Full Stack Development Using Javascript-1

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. Latest release Edition 14 June 2023.

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

10.1 Var, let and Const Keywords

Var keyword

The var is the keyword to declare a variable in JavaScript.

Scope: The scope of the var keyword is the global or function scope. It means variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function.

Example

```
<html>
	<head>
	<script type="text/javascript">
		function fun()
		{
			var a= 10;
			if(a>0)
			{
			a=100;
			document.write("A inside = "+a+"</br>");
			}
			document.write("A outside = "+a);
		}
	</script>
	</head>
	<body>
		<input type="submit" onclick="fun()"/>
	</body>
</html>
```

A inside = 100
A outside = 100

let keyword

The let keyword is an improved version of the var keyword.

Scope: block scoped: The scope of a let variable is only block scoped. It can't be accessible outside the particular block ({block}). Let's see the below example.

```
<html>
	<head>
	<script type="text/javascript">
		function fun()
		{
			let a= 10;
			if(a>0)
			{
			let a=100;
			console.log("A inside = "+a);
			}
			console.log("A outside = "+a);
		}

	</script>
	</head>
	<body>
		<input type="submit" onclick="fun()"/>
	</body>
</html>
```

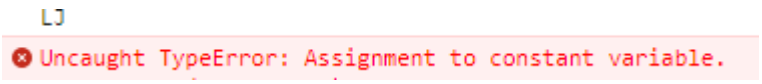
```
A inside = 100
A outside = 10
```

const keyword has all the properties that are the same as the let keyword, except the user cannot update it.

Scope: block scoped: When users declare a const variable, they need to initialize it, otherwise, it returns an error. The user cannot update the const variable once it is declared.

Example

```
<html>
	<head>
	<script type="text/javascript">
		function fun()
		{
			const name="LJ";
			console.log(name);
			name="JL";
			console.log(name);
		}
	</script>
	</head>
	<body>
		<input type="submit" onclick="fun()"/>
	</body>
</html>
```

Template Literals

Template Literals use back-ticks (``) rather than the quotes ("") to define a string

Example

```
<html>
   <head>
   </head>
   <body>
      <script>
            ans=10;
            console.log(`Answer = ${ans}`);
      </script>
   </body>
</html>
```

```
Answer = 10
```


Fat 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.

Syntax

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

Example

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

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

```
val=30
```

Example

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

```
Addition = 14
```

Example

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

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

```
Addition = 25
```

Example: Write a function using ES6 to calculate and print all prime numbers between given two numbers passed as an argument assume that first number is smaller then second number

```
<html>
  <head>
  </head>
  <body>
    <script>
        const prime = (j,k) =>
        {
            for(i=j;i<=k;i++)
            {
               flag=0;
               for (l = 2; l <= i / 2; ++l)
               {
                  if (i % l == 0)
                  {
                    flag = 1;
                    break;
                  }
               }
            if (flag == 0)
               document.write(i+"<br>");
            }
         }
        console.log(`Prime  = ${prime(10,15)}`);
```

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

```
11
13
```

Spread operator

The Spread operator is denoted by **three dots (…)**. The spread operator helps the iterable objects to expand into individual elements.

Example

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

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


Rest parameter

The rest parameter allows us to represent an indefinite number of arguments as an array. By using the rest parameter, a function can be called with any number of arguments.

Example

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

```
        function f(...args)
          {
           console.log(args);
          }
          f(1,2,3,4,5,6,7,8,9,10);
    </script>
  </body>
</html>
```

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

Example

```
<html>
  <head>
  </head>
  <body>
    <script>
        function f(a,b,...args)
          {
           console.log(args);
          }
          f(1,2,3,4,5,6,7,8,9,10);
    </script>
  </body>
</html>
```

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

Example

```
<html>
  <head>
  </head>
  <body>
```

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

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

```
length =5
```

Example

```
<html>
  <head>
  </head>
  <body>
    <script>
        myfun=(...args)=>
        {
           console.log(args);
           return args.length;
        }
        console.log(`length =${myfun(10,20,30,40,50,-500)}`);
    </script>
  </body>
</html>
```

```
[10, 20, 30, 40, 50, -500]

length =6
```

ES6 Default Parameters

```
<html>
   <head>
   </head>
   <body>
      <script>
            defaultfun = (p,q=10)=>
            {
               return(p+q);
            }
            console.log(`Add =${defaultfun(30)}`);
      </script>
   </body>
</html>
```

```
Add =40
```

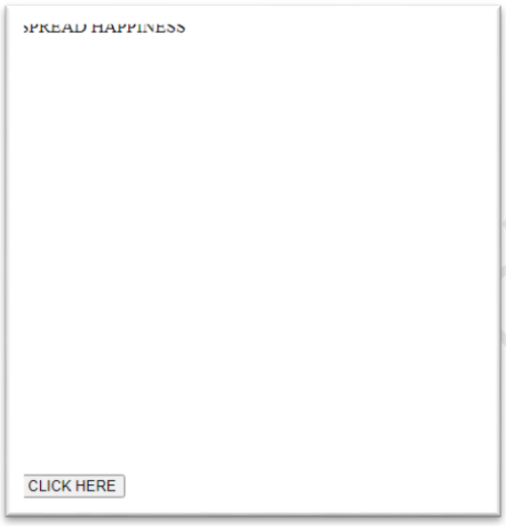
Note: Default argument must be filled from rear side only

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.

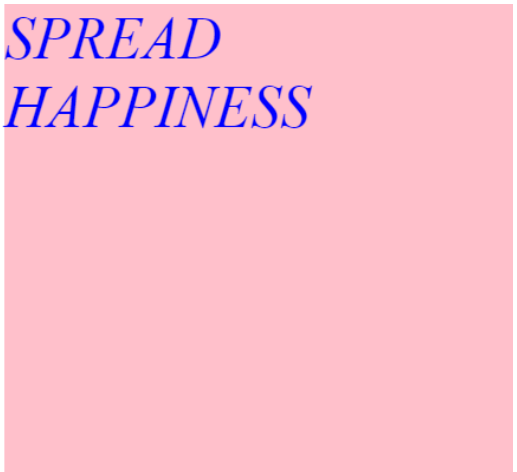
```
<html>
   <head>
   </head>
   <body>
      <script>
            fun=(bg,tx,size=50,style="italic")=>
            {
               i=document.getElementById("d1");
               i.style.color=tx;
               i.style.fontStyle=style;
```

```
            i.style.backgroundColor=bg;
            i.style.fontSize=size;
        }
    </script>
    <div id="d1" style="width:50%; height:50%">
       SPREAD HAPPINESS
    </div>
    <p id="p1"></p>
    <input type="submit" onclick="fun('pink','blue')" value="CLICK HERE"/>
  </body>
</html>
```

Before button click



After button click



For... of loop

ES6 introduced for...of that iterates over an iterable object such as array or map

Syntax

```
for (variable of iterable) {
  // ...
}
```

Example

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



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

```
Red is at index 0
Green is at index 1
Blue is at index 2
```

For…in loop

The for...in iterates over all enumerable properties of an object.

for (key in object)

```
{
  // code block to be executed
}
```

Example

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

```
0
1
2
```

Maps

This allows to store all element in key and value pair

Example

```
<html>
  <head>
  </head>
  <body>
    <script>
      let map=new Map();
       map.set("ABC","FACULTY");
       map.set("XYZ","DEAN");
       map.set("PQR","PEON");
       map.set("JKL","ADMIN");
       console.log(map);

       map.delete("PQR");
       console.log(map);
       map.set("PQR","SWIPPER");
       console.log(map);
       map.set("ABC","HOD");
       console.log(map);
    </script>
  </body>
</html>
```

```
▶ Map(4) {'ABC' => 'FACULTY', 'XYZ' => 'DEAN', 'PQR' => 'PEON', 'JKL' => 'ADMIN'}
▶ Map(3) {'ABC' => 'FACULTY', 'XYZ' => 'DEAN', 'JKL' => 'ADMIN'}          VM89:10
                                                                          VM89:12
▶ Map(4) {'ABC' => 'FACULTY', 'XYZ' => 'DEAN', 'JKL' => 'ADMIN', 'PQR' => 'SWIPPER'}
                                                                          VM89:14
▶ Map(4) {'ABC' => 'HOD', 'XYZ' => 'DEAN', 'JKL' => 'ADMIN', 'PQR' => 'SWIPPER'}
```

New Strings Methods: includes(), startsWith(), endsWith()

Example

```
<html>
   <head>
   </head>
   <body>
      <script>
          s="this is demo";
          s1="AAA";
          s2="ABC";

          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>
```

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

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

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

```
50
```

Array Destructuring

Array element transferred in individual variables.

Example(Without ES6)

```
<html>
   <head>
   </head>
   <body>
      <script>
            function getscore()
         {
            return [90,80,70];
         }
         let scores=getscore();
         let x=scores[0];
         let y=scores[1];
         let z=scores[2];
         console.log(`x=${x}, y=${y}, z=${z}`);
      </script>
   </body>
</html>
```

```
x=90, y=80, z=70
```

Example(Using ES6)

```
<html>
    <head>
    </head>
    <body>
        <script>
  destruct = () =>
            {
                return [10,20,30];
            }
            let[p,q,r]=destruct();
            console.log(`x=${p}, y=${q}, z=${r}`);
        </script>
    </body>
</html>
```

```
x=10, y=20, z=30
```

ES6 Objects

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

Example

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

```
thanks
29
```

If member and local variable are not same

Example

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

```
thanks
29
```

Constructor, Constructor Function

Example

```
<html>
   <head>
   </head>
   <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("ZALAK",29);
         console.log(obj.getname());
      </script>
   </body>
</html>
```

```
29
ZALAK
```

Object Destructuring

Concept to unbox an object.

Example(Before ES6)

```
<html>
   <head>
   </head>
   <body>
      <script>
          let person  =
          {
             firstname : "zalak",
             lastname : "bhatt"
          };
          console.log(`first=${person.firstname}, last=${person.lastname}`);
      </script>
   </body>
</html>
```

```
first=zalak, last=bhatt
```

Example(After ES6)

```
<html>
   <head>
   </head>
   <body>
      <script>
          let person  =
         {
            firstname : "zalak",
            lastname : "bhatt"
         };
          let
         {
            firstname : fname,
            lastname : lname
         }=person;
         console.log(`first:${fname}, last:${lname}`);
      </script>
   </body>
</html>
```

```
first:zalak, last:bhatt
```

Example: Calculate distance between two points.

Take two objects consisting (x, y, z) members. First object should be used to call distance function. Second should be passed inside an argument

```
<html>
   <head>
   </head>
   <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>
```

```
distance=1.7320508075688772
```