ES 6

Objective

- New ES6 syntax
- Classes
- Modules
- Arrow Functions
- Promises
- Collections
- Array extensions

New ES6 syntax

- let – declare block-scoped variables using the let keyword.
- let vs. var – understand the differences between let and var.
- const – define constants using the const keyword.
- Default function parameters – learn how to set the default value for parameters of a function.
- Spread operator – learn how to use the spread operator effectively.
- Object literal syntax extensions – provide a new way to define object literal.
- for…of – learn how to use the for…of loop to iterate over elements of an iterable object.

let Keyword

- The let keyword is similar to the var keyword, except that these variables are blocked-scope.
  - let variable_name;
- In JavaScript, blocks are denoted by curly braces {} , for example, the if else, for, do while, while, try catch and so on.

```
let x = 10;
if (x == 10) {
    let x = 20;
    console.log(x); // 20:  reference x inside the block
}
console.log(x); // 10: reference at the begining of the script
```

```
for (let i = 0; i < 5; i++) {
    setTimeout(() => console.log(i), 1000);
}
```

const Keyword

- The const keyword creates a read-only reference to a value.
  - const CONSTANT_NAME = value;
- const keyword declares blocked-scope variables.
- It can't be reassigned.
- variables created by the const keyword are "immutable".

```
const RATE = 0.1;
RATE = 0.2; // TypeError
---------------------------
const RED; // SyntaxError
---------------------------
const person = { age: 20 };
person.age = 30; // OK
person = { age: 40 }; // TypeError

const person = Object.freeze({age: 20});
person.age = 30; // TypeError
```

Default Parameters

- Parameters are what you specify in the function declaration whereas the arguments are what you pass into the function.
- if you don’t pass the arguments into the function, its parameters will have the default values of undefined.

```
function say(message) {
    console.log(message);
}

say(); // undefined
```

```
function say(message) {
    message = typeof message !== 'undefined' ? message : 'Hi';
    console.log(message);
}
say(); // 'Hi'
```

- ES6 provides you with an easier way to set the default values for the function parameters use the assignment operator (=) and the default value after the parameter name to set a default value for that parameter.

```
function say(message='Hi') {
    console.log(message);
}

say(); // 'Hi'
say(undefined); // 'Hi'
say('Hello'); // 'Hello'
```

Rest parameters

- A rest parameter allows you to represent an indefinite number of arguments as an array.
- The last parameter (args) is prefixed with the three-dots (...). It's called a rest parameter (...args).

```
function fn(a,b,...args) {
   //...
 }
```

```
function fun(...input){
    let sum = 0;
    for(let i of input){
        sum+=i;
    }
    return sum;
}
console.log(fun(1,2)); //3
console.log(fun(1,2,3)); //6
console.log(fun(1,2,3,4,5)); //15
```

JavaScript spread operator

- ES6 provides a new operator called spread operator that consists of three dots (...).
- The spread operator allows you to spread out elements of an iterable object such as an array, map, or set.

```
const odd = [1,3,5];
const combined = [2,4,6, ...odd];
console.log(combined);// [ 2, 4, 6, 1, 3, 5 ]
```

- The spread operator (...) unpacks the elements of an iterable object.
- The rest parameter (...) packs the elements into an array.

JavaScript for...of loop

- for...of that iterates over an iterable object such as
  - Built-in Array, String, Map, Set, ...
  - Array-like objects such as arguments or NodeList
  - User-defined objects that implement the iterator protocol.

```
const ratings = [
    {user: 'John',score: 3},
    {user: 'Jane',score: 4},
    {user: 'David',score: 5},
    {user: 'Peter',score: 2},
];

let sum = 0;
for (const {score} of ratings) {
    sum += score;
}

console.log(`Total scores: ${sum}`); // 14
```

```
let str = 'abc';
for (let c of str) {
    console.log(c);
}
```

ES6 Modules

- An ES6 module is a JavaScript file that executes in strict mode only. It means that any variables or functions declared in the module won't be added automatically to the global scope.
- To export a variable, a function, or a class, you place the export keyword

```
// cal.js
export let a = 10, b = 20, result = 0;

export function sum() {
  result = a + b;
  return result;
}

export function multiply() {
  result = a * b;
  return result;
}
```

```
import {a, b, result, sum, multiply } from './cal.js';
sum();
console.log(result); // 30

multiply();
console.log(result); // 200
```

- To import everything from a module as a single object
- import { a } from './cal.js';

ES6 class

- ES6 introduced a new syntax for declaring a class as shown in this example.
- JavaScript automatically calls the constructor() method when you instantiate an object of the class.

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

let john = new Person("John Doe");
let name = john.getName();
console.log(name); // "John Doe"
```

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


JavaScript static methods

- static methods are bound to a class, not the instances of that class. Therefore, static methods are useful for defining helper or utility methods.
- In ES6, you define static methods using the static keyword.

```
class Person {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
    static createAnonymous(gender) {
        let name = gender == "male" ? "John Doe" : "Jane Doe";
        return new Person(name);
    }
}
let anonymous = Person.createAnonymous("male");
```


JavaScript static properties

- Like a static method, a static property is shared by all instances of a class. To define static property, you use the static keyword followed by the property name.
- To access a static property in a class constructor or instance method, you use the following syntax:
  - className.staticPropertyName;
  - this.constructor.staticPropertyName;

```
class Item {
   constructor(name, quantity) {
     this.name = name;
     this.quantity = quantity;
     this.constructor.count++;
   }
   static count = 0;
   static getCount() {
     return Item.count;
   }
 }

 let pen = new Item('Pen', 5);
 let notebook = new Item('notebook', 10);

 console.log(Item.getCount()); // 2
```

JavaScript static properties

- Classes we can implement inheritance to make child inherits all methods of Parent Class. This can be done using the extends and super keywords.
- We use the extends keyword to implement the inheritance in ES6. The class to be extended is called a base class or parent class.
- The super() method in the constructor is used to access all parent's properties and methods that are used by the derived class.


```
<html>
<body>
   <h2>JavaScript Class Inheritance</h2>
   <p id="demo"></p>
   <script>
      class Mall {
         constructor(shopname) {
            this.shopname = shopname;
         }
         shopispresent() {
            return this.shopname + ' is present in the  ';
         }
      }
      class Shop extends Mall {
         constructor(name, mallname) {
            super(name);
            this.mallname = mallname;
         }
         showshop() {
            return this.shopispresent() + this.mallname;
         }
      }
      let newMall = new Shop("Domino", "Select City Walk Mall");
      document.getElementById("demo").innerHTML = newMall.showshop();
   </script>
</body>
</html>
```

JavaScript Class Inheritance

Domino is present in the Select City Walk Mall

Arrow function

- JavaScript Promise are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.
- The promise constructor takes only one argument which is a callback function
- The callback function takes two arguments, resolve and reject
- Perform operations inside the callback function and if everything went well then call resolve.
- If desired operations do not go well then call reject.

```
var promise = new Promise(function(resolve,
reject){
    //do something
});
```

```
let p = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve(10);
    }, 3 * 100);
});

p.then((result) => {
    console.log(result);
    return result * 2;
}).then((result) => {
    console.log(result);
    return result * 3;
});
```

Map object

- Before ES6, we often used an object to emulate a map by mapping a key to a value of any type. But using an object as a map has some side effects:
  - An object always has a default key like the prototype.
  - A key of an object must be a string or a symbol, you cannot use an object as a key.
  - An object does not have a property that represents the size of the map.
- let map = new Map([iterable]);

```
let john = {name: 'John Doe'},
    lily = {name: 'Lily Bush'},
    peter = {name: 'Peter Drucker'};
```

- Add elements to a Map
- userRoles.set(john, 'admin');

Map object methods

- clear() – removes all elements from the map object.
- delete(key) – removes an element specified by the key. It returns if the element is in the map, or false if it does not.
- entries() – returns a new Iterator object that contains an array of [key, value] for each element in the map object. The order of objects in the map is the same as the insertion order.
- forEach(callback[, thisArg]) – invokes a callback for each key-value pair in the map in the insertion order. The optional thisArg parameter sets the this value for each callback.
- get(key) – returns the value associated with the key. If the key does not exist, it returns undefined.
- has(key) – returns true if a value associated with the key exists or false otherwise.
- keys() – returns a new Iterator that contains the keys for elements in insertion order.
- set(key, value) – sets the value for the key in the map object. It returns the map object itself therefore you can chain this method with other methods.
- values() returns a new iterator object that contains values for each element in insertion order.

Set object

- ES6 provides a new type named Set that stores a collection of unique values of any type. To create a new empty Set, you use the following syntax:
- let setObject = new Set();

Set object methods

- add(value) – appends a new element with a specified value to the set. It returns the Set object, therefore, you can chain this method with another Set method.
- clear() – removes all elements from the Set object.
- delete(value) – deletes an element specified by the value.
- entries()– returns a new Iterator that contains an array of [value, value] .
- forEach(callback [, thisArg]) – invokes a callback on each element of the Set with the this value sets to thisArg in each call.
- has(value) – returns true if an element with a given value is in the set, or false if it is not.
- keys() – is the same as values() function.
- [@@iterator] – returns a new Iterator object that contains values of all elements stored in the insertion order.