**JAVASCRIPT:-**

Variables in javascript

We use variables for storing data in javascript we can use different keywords for defining a variable in javascript like

* Var (used to define the normal variable in the program)
* Let (used to define the normal variable in the program)
* Const (used to declare the constant variables in the program)

Some functions in javascript:-

Trim->removes all the spaces in the string

Slice-> use for indexing in the string

toUpperCase->used for converting all the characters in the string as uppercase

toLowerCase->Used for converting all the characters in the string as lowercase

Datatypes in javascript:-

Int ,string ,numbers ,Booleans ,undefined ,Null (BigInt ,Symbol newly introduced)

We use typeof(variable name) operator to know the type of the variable

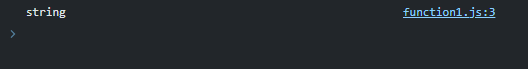
* Converting number to string

Method 1:

let age = 18;

console.log(typeof(age+""))

output



Method 2:

Typecasting

Let age = 18;

Age=String(age);

Console.log(typeof age);

Output:

string

* Converting string to a number

Method 1:

let name = "15";

console.log(typeof( +name));

output:



Method 2:

let name = “125”;

name = Number(name);

console.log(typeof name)

output:

number

STRING CONCATENATION:

By using “+” operator

let name="Mohammed",name1="Khimani";

console.log(name+ " " +name1);

output:

Mohammed Khimani

String formatting in javascript.

console.log(`This is ${name} age = ${age}`)

if the variable in js is not defined then the variable is type of undefined

var none;

let age;

console.log(typeof name,typeof age);

output:

undefined undefined

* Note if in constant the variable is not defined then it gives and error its must to define a constant variable with any value

Null:

Null is a data which denotes an empty variable which can be defined to any type later like

let name = null;

console.log(typeof(name))

output:

object

* There is bug in javascript that the typeof null is as an object but object and null are totally two different things .

BigInt:

It is a primitive datatype in javascript used to increase the range of the variable number

let age=152n

let incr=BigInt(12)

console.log(age+incr)

output: 164n

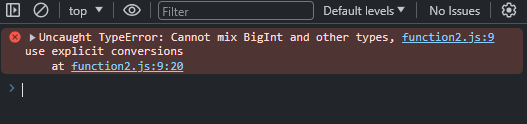
note: you can’t mix-up the normal number with an big int number like

let age=152

let incr=BigInt(12)

console.log(age+incr)

* it gives an error because age is in number and incr is in the bigint form



Boolean in JavaScript

True and false are used as Boolean in JavaScript

let num1=10 , num2 = 20;

console.log(num1<=num2)

output:

false

let num1=7,num2='7';

// == vs ===

console.log(num1 == num2)

console.log(num1 === num2)

here the difference in == and === is that the == does not compares the data type but the value but === also compares the datatype of the this feature is unique in javascript

same is with != and !==

if else,switch,ternary operator are same as c in JavaScript

basic if else program and here prompt is use to take the input from the user as a string

let num=+prompt("Enter the number")

if(num>10)

    {

        console.log("THE NUMBER IS GREATER THAN 10")

    }

else

{

    console.log("THE NUMBER IS LESSER THAN 10")

}

Loops in JavaScript

All the looping and non-looping statements in JavaScript are same as the c

Arrays:

Declaration of arrays in JavaScript.

let age=[1,2,3,4,5,6,7]

let name = ["asdf","asdf","banana","apple"];

console.log(name,age)

arrays are reference type in javascript and known as an object.

To check if the given name has an array then we should use Array.isArray(array\_name)

**PUSH AND POP OPERATION :-**

They change the values of the array and makes in mutable (changeable)

PUSH: ADDS THE ELEMENT IN THE LAST OF THE ARRAY

POP: REMOVES THE LAST ELEMENT OF THE ARRAY AND ALSO RETURNS THE REMOVED ELEMENT IF NEEDS TO BE STORED IN ANY VARIABLE

let age=[1,2,3,4,5,6,7];

let name = ["asdf","asdf","banana","apple"];

console.log(typeof name)

push and pop

name.push("HI");//adds element in the last of the array

console.log(name);

**SHIFT AND UNSHIFT:**

SHIFT: REMOVES THE FIRST ELEMENT OF THE ARRAY AND RETURNS THE REMOVED VALUE

UNSHIFT: ADDS THE ELEMENT TO FIRST OF THE ARRAY

let age=[1,2,3,4,5,6,7];

let name = ["asdf","asdf","banana","apple"];

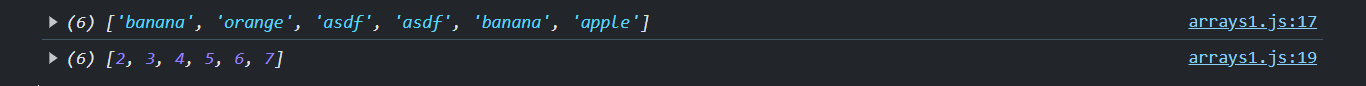
name.unshift("banana","orange");

console.log(name)

age.shift();

console.log(age);

OUTPUT:



Note:- pop and push operation are faster than the shift and unshift because the pop and push are the operation are done at the last of the array but in case of shift and unshift should iterate through the array till first element and also shift all the elements by one.

