ECE PARIS
ÉCOLE D'INGÉNIEURS

Introduction à JavaScript et NodeJS

2019/2020

APERCU

Dans ce document, vous apprendrez à programmer en JavaScript avec NodeJS.

Présentation de JavaScript

JavaScript, souvent abrégé JS, est un langage de programmation de haut niveau. Ce langage est caractérisé par le fait qu'il soit typé dynamiquement, orienté prototype et multi-paradigme. JavaScript est l'une des trois technologies qui font parties du noyau du World Wide Web.

Présentation et utilisation de NodeJS

NodeJS est un environnement d'exécution JavaScript open-source et multi-plateforme. Il permet d'exécuter du JavaScript en dehors d'un navigateur. Une fois installé sur votre machine, vous pouvez exécuter des scripts JavaScript depuis la ligne de commande.

Pour cela, vous allez tout d'abord créer un fichier **index.js** dans lequel vous rentrerez le code suivant:

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

Ensuite, ouvrez votre console et rendez vous à l'endroit où se situe votre fichier **index.js** puis exécutez la commande suivante:

```
node index.js
```

Vous avez désormais exécuté votre premier script JavaScript avec NodeJS, le message “Hello, World!” doit normalement être affiché dans votre console.

1. Les bases en JavaScript

1.1 Syntaxe et commentaires

En JavaScript, il est préférable de terminer une instruction avec un point-virgule (**;**), mais il est aussi possible d’omettre ce point-virgule dans le cas d’un saut à la ligne:

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

console.log("Hello")
console.log("World")
```

Attention: Certains standards demandent à ce que les points-virgules ne soient jamais omis après une instruction.

Il y a deux types de commentaires, les commentaires mono-ligne et les commentaires multi-ligne:

```
// Je suis un commentaire mono-ligne
/* Je suis
   un commentaire
   multi-ligne */
```

1.2 Les variables

Une variable JavaScript est un conteneur permettant de stocker temporairement une information. Il y a trois type de mot clefs permettant de stocker une variable:

- **const :** Stocke une variable en tant que constante, la variable devient donc immuable (sa valeur ne peut plus changer).
- **let :** Stocke une variable en tant que variable local. C’est le mot clef à utiliser majoritairement.
- **var :** Stocke une variable en tant que variable globale. Merci de ne jamais utiliser ce mot clef.

```
// Déclarer une variable et affecter directement une valeur
let x = 25;
// La casse est importante en JavaScript, X est différent de x
let X = 100;
```

```
// Plusieurs variables peuvent être déclarées à la suite
let a = 0, b = 1, c = "test";

// Déclarer une variable puis lui affecter une valeur
let variable;
variable = "ECE Paris";
```

1.3 Les types de valeurs des variables

Les types présentés ici sont appelés “valeurs primitives” car ce sont des données qui ne sont pas des objets et qui n’ont pas de méthode. Les différentes valeurs primitives sont:

```
// Valeur primitive: Number
let n = 5;

// Valeur primitive: String
let s = "Test";

// Valeur primitive: Boolean
let b = true;

// Valeur primitive: Null
let o = null;

// Valeur primitive: undefined
let u = undefined;

console.log(typeof n);
console.log(typeof s);
console.log(typeof b);
console.log(typeof o);
console.log(typeof u);
```

Remarque: lorsqu’une valeur est affectée à une variable qui contenait déjà des données, alors les données déjà présentes sont écrasées:

```
let x = 10;
console.log(x);
x = 25;
console.log(x);
```

1.4 Les opérations mathématiques

Il est possible d'utiliser des opérateurs spécifiques pour le type Number:

```
let x, y = 10, z = -2;

// Addition
x = 10 + 5;
x = x + 1;

// Soustraction
x = x - 4;

// Multiplication
y = y * 2;
let mult = x * y;

// Division
let div = y / z;

// Modulo
let mod = 13 % 3;
```

Attention à la priorité lors des calculs: les parenthèses sont prioritaires sur toute autre opération, puis viennent la multiplication, la division et le modulo et finalement l'addition et la soustraction.

