**Contents**

1.Object

2.class

3.Inheritance.

4.Exception Handling

5.Closure

**1.Object**: In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

|  |  |  |
| --- | --- | --- |
| **Object** | **Properties** | **Methods** |
| https://www.w3schools.com/js/objectExplained.gif | car.name = Fiat  car.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()  car.stop() |

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

**1.1 Object Types:**Objects in js fall in one of four categories.The four categories are

1. User defined objects/Custom objects: The objects which is created by user.

2.Built-in Objects:- The objects which is provided by javascript language.

Ex: a. Objects associated with data types(String,Number,Boolean)

b. Objects that allows creation of user-defined object ,complex types(Object,Array).

c. Object that simplify the tasks(Date,Time,RegExp).

3.Browser Objects:- The browser objects are not provided by java script language.But browser supports them.

Ex:Window,Screen,Location,History,Navigator.

4.Document Objects:- Document objects are part of DOM which is defined by w3c.

Ex:-Document,Window,Collection,NodeList.

**1.2.Properties:** The property of object is named data. It is written as name/key:value pair.

* If key does not has multi word, we access property value using (.) dot notation.

Syntax: 1. objectname.key/name

2. objectname[‘key/name’]

* If key has multiword, then we put key with in “ “ and we use ([]) square bracket notation to access property value.

Syntax: .objectname[‘key/name’]

b) The property may be method. We can access function using (.) dot notation.

Syntax: object name. key/name().

Where key is function name.

**1.3. Object Creation Methods:** In java script, there are two methods to create object.

1. literalnotation.

2.constructor Function.

**1.3.1. Literal Method:-**

Syntax:

Var variable-name={key1:value,key2:value,….. keyn:value}

Var variable-name={}//empty object.

Drawback: When create object, every time The programmer has to specify complete structure of object and its property values.

Example:

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    div:first-child

    {

      border:1px solid red;

      text-align: center;

      margin: 5px;

    }

    div:nth-child(2)

    {

      border: 1px solid green;

      text-align:center;

      margin:5px;

    }

   p{

      border:1px solid orange;

      margin:5px;

    }

</style>

<body>

  </div>

  <script>

  vara={name:'maruthi',age:9,price:7000,display:function(){console.log('Name:'+this.name);console.log('Age:'+this.age);console.log('Price:'+this.price);}};

  console.log(a);

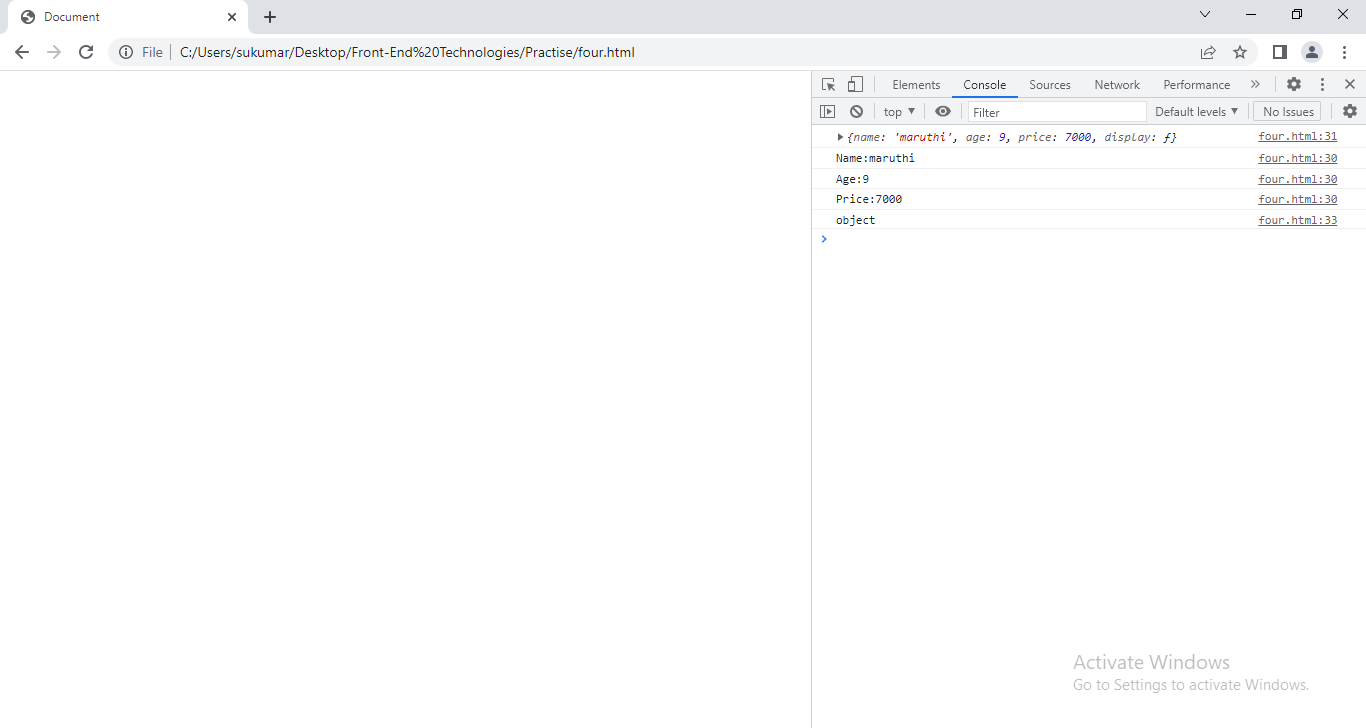
  a.display();

  console.log(typeof(a));

  </script>

</body>

</html>



1.3.1.1\_\_proto\_\_:Initially, \_\_proto\_\_ has [object Object] in literal object. In this we can put the another object reference.

By this hidden property, we can avoid code redundancy and we can achieve the reusability.

Example:

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    div:first-child

    {

      border:1px solid red;

      text-align: center;

      margin: 5px;

    }

    div:nth-child(2)

    {

      border: 1px solid green;

      text-align:center;

      margin:5px;

    }

   p{

      border:1px solid orange;

      margin:5px;

    }

</style>

<body>

  </div>

  <script>

  varcomAddress={name:'maruthi',town:'nlr',state:'ap'};

  varc1={owner:'A.sukumar',car\_no:7575};

  varc2={owner:'A.veena',car\_no:8575};

  c1.\_\_proto\_\_=comAddress;

  c2.\_\_proto\_\_=comAddress;

  console.log("===========================================================");

  for(vari in c1)

  {

    console.log(c1[i]);

  }

  for(vari in c2)

  {

    console.log(c2[i]);

