**Introduction**

Most of the challenges that you have done related to JavaScript were based on ECMAScript 5 (ES5). But as JavaScript is an evolving programming language, hence it has added and have revised existing features in the new versions of the language.

The most recent standardized version is called ECMAScript 6 (ES6), released in 2015. This new version of the language adds some powerful features that will be covered in this section, including:

* let Keyword
* constKeyword
* Arrow Function
* For of loop
* Spread attributes
* Destructuring
* Classes
* Static methods
* Getters and Setters

**Note**: Not all browsers support ES6 features. If you use ES6 in your own projects, you may need to use a program (transpiler) to convert your ES6 code into ES5 until browsers support ES6.

Also the current track don’t cover few of the advance topics related to Array and Promises, which is something you can learn later while in practice.

### Let Keyword

We have used the var keyword to define variables in ES5. The scope the **var** keyword is the entire enclosing function. Here’s an example:

function helloworld() {

for (var x = 0; x < 2; x++) {

// x should only be scoped to this block because this is where we have defined x.

console.log(x);

}

// But it turns out that x is available here as well!

console.log(x); // 2

}

As can be seen that variable “x” was declared within the for loop and then was later accessed outside the loop as well. In smaller programs, its not an issue at all as you can control the variables and their declarations. But in larger programs, sometimes changing a value outside the scope creates a huge impact. Hence with es6 we have a **let** Keyword which allows us to do exactly what var keyword use to do it but with little advance features.

As mentioned earlier, “**let**” is similar to “**var**” but let has scope. **let** is only accessible in the block level it is defined.

Take a look at this program:

if (true) {

let a = 40;

console.log(a); //40

}

console.log(a); // undefined

In the above example variable ‘a’ is defined inside If statement and so it’s not accessible outside the function.

Another example:

let a = 50; // declared First

let b = 100;

if (true) {

let a = 60; // declared Again

var c = 10;

console.log(a/c); // 6

console.log(b/c); // 10

}

console.log(c); // 10

console.log(a); // 50

In this example, we can see that we declared variable “a” once above the condition while once inside. In the very last line if we see, it returns 50 because it doesn’t cater the inside changed value as it was outside the scope.

### Const Keyword

**Const** is used to assign a constant value to the variable. And the value cannot be changed. Its fixed.

Example:

const a = 50;

a = 60; // shows error. You cannot change the value of const.

Another example:

const b = "Constant variable";

b = "Assigning new value"; // shows error.

Consider another example, this time with an array.

const LANGUAGES = ['Js', 'Ruby', 'Python', 'Go'];

LANGUAGES = "Javascript"; // shows error.

LANGUAGES.push('Java'); // Works fine.

console.log(LANGUAGES); // ['Js', 'Ruby', 'Python', 'Go', 'Java']

This may be little confusing.

Consider in this way. Whenever you define a const variable, JavaScript references the address of the value to the variable. In our example the variable ‘LANGUAGES’ actually references to the memory allocated to the array. So you cannot change the variable to reference some other memory location later. Throughout the program it only references to the array.

### Arrow Function

Functions in ES6 have changed a bit. I mean the syntax.

// Old Syntax

function oldOne() {

console.log("Hello World..!");

}

// New Syntax

var newOne = () => {

console.log("Hello World..!");

}

The new syntax may be confusing a little bit. But lets try and learn the new syntax.

There are two parts of the syntax.

var newOne = ()  
=> {}

The first part is just declaring a variable and assigning the function (i.e) () to it. It just says the variable is actually a function.  
Then the second part is declaring the body part of the function. The arrow part with the curly braces defines the body part.

So lets take few examples to learn more:

function helloWorld() {

console.log("Hello World");

}

The above is an old way of writing a function, if we try and replicate this to a newer form, it becomes:

var helloWorld = () => {

console.log("Hello World");

}

Now lets see what happens when we have some parameters.

For example:

function addNumbers(a,b) {

console.log(a+b);

}

This function now with new version becomes:

let addNumbers = (a,b) => {

console.log(a+b);

}

### Default Parameters:

If you are familiar with other programming languages then default parameters isn’t new to you.

Default parameters are parameters which are given by default while declaring a function. But it’s value can be changed when calling the function.

Example

let Func = (a, b = 10) => {

return a + b;

}

Func(20); // 20 + 10 = 30

In the above example, we are passing only one parameter. The function makes use of the default parameter and executes the function.

Consider another example:

let NotWorkingFunction = (a = 10, b) => {

return a + b;

}

NotWorkingFunction(20); // NAN. Not gonna work.

When you are calling the function with parameters they get assigned in the order. (i.e) the first value gets assigned to the first parameter and the second value gets assign to the second parameter and so on..

In the above example, the value 20 gets assigned to parameter ‘a’ and ‘b’ is not having any value. So we are not getting any output.

But,

NotWorkingFunction(20, 30); // 50;

Works fine.

### For of loop

**for..of** is very similar to for..in with slight modification.

**for..of** iterates through list of elements (i.e) like Array and returns the elements (not their index) one by one.

let arr = [2,3,4,1];

for (let value of arr) {

console.log(value);

}

Output:

2

3

4

1

Note: variable “value” outputs each element in the array not the index.

Another Example

let string = "Javascript";

for (let char of string) {

console.log(char);

}

Output:

J

a

v

a

s

c

r

i

p

t

Yes. It works for string too.

### Spread attributes

Spread attributes (...) help to spread the expression as the name suggests. In simple words, it converts a list of elements to an array and vice versa.

Example without spread attributes:

let SumElements = (arr) => {

console.log(arr); // [10, 20, 40, 60, 90]

let sum = 0;

for (let element of arr) {

sum += element;

}

console.log(sum); // 220.

}

SumElements([10, 20, 40, 60, 90]);

In the example above we are declaring a function to accept array as parameter and returning its sum. Its simple.

Now consider the same example with spread attributes

let SumElements = (...arr) => {

console.log(arr); // [10, 20, 40, 60, 90]

let sum = 0;

for (let element of arr) {

sum += element;

}

console.log(sum); // 220.

}

SumElements(10, 20, 40, 60, 90); // Note we are not passing array here. Instead we are passing the elements as arguments.

Hence in the above example, the spread attribute converts the list of elements (i.e) the parameters to an array.

So where we can use spread attribute?

Lets take an example:

Math.max(10, 20, 60, 100, 50, 200); // returns 200.

Math.max is a simple method that returns the maximum element from given list. It doesn’t accept an array.

let arr = [10, 20, 60];

Math.max(arr); // Shows error. Doesn't accept an array.

So lets use our savior.

let arr = [10, 20, 60];

Math.max(...arr); // 60

In the above example, the spread attribute converts the array to list of elements.

### Destructuring

Destructuring is an expression that allows us to extract properties from an object, or items from an array. Let’s say we have an address object like this:

const address = {

street: 'Pallimon',

city: 'Kollam',

state: 'Kerala'

};

Now, somewhere else we need to access these properties and store their values in a bunch of variables: here is both ES5 and ES6 versions

//ES5

var street = address.street;

var city = address.city;

var state = address.state;

//ES6

const { street, city, state } = address;

We can also destructure arrays but we use square brackets ([]) instead of curly braces ({}).

//ES5

var values = ['Hello', 'World'];

var first = values[0];

var last = values[1];

//ES6

const values = ['Hello', 'World'];

const [first, last] = values;

### Classes

Classes are the core of object oriented programming (OOP). They make your code more secure and encapsulated. Using classes gives your code a nice structure and keeps it oriented.

One way to define a class is using a class declaration. To declare a class, you use the class keyword with the name of the class (“Rectangle” here).

class Rectangle {

constructor(height, width) {

this.height = height;

this.width = width;

}

}

Another Example:

class myClass{

constructor(name,age){

this.name=name;

this.age=age;

}

}

const Home= new myClass("said",20);console.log(Home.name)// said

To learn more about the classes, visit the [official documentation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes).

### Static methods

Static methods are introduced in ES6. And it is pretty much easy to define it and use it.

class Example {

static Callme() {

console.log("Static method");

}

}

Example.Callme();

Output:  
Static method

Note that I didn’t use the keyword ‘function’ inside Class.

And I can call the function without creating any instance for the class.

### Getters and Setters

Getters and setters and one of the useful feature introduced in ES6. It will come in handy if you are using classes in JS.

Example without getters and setters:

class People {

constructor(name) {

this.name = name;

}

getName() {

return this.name;

}

setName(name) {

this.name = name;

}

}

let person = new People("Jon Snow");

console.log(person.getName());

person.setName("Dany");

console.log(person.getName());

Output:  
Jon Snow  
Dany

I think the above example is self-explanatory. We have two functions in class People that helps to set and get the name of the person.

Example with getters and setters

class People {

constructor(name) {

this.name = name;

}

get Name() {

return this.name;

}

set Name(name) {

this.name = name;

}

}

let person = new People("Jon Snow");

console.log(person.Name);

person.Name = "Dany";

console.log(person.Name);

In the above example, you can see there are two functions inside class People with ‘get’ and ‘set’ properties. The ‘get’ property is used to get the value of the variable and ‘set’ property is used to set the value to the variable.

And you can see that getName function is called without parenthesis. And setName function is called without parenthesis and it’s just like assigning a value to the variable.

// The purpose of use strict is to indicate that the code should be executed in strict mode, With strict mode, you can not, for example, use undeclared variables.

"use strict";







