**\*ES6**

ECMAScript 2015 was the second major revision to JavaScript.

ECMAScript 2015 is also known as ES6 and ECMAScript 6.

New Features of ES6

* [The let keyword](https://www.w3schools.com/js/js_es6.asp#mark_let)
* [The const keyword](https://www.w3schools.com/js/js_es6.asp#mark_const)
* [Arrow Functions](https://www.w3schools.com/js/js_es6.asp#mark_arrow)
* [For/of](https://www.w3schools.com/js/js_es6.asp#mark_forof)
* [Map Objects](https://www.w3schools.com/js/js_es6.asp#mark_map)
* [Set Objects](https://www.w3schools.com/js/js_es6.asp#mark_set)
* [Classes](https://www.w3schools.com/js/js_es6.asp#mark_class)
* [Promises](https://www.w3schools.com/js/js_es6.asp#mark_promise)
* [Symbol](https://www.w3schools.com/js/js_es6.asp#mark_symbol)
* [Default Parameters](https://www.w3schools.com/js/js_es6.asp#mark_param)
* [Function Rest Parameter](https://www.w3schools.com/js/js_es6.asp#mark_rest)

**\*Arrow Function**

ES6 arrow functions provide you with an alternative way to write a shorter syntax compared to the function expression.

**Exa : without arrow**

let add = function (x, y) {

return x + y;

};

console.log(add(10, 20));

**with arrow**

let add = (x, y) => x + y;

console.log(add(10, 20));

or

let add = (x, y) => {

return x + y; };

**\*For/of**

Exa :

const cars = ["BMW", "Volvo", "Mini"];

for (let x of cars) {

console.log(x) + "<br>";

}

**\*Map Objects**

* A Map holds key-value pairs where the keys can be any datatype.
* A Map remembers the original insertion order of the keys.
* A Map has a property that represents the size of the map.

You can create a JavaScript Map by:

* Passing an Array to new Map()
* Create a Map and use Map.set()

Exa :

const fruits = new Map([

["apples", 500],

["bananas", 300],

["oranges", 200]

]);

Console.log(fruits.get(“apples”))

**\*Set Objects**

* A JavaScript Set is a collection of unique values.
* Each value can only occur once in a Set.
* A Set can hold any value of any data type
* You can create a JavaScript Set by:
* Passing an Array to new Set()
* Create a new Set and use add() to add values
* Create a new Set and use add() to add variables

**Exa:**

const letters = new Set(["a","b","c"]);

console.log(letters.size);

**\*Classes**

* Use the keyword class to create a class.
* Always add a method named constructor():
* A JavaScript class is not an object.
* It is a template for JavaScript objects.

**Exa :**

class Car {

constructor(name, year) {

this.name = name;

this.year = year; }

}

const myCar = new Car("Ford", 2014);

**\*Default Parameters**

If a function in JavaScript is called with **missing arguments** (less than declared), the missing values are set to undefined.

**Exa :**

function myFunction(x, y = 2) {

return x \* y;

}

**\*Function rest parameter**

**Rest parameter** is an improved way to handle function parameter, allowing us to more easily handle various input 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.

Exa :

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

**Advance Topics**

**1). Constructor:**

A constructor is a special function that creates and initializes an object instance of a class. In JavaScript, a constructor gets called when an object is created using the new keyword.

The purpose of a constructor is to create a new object and set values for any existing object properties.

function Person(name,age,profession)

    {

        this.name  =name;

        this.age = age;

        this.profession = profession;

    }

    const father = new Person("john",45,'Teacher');

    console.log(father);

    console.log(father.name);

    console.log(father.age);

    console.log(father.profession);

**2). Prototype:**

he prototype is an object that is associated with every functions and objects by default in JavaScript, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible.

Every function includes prototype object by default.

The prototype object is special type of enumerable object to which additional properties can be attached to it which will be shared across all the instances of it's constructor function.

Exa :

 function Person(firstName, lastName) {

        this.FirstName = firstName || "unknown";

        this.LastName = lastName || "unknown";

    };

    // Construct Prototype

    prsn=new Person("ABC","Test")

    console.log(prsn.FirstName+" "+prsn.LastName);

    Person.prototype.getFullName = function () {

        console.log(this.FirstName+" "+this.lastName);

        return this.FirstName+" "+this.lastName;

    }

    Person.prototype.getFullName()

**3).Prototype Inheritance:**

The Prototypal Inheritance is a feature in javascript used to add methods and properties in objects. It is a method by which an object can inherit the properties and methods of another object. Traditionally, in order to get and set the [[Prototype]] of an object, we use Object.

Exa :

 function Person(firstName, lastName) {

        this.FirstName = firstName || "unknown";

        this.LastName = lastName || "unknown";

    };

    // Construct Prototype

    // prsn=new Person("ABC","Test")

    Person.prototype.getFullName = function () {

        return this.FirstName + " " + this.LastName;

    }

    function Student(firstName, lastName, schoolName, grade) {

        Person.call(this, firstName, lastName);

        this.SchoolName = schoolName || "unknown"; this.Grade = grade || 0;

    }

    Student.prototype = new Person;

    Student.prototype.constructor = Student

    var std = new Student("James", "Bond", "XYZ", 10);

    console.log(std);

    console.log(std.getFullName());

**4). Closure:**

*Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.*

Inner function can access variables and parameters of an outer function (however, cannot access arguments object of outer function).

Exa :

|  |
| --- |
| function Counter(params) { |
|  | var counter = 0; |
|  | function IncreaseCounter() |
|  | { |
|  | return counter += 1; |
|  | }; |
|  | return IncreaseCounter; |
|  | } |
|  | var counter = Counter(); |
|  | console.log(counter()); |
|  | console.log(counter()); |
|  | console.log(counter()); |
|  | console.log(counter()); |

**5). Callback**

A [JavaScript](https://www.simplilearn.com/tutorials/javascript-tutorial/introduction-to-javascript) callback is a function which is to be executed after another function has finished execution.

A more formal definition would be - Any function that is passed as an argument to another function so that it can be executed in that other function is called as a callback function.

We need callback functions because many JavaScript actions are asynchronous, which means they don't really stop the program (or a function) from running until they're completed, as you're probably used to. Instead, it will execute in the background while the rest of the code runs.

Exa :

function display(some)

    {

        console.log(some);

    }

function mycalc(num1,num2)

{

    let sum = num1 + num2;

    return sum;

}

let result = mycalc(5,5);

display(result);

**\*Promises**

* **Promises** are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.
* Prior to promises events and callback functions were used but they had limited functionalities and created unmanageable code.
* Multiple callback functions would create callback hell that leads to unmanageable code. Also it is not easy for any user to handle multiple callbacks at the same time.  
  Events were not good at handling asynchronous operations.

**Exa :**

const myprom = new Promise((resolve,reject)=>{

        const a = 1

        const b = 5

        if (a == b) {

            resolve("Success")

            // console.log("true");

        } else {

            reject("try again")

            console.log("false");

        }

    })

    myprom.then((res)=>{

        console.log(res);

    }).catch((res)=>{

        console.log(res);

    })

**\*FetchPromises:**

Exa:

 fetch('https://jsonplaceholder.typicode.com/posts').then((response)=>response.json()).then((resdata)=>{

        HTMLDATA = ''

        resdata.forEach(element => {

            HTMLDATA+=`<tr>

                <td>${element.id}</td>

                <td>${element.title}</td>

            </tr>`

        });

        document.querySelector('#dispdata').innerHTML = HTMLDATA

    }).catch()