ES6

AGENDA

Introduction

Features

Examples

INTRODUCTION

ES6 or the ECMAScript 2015 is the 6th and major edition of the ECMAScript language specification standard. It defines the standard for the implementation of JavaScript and it has become much more popular than the previous edition ES5.

ES6 comes with significant changes to the JavaScript language. It brought several new features like, let and const keyword, rest and spread operators, template literals, classes, modules and many other enhancements to make JavaScript programming easier and more fun. In this article, we will discuss some of the best and most popular ES6 features that we can use in your everyday JavaScript coding.

FEATURES

- Let and Const Keyword
- Arrow Functions
- Default Parameters
- Template Literals
- Rest and Spread
- Destructuring
- Classes
- Promises
- Modules

LET AND CONST KEY WORD

The keyword "let" enables the users to define variables and on the other hand, "const" enables the users to define constants. Variables were previously declared using "var" which had function scope and were hoisted to the top. It means that a variable can be used before declaration. But, the "let" variables and constants have block scope which is surrounded by curly-braces "{}" and cannot be used before declaration.

ARROW FUNCTION

ES6 provides a feature known as Arrow Functions.
It provides a more concise syntax for writing function expressions
by removing the "function" and "return" keywords.

Arrow functions are defined using the fat arrow (=>) notation.
It is evident that there is no "return" or "function"
keyword in the arrow function declaration.

We can also skip the parenthesis in the case when there is exactly one parameter,
but will always need to use it when you have zero or more than one parameter.

But, if there are multiple expressions in the function body,
then we need to wrap it with curly braces ("{}").
We also need to use the "return" statement to return the required value.

DEFAULT PARAMETERS

In ES6, users can provide the default values right in the signature of the functions.

```
let calculateArea = function(height = 100, width = 50) {
   // logic
}
```

TEMPLATE LITERALS

ES6 introduces very simple string templates along with placeholders for the variables.

The syntax for using the string template is ${PARAMETER}
and is used inside of the back-ticked string.

```
let name = `My name is ${firstName} ${lastName}`
```

DESTRUCTURING ASSIGNMENT

Destructuring is one of the most popular features of ES6.
The destructuring assignment is an expression that makes it
easy to extract values from arrays, or properties from objects, into distinct variables.

There are two types of destructuring assignment expressions,
namely, Array Destructuring and Object Destructuring. It can be used in the following manner :

```
//Array Destructuring
let fruits = ["Apple", "Banana"];
let [a, b] = fruits; // Array destructuring assignment
console.log(a, b);
```

Object Destructuring

```
let person = {name: "Peter", age: 28};
let {name, age} = person; // Object destructuring assignment
console.log(name, age);
```

PROMISES

In ES6, Promises are used for asynchronous execution.
We can use promise with the arrow function as demonstrated below.

```
var asyncCall =  new Promise((resolve, reject) => {
  // do something
  resolve();
}).then(()=> {
  console.log('DON!');
})
```

A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future. A Promise is in one of these states: pending:

initial state, neither fulfilled nor rejected.

fulfilled: meaning that the operation was completed successfully.

rejected: meaning that the operation failed.

CLASSES

Previously, classes never existed in JavaScript. Classes
are introduced in ES6 which looks similar to classes in other object-oriented languages,
such as C++, Java, PHP, etc. But, they do not work exactly the same way.
ES6 classes make it simpler to create objects,
implement inheritance by using the "extends" keyword and also reuse the code efficiently.

In ES6, we can declare a class using the new "class" keyword followed by the name of the class.

```
class UserProfile {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getName() {
   console.log(`The Full-Name is ${this.firstName} ${this.lastName}`);
  }
}
let obj = new UserProfile('John', 'Smith');
obj.getName(); // output: The Full-Name is John Smith
```

MODULES

Previously, there was no native support for modules in JavaScript.

ES6 introduced a new feature called modules, in which each module is represented by a separate ".js" file.

We can use the "import" or "export" statement in a module to import or export variables,

functions, classes or any other component from/to different files and modules.

```
export var num = 50;
export function getName(fullName) {
  //data
};
import {num, getName} from 'module';
console.log(num); // 50
```

FETCH

The Fetch API provides a JavaScript interface for accessing and manipulating parts of the protocol, such as requests and responses. It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network.

This kind of functionality was previously achieved using XMLHttpRequest.

Fetch provides a better alternative that can be easily used by other technologies such as Service Workers.

Fetch also provides a single logical place to define other HTTP-related concepts such as CORS and extensions to HTTP.

ARRAY HELPER METHODS

Map
Filter
Foreach

CLOSURE

A closure is a feature of JavaScript that allows inner functions to access the outer scope of a function. Closure helps in binding a function to its outer boundary and is created automatically whenever a function is created. A block is also treated as a scope since ES6. Since JavaScript is event-driven so closures are useful as it helps to maintain the state between events.

```
function calc(a)
{
   function docalc(x)
   {
      return a+x
   }
 return docalc
}

let getcalc=calc(5)

console.log(getcalc(3))
console.log(getcalc(5))
```

CURRYING

Currying in JavaScript is a process in functional programming in which you can transform a function with multiple arguments into a sequence of nesting functions. It returns a new function that expects the next argument inline.

In other words, instead of a function taking all arguments at one time, it takes the first one and returns a new function, which takes the second one and returns a new function, which takes the third one, and so on, until all arguments have been fulfilled.

```
function sum(a) {
  return (b) => {
    return (c) => {
      return a + b + c
    }
  }
}
console.log(sum(1)(2)(3)) // 6
```