1. Variables
2. Functions
3. Arrays
4. JSON
5. Promises
6. DOM Manipulation
7. CallBacks
8. Events Handling
9. Event Bubbling and Event Capturing
10. Debouncing
11. Throttling
12. AJAX Calls
13. Closuers

JavaScript

Netscape company released the JS

<script> is used to include js in html

ES23 latest ECMA Script

-JavaScript is scripting lang

-JavaScript used to build the following applications

Angular React NodeJs Dynamic web pages

-We will save the js files with .js extension

-JS execution is synchronous execution(line by line)

-JavaScript is the object based scripting language.

Ex: Window,document,date,console..….

console.log(100+ +"100"); gives 200 inbuilt conversion with only one extra +

console.log(10+"10"+10);  //101010

console.log(10+ +"10"+10);  //30

console.log(10-"2") //8

console.log(10+"2"-"2") //102-2=100

console.log(10>9>8);    //10>9 ->true =1 ; 1>8 false o/p=false

console.log(10\*"10");   //100

console.log(10/"10");   //1

console.log(100/"0");   //infinity

console.log(100+ +"100"-"100"\*"100");   //-9800 BODMAS

== checks value

=== value and datatype

(1 == “1”) //true

(1 === “1”)//false not same date type

console.log(1 == "1"); //true same value

console.log(1 === "1"); //false not same date type

console.log(1=="one")//false not same value

console.log(1==="one")//false not same value and type

console.log(1+true);    //2 true=1

console.log(1-true);    //0 true=1

Variables

Variables are used to store the data

String,number,boolean,object

We can declare the variables by using “var”,”let”,”const”

Let and const keywords are introduced in es6

Variables should contain a-z, A-Z 0-9,$ and \_

Variables should not start with digits

Syntax

Var/let/const varibalename=value;

Numbers

1.decimal 2.float or double 3. Hexadecimal 4.ocatal 5.binay

Hexadecimal numbers will prefix with “0x”

Octal number will prefix with “0o” “0b” for binary

var decimalNum=100;

var doubleNum=100.113;

var hexadecimalNum=0x1321ABC;

var octalNum=0o31231;

var binaryNum=0b01010;

true=1;

false=0;

console.log(1+true);

console.log(true-false); //1-0=1

console.log("1"+true);//1tr

// ue

console.log(+"1"+true); //2

var x=true;

x==true?console.log("Hello"):console.log("Bye");

Strings

Collection of characters called as string

` backtick operator called as template literal in es6 and used to define paragraphs

Var sub=”JavaScript”

var sub\_one="ReactJs";

var sub\_two="NodeJs";

var stack=`${sub\_one}=>${sub\_two}`

console.log(stack);

var tb\_name="employees";

var sal=4000;

var sql=`select \* from ${tb\_name} where esal>${sal}`;

console.log(sql);

//undefined ->value will be there but it is of no use that is memory is occupied like bench people

//null->no value no memory like no job

Var x; //undefined

x1=null;

console.log(x1);

//bigint at the end keep "n" so that we can print big number also

var bigint=123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456123333333333333333333333333333333333333333332387654688765567887655678765567887654678763456n;

console.log(bigint);

console.log(typeof "hello");    //string

console.log(typeof 100);    //number

console.log(typeof true);   //boolean

console.log(typeof undefined);

console.log(typeof null); //object

console.log(typeof []); //object

//for loop

for(var i=0;i<5;i++){

}

console.log(i);//var:5 //let:error i is not defined

//var keyword breaks the scope rule

//let keyword obeys the scope rule

var sub="JavaScript";

var sub="Nikhil Thula";

console.log(sub);   //var:Nikhil Thula  //let gives error it wont allow duplicates

var and let and const

//global

var data=100;

//console.log(data);

//block

{

    //local

    var data=200;   //actually 200 should be accessible inside that block only

}

console.log(data);  //200

here var is not obeying its rule like we think var data =200 will be inside block but when we print data 200 is printing outside which is not wrong 100 should print because it is global variable. So use let which is scope specific. Let data=200; and for 100 also same.

If any block of code effecting global member called as global polluting issue

Global polluting issue raised bcz of var keyword

We can overcome global polluting issue by using keyword

//var msg;    //variable declaration

//var msg"Hello"  //variable intilization

var msg="Hello";

console.log(msg);

console.log(xyz);   //undefined with var but expected error

var xyz=600;        //so use let

//getting undefined before declaration and initialization called as variable hoisting

//we can overcome variable hoisting with the help of let keyword.

|  |  |
| --- | --- |
| var | let |
| ES1 | ES6 |
| Scope rule break | Scope rule follows |
| Duplicate members allows | Duplicate not allow |
| Global polluting issue | We can overcome |
| Variable hoisting raised | We can overcome |

//we cant modify complete array but we can modify inside its values by using const keyword

const arr=[1,2,3,4,5];

arr[0]=100;

arr[2]=300;

arr[5]=600; //extra added

console.log(arr);

arr.pop();  //remove last

console.log(arr);

arr.shift();

console.log(arr);

const obj={

    "one":"React",

    "two":"Angular",

    "three":"java"

};

//error due to const we can modifiy inside only

 /\*    obj={

} \*/

 obj.one="harish";

 obj.three="kiran";

 console.log(obj);

Functions

It is also called as business logic

Functions are used to reuse the business logic

We will define the functions with function keyword

We have following type of functions

1)Named functions

2)Anonymous Functions/ArrowFunctions/Callback functions/fat arrow functions/nameless

3)Rest Parameters in functions

4)Default parameters

5)Optional parameter

6)Generators

7)IIFE (Immediate Invokable functional expression)

8)constructor functions

//Named Functions

The function with the name is called named function

Syntax

1)Function Definition

2)Function Calling

1)Function Definition

Function functionname(arg1,arg2,arg3….argn){

Business logic

}

2)Function Calling

Functionname(parm1,parm2,parm3…paramn);

<script>

    function fun\_one(arg1,arg2){

        console.log(arg1,arg2);

    }

    fun\_one("vamshi","shiva");

    fun\_one("vamshi","shiva");

    fun\_one("vamshi","shiva");

    fun\_one();  //undefined

    fun\_one(null,null);

    fun\_one(undefined,"harish");

</script>

Check this imp

<script>

    function fun\_one(){

        return fun\_two;     //check no bracket thatwhy ()()

    }

    function fun\_two(){

        return "Hello";

    }

    console.log(fun\_one()()); //hello;

</script>

------------------------------------------

<script>

    let arr=[];

    function fun\_one(){

        return "Hello";

    }

    for(let i=0;i<5;i++){

        arr.push(fun\_one());

    }

    for(let j=0;j<arr.length;j++){

       console.log(arr[j]);

    }

</script>

<script>

    let arr=[];

    function fun\_one(){

        return "Hello";

    }

    for(let i=0;i<5;i++){

        arr.push(fun\_one);

    }

    for(let j=0;j<arr.length;j++){

        console.log(arr[j]());  //hello

    }

</script>

<script>

    function fun\_one(){

        return "Hello";

    }

    //it is executed only once that is setTimeout

    setTimeout(function(){

        console.log(fun\_one());

    },3000);

</script>

<!-- If we want to execute the function for every interval of time we need to use setInterval -->

<script>

    function fun\_one(){

        return "Hello";

    }

    setInterval(function(){

        console.log(fun\_one());

    },3000);

</script>

=================================

<script>

    for(let i=0;i<5;i++){

        setTimeout(function(){

            console.log(i);

        },3000);

    }

//var : 5 5 5 5 5

//let : 0 1 2 3 4

</script>

========================================================================

Rest parameters in Functions

It is used to store more than one value in single argument

… called spread operator

It is introduced in ES6.