**Primitive vs reference datatype:-**

let num1=6;

let num2=num1;

num2++;

console.log(num1,num2);

here num1 is a primitive datatype when num2 is made equal to num1 and the changes made in num2 are not reflected to num1 these types of datatypes are known as primitive datatype

**Reference datatype :-**

let arr=[1,2,3,4,5,6,7],arr1=arr;

arr1.pop();

arr1.shift();

arr1[0]++;

console.log(arr);

We know that arrays are the reference datatype so when arr1 is made equal to arr

The changes made in arr1 will be reflected to the arr array.

let arr=[1,2,3,4,5,6,7],arr1=arr;

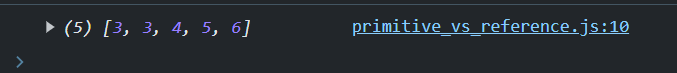
arr1.pop();

arr1.shift();

arr1[0]++;

console.log(arr);

OUTPUT:



**CLONING AND CONCATINATION IN THE ARRAY:-**

let arr=[1,2,3,5,4,5]

let arr1=[].concat(arr);

console.log(arr1);

Concatenates all the element of the array arr to arr1.

Output:-



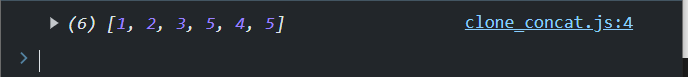
Using slicing method:

let arr=[1,2,3,5,4,5]

let arr1=arr.slice(0);

console.log(arr1);

OUTPUT:



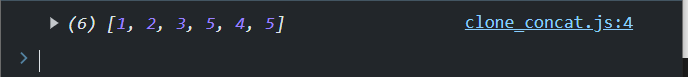
NEW WAY USING SPREAD OPERATOR:

let arr=[1,2,3,5,4,5]

let arr1=[...arr]

console.log(arr1);

OUTPUT:

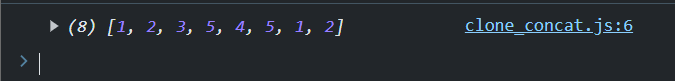


Concatenation of array:-

let arr1=[...arr].concat([1,2]);

console.log(arr1);

**OUTPUT:**

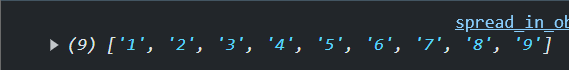


Another way to use spread operator:-

let arr=[...'123456789']

console.log(arr);

OUTPUT:-



**CONSTANT ARRAY:**

In javascript we can make a constant array with keyword const

This concept of

**ARRAY TRANSVERSE USING FOR LOOP:**

let fruits=['apple','banana','orange','grapes'];

for( let fort of fruits)

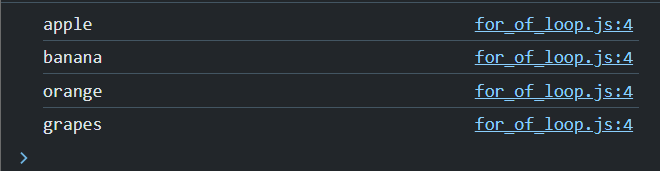
    {

        console.log(fort);

    }

Here the variable fort is first assigned the value of the first element till the last element.

Output:-



Array destructing:- assigning the values of the arrays to the variables is known as array destructing

let num=[5,1,2,444,6,6,4,8,2,44];

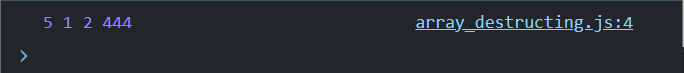
let [myarr1,asdf,f,g]=num;

console.log(myarr1,asdf,f,g);

here myarr1 is assigned the value of num[0] and asdf = num[1] and f = num[2] and g = num[3]

WE CAN ASLO USE THE VARIABLES OF DESTRUCTING AS THE NORMAL VARIABLES

OUTPUT:-



let num=[5,1,2,444,6,6,4,8,2,44];

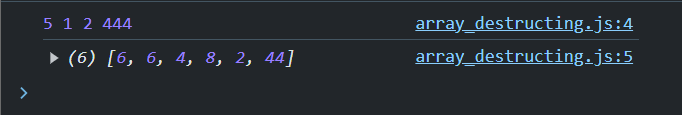
let [myarr1,asdf,f,g, ...NEWARR]=num;

console.log(myarr1,asdf,f,g);

console.log(NEWARR);

IF WE WANT TO ADD THE REMAINING ELEMENTS IN THE NEW ARRAY THEN WE USE SPREAD OPERATOR AFTER THE VARIABLES

Output:



OBJECT IS SAME AS THE STRUCTURES IN C (STRUCTURE VARAIBLES, JUST FOR BETTER UNDERSTANDING NOT EXACTLY)

const person={

    name: 'Aftaab',

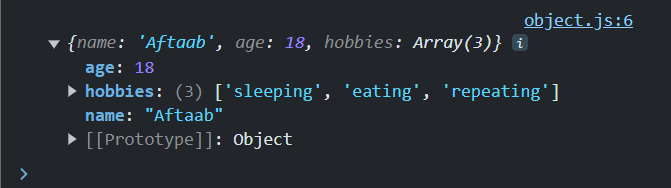
    age:18,

    hobbies:['sleeping','eating','repeating']

