "cherry"],

["carrot", "lettuce", "pepper"],

["milk", "cheese", "yogurt"]

];

// Accessing elements in a nested array

console.log(nestedArray[0][1]); // Outputs: banana

console.log(nestedArray[2][0]); // Outputs: milk

a.length

a[0].lenght

a[1].lenght

**Array Methods:**

In JavaScript both array and string both are same. Except string manipulation.

Array support manipulation and string not support manipulation.

Ex of array and string

      const eles = [10, 20, 30, 40, 50, 30, 60, 70, 80, 30, 90, 30];

      let str = "green apples";

**Includes():**

* It shows whether that particular sub string or element present in that array /string or not.

   document.write(`30 found  ${eles.includes(30)}<br>`); //true

      document.write(`app found ${str.includes("app")}<br>`);//false

* It gives answer in true or false format

**IndexOf():**

* It finds the particular index of a element or character in a array/string .

document.write(`30 found @ ${eles.indexOf(30)}<br>`);//2

document.write(`66 found @ ${eles.indexOf(66)}<br>`);//-1

->the second parameter represents from where it will start search

document.write(`30 found @ ${eles.indexOf(30, 6)}<br>`);//9

document.write(`${str.indexOf("e")}<br>`);//2

**lastIndexOf():**

* It find the last occurrence of that element of char in array/string.

document.write(`30 last position @ last   ${eles.lastIndexOf(30)}<br>`);//11

document.write(`e last position ${str.lastIndexOf("e")}<br>`);//10

document.write(`30 second last @ 2 => ${eles.indexOf(30, eles.indexOf(30) + 1)}<br>`);

**reverse():**

* It reverse the array.

eles.reverse();

document.write(`reversed eles => ${eles}<br>`);

**sort():**

* It sorts the array In ascending order or alphabetical order.

eles.sort();

document.write(`sorted eles => ${eles}<br>`);

**join():**

* It joins the array elements by space or comma etc separator.

document.write(`${eles.join(" &nbsp; ")}<br>`);

document.write(`${names.join("&#128556;")}<br>`);

**Array Manipulation:**

**Splice():**

* This method helps to delete, update, and insert new elements to array.
* // deletion   splice(start-index ,count)
* let temp=eles.splice(2,2);
* document.write(`afeter delete ${eles}<br>`);
* document.write(`deleted data ${temp}<br>`);
* //updation    splice(st-index,count,new eles)
* //[10,20,50,60,70,80,90]
* eles.splice(3,2,101,202);
* document.write(`update ${eles}<br>`);
* //insertion   splice(st-index,0,new eles)
* //[10,20,50,101,202,80,90]
* eles.splice(5,0,103,204);
* document.write(`insert ${eles}<br>`);

Output:

eles 10,20,30,40,50,60,70,80,90  
afeter delete 10,20,50,60,70,80,90  
deleted data 30,40  
update 10,20,50,101,202,80,90  
insert 10,20,50,101,202,103,204,80,90

**slice():**

* This Methods helps to cut the array into pieces .
* const eles = [10, 20, 30, 40, 50, 60, 70, 80, 90];
* //slice
* //eles.slice(st-index,end-index);
* let a1=eles.slice(0,5);
* let a2=eles.slice(5);  //(5,9)
* document.write(`eles ${eles}<br>`);
* document.write(`array a1 ${a1}<br>`);
* document.write(`array a2 ${a2}<br>`);
* document.write(`last ele ${eles.slice(-1)}<br>`);
* document.write(`last 2 ele ${eles.slice(-2)}<br>`);
* document.write(`last 3 ele ${eles.slice(-3)}<br>`);

**For in loop:**

for(temp-var in array/coll){ //fetching index 1 by 1

steps a[temp-var];

}

>we can use array with in for in loop.

>for in loop supports to manipulate array.

>its index based loop.

const eles = [10, 20, 30, 40, 50];

for(let i in eles){

        document.write(`${i} =>${eles[i]} <br>`);

        // eles[i]+=5;

        // document.write(`${i} =>${eles[i]} <br>`);

        } //0,1,2,3,4

**For of loop:**

for(temp-var of array/coll){ //fetching elements 1 by 1

steps temp-var; }

 const b = [11,22,33,44,55];

      for(let e of b){

        document.write(`${e} <br>`);

       } //11,22,33,44,55

>we can't use array with in for of loop.