1)We can take only one spread operator per function (…arg1) same as varargs in java

2)position of sread operator always last in occurances (arg1,...arg2)

<script>

    function fun\_one(...arg1){      //this ...is called "spread operator" Rest Parameter same as var agrs there we keep at the end

        console.log(arg1);           //we can take only one spread operator ...arg1...arg2 error

    };

    fun\_one("Hello1","Hello2");

    fun\_one("Hello1","Hello2","Hello3");

    fun\_one();

</script>

<script>

    function fun\_one(arg1,...arg2){

        console.log(arg1,arg2);

    };

    fun\_one();      //undefined[]

    fun\_one(undefined); //undefined[]

    fun\_one("Hello");   //Hello[]

    fun\_one("Hello1","Hello2","Hello3");//Hello ['Hello2','Hello3']

</script>

<script>

    function fun\_one(...arg1,arg2){     //error ...arg1 should be last parameter

    }

</script>

Default

<!-- Default parameters in function -->

<!-- While defining the functions, we will assign some default values to parameters. -->

 <script>

    function fun\_one(arg1="Hello1",arg2="Hello2"){

        console.log(arg1,arg2);

    }

    fun\_one();  //Hello1,Hello2

    fun\_one("ReactJS"); //ReactJS,Hello2

    fun\_one(undefined,undefined); //Hello1,Hello2

    fun\_one(null,null); //null,null

    fun\_one(undefined,"Welcome"); //Hello1,Welcome

 </script>

<script>

    function fun\_one(arg1,arg2="Hello2"){

        console.log(arg1,arg2);

    }

    fun\_one();  //undefined 'Hello2'

    fun\_one(undefined); //undefined'Hello2'

    fun\_one('Hello1'); //hello1,hello2

    fun\_one(null);//null,hello2

    fun\_one(undefined,undefined);   //undefined,hello2

    fun\_one(null,null); //null null

 </script>

<script>

    function fun\_one(arg1,arg2="Hello",...arg3){

        console.log(arg1,arg2,arg3);

    }

    fun\_one();      //undefined 'Hello' []

    fun\_one(undefined,undefined,undefined); //undefined,Hello,[undefined]

    fun\_one(null,null,null);    //null null [null]

    fun\_one("Hello1",undefined,"Hello3");

</script>

Arrow Functions

Arrow Functions:

The function without name called as Anonymous function.

Anonymous functions also called as arrow Functions

arrow functions behaves like callback function

we will represent arrow fucntion with =>

arrow function introduced in es6

syntax

1.function definition

2.function calling

var/let/const variablename=(arg1,arg2,arg3,....argn)=>{

    //bussiness logic

}

variable(para1,para2,.....paran);

 <script>

    let fun\_one=()=>{

        return "Hello";

    }

    console.log(fun\_one);   //fun def

    console.log(fun\_one()); //Hello

</script>

<script>

    let fun\_one=()=>"Hello";

    console.log(fun\_one());

</script>

<script>

    let fun\_one=(arg1,arg2,arg3)=>{

        console.log(arg1,arg2,arg3)

    }

    fun\_one("Hello1","Hello2","Hello3");//Hello1 Hello2 Hello3

    fun\_one();//undefined undefined undefined

    fun\_one(null,null,null);    //null null null

</script>

<script>

    let fun\_one=(arg1,arg2,arg3)=>{

        console.log(arg1(),arg2(),arg3());  //to get values bcz they are fun def we are calling arg1(),...

    }

    fun\_one(()=>{

        return "Hello1"

    },()=>{

        return "Hello2"

    },()=>{

        return "Hello3"         //Hello1 Hello2 Hello3

    });

</script>

<script>

    let intilize=(user,getElementsByEmail,getElementsById)=>{

        console.log(user,getElementsByEmail(),getElementsById());   //getElementsByEmail if write only this then it will only definition if we want o/p write getElementsByEmail()

    };

    let getElementsByEmail=()=>{

        return "hr@nikhl.com";

    }

    let getElementsById=()=>{

        return "Nikhil thula";

    }

    intilize("Hello1",getElementsByEmail,getElementsById);

</script>

<script>

    let arr=[];

    for(let i=0;i<5;i++){

        arr.push(()=>{

            return "Hello";

        })

    }

    for(let j=0;j<arr.length;j++){

        console.log(arr[j]());

    }

    //console.log(arr[0]());

</script>

IIFE

Immediate Invokable Functional Expression

es9

these functions called as self invokable Functions

syntax

((arg1,arg2,arg3,.....,argn)=>{

})(para1,para2,para3,.....paran);

<script>

    (()=>{      //Direct function calling

        console.log("Welcome to IIFE");

    })();

</script>

<script>

    let msg=(()=>{

        return "welcome to iife";

    })();

    console.log(msg);

</script>

<script>

    function fun\_one(arg1,arg2,arg3){

        console.log(arg1,arg2,arg3);

    }

    fun\_one((()=>{

        return "Hellooo"

    })());

</script>

<script>

    ((arg1,arg2)=>{

        console.log(arg1,arg2);

    })("vijay","ajay");

</script>

<script>

    for(let i=0;i<5;i++){

    setTimeout(()=>{

        console.log(i)

    },2000);

    }

</script>

constructor functions

-construcotr functions used to create the classes like structures before es6

-in constructor functions(variabels and functions) shoul start with this keyword

-this keyword representing the "current" class members

we will create the object with the help of "new" keyword

<!-- -It is same as class in java but use this keyword for variables and functions-->

<!-- <script>

    function class\_one(){

        this.sub\_one="react";

        this.sub\_two="angular";

    };

    let obj1=new class\_one();

    console.log(obj1.sub\_one);

    console.log(obj1.sub\_two);

     let obj2=new class\_one();

    console.log(obj2.sub\_one);

    console.log(obj2.sub\_two);

</script> -->

<script>

    function class\_one(){

        this.wish="JavaScript";

        this.my\_fun=function(){     //function using this keyword

            return `welcome to ${this.wish}`;

        };

    };

    let obj1=new class\_one();

    console.log(obj1.my\_fun()); //welcome to javascript

</script>

<script>

    function class\_one(){

        this.var\_one="Hello1";

        this.var\_two="Hello2";

        this.var\_three="Hello3";    //variables by this keyword

        this.fun\_one=function(){    //function  by this keyword

            return this.var\_one;

        }

        this.fun\_two=function(){

            return this.var\_two;

        }

        this.fun\_three=function(){

            return this.var\_three;

        }

    }

    let obj1=new class\_one();

    console.log(

    obj1.fun\_one(),

    obj1.fun\_two(),

    obj1.fun\_three());

</script>

<script>

    function class\_one(arg1,arg2,arg3){

        this.var\_one=arg1;

        this.var\_two=arg2;

        this.var\_three=arg3;

    }

    let obj1=new class\_one("Hello1","Hellooo2","Hello3");

    console.log(obj1.var\_one,obj1.var\_two,obj1.var\_three);  //Hello1 Hellooo2 Hello3

    let obj2=new class\_one("Kiran","vamshi","venkat");

    console.log(obj2.var\_one,obj2.var\_two,obj2.var\_three);  //Kiran vamshi venkat

</script>

<script>

    function class\_one(){

    };

    class\_one.prototype.var\_one="Hello1";

    let obj1=new class\_one();

    console.log(obj1.var\_one);