}

console.log(person)

output:



* Accessing elements of and object in JavaScript by their key values

const person={

    name: 'Aftaab',

    age:18,

    hobbies:['sleeping','eating','repeating']

}

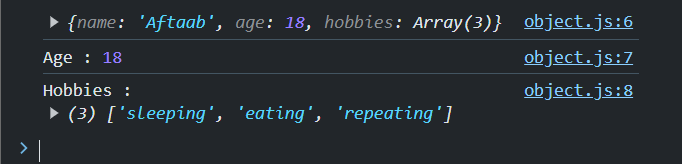
console.log("Age :",person["age"]);

console.log("Hobbies :",person.hobbies);

Note : here we should define the name of the variable in object as string in the square braces

We can also access the variables in an object by ‘.’ Operator

OUTPUT:



We can also use the name of the key in an object as a string

We can also add the key to the object using these braces or by dot operator

For eg

Person.gender= ‘Male’

Adds the new key to the object person.

* Difference between the ‘.’ and square braces
* We cannot access the key of an object which has a space in its name with ‘.’ Operator
* We should use braces there to access the element with space in its name

**Iterating an object:-**

We can iterate an object with several methods

Like using

* for in loop
* object.keys method

using for in loop:-

let person={

    name:"aftaab",

    age :18,

    work:'student'

}

for(let key in person)

    {

        console.log("OBJECT: ",key)

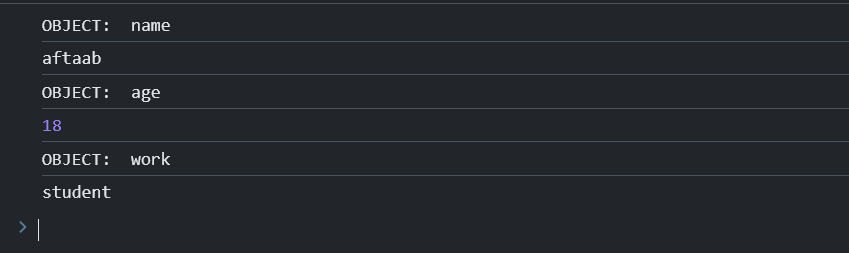
        // console.log(person.key)//wrong method

        console.log(person[key]);

    }

Here person is an object which is to be iterated we used a ‘for’ loop with ‘in’ keyword to access the keys in the object the compilation of the above code is explained below.

* In for loop ‘key’ is initialized by the keys of the object every time the value of key changes to next key until all the keys are accessed
* To access the elements of the object we use person[key] here the square braces computes the value of the key and prints the value on the console



Object.keys(object\_name) method

In this method the function returns an array of names of the keys in the object to access it we use for of loop

let person={

    name:"aftaab",

    age :18,

    work:'student'

}

console.log(Object.keys(person))

for(let i of Object.keys(person))

    {

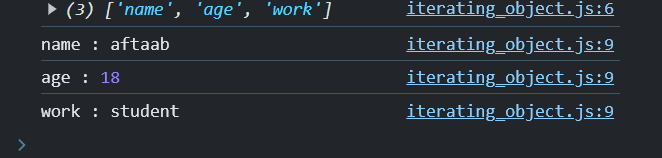
        console.log(i,":",person[i])

    }

Here the function returns the array which stores the names of the keys in an object

And then a for of loop to access the values of keys of it

OUTPUT :-



***Computed values ([]):-***

We use computed values to assign the value of the variable as the key of the object like const key1="objkey1",key2="objkey2";

const obj1='aftaab',obj2="Khimani";

let obj={

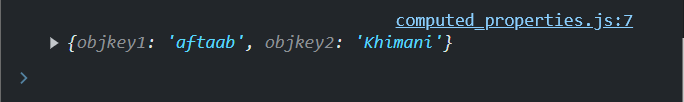
    [key1]:obj1,

    [key2]:obj2

}

console.log(obj);

here we can access the values of the keys by the strings assigned to the variable as first the values of the variable are computed then the values are assigned to it



Or we can use other way also to do same task

const key1="objkey1",key2="objkey2";

const obj1='aftaab',obj2="Khimani";

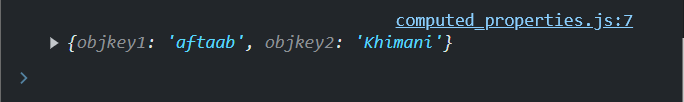
let obj={};

obj[key1]=obj1;

obj[key2]=obj2;

console.log(obj);

here first the string is assigned as the key and the element is added to the object by the string name



**SPREAD OPERATOR IN OBJECTS:-**

let obj={

    key1:"value1",

    key2:"value2",

    key3:"vlaue3"

};

let obj1={

    key1:"value4",

    key2:"value5",

    key3:"vlaue6"

};

//to make clone

let clone={ ...obj1};

