INTRODUCTION À ECMASCRIPT 6 : LE JAVASCRIPT DE MAINTENANT

Sommaire

I. Introduction ES 6 :

Les technologies web ne cessent d'évoluer, aujourd'hui nous allons parler d'ES6, la nouvelle version du langage JavaScript. Elle apporte un lot conséquent de nouveautés, de l'ajout de nouvelles fonctions à la bibliothèque standard, à plusieurs nouvelles syntaxes, dont les classes, les modules et bien plus encore.

Historiquement, le langage JavaScript a longtemps été mis au deuxième plan dans la conception des pages web, servant principalement à réaliser des animations diverses. Aujourd'hui les choses sont totalement différentes, on parle désormais d'application web, entièrement contrôlées via du code JavaScript. Malheureusement, le langage n'a pas été conçu pour créer des applications complexes, il en résulte ainsi des syntaxes très lourdes.Cela va changer très prochainement avec l'arrivée d'une nouvelle version du langage : **ECMAScript 6, alias JavaScript 2015** !

ES6 a été pensé pour créer des applications web facilement maintenables, tout en restant compatibles avec le code existant. L'idée a été d'ajouter de nouvelles fonctionnalités au langage. Ainsi, la bibliothèque standard **s'enrichit de nouvelles méthodes**, mais surtout, **le langage adopte de nouvelles syntaxes**, comme les classes ou les modules (pour ne citer qu'eux) permettant d'**avoir du code beaucoup plus structuré et lisible**. Un article de blog serait beaucoup trop court pour faire le point de toutes les nouveautés qui vont s'offrir à nous, je vous propose cependant de faire un tour d'horizon de certaines fonctionnalités très intéressantes.

II. Evolution du code JS :

Je vous invite à créer une page web html vide et à y coller le contenu du code ci-dessous. Votre navigateur n'est pas compatible ES6 ? Ce n'est pas un problème pour aujourd'hui ! Effectivement nous allons utiliser deux scripts qui vont nous transformer à la volée le code ES6 en code compatible avec votre navigateur.

```
<!doctype html>
<html lang="fr">
<head>
   <meta charset="utf-8" />
   <title>ECMAScript 6 - Demos</title>
</head>
<body>
   <script src="https://google.github.io/traceur-
compiler/bin/traceur.js"></script>
   <script src="https://google.github.io/traceur-
compiler/src/bootstrap.js"></script>
   <script type="module">
      // Notre code ES6 ici
   </script>
</body>
</html>
```

Cette opération "magique" est réalisée par le compilateur **Traceur**, un outil mis au point par Google dont le but est de **convertir le JavaScript de demain en JavaScript d'aujourd'hui**. Il est disponible sous plusieurs formes (node, web). Ici nous utilisons la version web. Ainsi, tout le code JavaScript placé dans la balise *script type="module"*sera automatiquement converti à la volée.

III. Le mot clé let

Le mot clé let permet de déclarer une variable limitée à la portée d'un bloc, c'est-à-dire qu'elle ne peut être utilisée que dans le bloc où elle a été déclarée, ce qui n'est pas le cas avec var.

```
function swap(x, y) {
   if (x != y) {
      var old = x;
      let tmp = x;
      x = y;
      y = tmp;
   }

   console.log(typeof(old));   // number
   console.log(typeof(tmp));   // undefined
}
```

Comme prévu, la variable *tmp* n'existe que dans le bloc conditionnel, à l'inverse de *old* qui existe partout dans la fonction.

IV. Templates et chaînes de caractères

Vous avez sans doute déjà créé des chaînes de caractères en utilisant des concaténations, bien que cela fonctionne, ce n'est pas forcément pratique et lisible. Le Template String va nous permettre de déclarer des chaînes avec des variables à évaluer à l'intérieur. À la différence d'une chaîne de caractères classique,**on utilise des *anti quotes* (`) à la place des *double quotes***.

```
let me = "Yannick";
let mAge = 29;
let result = `Je suis ${me} et j'ai ${mAge} ans`;
```

La variable *result* aura comme valeur *"Je suis Yannick et j'ai 29 ans"*. Sachez que vous n'êtes pas limité à une variable, vous pouvez évidemment faire des opérations et même utiliser des fonctions. (si vous vous appelez Bertrand, Michel ou Louise, que vous n'avez pas 29 ans, ça marche aussi, évidemment !).