</script>

<script>

    function class\_one(){

    };

    class\_one.prototype.sub="JS";   //prototype is used to assign the values after the function declaring

    class\_one.prototype.fun\_one=function(){

        return "Hello";

    }

    let obj1=new class\_one();

    console.log(obj1.sub);

    console.log(obj1.fun\_one());

</script>

<script>

    function class\_one(){

    };

    class\_one.prototype.var\_one="Hello1";

    function class\_two(){

    };

    class\_two.prototype=Object.create(class\_one.prototype); //making classone as parent for classtwo with Object.create keyword

    class\_two.prototype.var\_two="Hello2";

    let obj1=new class\_two();   //here var\_one will get from parent class\_one

    console.log(obj1.var\_one,obj1.var\_two);

</script>

<script>

    function class\_one(){

    }

    class\_one.prototype.fun\_one=function(){

        return "Hello1";

    }

    function class\_two(){

    }

    class\_two.prototype=Object.create(class\_one.prototype);

    class\_two.prototype.fun\_two=function(){

        return "Hello2";

    }

    function class\_three(){

    }

    class\_three.prototype=Object.create(class\_two.prototype);

    class\_three.prototype.fun\_three=function(){

        return "Hello3";

    }

    let obj1=new class\_one();

    console.log(obj1.fun\_one());    //Hello1

    let obj2=new class\_two();

    console.log(obj2.fun\_one(),obj2.fun\_two());  //Hello1 Hello2

    let obj3=new class\_three();

    console.log(obj3.fun\_one(),obj3.fun\_two(),obj3.fun\_three());//Hello1 Hello2 Hello3

</script>

<script>

    function class\_one(){

    }

    class\_one.prototype.dbFun=function(){

        return "Data is from oracle db";

    }

    function class\_two(){

    }

    class\_two.prototype=Object.create(class\_one.prototype);//inheritance

    class\_two.prototype.dbFun=function(){

        return "Data is from Mongoodb";

    }

    let obj1=new class\_two();

    console.log(obj1.dbFun());

</script>

Generators

Generators

Generators controls the statement in Functions

Generator should have \*

Each statement should have yield statement

Generators will return cursor mechanism

by using cursors we can control statement

here \* is used in functions then they are generators

ex: line 1—1 st execution

line 2 --4th

line 3 –2 nd

line 4—5th

line 5—3rd

like this by generators we can control flow

<script>

    function \*fun\_one(){

        yield "stmt-1";

        yield "stmt-2";

        yield "stmt-3";

        yield "stmt-4";

    }

    let cursor=fun\_one();   //cursor position is above fun\_one();

    console.log(cursor.next()); //goes line by line

    console.log(cursor.next());

    console.log(cursor.next());

    console.log(cursor.next());

    console.log(cursor.next());

</script>

<!-- o/p

{value: 'stmt-1', done: false}  //stmt one say false bcz behind me there is one more st2

{value: 'stmt-2', done: false}  //stmt one say false bcz behind me there is one more st3

{value: 'stmt-3', done: false}  //stmt one say false bcz behind me there is one more st4

{value: 'stmt-4', done: false}  .....

{value: undefined, done: true}  .....  returns true no behind stmt-->

<script>

    function \*fun\_one(){

        yield 100;

        yield 200;

        return 300;

        yield 400;

    }

    let cursor=fun\_one();

    console.log(cursor.next());

    console.log(cursor.next());

    console.log(cursor.next());

    console.log(cursor.next());

</script>

<!-- o/p

{value: 100, done: false}

{value: 200, done: false}

{value: 300, done: true}

{value: undefined, done: true} 400 not executed due to return-->

<script>

    function \*fun\_one(){

        yield 100;

        yield \*fun\_two();

        yield \*fun\_three();

        yield 400;

    }

    function \*fun\_two(){

        yield 300;

    }

    function \*fun\_three(){

        yield 200;

    }

    let cursor=fun\_one();

    for(let i=0;i<5;i++){

        console.log(cursor.next());

    }

</script>

Optional parameter

-while calling the functions we will make few parameters as Optional

-this concept introduced in es6

-optional parameters we will represent with ?

-optional parameters will work in Typescript

-supersert of javascript called as Typescript

-we will save typescritp files with .ts extension

/\* ? means optional not compulsary and void means return nothing rem syntax

this optional option is not there in js so we need to use this ts

by command-> tsc optional.ts it will automatically create optional.js \*/

function fun\_one(arg1?:string,arg2?:string,arg3?:string):void{

    console.log(arg1,arg2,arg3);

}

fun\_one();          //undefined undefined undefined

fun\_one("Hello1");  //Hello1 undefined undefined

fun\_one(undefined,undefined,"Hello3");  //undefined,undefined,Hello3

fun\_one(null,null,null); //null null null

/\* function fun\_one(arg1:string,arg2?:string,arg3:string="Hello",...arg4:string[]):void{

} \*/

/\* any is also datatype here \*/

/\* op1 is mandatory and op2 is static and op3 is optional and react args rem this order \*/

/\* function fun\_one(op1:any,op2:any="Helllooo",op3?:any,...op4:any[]):void{

    console.log(op1,op2,op3,op4);

}

//fun\_one();  // Expected at least 1 arguments, but got 0.

fun\_one("Hello1");//Hello1 Helllooo undefined []

fun\_one("Hello1","Hello2","Hello3","Hello4");//Hello1 Hello2 Hello3 [ 'Hello4' ]

fun\_one(undefined,undefined,undefined,undefined); \*/

CallBacks

<!-- passing one function def to another function argument is called callback -->

<script>

    function fun\_one(arg1){

        console.log(arg1);

        console.log(arg1());

    }

    function fun\_two(){

        return "Welcome to callbacks";

    }

    fun\_one(fun\_two);

</script>

<!-- o/p

1)fun\_two(){

        return "Welcome to callbacks";

}

2)Welcome to callbacks -->

<script>

    function fun\_one(arg1,arg2,arg3){

        console.log(arg1(),arg2(),arg3());  //Hello1 Hello2 Hello3

    };

    fun\_one(function fun\_two(){

        return "Hello1";

    },function fun\_three(){

        return "Hello2";

    },function fun\_four(){

        return "Hello3";

    });

</script>

<script>

    function fun\_one(arg1){

        return arg1("Hello");

    };

    fun\_one(function fun\_two(arg1){

        console.log(arg1);

    });

</script>

<script>

    function fun\_one(arg1,arg2,arg3){

        return arg1("Hello1")+"......"+arg2("Hello2")+"....."+arg3("Hello3");

    }

    fun\_one(function fun\_two(arg1){

        console.log(arg1);

    },function fun\_three(arg1){

        console.log(arg1);

    },function fun\_four(arg1){

        console.log(arg1);

    });

</script>

<script>

    function fun\_one(arg1){

        return arg1(fun\_three);

    };

    function fun\_two(arg2){

        return arg2("Hello");

    }

    function fun\_three(arg3){

        console.log(arg3);

    }

    fun\_one(fun\_two); //first this

</script>

<script>

    function calc(num,add){ //add() it is addition function

        return add(num+5,false);

    }

    function addition(addRes,error){

        if(!error){

           console.log(addRes);