```
let x = 5, y = 10, z = -2;
let priorite = x + y / (4 + z) % 3;
console.log(priorite);
```

Remarque: Il est possible de simplifier les opérations sur les variables avec la syntaxe suivante:

```
let x = 0;
x += 1; // Equivalent à "x = x + 1"
x -= 2; // Equivalent à "x = x - 2"
x *= 3; // Equivalent à "x = x * 3"
x /= 4; // Equivalent à "x = x / 4"
x %= 5; // Equivalent à "x = x % 5"
```

1.5 Chaine de caractère et concaténation

En JavaScript, il est aussi possible d'utiliser l'opérateur + sur la valeur primitive **String**. Cela permet "d'additionner" les chaînes de caractères, on parle alors de concaténation.

```
let prenom = "Jean", espace = " ", nom = "Dupont";
let nom_entier = prenom + espace + nom;
let nom_entier_2 = "Martin" + " " + "Dupont";
let nom_entier_3 = "Valérie " + nom;
```

Il est aussi possible de concaténer un nombre avec une chaîne de caractères:

```
let x = 4;
console.log("La variable x vaut " + x);
```

Depuis les nouvelles versions de JavaScript (ES6), il est aussi possible de concaténer plusieurs variables entre elles avec le principe de modèle de libellés:

```
let str = "World";
let concat = `Hello, ${str}`;
console.log(concat);
```

Remarque: Attention au caractère qui est utilisé pour englober un modèle de libellés !

1.6 Les conditions et la comparaison

En JavaScript, il est possible de tester une valeur afin d'exécuter un bloc de code en fonction du résultat de ce test. Tout d'abord, voici la liste des différents opérateurs de comparaison disponibles:

Symbole	Signification
==	Est égal à (en valeur)
!=	Est différent de (en valeur)
===	Est égal à (en valeur et en type)
!==	Est différent de (en valeur ou en type)
<	Est strictement inférieur à (en valeur)
<=	Est inférieur ou égal à (en valeur)
>	Est strictement supérieur à (en valeur)
>=	Est supérieur ou égal à (en valeur)

Remarque: Il est fortement préférable d'utiliser les opérateurs "**===**" et "**!==**" plutôt que "**==**" et "**!=**" pour des raisons de sûreté de la comparaison. En effet, le code ci-dessous n'aura pas le résultat escompté:

```
console.log('5' == 5);
```

Chaque opérateur de comparaison retourne une valeur de type booléen qui correspondra au résultat de la comparaison. Il est tout à fait possible de stocker le résultat d'une comparaison dans une variable:

```
let check_true = 4 < 10;
let check_false = 4 >= 10;

console.log(check_true);
console.log(check_false);
```

Nous allons désormais aborder les structures conditionnelles qui permettent d'exécuter un bloc de code en fonction de la valeur du booléen. Premièrement le bloc **if**:

```
if (true) {
    console.log("Condition executed");
}

let hour = 9
if (hour > 12) {
    console.log("It is afternoon");
}
```

Ensuite le bloc **if...else**:

```
if (false) {
    console.log("Condition not executed");
} else {
    console.log("Condition executed");
}

let hour = 9;
if (hour > 12) {
    console.log("It is afternoon");
} else {
    console.log("It is morning");
}
```

Et finalement le bloc **if...else if...else**:

```
let hour = 9;
if (hour > 12) {
    console.log("It is afternoon");
} else if (hour < 12) {
    console.log("It is morning");
} else {
    console.log("It is noon");
}
```

1.7 Les opérateurs logiques

En JavaScript, il est possible de reproduire l'algèbre de Boole grâce aux opérateurs logiques:

Opérateur logique	Symbole
AND	&&
OR	||
NOT	!

Chaque opérateur correspond à la fonction de Boole qui lui est associée.

```
if (true && false) {
    console.log("Not executed");
} else if (true && true) {
    console.log("Executed");
}

if (true || false) {
    console.log("Executed");
} else if (false || false) {
    console.log("Not executed");
}

if (!false) {
    console.log("Executed");
}
```

1.8 Le switch

Plutôt que d'utiliser un bloc avec beaucoup de **else if** pour tester plusieurs valeurs différentes, il existe l'instruction **switch**:

```
let hour = 9;

switch(hour) {
    case 8:
        console.log("It is 8");
        break;

    case 9:
        console.log("It is 9");
        break;

    case 10: case 11:
        console.log("It is morning");
        break;

    case 13:
        console.log("It is afternoon");

    case 14:
        console.log("It is 13 or 14");
        break;

    default:
        console.log("It is " + hour);
}
```

1.9 Les boucles

En JavaScript, il y a trois boucles différentes: **while**, **do...while**, et **for**.

```
let x = 0;
while (x < 10) {
    x++; // Equivaut à "x = x + 1" ou "x += 1"
    console.log("x: " + x);
}

for(let y=10; y>=0; y--) { // Equivaut à "y = y - 1" ou "y -= 1"
    console.log("y: " + y);
}

let z = 0;
do {
    z++;
    console.log("z: " + z);
} while(z < 5);
```

1.10 Les fonctions

En JavaScript, une fonction correspond à un bloc de code dont le but est d'effectuer une tâche précise. Pour cela, la syntaxe est la suivante:

```
// Fonction simple
function hello_world() {
    console.log("Hello, World!");
}

// Fonction avec paramètre
function mul(a, b) {
    console.log(a * b);
}

// Fonction avec paramètre et retour
function add(a, b) {
    return a + b;
}

// Fonction avec paramètre et valeur par défaut
function display(str = "Hello") {
    console.log(str);
}

// Utilisation des fonctions
hello_world();
mul(2, 3);
console.log(add(1, 1));
display();
display("Bonjour");
```

Il est aussi possible de créer des fonctions dites **fonctions anonymes**. Il s'agit de fonctions qui sont créées quand JavaScript est en cours d'exécution:

```
let anonymous = function() {
    console.log("Anonymous function");
}

anonymous();
```

Et depuis peu, grâce aux nouvelles versions de JavaScript (ES6), il est possible de simplifier la création de ces fonctions anonymes avec ce qui est communément appelé “**fonction flèche**”:

```
let anonymous = () => {
    console.log("Anonymous function");
}

anonymous();
```

2. Les objets en JavaScript

JavaScript introduit le concept d'objet grâce à son fonctionnement orienté prototype. Il est possible de créer un objet grâce à la syntaxe suivante:

```
let object = {};
console.log(typeof object);
```

Les accolades permettent de déclarer un objet de manière simple. Il est ensuite possible de rajouter des propriétés et des méthodes:

```
let character = {
    name: "John Doe",
    age: 25,
    display_info: function() {
        console.log(this.name);
        console.log(this.age);
    }
};

character.display_info();
console.log(character.name);
```

Il est possible en JavaScript de créer un constructeur afin d'utiliser le mot-clef **new** pour créer nos objets:

```
function Car(name) {
    this.name = name;
    this.wheels = 4;
    this.display = function() {
        console.log("This car is named " + this.name + " and has " +
this.wheels + " wheels.");
    }
}

let c = new Car("Toto");
c.display();
```

Il existe beaucoup d'objets natifs en JavaScript, nous ne les étudierons pas tous. Vous pouvez aller les chercher ici: https://developer.mozilla.org/fr/docs/Web/JavaScript

2.1 L'objet String

L'objet String est l'objet qui gère les chaînes de caractères. Il possède plusieurs propriétés et méthodes bien utiles:

```
let str = new String("Hello, World!");
let ece = str.replace("World", "ECE");

console.log("String length is: " + str.length);
console.log(ece);
```

2.2 L'objet Array

L'objet Array est un objet natif JavaScript qui permet de créer des tableaux:

```
let names = ["Jean", "John", "Lucas"];
console.log(names[1]);
```

Il est aussi possible de créer l'objet Array avec le mot-clef **new**:

```
let array = new Array();

array.push("Hello");
array.push("World");

for (let i=0; i<array.length; i++) {
    console.log(array[i]);
}
```

Il existe plusieurs manières bien plus pratiques de parcourir un objet Array. Tout d'abord, l'objet Array possède une méthode **forEach** permettant de parcourir toutes les valeurs du tableau:

```
let a = ["1", "2", "3"];

a.forEach((value, index) => {
    console.log(index, value);
});
```

Mais il existe aussi deux opérateurs utilisables dans une boucle **for**, l'opérateur **in** et **of**:

```
let array = ["1", "2", "3"];

for (let value of array) {
    console.log(value);
}

for (let index in array) {
    console.log(index);
}
```

Le mot-clef **of** permet de parcourir les valeurs du tableau tandis que le mot-clef **in** permet de parcourir les indices du tableau.

Remarque: Le mot-clef **in** permettant de parcourir les indices d'un tableau, il permet donc de parcourir les propriétés d'un objet:

```
let object = {
    test1: "1",
    test2: "2",
    test3: "3",
};

for (let index in object) {
    console.log(index, object[index]);
}
```