//note here the keys are same of both the objects so the object which is lastly spreaded has the existence of its variables in it

let merged\_obj={...obj1,key87:'value87'};//adding key

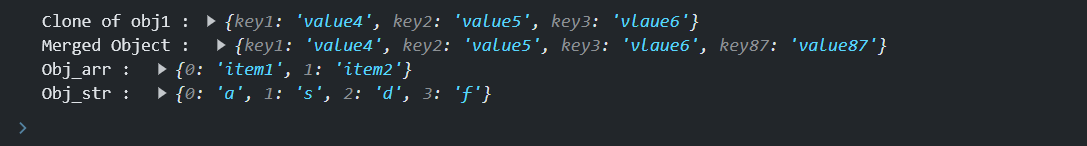
let obj\_arr={...["item1",'item2']}

// console.log(obj\_arr);

let obj\_str ={..."asdf"};

console.log('Clone of obj1 :',clone,'\nMerged Object : ',merged\_obj,'\nObj\_arr : ',obj\_arr,'\nObj\_str : ',obj\_str);

OUTPUT:-



**HOISTING:-**

The behavior of JavaScript in which the function declaration is done after the function call is called hoisting

Note:-

* We can’t use the hoisting in case of function variable and arrow function

(Later in detail)

Function inside the function :-

We can declare an function inside the function in JavaScript

function app()

{

    const func1= ()=>{

        console.log("This is func1");

    }

    const addTwo = (num1,num2)=>

        {

            return num1+num2;

        }

        console.log(addTwo(2,110));

        func1();

}

app();

when we call the outer function then it consecutively declares the function and then calls them for the results

lexical scope:-

if the function is declared inside the function then the lexical environment of the declared function is the parent function in which it was declared

Consider a case where variable var1 is used in the function and not declared in the function then js has the ability to check in it lexical environment of the function if it does not find the variable in the parent function then it checks in the lexical environment of parent function (that’s its global environment) and if further variable is not declared then gives the error

function myparent()

{

    let var1 = "HELLO WORLD";

    let print= () =>{

    //   let  var1 = 'hello world';

      console.log(var1)

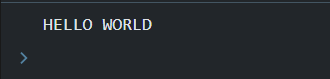
    }

    print();

}

myparent();

OUTPUT:



    function myparent()

    {

        let var1 = "HELLO WORLD";

        let print= () =>{

          let  var1 = 'hello world';

        console.log(var1)

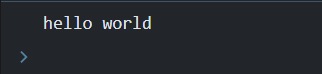
        }

        print();

    }

    myparent();

OUTPUT:



let var1="HeLLo wORLd"

function myparent()

{

    // let var1 = "HELLO WORLD";

    let print= () =>{

    //   let  var1 = 'hello world';

      console.log(var1)

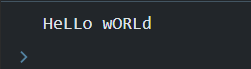
    }

    print();

}

myparent();

Output :



**FUNCTION SCOPE VS BLOCK SCOPE :-**

* Let and const have the block scope
* Var has the function scope

Block scope : The variables which are defined between the block {} and can only be accessed within it is known as block scope.

Function scope : The variable which can be accessed even outside the block are known as function scope and throughout the function.

function myfunc(){

    if(true)

    {

        var var1='Aftaab'

    }

    console.log(var1);

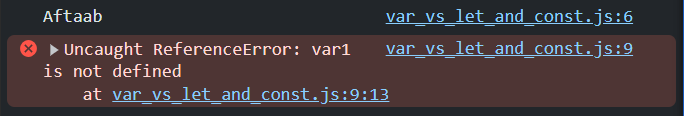
}

myfunc();

console.log(var1);

here we can see that var1 can be printed inside the function but not outside the function

OUTPUT:-



**Default parameters:-**

//  old method

//  function func(a,b)

// {

//     if( typeof b === "undefined")

//     {

//         b=0;

//     }

//     return a+b;

// }

// new method of default parameters

function func(a,b=2)

{

    return a+b;

}

console.log(func(45));

**Rest parameters:-**

There is a chance that the arguments passed to the function are more than the parameters of the function to store them in the function we use a variable using a spread operator to take all the parameters or remaining arguments as an array and doesn’t throw any error

Eg:-

function add(a,b,c,...d)

{

    console.log(a,b,c);

    for(let i=0;i<d.length;i++)

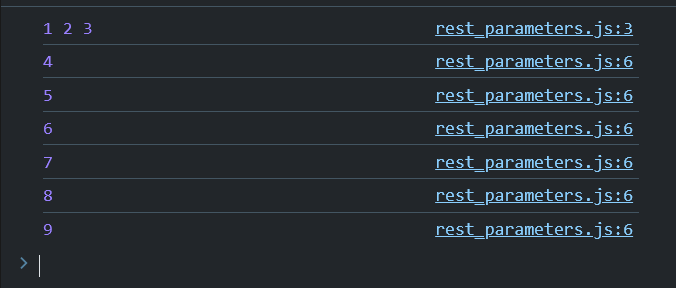
        {

            console.log(d[i]);

        }

}

add(1,2,3,4,5,6,7,8,9)



**Parameter destructing:-**

//param destructing

// objects and react

function des\_Obj({firstname, gender, age})

{

    console.log(firstname,gender,age);

}

let person={

    firstname:"Mohammed Aftaab Khimani",

    gender:'Male',

    age:18

}

des\_Obj(person);

function des\_Arr([var1,var2]){

    console.log(var1,var2);

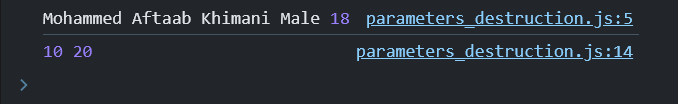
}

let arr=[10,20,30,50]

des\_Arr(arr);

in the above code we use to destruct the parameters of the object data type

OUTPUT:



**CALLBACK PROPERTY OF FUNCTIONS:-**

We can pass the function as the parameter in another function

Points to be remembered

* If we print the argument then we get the function printed
* If we call the argument (function) then it will do its job

Here func1 is passed as the parameter to func2.

function func1()

{

    console.log("This is func1");

}

function func2(callback)

{

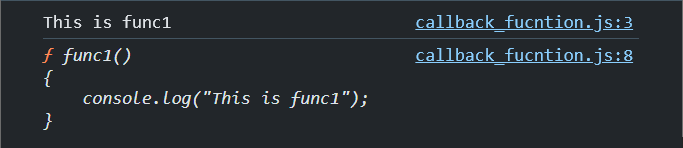
    callback();

    console.log(callback);

}

func2(func1);

Output:



Example 2:

function func1(a,b)

{

    return a+b;

}

function func2(callback)

{

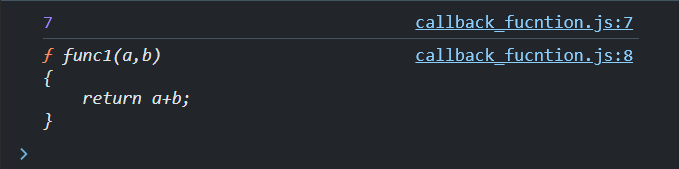
    console.log(callback(4,3));

    console.log(callback);

}

func2(func1);

Output:



**FUNCTION RETURNING A FUNCTION:**

We know that a function can return anything like array, strings, object and number etc.

Interesting part is we can also return a function in js

function func1()

{

    function hello()

    {

        console.log("Hello World");

    }

    return hello;

}

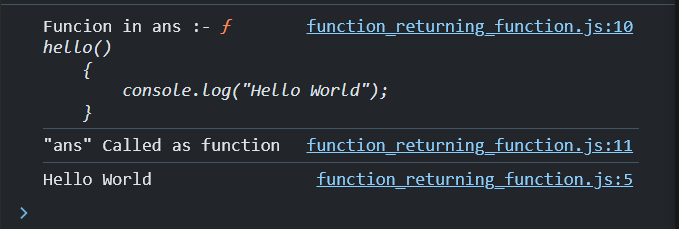
let ans=func1();

console.log("Funcion in ans :-",ans);

console.log("\"ans\" Called as function");

ans();

**OUTPUT:-**



**IMPORTANT FUNCTION FOR ARRAYS:-**

1. **FOR EACH**

This function is used iterate the elements of the array and pass them to a function or do the specific task it passes the elements and also the indexes of the first argument is as the element and the second argument is as index when the function is called.

**GENERAL SYNTAX OF FOR EACH (LOOP) :-**

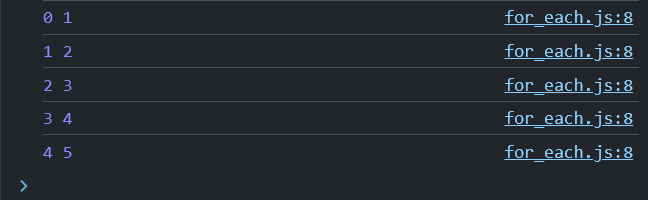
There are two ways of implementing the for each function

1. To define the var for element and index in the for each and use them in the inner function as defined

num.forEach((element,index) => {

    console.log(index,element);

});



1. Pass the array and the function to implement the function on the array

**Note:- The parameters of the function should be like (element,index).**

let num=[1,2,3,4,5]

function square(a,index)

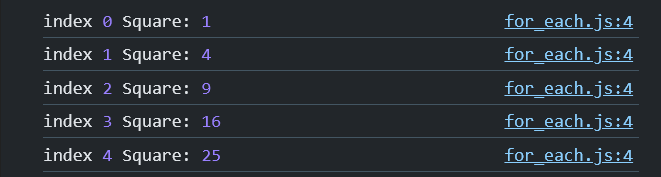
{

    console.log("index",index,"Square:",a\*a);

}

num.forEach(square);

OUTPUT:-



**MAP FUNCTION:-**

Map function iterates the array and passes it to the function then it creates a new array and stores the returned value in it in a sequence and if nothing is returned then by default it takes as undefined.

let num=[1,2,3,4,5,6];

const square=function(num)

{

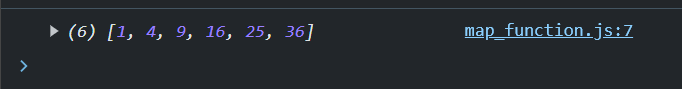
    return num\*num;

}

const sq\_arr= num.map(square);

console.log(sq\_arr);

OUPUT:-



Another way of implementation

let num=[1,2,3,4,5,6];

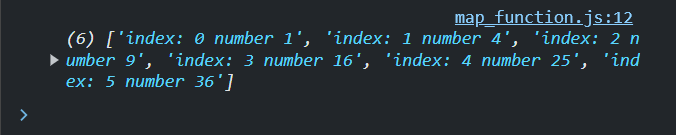
const sq\_arr=num.map((numbers,index)=>{

    return `index: ${index} number ${numbers\*numbers}`

})

console.log(sq\_arr);

output:-



**FILTER FUNCTION:-**

It filters the array with the function which returns an condition and makes an new array with the elements satisfying the condition.