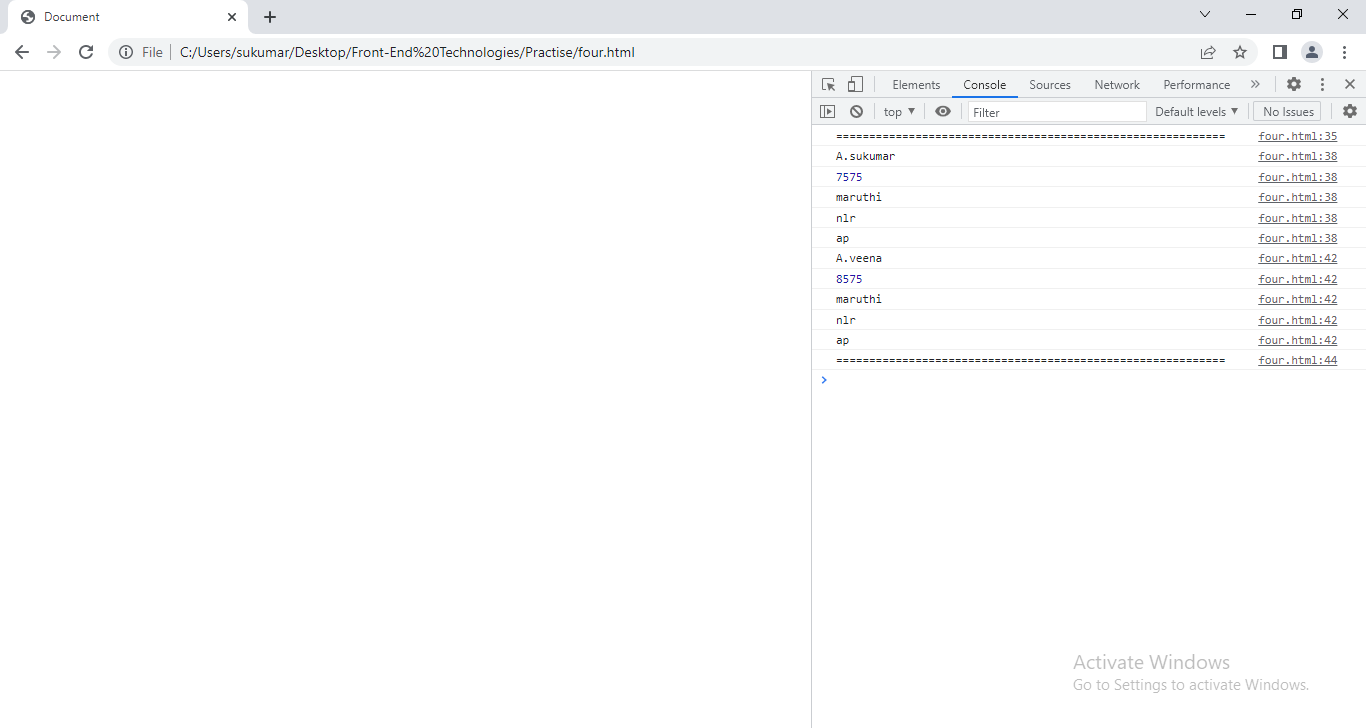
  }

  console.log("===========================================================");

  </script>

</body>

</html>



1.3.1.2. Adding & Deleting Properties:-

Syntax:

Var-name.propertyname=value;

Delete varname.propertyname=value;

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

  varc1={owner:'A.sukumar',car\_no:7575};

  c1.company='maruti';

  c1.price=9000;

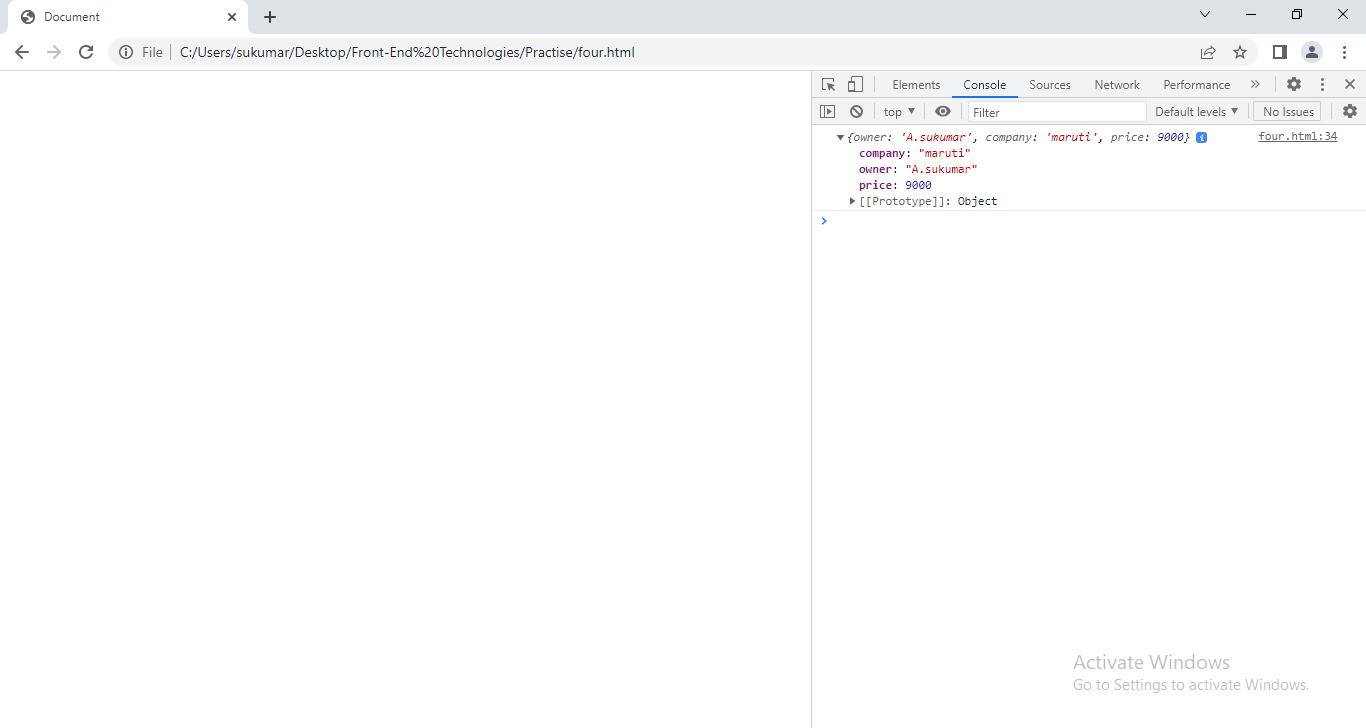
  delete c1.car\_no;

  console.log(c1);

  </script>

</body>

</html>



**1.3.2 Functional Literal:-**If we want to create more than one same objects which have same properties and different values than Object literal is not useful. In this case, we should use constructor function notation.

The programmer specifies structure of object only one time. Based on it and given values, JS engine create n number of same kind objects.

Syntax:

Function objectname([parameters])

{

This. Key1[=value];

This.key2[=value];

This.key3[=function([parameters]){body};]

….

This.keyn[=value];

}

Syntax to create object:

Varvar-name= new objectname([parameters]);

Varvar-name= new obejctname;//use this if constructor doe’t take arguments.

Example:

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

   functioncar(a,b,c){

    this.name=a;

    this.model=b;

    this.price=c;

   }

   c1=new car('A.sukumar',800,9000);

   c2=new car('A.veena',800,12000);

   console.log(c1);

   console.log(c2);

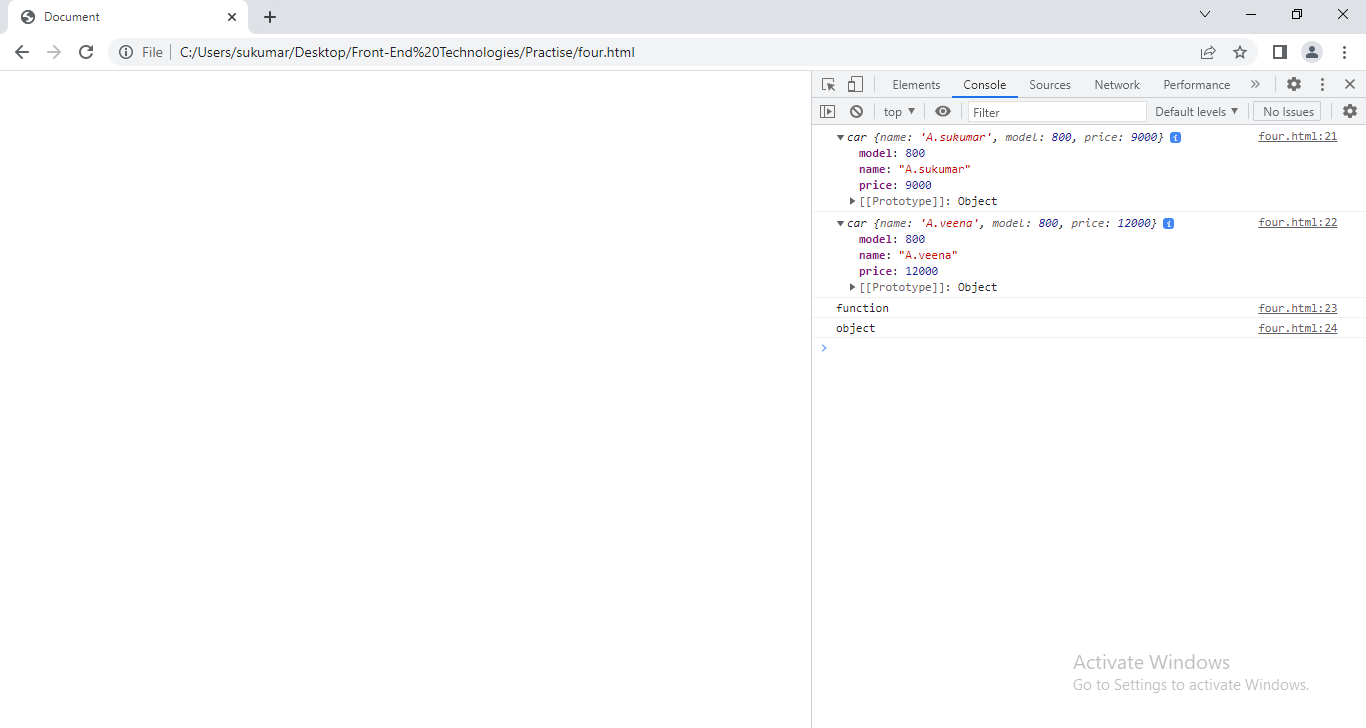
console.log(typeof(car));

console.log(typeof(c1));

  </script>

</body>

</html>



**1.3.2.1. \_\_proto\_\_**:- In this we can put the another object reference.

By this hidden property, we can avoid code redundancy and we can achieve the reusability.

