ES6

Topics Covered:

- JavaScript ES6.
- JavaScript Let.
- JavaScript Const.
- JavaScript Arrow Function.
- JavaScript Array methods.
- JavaScript Objects.
- JavaScript Classes.
- JavaScript Strings.

JavaScript ES6:

- JavaScript ES6 is also known as ECMAScript 2015 or ECMAScript 6.
- ECMAScript is the standard that JavaScript programming language uses.
- ECMAScript provides the specification on how the JavaScript programming language should work.

JavaScript Let:

- JavaScript ***let*** is used to declare variables. Previously, variables were declared using only the ***var*** keyword.
- The variables declared using let are block-scoped.
- The variables declared using let are only accessible within a particular block.
- Example:

```
// variable declared using let
let name = 'Sara';
{
    // can be accessed only inside
    let name = 'Peter';

    console.log(name); // Peter
}
console.log(name); // Sara
```

Difference between let and var:

let	var
Block scoped variable	Function scoped variable
Does not allows to redeclare variables	Allows to redeclare variables
Hoisting does not occur in let	Hoisting occurs in var

JavaScript Const:

- The const keyword is used to declare constants in JavaScript.
- Constants are similar to let variables, except that the value cannot be changed.
- Once declared the value of the const variable cannot be changed.
- Example:

```
// name declared with const cannot be changed
const name = 'Sara';
```

JavaScript Arrow Function:

- Arrow functions are the short syntax for writing function expressions.
- To write an arrow function you don't need function keyword, return keyword, and curly brackets.
- The return statement and the curly braces are not needed only when the function is a single statement.
- Arrow functions allows to create a function in a cleaner way compared to regular functions.
- Syntax:

```
let myFunction = (arg1, arg2, ...argN) => {
    statement(s)
}
```

- Where,
    - myFunction → name of the function.
    - arg1, arg2, ….argN → function arguments.
    - statement(s) → function body.
- Arrow function VS Regular function:

```
// function expression using arrow function
let x = (x, y) => x * y;
```

```
// function expression
let x = function(x, y) {
   return x * y;
}
```

- Example: Arrow function with no argument:

```
let greet = () => console.log('Hello');
greet(); // Hello
```

- Example: Arrow function with one argument:

```
let greet = x => console.log(x);
greet('Hello'); // Hello
```

- Example: Arrow function as expression:

```
let age = 5;

let welcome = (age < 18) ?
  () => console.log('Baby') :
  () => console.log('Adult');

welcome(); // Baby
```

- Example: Multiline arrow function:

```
let sum = (a, b) => {
    let result = a + b;
    return result;
}

let result1 = sum(5,7);
console.log(result1); // 12
```

- Arrow functions do not have their own this.
- Arrow functions are not hoisted; they must be defined before they are used.

- Using const keywords is safer than using var/let keywords, because a function expression is always a constant value.
- Arrow function should not be used to create methods inside objects.

```
let person = {
    name: 'Jack',
    age: 25,
    sayName: () => {

        // this refers to the global .....
        //
        console.log(this.age);
    }
}

person.sayName(); // undefined
```

- Arrow function cannot be used as a constructor.

```
let Foo = () => {};
let foo = new Foo(); // TypeError: Foo is not a constructor
```

JavaScript Array methods:

Array.form()

- The Array.form() method returns an array object with a length property or any iterable object.
- HTML:

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

- JS:

```
const myArr = Array.from("ABCDEFG");
document.getElementById("demo").innerHTML = myArr;
```

- Output:

A,B,C,D,E,F,G

Array keys()

- The keys() method returns an array iterator object with keys of an array.
- HTML:

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

- JS:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const keys = fruits.keys();

let text = "";
for (let x of keys) {
  text += x + "<br>";
}

document.getElementById("demo").innerHTML = text;
```

- Output:

0
1
2
3

Array find()

- The find() method returns the value of the first array element that passes a test function.
- HTML:

```
<h2>JavaScript Array.find()</h2>
<p id="demo"></p>
```

- JS:

```
const numbers = [4, 9, 16, 25, 29];
let first = numbers.find(myFunction);

document.getElementById("demo").innerHTML = "First number over 18 is " +
first;

function myFunction(value, index, array) {
  return value > 18;
}
```

- Output:

JavaScript Array.find()

First number over 18 is 25

Array findIndex()

- The findIndex() method returns the index of the first array element that passes a test function.
- HTML:

```
<h2>JavaScript Array.findIndex()</h2>
<p id="demo"></p>
```

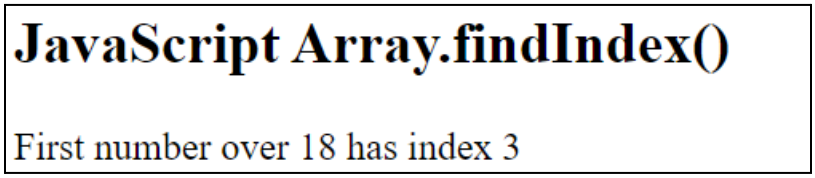
- JS:

```
const numbers = [4, 9, 16, 25, 29];

document.getElementById("demo").innerHTML = "First number over 18 has
index " + numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

- Output:



JavaScript Objects:

object.entries() and object.values()

- There are two ways to access objects, object.entries() and object.values().
- object.values() returns an array of all values of the object.
- object.entries() returns an array that contains both keys and values.
- Example - object.values():
  - HTML:

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

  - JS:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const f = fruits.values();

for (let x of f) {
  document.getElementById("demo").innerHTML += x + "<br>";
}
```

  - Output:

Banana
Orange
Apple
Mango

- Example - object.entries():
  - HTML:

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

  - JS:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const f = fruits.entries();

for (let x of f) {
  document.getElementById("demo").innerHTML += x + "<br>";
}
```

  - Output:

0,Banana
1,Orange
2,Apple
3,Mango

JavaScript Classes:

- A class is a blueprint for the object. Objects can be created from class.
- Class is a sketch of a house, which contains all details about the floors, doors, windows, etc…
- JavaScript class is similar to constructor function.
- Constructor function:

```
// constructor function
function Person () {
    this.name = 'John',
    this.age = 23
}

// create an object
const person1 = new Person();
```

- Instead of using a function keyword, a class keyword should be used to create a class.

- Example:

```
// creating a class
class Person {
  constructor(name) {
    this.name = name;
  }
}
```

- Properties are assigned in a constructor function, and objects can be created for the class.
- Objects for above mentioned class “Person”:

```
// creating an object
const person1 = new Person('John');
const person2 = new Person('Jack');

console.log(person1.name); // John
console.log(person2.name); // Jack
```

- person1 and person2 are objects of Person class.
- The constructor method inside the class gets called automatically each time an object is created.
- It is easy to define and access methods in JavaScript class by
    - Defining: Giving a method name followed by ().
    - Accessing: Call the method using its name followed by ().

```
class Person {
    constructor(name) {
    this.name = name;
  }

    // defining method
    greet() {
        console.log(`Hello ${this.name}`);
    }
}

let person1 = new Person('John');

// accessing property
console.log(person1.name); // John

// accessing method
person1.greet(); // Hello John
```

- In JavaScript, getter methods get the value of an object and setter methods set the value of an object.
- Example:

```
class Person {
    constructor(name) {
        this.name = name;
    }

    // getter
    get personName() {
        return this.name;
    }

    // setter
    set personName(x) {
        this.name = x;
    }
}

let person1 = new Person('Jack');
console.log(person1.name); // Jack

// changing the value of name property
person1.personName = 'Sarah';
console.log(person1.name); // Sarah
```

JavaScript Strings:

string.includes()

- The includes() method returns true if a string contains a specific value, else false.
- HTML:

```
<p>Check if a string includes "world":</p>

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

- JS:

```
let text = "Hello world, welcome to the universe.";
document.getElementById("demo").innerHTML = text.includes("world");
```

- Output:

Check if a string includes "world":

true

string.startsWith()

- The startsWith() method returns true if a string begins with a specified value, otherwise false.
- HTML:

```
<p>Check if a string starts with "Hello":</p>

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

- JS:

```
let text = "Hello world, welcome to the universe.";
document.getElementById("demo").innerHTML = text.startsWith("Hello");
```

- Output:

Check if a string starts with "Hello":

true

string.endsWith()

- The endsWith() method returns true if a string ends with a specified value, otherwise false.
- HTML:

```
<p>Check if a string ends with "end":</p>

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

- JS:

```
let text = "Hello World";
document.getElementById("demo").innerHTML = text.endsWith("end");
```

- Output:

Check if a string ends with "end":

false