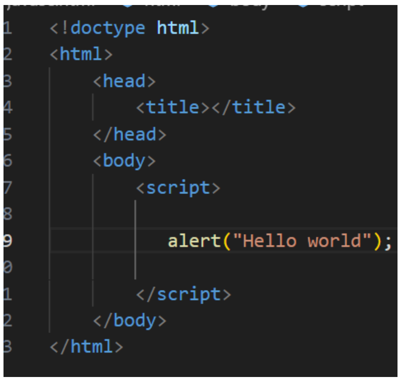
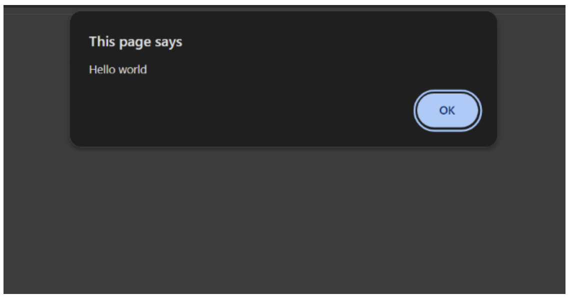
Task 1: Write a simple script that displays "Hello, World!" on the web page using an alert box



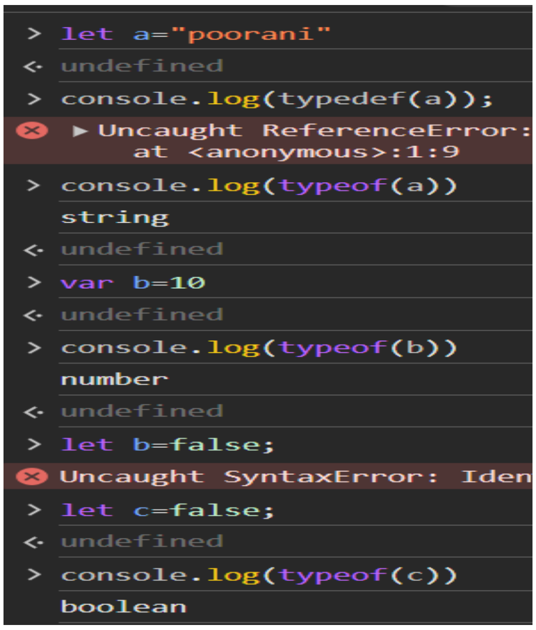
```html
<!doctype html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <script>

            alert("Hello world");

        </script>
    </body>
</html>
```



This page says

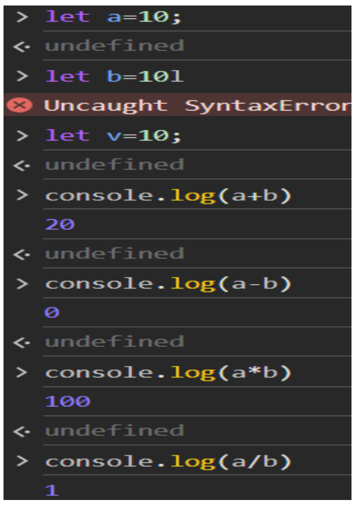
Hello world

Task 2: Experiment with different data types in JavaScript (e.g., string, number, boolean) by declaring and logging them in the console.



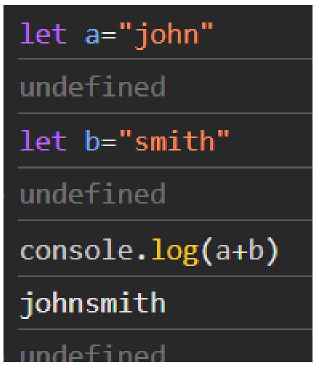
```
> let a="poorani"
<· undefined
> console.log(typedef(a));
✖ ▶ Uncaught ReferenceError:
      at <anonymous>:1:9
> console.log(typeof(a))
  string
<· undefined
> var b=10
<· undefined
> console.log(typeof(b))
  number
<· undefined
> let b=false;
✖ Uncaught SyntaxError: Iden
> let c=false;
<· undefined
> console.log(typeof(c))
  boolean
```

Task 3: Use the console to perform basic math operations like addition, subtraction, multiplication, and division.



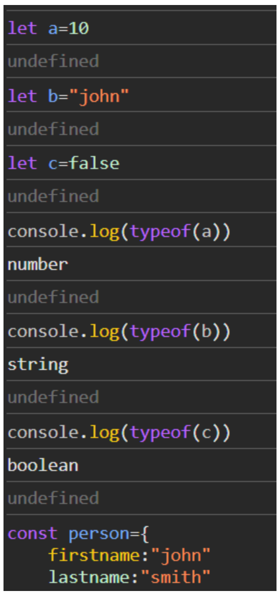
```
> let a=10;
<· undefined
> let b=101
✖ Uncaught SyntaxError
> let v=10;
<· undefined
> console.log(a+b)
  20
<· undefined
> console.log(a-b)
  0
<· undefined
> console.log(a*b)
  100
<· undefined
> console.log(a/b)
  1
```

Task 4: Declare two strings and concatenate them using the + operator.

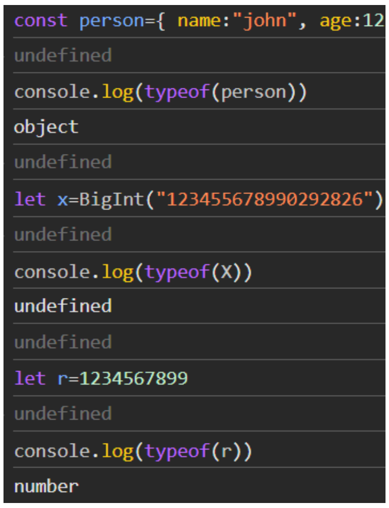


```
let a="john"
undefined
let b="smith"
undefined
console.log(a+b)
johnsmith
undefined
```

Task 5: Use the typeof operator to check the data type of various variables.



```
let a=10
undefined
let b="john"
undefined
let c=false
undefined
console.log(typeof(a))
number
undefined
console.log(typeof(b))
string
undefined
console.log(typeof(c))
boolean
undefined
const person={
    firstname:"john"
    lastname:"smith"
```

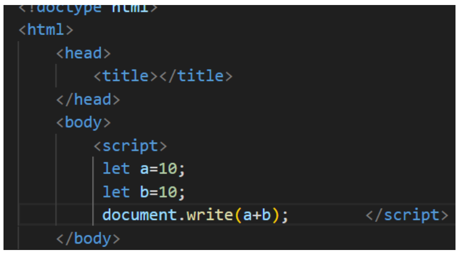


```
const person={ name:"john", age:12
undefined
console.log(typeof(person))
object
undefined
let x=BigInt("123455678990292826")
undefined
console.log(typeof(X))
undefined
undefined
let r=1234567899
undefined
console.log(typeof(r))
number
```

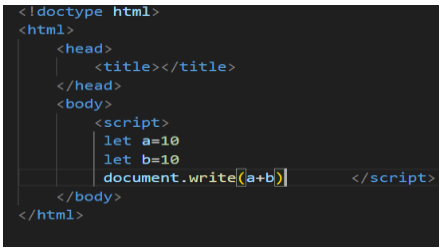
Task 6: Write a multi-line JavaScript comment and a single-line comment. Explain the difference.

Multi-line comment	Single-line comment
• /* this is a multi-line comment.	// this is single-line comment.
In this multi-line comment ,all the text between /*and */. For example, /* This is a multi-line comment. It explains that numbers and logs the result. */	In the single-line comment start from //. For example, // This is a single-line comment explaining the next line

Task 7: Create a script with both semicolon-separated and not separated lines. Note any differences in behavior



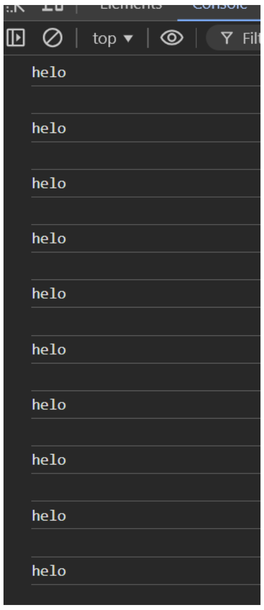
```
<!doctype html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <script>
        let a=10;
        let b=10;
        document.write(a+b);        </script>
    </body>
```



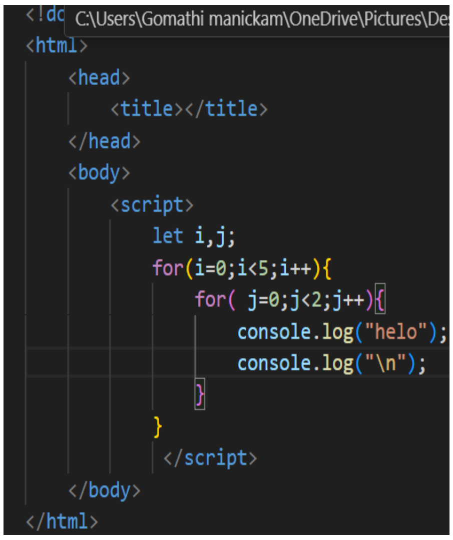
```
<!doctype html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <script>
        let a=10
        let b=10
        document.write(a+b)        </script>
    </body>
</html>
```

In this semicolon script, Semicolons explicitly mark the end of a statement. This avoids ambiguity, and the code is parsed exactly as written. This both give same output.but, in without semicolon script for small code it does not give any bugs and sometimes,it may cause bugs.

o Task 8: Use proper indentation to format a nested loop

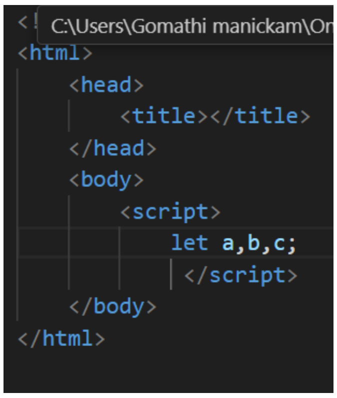


```
helo
helo
helo
helo
helo
helo
helo
helo
helo
helo
```



```
<!dc C:\Users\Gomathi manickam\OneDrive\Pictures\De
<html>
    <head>
        <title></title>
    </head>
    <body>
        <script>
            let i,j;
            for(i=0;i<5;i++){
                for( j=0;j<2;j++){
                    console.log("helo");
                    console.log("\n");
                }
            }
         </script>
    </body>
</html>
```

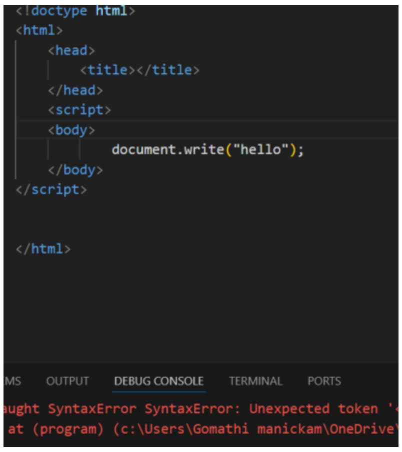
Task 9: Declare multiple variables in a single line.



```
<! C:\Users\Gomathi manickam\Or
<html>
    <head>
        <title></title>
    </head>
    <body>
        <script>
            let a,b,c;
         </script>
    </body>
</html>
```

Task 10: Place a script tag at the top and bottom of an HTML document. Note any differences in behavior.

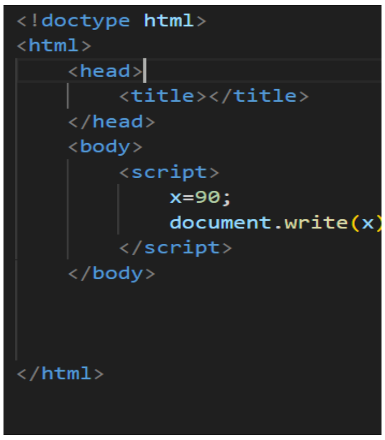
It does not give any output,it make syntax error.



```
<!doctype html>
<html>
    <head>
        <title></title>
    </head>
    <script>
    <body>
            document.write("hello");
    </body>
</script>


</html>
```

```
MS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
ught SyntaxError SyntaxError: Unexpected token '
 at (program) (c:\Users\Gomathi manickam\OneDrive
```

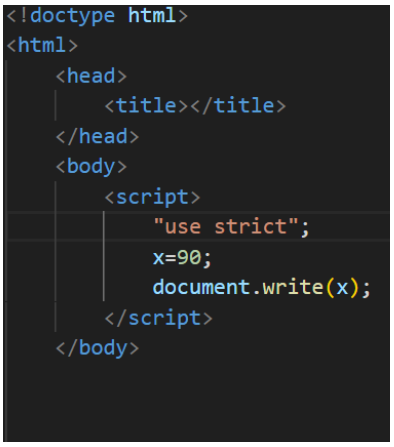
Task 11: Write a script without using “use strict” and try to assign a value to an undeclared variable. Note the result.



```
<!doctype html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <script>
            x=90;
            document.write(x
        </script>
    </body>


</html>
```

It will give the output has 90.

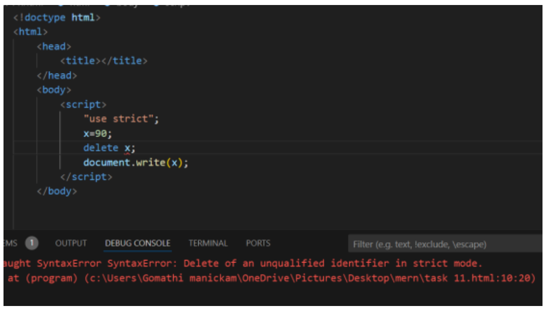
Task 12: Enable “use strict” mode and repeat the above action, noting the difference.



```
<!doctype html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <script>
            "use strict";
            x=90;
            document.write(x);
        </script>
    </body>
```

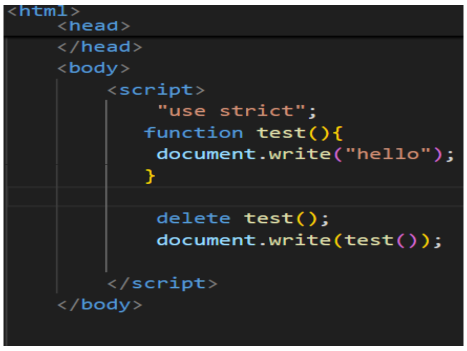
It does not give the output.

Task 13: In “use strict” mode, try to delete a variable, function, or function parameter.



```
<!doctype html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <script>
            "use strict";
            x=90;
            delete x;
            document.write(x);
        </script>
    </body>
```

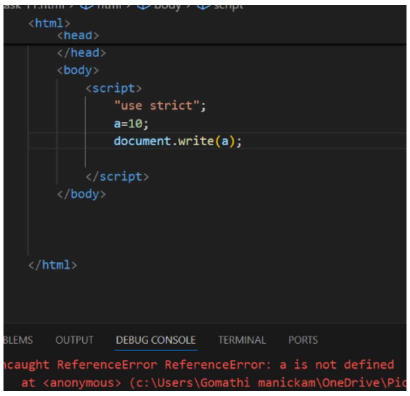
```
EMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
aught SyntaxError SyntaxError: Delete of an unqualified identifier in strict mode.
 at (program) (c:\Users\Gomathi manickam\OneDrive\Pictures\Desktop\mern\task 11.html:10:20)
```



```
<html>
    <head>
    </head>
    <body>
        <script>
            "use strict";
         function test(){
            document.write("hello");
         }

            delete test();
            document.write(test());

        </script>
    </body>
```

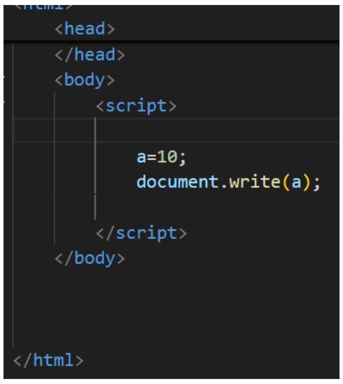
In that use strict mode we delete function means it works,suppose we delete variable means it does not works.

Task 14: Assign a value to an undeclared variable without “use strict” and then with “use strict”.



```
<html>
    <head>
    </head>
    <body>
        <script>
            "use strict";
            a=10;
            document.write(a);

        </script>
    </body>



</html>
```

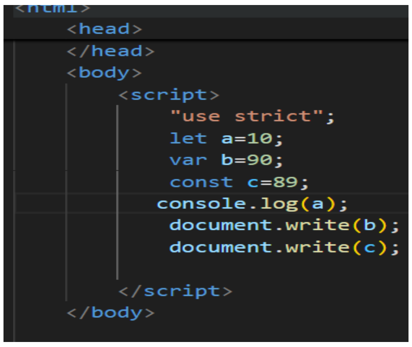
```
BLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
ncaught ReferenceError ReferenceError: a is not defined
   at <anonymous> (c:\Users\Gomathi manickam\OneDrive\Pic
```



```
    <head>
    </head>
    <body>
        <script>

            a=10;
            document.write(a);

        </script>
    </body>



</html>
```

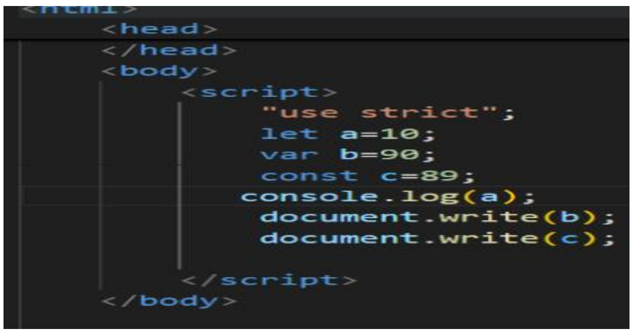
In the use strict mode it will make error because of undefined a variable.without use strict mode it does not give any error.it gives a output as 10.

Task 15: Declare a variable with a reserved keyword in “use strict” mode.



```
    <head>
    </head>
    <body>
        <script>
            "use strict";
            let a=10;
            var b=90;
            const c=89;
           console.log(a);
            document.write(b);
            document.write(c);

        </script>
    </body>
```

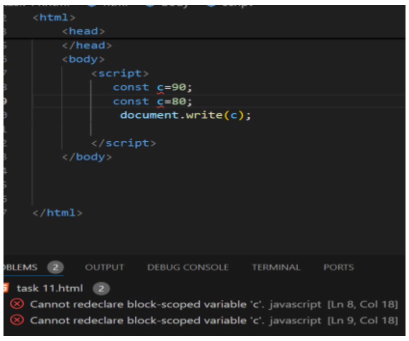
Task 16: Declare variables using let, const, and var. Discuss when each should be used.



```
    <head>
    </head>
    <body>
        <script>
            "use strict";
            let a=10;
            var b=90;
            const c=89;
           console.log(a);
            document.write(b);
            document.write(c);

        </script>
    </body>
```

var	let	Const
The scope of a *var* variable is functional or global scope.	The scope of a *let* variable is block scope.	The scope of a *const* variable is block scope.
It can be updated and re-declared in the same scope.	It can be updated but cannot be re-declared in the same scope.	It can neither be updated or re-declared in any scope.
It can be declared without initialization	It can be declared without initialization	It cannot be declared without initialization.

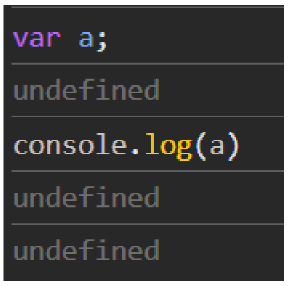
Task 17: Attempt to reassign a const variable and observe the result.



```
<html>
    <head>
    </head>
    <body>
        <script>
          const c=90;
          const c=80;
           document.write(c);

        </script>
    </body>



</html>
```

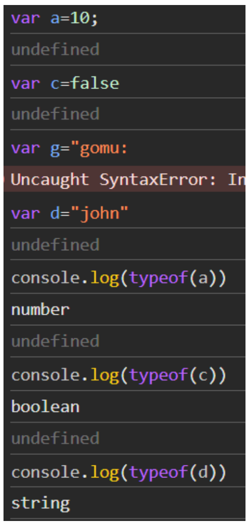
```
PROBLEMS 2    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
task 11.html 2
Cannot redeclare block-scoped variable 'c'. javascript [Ln 8, Col 18]
Cannot redeclare block-scoped variable 'c'. javascript [Ln 9, Col 18]
```

Task 18: Declare a variable without initializing it and print its value.



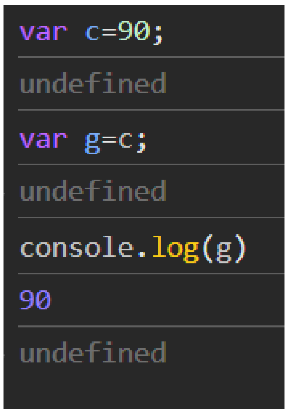
```
var a;
undefined
console.log(a)
undefined
undefined
```

Task 19: Assign a number, string, and boolean value to a variable and print its type using typeof.



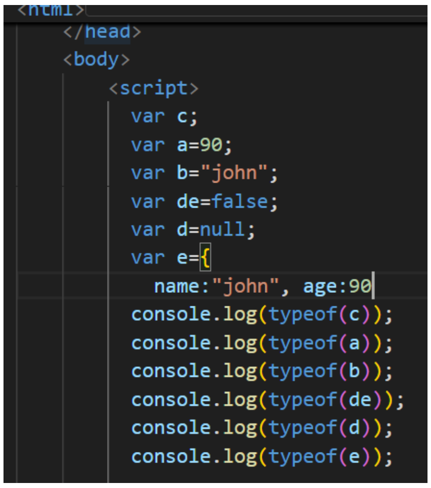
```
var a=10;
undefined
var c=false
undefined
var g="gomu:
Uncaught SyntaxError: In
var d="john"
undefined
console.log(typeof(a))
number
undefined
console.log(typeof(c))
boolean
undefined
console.log(typeof(d))
string
```

Task 20: Rename a variable and observe the outcome.

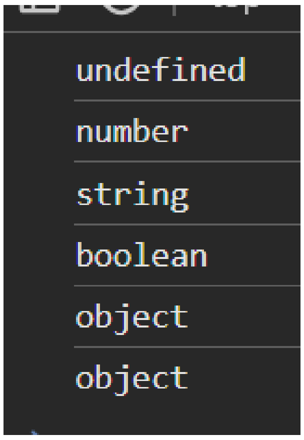


```
var c=90;
undefined
var g=c;
undefined
console.log(g)
90
undefined
```

Task 21: Create variables of different data types (e.g., string, number, boolean, null, undefined, object).

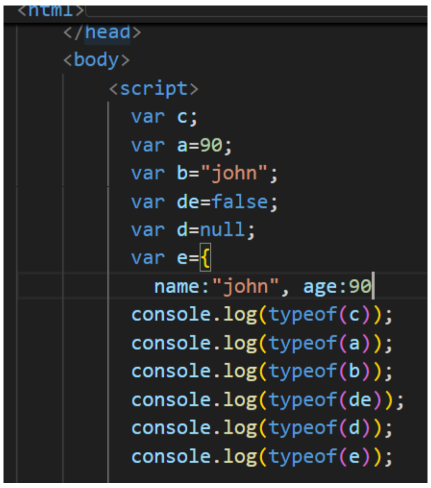


```
<html>
    </head>
    <body>
        <script>
          var c;
          var a=90;
          var b="john";
          var de=false;
          var d=null;
          var e={
            name:"john", age:90
          console.log(typeof(c));
          console.log(typeof(a));
          console.log(typeof(b));
          console.log(typeof(de));
          console.log(typeof(d));
          console.log(typeof(e));
```

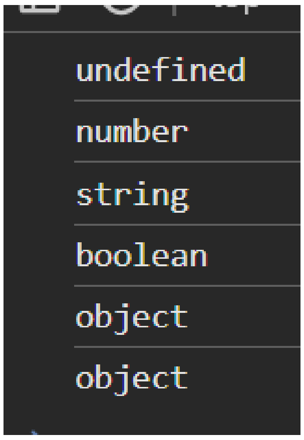


```
undefined
number
string
boolean
object
object
```

Task 22: Use the typeof operator to determine the type of various variables.

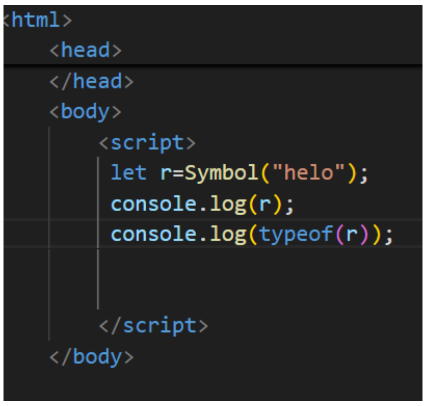


```
<html>
    </head>
    <body>
        <script>
          var c;
          var a=90;
          var b="john";
          var de=false;
          var d=null;
          var e={
            name:"john", age:90
          console.log(typeof(c));
          console.log(typeof(a));
          console.log(typeof(b));
          console.log(typeof(de));
          console.log(typeof(d));
          console.log(typeof(e));
```

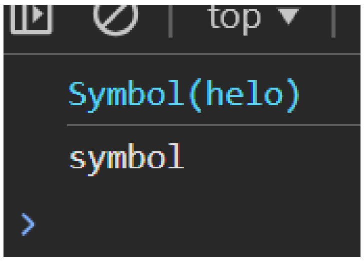


```
undefined
number
string
boolean
object
object
```

Task 23: Declare a symbol and print its type.

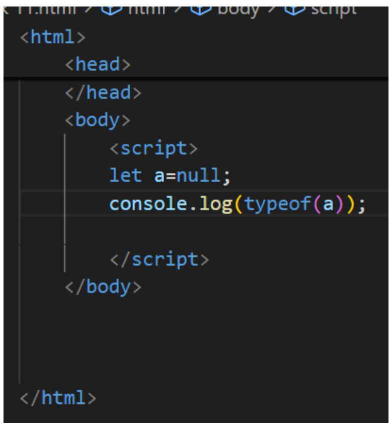


```
<html>
    <head>
    </head>
    <body>
        <script>
        let r=Symbol("helo");
        console.log(r);
        console.log(typeof(r));

        </script>
    </body>
```

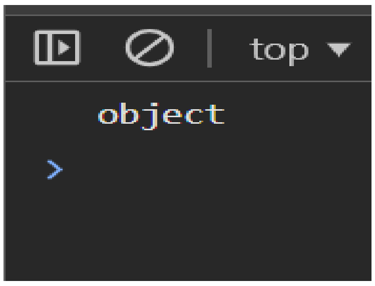


```
Symbol(helo)
symbol
```

Task 24: Assign the value null to a variable and check its type using typeof.



```
<html>
    <head>
    </head>
    <body>
        <script>
        let a=null;
        console.log(typeof(a));

        </script>
    </body>



</html>
```

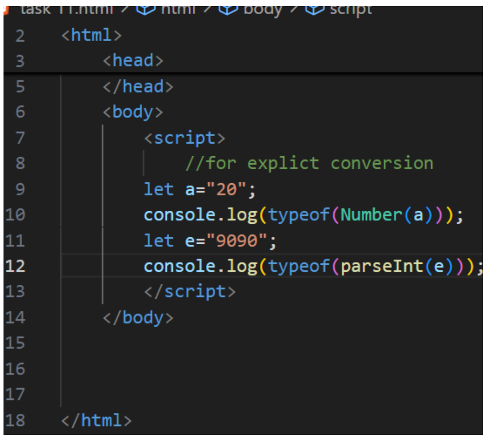


```
object
```

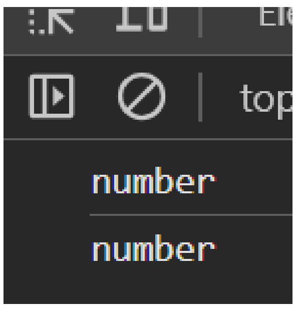
Task 25: Differentiate between declaring a variable using var and let in terms of scope.

var	let
The scope of a *var* variable is functional or global scope.	The scope of a *let* variable is block scope.
It can be updated and re-declared in the same scope.	It can be updated but cannot be re-declared in the same scope.
It can be declared without initialization	It can be declared without initialization

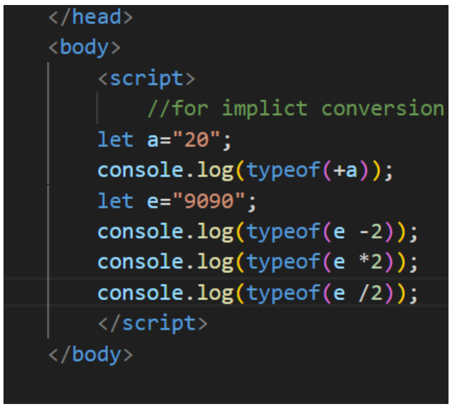
Task 26: Convert a string to a number using both implicit and explicit conversion.



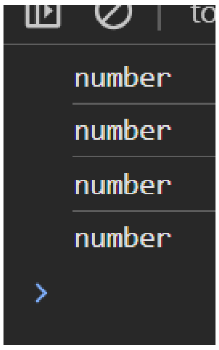
```
<html>
    <head>
    </head>
    <body>
        <script>
            //for explict conversion
        let a="20";
        console.log(typeof(Number(a)));
        let e="9090";
        console.log(typeof(parseInt(e)));
        </script>
    </body>




</html>
```



```
number
number
```

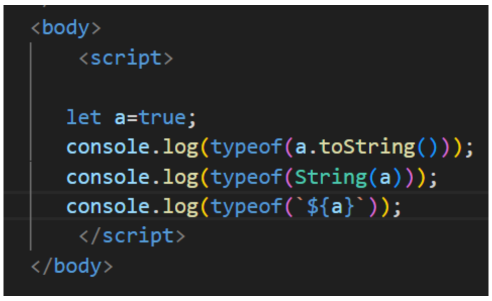


```
</head>
<body>
    <script>
        //for implict conversion
    let a="20";
    console.log(typeof(+a));
    let e="9090";
    console.log(typeof(e -2));
    console.log(typeof(e *2));
    console.log(typeof(e /2));
    </script>
</body>
```

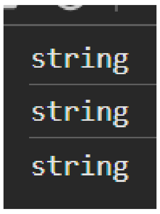


```
number
number
number
number
>
```

Task 27: Convert a boolean to a string and vice versa.

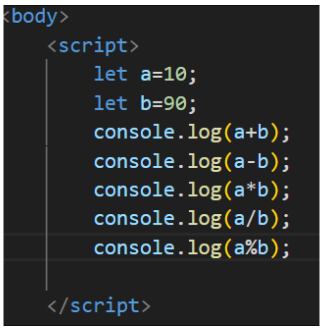


```
<body>
    <script>

  let a=true;
  console.log(typeof(a.toString()));
  console.log(typeof(String(a)));
  console.log(typeof(`${a}`));
    </script>
</body>
```

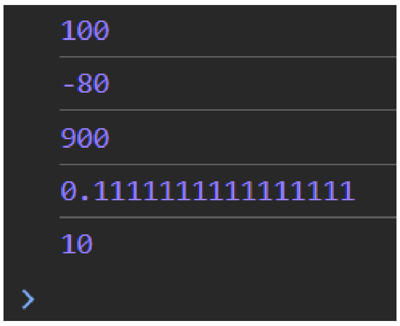


```
string
string
string
```

Task 28: Practice basic arithmetic operators (+, -, *, /, %).

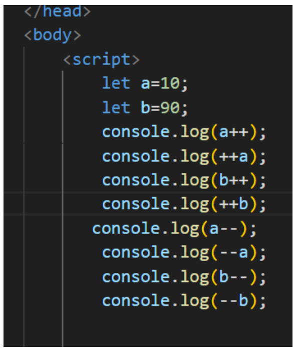


```
<body>
    <script>
        let a=10;
        let b=90;
        console.log(a+b);
        console.log(a-b);
        console.log(a*b);
        console.log(a/b);
        console.log(a%b);

    </script>
```

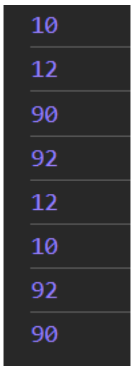


```
100
-80
900
0.1111111111111111
10
>
```

Task 29: Use the ++ and -- operators on a numeric variable.

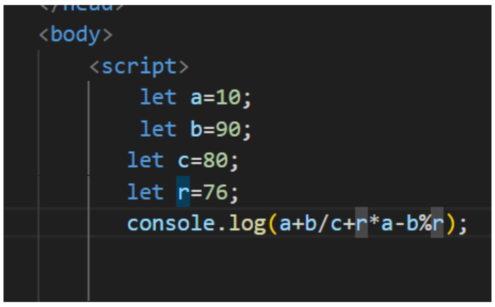


```
</head>
<body>
    <script>
        let a=10;
        let b=90;
        console.log(a++);
        console.log(++a);
        console.log(b++);
        console.log(++b);
       console.log(a--);
        console.log(--a);
        console.log(b--);
        console.log(--b);
```

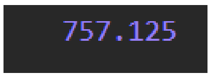


```
10
12
90
92
12
10
92
90
```

Task 30: Explore the precedence of operators by combining multiple operators in a single expression.

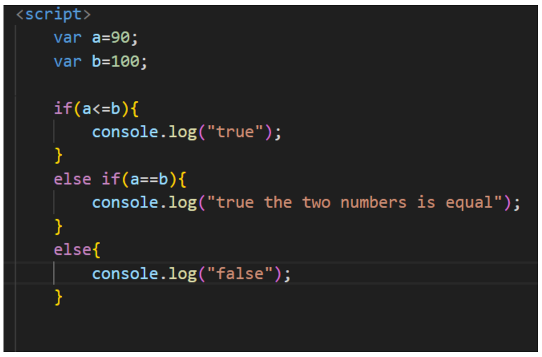


```
<body>
    <script>
        let a=10;
        let b=90;
      let c=80;
      let r=76;
      console.log(a+b/c+r*a-b%r);
```

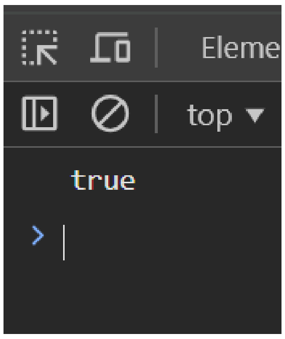


```
757.125
```

o Task 31: Compare two numbers using relational operators (>, =, <=).

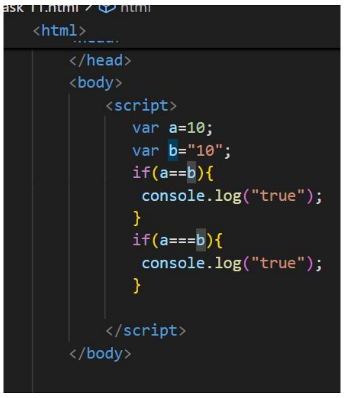


```
<script>
    var a=90;
    var b=100;

    if(a<=b){
        console.log("true");
    }
    else if(a==b){
        console.log("true the two numbers is equal");
    }
    else{
        console.log("false");
    }
```

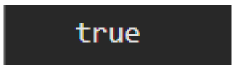


```
true
```

Task 32: Use equality () and strict equality (=) operators to compare different data types and note the differences.

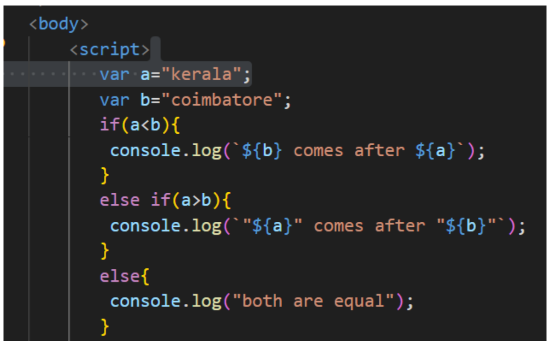


```
<html>
    </head>
    <body>
        <script>
            var a=10;
            var b="10";
            if(a==b){
             console.log("true");
            }
            if(a===b){
             console.log("true");
            }

        </script>
    </body>
```

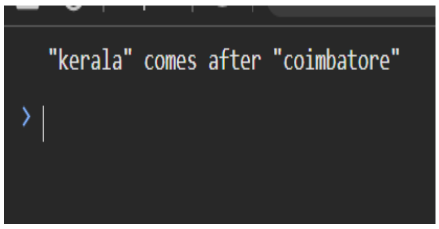


```
true
```

Task 33: Compare two strings lexicographically.

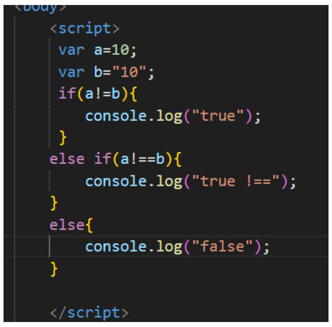


```
<body>
    <script>
        var a="kerala";
        var b="coimbatore";
        if(a<b){
         console.log(`${b} comes after ${a}`);
        }
        else if(a>b){
         console.log(`"${a}" comes after "${b}"`);
        }
        else{
         console.log("both are equal");
        }
```

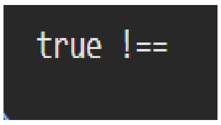


```
"kerala" comes after "coimbatore"
```

Task 34: Use the inequality (!=) and strict inequality (!==) operators to compare values.

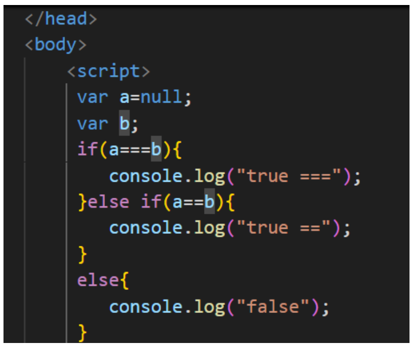


```
<body>
    <script>
     var a=10;
     var b="10";
     if(a!=b){
         console.log("true");
     }
    else if(a!==b){
        console.log("true !==");
    }
    else{
        console.log("false");
    }

    </script>
```

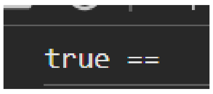


```
true !==
```

Task 35: Compare null and undefined using both == and ===.

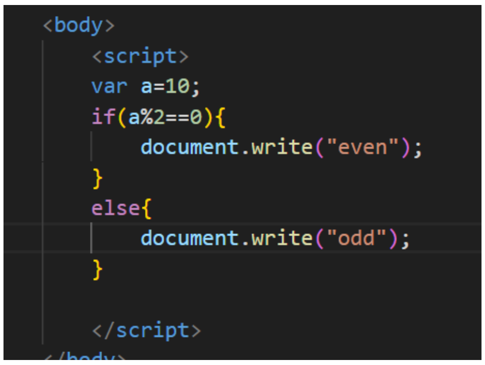


```
</head>
<body>
    <script>
    var a=null;
    var b;
    if(a===b){
        console.log("true ===");
    }else if(a==b){
        console.log("true ==");
    }
    else{
        console.log("false");
    }
```

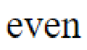


```
true ==
```

Task 36: Write an if statement that checks if a number is even or odd.

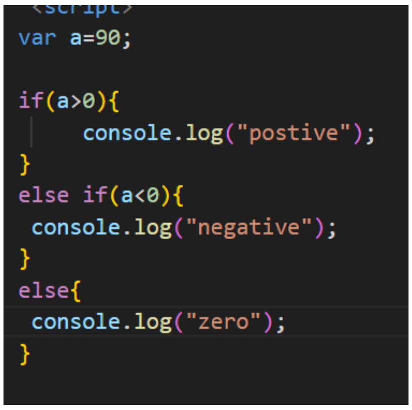


```
<body>
    <script>
    var a=10;
    if(a%2==0){
        document.write("even");
    }
    else{
        document.write("odd");
    }

    </script>
</body>
```

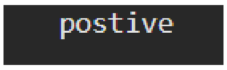


```
even
```

Task 37: Use nested if statements to classify a number as negative, positive, or zero.

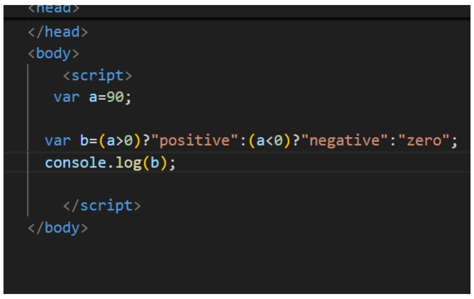


```
<script>
var a=90;

if(a>0){
     console.log("postive");
}
else if(a<0){
 console.log("negative");
}
else{
 console.log("zero");
}
```

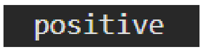


```
postive
```

Task 38: Use the conditional (ternary) operator '?' to rewrite a simple if...else statement.

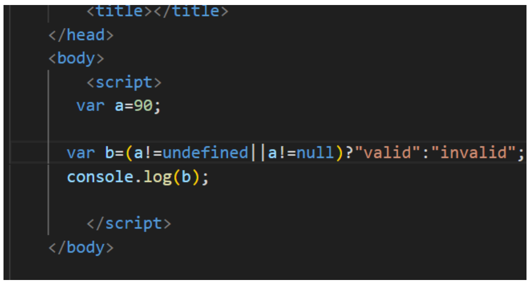


```
<head>
</head>
<body>
    <script>
   var a=90;

  var b=(a>0)?"positive":(a<0)?"negative":"zero";
  console.log(b);

    </script>
</body>
```

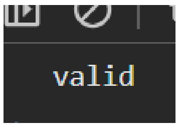


```
positive
```

Task 39: Check the validity of a variable using the ? operator.

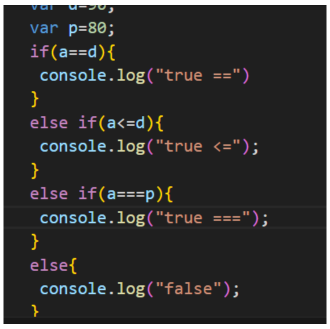


```
        <title></title>
    </head>
    <body>
        <script>
      var a=90;

    var b=(a!=undefined||a!=null)?"valid":"invalid";
    console.log(b);

        </script>
    </body>
```

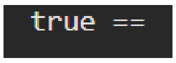


```
valid
```

Task 40: Use the conditional operator to assign a value to a variable based on a condition.

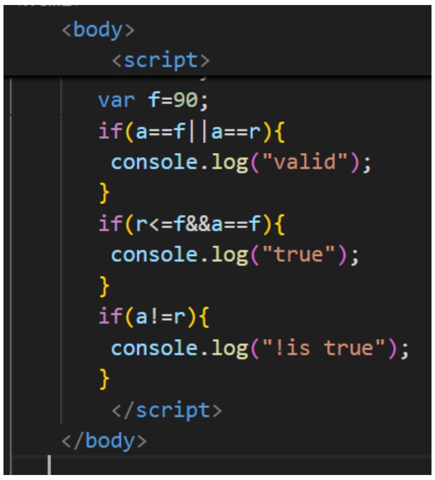


```
var d=90;
var p=80;
if(a==d){
 console.log("true ==")
}
else if(a<=d){
 console.log("true <=");
}
else if(a===p){
 console.log("true ===");
}
else{
 console.log("false");
}
```

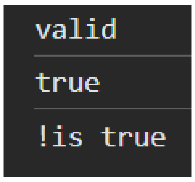


```
true ==
```

Task 41: Evaluate various combinations of logical operators (&&, ||, !).

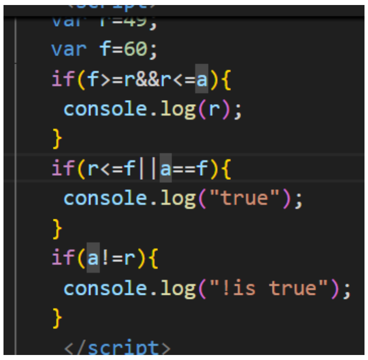


```
    <body>
        <script>
      var f=90;
      if(a==f||a==r){
       console.log("valid");
      }
      if(r<=f&&a==f){
       console.log("true");
      }
      if(a!=r){
       console.log("!is true");
      }
        </script>
    </body>
```

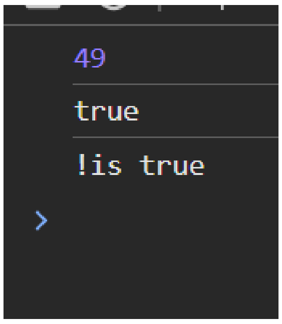


```
valid
true
!is true
```

Task 42: Use logical operators to write a condition that checks if a number is in a given range.

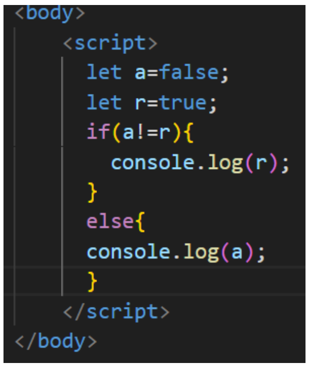


```
var f=60;
if(f>=r&&r<=a){
 console.log(r);
}
if(r<=f||a==f){
 console.log("true");
}
if(a!=r){
 console.log("!is true");
}
 </script>
```

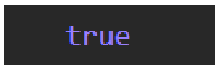


```
49
true
!is true
>
```

Task 43: Use the NOT (!) operator to invert a boolean value.



```
<body>
    <script>
      let a=false;
      let r=true;
      if(a!=r){
        console.log(r);
      }
      else{
      console.log(a);
      }
    </script>
</body>
```

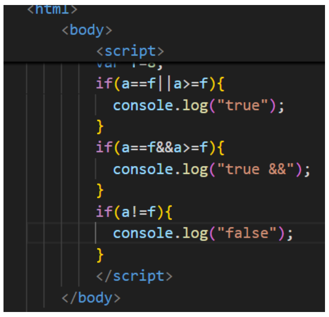


```
true
```

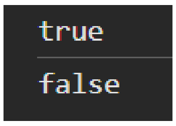
Task 44: Evaluate the short-circuiting nature of logical operators.

Basically,the logical operators is used to check the values and return Boolean expression as true or false.comparsion operator is used in logical statement to determine the equality or difference between the values or variables.

Task 45: Compare two non-boolean values using logical operators and observe the result.

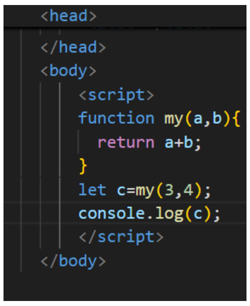


```
  <html>
    <body>
      <script>
      if(a==f||a>=f){
        console.log("true");
      }
      if(a==f&&a>=f){
        console.log("true &&");
      }
      if(a!=f){
        console.log("false");
      }
      </script>
    </body>
```

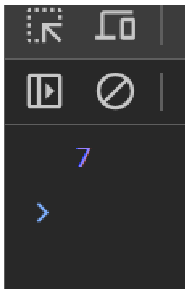


```
true
false
```

Task 46: Write a function that takes two numbers as arguments and returns their sum.

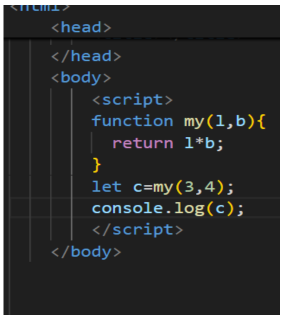


```
  <head>
  </head>
  <body>
    <script>
    function my(a,b){
      return a+b;
    }
    let c=my(3,4);
    console.log(c);
    </script>
  </body>
```

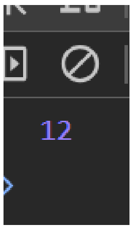


```
7
>
```

Task 47: Create a function that calculates the area of a rectangle.

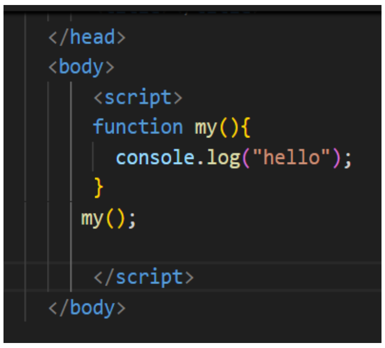


```
<head>
</head>
<body>
    <script>
    function my(l,b){
      return l*b;
    }
    let c=my(3,4);
    console.log(c);
    </script>
</body>
```

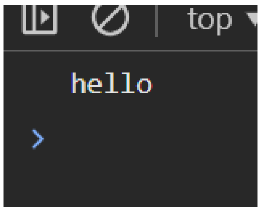


```
12
```

Task 48: Declare a function without parameters and call it.

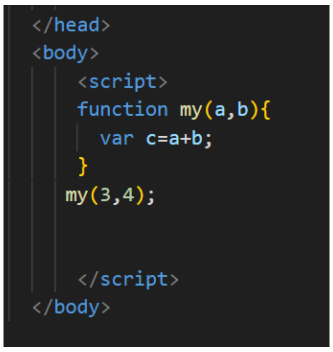


```
</head>
<body>
    <script>
    function my(){
      console.log("hello");
    }
   my();

    </script>
</body>
```

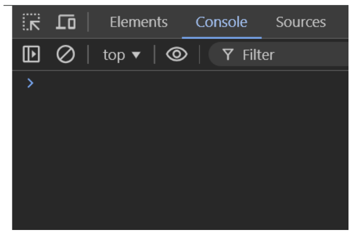


```
hello
```

Task 49: Write a function that returns nothing and observe the default return value.

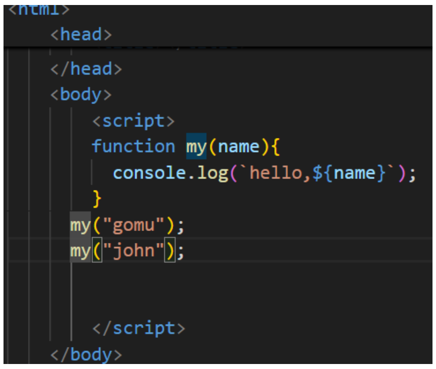


```
</head>
<body>
    <script>
    function my(a,b){
      var c=a+b;
    }
   my(3,4);


    </script>
</body>
```

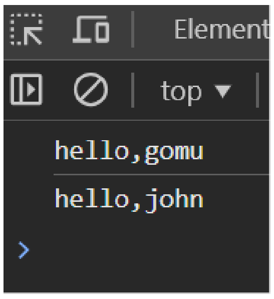


It does not return anything.so,it does not print anything.

Task 50: Declare a function with default parameters and call it with different arguments.

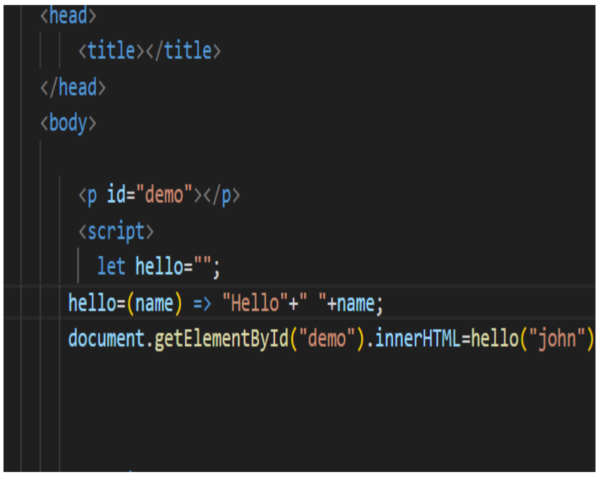


```
<head>
</head>
<body>
    <script>
    function my(name){
      console.log(`hello,${name}`);
    }
   my("gomu");
   my("john");


    </script>
</body>
```

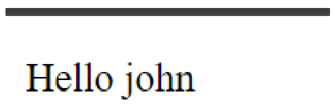


```
hello,gomu
hello,john
```

Task 51: Declare a simple arrow function named greet that takes one parameter name and returns the string “Hello, name!”. Test your function with various names.

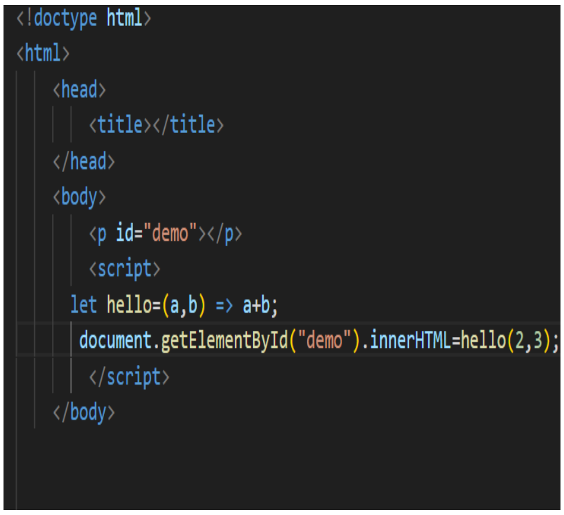


```
<head>
    <title></title>
</head>
<body>

    <p id="demo"></p>
    <script>
     let hello="";
   hello=(name) => "Hello"+" "+name;
   document.getElementById("demo").innerHTML=hello("john")
```

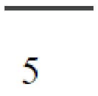


```
Hello john
```

Task 52: Write an arrow function named add that takes two parameters and returns their sum. Validate your function with several pairs of numbers.

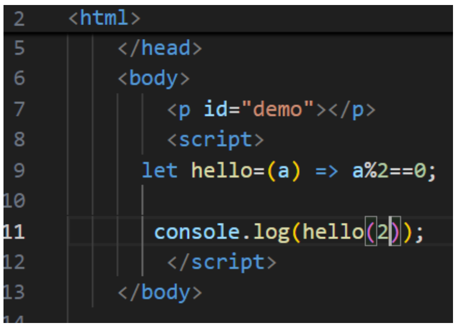


```
<!doctype html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <p id="demo"></p>
        <script>
      let hello=(a,b) => a+b;
       document.getElementById("demo").innerHTML=hello(2,3);
        </script>
    </body>
```

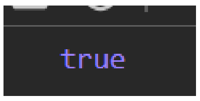


```
5
```

Task 53: Declare an arrow function named isEven that checks if a number is even. If the number is even, it should return true; otherwise, false. Remember that if the arrow function body has a single statement, you can omit the curly braces.

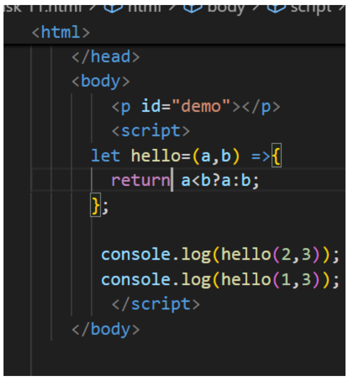


```
<html>
    </head>
    <body>
        <p id="demo"></p>
        <script>
      let hello=(a) => a%2==0;

       console.log(hello(2));
        </script>
    </body>
```

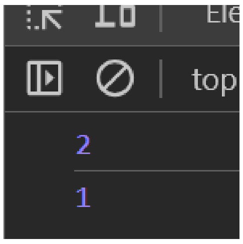


```
true
```

Task 54: Implement an arrow function named maxValue that takes two numbers as parameters and returns the larger number. Here, you'll need to use curly braces for the function body and the return statement.



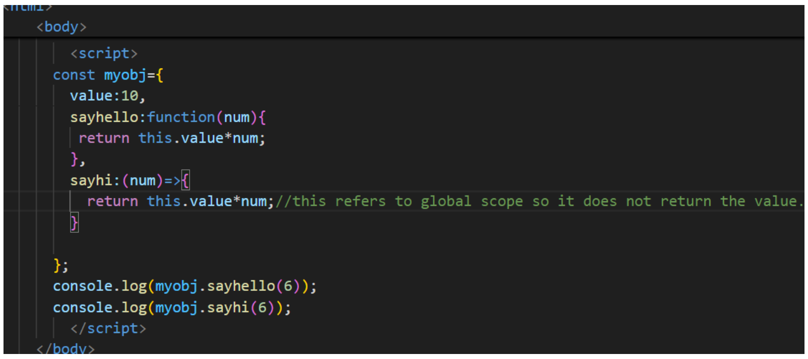
```
<html>
    </head>
    <body>
        <p id="demo"></p>
        <script>
    let hello=(a,b) =>{
        return a<b?a:b;
    };

     console.log(hello(2,3));
     console.log(hello(1,3));
        </script>
    </body>
```



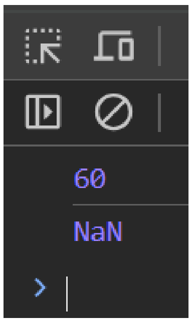
```
2
1
```

Task 55: Examine the behavior of the this keyword inside an arrow function vs a traditional function. Create an object named myObject with a property value set to 10 and two methods: multiplyTraditional using a traditional function and multiplyArrow using an arrow function. Both methods should attempt to multiply the value property by a number passed as a parameter. Check the value of this inside both methods.

- The handling of this is also different in arrow functions compared to regular functions.In short, with arrow functions there are no binding of this.
- In regular functions the this keyword represented the object that called the function, which could be the window, the document, a button or whatever.
- With arrow functions the this keyword *always* represents the object that defined the arrow function.



```
  <body>
      <script>
    const myobj={
      value:10,
      sayhello:function(num){
       return this.value*num;
      },
      sayhi:(num)=>{
        return this.value*num;//this refers to global scope so it does not return the value.
      }

    };
    console.log(myobj.sayhello(6));
    console.log(myobj.sayhi(6));
      </script>
  </body>
```



```
60
NaN
>
```