        }

    }

    calc(5,addition);   //jhere addition is def and it has two paras so use add(arg1,arg2) use two params

</script>

<script>

    function fun\_one(arg1){

        console.log(arg1());

    };

    function fun\_two(){

        return "Hello";

    }

    fun\_one(fun\_two)

</script>

<script>

    function fun\_one(arg1,arg2,arg3){

        console.log(arg1());

        console.log(arg2());

        console.log(arg3());

    };

    function fun\_two(){

        return "Hello1";

    }

    function fun\_three(){

        return "Hello2";

    }

    function fun\_four(){

        return "Hello3";

    }

    fun\_one(fun\_two,fun\_three,fun\_four);

</script>

<script>

    function fun\_one(arg1){

        console.log(arg1());

    };

    fun\_one(function fun\_two(){

        return "Hello";

    });

</script>

<script>

    function fun\_one(arg1){

        return arg1("Helloooo");

    };

    fun\_one(function fun\_two(param1){

        console.log(param1);

    });

</script>

<script>

    function fun\_one(arg1,arg2,arg3){

        return arg1("Hello1")+"...."+arg2("Hello2")+"....."+arg3("Hello3");

    };

    fun\_one(function fun\_two(para1){

        console.log(para1);

    },function fun\_two(para2){

        console.log(para2);

    },function fun\_three(para3){

        console.log(para3);

    });

</script>

<script>

    function fun\_one(posRes,errorRes){

        return posRes("sucess")+"...."+errorRes("error");

    };

    fun\_one((para1)=>{

        console.log(para1);

    },(para2)=>{

        console.log(para2);

    })

</script>

<!-- Example for callback hell -->

<script>

    let cal=(num,add)=>{

        return add(num+5,false);

    };

    let sub=(num,callback)=>{

        return callback(num-3,false);

    };

    let mul=(num,callback)=>{

        return callback(num\*2,false);

    }

    let div=(num,callback)=>{

        return callback(num/2,false);

    }

    cal(5,(addRes,error)=>{

        if(!error){

            sub(addRes,(subRes,error)=>{

                if(!error){

                    mul(subRes,(mulRes,error)=>{

                        if(!error){

                            div(mulRes,(divRes,error)=>{

                                 console.log(divRes);

                            })

                        }

                    })

                }

            })

        }

    })

</script>

callback hell is not recommended and need to use promises for that

JSON

JSON stands for JavaScript Object Notation

Json also called as JavaScript objects

We can exhange the data in fullstack application in two ways

xml and JSON

xml is heavy weight

json is light weight

Json data suggested whie transferring the data

syntax

------

Objects---{}

Arrays---[]

data ---key&value pairs

        key & value sepaarated by using ":"

<script>

    let obj={

        "sub\_one":"ReactJs",

        "sub\_two":"NodeJS",

        "sub\_three":"MongoDB"

    }

    console.log(obj.sub\_one,

        obj.sub\_two,

        obj.sub\_three

    )

</script>

<script>

    let obj={

        "obj1":{

            "key1":"val1",

            "key11":"val11"

        },

        "obj2":{

            "key2":"val2",

            "key22":"val22"

        },

        "obj3":{

            "key3":"val3",

            "key33":"val33"

        }

    }

    console.log(obj.obj1);

    console.log(obj.obj2);

    console.log(obj.obj3);

    console.log(obj.obj1.key1);

    console.log(obj.obj1.key11);

    console.log(obj.obj2.key2);

    console.log(obj.obj2.key22);

    console.log(obj.obj3.key3);

    console.log(obj.obj3.key33);

</script>

<script>

    let obj={

        "key1":"Hello\_1",

        "key2":"Hello\_2",

        "key3":"Hello\_3"

    }

    //iterate the json object

    //ES6

    //for in loop

    for(let k in obj){

        console.log(k);   //key1 key2 key3

        console.log(typeof k);    //string string string rem imp

        console.log(obj[k]);

    }

</script>

<script>

    let obj={

        "obj1":{

        "key1":"Hello\_1",

        "key2":"Hello\_2",

        "key3":"Hello\_3"

        }

    }

    //iterate the json object

    //ES6

    //for in loop

    for(let key in obj.obj1){

        console.log(obj.obj1[key]);

    }

</script>

<script>

    function dbFun(){

        return "Hello";

    }

    console.log(dbFun());

</script>

<script>

    let obj={

        "oracle":oracle,

        "mysql":mysql,

        "mongodb":mongodb,

        "casandara":casandara()

    }

    function oracle(){

        return "oracle connection soon!..";

    }

    function mysql(){

        return "mysql connection soon!";

    }

    function mongodb(){

        return "mongodb connection soon!";

    }

    function casandara(){

        return "casandara connection soon!";

    }

    console.log(obj.oracle(),obj.mysql(),obj.mongodb(),obj.casandara);

</script>

<script>

    let obj={

        "key1":()=>{

            return "Hello\_1";

        },

        "key2":()=>{

            return "Hello\_2";

        },

        "key3":()=>{

            return "Hello\_3";

        }

    };

     console.log(obj.key1(),    //Hello\_1 Hello\_2 Hello\_3

     obj.key2(),

     obj.key3());

</script>

With the help of Object.keys we can get keys -->

 with the help of Object.values we can get values  -->

<script>

    let obj={

        "key1":"value1",

        "key2":"value2",

        "key3":"value3"

    }

    console.log(Object.keys(obj));  //['key1', 'key2', 'key3']

    console.log(Object.values(obj)); // ['value1', 'value2', 'value3']

</script>

<script>

    let obj1={

        "key1":"Hello1"

    };

    let obj2={

        ...obj1    //here we making the copy of obj1 by ...obj1

    };

    console.log(obj1);  //{key1: 'Hello1'}

    console.log(obj2);  //{key1: 'Hello1'}

</script>

<!-- here we making the copy of obj1 by ...obj1 i.e spread operator -->

<script>

    //deep copy

    let obj1={

        "key1":"Hello1"

    };

    let obj2={

        ...obj1    //here we making the copy of obj1 by ...obj1

    };

    obj1.key1="Original Copy";

    obj2.key1="Xerxo copy";

    console.log(obj1);  //{key1: 'Hello1'}

    console.log(obj2);  //{key1: 'Hello1'}

</script>

<script>

    let obj={

        "key1":"Hello1",

        "key2":"Hello2",

        "key3":"Hello3"

    };

    console.log(obj);   //{key1: 'Hello1', key2: 'Hello2', key3: 'Hello3'}

    delete obj.key3;

    console.log(obj);

</script>

<script>

    let obj1={

        "key1":"Hello1"

    };

    let obj2={

        "key2":"Hello2"

    };

    let obj3={

        "key3":"Hello3"

    };

    let obj4={

        ...obj1,

        ...obj2,

        ...obj3

    }

    console.log(obj4);

</script>

<!-- Adding keys dynamically -->

<script>

    let obj1={

    };

    obj1.key1="Hellooo1";

    obj1.key2="Hellooo2";

    obj1.key3="Helloo3"

    console.log(obj1);  //{key1: 'Hellooo1', key2: 'Hellooo2', key3: 'Helloo3'}key1: "Hellooo1"key2: "Hellooo2"key3: "Helloo3"[[Prototype]]: Object

</script>

<script>

    let obj={

        "key1":"Hello1",

        "key2":"Hello2",

        "key3":"Hello3"

    };

    obj.key1="welcome1";

    obj.key2="welcome2";

    obj.key3="welcome3";

    console.log(obj);

</script>

Object.freeze is used to only the read data we can not modify any thing.

