ES6

ECMAScript 2015

What is ES6?

New version of JavaScript

What is ECMAScript?

ECMAScript is Standard, JavaScript is implementation of ECMAScript in to browsers

JavaScript History

- 1995 - JavaScript Created
- 1997 - ECMAScript 1
- 2009 - ECMAScript 5
- 2015 - ECMAScript 6

Why we need to use ES6?

- Better features
- Compatible with latest libraries
- Improved performance

ES6 Features

- let keyword
- const keyword
- Iterator & for-of
- template literals
- spread operator

- classes
- class inheritance

- Default function parameters
- Enhanced object literals
- Arrow Functions
- Arrow Functions and 'this' scope
- Destructuring assignment

- generators
- promises

let keyword

let allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used.

let keyword

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2;  // different variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

const keyword

Constants are block-scoped, much like variables defined using the let statement. The value of a constant cannot change through re-assignment, and it can't be redeclared.

```
const PI = 3.14;

PI=2.4 //TypeError: Assignment to constant variable.
```

Default parameters

Default function parameters allow formal parameters to be initialized with default values if no value or undefined is passed.

```
//ES5

var getAccounts = function(limit) {
    var limit = limit || 10
    // ...
}

var link = function (height, color, url) {
    var height = height || 50
    var color = color || 'red'
    var url = url || 'https://node.university'
    // ...
}
```

Default parameters

Default function parameters allow formal parameters to be initialized with default values if no value or undefined is passed.

```
/* ES6, we can put the default values right in the
signature of the functions like below
*/

const getAccounts = function(limit = 10) {
    //...
}

const link = function(height = 50,
                      color = 'red',
                      url = 'https://node.university') {
    //...
}
```

Template literals

Template literals are string literals allowing embedded expressions.

- enclosed by the back-tick (` `) (grave accent) character instead of double or single quotes.
- can contain place holders. These are indicated by the Dollar sign and curly braces (${expression})

```
var first="Varma";
var last="Bhupatiraju";

//ES5
var myName = "My full name is " +  first + " " + last;

//ES6
var myFullName = `My full name is ${first} ${last}`;
```

Rest & Spread Operator

The **spread syntax** allows an expression to be expanded in places where multiple arguments (for function calls) or multiple elements (for array literals) or multiple variables (for destructuring assignment) are expected.

```
function(...args) {
    // args instanceof array === true
}

[head, ...tail] = [1, 2, 3, 4];

// head === 1, tail === [2, 3, 4]
new Date(...[2014, 1, 1]);
```

Destructuring assignment

Destructuring is a convenient way of extracting multiple values from data stored in (possibly nested) objects and Arrays.

```
// Array matching
let [x, y] = [22, 18]
console.log(x, y) // 22, 18

let [a, , b] = [1, 2, 3]
console.log(a, b) // 1, 3

// Using the splat operator
let [a, b, ...c] = [1, 2, 3, 4, 5]
console.log(a, b, c) // 1, 2, [3, 4, 5]


// Object matching
{ a, b } = { a: 'foo', b: 'bar' }
console.log(a, b) // "foo", "bar"
```

Arrow Functions

ES6 fat arrow functions have a shorter syntax compared to function expressions and lexically bind the this value. Arrow functions are always anonymous.

```
//ES5
var f = function(data) {
    //..
    return data;
}

//ES6
const f = (data) => {
    //...
    return data;
}

//ES5
var sum=function(a,b) { return a+b; }

//ES6
const sum = (a,b) => a+b;
```

Class

JavaScript classes introduced in ECMAScript 2015 are syntactical sugar over JavaScript's existing prototype-based inheritance. JavaScript classes provide a much simpler and clearer syntax to create objects and deal with inheritance.

```
//ES5
function Cat(name) {
    this.name = name
}

Cat.prototype.speak = function() {
    console.log('Bark, woof')
}

Cat.prototype.attack = function(enemy
    console.log('Feed me')
}
```

```
//ES6
class Cat {
    constructor(name) {
        this.name = name
    }

    speak() {
        console.log('Bark, woof')
    }

    attack(enemy) {
        console.log('Feed me')
    }
}


const myCat = new Cat("Monty");

console.log(myCat.speak());
```