Unit-10 ECMAScript Version 6

ECMAScript (ES) is a scripting language specification standardized by ECMAScript International. It is used by applications to enable client-side scripting. ECMAScript 2015 was the second major revision to JavaScript. ECMAScript 2015 is also known as ES6 and ECMAScript 6.

"ECMA" stood for "European Computer Manufacturers Association" until 1994.

ES6 allows one to write code in a clever way which makes the code more readable. In short, using ES6, we write less and do more, hence it work on principle “write less. Do more”.

ES6 Template Literals

Template literals are a new feature introduced in ECMAScript 2015/ ES6. It provides an easy way to create multiline strings and perform string interpolation. Template literals are the string literals and allow embedded expressions.

Before ES6, template literals were called as template strings. Unlike quotes in strings, template literals are enclosed by the **backtick (` `)** character (key below the ESC key in QWERTY keyboard). Template literals can contain placeholders, which are indicated by the dollar sign and curly braces **(${expression})**. Inside the backticks, if we want to use an expression, then we can place that expression in the (${expression}).

Syntax

```
var str = `string value`;
```

Variable Substitutions: Template Strings allow variables in strings

```
Template Strings allow variables in strings:
Example
let firstName = "ABC";
let lastName = "XYZ";
let text = `Welcome ${firstName}, ${lastName}!`;
```

Expression Substitution

Template Strings allow expressions in strings:

```
let a = 10;
let b = 25;
let total = `Total: ${(a * (1 + b))}`;
```

Multiline strings

In normal strings, we have to use an escape sequence \n to give a new line for creating a multiline string. However, in template literals, there is no need to use \n because string ends only when it gets backtick (`) character.

Example:

```
<script> ans=10;
   console.log(`Welcome
                    Answer = ${ans}`);
// console.log("Welcome \n Answer");
 </script>
```

Output:

```
Welcome

Answer = 10
```

Arrow Function

Arrow functions are introduced in ES6, which provides you a more accurate way to write the functions in JavaScript. They allow us to write smaller function syntax. Arrow functions make your code more readable and structured. If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword.

Syntax

```
const functionName = (parameter received) =>{
                          //body of the function
                          }
```

Simple Example:

```
Before Arrow:
// fun1 is a variable
fun1 = function() {
  return "Hello World!";
}
After Arrow:
// fun1 is a function name
fun1 = () => {
  return "Hello World!";
}
```

If the function has only one statement, and the statement returns a value, you can remove the brackets and the return keyword:

```
Arrow Functions Return Value by Default:
fun1 = () => "Hello World!";
console.log(fun1())
```

Note: This works only if the function has only one statement.

Example 1

```
<script>
const newfun = p => `val=${p}`;
console.log(newfun(30));
</script>
```

Example 2 : Concept of return keyword and {} in ES6

```
<script>
const add = (j,k) => `${j+k}`;
console.log(`Addition = ${add(5,9)}`);
</script>
Output: Addition = 14
```

The function uses a concise arrow function syntax (without curly braces). In such cases, the value of the expression after the arrow (=>) is implicitly returned.
The expression ans = \${j + k}`` is evaluated and returned.

But If you add only {} then The function doesn't explicitly return anything because there is no return statement. In JavaScript, when a function doesn't return a value explicitly, it implicitly returns undefined. add(8, 9) is called, but it returns undefined.

```
<script>
const add = (j,k) => {`${j+k}`};
console.log(`Addition = ${add(5,9)}`);
</script>
```

Output: Addition = undefined

Always Use Arrow Function with a Block Body For functions with multiple statements, use a block body with curly braces {}. You must use an explicit return statement to return a value.

```
<script>
const add = (j,k) =>{
let ans = j+k;
return ans;
}
console.log(`Addition = ${add(10,15)}`);
</script>
```

… Operator

- Spread operator is used to distribute the element of array.
- Rest operator is used to merge/combine elements of array.

Spread operator

The Spread operator is denoted by three dots (…). The Spread operator allows an iterable to expand in places where 0+ arguments are expected. It is mostly used in the variable array where there is more than 1 value is expected. It allows us the privilege to obtain a list of parameters from an array. The syntax of the Spread operator is the same as the Rest parameter but it works opposite of it.

Example:

```
<html> <body> <script>
   odd = [1,2,4];
   combined = [5,6,7,...odd];
   console.log(combined);
 </script> </body></html>
```

Output:

```
(6) [5, 6, 7, 1, 2, 4]
```

Example:

```
<script type="text/javascript">
function fun(){
const num1 = [23,55,21,87,56];
const num2=[15,16,18,99]
const fnum=[...num1,...num2] //Replace concat
 let maxValue = Math.max(...fnum);
console.log(fnum) //shows array
console.log(...fnum) //shows value
document.getElementById("demo").innerHTML = maxValue;
}
</script></head>
<body>
<p onclick="fun()">max value of num1=[23,55,21,87,56] and num2=[15,16,18,99]</p>
<p id="demo"></p>
</body>
```

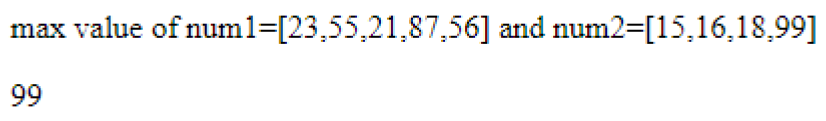
Output:

On console:

```
(9) [23, 55, 21, 87, 56, 15, 16, 18, 99]

23 55 21 87 56 15 16 18 99
```

On browser after clicking on paragraph:



```
max value of num1=[23,55,21,87,56] and num2=[15,16,18,99]

99
```

Rest Operator:

- **The rest parameter is an improved way to handle function parameters, allowing us to more easily handle various inputs as parameters in a function.**
- The rest parameter syntax allows us to represent an indefinite number of arguments as an array.
- With the help of a rest parameter, a function can be called with any number of arguments, no matter how it was defined.
- Rest parameter is added in ES2015 or ES6 which improved the ability to handle parameter.

Syntax:

```
//... is the rest parameter (triple dots)

function functionname(...parameters){

statement; }
```

Note: When … is added with parameter then it is the rest parameter. It stores n number of parameters as an array.

Example: without rest parameter. (passing arguments more than the parameters**)**

```
  function fun(a, b){
      console.log(a);
      console.log(b);
   }
   fun(1,2,3,4,5);
```

Output:

```
1
2
```

- In the above code, no error will be thrown even when we are passing arguments more than the parameters, but only the first two arguments will be evaluated.
- It’s different in the case of the rest parameter. With the use of the rest parameter, we can gather any number of arguments into an array and do what we want with them.

Example

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

function fun(...args)

 {

        console.log(args);

 }

 fun(1,2,3,4,5,6,7,8,9,10);

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

Output:

```
(10) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Example: JavaScript code demonstrating the addition of numbers using the rest parameter.

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

 function fun(a,b,...args)

 {

       console.log(args);

 }

 fun(1,2,3,4,5,6,7,8,9,10);

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

Output:

```
(8) [3, 4, 5, 6, 7, 8, 9, 10]
```

Example:

```
<html>
 <body>
 <script>
 myfun=(...args)=>
 {
   return args.length;
 }
 console.log(`length =${myfun(10,20,30,40,50)}`);
 </script>
 </body>
</html>
```

Output:

length =5

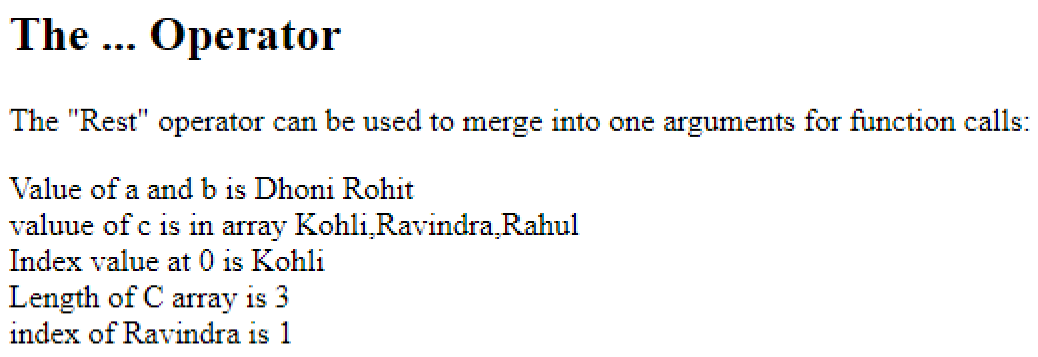
Example:

```
<html>
<body>
   <h2>The ... Operator</h2>
   <p>The "Rest" operator can be used to merge into one arguments for function calls:</p>

   <p id="demo"></p>

   <script>
   // rest with function and other arguments
function fun(a, b, ...c) {
         document.write( `Value of a and b is ${a} ${b} <br>`);
         document.write(`valuue of c is in array ${c}`);
         document.write(`<br>Index value at 0 is ${c[0]}`);
         document.write(`<br> Length of C array is ${c.length}`); //3
         document.write(`<br> index of Ravindra is ${c.indexOf('Ravindra')}`); //0
//console.log(c.indexOf('abc')); //-1 (incase of mismatch answer is -1)
}
fun('Dhoni', 'Rohit', 'Kohli', 'Ravindra', 'Rahul');
</script>
      </body>    </html>
```

Output:



```
The ... Operator

The "Rest" operator can be used to merge into one arguments for function calls:

Value of a and b is Dhoni Rohit
valuue of c is in array Kohli,Ravindra,Rahul
Index value at 0 is Kohli
Length of C array is 3
index of Ravindra is 1
```

Default Parameters

The term default can be understood as an option that is always available when no other options are appropriate. Similarly, a default value is a value that will always be available.

JavaScript function parameters are defined as undefined by default.

However, it may be useful to set a different default value. That is where default parameters come into play.

Syntax:

```
function name(parameters_n, parameter_n=value) {

}
```

Example 1: If we multiply two numbers in below example without passing a second argument and without using the default parameter, the answer that this function will return is NAN(Not a Number), since if we do not pass the second parameter, the function will multiply the first number with undefined.

```
function multiply(a, b) {
   return a * b;
}
let num1 = multiply(5);
console.log(num1);
let num2 = multiply(5, 8);
console.log(num2);

Output:
NaN
40
```

Example 2: If we do not pass a number as the second argument and take the default parameter as the second parameter, it will multiply the first number with the default parameter, and if we pass two numbers as parameters, it will multiply the first number with the second number.

```
function multiply(a, b = 2) {
   return a * b;
}
let num1 = multiply(5);
console.log(num1);
let num2 = multiply(5, 8);
console.log(num2);
```

Output:

```
10
40
```

Default argument must be filled from rear(right) side only

Example to understand above sentence.

```
<script>
defaultfun = (p=2,q)=>
{
return(p+q);
}
console.log(`Add =${defaultfun(30)}`);
</script>
```

Output:

```
NaN
```

Passed argument assigned to the p so here q is undefined. Answer is NaN

Example: Default parameter on left side gives output properly if you pass as much arguments as parameters. But there will be no use of default parameter as it will use arguments value not default as explain in below exaple.

```
<script>
      let demo=(a=2,b)=>
      {
      return a+b;
      }
      console.log(demo(4,5));
</script>
```

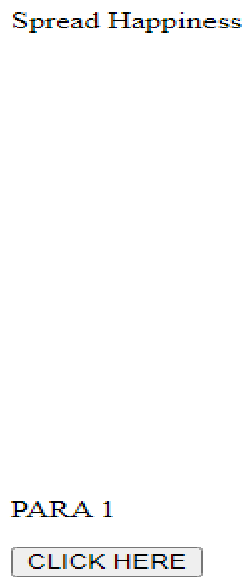
Output:

```
9
```

Example: Write Es6 function to be called on button click. Take one division with some text and no decoration initially. Take font size and style as default argument and pass background color and text color while passing argument to function style should change on button click.

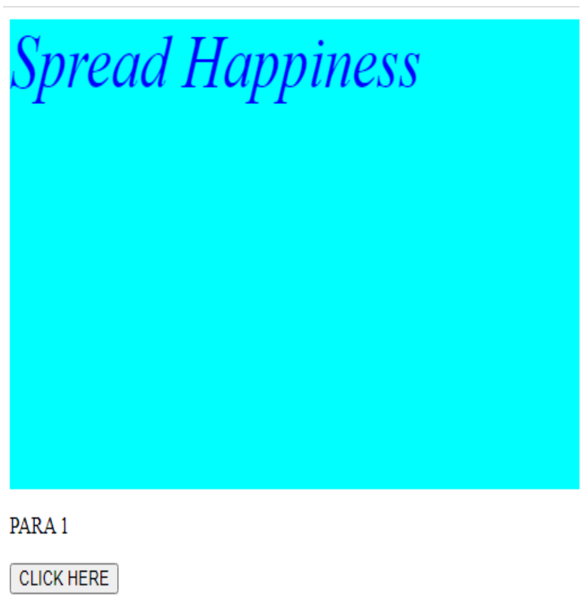
```
<body><script>
fun=(bg,tx,size=50,style="italic")=> {
i=document.getElementById("d1");
i.style.color=tx;
 i.style.fontStyle=style;
i.style.backgroundColor=bg;
i.style.fontSize=size;}
</script>
<div id="d1" style="width:50%; height:50%">
Spread Happiness
</div>
<p id="p1">PARA 1</p>
<input type="submit" onclick="fun('cyan','blue')" value="CLICK HERE"/>
</body>
```

Output: ***(After Clicking)***



```
Spread Happiness

PARA 1
```



```
Spread Happiness

PARA 1
```

For... of loop

JavaScript for…of statement iterates over the values of an iterable object (like Array, Map, Set, arguments object, …,etc), executing statements for each value of the object.

JavaScript for…of loop makes it easy to loop through the elements without needing to handle the index or iteration logic which makes the code short and easier to understand.

Syntax:

```
for (variable of iterableObjectName) {
      // code block to be executed
}
```

Example:

```
<html> <body> <script>
   ListOfScores = [80, 90, 70];
   for (let score of ListOfScores)
   {
            console.log(score);
   }
 </script> </body></html>
```

Output:

```
80
90
70
```

array.entries()

To access the index of the array elements inside the loop, you can use the for...of statement with the entries() method of the array. The array.entries() method returns a pair of [index, element] in each iteration.

Example:

```
<html> <body> <script>
 let colors = ['Red', 'Green', 'Blue'];
for (let [index, color] of colors.entries())
{
      console.log(`${color} is at index ${index}`);
}
 </script></body></html>
```

Output:

```
Red is at index 0

Green is at index 1

Blue is at index 2
```

For…in loop

JavaScript for in loop is used to iterate over the properties of an object. JavaScript for in loop iterates only over those keys of an object which have their enumerable property set to “true”.

Syntax

```
for (key in object) {

        // code block to be executed

}
```

Example:

```
<html>
 <body>
 <script>
    obj = {a:"hello", b:"Good", c:"Morning"}
    for (let k in obj)
    {
         console.log(`the keys are ${k}`);
    }
 </script>
 </body>
</html>
```

Output:

```
the keys are a

the keys are b

the keys are c
```

Example:

```
<html> <body> <script>
let scores = [10,20,30,40];
for (let score in scores)
{
            console.log(score);
           // console.log(scores[score]);
}
</script> </body></html>
```

Output:

```
0
1
2
3
```

Note: It returns only index position. To fetch value at that position, write code as: console.log(scores[score]);

Example: Write ES6 script to sort five string alphabetically stored in one array

```
<html>  <body>  <script>
 arr  =["abc","xyz","def"];
 for(i=0;i<=2;i++)
 {
 for(j=i+1;j<=2;j++)
 {
 if(arr[j].localeCompare(arr[i])<0)
 {
 temp = arr[j];
 arr[j]=arr[i];
 arr[i]=temp;
 } } }
 console.log(arr);
 for(let s of arr) {
    console.log(s);
 }
 </script> </body></html>
```

Output:

```
(3) ['abc', 'def', 'xyz']

abc
def
xyz
```

Example:

Write Es6 script to find maximum number from an array using for… of loop

```
large=0;
arr =[10,50,30];
for(let x of arr)
{
  if(x>large)
  {
     large=x;
  }
}
console.log(large);
```

Output:

```
50
```

Just for the understanding

The .entries() method returns an array containing arrays of an object's key-value pairs in the following format: [[key, value], [key, value], ...].

```
const object = { name: 'abc', age: 42 };
console.log(Object.entries(object));
```

Output: [[name, 'abc'],[age, 42]]

Example:

```
let arr=[1,2,3,4,5,6,7,8];
       let sum=0;
       for([ind,vals] of arr.entries())
       {
          if(ind%2==1)
          {
             sum=sum+vals;
          }
       }
       console.log(`Sum=${sum}`);
```

Output
Sum=20

Explanation: Elements from indexes 1,3,5,7 will be fetched. Elements are 2,4,6,8 and addition of it is 20.

ES6 Map

JavaScript map is a collection of elements where each element is stored as a Key, value pair. Map objects can hold both objects and primitive values as either key or value. When we iterate over the map object returns the key, and value pair in the same order as inserted. Map() constructor is used to create Map in JavaScript.

JavaScript Map has a property that represents the size of the map.

Steps to Create a Map

Method -1 Passing an Array to new Map()

```
let map = new Map([iterable])
```

*** new Map() :** It is a map constructor.

Method-2 Create a Map and use Map.set()

```
let map = new Map()
map.set([a,b])
```

Method	Description
.set()	add elements to a Map and also be used to change existing Map values
.delete()	removes a Map element
.clear()	removes all the elements from a Map

There are some differences between maps and objects. These are listed below -

Object	Map
Keys cannot be Object type	Keys can be any type
Keys are not ordered	Keys are ordered
not iterable	iterable

Example:

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

// method1:

let map=new Map(); map.set("ABC","FACULTY"); map.set("XYZ","DEAN");
map.set("PQR","PEON"); map.set("JKL","ADMIN");



//  method2:

let map1=new Map([["ABC","FACULTY"], ["XYZ","DEAN"], ["PQR","PEON"],
["JKL","ADMIN"], ["MNO","STUDENT"]])
console.log(map);
console.log(map1);
map.delete("PQR");
console.log(map);
map.set("PQR","SWIPPER");
console.log(map);
map.set("ABC","HOD");
console.log(map);
</script> </body></html>
```

Output:

```
Map(4) {size: 4, ABC => FACULTY, XYZ => DEAN, PQR => PEON, JKL => ADMIN} Map(5)
{size: 5, ABC => FACULTY, XYZ => DEAN, PQR => PEON, JKL => ADMIN, MNO =>
STUDENT}
Map(3) {size: 3, ABC => FACULTY, XYZ => DEAN, JKL => ADMIN}
Map(4) {size: 4, ABC => FACULTY, XYZ => DEAN, JKL => ADMIN, PQR => SWIPPER}
Map(4) {size: 4, ABC => HOD, XYZ => DEAN, JKL => ADMIN, PQR => SWIPPER}
```

String methods

String Method	Description
startsWith(substring,position)	Returns true if a string starts with a specified string. Otherwise it returns false. Position is option. It is case sensitive. Default value is 0.
endsWith(substring,length)	Returns true if a string ends with a specified string. Otherwise it returns false. The length determines the length of the given string from the beginning to be searched for the search string. It is case sensitive. Default value is the length of the string.
includes(searchstring,position)	The includes() method returns true if a string contains a specified string. Otherwise it returns false. If position is given, it starts searching from that position only. Position is option. It is case sensitive. Default value is 0.

Example:

```
<html> <body> <script>
s="this is demo";
console.log(s.startsWith("this"));
 console.log(s.startsWith("this",8));
 console.log(s.startsWith("demo",8));
 console.log(s.endsWith("demo",12));
console.log(s.endsWith("is",7));
 console.log(s.includes("is"));
</script> </body></html>
```

Output:

```
true
false
true
true
true
true
```

ES6 Destructuring Assignment

ES6 provides a new feature called destructing assignment that allows you to destructure properties of an object or elements of an array into individual variables.

How to use the ES6 destructuring assignment that allows you to destructure an array into individual variables.

Array destructuring

Array element transferred in individual variables. We may have an array or object that we are working with, but we only need some of the items contained in these. Destructuring makes it easy to extract only what is needed.

Example:

```
<script>
const National = ['Mango', 'Tiger', 'Rose'];

const Fruits = National[0];
 const Animal = National[1];
const Flower = National[2];   // old way

console.log(`${Fruits} ${Animal} ${Flower}`);

//Array Destructuring method
const Nation = ['Peacock', 'Rupees', 'Ganga'];
const[Bird,Currency, River] = Nation
console.log(`${Bird} ${Symbol} ${River}`)
</script>
```

Output:

```
Mango Tiger Rose
Peacock Rupees Ganga
```

```
function getScores() {
  return [70, 80, 90];
}
let [x, y, z] = getScores();
console.log(x); // 70
console.log(y); // 80
console.log(z); // 90
```

The variables x, y and z will take the values of the first, second, and third elements of the returned array.

Note that the square brackets [ ] look like the array syntax but they are not.

Example:

If you want to choose random elements from the given array then in array destructuring you can perform it as follows:

```
<html>

<head>

<script>

var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"];

  // destructuring assignment

var[color1, ,color3, ,color5] = colors; //Leave space for unpick elements

console.log(color1); // Violet

console.log(color3); // Blue

console.log(color5); // Yellow

</script>

</head>

</html>
```

In the above example, we have defined an array named colours which has seven elements. But we have to show three random colours from the given array that are Violet, Blue, and Yellow. These array elements are in positions 0, 2, and 4.

During restructuring, you have to leave the space for unpick elements, as shown in the above example.

ES6 Objects

Object are collection of key-value pair of custom type.
Example:

```
<html> <body> <script>
        let username="ABC", age=29;
        let user ={
        username,
        age    };
        console.log(user.username);
        console.log(user.age);
 </script> </body></html>
```

Output:
ABC
29

Example: If member and local variable are not same

```
<html> <body> <script>
    let username="Abc", age=29;
    let user1 =
    {
    uname:username,
    a:age
    };
 console.log(user1.uname);
 console.log(user1.a);
 </script> </body></html>
```

Output:

Abc
29

Object Destructuring

- It is similar to array destructuring except that instead of values being pulled out of an array, the properties (or keys) and their corresponding values can be pulled out from an object.
- When destructuring the objects, we use **keys as the name of the variable**.
- **The variable name must match the keys name of the object.**
- If it does not match, then it receives an **undefined** value.
- This is how JavaScript knows which property of the object we want to assign.

First, try to understand the basic assignment in object destructuring by using the following example.

```
const employee = {name: 'ABC', position: 'Developer', id: '24'};
const {name, position, id} = employee;
console.log(name); // ABC
console.log(position); // Developer
console.log(id); // 24
```

Output:

```
ABC
Developer
24
```

```
const employee = {name: 'ABC', position: 'Developer', id: '24'};
const { name : n, position : p, id : i } = employee;
console.log(n); // ABC
console.log(p); // Developer
console.log(i); // 24
```

Output:

```
ABC
Developer
24
```

Example (Using ES6)

```
<script>
 let person =
 {
 firstname : "abc",
 lastname : "def"
 };
 let
 {
 firstname : fname,
 lastname : lname
 }=person;
 console.log(`first:${fname}, last:${lname}`);
 </script>
```

Output:

```
first:abc
```

Class Constructor

The constructor() method is a special method for creating and initializing objects created within a class.

The constructor() method is called automatically when a class is initiated, and it has to have the exact name "constructor", in fact, if you do not have a constructor method, JavaScript will add an invisible and empty constructor method.

Syntax

```
 constructor()
constructor(argument0)
constructor(argument0, argument1)
constructor(argument0, argument1,  …, argument N)
```

Example:

```
<script> class
clg {
constructor() {
    this.name = 'LJU';
  }
}
const clg1 = new clg();
console.log(clg1.name);
</script>
```

Output:

```
LJU
```

Note: A class cannot have more than one constructor() method. This will throw a Syntax Error.

Example:

```
<html> <body> <script>
class person
 {
 constructor(n,a)
 {
 this.name=n;
 this.age=a;
 }
 getname = () =>
 {
    console.log(this.age);
    return this.name;
     }
 }
 let obj = new person("ABC",29);
 console.log(obj.getname());
 </script> </body></html>
```

Output:

```
29

ABC
```

Example:

Write an ES6 script to calculate distance between two points. Take two-point object consisting of x, y, z members. 1st object should be used to call distance() function and 2nd object should be passed inside an argument.

```
<html> <body><script>
class point
{
 constructor(x,y,z)
{
    this.x=x;
    this.y=y;
    this.z=z;
    }
    dist=(p2)=>
   {
    let ans=Math.pow((this.x-p2.x),2)+Math.pow((this.y-p2.y),2)+Math.pow((this.z-p2.z),2);
    let d= Math.sqrt(ans);
    return d;
   }
   }
   let p1=new point(1,1,1); let p2=new
   point(2,2,2);
   console.log(`distance=${p1.dist(p2)}`)
   </script> </body></html>
```

Output:

```
distance=1.7320508075688772
```

Example:

Write an ES6 script that creates a class time having members hours, minutes and second.

Create two time objects t1 and t2 and add both the time objects so that it should return in third time object t.

Third time object should have hour, minute and second such that after addition if second exceeds 60 then minute should be incremented also if minutes exceeds 60 then hour should be incremented.

```
class time{
   constructor(hour,min,sec)
   {
      this.hour=hour;
      this.min=min;
      this.sec=sec;
   }
   timer()
   {
      var t=new time();
      t.hour=t1.hour+t2.hour;
      t.min=t1.min+t2.min;
      t.sec=t1.sec+t2.sec;
      if(t.sec>60)
      {
         t.sec%=60;
         t.min++;

      }
      if(t.min>60)
      {
         t.min%=60;
         t.hour++;
      }
      return t;
   }
}
var t1= new time(1,50,50);
var t2= new time(2,30,50);
console.log(t1.timer());
</script>
```

Output: time {hour: 4, min: 21, sec: 40}