<script>

    let obj={

        "key1":"Hello1",

        "key2":"Hello2",

        "key3":"Hello3"

    };

    Object.freeze(obj);

    //read operation

    console.log(obj);

    //add

    obj.key4="Hello4";

    console.log(obj);

    //update

    obj.key1="Nikhil Raj";

    console.log(obj);

    //delete

    delete obj.key1;

    console.log(obj);

</script>

Object.seal is used for reading and updation of json data

<script>

    let obj={

        "key1":"Hello1",

        "key2":"Hello2",

        "key3":"Hello3"

    };

    Object.seal(obj);

    //read

    console.log(obj);   //{key1: 'Hello1', key2: 'Hello2', key3: 'Hello3'}

    //add

    obj.key4="Hello4";

    console.log(obj);   //{key1: 'Hello1', key2: 'Hello2', key3: 'Hello3'}

    //update

    obj.key1="Hi Raj";

    console.log(obj);   //{key1: 'Hi Raj', key2: 'Hello2', key3: 'Hello3'}

    //delete

    delete obj.key1;

    console.log(obj);   //{key1: 'Hi Raj', key2: 'Hello2', key3: 'Hello3'}

</script>

 freeze                                      seal

read allow                                  read allow

write not allow                             write not allow

update not allow                            update allowed

delete not allow                            delete not allowed

<script>

    let obj={

        "key1":"Hellooo1"

    }

    Object.defineProperty(obj,"key2",

        {

            value:"Helloo2",

            writable:false

        }

    );

    console.log(obj);   //{key1: 'Hellooo1', key2: 'Helloo2'}

    obj.key2="Welcome 2";

    console.log(obj);   //{key1: 'Hellooo1', key2: 'Helloo2'}

    delete obj.key2;

    console.log(obj);

</script>

<!-- Object.defineProperty is used for giving permission for newly adding keys like writable true or false

Object.defineProperty advance is Object.defineProperties used for multiple defineProperties -->

<script>

    let obj={

        "key1":"Hello1"

    };

    //add key2  --disable update and delete

    //add key3  --enable update and delete

    Object.defineProperty(obj,"key2",{

        value:"Helloooooo",

        writable:false

    });

    Object.defineProperty(obj,"key3",{

        value:"Hi ra",

        writable:true

    });

    obj.key2="Welcomeeeee";

    obj.key3="it is changing";

    console.log(obj);

    delete obj.key2;

    delete obj.key3;

    console.log(obj);

</script>

<!-- Object.defineProperties is used for multiple managing the access of keys -->

<script>

    let obj={

        "key1":"helloo1"

    }

    Object.defineProperties(obj,{

        "key2":{

            value:"It is working",

            writable:false

        },

        "key3":{

            value:"Working in val3",

            writable:true

        }

    })

    console.log(obj);   //{key1: 'helloo1', key2: 'It is working', key3: 'Working in val3'}

    obj.key2="Welcome2";

    obj.key3="Welcome3";

    console.log(obj);   //{key1: 'helloo1', key2: 'It is working', key3: 'Welcome3'} third changed due to writable true

</script>

<!-- obj.123 it is giving error so we cant call in that way -->

<!-- use obj[123] for non string keys i.e use [] -->

<script>

    let obj={

        "key1":"Hello1",

        123:"Hello2"

    }

    console.log(obj.key1)    //Hello1

    console.log(obj[123]);   //It is number so it is giving error

</script>

<script>

    let obj={

    };

    let key1={

    };

    let value1="Hleeooo";

    obj[key1]=value1;   //used [] bcz not string,key is {} it is json obj

    let key2={

    };

    let value2="Hiiiiii";

    obj[key2]=value2;

    console.log(obj);   //{[object Object]: 'Hiiiiii'}  only one output last is overriding

    /\* here we are not getting two key,values it is overriding the last one so only one we can add dynamically. \*/

</script>

<!-- We can not take the duplicate keys it will override -->

<!-- <script>

    let obj={

        "key1":"Hello1",

        "key1":"Hello2"

    }

    console.log(obj);   //{key1: 'Hello2'}

</script> -->

<script>

    let obj={

        "key1":"Hellooooo11",

        "key2":"Heloo2"

    }

    console.log(obj);   //{key1: 'Hellooooo11', key2: 'Heloo2'}

</script>

<!-- Important destructing and types -->

<!-- <script>

    let obj={

        "key1":"Hello1",

        "key2":"Hello2",

        "key3":"Heelloo3"

    };

    const{key1,key2,key3}=obj;  //It is called destructing

    console.log(obj.key1,obj.key2,obj.key3);    //By direct . method used for only strings

    console.log(obj["key1"],obj["key2"],obj["key3"]);   //By [] methods boss of all

    console.log(key1,key2,key3);

    for(let key in obj){

        console.log(obj[key]);

    }

</script>

This is example of destructing

<script>

    let myson\_function={

        "money":{"amount":1000},

        "gold":{"weight":"1gr"},

        "book":{"company":"nirmal"},

        "pen":{"brand":"bitco"}

    }

    let {money,gold,book,pen}=myson\_function    //rem syntax const={insidekeys}=mainobj

    let {amount}=money;

    let {weight}=gold;

    let {company}=book;

    let {brand}=pen;

    console.log(amount,weight,company,brand);

</script>

<!--imp: document.write is used for printing on the screen from script tag -->

<script>

    let arr=[

        {"eid":111,"ename":"nikhil","esal":1000},

        {"eid":222,"ename":"raj","esal":2000},

        {"eid":333,"ename":"venkat","esal":3000},

        {"eid":444,"ename":"harish","esal":4000},

        {"eid":555,"ename":"anil","esal":5000}

    ];

    /\* Important:  we have to use `` bcz it is inside the javascript code js is considered as string in js \*/

    document.write(`

        <table border="1",

        cellpadding="10px",

        cellspacing="10px",

        align="center">

        <tr>

            <th>EID</th>

            <th>ENAME</th>

            <th>ESALARY</th>

        </tr>

    `);

    arr.forEach((element)=>{

        document.write(`

        <tr>

            <td>${element.eid}</td>

            <td>${element.ename}</td>

            <td>${element.esal}</td>

        </tr>

        `);

    });

    document.write(`</table>`);

</script>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css">

<script>

    //json array

    //here name and images " "quotes are not mandatory those are optional

    let products=[

        {"name":"shirt1","cost":100,"images":"https://images.bewakoof.com/utter/content/2640/content\_Brown\_pant\_shirt\_combination.jpg"},

        {"name":"shirt2","cost":200,"images":"https://i.pinimg.com/474x/f8/13/d6/f813d6b033ffd7ed4f2b2aaf8a03565a.jpg"},

        {"name":"shirt3","cost":300,"images":"https://img.kwcdn.com/product/fancy/d86e060d-3816-4b1a-8383-572dca3fd043.jpg?imageMogr2%2Fauto-orient%7CimageView2%2F2%2Fw%2F800%2Fq%2F70%2Fformat%2Fwebp="}