>for of loop not supports to manipulate array.

>its value based loop.

**Spread Operator:**

>"..." is unary operator, we should use this operator as prefix.

>Its rest and spread operator.

>the spread /rest operator represents all remaining values/so on values.

>this we can use in methods and arrays.

syn: ...array <-spread

...collection <-spread

>it mainly helps in merging and coping

# merging and delete the duplicate elements in array is important .

Copy syn:

const a = [10, 20, 30, 40, 50];

      //coping

      /\*let b=[];

      b=a;  //array not copied,ref only copid\*/

      let b=[...a]; //actual copy of a array into b.

Merging syn:

let ft=['html','css','js','bs'];

      let bt=['servlet','pht','django','asp.net'];

      let db=['oracle','mongodb','sqlserver'];

      let web=[...ft,...bt,...db];

      document.write(`web ${web} <br>`); //the above three array are merged into web array.

**Array Destructure:**

🡪By the help of array destructure we can store a array elements in separate variable or a new array in single statement.

const a = [10, 20, 30, 40, 50];

      //let x=a[0],y=a[1],z=a[2];  this is a long process

let [x,y,z]=a;    //destrureing

      document.write(`x ${x} <br> y ${y} <br> z ${z} <br>`);

     const eles=['html','css','Js','bs','ajax','jquery','angular','react','node'];

     let [i,,j,k,...e]=eles;    //destrureing with spread & skip

     document.write(`i ${i} <br> j ${j} <br> k ${k} <br>`);

      document.write(`e array values= ${e} <br>`);

swapping using distructure

//swaping using distructuring

      let v1=100,v2=200;

      document.write(`v1 ${v1} <br> v2 ${v2} <br>  <br>`);

      [v2,v1]=[v1,v2];    //swapping

      document.write(`v1 ${v1} <br> v2 ${v2} <br> `);

**#note** : push() function helps to pushing new element into array from last .

Focus() function helps to focus the mouse pointer at desired position.

**Functions In JavaScript**

**functions in JavaScript:**

🡪quite often we need to perform a similar action in many places of the script.

🡪for example, we need to show a message when a visitor logs in , logs out and maybe somewhere else.

🡪Functions are the main "building blocks" of the program. We've already seen examples of built-in functions, like alert(message),prompt(message, default) and confirm (question).

But we can create functions of our own as well.

function is named block; it consists group of statements

function is used to perform specific task/operation

**Adv**:

🡪Reusable means they allow the code to be called many times without repetition.

🡪reduce length of code.

🡪Easy maintenance code (readability, easy debugging and modification of code...)

🡪Generally 2 types

🡪predefined

🡪user-defined

**Types of functions:**

1.Named funs 2.Expressional Funs

3.Arrow funs 4.IE funs/II funs(immediate Execution / immediate invoke funs)

🡪We can develop functions either internal or external

Internal =>within the script tag

External =>in sep .js file, but no script tag(any no of funs)

**1.Named Functions:**

by using "function" keyword we can define /develop function.

syn:

function fun-name(parameters) //function prototype

{

local declaration

statements //body

return value;

}

function isPrime(n){

    let flag=true;

    for(let i=2;i<=n/2;i++){

        if(n%i==0)flag=false;

        break;

    }

    //return flag;

    return flag?`${n} is prime number`:`${n} is not a  prime number`;

}

//namedfuns.js

document.write(isPrime(5)+'<br>');//calling the function

**Where to call a function?**

we can call a function , from diff places, those are

>from script tag

>from another function

>event attribute

**How to calling:**

fun-name()

fun-name(arg1,arg2....)

#**Parameter :- we can pass arbitrary data to functions using parameters (also called function arguments).**

**🡪A function can return a value back into the calling code as the result.**

**syn: return var/value/expr;**

**syn: fun-name(param,param,....)**

**Note:**

**🡪 while declaring parameters don't use let,const and var keywords.**

**🡪A function or a method can return one value at a time.if needed to return more than one use array 🡪we should use 10(0-9) parameters maximum(suggested) if needed we can use array parameters like printf(), scanf() using array parameters**

**# parameters are used to pass the data from anywhere to anywhere in the program**

**2.expressional functions 3.Arrow functions**

**functions(params){ (params)=>{**

**body body**

**} }**

//Named Function

const add=function(x,y){

    return x+y;

