JavaSript

Javascript is base for the nodejs.

**Object**

* through moucse event and the keybord events we have to achive through the javascript
* JavaScript is oobject oriented programming.
* Object in java script is very flexiable and it will be add any property at any time if it is not their in the object.
* Objects are key value pair.
* Two type of the notation.
* Student{“ marks”:60}
  + - * 1. . Notation –use object name.property name that’s it . ex-student.marks
        2. Sruare bracket-use object name["property name ”]. Ex-student[“marks”]
        3. Use square backet in where the property invalid but present in the object. And also in the daynamic property condition.
* for var obj={

p1:20;

}

Obj is just referring to the object where it is store.

* Array is also the object like here actually we say that the arr.0 but dott notation is the issue with the invilade property so we use the square bracket arr[0]. But we know the in square bracket we need the “” but here arr[0]= arr[“0”] bocs here the type conversion take place means the numeric will convert into the string.

Advance Js

Closures

If we create a function in the global context then it is added in the window object as a property or we can say that property is created under the window object as the name of the function and the value of the property is same as the function it self

Or we can say that anything is created in the global it will part of the window object or as a property of the window object

Every excition context have lexical environment which have the variable, this, and also the reference to the outer context

Avoid the global vairiable bocs if we create two JS file with the same name it will in golobal and it will take the last js as the value of the name

Ex

Js-1

Var name=” file1”;

Console.log(name);

Js-2

Var name=” file2”;

Console.log(name);

And add both in the HTML file so its look like that

Var name=” file1”;

Console.log(name);

Var name=” file2”;

Console.log(name);

Although it write the name of the both file1 and file2 on the console but when asking about the name as the var of the window it write file2 bocs it will overwrite.

Or we write like the

In js-1

Var name=” file1”;

setTimeout(function(){

Console.log(name);

},1000);

In js-2

Var name=” file2”;

Console.log(name);

Both print the file 2

Bocs first is print after the 1 sec so it take name as file2 so due to overwritten

**IIFE(**Immediately Invoked Function Expression**)**

but if take both the JS in function but with different name then it will not in global scope now they are in the function scope but if again we create with the same name function it will create the same issue

so we need to make name of function different now answer is file2

file1

so if we don’t need to create the function then we use to IIFE means write the function without the name inclos in the ()and straight away call it like

(function(){

Var name=” file2”;

Console.log(name);

})();

But ES6 onward we use let why we not use here although it is block level scope yes we use it also here like this not need to bound in the function like IIFE

Just straight away write {

let name=” file2”; (instead of use var use let here)

Console.log(name);

}

Closures

Closure is the combination of the function and the lexical environment in which function was created.

var i=10;

        function outer(){

            var j=20;

            console.log(i,j);

            var inner=function(){

                var k=30;

                console.log(j,k);

            }

*return* inner;

        }

        var x=outer();

        x();

as in this see that inner function return in the outer function and it will call from global but In the inner function call the outer function var that is j the outer function is removed from the execution stack in all the laungauge but not in js bocs here not the var remove due to closer that’s why it print also j and not show any error

basically in the closure the inner function actually has access to the scope of the outer function they servive not remove from the memory and one think closure is created when the function is called from the out side of the function in which it Is created ex. In this we create the inner function in outer but call it from the global so that

var i=10;

function outer()

{

Var j=20;

Consol.log(I,j);

Function inner(){

Var k=30;

Consol.log(j,k);

}

Return inner;

}

Var x=outer();

X();

X();

Result is 20,30

21,30

Function test(){

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

Settimeout(function(){

Console.log(i);

},1000);

}

Consol.log(“after for loop”,i);

}

Test();

After for loop 6

Here the result is 6 6 6 6 6

Function test(){

For(let i=0;i<=5;i++){

Settimeout(function(){

Console.log(i);

},1000);

}

}

Test();

Result is 1 2 3 4 5bocs let has the block scope it means in let the vairiable will bind for the one itration only.

While in var it is global so print for all 6

**Let in the loop**

Let Is bound on the block scope so its easy when we say that it is in genrel its ok but it is deffrient in loop if it is block level then how the value will be increase in the for loop bocs every round of the for loop is one block so they have its own binding of i and it is destroy the block closed then how it is increase in 1st round i=1,in second

round it is i=2 how bocs in first loop it is destroyed.

**Arrow Function**

It is introduce in the ES6

Var multiply= function(x,y)

{

Return x\*y;

}

We can write it also as the

Var multiply=(x,y)=>{return x\*y;}

We can omit the return and paranthisis also if the single statement return like inn above

Var multiply=(x,y)=>x\*y; its only ok when the function have only single statement to be exciuted

If here the single input taking then also we can write the as also

Var double= x => x\*2;

**Binding arrow function**

Normal function have own binding to this,auguments and super

Arrow function do not have the binding with this they acttully look for the outer scope for the value of this

Function person(name){

This.name=name;

Consol.log(this);

setTimeout (function(){

consol.log(this);

},1000);

}

Var p=new person (“Manisha”);

Here is the two this frist in the above console and second in the second consol both have their own binding

So

1st this=prerson(name:”Manicha”);

2nd this =window{all property }

Function person(name){

This.name=name;

Consol.log(this);

setTimeout (()=>{

consol.log(this);

},1000);

}

Var p=new person (“Manisha”);

1st and 2nd both have same bocs the arrow function do not have their binding of this so call from outer scope that’s why same

**Constructors and Prototypes**

This keyward this behave in JS slightly different .

This in JS refer to the JS context object in which the current code is running.

(JS function is first class function it means you can pass function as argument and you can return them as values )

If we call a function in the simple way then this is going to refer the globe object is simple as the window.

Function demo(){

Consol.log(this);

}

Var demo2=function(){

Console.log(this);

}

Consol.log(this);

}

demo();

demo2();

all the answer is the window.

Above we cal in the simple way not we cal in the object way onther way calling a function is the via an object

Var obj={

‘prop1’:12,

‘pront’:function(){

Consol.log(this);

}

}

Obj.print();

In this case this is not referring to the global obj but it is referring in which it is created so the contxt of the value of this actually depend on which you calling a function.

There is 4 way to calling a function.

Above result in slopy mode not in the stricket mode

In strickt mode the plain way of calling the function is in the is showing the window function in the sloppy mode no it is showing the undefined in the strict mode but not effect in the when function is called vai the object .bocs in the plan method calling the function not bound to any the object that’s why it is showing the undefined due to the strict mode although show the window.

We can bound a function to a object vai call and apply.

Function demo(){

Consol.log(this);

}

Var demo2=function(){

Console.log(this);

}

Consol.log(this);

}

Demo.call(obj); now the palin call is bound to the obj we can bound to any obj just pass that obj.

We also call vai apply(obj ) the differenc in both of them if there is any aurgument we can direct pass in the call like demo.call(obj,3,4) but in the apply we have to pass that array in the argument like this (obj,[3,4])

**Function to create objects**

Class is as the template and objects are the instances of the classes in OOPS.

But this is also different in the JS,JS had not that tipically class concept like other laungauge

But in the javascript what ever we have is object even function are object.so create the no. of student we have to create the function in which the obj create and in which set the properties.

Function createStudent(name,rollNo,marks){  
var student ={};

Student.name=name;

Student.rollNo= rollNo;

Student.marks= marks;

Return student;

}

Var student1 =createStudent();

Var student2 =createStudent();

Here in js the tamplate is not like other laungauge in other lungauge its like the class but in the js it is created the function in which we create the object.

**Constructor**

Create the object every time we create the function which create this object and return the object to me

Js let you to call a method as a constructor it make thing as easer so this we don’t have to create the object in the function and do not need to return the object only think to need to call the method as the constructor like this **Var student1 =new createStudent(“abcd”,15,78);** now the function is called as the constructor so now the js it self take the values from above and create the object vai function and return to us.but here we don’t write like the student.name=name;

Bocs here the student not define so we have to use this keyword js create the object and call the function vai this object so the function is bound to the object and if we call a function and that function is bond to any object then this referes to that object

Function createStudent(name,rollNo,marks){

this.name=name;

this.rollNo= rollNo;

this.marks= marks;

}

Var student1 =new createStudent(“abcd”,5,58);

Var student2 =new createStudent(“xyza”,10,52);

If we don’t call vai the object meamns not use the new keyward this not bind the function to the object and this is undefined for that. And this in this objedct refer to the window object.

Other place we use camle case but here we sue the first letter is capital.

**Adding behavoiur to the object**

In the other laungauge like c++ and the java we add in class data as well as behavior like get function and all but this behavior function is not created many times in the other laungauge but in the JS it is created many time so this is waste of storage so use prototype for it

**What Is prototype**

Every function we create JS engine create its prototype also we access by function name.prototype

Every non primitive data type in JS is object and function is also a object s for every function two object is created one for function and other for its prototype.

Vehichle.prototype take me to the prototype of the function but vehichle.prototype.constructor take to the function

Prototype usefull for the function in constructore mode.

**Why Prototypes**

All the object are created have the interanal eference to the prototype

So the getprice is the behavioour so we need not to add in the function we have to write in the

Vehichle.prototype.getprice=function[

Return this.price;

]

No its not the part of the vehichle any more it is the part of the vehicle.prototype

Function vehicle(numwheels,price){

This.numwheels=numwheels:

This.price=price;

}

Vehicle.prototype.getprice=function(){

Return this.price:

}:

Var vehicle1 =new vehicle(2,50000);

Var vehicle2=new vehicle(4,500000);

Now it have not getprice in the vehicle function although it is use the prototype by the object have the reference of the prototype no getprice have only one copy and share by all the object.

We can also add the property in all the object in runtime in the and share by all the object this thing add property in the sun time cannot possible in the java and c++.

And if the property is not in the prototype it we lookup prototype cahin.

**More Properties around the prototype**

The \_\_proto\_\_ will take you to the prototype of the object vehicle1.\_\_proto\_\_===vehicle.prototype true bocs both are same but ES6 onwards its dipricated not use even although before it it is uses.

We have alternative object.getprotoypeog(vehicle1)

**Objects**

Var obj=new Object(); its similer to the var vehicle1=new vehicle(); so as the vehicle use in constructor mode and it is created by use but same type the Object is called but its not created by us so this is created in the JS and if it is present then the prototype of this also will be present

All the object is created at the top of it is object.prottype and they are inherent of it

If we create the vehicle1.prottype it have the internal reference to the vehicle.protype if we check the vehicle.prototype so it have the internal reference to the object.prottype

So we write and property 1st it will see in the v1 if not found go for the vehicle.prototype and then go in the object.prototype(this called lookup)

**Object Vs Object**

Object is the constructor function object is non primitive data type is key value pair.

**Class**

We can also use the class like

Class Vehicle{

Constructor(numwheel,price){

This.numwheels=numwheels;

This.price=price;

}

getPrice(){

return this.price;

}

}

In class we have to call constructor and also the getPrice function in deep down it work like prototype like

Class is not similar like the other laungauge it is different in the JS it is just a sentical presentation

**Class Expression and Hosting**

We cant call the class without the new keyward

If genral function hosting works but not work in the class although classes are the special function.

Like function expression class expression also works we can create the named and unnamed class expression

**Inheritance Using Classes**

Class vehicle {

Constructor(numwheeels,price){

This.numwheels=numwheels;

This.price=price;

}

getPrice(){

return this.price;

}

PrintDescription(){

Console.log(“vehivle”,this.numwheels,this.price);

}

}

Calss car extends vehicle{

Constructor(numdoors){

Superz(4,01000000000); (they call the parent class constructor )

This.numdooors=numdoors;  
}

}

Var c =new car(4);

Consol.log©;