    ];

    document.write(`

    <table border="1",cellpadding="10px",cellspacing="10px",align="center">

        <tr>

            <th>Name</th>

            <th>cost</th>

            <th>Images</th>

            <th>Actions</th>

        </tr>

    `);

    products.forEach((ele)=>{

        document.write(`

        <tr>

            <td>${ele.name}</td>

            <td>${ele.cost}</td>

            <td><img width="30px" src=${ele.images}></td>

            <td>

                    <i class="fa-solid fa-trash" style="color:red; cursor:pointer;"></i>

                    &nbsp;

                    <i class="fa-solid fa-pen-to-square" style="color:blue; cursor:pointer;"></i>

                </td>

        </tr>

        `);

    })

</script>

<script>

    let countries={

    "regional": [

      {

        "loc": "Andaman and Nicobar Islands",

        "confirmedCasesIndian": 10039,

        "confirmedCasesForeign": 0,

        "discharged": 9907,

        "deaths": 129,

        "totalConfirmed": 10039

      },

      {

        "loc": "Andhra Pradesh",

        "confirmedCasesIndian": 2319797,

        "confirmedCasesForeign": 0,

        "discharged": 2304995,

        "deaths": 14730,

        "totalConfirmed": 2319797

      },

      {

        "loc": "Arunachal Pradesh",

        "confirmedCasesIndian": 64503,

        "confirmedCasesForeign": 0,

        "discharged": 64203,

        "deaths": 296,

        "totalConfirmed": 64503

      },

      {

        "loc": "Assam",

        "confirmedCasesIndian": 724220,

        "confirmedCasesForeign": 0,

        "discharged": 716232,

        "deaths": 7986,

        "totalConfirmed": 724220

      },

      {

        "loc": "Bihar",

        "confirmedCasesIndian": 830653,

        "confirmedCasesForeign": 0,

        "discharged": 818348,

        "deaths": 12256,

        "totalConfirmed": 830653

      },

      {

        "loc": "Chandigarh",

        "confirmedCasesIndian": 92208,

        "confirmedCasesForeign": 0,

        "discharged": 90959,

        "deaths": 1165,

        "totalConfirmed": 92208

      },

      {

        "loc": "Chhattisgarh",

        "confirmedCasesIndian": 1152333,

        "confirmedCasesForeign": 0,

        "discharged": 1138271,

        "deaths": 14034,

        "totalConfirmed": 1152333

      },

      {

        "loc": "Dadra and Nagar Haveli and Daman and Diu",

        "confirmedCasesIndian": 11441,

        "confirmedCasesForeign": 0,

        "discharged": 11437,

        "deaths": 4,

        "totalConfirmed": 11441

      },

      {

        "loc": "Delhi",

        "confirmedCasesIndian": 1899071,

        "confirmedCasesForeign": 1,

        "discharged": 1868543,

        "deaths": 26188,

        "totalConfirmed": 1899072

      },

      {

        "loc": "Goa",

        "confirmedCasesIndian": 245547,

        "confirmedCasesForeign": 1,

        "discharged": 241649,

        "deaths": 3832,

        "totalConfirmed": 245548

      },

      {

        "loc": "Gujarat",

        "confirmedCasesIndian": 1224656,

        "confirmedCasesForeign": 1,

        "discharged": 1213502,

        "deaths": 10944,

        "totalConfirmed": 1224657

      },

      {

        "loc": "Haryana",

        "confirmedCasesIndian": 998538,

        "confirmedCasesForeign": 14,

        "discharged": 985996,

        "deaths": 10621,

        "totalConfirmed": 998552

      },

      {

        "loc": "Himachal Pradesh",

        "confirmedCasesIndian": 284926,

        "confirmedCasesForeign": 0,

        "discharged": 280721,

        "deaths": 4136,

        "totalConfirmed": 284926

      },

      {

        "loc": "Jammu and Kashmir",

        "confirmedCasesIndian": 454131,

        "confirmedCasesForeign": 0,

        "discharged": 449325,

        "deaths": 4751,

        "totalConfirmed": 454131

      },

      {

        "loc": "Jharkhand",

        "confirmedCasesIndian": 435289,

        "confirmedCasesForeign": 0,

        "discharged": 429933,

        "deaths": 5318,

        "totalConfirmed": 435289

      },

      {

        "loc": "Karnataka",

        "confirmedCasesIndian": 3949446,

        "confirmedCasesForeign": 0,

        "discharged": 3907480,

        "deaths": 40105,

        "totalConfirmed": 3949446

      },

      {

        "loc": "Kerala\*\*\*",

        "confirmedCasesIndian": 6546235,

        "confirmedCasesForeign": 0,

        "discharged": 6473557,

        "deaths": 69355,

        "totalConfirmed": 6546235

      },

      {

        "loc": "Ladakh",

        "confirmedCasesIndian": 28256,

        "confirmedCasesForeign": 0,

        "discharged": 28019,

        "deaths": 228,

        "totalConfirmed": 28256

      },

      {

        "loc": "Lakshadweep",

        "confirmedCasesIndian": 11402,

        "confirmedCasesForeign": 0,

        "discharged": 11350,

        "deaths": 52,

        "totalConfirmed": 11402

      },

      {

        "loc": "Madhya Pradesh",

        "confirmedCasesIndian": 1041867,

        "confirmedCasesForeign": 0,

        "discharged": 1030900,

        "deaths": 10735,

        "totalConfirmed": 1041867

      },

      {

        "loc": "Maharashtra",

        "confirmedCasesIndian": 7880334,

        "confirmedCasesForeign": 3,

        "discharged": 7731029,

        "deaths": 147853,

        "totalConfirmed": 7880337

      },

      {

        "loc": "Manipur",

        "confirmedCasesIndian": 137230,

        "confirmedCasesForeign": 0,

        "discharged": 135102,

        "deaths": 2120,

        "totalConfirmed": 137230

      },

      {

        "loc": "Meghalaya",

        "confirmedCasesIndian": 93819,

        "confirmedCasesForeign": 0,

        "discharged": 92213,

        "deaths": 1593,

        "totalConfirmed": 93819

      },

      {

        "loc": "Mizoram",

        "confirmedCasesIndian": 227962,

        "confirmedCasesForeign": 0,

        "discharged": 227005,

        "deaths": 697,

        "totalConfirmed": 227962

      },

      {

        "loc": "Nagaland",

        "confirmedCasesIndian": 35492,

        "confirmedCasesForeign": 0,

        "discharged": 34731,

        "deaths": 760,

        "totalConfirmed": 35492

      },

      {

        "loc": "Odisha",

        "confirmedCasesIndian": 1288272,

        "confirmedCasesForeign": 0,

        "discharged": 1278980,

        "deaths": 9126,

        "totalConfirmed": 1288272

      },

      {

        "loc": "Puducherry",

        "confirmedCasesIndian": 165806,

        "confirmedCasesForeign": 0,

        "discharged": 163832,

        "deaths": 1962,

        "totalConfirmed": 165806

      },

      {

        "loc": "Punjab",

        "confirmedCasesIndian": 760028,

        "confirmedCasesForeign": 0,

        "discharged": 742107,

        "deaths": 17751,

        "totalConfirmed": 760028

      },

      {

        "loc": "Rajasthan",

        "confirmedCasesIndian": 1284616,

        "confirmedCasesForeign": 2,

        "discharged": 1274448,

        "deaths": 9554,

