-Lecture 4-

Chapter 2 – **Advanced Javascript Concepts**

Part II : EcmaScript2015 - ES6

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz

Prerequisites

- ❑ Basic JavaScript Proficiency.
- ❑ Basic Knowledge of Functions and Scope.
- ❑ Object-Oriented Programming

EcmaScript2015 - ES6

Objectives

- ➔ Understand the new features and syntax enhancements introduced in ES6
- ➔ Analyze the advantages of ES6's destructuring assignments

1. Origin of ES6

- JavaScript was named after the success of Java.
- Netscape submitted JavaScript to ECMA International (European Computer Manufacturer's Association) for standardization.
- ECMA is a standardization organization which led to a new standard language called **ECMAScript**, which is a standardized version of JavaScript.

From this came the various ESi versions:

- ➔ **ES1:** June1997 - **ES2:** June1998 - **ES3:** December1999 - **ES4:** Discontinued, **ES5:** December 2009.
- ➔ **ES6 / ES2015:** June 2015, ES6 and ES2015 indicate the same version under two different names.
- ➔ **ES2016 (ES7)**: June 2016: 7th edition of ECMAScript.
- ➔ **ES2017 (ES8):** June 2017: 8th edition of ECMAScript.
- ➔ **ES.Next:** dynamic term referring to the next version of ECMAScript to come.

1. Origin of ES6

ES6 is characterized by a set of structures added to the basic JavaScript language with a view to simplifying it and making it more convenient to use.
In this chapter, we'll look at some of these of these structures, which will later be used in React.js.

2. New ES6 concepts

2.1. let, const and string templates

a) Let and Const:

You can declare variables or constants whose scope is limited to a block.
A block exists in **if** statements, **for/while** loops, **switches** and **try/catch** statements.

2. New ES6 concepts

2.1. let, const and string templates

Example :

```
let teacher = "Ali";
if (teacher === "Ali") {
let anotherTeacher = "Mourad";
}
console.log(anotherTeacher); // ReferenceError!
for (let i = 0; i < 5; i++){
setTimeout(function(){
console.log(i);
},1000);
}
console.log(i); // ReferenceError!
```

Constants can also be created as follows:

```
const favFood = "Bananas";
favFood = "chips";// Uncaught TypeError: Assignment to constant variable.
const person; // Uncaught SyntaxError: Missing initializer in const declaration
```

2. New ES6 concepts

2.1. let, const and string templates

Let and **const** have a block scope,
unlike **var** which has a global or functional scope (inside a function).

2. New ES6 concepts

2.1. let, const and string templates

b) Strings template:

ES2015 allows you to perform easier string treatments (known as interpolations) by using apostrophes and enclosing variables in the **${}** structure.

Example of string interpolation as ${1+1}`;

```
var person = "Ali";
`My name is ${person}`;
// My name is Ali
```

2. New ES6 concepts

2.1. let, const and string templates

Interpolation avoids many concatenations:

```
var firstName = "Mehdi";
var lastName = "Lamine";
var title = "teacher";
var employer = "IT Institute";
// exemple avec beaucoup de concatenations...
var greeting1 = "Hi, my name is " + firstName + " " + lastName + ", and I am an " + title + " at " + employer +
"!";
// avec l'interpolation et les templates
var greeting2 = `Hi, my name is ${firstName} ${lastName}, and I am an ${title} at ${employer}!`;
```

2. New ES6 concepts

2.2. Arrow Functions

Arrow functions are simplified declarations of functions using => arrow notation.

They replace the function keyword by adding a few rules:

- ➔ If the arrow function contains only one instruction, an implicit return is added.
- ➔ If the arrow function has several instructions, {} must be used (like ordinary functions).
- ➔ If the arrow function has only 1 argument, there's no need to enclose it in parentheses (unlike in the case of multiple arguments, parentheses are mandatory).
- ➔ Arrow functions are always anonymous.

2. New ES6 concepts

2.2. Arrow Functions

Example:

```
// basic examples:
var add = (a, b) => a + b;
add(2,3);// 5
var sup = str => str.toUpperCase();
var multilineArrowFunction = a => {
let b = a * a;
return b + a;
}
```

```
// callback examples:
var arr = [1,2,3,4];
// with a normal declaration of function
arr.map(function(val){
return val*2;
})
// with arrow function
arr.map(val => val *2)
```

2. New ES6 concepts

2.2. Arrow Functions

- ➔ Another difference between an arrow function (**=>**) and a function declared by `function` is the meaning of of this.

- ➔ Arrow functions make a lexical binding of the value of **this** (for more details, see : MDN)

- ➔ Before arrow functions, each new function defined its own **this** value (a new object in the case of a **constructor** function, **undefined** if the function call is in strict mode, the **object** in whose context the function is declared as a function within an object, etc.).

- ➔ For simplicity's sake, arrow functions lexically define their **this** context to be closer object-oriented programming.

2. New ES6 concepts

2.2. Arrow Functions

The basic syntax of arrow functions is as follows:

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression. // equivalent to :{ return expression; }
// Parentheses are optional when there's only one parameter name:
(singleParam) => { statements }
singleParam => { statements }
// The parameter list for a function with no parameters should be written with a pair of
parentheses.
() => { statements }
```

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

a) Default parameter:

ES2015 lets you add default values to function parameters.

Example:

```
// old statement - gives unexpected results!
function multiply(a, b) {
    return a * b;
}
multiply(5, 2); // 10
multiply(5); // NaN ! erreur

// with ES6
function multiply(a, b = 1) {
return a * b;
}
multiply(5, 2); // 10
multiply(5); // 5
```

➔ (for more details on the Default parameter, see : MDN)

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

b) Rest:

ES2015 provides 2 operators with the same syntax: **spread** and **rest** in the form of `(...)`.

The **rest** operator, is used in the definition of a function in the list of its parameters and indicates "the rest" of its parameters, according to the following syntax:

```
function f(a, b, ...theArgs) {
// ...
}
```

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

b) Rest:

Example:

```
function data(a,b,...c){
    console.log(a,b,c);
}
data(1,2,3,4,5); // 1, 2, [3,4,5] rest of arguments considered as a table.
```

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

b) Rest:

By using this operator, the rest of the arguments are real arrays and will be treated as arrays. Unlike the **arguments** object, which is an object containing the list of arguments to a function.

Example:

```
function checkArguments() {
    return Array.isArray(arguments)
}

function checkArgumentsES2015(...x) {
    return Array.isArray(x);
}

checkArguments(1, 2, 3); // false
checkArgumentsES2015(1, 2, 3); // true
```

➔ (for more details on the operator Rest, see : exploringjs)

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

c) Spread:

The spread operator is used to convert a compound element (iterable or enumerated type) such as: array, expression, string or object. into a list of the simple elements that compose it, using the following syntax as appropriate:

- For function calls:
  ```
  myFunction(...iterableObj);
  ```
- For array elements and strings:
  ```
  [...iterableObj, '4', 'five', 6];
  ```
- For objects (new for ECMAScript 2018):
  ```
  let objClone = { ...obj };
  ```

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

c) Spread:

- ➔ The **spread** operator does the opposite of the rest operator: instead of converting a list of values into an array into an array, it explodes an array into a list of values.
- ➔ This is why the **spread** operator is used when calling a function, unlike the **rest** operator, which is used in the declaration of a function.

Example:

```
var arr = [1,2,3,4];
function addFourNumbers(a,b,c,d){
return a + b + c + d;
}
addFourNumbers(...arr);
```

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

c) Spread:

➔ The **spread** operator also applies to the **`arguments`** object, even if it's not an array.

Example:

```
function addThree(a,b,c) {
    return a + b + c;
 }
function addThreeArgs() {
    return addThree(...arguments);
}
addThree(1, 2, 3); // 6
addThreeArgs(1, 2, 3); // 6
```

➔ (For more details on the operator spread see : exploringjs)

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

c) Spread:

The Spread operator is widely used in **React.js**. It simplifies operations such as concatenations.

Example without the spread operator:

```
a = [1,2,3];
b = [4,5,6];
c = a.concat(b);
console.log("c: " + c);
```

with the Spread operator:

```
a = [1,2,3];
b = [4,5,6];
c = [...a, ...b]; //spread operator
console.log("c: " + c);
```

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

Uses of spread in React:

- ➔ You can easily add elements between two arrays:
  `[...a, 'something', ...b];`
- ➔ Spread is easier to use and visualizes well what is going to be done.
- ➔ Arrays can be cloned as follows: `clone = [...a];`
- ➔ In React, you can combine two objects using the Spread operator, and also add other properties.

2. New ES6 concepts

2.3. Default parameters, Rest and Spread

Uses of spread in React:

Example:

```
const person = { name: "Jhon"};
const student = { ID: "21", GPA: "3.0"};

const newObject = { ...person, ...student, semester: '3'};
console.log(newObject);
```

2. New ES6 concepts

2.4. Shorthand notation and object destructuring

a) Shorthand notation:

ES2015 provides a number of improvements in object manipulation, resulting in shorter code with fewer repetitions.

Example:

```
var obj = {
    firstName: "Ali",
    sayHi: function(){
        return "Hello from ES5!";
    },
    sayBye() { // eliminating : and the function keyword
        return "Bye from ES2015!";
    }
}
var person = "Ali";
var es5Object = {person: person};
es5Object; // {person: "Ali"}

var es2015Object = {person}; // variable in an object
es2015Object; // {person: "Ali"}
```

2. New ES6 concepts

2.4. Shorthand notation and object destructuring

a) Shorthand notation:

In ES2015, when adding a method to an object, we can eliminate the "`:`" and the **`function`** keyword.

The following 2 examples are equivalent:

```
var o1 = {
    sayYo: function() {
        console.log("Yo!");
    }
};

var o2 = {
    sayYo() {
        console.log("Yo!");
    }
}
```

2. New ES6 concepts

2.4. Shorthand notation and object destructuring

b) Object Destructuring:

- ES2015 lets you destructure objects or arrays.
- Destructuring assignment is a JavaScript expression that extracts data from objects or arrays and transforms them into distinct variables.
- This functionality is used if you want to make a multiple assignment (of several variables at the same time).

➔ (for more details on Object Destructing, see : MDN)

2. New ES6 concepts

2.4. Shorthand notation and object destructuring

b) Object Destructuring:

Example:

```
var obj = {
    a:1,
    b:2,
    c:3
};

var {d,e,f} = obj; // instead of d= obj.a ; e=obj.b ...
d; // 1
e; // 2
f; // 3
```

Example with a table:

```
var arr = [1,2,3,4];
var [a,b,c,d] = arr; // instead of a= arr[0]…
a; // 1
b; // 2
c; // 3
d; // 4

var [first,second] = [1,2];
first; // 1
second; // 2
```

2. New ES6 concepts

2.4. Shorthand notation and object destructuring

b) Object Destructuring:

Example with exchange of elements in an array:

```
var [a, b] = [1, 2];
a; // 1
b; // 2
[a, b] = [b, a];
a; // 2
b; // 1
```

2. New ES6 concepts

2.5. Classes and associated syntax

This addition to JavaScript is frequently used in **React** and **Angular 2**.
In ES2015, a class is declared as follows:

```
class Person {
    constructor(firstName, lastName){
        this.firstName = firstName;
        this.lastName = lastName;
    }
    sayHi(){
        return `${this.firstName} ${this.lastName} says hello!`;
    }
    static isPerson(person){
        return Person.constructor === Person;
    }
}
```

2. New ES6 concepts

2.5. Classes and associated syntax

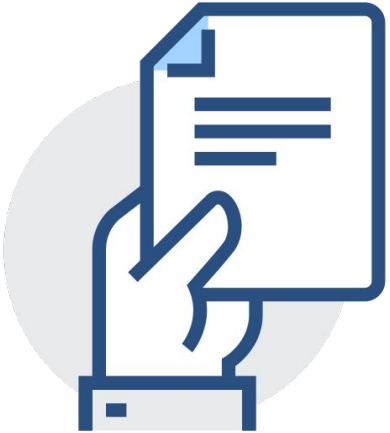
- The function called **`constructor`** must have this name (in order to execute it when the keyword **`new`** is used).
- If we only use **`Person()`** we'll get a **TypeError** with the message "**`Class constructor Person cannot be invoked without 'new'`**".
- If you want to add functions directly to the class (like class methods in Java), use **`static`**.

➔ (For more details on Classes see : MDN)
➔ (For more details on Static see : MDN)

Lab Exercises Submission Guidelines

- **Deadline:**
  At the end of each Lab session (no later than Saturday at 23:59)
  To: adil.chekati@univ-constantine2.dz

- **File's Name to be submitted:**
  CAW_Lab%_Gr%_NAMEPair1_NAMEPair2.zip
  Example : "CAW_Lab1.part1_Gr1_CHEKATI_BOUZENADA.zip"



Textbook

➔ All academic materials will be available on:

Google Drive.
E-learning platform of Constantine 2 University.
Google Classroom.

aoa5lne

Google Classroom

SCAN ME!

References

➔ **Book:**
Haverbeke, Marijn - *Eloquent JavaScript: A Modern Introduction to Programming*- (2019)

Online Resource:
Mozilla Developer Network-"JavaScript Guide"
(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide)

Next Lecture

-Lecture 5-

Chapter 3 – **Introduction to React.JS**

Adil **CHEKATI**, PhD

adil.chekati@univ-constantine2.dz

Questions, & comments...

adil.chekati@univ-constatine2.dz