const num = [1,2,3,55,6,,8,88];

const condition = function(number){

    return number%2!=0;

}

const filtered\_array= num.filter(condition);

console.log(filtered\_array);

output:-



Another method to use filter using arrow function

const filtered\_array=num.filter((num)=>{

    return num%2!=0;

})

console.log(filtered\_array);

const num = [1,2,3,55,6,,8,88];

OUTPUT:



**REDUCE FUNCTION:-**

const num=[1,2,3,4,55];

const sum =num.reduce((accumulator, currentValue)=>{

return accumulator+currentValue

});

console.log(sum);

output: 65

const num=[1,2,3,4,55];

const sum =num.reduce((accumulator, currentValue)=>{

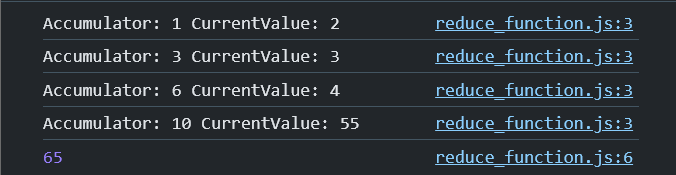
    console.log(`Accumulator: ${accumulator} CurrentValue: ${currentValue}`);

return accumulator+currentValue

});

console.log(sum);

Output:



Accumulator first takes the first element of the array and then the returned value of the function is the stored in accumulator and current value always shifts to the next element of the array till the last element is encountered.

Real life application of reduce:-

const usercart =[

    {productId: 1,pro:"mobile",amt: 50000},

    {productId: 2,pro: "TV",amt : 1000000},

    {productId: 3,pro: "mobile2",amt : 40000}

]

const totalprice = usercart.reduce((totalprice,usercart2)=>{

    return totalprice+usercart2.amt;

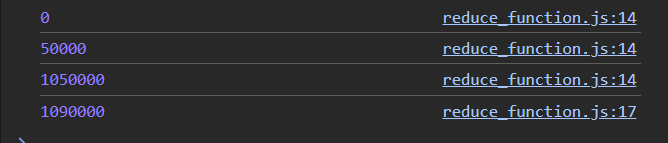
},0)

console.log(totalprice);

Output:-



ELEMENTS IN totalprice variable of the function



Here we can see that the value of totalprice is initialized with ‘0’ instead of the first object in the array therefore we can initialize the (accumulator) after the function is ended

**SORT() METHOD:-**

Sorting in js is like sorting of strings if we give the array of numbers it converts the numbers to strings and then sort it

* To sort the numbers we use a method for them

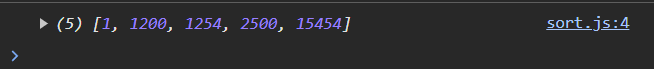
// sort

const num=[1,2500,1200,15454,1254];

num.sort((a,b)=>a-b);

console.log(num);

Output:-



// sort

const product=[

    {pro:"MOBILE", amt: 200000},

    {pro:"TV", amt:1000000},

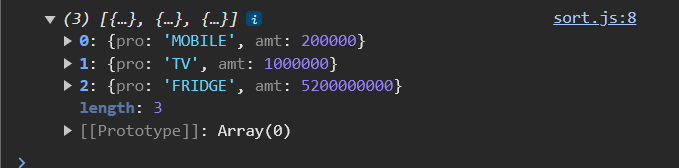
    {pro:"FRIDGE",amt:5200000000}

];

product.sort((a,b)=>a.amt-b.amt);

console.log(product);

Output:-



**FIND FUNCTION:-**

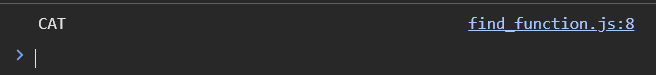
Find function finds the element in the array which satisfies the condition and then returns the first element to satisfy the condition.

const myArray = ["HELLO",'CAT','DOG','DFD']

const ans=myArray.find((str)=>str.length === 3)

console.log(ans)

Output:-



**EVERY FUNCTION:**

Checks if all the elements in the array satisfies the condition and returns true or false accordingly.

const num=[2,3,4,88,9,6,2,1];

const ans=num.every((num)=>num%2===0);

console.log(ans);

Output:-



**SOME FUNCTION:-**

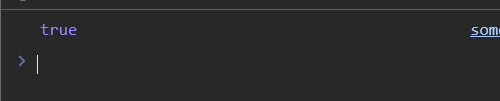
Checks if at least one of the element in the array satisfies the condition

const a= [2,5,8,9];

const ans = a.some((num)=>num%2===0);

console.log(ans);

Output:-



**FILL FUNCTION:-**

FILL function takes input as the value\_to\_be\_filled,start,end and fills the value in the array from starting index to ending index.

const n = [1,2,3,5,8,9,4,4];

console.log(`ORIGINAL ARRAY: ${n}`);

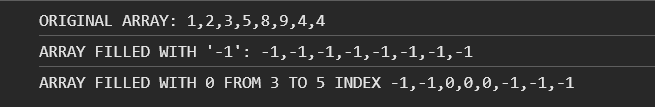
let a = n.fill(-1);

console.log(`ARRAY FILLED WITH '-1': ${a}`);

a=n.fill(0,2,5);

console.log(`ARRAY FILLED WITH 0 FROM 3 TO 5 INDEX ${a}`);

Output:-



**SPLICE FUNCTION:**

It has the ability to do both deletion and insertion in the array the general syntax is array\_name.splice(start\_index,delete\_index,insert\_items) it also returns the deleted items as an array.