        "totalConfirmed": 1284618

      },

      {

        "loc": "Sikkim",

        "confirmedCasesIndian": 39161,

        "confirmedCasesForeign": 0,

        "discharged": 38702,

        "deaths": 452,

        "totalConfirmed": 39161

      },

      {

        "loc": "Tamil Nadu",

        "confirmedCasesIndian": 3454550,

        "confirmedCasesForeign": 6,

        "discharged": 3416107,

        "deaths": 38025,

        "totalConfirmed": 3454556

      },

      {

        "loc": "Telangana",

        "confirmedCasesIndian": 792526,

        "confirmedCasesForeign": 0,

        "discharged": 787997,

        "deaths": 4111,

        "totalConfirmed": 792526

      },

      {

        "loc": "Tripura",

        "confirmedCasesIndian": 100887,

        "confirmedCasesForeign": 0,

        "discharged": 99963,

        "deaths": 923,

        "totalConfirmed": 100887

      },

      {

        "loc": "Uttarakhand",

        "confirmedCasesIndian": 437673,

        "confirmedCasesForeign": 1,

        "discharged": 429504,

        "deaths": 7693,

        "totalConfirmed": 437674

      },

      {

        "loc": "Uttar Pradesh",

        "confirmedCasesIndian": 2077405,

        "confirmedCasesForeign": 1,

        "discharged": 2052615,

        "deaths": 23513,

        "totalConfirmed": 2077406

      },

      {

        "loc": "West Bengal",

        "confirmedCasesIndian": 2018763,

        "confirmedCasesForeign": 0,

        "discharged": 1997153,

        "deaths": 21203,

        "totalConfirmed": 2018763

      }

    ]

  }

let {regional}=countries;

document.write(`

<table border="1">

<tr>

    <th>loc</th>

    <th>confirmedCasesIndian</th>

    <th>confirmedCasesForeign</th>

    <th>discharged</th>

    <th>deaths</th>

    <th>totalConfirmed</th>

</tr>

`);

regional.forEach((ele)=>{

    document.write(`

    <tr>

        <td>${ele.loc}</td>

        <td>${ele.confirmedCasesIndian}</td>

        <td>${ele.confirmedCasesForeign}</td>

        <td>${ele.discharged}</td>

        <td>${ele.deaths}</td>

        <td>${ele.totalConfirmed}</td>

    </tr>

    `);

})

document.write('</table>');

</script>

Part 3

<script>

    let obj={"key1":"Helloo1"};

    console.log(typeof obj);    //object

    /\* to convert object to string we have JSON.stringify rem stringify means converting to string \*/

    let str=JSON.stringify(obj);

    console.log(typeof str);    //string

</script>

1.JSON.stringify()->obj to string

2.JSON.parse()->string to obj

<script>

    let obj={

        "key1":"Hellooo1",

        "key2":"hello2",

        "key3":"Hello3"

    };

    console.log(obj.hasOwnProperty("key1"));    //true

    console.log(obj.hasOwnProperty("key2"));    //true

    console.log(obj.hasOwnProperty("key3"));    //true

    console.log(obj.hasOwnProperty("key4"));    //false

</script>

<script>

    let obj={

    };

    //check obj is empty or not

    //rem this Object.keys(obj) syntax

    Object.keys(obj).length===0?console.log("Empty"):console.log("Not Empty");

    Object.values(obj).length===0?console.log("Empty"):console.log("Not Empty");

</script>

<script>

    let obj={

        "a":"b",

        "c":"d",

        "e":"f"

    };

    let arr=Object.entries(obj);

    console.log(arr);

</script>

Important: Object.assign is used for concatination

<script>

    let obj1={

        "key1":"Hello1"

    };

    let obj2={

        "key2":"Hello2"

    }

    let obj3={

        "key3":"Hello3"

    }

    console.log(Object.assign(obj1,obj2,obj3));

</script>

<script>

    let arr=[100,200,300,400,500];

    console.log(

        arr.filter((ele)=>{

            return ele>=300;

        })

    );

</script>

<script>

    let arr=[100,200,300,400,500];

    console.log(

        arr.filter((element)=>{

            return element<=300;

        }).map((ele)=>{

            return {"key":ele};

        })

    );

</script>

<script>

    let arr=[

        {"price":100},

        {"price":200},

        {"price":300},

        {"price":400}

    ];

    console.log(

        arr.filter((ele)=>{

            return ele.price>=300;

        })

    );

</script>

<script>

    let key1="Hello1";  //key=value

    let key2="Hello2";

    let obj={

        key1,

        key2            //taking both key and value

    };

    console.log(obj);   //{key1: 'Hello1', key2: 'Hello2'}

</script>

part-4

**JSON Server**

rest apis

 GET,POST,PUT,DELETE

 JSON server by default running on port number "3000"

 we will install JSON server by following command

 npm install -g json-server

 npm stands for node packaging manager

 -g stands for global installation

we will load the data in JSON server by using following command

json-server --watch data.json(need to create data.json)

we will test rest apis with postman

create one file name data.json

<!-- this is so much imp-> json-server --watch data.json and use post man -->

{

  "employees": [

    {

      "id": "111",

      "name": "e\_one",

      "sal": 1000

    },

    {

      "id": "555",

      "name": "e\_five",

      "sal": 5000

    },

    {

      "id": "222",

      "name": "e\_two",

      "sal": 2000

    },

    {

      "id": "444",

      "name": "e\_four",

      "sal": 4000

    },

    {

      "id": "333",

      "name": "e\_three\_updated",

      "sal": 30

    },

    {

      "id": "666",

      "name": "e\_six",

      "sal": 6000

    }

  ]

}

<!-- GET->http://localhost:3000/products

POST->http://localhost:3000/products

PATCH->http://localhost:3000/products/4

DELETE->http://localhost:3000/products/6

GET->http://localhost:3000/products?\_sort=id

This is used for sorting the data in descending order rem ?\_sort=id (\_ is imp)

http://localhost:3000/products?\_sort=id&\_order=desc (for desc) -->

<!-- http://localhost:3000/employees/111

http://localhost:3000/employees?sal=4000

http://localhost:3000/employees?\_start=0&\_end=3 (index based and from 3 excluded)

http://localhost:3000/employees?id\_ne=111(ne=not equal to)

http://localhost:3000/employees?q=111

lte->lessthanorequalto -->gte greater…

Arrays

/\* array-collection of indexed elements called as array

index starts from 0

we will represent the array by using [] but in java int[] arr=new int[5] or int[] arr={1,2,3}; diff is {}

 \*/

//length

/\* let arr=[10,20,30,40,50]

console.log(arr.length);    //5

console.log(arr[0],arr[1],arr[2],arr[3],arr[4],arr[5],arr[-1]);    //10 20 30 40 50 undefined undefined

arr.length=3;   //making the length to 3 removing 40 and 50

console.log(arr[0],arr[2],arr[3],arr[4],arr[-1]);   //10 30 undefined undefined undefined \*/

//push()->add the element at end

//pop()->delete element at end

//unshift->add the element at beginning

//shift->remove the element from begining

let arr=[20,30,40];

console.log(arr);   //[ 20, 30, 40 ]

arr.push(50);

console.log(arr);//[ 20, 30, 40, 50 ]

arr.unshift(10);

console.log(arr);   //[ 10, 20, 30, 40, 50 ]

arr.pop();

console.log(arr);   //[ 10, 20, 30, 40 ]

arr.shift();

console.log(arr);//[ 20, 30, 40 ]

