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>