***WORKING:***

1. **It deletes the items or the elements in the array from starting\_ index to (deleted\_index - 1) (if we want we can store it in any variable as an array)**
2. **The of the new elements also starts from the star\_index and ends at deleted\_index-1 of the array**

const arr =['item1','item2','item3'];

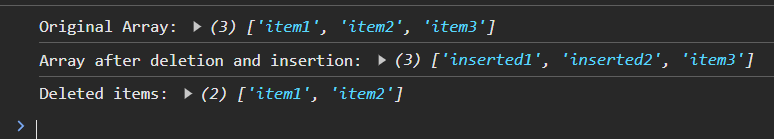
console.log("Original Array:",arr);

const del = arr.splice(0,2,"inserted1",'inserted2');

console.log("Array after deletion and insertion:",arr);

console.log("Deleted items:",del);

Output:



**ITERABLES:** Those datatypes for which we can use the for of loop or are known as iterables.

**ARRAY LIKE OBJECTS:** Those which has length property and also can be accessed by their index are known as array like objects.

**OBJECTS:**

* We use spread operator to clone the object in the other object
* Object.assign({}, object\_to\_added).

**SETS:**

* It is a iterable
* Stores data
* Sets have its own methods
* No index-based access
* Order is not guaranteed
* We use sets to handle the data which has unique values
* Only unique elements are stored (no duplicates are allowed they are ignored)

**Add () function:**

Adds the entry to the set

const items = ['item1','item2','item3'];

const sets = new Set();

sets.add(1);

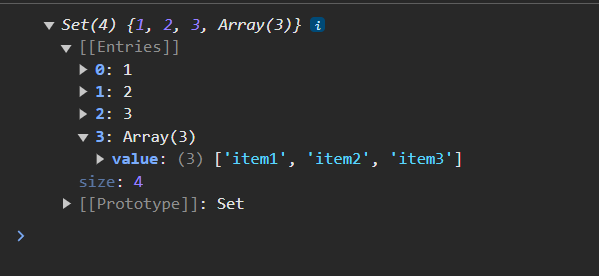
sets.add(2);

sets.add(3);

sets.add(items);

console.log(sets);

Output:



*Note: When we add directly array having same elements JavaScript takes them as the different arrays and adds them but when we add the same array then it gives an error*

Eg1:

const items = ['item1','item2','item3'];

const sets = new Set();

sets.add(1);

sets.add(2);

sets.add(3);

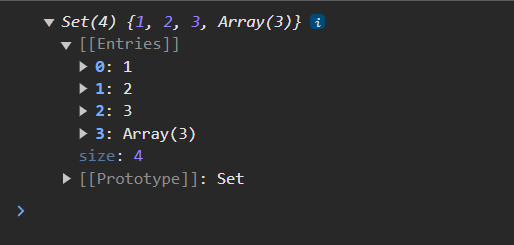
sets.add(items);

sets.add(items);

console.log(sets);

Here items array is added two times hence it is ignored.

Output:



Eg2:

const sets = new Set();

sets.add(1);

sets.add(2);

sets.add(3);

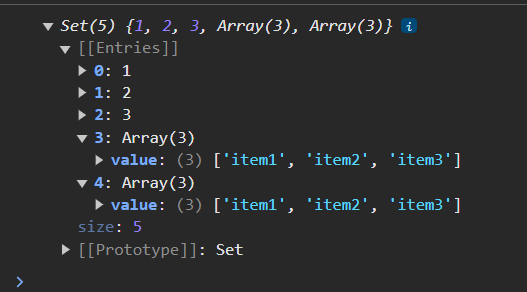
sets.add(['item1','item2','item3']);

sets.add(['item1','item2','item3']);

console.log(sets);

Here we can see that both the arrays are added to set.

Output:



Iteration of sets:

const items = ['item1','item2','item3'];

const sets = new Set();

sets.add(1);

sets.add(2);

sets.add(3);

sets.add(['item1','item2','item3']);

sets.add(['item1','item2','item3']);

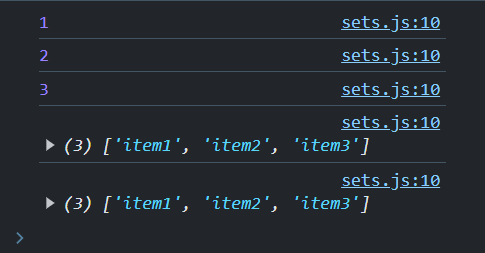
for(ele of sets)

    {

        console.log(ele);

    }

Output:



has() function:

has is the function used to check it element is present in the set it returns as true and if not returns false

const items = ['item1','item2','item3'];

const sets = new Set();

sets.add(1);

sets.add(2);

sets.add(3);

sets.add(['item1','item2','item3']);

sets.add(['item1','item2','item3']);

if(sets.has(3))

    {

        console.log("It has 3");

    }

Output:



Method used to find the unique elements in the array.

const arr2 = [1,1,2,2,3,3,4,5,6,7,88,9,9];

const unique\_elements = new Set(arr2);

console.log(unique\_elements);

**MAP (datatype):**

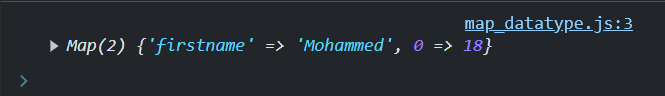
* We already know that the keys of the object are taken as a string.
* But in map we can keep key of any of the datatype like Numbers, string, array, etc.
* The items in the Map datatype uses ordered fashion.
* If you are willing to make a key to a variable make sure it is declared first in map datatype.