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

console.log(arr.length);       //5

delete arr[0];

console.log(arr);   //[ <1 empty item>, 20, 30, 40, 50 ]

console.log(arr.length);    //5->drawback of delete it wont remove memory

//making duplicate or xerox copy by ...spread operator same as json line no 178

let arr1=[10,20,30];

let arr2=[...arr1];

console.log(arr1);  //[ 10, 20, 30 ]

console.log(arr2); //[ 10, 20, 30 ]

arr1.push(46);

arr2.pop();

console.log("Arr1 ",arr1);  //Arr1  [ 10, 20, 30, 46 ] added thing wont change in duplicat copy

console.log("Arr2 ",arr2); //Arr2  [ 10, 20 ]

we are creating duplicate copy of arr1 and creating it as arr2 if made changes in arr2 it wont reflect in arr1 it Is called immutablitiy

let arr1=[20,30,40];

let arr2=[...arr1];

//console.log(arr1);

//console.log(arr2);

arr2.push(50); //add end

arr2.unshift(10);//add first

console.log(arr1);

console.log(arr2);

let arr1=[10,20,30,40,50,60,70,80,90,100];

//indexs start from 0 to 9 if I want to remove 40 its index is 3 how many no needed give that like 2

arr1.splice(3,4);   //to remove 40and50,60,70(4 elements);//[ 10, 20, 30, 80, 90, 100 ]

console.log(arr1);  //[ 10, 20, 30, 80, 90, 100 ]

arr1.splice(5,1);

console.log(arr1);  //[ 10, 20, 30, 80, 90 ]

arr1.splice(3,0,40,50);     //we can add by using splice 3->index,0->we are not deleting so 0,40 and 50 we are adding

console.log(arr1);  //[10, 20, 30, 40, 50, 80, 90]

arr1.splice(0,1,5,6,7); //0 index to add,1 no.of items are deleted i.e 10,567 are added

console.log(arr1);      //[5,  6,  7, 20, 30,40, 50, 80, 90]

arr1.splice(9,0,100);

console.log(arr1);

let arr1=[0,1,2,3,4,5,6,7,8,9]

arr1.splice(0,1);

console.log(arr1);

arr1.splice(8,1);

console.log(arr1);  //[1, 2, 3, 4, 5, 6, 7, 8]

arr1.splice(3,1);   //rem upto 3 it will be and after 3 i.e 4th place it will delete

console.log(arr1);  //[1, 2, 3, 5, 6, 7, 8]

arr1.splice(7,0,9); //at 7th place we need add and 0 no need to delete and add 7

console.log(arr1);  //[1, 2, 3, 5, 6, 7, 8, 9]

arr1.splice(0,0,0);  //[0, 1, 2, 3, 5, 6, 7, 8, 9]

console.log(arr1);

arr1.splice(4,0,4);

console.log(arr1);  //[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

//map()

//used to manipulate each element

console.log(

    [1,2,3,4,5].map((ele,index)=>{

        return ele\*100;

    })  //[ 100, 200, 300, 400, 500 ]

)

console.log(

    [1,2,3,4,5].map((element,index)=>{

        return [element];   //[ [ 1 ], [ 2 ], [ 3 ], [ 4 ], [ 5 ] ]

    })

)

console.log(

    [1,2,3,4,5].map((ele,index)=>{

        return [ele,index]; //[ [ 1, 0 ], [ 2, 1 ], [ 3, 2 ], [ 4, 3 ], [ 5, 4 ] ]

    })

)

//rem here {[i]:e} where[] is mandatory rem

console.log(

    [1,2,3,4,5].map((e,i)=>{

        return {[i]:e};         //[ { '0': 1 }, { '1': 2 }, { '2': 3 }, { '3': 4 }, { '4': 5 } ]

    })

)

console.log(

    [1,2,3,4,5].filter((ele,index)=>{

        return ele>=3;

    }).map((ele,ind)=>{     //index is not mandatory

        return ele\*100; //[ 300, 400, 500 ]

    })

)

console.log(

    [{"e\_id":111,"e\_name":"Nikhil","e\_sal":1000},

    {"e\_id":222,"e\_name":"Kiran","e\_sal":2000},

    {"e\_id":333,"e\_name":"vamshi","e\_sal":3000},

    {"e\_id":444,"e\_name":"Ajay","e\_sal":4000},

    {"e\_id":555,"e\_name":"Venkat","e\_sal":5000}].filter((ele,ind)=>{

        return ele.e\_sal>=3000;

    }).map((ele,ind)=>{

        return ele.e\_name==="Ajay" ? {"e\_id":444,"e\_name":"Suresh","e\_sal":4000}:ele;

    })

)

//reduce()

//it is used to make the array values to single digit like sum of array values

//The process is first 1 will store in firstEle, and 2 will store in accu 1+2=3, 3 will store in firstEle, 3+3(next ele)=6,again stored in firstEle,6+4=10,10+5=15

console.log(

    [1,2,3,4,5].reduce((firstElement,accumlator)=>{     //any thing can be written not only firstElement,accumilator

        return firstElement+accumlator;

    })  //15

)

console.log(

    [100,200,300,400,500].reduce((firstElement,accumlator)=>{     //any thing can be written not only firstElement,accumilator

        return firstElement+accumlator;

    })  //1500

)

console.log(

    ["javascript","to","welcome"].reduceRight((firstele,acc)=>{

        return firstele+" "+acc;

    })

)

//REM: splice and slice both are different slice cut directly but splice will print after that elements also

//slice means give only that elements but splice means gives remaining elements

//index will be negative also -1,-2,-3,... from backside i.e from 100

//100=-1,90=-2,80=-3

let arr1=[10,20,30,40,50,60,70,80,90,100];

console.log(arr1.slice(4,6))   //[ 50, 60 ] //4 included and 6 exclued in index

console.log(arr1.slice(7,9));   //[80,90] //7 means 80 included and 8 means 90 included but 9 is not included i.e 100

console.log(arr1.slice(0,1));   //[10]

console.log(arr1.slice(9,10));  //[100]

console.log(arr1.slice((1,4))); //[20,30,40]

console.log(arr1.slice(-6,-4)); //[50,60] -from backside

console.log(arr1.slice(-10,-9));  //[10]

console.log(arr1.slice(-1));    //100

console.log(arr1.slice(-4,-1));//70,80,90

//if [[]] is used then only use flat(1), if more than one use flat(Infinity)

console.log(

    [[1],[2],[3]].flat(1)

) // Output: [1, 2, 3]

let result = [

    [[[[[[[[1]]]]]]]],

    [[[[[[[[2]]]]]]]],

    [[[[[[[[[[[[3]]]]]]]]]]]]

].flat(Infinity);

console.log(result); // Output: [1, 2, 3]

**//imp**

**let arr1=[1,2,3]**

**let arr2=["one","two","three"]**

**console.log(**

**arr1.map((ele,ind)=>{**

**return [ele,arr2[ind]];**

**})  //[ [ 1, 'one' ], [ 2, 'two' ], [ 3, 'three' ] ]**

**)**

//here we are making it individual array

let arr1=[1,2,3]

let arr2=["one","two","three"]

console.log(

    arr1.map((ele,ind)=>{

        return[ele,arr2[ind]];

    }).flat(Infinity)   //[ 1, 'one', 2, 'two', 3, 'three' ]

)

//instead of using two function like map and flat use flatMap()

let arr11=[1,2,3]

let arr22=["one","two","three"]

console.log(

    arr1.flatMap((ele,ind)=>{

        return[ele,arr2[ind]];

    })   //[ 1, 'one', 2, 'two', 3, 'three' ]

)