    }

document.write(`${add(11,22)}<br>`)//calling

1.these are reference datatype

2.anonymos functions

3.these direct calling is not possible

4.using reference variable we can call

5.used in callback funs

**fun1 (code ,function(){})**

**fun1 (code ,()=>{})**

**writing a new function inside of other functions parameter is called callback function.**

**use->extending functionality**

**forEach() function**

**🡪introduced in js6 and not a loop, but extracting the elements one by one.**

**🡪used for arrays and collection objects**

**🡪it is a inbuilt function**

**🡪data collecting forEach method and printing exper function**

**Arrow functions (since JS6)**

1. No Parameter with 1step

Syn: () => statement

refvar = () => statement

2. No Parameter with no.of steps

Syn: () => { statements }

3. 1 Parameter with 1 step

Syn: param => statement

4. 1 Parameter with no.of steps

Syn: param => {statements }

5. No.of Parameter with 1step

Syn: (params) => statement

6. No.of Parameter with no.of steps

Syn: (params) => { statements }

Note: > without { } implicit return

**> with { } explicit return**

**> 1step no need { }**

**> 1param no need ()**

calling: refvar ===> it displays AF code on webpage

refvar() ===> it execute AF then displaying output

**4.Imediate invoking or execution function(IIF)**

An IIFE (Immediately Invoked Function) is a function that runs the moment it is invoked or called in the JavaScript event loop. Having a function that behaves that way can be useful in certain situations.

Adv:

1. IIFE does not create unnecessary global variables and functions.

2. Functions and variables created in IIFE do not conflict with other functions & variables even if they have the same name.

3. Organize JavaScript code.

Syn:

🡪 (function(){

document.write(`hi, I am II exper`);

}) ();

🡪 ( ()=>{

document.write(`hi, I am II arrow`);

}) ();

**Rest operator**

example on functions with rest operator

function fun-name(...param1, param2, param3) X

function fun-name(param1, ...param2, param3) X

function fun-name(param1, param2, ...param3) valid

function fun-name(...param)

{

code

}

fun-name(val1, val2, val3 );\

🡪minimum 0 and maximum no-range can taken by rest parameter.

🡪a single parameter can take any no of parameter.

🡪in function only one rest-parameter we can use. more than one is waste

🡪function(x,y,...n) it is allowed, but (...n,x,y) and (x,...n,y)not allowed.

**Shadow vars**

>to access the global variable after create shadow var (i.e variable with same name)

#we can use:window.var-name

#we can use:this.var-name("this" is reference variable which carries address of current object.)

**Object Oriented Programming in JavaScript**

We have two types of OOP language:

1.class-based Object-Oriented Programming

ex: java, .net, python , cpp etc.

1.prototype-based(object based) Object-Oriented Programming

ex: javaScript, typescript, perl, php etc.

🡪Object represents a physical component.

🡪Object is a real-time entity.

>we can see

>we can touch

>we can use

Ex: Para, button, heading textbox, div, table, li....

🡪Object is an instance of a class, nothing but memory block(one copy of class).

🡪"object" is a predefine class; every class/ object should be derived from "object" prototype.

🡪object is a collection of members:

1.properties (variable or fields)

2.methods (functions)

🡪**Properties:** details about the object. Properties are the variables which are stored inside the Object.

properties are used to store data about specific person, product or thing.

Ex: array.length=5

🡪**Methods:** to perform manipulation on the properties. Methods are the functions stored inside the object. Methods read values from properties, write values into properties, to perform logical operation.

Ex: array.sort() array.push() array.indexOf()

Note: object are used to data maintenance

**Array Object**

Sequence random

index base properties(key)

[] {}

const a=[] const ob={}

Example:

Car is an object Person is an object:

**-Properties -properties**

>car model:120 >name:siva

>car color:white >age:50

>car no:5579 >gen:male

-**Methods -Methods**

**Object literals**

>object literals are represented as curly braces{}, which can include properties and methods

>The property and values are separated with : symbol

>The method-name and body are separated with : symbol

syn: const refname={

"propertie":value,property:value,...,

methods

}

Note:

Methods we can write in any of 3 froms,i.e like normal function or expression or arrow

How to access?

refname.property <--to get value from object

refname.property=value <--to assigning value to object

refname.method-name() <--calling

Note: