JavaScript ES6

Objectives

After completing this lesson, you should be able to do the following:

- ES6 Fundamentals
- Functions and Arrays
- Classes and Objects
- Strings

Introdution to ES6

Introduction

- ES6 or ECMAScript 6 is a scripting language specification which is standardized by ECMAScript International. This specification governs some languages such as JavaScript
- , ActionScript, and Jscript. ECMAScript is generally used for client-side scripting, and it is also used for writing server applications and services by using Node.js
- ES6 allows you to write the code in such a way that makes your code more modern and readable. By using ES6 features, we write less and do more, so the term 'Write less, do more' suits ES6.

What is ES6?

- ES6 is an acronym of **ECMAScript 6** and also known as ECMAScript 2015.
- ES6 or **ECMAScript6** is a scripting language specification which is standardized by ECMAScript International. It is used by the applications to enable client-side scripting. This specification is affected by programming languages like Self, Perl, Python, Java, etc.
- This specification governs some languages such as JavaScript, ActionScript, and Jscript. ECMAScript is generally used for client-side scripting, and it is also used for writing server applications and services by using Node.js.

- ES6 introduces you many great features such as scope variable, arrow functions, template strings, class destructions, modules, etc.
- ES6 was created to standardize JavaScript to help several independent implementations. Since the standard was first published, JavaScript has remained the well-known implementation of ECMAScript, comparison to other most famous implementations such as **Jscript** and **ActionScript**.

History

- The ECMAScript specification is the standardized specification of scripting language, which is developed by **Brendan Eich (**He is an American technologist and the creator of JavaScript programming language**)** of **Netscape (**It is a name of brand which is associated with Netscape web browser's development**).**
- Initially, the ECMAScript was named **Mocha,** later **LiveScript,** and finally, **JavaScript.** In December 1995, **Sun Microsystems (**an American company that sold the computers and its components, software, and IT services. It created Java, NFS, ZFS, SPARC, etc.**)** and **Netscape** announced the JavaScript during a press release.
- During November 1996, Netscape announced a meeting of the ECMA International standard organization to enhance the standardization of JavaScript.

ES6 Version

1.	June 1997	First Edition
2.	June 1998	Editorial changes for keeping the specification fully aligned with the ISO/IEC 16262 international standard.
…	….	…
10.	June 2019 ECMAScript 2019 (ES 2019)	Addition of new features include Array.prototype.flatMap, Array.prototype.flat, and changes to Array.sort and Object.fromEntries.

ES6 Syntax

ES6 Syntax

- The syntax is the set of rules which defines the arrangements of symbols. Every language specification has its syntax. Syntax applies to **programming languages** in which the document represents the **source code** and also applies to **markup languages ,** in which the document describes the **data.**
- ES6 does contains same aspects of JavaScripts like
  - **Literals, Variables, Keywords, Operators, Comments, Identifiers**
  - **Line Breaks and Whitespaces**
  - **Strict Mode**

ES6 Spread Operator

- ES6 introduced a new operator referred to as a spread operator, which consists of three dots (...). It allows an iterable to expand in places where more than zero arguments are expected. It gives us the privilege to obtain the parameters from an array.
- Spread operator syntax is similar to the rest parameter, but it is entirely opposite of it. Let's understand the syntax of the spread operator.

Syntax

```
var variablename1 = [...value];
```

- The three dots (...) in the above syntax are the spread operator, which targets the entire values in the particular variable.

Constructing array literal

- When we construct an array using the literal form, the spread operator allows us to insert another array within an initialized array.

Example

```
let colors = ['Red', 'Yellow'];
let newColors = [...colors, 'Violet', 'Orange', 'Green'];
console.log(newColors);
```

Output

```
[ 'Red', 'Yellow', 'Violet', 'Orange', 'Green' ]
```

ES6 Rest Parameter

- The rest parameter is introduced in ECMAScript 2015 or ES6, which improves the ability to handle parameters. 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.
- Before ES6, the **arguments** object of the function was used. The **arguments** object is not an instance of the Array type. Therefore, we can't use the **filter()** method directly.
- The rest parameter is prefixed with three dots (...). Although the syntax of the rest parameter is similar to the spread operator, it is entirely opposite from the spread operator. The rest parameter has to be the last argument because it is used to collect all of the remaining elements into an array.

Syntax

```
function fun(a, b, ...theArgs) {
  // statements
}
```

Example

```
function show(...args) {
  let sum = 0;
  for (let i of args) {
     sum += i;
  }
  console.log("Sum = "+sum);
}

show(10, 20, 30);
```

ES6 Variables

- A variable is a named space in the memory, which stores the values. The names of the variable are known as **identifiers.** There are some rules that you should keep in mind while naming an identifier. These rules are as follows:
  1. It can contain numbers and alphabets.
  2. You cannot start the name of the variable with a number.
  3. Keywords cannot be used as the name of the variable.
  4. Identifiers do not contain spaces and special characters except the **dollar ($) symbol** and **underscore (_).**

Initialization of Variable

- It is the process to store the value in the variable. A variable can be initialized at any time before its use.
- The ES6 syntax used the keyword **var** to declare a variable and also the variables are declared by:
  - Using **let**.
  - Using **const**.

Let and Const

- Let
  - Any variable which is declared by using the **let** keyword is assigned the block scope. **Block scope** is nothing but a section where the **let** variable gets declared whether it is a **function{}**, a **block{}**, or a **global (script)**.

- Const:
  - ES6 gives a new way to declare a constant using the **const** keyword. The keyword **const** creates a read-only reference to the value. There are some properties of the **const** that are as follows:
  - **Properties:**
    - It cannot be reassigned a value.
    - It is block scoped.
    - A constant cannot be re-declared.
    - Constants must be initialized at the time of declaration.

- Operators, Loops and Decision Making
  - Is Same as JavaScript

ES6 Functions

- A function is the set of input statements, which performs specific computations and produces output. It is a block of code that is designed for performing a particular task. It is executed when it gets invoked (or called). Functions allow us to reuse and write the code in an organized way.
- Functions in JavaScript are used to perform operations.

Example

```
function hello() //defining a function
{
  console.log ("hello function called");
  }
hello(); //calling of function
```

Default Function Parameters

- In ES6, the function allows the initialization of parameters with default values if the parameter is undefined or no value is passed to it.

```
function show (num1, num2=200)
{
   console.log("num1 = " +num1);
   console.log("num2 = " +num2);

}
show(100);
```

Rest Parameters

- Rest parameters do not restrict you to pass the number of values in a function, but all the passed values must be of the same type. We can also say that rest parameters act as the placeholders for multiple arguments of the same type.
- For declaring the rest parameter, the name of the parameter should be prefixed with the **spread operator** that has three periods (not more than three or not less than three).

```
function show(a, ...args)
{
   console.log(a + " " + args);
}
show(50,60,70,80,90,100);
```

Output

```
50,60,70,80,90,100
```

ES6 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.
- Arrow functions are **anonymous functions** (the functions without a name and not bound with an identifier). They don't return any value and can declare without the function keyword. Arrow functions cannot be used as the constructors. The context within the arrow functions is lexically or statically defined. They are also called as **Lambda Functions** in different languages.
- Arrow functions do not include any prototype property, and they cannot be used with the new keyword.

There are three parts to an Arrow Function or Lambda Function:

- **Parameters:** Any function may optionally have the parameters.
- **Fat arrow notation/lambda notation:** It is the notation for the **arrow (=>).**
- **Statements:** It represents the instruction set of the function.

```
const functionName = (arg1, arg2, ?..) => {
   //body of the function
}
```

```
// function expression

var myfun1 = function show() {
 console.log("It is a Function Expression");
}

// Anonymous function

var myfun2 = function () {
   console.log("It is an Anonymous Function");
  }

//Arrow function

var myfun3 = () => {
   console.log("It is an Arrow Function");
  };

myfun1();
myfun2();
myfun3();
```

Arrow Function do Contain

1. Optional parentheses for the single parameter
2. Optional braces for single statement and the empty braces if there is not any parameter required.
3. Arrow Function Can Have Parameters
4. Arrow function Can Have default parameters
5. Arrow function Can Have Rest parameters

ES6 Arrays

Arrays

- Arrays in ES6 is Same as Arrays in JavaScript

Creating Arrays

- Initializing an array:

```
var myNewEmptyArray = [];

var numbers = [1, 2, 3, 4, 'five'];
```

- Creating an array with `new`:

```
var myNewEmptyArray = new Array();

var myNewNonEmptyArray = new Array(15);

var numbers = new Array(1, 2, 3, 4, 'five');
```

Accessing Array Elements

- Length of an array:

```
var c = ["blue","red","green","purple"];
c.length;
c[20] = "white";
c.length;
```

4

21

- Iterating arrays:

```
for (var i = 0; i < c.length; i += 1)
{
    console.log(c[i]);
}
```

```
for (var i in c)
{
    console.log(c[i]);
}
```

Multidimensional Arrays

- Creating multidimensional arrays:

```
var myArr = new Array(10);
for (i = 0; i < 10; i++) {
    myArr[i] = new Array(10);
    for (j = 0; j < 10; j++) {
      myArr[i][j] = "[" + i + j + "]";
   }
}
```

```
var tic_tac_toe = [
[ 0, 0,'X'],
['X',0,'X'],
['X',0, 0]
];
```

ES6 Array methods

1.	Array.from()	It converts array-like values and iterable values into arrays.	ECMAScript 6
2.	Array.of()	It creates an instance from a variable number of arguments instead of the number of arguments or type of arguments.	ECMAScript 6
3.	Array.prototype.copyWithin()	It copies the part of an array to a different location within the same array.	ECMAScript 6
4.	Array.prototype.find()	It finds a value from an array, based on the specific criteria that are passed to this method.	ECMAScript 6
5.	Array.prototype.findIndex()	The Array.prototype.findIndex() returns the index of the first element of the given array that satisfies the given condition.	ECMAScript 6
6.	Array.prototype.entries()	It returns an array iterator object, which can be used to loop through keys and values of arrays.	ECMAScript 6
7.	Array.prototype.keys()	It returns an array iterator object along with the keys of the array.	ECMAScript 6
8.	Array.prototype.values()	it provides the value of each key.	ECMAScript 6
9.	Array.prototype.fill()	It fills the specified array elements with a static value	ECMAScript 6

ES6 Array destructuring

- Destructuring means to break down a complex structure into simpler parts. With the syntax of destructuring, you can extract smaller fragments from objects and arrays. It can be used for assignments and declaration of a variable.
- Destructuring is an efficient way to extract multiple values from data that is stored in arrays or objects. When destructuring an array, we use their positions (or index) in an assignment.

Sample Illustrations

```
var arr = ["Hello", "World"]

// destructuring assignment
var [first, second] = arr;

console.log(first); // Hello
console.log(second); // World
```

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

// destructuring assignment
var[color1, color2, color3] = colors;

console.log(color1); // Violet
console.log(color2); // Indigo
console.log(color3); // Blue
```

Output

```
Violet
Indigo
Blue
```

```
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
```

Output

```
Violet
Blue
Yellow
```

ES6 Map

- ES6 is a series of new features that are added to the JavaScript. Prior to ES6, when we require the mapping of keys and values, we often use an object. It is because the object allows us to map a key to the value of any type.
- ES6 provides us a new collection type called **Map**, which holds the key-value pairs in which values of any type can be used as either keys or values. A Map object always remembers the actual insertion order of the keys. Keys and values in a Map object may be primitive or objects. It returns the new or empty Map.
- Maps are ordered, so they traverse the elements in their insertion order.

Syntax and Creation

```
var map = new Map([iterable]);
```

The **Map ()** accepts an optional iterable object, whose elements are in the key-value pairs.

Map Properties

S.no.	Properties	Description
1.	**Map.prototype.size**	This property returns the number of key-value pairs in the Map object.

Example

```
var map = new Map();
   map.set('John', 'author');
   map.set('arry', 'publisher');
   map.set('Mary', 'subscriber');
   map.set('James', 'Distributor');

console.log(map.size);
```

Output

```
4
```

Map Methods

S.no.	Methods	Description
1.	Map.prototype.clear()	It removes all the keys and values pairs from the Map object.
2.	Map.prototype.delete(key)	It is used to delete an entry.
3.	Map.prototype.has(value)	It checks whether or not the corresponding key is in the Map object.
4.	Map.prototype.entries()	It is used to return a new iterator object that has an array of key-value pairs for every element in the Map object in insertion order.
5.	Map.prototype.forEach(callbackFn[, thisArg])	It executes the **callback** function once, which is executed for each element in the Map.
6.	Map.prototype.keys()	It returns an iterator for all keys in the Map.
7.	Map.prototype.values()	It returns an iterator for every value in the Map.

ES6 Set

- A set is a data structure that allows you to create a collection of unique values. Sets are the collections that deal with single objects or single values.
- Set is the collection of values similar to arrays, but it does not contain any duplicates. It allows us to store unique values. It supports both primitive values and object references.
- As similar to maps, sets are also ordered, i.e., the elements in sets are iterated in their insertion order. It returns the set object.

Syntax

```
var s = new Set("val1","val2","val3");
```

Example

Example

```
let colors = new Set(['Green', 'Red', 'Orange', 'Yellow', 'Red']);
console.log(colors);
```

All the elements of a Set must be unique. Therefore the set **colors** in the above example only contain four distinct elements. We will get the following output after the successful execution of the above code.

Output

```
Set { 'Green', 'Red', 'Orange', 'Yellow' }
```

Set Methods

The set object includes several methods, which are tabulated as follows:

S.no.	Methods	Description
1.	**Set.prototype.add(value)**	It appends a new element to the given value of the set object.
2.	**Set.prototype.clear()**	It removes all the elements from the set object.
3.	**Set.prototype.delete(value)**	It removes the element which is associated with the corresponding value.
4.	**Set.prototype.entries()**	It returns a new iterator object, which contains an array of each element in the Set object in insertion order.
5.	**Set.prototype.forEach(callbackFn[, thisArg])**	It executes the callback function once.
6.	**Set.prototype.has(value)**	This method returns true when the passed value is in the Set.
7.	**Set.prototype.values()**	It returns the new iterator object, which contains the values for each element in the Set, in insertion order.

ES6 Objects

ES6 Objects

- Objects are the collection of key/value pairs that can be modified throughout the lifecycle of an object as similar to hash map or dictionary. Objects allow us to define custom data types in JavaScript.
- Unlike primitive data types, we can represent complex or multiple values using objects. The values can be the array of objects or the scalar values or can be the functions. Data inside objects are unordered, and the values can be of any type.
- An object can be created by using curly braces **{...}** along with an optional properties list. The property is a **"key:value"** pair, where the key is a string or a property name, and the value can be anything.

Creating Objects

- **Creating an empty object:**

```
var obj_1 = {};
var obj_2 = new Object();
var obj_3 = Object.create(null);
```

- **Creating an object:**

```
var person = {
    "full-name" : "John Doe",
     age: 35,
     address: {
       address_line1: "Clear Trace, Glaslyn, Arkansas",
       "postal code": "76588-89"
     }
};
```

- **Creating an object with `new`:**

```
function Tree(type1, height1, age1) {
    this.type = type1;
    this.height = height1;
    this.age = age1;
}

var mapleTree = new Tree("Big Leaf Maple", 80, 50);
```

Creating an object with prototypes (syntax):

```
var obj = Object.create(Object.prototype [,properties]);
```

Examples:

```
var myChild = Object.create(Object.prototype); //Object {}
myChild = Object.create({a:10, b:30}); // Object {a=10, b=30}

var myObj = {
    a : 10,
    b : 30
};
myChild = Object.create(myObj);
myChild.a;        //10
myChild.b;        //30
```

Accessing Object Properties

```
var myObject = {
    name: "luggage",
    length: 75,
    specs: {
      material: "leather",
      waterProof: true
    }
}
```

- **Getting and setting properties:**

```
myObject["name"];                    // "luggage"
myObject.name;                       // "luggage"
myObject.specs.material;             // "leather"
myObject["specs"]["material"];       // "leather"
myObject.width;                      // undefined
myObject.tags.number;                // TypeError thrown
```

```
myObject["name"] = "suitcase";       // name : "suitcase"
myObject.name = "suitcase";          // name : "suitcase"
myObject.width = 40;                 // creates a new property
myObject.tags.number = 6;            // TypeError thrown
```

Working with Object Properties

- Deleting properties:

```
delete myObject.length;
delete myObject.height;
```

true, even though it doesn't exist in the object

- Testing and enumerating properties:

```
var myObject = {
    one: "property value one",
    two: "property value two"
}
```

```
for (var myVar in myObject) {
    if (myObject.hasOwnProperty(myVar)) {
      console.log(myVar);
    }
}
```

Accessing Object Properties

- **Property `get` and `set`:**

```
var obj = {
    a : 45,
   get double_a() {return this.a * 2},
   set modify_a(x) {this.a -= x;}
};
```

```
obj.a;              // 45
obj.double_a;       // 90
obj.modify_a = 40;
obj.a;              // 5
obj.double_a;       // 10
```

- **Object method example:**

```
var myObj = {
    print: function() {
      console.log("Hello World!");
    }
};

myObj.print();
```

Sample Illustration

In ES6

```
var department = 'dep_name';
var emp = {
    id : 102,
    name : 'Anil',
    [department]:'Production'
}
console.log(emp);
```

Output

```
{ id: 102, name: 'Anil', dep_name: 'Production' }
```


ES6
ES6 Classes

ES6 Classes

- Classes are an essential part of object-oriented programming (OOP). Classes are used to define the blueprint for real-world object modeling and organize the code into reusable and logical parts.
- Before ES6, it was hard to create a class in JavaScript. But in ES6, we can create the class by using the **class** keyword. We can include classes in our code either by class expression or by using a class declaration.
- A class definition can only include **constructors** and **functions**. These components are together called as the data members of a class. The classes contain **constructors** that allocates the memory to the objects of a class. Classes contain **functions** that are responsible for performing the actions to the objects.

Syntax and Illustration

Syntax: Class Declaration

```
class Class_name{
}
```

```
class Student{
    constructor(name, age){
    this.name = name;
    this.age = age;
    }

}
```

Instantiating an Object from class

Syntax

```
var obj_name = new class_name([arguements])
```

Example

```
var stu = new Student('Peter', 22)
```

Accessing functions

- The object can access the attributes and functions of a class. We use the **'.' dot notation (**or **period)** for accessing the data members of the class.

Syntax

```
obj.function_name();
```

Example

```
'use strict'
class Student {
  constructor(name, age) {
    this.n = name;
    this.a = age;
  }
  stu() {
    console.log("The Name of the student is: ", this.n)
    console.log("The Age of the student is: ",this. a)
  }
}

var stuObj = new Student('Peter',20);
stuObj.stu();
```

Class inheritance

- Before the ES6, the implementation of inheritance required several steps. But ES6 simplified the implementation of inheritance by using the **extends** and **super** keyword.
- Inheritance is the ability to create new entities from an existing one. The class that is extended for creating newer classes is referred to as **superclass/parent** class, while the newly created classes are called **subclass/child class**.
- A class can be inherited from another class by using the **'extends'** keyword. Except for the constructors from the parent class, child class inherits all properties and methods.

Syntax

```
class child_class_name extends parent_class_name{
}
```

A class inherits from the other class by using the **extends** keyword.

Example

```
'use strict'
class Student {
  constructor(a) {
   this.name = a;
  }
}
class User extends Student {
  show() {
    console.log("The name of the student is:  "+this.name)
  }
}
var obj = new User('Sahil');
obj.show()
```

Method Overriding and Inheritance

- It is a feature that allows a child class to provide a specific implementation of a method which has been already provided by one of its parent class.
- There are some rules defined for method overriding -
- The method name must be the same as in the parent class.
- Method signatures must be the same as in the parent class.

```
class Parent {
  show() {
    console.log("It is the show() method from the parent class");
  }
}
class Child extends Parent {
  show() {
    console.log("It is the show() method from the child class");
  }
}
var obj = new Child();
obj.show();
```

ES6 Strings

JavaScript string

- is an object which represents the sequence of characters. Generally, strings are used to hold text-based values such as a person name or a product description.

In JavaScript

- Any text within the single or double quotes is considered as a string. There are two ways for creating a string in JavaScript:
  - By using a string literal
  - By using string object (using the **new** keyword)

By using a string literal

```
var stringname = "string value";
```

By using String object (using the new keyword)

Here, we will use a **new** keyword for creating the string object. The syntax for creating the string object is given below:

```
var stringname = new String ("string literal");
```

ES6 String Methods

Apart From Traditional String Methods from JavaScript Additional Methods are :

S.no.	Methods	Description	JavaScript Version
1.	startsWith	It determines whether a string begins with the characters of a specified string.	ECMAScript 6
2.	endsWith	It determines whether a string ends with the characters of a specified string.	ECMAScript 6
3.	includes	It returns true if the specified argument is in the string.	ECMAScript 6
4.	repeat	It returns a new string repeated based on specified count arguments.	ECMAScript 6

Summary

In this lesson, you should have learned how to:

- ES6 Fundamentals
- Functions and Arrays
- Classes and Objects
- Strings