Var var-name=new function\_name();

Var-name.\_\_proto\_\_=Reference of another object;

Example:-

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

   varcom\_add={com\_name:'maruti',dist:'nlr'};

   functionaddr(){

    this.state='ap';

    this.pin=524001;

   }

   varone=new addr();

   functioncar(a,b,c){

    this.name=a;

    this.model=b;

    this.price=c;

   }

   c1=new car('A.sukumar',800,9000);

   com\_add.\_\_proto\_\_=one;

   c1.\_\_proto\_\_=com\_add;

   c2=new car('A.veena',800,12000);

   c2.\_\_proto\_\_=com\_add;

   console.log(c1);

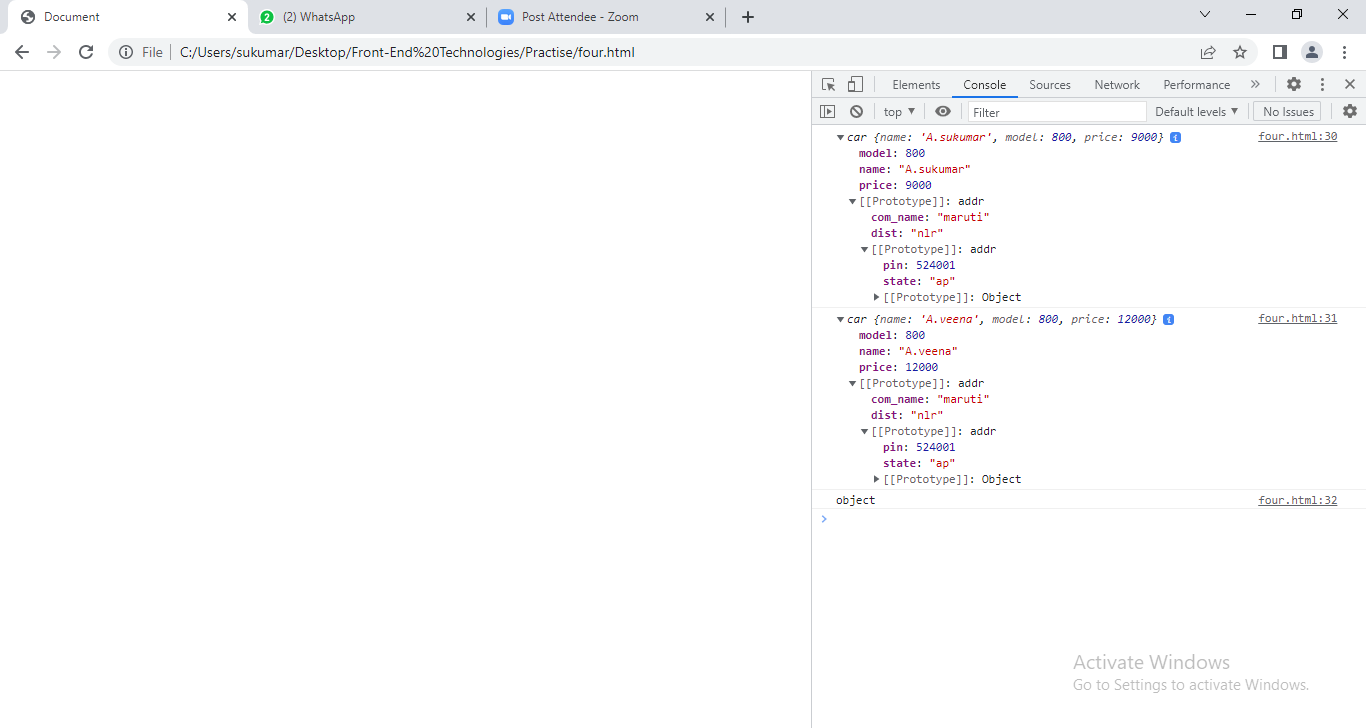
   console.log(c2);

   console.log(typeof(c1));

  </script>

</body>

</html>

.

**1.3.2.2.Prototype:-**

There is a problem in \_\_proto\_\_ .

Problem: After creating every object, we should inherit another object properties using \_\_proto\_\_ propterty.

EX:- 10 objects wants to inherit the properties of another same object. In program, the following lines should be written 10 times.

Function function\_name(){ }

Function\_name.\_\_proto\_\_= another object.

It leads to redundancy. To avoid redundancy, The prototype property has been introuduced.

a.Every function has the "prototype" property .It stores reference of object. That object has key named constructor that points back to the function itself.

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

   functioncar(a,b,c){

    this.name=a;

    this.model=b;

    this.price=c;

   }

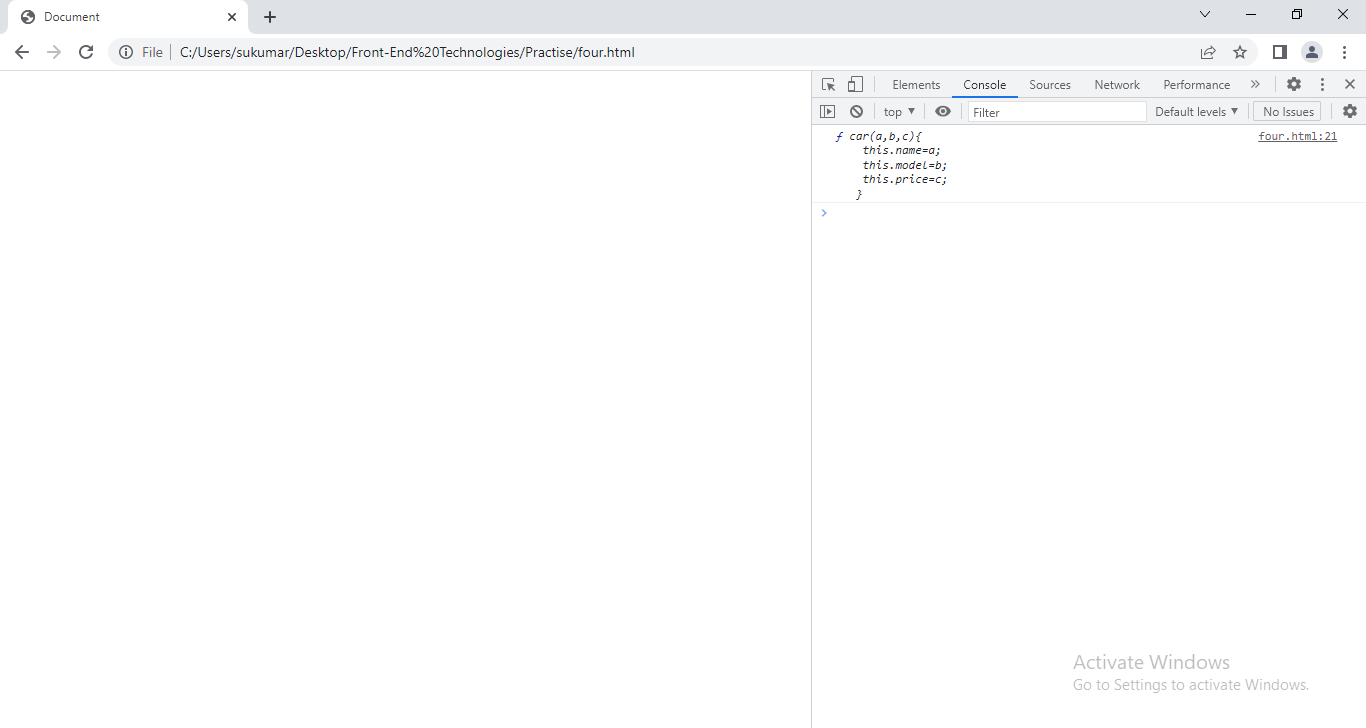
   one= new car();

   console.log(car.prototype.constructor);

  </script>

</body>

</html>



b. using property of constructor object,we can add new properties to built-in objects/user defined object.

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

   functionaddress(){

    this.dist='nlr';

    this.man='kvr';

   }

   varx=new address();

   functioncar(a,b,c){

    this.name=a;

    this.model=b;

    this.price=c;

   }

   car.prototype=x;

   one= new car('maruti',800,6000);

   two=new car('maruti',800,3000);

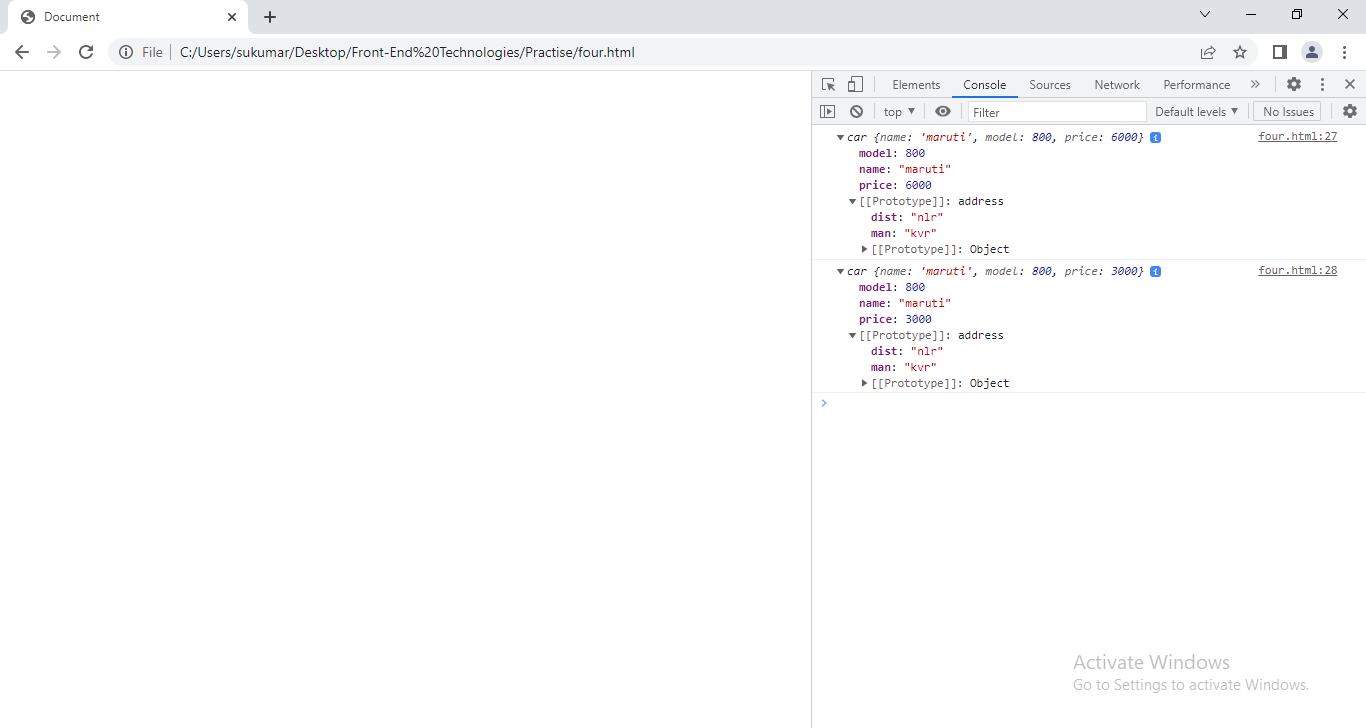
   console.log(one);

   console.log(two);

  </script>

</body>

</html>



**1.4.Object Destruction**: Object destructing means assigns the all or specific properties of object to individual variables.

Syntax:- let{ propertyname1:var1,propertyname2:var2….etc}=object\_name;

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

   varstuData={name:'suku',age:41,qual:'mca'};

   var{name:sname,age:sage,qual:squal}=stuData;

   console.log(sname);

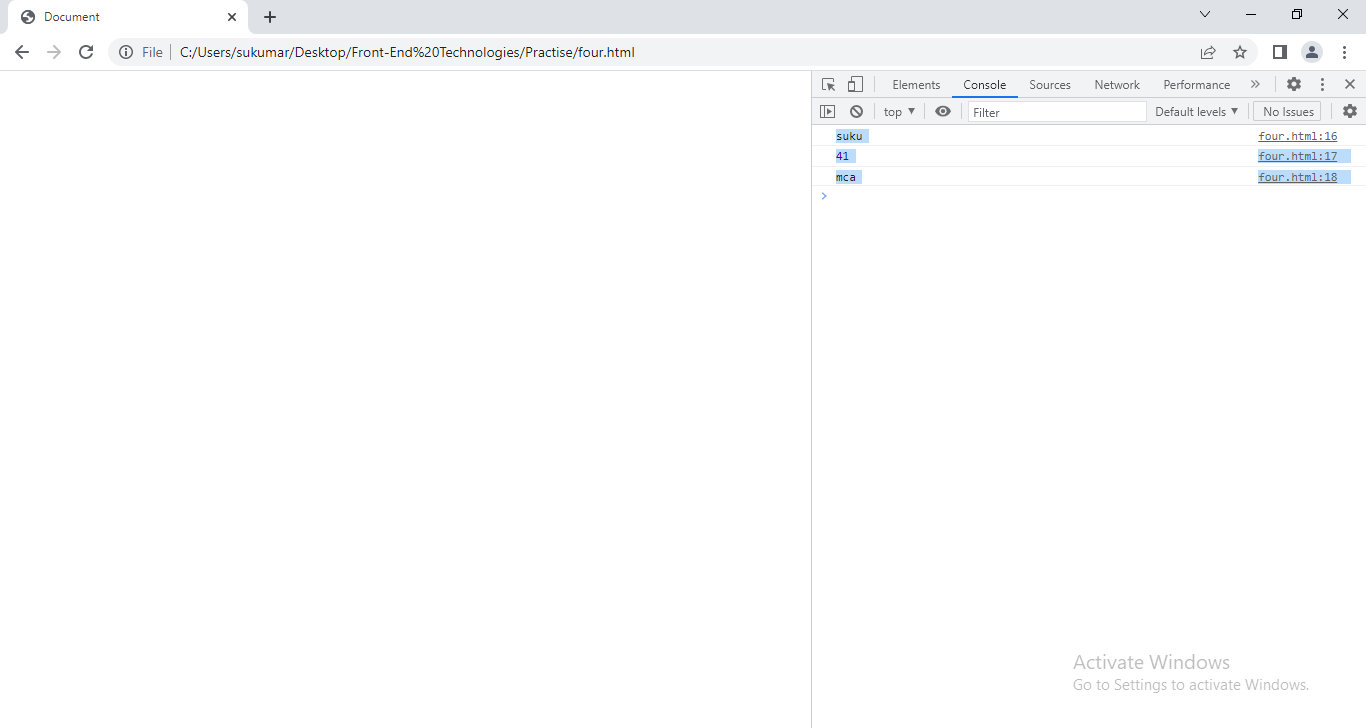
   console.log(sage);

   console.log(squal);

  </script>

</body>

</html>



**2.CLASS:**Class indicates or represents a predefined structure of an object so that we can create any number of objects having same structure but with different data .Until ECMA5 ,Classes we are not directly supported in JavaScript, using functions we could indirectly get the feature of Class. From ECMA6, Class is a predefined Keyword through which we could able to create classes directly.

Syntax:-

Class class-Name {

Constructor([parameters]){

This.key1[=value1];

This.key2[=value2];

…etc.

}

methodName(){ … }

…etc

}

Example:

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

   classStudent{

    constructor(){

      this.sName='suku';

      this.rNo='7';

    }

    display(){

      console.log(this.sName);

      console.log(this.rNo);

    }

  }

    vars1=new Student();

    vars2=new Student();

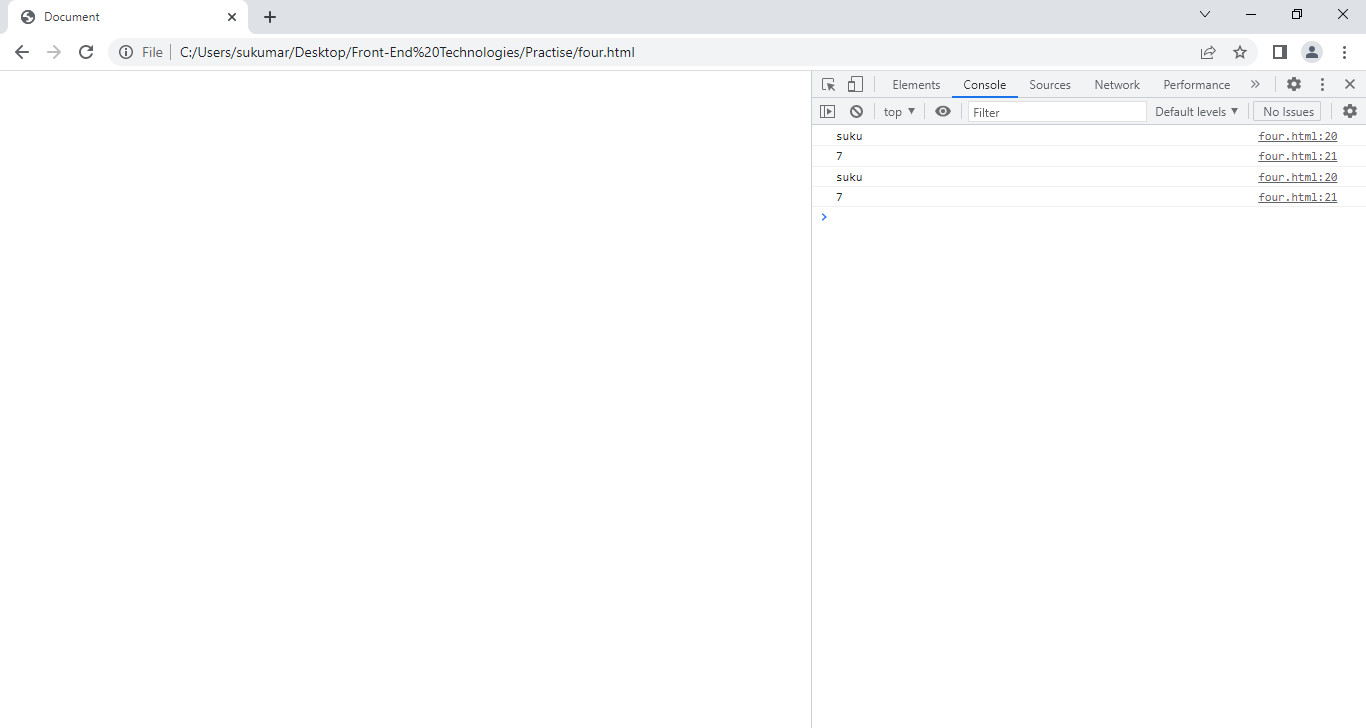
    s1.display();

    s2.display();

  </script>

</body>

</html>



Example:2

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

   classStudent{

    constructor(x){

      this.sName=x.name;

      this.rNo=x.age;

    }

    display(){

      console.log(this.sName);

      console.log(this.rNo);

    }

  }

    vars1=new Student({name:'suku',age:41});

    vars2=new Student({name:'sv',age:38});

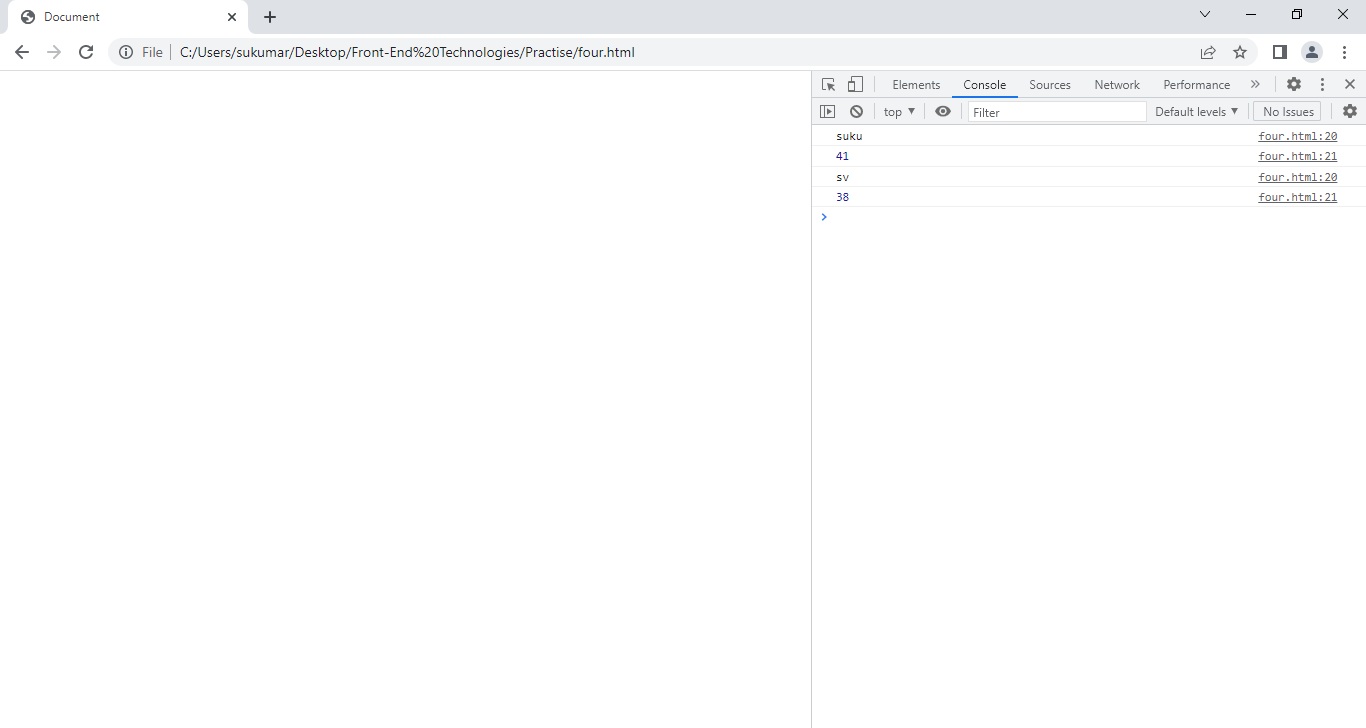
    s1.display();

    s2.display();

  </script>

</body>

</html>



**3.Inheritance:**Inheritance is useful for code reusability: reuse properties and methods of an existing class when you create a new class.The extends keyword is used to create a child class of another class (parent). The child class inherits all the methods from another class.

Syntax:

class Parent {

Constructor([arguments]) {

……. }

……… }

class child extends parent {

constructor () { super([arguments]); ……… }

}

The super() method refers to the parent class. By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods. Super() keyword should be declared in constructor in the first line itself in the child class.

Example:

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

  classParent{

         constructor(){

         this.branch='mca';

         this.year='2022';

         }

  }

  classChildextendsParent{

    constructor(a,b){

      super();

      this.name=a;

      this.age=b;

    }

    display(){

      console.log(this.name);

      console.log(this.age);

      console.log(this.branch);

      console.log(this.year);

    }

  }

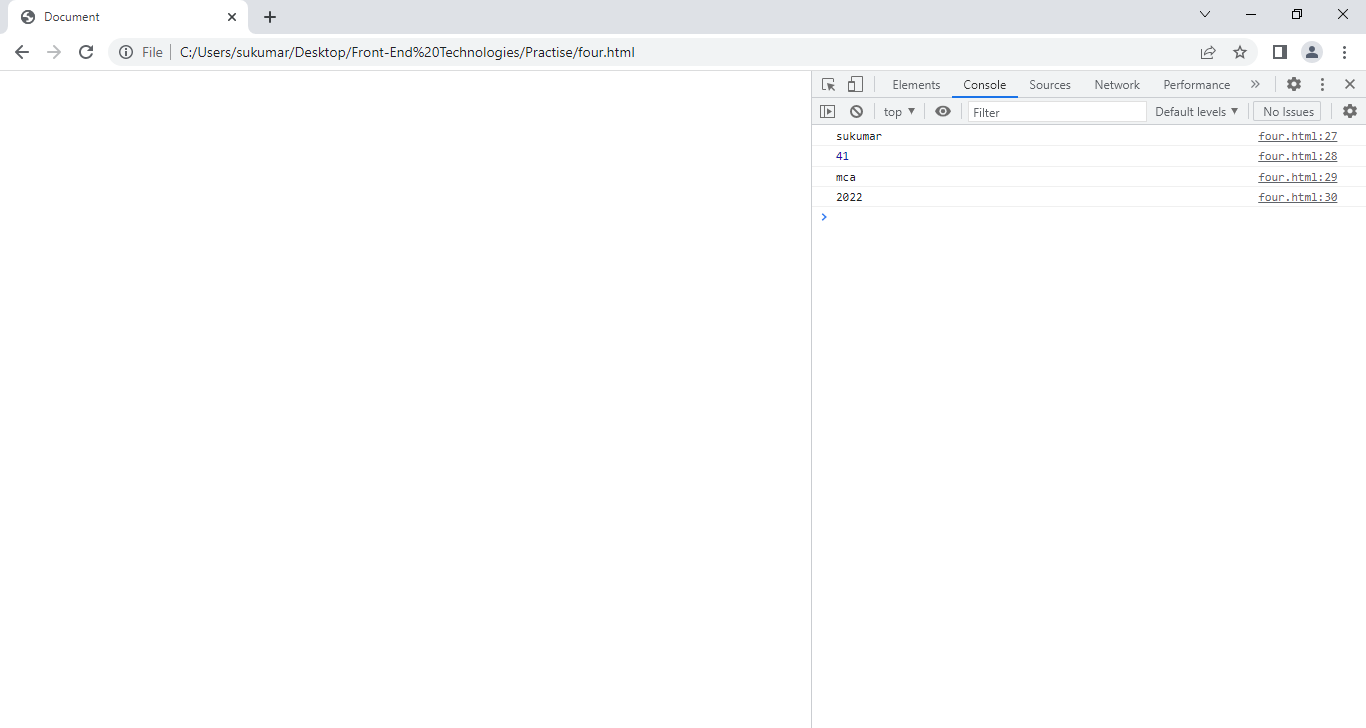
  vars1=new Child('sukumar',41);

  s1.display();

  </script>

</body>

</html>



Example:2

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<style>

    </style>

<body>

  </div>

  <script>

  classParent{

         constructor(x,y){

         this.branch=x;

         this.year=y;

         this.name='PPGC';

          }

         show(){

          console.log(this.branch);

          console.log(this.year);

         }

  }

  classChildextendsParent{

    constructor(a,b,c,d){

      super(c,d);

      this.name=a;

      this.age=b;

    }

    display(){

      console.log(this.name);

      console.log(this.age);

      super.show();

    }

  }

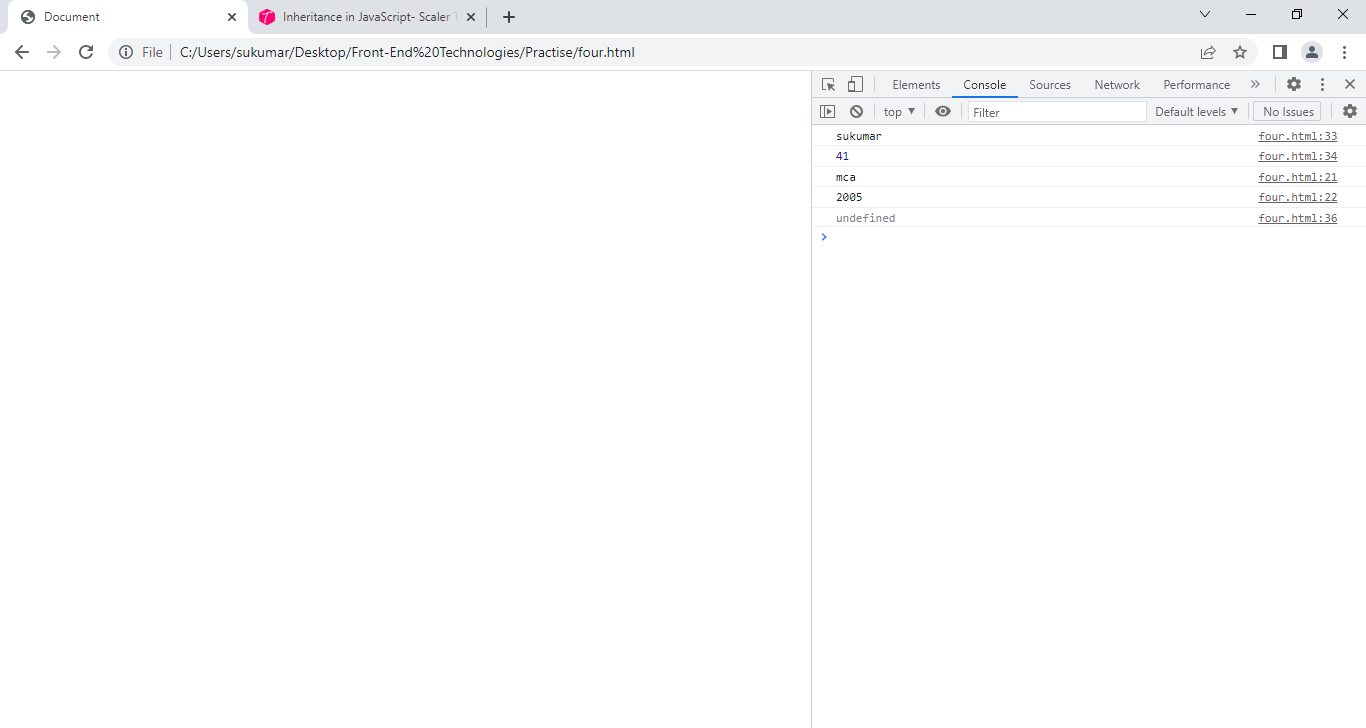
  vars1=new Child('sukumar',41,'mca','2005');

  s1.display();

</script>

</body>

</html>



**4.Exception Handling:**

4.1.Error:- An **error** describes any issue that arises unexpectedly that cause a program to not function properly.

4.2.Types of Errors:-There are three types of errors in programming:

* Syntax Errors
* Runtime Errors
* Logical Errors.

A. Syntax Errors:- Syntax errors, also called **parsing errors. Error** that occur when we don’t follow rules of writing language syntax is called syntax error. The syntax error occurs at compile time in traditional programming language(c,c++) and at interpreting time in java script.

Example: 1. missing the assignment operator between var-name and value.

2. missing the open/close parenthesis.

B. Logical errors:- On execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appears to be error free are called logical errors.

Example: Input values are 4,5.

We expect 9 as output. But program gave output 1.

Rong output has been got because of logical error in program.

C.Runtime error:- Errors which occur during program execution(run-time) after successful compilation/Interpreation are called run-time errors. The runtime error abnormally stops program execution.

Example:

Lack of memory in RAM.

The runtime error is also called ‘Exception’.

The rumtime error can be handled by exception handling mechanism. This mechanism is implemented using try,catch,finally statements and throw operator.

**When runtime error occurs,JS engine create Error object and JS stores error into in that object. JS engine looks for user-defined exception handling mechanism in program. If program has it then JS engine submit run time error to it otherwise it is handled by pre-defined exception handling mechanism in JS engine.**

C.1. try:-

Syntax: try{ statement[s];}

In try block, we write statements that may lead to run time error.

Note: once execution control is thrown to catch statement from try block because of error, It does not return to try block. Therefore some statements may not be executed.

C.2. catch:-The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

Syntax:

catch(err){ block of code to handle error;}

Note:- One or more than one catch statement must follow the try statement.

C.3.finally:-It is optional statement. The final block is unconditionally executed after try/catch statement.

Syntax:

finally{ statement[s];}

C.4.throw:- It is operator. It is used to raise customized exception or built-in Exception. The raised exception can be caught at catch statement. We may write a statements in catch block to take appropriate action. When throw statement is executed, the statements present after it will not execute.

Syntax:

throw(‘string/number/object’);

C.5. Error Object:- when runtime error occurs,The error information is kept in error object.

It has only two properties.

1.name:set/return error name.

It returns one of following errors.

a)EvalError: error occurred in eval()function.In modern browsers , it is treated as syntax error.

b)rangeError: that occurs when a numeric variable or parameter is out of its valid range.

Ex: occurs, when we create illegal no.of elements. And pass illegal values to numeric methods(toFixed(),toPrecision()…etc).

c)SyntaxError:

Ex:-if(){

There is no closed brace. That is syntax error.

d)ReferrenceError:- A “ReferenceError” exception is thrown when a non-existent variable is accessed.

Ex: var I;

I=j+10;// where j is illegal reference because it has not been declared.

e)TypeError:- A “TypeError” exception occurs when a value is not of the expected type..

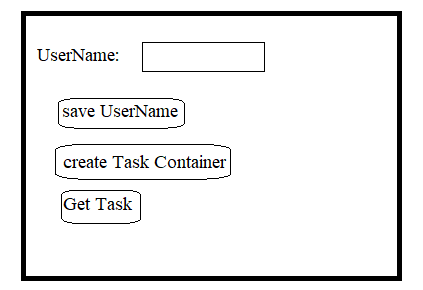
Vari=10;

Document.writeln(i.toUpperCase());

f)URIError:- An error in encodeURI() has occurred.

2. message: set/return description of error.

Example:



1. When we click username, user name and pressing time should be saved.
2. After pressing second button, The empty rectangle box must be displayed.
3. To display task inside the box, we should press get task.
4. After displaying task and after some delay, user should see the ‘thanks msg’.

Note: - User can only get task , after generating task container.

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" cofntent="width=device-width, initial-scale=1.0">

  <title>Document</title>

  <style>

    ul{

      list-style-type:none;

      border:2px solid blue;

      margin:25px0px;

      position:relative;

      left:25vw;

      width:50vw;

      padding-left:10px;

    }

    ul>li

    {

      margin:5px5px;

      padding-bottom: 5px;

      border-bottom:0.25px solid ;

    }

    div{

      width:50vw;

      position: relative;

      left:25vw;

      border:0.5px solid green;

      background-color: white;

      height:50px;

      text-align: center;

      margin:30px0px;

      background-color: yellow;

      display:none;

      padding:10px;

      font-size: 1.2rem;

      color:blue;

    }

  </style>

</head>

<body>

  <ul>

    <li>UserName:

        <inputtype="text" required>

    </li>

    <li>

      <buttononclick="save(event)">SaveUserName</button>

    </li>

    <li>

      <buttononclick="generate(event)">Generate Task Container</button>

    </li>

    <li>

      <buttononclick="display(event)">Display Task</button>

    </li>

  </ul>

  <div>

  </div>

  <div>

  </div>

  <script>

    vartaskList=['Read English','Meet Chairman','Spend with children','Goto Nellore','Upload Bio-Data','Apply For Job','Swim ToDay Evening','Eat Lunch','Complete Work Fastly','ping in WhatsApp'];

    varuserData=[{}];

    varbtnRef=document.querySelectorAll('button');

    var  uName,con;

    functionsave(a){

      btnRef[0].disabled=true;

      uName=document.querySelector('input').value;

      xy=new Date();

      xy=xy.getDay()+'-'+xy.getMonth()+'-'+xy.getFullYear();

      userData.user\_name=uName;

      userData.log\_Date=xy;

    }

    functiongenerate(a)

    {

      btnRef[1].disabled=true;

      con=document.querySelector('div');

      con.style.display='block';

    }

    functiondisplay(a)

    {

      btnRef[0].disabled=false;

      btnRef[1].disabled=false;

      vari=Math.random()\*10;

      i=Math.floor(i);

      try{

        document.querySelector('input').value='';

        con.innerHTML='Dear'+''+uName+''+'Your Task is:'+''+taskList[i] ;

      }

      catch(e){

        window.alert('Dear Boss, First Box has to be generated');

      }

      finally{

      setTimeout(rock,5000);

      }

    }

    varrock=()=>{

      vardivEles=document.querySelectorAll('div');

      divEles[1].style.display='block'

      divEles[1].innerHTML='Thanks Dear User';

      setTimeout(rock1,5000);

    }

    varrock1=()=>{

      vardivEles=document.querySelectorAll('div');

      divEles[1].style.display='none';

      divEles[0].style.display='none';

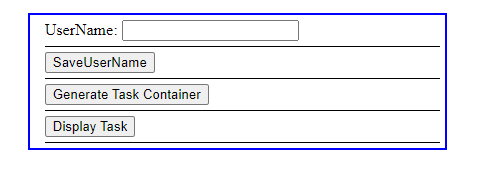
    }

  </script>

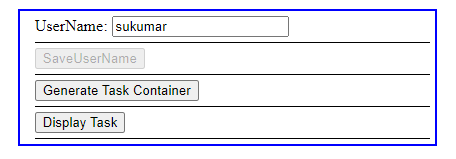
</body>

</html>

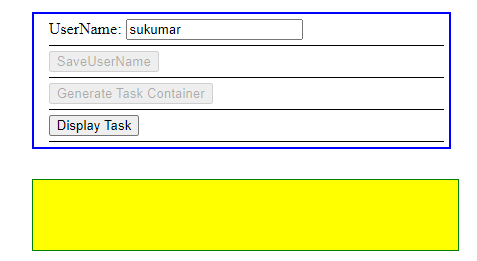
**Execution:1**

****

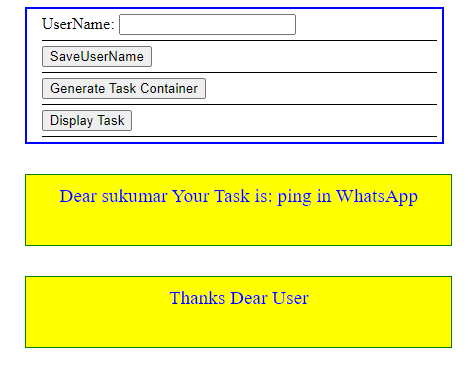
Entered user name and clicking the ‘save user name’ button.

****

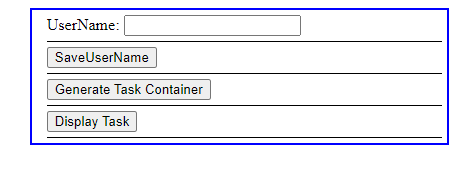
Press the Generate Task container.

****

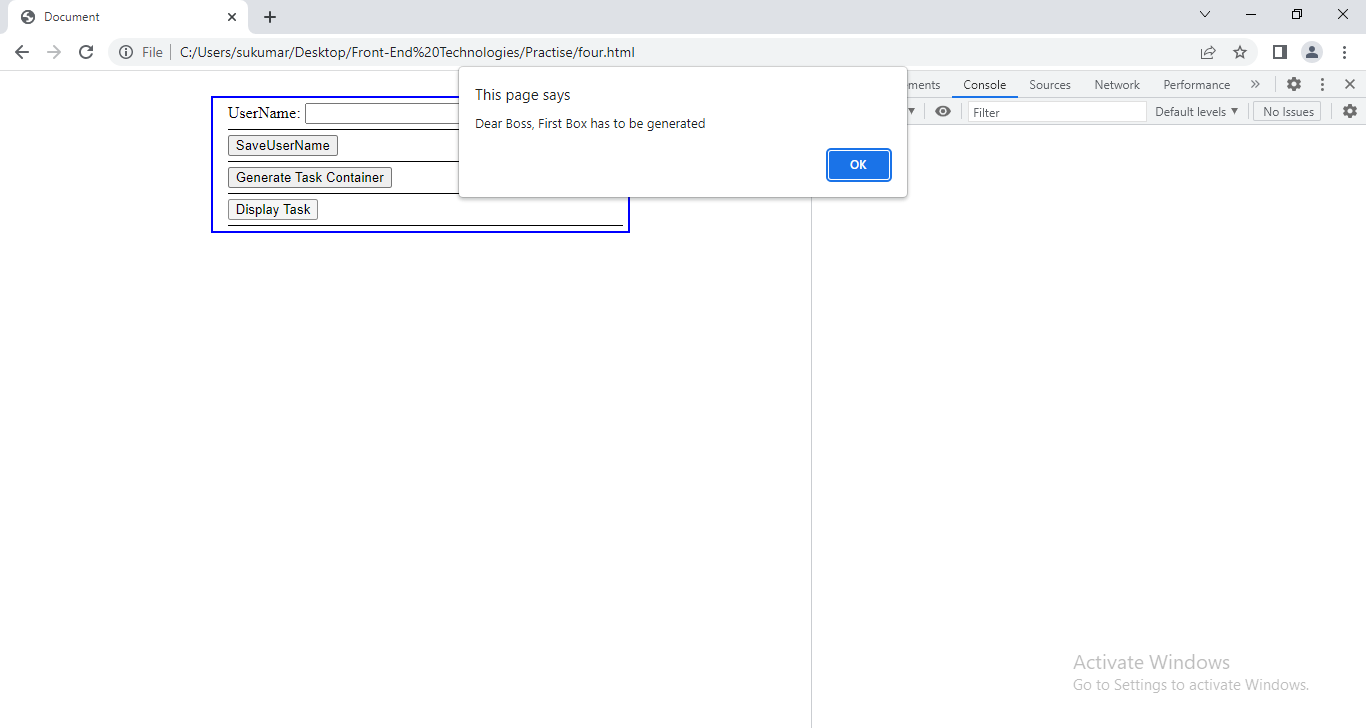
**Press the ‘Display Button’**

****

Execution:2



Press Display Task



4.3.User Defined Error:-

Step1: create Error object.

Step2: throw it when error arises using throw keywork.

4.3.1Throws:-It is used to raise customized exception or built-in Exception. The raised exception can be caught at catch statement. We may write a statements in catch block to take appropriate action. When throw statement is executed, the statements present after it will not execute.

Syntax: throw(‘string/number/object’);

Example:In your program, you want to register people only above 18, and you want to throw an exception if someone enters a number below 18. This is done through a throw statement.

<!DOCTYPEhtml>

<htmllang="en">

<head>

  <metacharset="UTF-8">

  <metahttp-equiv="X-UA-Compatible" content="IE=edge">

  <metaname="viewport" cofntent="width=device-width, initial-scale=1.0">

  <title>Document</title>

  <style>

    ul{

     position:relative;

     left:25vw;

     width:50vw;

     height:20vh;

     border:1px solid green;

     list-style-type:none;

     padding:0px;

     box-sizing: border-box;

    }

    ul>li>input{

      height:2rem;

      width:75%;

      margin:5px12.5%;

      border-bottom: 0.5pd solid;

    }

    button{

      width:25%;

      text-align: center;

      margin:2px33%;

      height:1.5rem;

      background-color: red;

    }

  </style>

<body>

  <ul>

    <li><inputtype="text" placeholder="Age" onblur="rock()"></li>

    <li><inputtype="text" placeHolder="UserName"></li>

    <li>

      <button>submit</button>

    </li>

  </ul>

  <script>

    varele1=document.querySelector('input');

    functionrock(){

      vare1=new Event('age');

      e1.name='Dear User';

      e1.message='Your are minor';

      varageVal=Number(ele1.value);

      console.log(ageVal);

      if (ageVal<18)

      {

        try{

        throwe1;

        }

       catch(e1)

       {

          alert(e1.name+':'+e1.message);

       }

    }

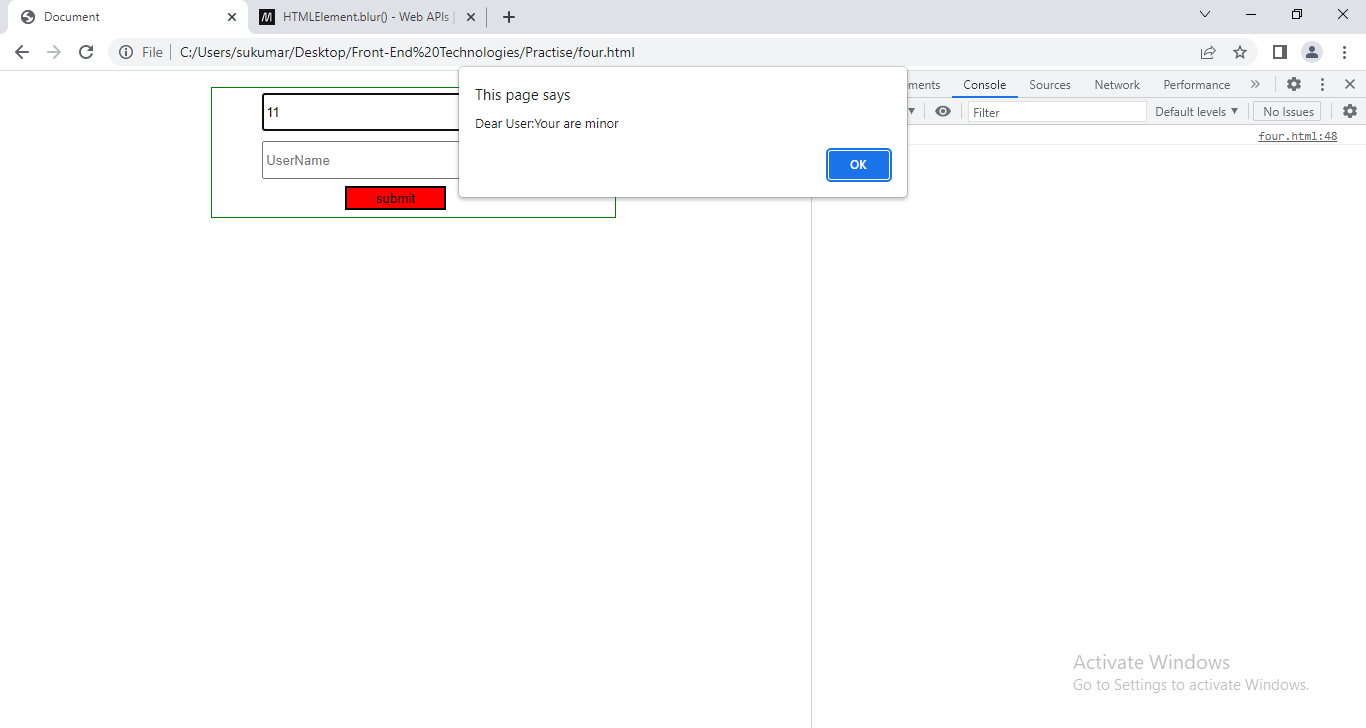
  }

  </script>

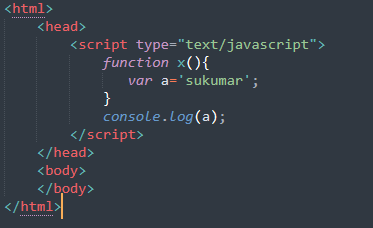
</body>

</html>

Output:

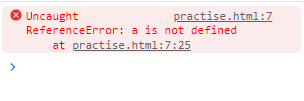


**5.Closure:-**

****

**Output:**

**=====**

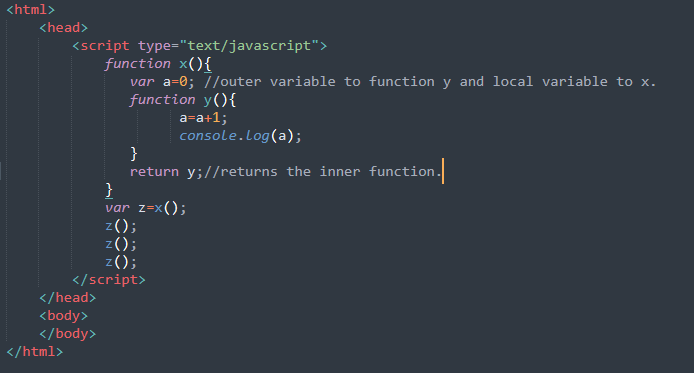
****

The variable named “a” had local scope. When x() execution completed, a is deleted. So It is not visible and accessible out side the x().

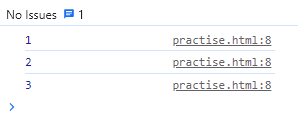
Definition:- Accessing variables of function even its execution completes or out of its scope only through its inner function is **“closure”**.

* Outer Variables can keep their states between function calls. Inner function does not keep the separate copy of outer variables but it reference outer variable,that means value of outer variables will be changed if you change it using inner function.
* Closure is useful to create private variables in function.

Example:-



Output:



Usecase:-

