**Contents**

1. ECMA

2. Let

3. Const

4. String Interpolation

5. Default Arguments.

6. Rest Parameter

7. IIFE

8. \*\* operator

9. New Concepts related to Array.

10. Import & Export

11. Set

12. map

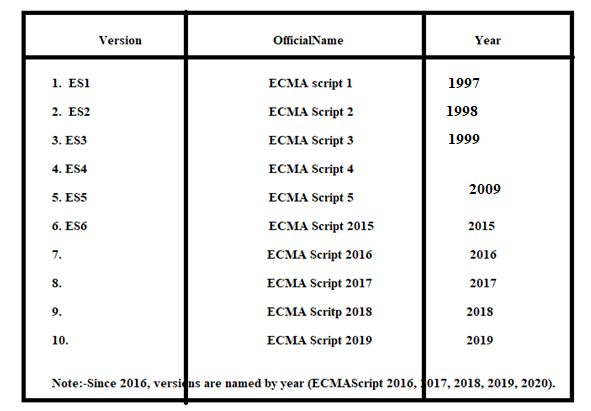
13. JS Code Minification

14. Async/await

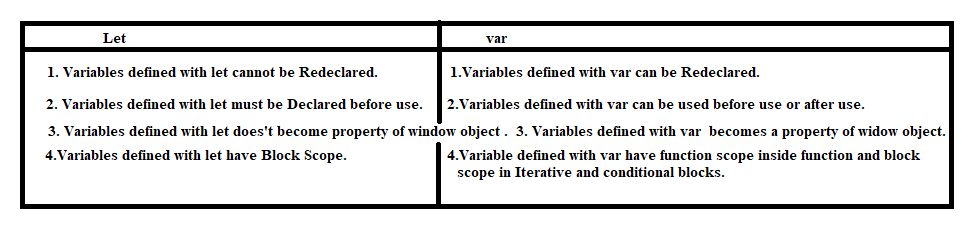
15. Promises

**1.ECMA:** Ecma International (formally **European Computer Manufacturers Association**) is a non-profit organization that develops standards (Rules) in computer hardware, communications, and programming languages.

The ECMA provided one specification/standard . That specification name is ECMA Script. The Java Script is implementation of that ECMA script standard. ECMA has been releasing several versions of ECMA Script since 1997.



**2.Let:**



**Example:1**

<script>

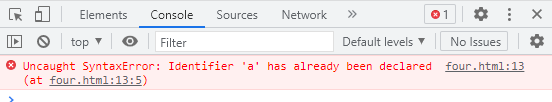
leta=50;

letb=60;

leta=100;

console.log(a);

</script>



Example:2

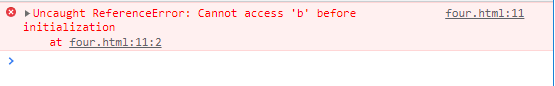
<script>

b=20;

letb;

console.log(b);

</script>



Example:3

<script>

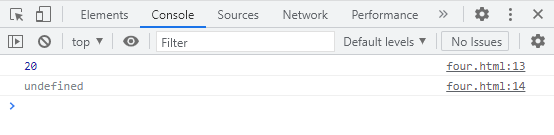
varx=20;

lety=40;

console.log(window.x);

console.log(window.y);

</script>



Example:4 In modern scripts, the var is not used to declare variables. Because it behavies differently in function block and conditional statement block.

<script>

vara=20;

if(a>14){

 varc=50;// c has block scope. It can be accessed with in block and outside the block.

}

console.log("Variable-a has block Scope:"+ c);

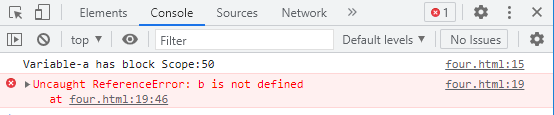
functionsample(){

  varb=30;// b has function scope. It can only be accessed with sample function.

}

console.log("Variable-b has Function Scope:"+b);

</script>



Exampl:5 Modern Scripts , let is used to declare variables. Because variable has only block scope. It behaves samely in function block and conditional statement block.

<script>

leta=20;

if(a>14){

 letc=50;// c has block scope. It can be accessed with in block .

}

console.log("Variable-a has block Scope:"+ c);

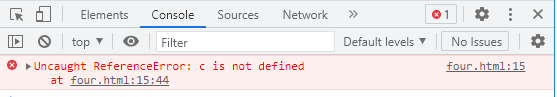
functionsample(){

  letb=30;// b has block scope. It can only be accessed with sample function.

}

console.log("Variable-b has Function Scope:"+b);

</script>



**3.const: It is a keyword.**

3.1.Variables defined with const cannot be Redeclared.

3.2.Variables defined with const cannot be Reassigned.

3.3.Variables defined with const have Block Scope.

Syntax: const var-name=value;

**4. string interpolation:** The string interpolation is process of embedding the variable,function call or arthimetic expressions in string.

Syntax:

` String….${variable/functioncall/arithmetic expression} string continuation`.

Example:

<script>

qual='MCA';

functionageCal(){

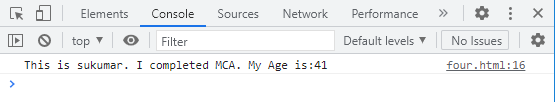
  return41;

}

vardis=`This is sukumar. I completed ${qual}. My Age is:${ageCal()}`;

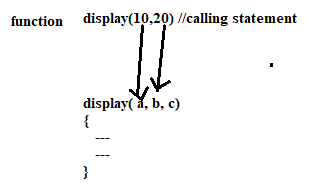
console.log(dis);

</script>



**5. Default Arguments:** If no value/argument is came from function calling sttement then ‘undefined’ value is initialized in parameter in function definition.

Example:



C has not get value from calling statement. Therefore it was initialized with ‘undefined’ value.

<script>

functiondisplay(a,b,c)

{

  console.log(a);

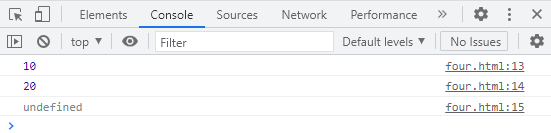
  console.log(b);

  console.log(c);

}

display(10,20);

</script>



Where parameter default value is ‘undefined’. We can change this default value using following syntax.

Syntax:

Function name(par1[=defaultvalue1],par2[=defaultvalue2]…)

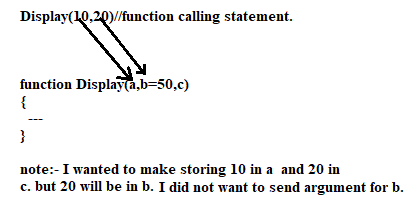
{

----

}

Note:- It is better to write all default parameter and their values extreme right side. Otherwise, argument may be stored in incorrect parameter.

Example:



Example:

<!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>

</head>

<body>

<script>

classEmployee{

  constructor(a,b,c,d='Account'){

    this.id=a;

    this.name=b;

    this.salary=c;

    this.dept=d;

  }

}

vare1=new Employee(1,'suku',7000);

vare2=new Employee(2,'samba',5000);

varemp=[e1,e2];

for(leti of emp){

  console.log(i.id);

  console.log(i.name);

  console.log(i.salary);

  console.log(i.dept);

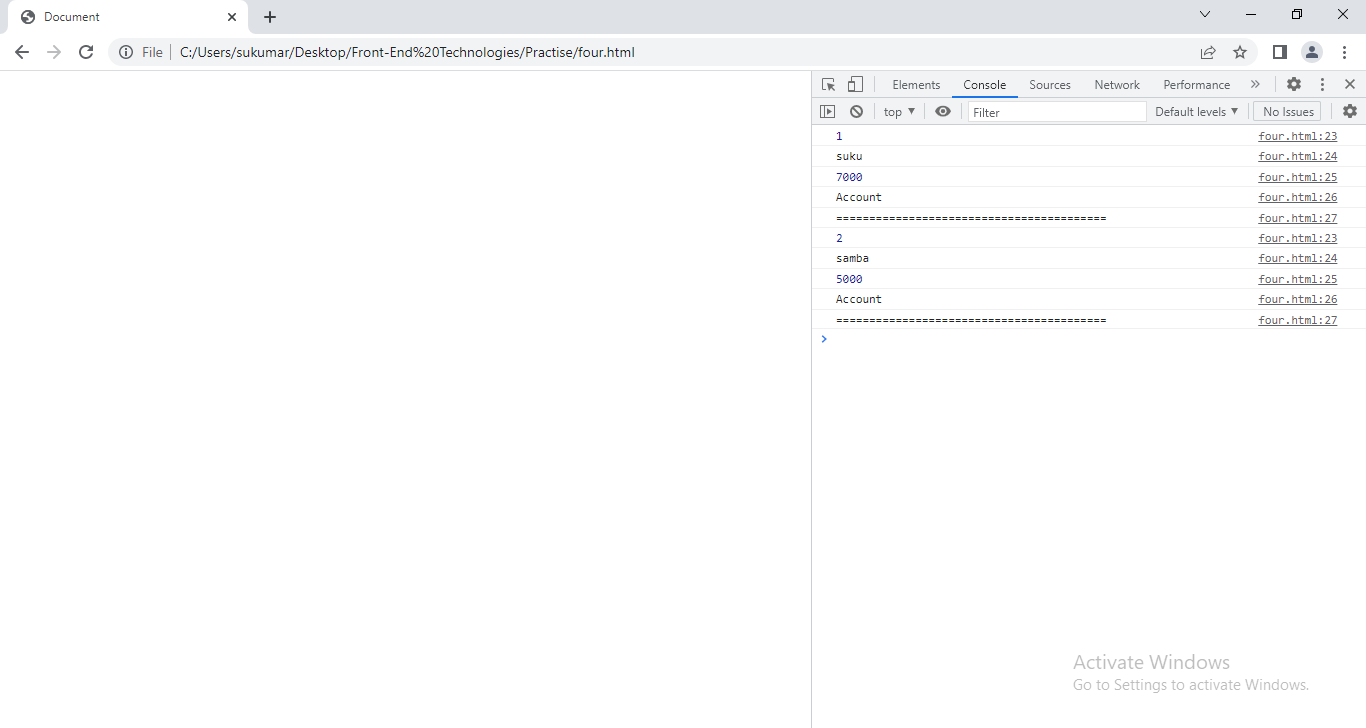
  console.log("=========================================");

}

</script>

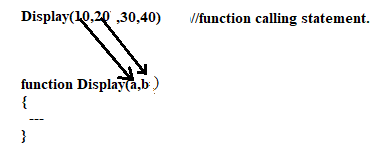
</body>

</html>



**6.Rest Parameter:**We can pass enormous arguments to functions. Although we don’t get error.

Example:



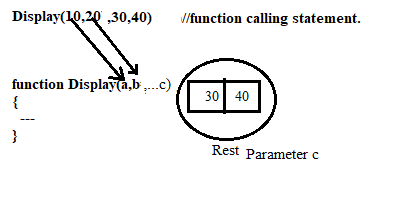
In our example, display can only receives two arguments. But we have passed more than two arguments. Although we don’t get error and function don’t consider remaining arguments.

To consider remaining arguments, we put rest parameter in function definition. The rest parameter

syntax :

… varname

* The Rest parameter should be last parameter in function definition.
* The rest parameter is Array which stores the remaining arguments.



Example:

<!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>

</head>

<body>

<script>

classEmployee{

  constructor(a,b,...c){

    this.id=a;

    this.name=b;

    this.workLocations=new Array();

    for(leti of c)

    {

      this.workLocations.push(i);

    }

  }

}

vare1=new Employee(1,'suku','nlr','clx','gdr');

vare2=new Employee(2,'samba','nlr','clx');

varemp=[e1,e2];

for(leti of emp){

  console.log(i.id);

  console.log(i.name);

  console.log(i.workLocations);

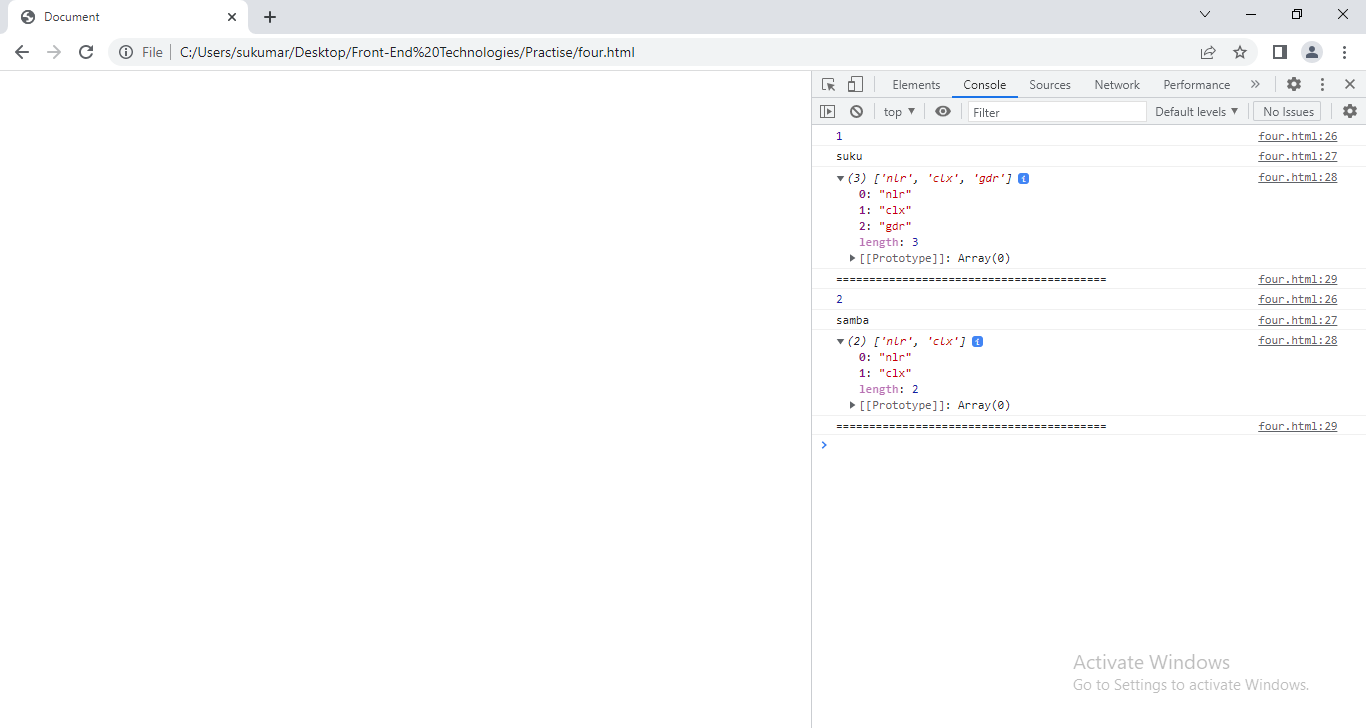
  console.log("=========================================");

}

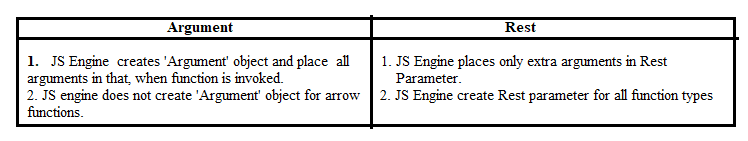
</script>

</body>

</html>



6.1.Difference Between Argument object and Rest parameter.



Example:

<!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>

</head>

<body>

<script>

varx=(a,b,c)=>{

for(leti in arguments)

{

  console.log(arguments[i]);

}

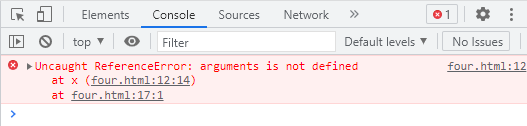
}

x(10,20,30);

</script>

</body>

</html>



7.IIFE(Immediately invoked function Expression):The function is invoked when it is encountered. We don’t need to explicitly write calling statement.

Syntax:1

(function([parameter[s]]){body})([parameter[s]));

Syntax:2

(([paramaters])=>{})([parameters]);

<!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>

</head>

<body>

<script>

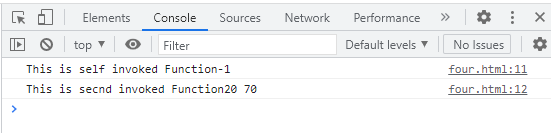
(function(){console.log('This is self invoked Function-1');})();

(function(a,b){console.log(`This is secnd invoked Function${a} ${b}`);})(20,70);

</script>

</body>

</html>



**8.\*\* operator:-**The exponentiation operator ( \*\* ) **returns the result of raising the first operand to the power of the second operand**. It is equivalent to Math. pow , except it also accepts BigInts as operands.

Example:

Console.log(2\*\*3);

Output:8.

**9. New Concepts related to Array:-**

**9.1.ForEach():-**The forEach() method calls a function for each element in an array.The forEach() method is not executed for empty elements.

**Syntax:**

Array-name.Find(callback-function,thisvalue);

Function callback-function(value1[,b,arr])

{

}

Where value1 has value of array and b has index of value.

Example:

<!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>

</head>

<body>

<script>

varabc=[10,20,30,40,50];

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

for(leti in abc)

{

  console.log(i+':'+abc[i]);

}

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

for(leti of abc)

{

  console.log(i);

}

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

letsum=0;

abc.forEach(suku);

functionsuku(a,b)

{

  sum=sum+a;

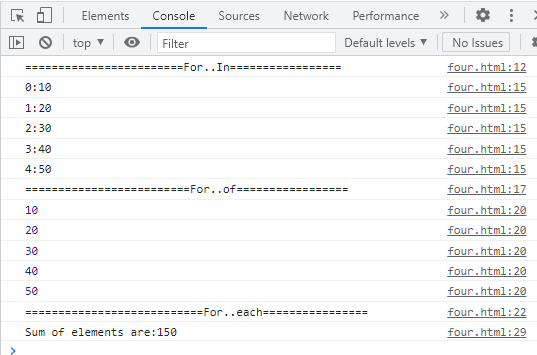
}

console.log('Sum of elements are:'+sum);

</script>

</body>

</html>



9.2.Find:-

Syntax:

Array-name.Find(callback-function,thisvalue);

Function callback-function(value[,index,arr])

{

}

Value is value of current element. Index is index of current element.

Arr is array of current element.

The second argument of find method(thisvalue) is accessed in callback-back function using ‘this’.

The find() method executes a function for each array element. The find() method returns the value of the first element that passes a test.The find() method returns undefined if no elements are found.The find() method does not execute the function for empty elements.The find() method does not change the original array.

<!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>

</head>

<body>

<script>

varabc=[10,50,30,40,50];

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

letres=abc.find(abcd,50);

functionabcd(value,index)

{

  if(value==this)

  {

    returnvalue;

  }

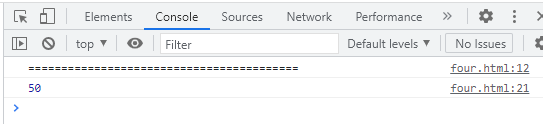
}

console.log(res);

</script>

</body>

</html>



9.3findIndex:-Array-name.findIndex(callback-function,thisvalue);

Function callback-function(value[,index,arr])

{

}

Value is value of current element. Index is index of current element.Arr is array of current element. The second argument of find method(thisvalue) is accessed in callback-back function using ‘this’.

The findIndex() method executes a function for each array element. The findIndex() method returns the index of the first element that passes a test.The findIndex() method returns -1 if no elements are found.The findIndex() method does not execute the function for empty elements.The findIndex() method does not change the original array.

Example:

<!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>

</head>

<body>

<script>

varabc=[10,40,30,40,50];

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

letres=abc.findIndex(abcd,50);

functionabcd(value,index)

{

  if(value==this)

  {

    returnvalue;

  }

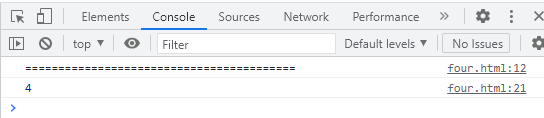
}

console.log(res);

</script>

</body>

</html>



9.4.Array.isArray:-

Syntax:

Array.isArray(arg1)

If arg1 is array then it returns true otherwise returns false.

**10.import & Export:** we can break up script code in to modules. Module is nothing but a ‘separate .js file’. Module has a variables,functions,classes..etc. By breaking code as modules, programmer can easily maintain the code. JS module rely on export and import keywords.

10.1. Export: The variables,function,classes …etc are exported from module either individually or all at once.

10.1.1 Individually:

Syntax:

Export let/var Var-name ;

Export function fun-name([parameters]){}

Export class class-name{}

10.1.2. At Once:

Syntax:

Export{ ele1,ele2,…n};

10.2. Import:- we can import modules into a file in 2 ways.

10.2.1. Import only necessary elements:

Syntax:

Import {ele1,ele2,….} from “address of module”;

10.2.2. Import all:

Syntax:

Import \* as obj-name from “address of module”;

Note:-Modules only work with the HTTP(s) protocol.A web-page opened via the file:// protocol cannot use import / export.

<script type=’module’></script>

Example:

1.four.html

<!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>

</head>

<body>

<scripttype="module">

  import\*asus1from "./util.js";

  import {Arithmetic1} from "./util1.js";

  letxy=new us1.Arithmetic();

    console.log("sum:"+xy.add(20,30,40,60));

  console.log("Difference:"+xy.sub(20,30,40,60));

  letxy1=new Arithmetic1();

  console.log("Product:"+xy1.mul(10,20,30));

</script>

</body>

</html>

2.util1.js

classArithmetic1{

   mul(a,b,...c)

{   letsum;

   if(c.size==0)

   {

       sum=a\*b;

   }

   else

   {

        sum=a\*b;

       for(letx of c)

       {

           sum=sum\*x;

       }

   }

   returnsum;

}

}

export {Arithmetic1};

3.util.js

 letuName;

 classArithmetic{

    add(a,b,...c)

{   letsum;

    if(c.size==0)

    {

        sum=a+b;

    }

    else

    {

         sum=a+b;

        for(letx of c)

        {

            sum=sum+x;

        }

    }

    returnsum;

}

    sub(a,b,...c)

    {

        letdiff;

    if(c.size==0)

    {

        diff=a-b;

    }

    else

    {

         diff=a+b;

        for(letx of c)

        {

            diff=diff-x;

        }

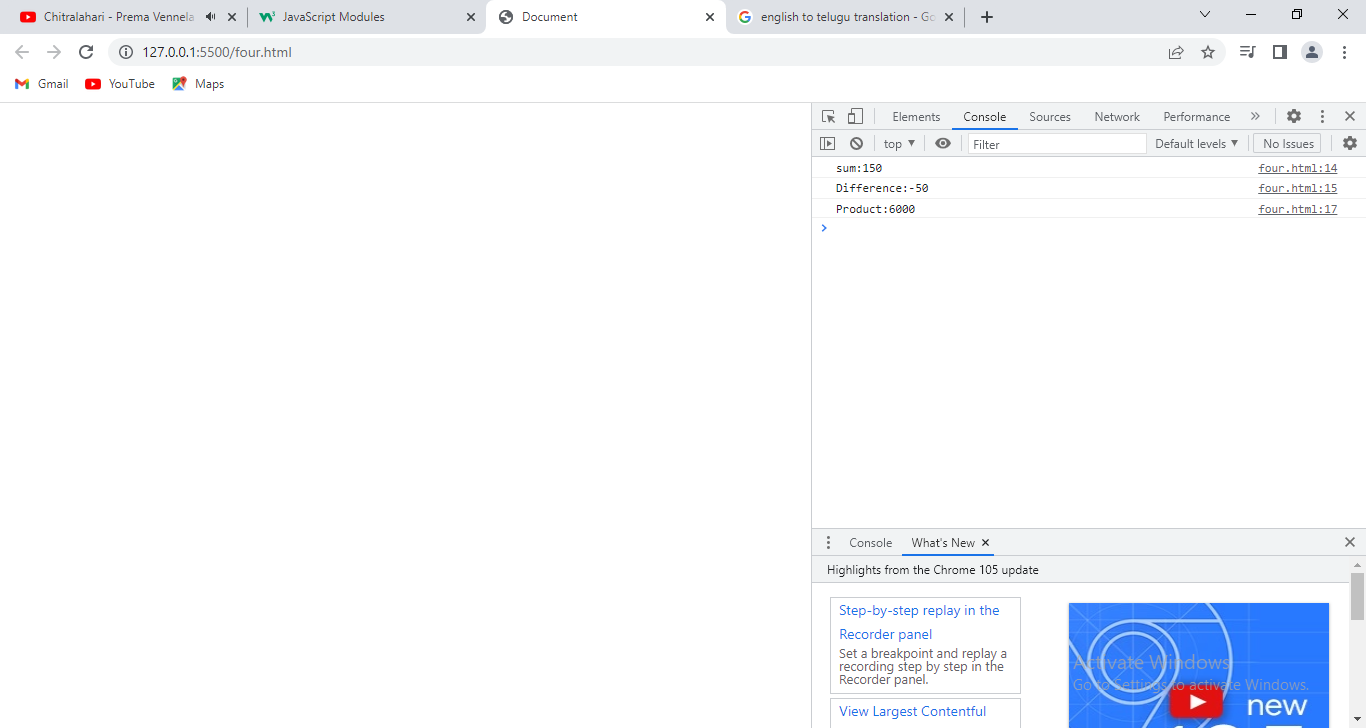
    }

    returndiff;

    }

}

export {uName,Arithmetic};



**11. Set:**The **Set** object lets you store unique values of any type, whether [primitive values](https://developer.mozilla.org/en-US/docs/Glossary/Primitive) or object references.

11.1 Creating Set:-

Syntax:1

Let/var var-name=new Set([v1,v2,v3]);

Syntax:2

Let/var var-name=new Set();

Var-name.add(v1).add(v2)….etc // chaining method.

Var-name.add(v1);

…etc.

11.2. properties & methods:

a.size:- it returns number of elements of set.

b. add(v1):- It add element to set.

c.delete(v1):- It delete element from set.

d. clear():- it removes all elements from set.

f. has(v1):-It returns true ,it set has v1. Otherwise returns false.

g.forEach():- It invokes the callback for each element of set.

h.values():- It returns iterator with all values of set.

i. entires():- It returns iterator with[value,value] pairs from set.

Example:

<!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>

</head>

<body>

<script>

    vardata=new Set([10,'suku',20,'sv',35.7]);

    for(leti of data)

    {

      console.log('set Element is'+':'+i);

    }

    data.add(80);

    console.log("=========After Adding Element, Set is:================");

    for(leti of data)

    {

      console.log('set Element is'+':'+i);

    }

    data.delete(80);

    console.log("=========After Deleting Element, Set is:================");

    for(leti of data)

    {

      console.log('set Element is:'+':'+i);

    }

    console.log('Does set has element sv?:'+data.has('sv'));

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

    varx=data.values();

    for(leti of x)

    {

      console.log(i);

    }

    console.log("=================Entries Function=======================")

    varx=data.entries();

    for(leti of x)

    {

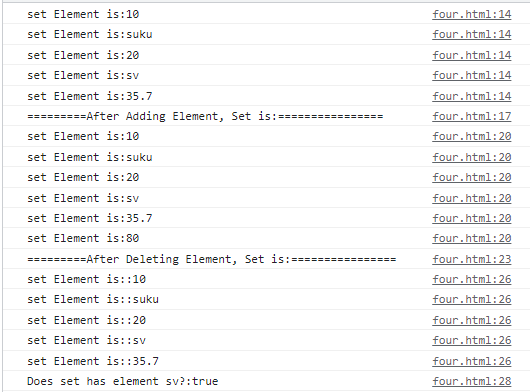
      console.log(i);

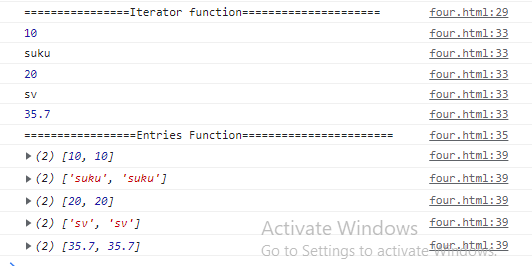
    }

</script>

</body>

</html>





**12.Map:**The map is collection of key/value pairs. The difference between object and map is object allows key of string type where as **map allows key of any type**.

12.1.properties:

a.size: It returns current element count.

12.2 .Methods:

a.set(key,value): It stores key and value in map.

b.get(key): It returns value of key.

c. has(key):- It returns true, if map has key. Otherwise returns false.

d.delete(key):- It remove value by key,if key exist in map.

e.clear():- It removes the everything from map.

f. keys():- It returns iterable for keys.

g. values():- It returns iterable for values.

h.entries():- It returns iterable for entries [key,value].It is used by default in for..of.

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>

<body>

    <script>

        varm1=newMap();

        m1.set('uname','suku');

        m1.set('age',41);

        m1.set('qual','mca');

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

        console.log('Size of Map:'+m1.size);

        console.log('Does map has age key?'+m1.has('age'));

        console.log('Does map has exp key?'+m1.has('exp'));

        console.log('Getting value age of sukumar:'+m1.get('age'));

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

        for(iofm1.keys())

        {

        console.log(i);

        }

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

        for(iofm1.values())

        {

            console.log(i);

        }

        console.log("====================keys and values=================");

        for(iofm1.entries())

        {

            console.log(i[0]+':'+i[1]);

        }

        console.log("====================Another Map Object created with existing JSObject===============");

        letd1={'branch':'mca','loc':'nlr'};

        letv1=newMap(Object.entries(d1));

        for(letzofv1)

        {

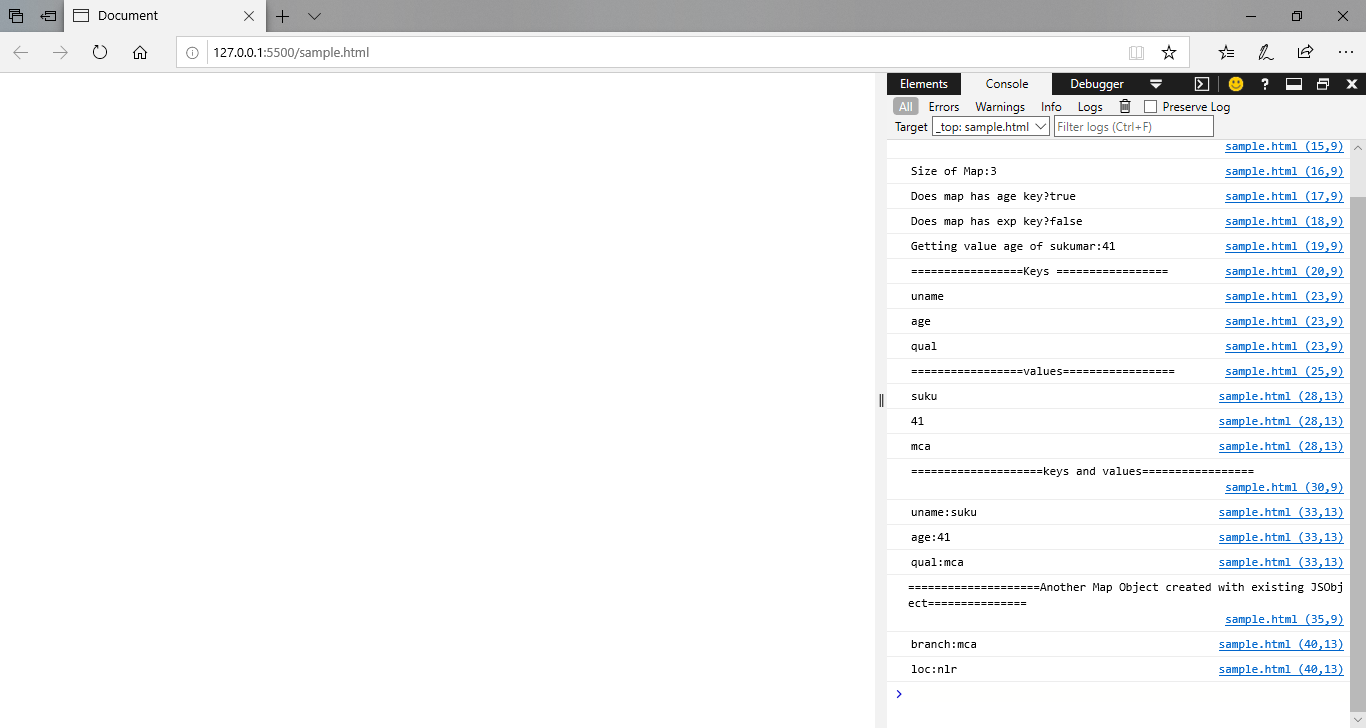
            console.log(z[0]+':'+z[1]);

        }

</script>

</body>

</html>



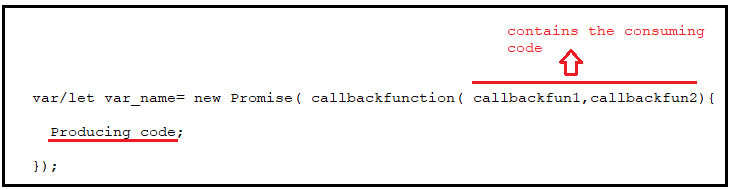
**13.JSCode Minification:**Minification, also known as minimization, is the process of removing all unnecessary characters from JavaScript source code without altering its functionality. This includes the removal of whitespace, comments, and semicolons, along with the use of shorter variable names and functions.

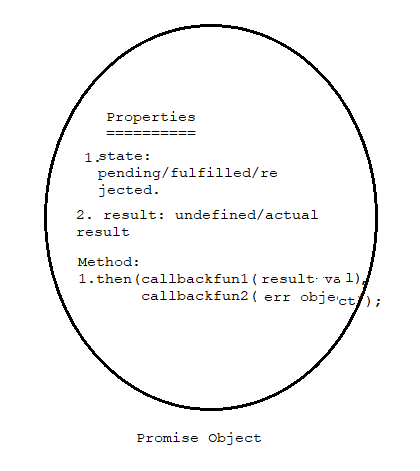
Use JavaScript minifier and compressor tool to reduce java script code to reduce code size and make your website load faster.

Example: JS Min , YUI compressor.

**14.Promises:**

* The promise is java script object.
* The promise object contains the producing code( one call back function) and consuming code(two call back functions(success,fail)).
* The promise links the producing code and consuming code.





* The promise object can be in one of 3 states:
* Pending.(if the result is undefined)
* Fulfilled(if promise is fulfilled).
* Rejected(if promise is not fulfilled).

**Note:**- we can’t access state and result properties.

We use “then” method to handle the promises.

**then**(callbackfun1(result val){}, callbackfun2(error obj){});

First callbackfun1 for success.

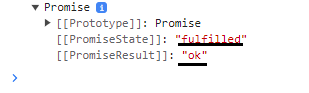
Second callbackfun2 for failure);

Both are optional.

Example:1-

****

**Output:**



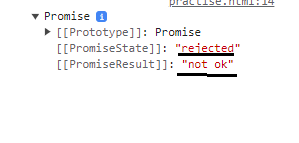
**Example:2**

**========**

****

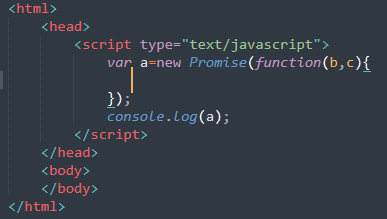
**Output:**

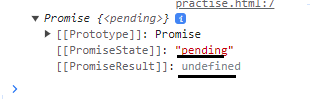
**----------**



Example:3

=======



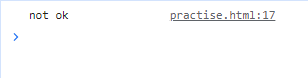


Example:4

=======

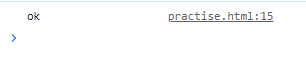


Run1:





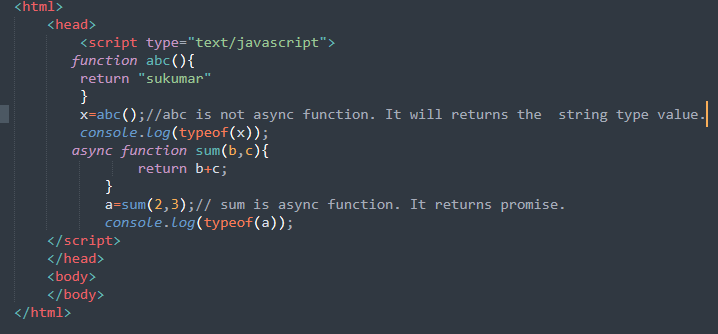
output:



**15. Async/await:**

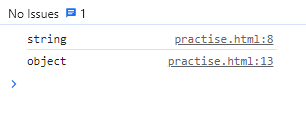
**15.1.Async**:- The keyword **“async**” before a function makes function return promise:

Example:

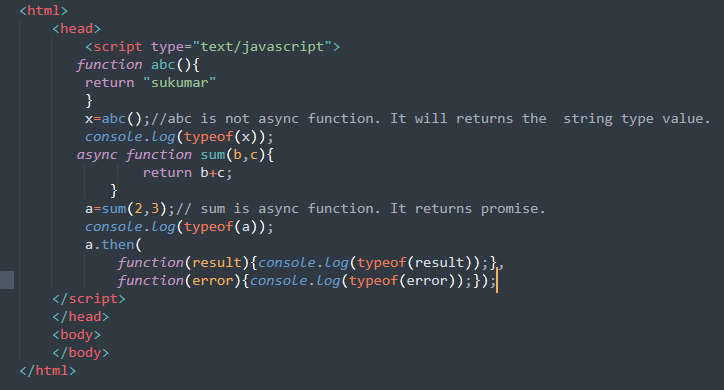


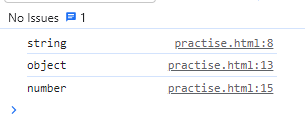
Output:

=====



2.





15.2. await:- await is keyword. It is only used inside the async function to wait for the asynchronous operation.

Example:

1. Let result=await promise;

Await pauses the async function until promise returns a result(resolve or reject) value.

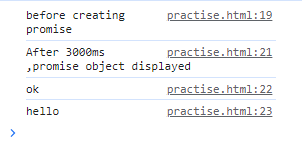
2. let sum1= await a+b;

Await pauses the async function until a+b calculation completed.

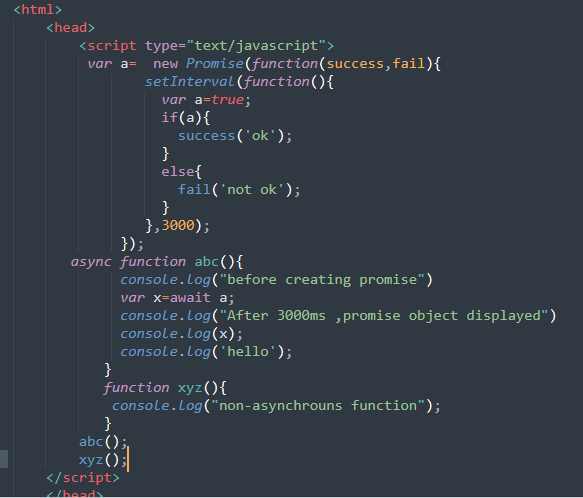
Example1:-



Output:-



Example:2 while waiting for asyn function completion, the JS engine can executes other non-async or async function.



Output:-