V. Les paramètres par défaut

Prenons un cas classique avec l'écriture d'une fonction qui doit additionner deux nombres. Si les paramètres ne sont pas définis alors votre programme plante, pour éviter cela il faut soit être sûr que les paramètres sont bien utilisés ou alors les tester dans le code, ce qui revient à faire quelque chose comme ça :

```
var add = function (x, y) {
   var x = typeof(x) === "number" ? x : 0;
   var y = typeof(y) === "number" ? y : 0;
   return x + y;
};
```

Demain vous pourrez utiliser des **paramètres par défaut dans vos fonctions et méthodes**, ça se passe de commentaires non ?

```
let add = function (x = 0, y = 0) {
   return x + y;
};
```

VI. Boucle for - of

Il n'existe toujours pas de mot clé *foreach* en JavaScript 2015, cependant il y a désormais une nouvelle façon d'itérer sur un tableau en utilisant le couple for-of.

```
let languages = [ "C", "C++", "C#", "JavaScript" ];

for (let language of languages) {
    console.log(language);
}
```

Il est aussi possible d'utiliser un système clé/valeur en spécifiant un tableau en paramètre et en exploitant la fonction Array::entries() qui va renvoyer un objet de type Iterator contenant le couple clé/valeur pour chaque élément du tableau.

```
for (let [index, language] of languages.entries()) {
    console.log(`Index: ${index} => Valeur: ${language}`);
}
```

VII.Évolution des objets littéraux

Les objets littéraux peuvent être vus comme une structure, ils permettent de stocker des variables et des méthodes. Il est désormais possible d'avoir une écriture beaucoup plus légère pour les déclarer, en se passant du mot clé *function*.

```
let Engine = {
   initialize() {
      this.display("Init") ;
   },
   display(message) {
      console.log(message);
   }
};
```

VIII. Fonctions lambda alias Arrow function

Cette nouveauté est **géniale**, vous allez voir pourquoi dans quelques instants ! Voici un premier exemple avec la fonction Array.sort().

```
let values = [65, 7, 78, 1, 32, 66];
values.sort((a, b) => { return b - a });
```

Évidemment, l'intérêt des Arrow Function ne s'arrête pas à une écriture plus light ! En réalité le contexte *this* du scope courant est conservé à l'intérieur de la déclaration. Voyons plutôt l'exemple ci-dessous.

```
let Engine = {
   initialize (element) {
      element.addEventListener("keydown", (event) => {
         this.display(event.type);
      }, false);
   },
   display (message) {
      console.log(message);
   }
};
```

IX. Les classes et l'héritage

On termine avec une chose très attendue par une majorité de développeurs : les classes et l'héritage "facile". Comme pour un objet littéral, vous n'avez pas besoin d'utiliser le mot clé *function* pour déclarer une méthode.

```
class Person {
   constructor(name, age) {
      this.name = name;
      this.age = age;
   }

   toString() {
      return `Hi I'm ${this.name} and I'm ${this.age} years old!`;
   }
```

```
}

let p = new Person("John", 18);
console.log(p.toString());
// Hi I'm John and I'm 18 years old
```

La création d'une classe est relativement facile, vous noterez que nous **ne mettons pas de point virgule après sa déclaration**. Une chose à noter est l'absence d'encapsulation des données, ainsi tous vos champs et méthodes seront accessibles depuis l'extérieur de la classe. Rassurez-vous, il existe une possibilité pour créer des membres privés, avec des Symbols, mais cela va au-delà de notre article de présentation.

Passons maintenant à l'héritage Simple (il n'existe pas d'héritage multiple avec ES6).

```
class Developer extends Person {
  constructor(name, age, language) {
    super(name, age);
    this.language = language;
  }

  toString() {
    return super.toString() + ` :: I'm a Developer who likes ${this.language}`;
  }
}

let d = new Developer("Yannick", 29, "JavaScript");
```

```
console.log(d.toString());
//  Hi I'm Yannick and I'm 29 years old :: I'm a Developer who likes JavaScript
```

Sans surprise l'héritage de classe se fait avec le mot clé *extends,* les méthodes parentes s'appellent quant à elles avec le mot clé *super*.

X. Conclusion

Toutes ces nouveautés vont permettre de créer des applications web beaucoup plus rapidement et facilement. L'ajout des classes avec un système d'héritage facile est un vrai plus et permettra de structurer plus facilement le code. Les fonctions lambda sont aussi un vrai plus, car elles offrent une écriture très légère et permettent de conserver la référence du pointeur this du bloc courant.