**DECLARING MAP VARIABLE:**

const person = new Map([['firstname', 'Mohammed'], ['age',18]],[[1,2,3],'array']);

console.log(person);

OUTPUT:



const person = new Map();

map\_name.set(key,value);

person.set('FIRSTNAME','Mohammed');

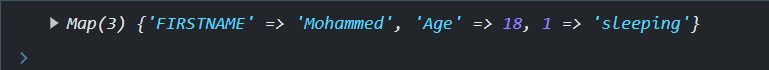
person.set('Age',18);

person.set(1,'sleeping');

console.log(person);

.set() is used to add the element to the map datatype

Output:



Checking type of the keys of Map datatype.

const person = new Map();

person.set('FIRSTNAME','Mohammed');

person.set('Age',18);

person.set(1,'sleeping');

// for of loop in map

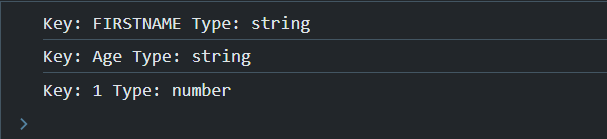
for (let key of person.keys())

    {

        console.log(`Key: ${key} Type: ${typeof key}`);

    }

Output:



Accessing the keys of Map:

Map\_name.get(key) is used to access the key of Map.

const person = new Map();

person.set('FIRSTNAME','Mohammed');

person.set('Age',18);

person.set(1,'sleeping');

person.set([1,0,2],'array');

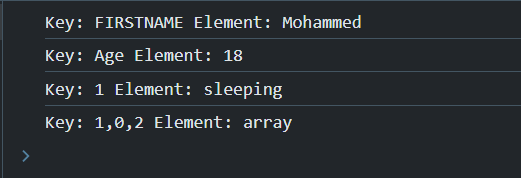
for(keys of person.keys())

    {

        console.log(`Key: ${keys} Element: ${person.get(keys)}`)

    }

Output:



const person = new Map();

person.set('FIRSTNAME','Mohammed');

person.set('Age',18);

person.set(1,'sleeping');

person.set([1,0,2],'array');

// accessing whole entry in map

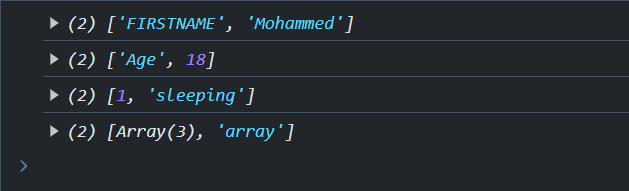
for(let entry of person)

    {

        console.log(entry);

    }

Output:



DESTRUCTURING THE ENTRY WHILE USING FOR OF LOOP:

const person = new Map();

person.set('FIRSTNAME','Mohammed');

person.set('Age',18);

person.set(1,'sleeping');

person.set([1,0,2],'array');

// accesing by destruction of entries

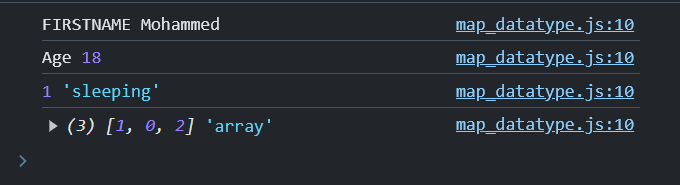
for([key,value] of person)

    {

        console.log(key,value);

    }

OUTPUT:



Consider a case where you want to store the extra data of the user made using the object but you don’t want to store in the users object

// REAL LIFE EXAMPLE

const people= {

    'ID':45,

    'NAME':'Mohammed'

}

const Info = new Map()

Info.set(people,{age:18,gender:'male'});

console.log(Info.get(people).age);

Output:



Here the object people act’s as a key to the object stored in the map

**Optional chaining:**

Consider an object that there is a key which does not existed then we if we access that key the console gives the error to reduce this error then we use optional chaining by *(?.)* it checks the if the object and the key is there or not and returns undefined if the object or the key doesn’t exist.

* Main use of optional chaining is in the nested objects
* It can be when the key currently does not exist but is going to be there after some function is done so we use this method

const obj = {

    key1 : 'hi',

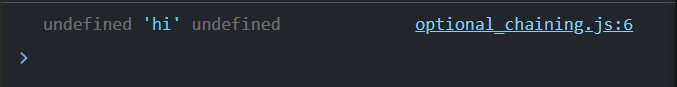
    key2:'hello'

};

let obc;

console.log(obj?.key,obj?.key1,obc?.asdf);

Output:



Methods: these are the function inside the object.

**METHODS**: