Javascript ES6

Louis Harang - Dylan Correia
Wizards Technologies

Qui sommes-nous ?

Dylan Correia
Formation :
- DUT Informatique (Orsay)
- Master Ingénierie du Web (ESGI)

24 ans
Email : dcorreia@wedigital.garden

Louis Harang
Formation :
- Master Ingénierie du Web (ESGI)

27 ans
Email : cours@narah.io

Développeurs Full Stack @ Wizards Technologies

Faisons connaissance

Présentez vous !

- Prénom
- Dans quelle boîte vous travaillez et avec quelles technos
- Ce que vous connaissez du javascript
- Ce que vous attendez du cours

Sommaire

Introduction

- Histoire
- ECMAScript
- Utilisation du Javascript
- C’est quoi le DOM ?
- Manipuler le DOM *

Les bases de Javascript ES6

- Les types
- Let, const (& var)
- Fonctions *
- Les events *
- Itérations & Conditions *
- Manipuler les strings *

Manipuler des tableaux et des objets

- Les tableaux et les objets
- Les méthodes des tableaux *
- Les méthodes des objets *

Javascript avancé :

- Promesses *
- IIFE
- Template string
- Déstructuration
- Spread operator
- Rest operator

*** TP dans le chapitre**

Modalités d’évaluation

50% TP notés + 50% projet de groupe

Organisation des TP

- Chaque TP sera **numéroté**
- Le rendu doit être mis sur **Github** et envoyé par email
- Chaque repository doit comporter un readme avec :
    - Votre nom
    - Votre prénom
    - Votre adresse email
- Un seul repo pour tous les TP avec 1 dossier par TP comme ci-dessous:
        - TP1
        - TP2
        - TP3
        - ...
        - README.MD
- 1 fichier par exercice sauf si mention contraire dans l'énoncé
- Les TP sont des travaux personnels, si il y a la moindre ressemblance dans le code entre plusieurs personnes = 0 pour tout le monde

Projet de groupe

Plus loin dans le cours nous vous présenterons le sujet pour le projet de Javascript.

- Le sujet sera imposé
- Travail en groupe de 1 à 4 personnes (max).
- Le projet devra contenir un README avec le nom + prénom de tous les développeurs ainsi que vos emails.
- Vous devez utiliser **github** obligatoirement
- Plagiat ou triche = 0
- Tous les membres du groupe doivent travailler de manière égale sur le projet (nous regarderons les commits)

Le barème du projet vous sera transmis en même temps que le sujet.

Introduction

Histoire

Origine du javascript

Développé en dix jours en **mai 1995** pour le compte de la **Netscape Communications Corporation** par **Brendan Eich.** Le langage est inspiré de plusieurs langages, notamment **Java** mais en simplifiant la syntaxe.

Ce nouveau langage est appelé **LiveScript.**

Quelques jours avant sa sortie, Netscape change le nom de **LiveScript** pour **JavaScript.**

En **mars 1996,** Netscape met en œuvre le moteur JavaScript dans son navigateur web. Le succès de ce navigateur contribue à l'adoption rapide de JavaScript dans le **développement web orienté client.**

Netscape soumet alors **JavaScript** à **ECMA International** pour standardisation**.**

ECMAScript

ECMA quoi ?! 1/2

Ecma International - *European association for standardizing information and communication systems* - est une organisation de standardisation active dans le domaine de l'informatique.

ECMA développe des standards sur les langages de script et de programmation, les technologies de télécommunications les produits de sécurité et pleins d’autres sujets informatiques.

Les travaux de standardisation débutent en **novembre 1996** et se terminent en **juin 1997**. Ces travaux donnent naissance au standard ECMA-262 qui spécifie le langage **ECMAScript**.

ECMAScript

Le versionning de ECMAScript 2/2

- **ECMAScript 1 :** Juin 1997
- **ECMAScript 2 :** Juin 1998
- **ECMAScript 3 :** Décembre 1999
- **ECMAScript 4 :** Abandonné car trop complexe, les features de la version 4 ont été réintégrées dans la version 6
- **ECMAScript 5 :** Décembre 2009, pleins d'ajouts importants
- **ECMAScript 6 :** Juin 2015, révolution de la syntaxe et des fonctionnalités
- **ECMAScript 7 à ? :** Depuis la version 6, les versions ECMAScript prendront le nom d'ES<Année en cours>. Chaque année une nouvelle version sort avec des nouveautés.

Aujourd'hui nous sommes donc à **ES 2021**

Dans ce cours nous parlons d'ES 6 (ES 2015), ce n'est pas que nous allons nous arrêter à la version 6, mais simplement ES6 est le nom officieux du **Javascript moderne post-ES6**.

Utilisation du Javascript

Où et comment

Où peut on exécuter du javascript ?

Historiquement, dans votre **navigateur**

Aujourd'hui avec Node, il est aussi possible d'exécuter du **javascript dans le terminal**.

Il est aussi possible de lancer des **scripts JS dans MongoDB .**

Et même dans des **applications desktop** (Electron) et **mobile** (Flutter, React Native...) !

Dans notre cours, nous allons nous concentrer sur l'exécution du javascript dans le **navigateur** et dans le **terminal**.

Utilisation du Javascript

Pour commencer, vous pouvez exécuter du javascript dans la console de votre navigateur ou alors créer un page HTML avec une balise <script>

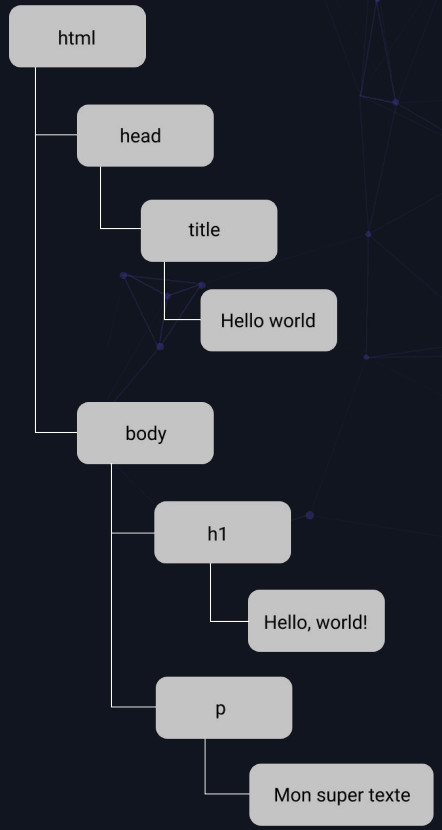
```
<!DOCTYPE html>
<html>
<head>
  <title>My first page</title>
</head>
<body>
</body>
<script type="text/javascript">
  alert('hello world !');
</script>
</html>
```

C’est quoi le DOM ?

L'arborescence d’une page web

Le DOM (Document Object Model) est une représentation d’un document HTML sous forme d’un **arbre d’objets**.

```html
<!DOCTYPE html>
<html lang="fr">
  <head>
    <title></title>
  </head>
  <body>
    <h1> Hello, world !</h1>
    <p>Mon super texte</p>
  </body>
</html>
```



Manipuler le DOM

Browser Object Model (BOM)

Le BOM est le coeur du Javascript dans le navigateur. Le BOM est composé de plusieurs objets qui permettent d'ajouter des fonctionnalités et de l'interaction avec une page web **indépendamment de son contenu**. Attention, même si les BOM sont semblables entre les navigateurs, ils ne sont pas tous forcément disponible. Chaque navigateur en a un implémentation différente.

- **window :** supporté par tous les navigateurs, **window** représentent la fenêtre du navigateur.
- **window.location :** permet de manipuler l'URL de la page
- **window.history :** permet de gérer l'historique de navigation de la page
- **window.document (document) :** document, c'est la représentation de la page web, c'est le DOM
- **window.alert (alert) :** permet d'afficher des modales d'alertes

Manipuler le DOM

Récupérer des infos

Via l'objet **document** on a accès à plusieurs méthode permettant de retrouver des éléments du DOM.

```
/*
 * On récupère l'element avec l'id header
 * Si il y a plusieurs element avec header,
 * la méthode retournera la première occurence
*/
const element = document.getElementById('header');

/*
 * On récupère les elements avec la classe button
 * La méthode retourne un tableau d'element
*/
const elements = document.getElementsByClassName('button');

/*
 * On récupère les elements avec le tag li
 * La méthode retourne un tableau d'element
*/
const elements = document.getElementsByTagName('li');
```

Manipuler le DOM

Récupérer des infos

Via l'objet **document** on a accès à plusieurs méthode permettant de retrouver des éléments du DOM.

```
/*
 * On peut récuperer des informations sur un element du DOM
 * Ici on récupère la valeur de l'attribut src
*/
const element = document.getElementById("avatar").getAttribute("src");

/*
 * On peut aussi récuprer d'autres champs
 *
*/
const element = document.getElementById("avatar").value("src");

/*
 * Pour modifier le texte entre la balise ouvrante et fermante d'un element du DOM
 * on utilise innerHTML
*/
const element = document.getElementById("bio").innerHTML = "Ma super bio";

/*
 * On peut aussi modifier le CSS d'un element
 *
 */
const element = document.getElementById("title").style.color = "red";
```

Manipuler le DOM

Créer des éléments HTML

On peut créer des éléments dans le DOM directement en javascript, pour cela il faut tout d'abord créer un élément puis l'attacher à un autre élément du DOM.

```
// On créer un élement p avec la méthode createElement
const myElement = document.createElement("p");
// On personnalise notre élément
myElement.innerHTML = "Hello world !";
myElement.style.color = "red";

// Il faut attacher notre nouvel élément à un élément existant dans le DOM
// On va récupérer le body et lui attacher notre élément
document.getElementsByTagName('body')[0].appendChild(myElement);
```

Manipuler le DOM

Créer des éléments HTML

On peut bien aussi supprimer un élément du DOM ou encore remplacer un élément par un autre

```
const element = document.getElementById('title');

// On supprime avec removeChild
document.removeChild(element);

// On peut aussi remplacer un élément avec replaceChild
document.replaceChild(newElement, oldElement);
```

Manipuler le DOM

Créer des éléments HTML

On peut aussi utiliser **document.write** pour écrire directement dans le stream HTML. **Attention** cette méthode remplace tout le HTML de la page.

```
document.write("Hello world !");
```

Méthodes à savoir

Pour vous aider à développer et déboguer vos applications client javascript, vous pouvez utiliser les méthodes ci-dessous pour dump vos variables.

```
// console.log permet d'afficher des variables dans la console du navigateur
console.log("foo"); // affiche dans la console : "bar"

// alert permet d'afficher une pop up dans votre navigateur
alert('hello world');
```

TP 1

Manipulation du DOM

Pour chaque exercice, faites l'exo dans un fichier HTML avec une balise script

1. Coder un script JS qui change la couleur de fond du body
2. Créer une page HTML avec 3 balises "p" contenant du texte, coder un script permettant de mettre en gras le 1er élément, en italique le 2ème et en uppercase le 3ème
3. Créer une page HTML avec 3 images, coder un script qui remplace l'attribut src de la 1ère et 2ème image par celui de la 3ème
4. Coder un script qui permet d'ajouter un élément div avec un fond violet, une largeur de 100% et une hauteur de 300px. Créer un autre élément "h1" avec une couleur de texte blanche, en majuscule. Attacher l'élément H1 à l' élément div.
5. Récupérer une page HTML sur internet, puis rajouter un script permettant de supprimer tous les éléments "p"
6. Créer une page HTML avec une liste "Toto" "Tata" "Titi" "Tutu". Créer un nouvel élément li et remplacer le 2ème élément de la liste par le nouvel élément.
7. Récupérer une page HTML sur internet, avec **console.log** afficher la largeur et la couleur de fond de la balise body

Les bases de Javascript

Les types

Déclarer des variables

Même si le Javascript est faiblement typé, il reste typé, il existe plusieurs types directement accessible :

Types primitifs :

- Undefined
- Boolean
- String
- Number

Types complexes :

- Object (Symbol, Array et null sont aussi des objets)
- Function

```
// Non rcommandé, ne fonctionne pas en strict mode
name = 'Jack';

var price = 100;

let isOpen = false;

const name = 'Javascript'
```

Let, const (& var)

Déclarer des variables

La version ES 6 de Javascript a introduit deux nouveaux mot-clés pour déclarer des variables : **const** & **let.** Pour déclarer des variables en JS rien de plus simple.

```
// Non rcommandé, ne fonctionne pas en strict mode
name = 'Jack';

var price = 100;

let isOpen = false;

const name = 'Javascript'
```

Ces 3 mot-clés sont différents à 3 niveaux :

- La portée
- La possibilité d'assigner une nouvelle valeur
- L'accès à la variable avant sa déclaration

Let, const (& var)

La portée des variables

La portée (qu'on appelle souvent **scope**) permet d'identifier où et si on peut utiliser une variable. Les variables peuvent exister à l'intérieur d'un bloc, à l'intérieur d'une fonction ou à l'extérieur d'une fonction et d'un bloc.

Qu'est-ce qu'un bloc ? Un bloc est une section du code que nous définissons à l'aide d'une paire d'accolades { }

```
{
  let message = "Hello world !";
}
```

Même chose pour les fonctions, selon le mot-clé, la portée est différente.

Toute déclaration en dehors d'un bloc ou d'une fonction sera une variable considérée comme **globale.**

```
function test() {
  var message = "hello";
}
```

Let, const (& var)

La portée des variables 2

La portée dans un bloc :

```
{
    let name  = 'Toto';
    const zipCode = 75011;
    var age = 18;
}

console.log(name); // Uncaught ReferenceError: name is not defined
console.log(zipCode);  // Uncaught ReferenceError: zipCode is not defined
console.log(age);  // 25
```

On voit bien ici que la variable déclarée avec **var** est accessible à l'extérieur du bloc, cela rend donc possible l'écrasement de cette variable et donc l'apparition de bogue.

A l'intérieur d'un bloc utilisez **const** & **let** uniquement qui possède une portée de bloc.

Let, const (& var)

La portée des variables 3

La portée dans une fonction :

```
function test(){
    let name  = 'Toto';
    const zipCode = 75011;
    var age = 18;
}
test();

console.log(name); // Uncaught ReferenceError: name is not defined
console.log(zipCode);  // Uncaught ReferenceError: zipCode is not defined
console.log(age);  // Uncaught ReferenceError: age is not defined
```

Aucune variable n’est accessible à l’extérieur, var à une portée au niveau de la fonction.

Let, const (& var)

La portée des variables 4

A retenir :

- **var** : niveau de portée fonctionnelle
- **let :** niveau de portée de bloc
- **const :** niveau de portée de bloc

Let, const (& var)

Assigner une nouvelle valeur

```
let name = "Toto";
const zipCode = 75011;
var age = 18;

// On réassigne des valeurs
name = "Bob"; // name value = 'Bob"
zipCode = 75001; // Uncaught TypeError: Assignment to constant variable.
age = 78; // age value = 78
```

Sur l'exemple ci-dessus, on peut voit qu'on peut tout à fait modifier les valeurs d'une variable déclarée avec **let** ou **var.**

Pour const, comme son nom le laissait présager, il n'est pas possible de modifier la valeur... enfin presque

Let, const (& var)

Assigner une nouvelle valeur, const

On peut modifier la valeur d'une variable déclarée avec const dans le cas où celle-ci est déclarée sous forme d'objet. Il est alors possible de modifier les propriétés de l'objet sans problème. Cependant il n'est pas possible d'attribuer un nouvel objet.

```
const links = {
    'github': 'https://github.com/javascript'
}

links.github = 'https://github.com/nodejs'; // Ok

links = {}; // Uncaught TypeError: Assignment to constant variable.
```

Let, const (& var)

Accéder à la variable avant sa déclaration

Vous ne devriez jamais essayer d'accéder à une variable sans la déclarer. Si le cas se présente, voyons comment les variables peuvent se comporter.

En mode non strict, on peut déclarer la variable avec var après lui avoir assignée une valeur.

Pourquoi ça marche alors que la variable est déclarée après ?

Cela ça s'appelle le **hoisting** (qu'on peut traduire par "hissé"). C'est le comportement de base de javascript qui va "remonter" toutes les déclarations en haut de la portée.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
x = 5; // On assigne 5 à x

elem = document.getElementById("demo");
elem.innerHTML = x; // On affiche la variable

var x;
</script>
</body>
</html>
```

Let, const (& var)

Accéder à la variable avant sa déclaration

Si on tente de faire la même chose avec let, on aura une erreur :

Uncaught ReferenceError: can't access lexical declaration 'x' before initialization

Les variables déclarées avec **let** & **const** sont remontée en haut du bloc mais ne sont pas initialisées : le bloc est conscient de la variable mais ne peut pas l'utiliser car elle n'est pas déclarée.

Avec **const** ce n'est pas possible d'accéder à la variable avant sa déclaration car on ne peut pas modifier une variable déclarée avec const.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
x = 5; // On assigne 5 à x

elem = document.getElementById("demo");
elem.innerHTML = x; // On affiche la variable

let x;
</script>
</body>
</html>
```

Let, const (& var)

Conclusion

Utilisez **let** si vous allez modifier votre variable dans le bloc.

Utilisez **const** si n'allez pas modifier votre variable dans le bloc.

N'utilisez pas var à cause des effets de bord qu'on vient d'évoquer.

Les fonctions

Javascript permet de créer des fonctions comme tous les langages de programmation, la syntaxe est basique.

Vous pouvez bien entendu rajouter des variables scopés (sauf var)

```
function getDate() {
  return new Date();
}

console.log(getDate());
```

```
function getDate() {
  const date = new Date()
  return date;
}
```

Les events

De l'interactivité avec vos pages web

Les events sont les différentes interactions, actions ou états d'une page HTML.

Il en existe plusieurs, parmis les plus connus :

- onclick
- onchange
- onmouseover
- onmouseout
- onkeydown
- onload

Pour déclencher une fonction lors d'un événement sur un élément du DOM :

```
<button onclick="console.log('click !')">click me</button>
```

Les events

Usage de this avec les éléments du DOM

Javascript possède un mot-clé **this** qui est un peu différent des autres langages de programmation.

Avec **this**, on peut passer la référence de l'élément à la méthode exécutée lors d'un événement.

Dans l'exemple ci-contre, on passe la référence de l'élément cliqué en paramètre de la fonction **changeColor** ce qui permet d'interagir directement avec l'élément du DOM.

```
<!DOCTYPE html>
<html>
<body>
<button onclick="changeColor(this)">click me</button>
<script>
  function changeColor(el) {
    el.style.color = "red";
  }
</script>
</body>
</html>
```

Les events

Usage de this avec les éléments du DOM

Il est aussi possible de récupérer les informations d'un événement en passant en paramètre "event" dans la méthode au niveau de la déclaration de l'event.

```
<!DOCTYPE html>
<html>
<body onkeydown="getKey(event)">
<p> you pressed the key <span id="value"></span></p>
<script>
  function getKey(event) {
    document.getElementById('value').innerHTML = event.key;
  }
</script>
</body>
</html>
```

TP 2

Events

1. Créer une page HTML avec une image. Lorsque vous cliquez sur cette image elle pivote dans le sens des aiguilles d'une montre de 90°.
2. Créer une page HTML avec une liste contenant des noms de ville. Au clic d'un élément de la liste, remonter l' élément en haut de la liste.
3. Créer une page HTML avec 2 images. Lorsque vous cliquez sur une des deux images, la largeur de l'image cliquée augmente de 20% de sa largeur actuelle et la largeur de l'autre image diminue de 20% de sa largeur actuelle. Attention : la largeur ne peut pas être négative.
4. Créer une page HTML contenant une div de 50 px sur 50px avec un fond rose. Avec un event, faite en sorte de pouvoir déplacer la div avec les touches fléchées de votre clavier.
5. Créer une page HTML avec plusieurs images, au survol d'une image faites un zoom x3 sur cette image.
6. Récupérer une page HTML sur internet, au survol de n'importe quel élément, changer la couleur du texte en rouge
7. **Bonus :** Récupérer une page HTML sur internet, au clic sur un élément de la page, supprimer l'élément du DOM
8. **Bonus 2 :** Récupérer une page HTML sur internet. Au clic dans le body de la page, mélanger tous les éléments du DOM (à l'intérieur de body)

Itérations & Conditions

Conditions

A l'instar d'autres langages, javascript donne la possibilité d'écrire des conditions logiques.

```
if (age < 18) {
   return 'Not allowed to drive a car';
}


if (age > 18) {
   return 'Adult';
}
else if (age > 10) {
  return 'Teenage';
}
```

```
switch(colorHex) {
case "red":
 return "#af4154";
case "blue":
 return "#0f0ade";
default:
    console.log('color not found');
    break;
}
```

Itérations & Conditions

Itérations

On peut utiliser **for** pour faire des itérations logiques en Javascript. On utilisera bien sûr **let** et pas **var** pour déclarer notre itérateur **i.** Comme vu dans les slides précédentes, la portée de **var** est fonctionnelle et donc si on déclare i en **var** on pourra y accéder en dehors de notre itération.

```
let str = '';

for (let i = 0; i < 9; i++) {
  str = str + i;
}

console.log(str);
// valeur = "012345678"
```

Itérations & Conditions

Itérations

Vous pouvez aussi utiliser **while** et **do while** pour faire des itérations

```
while (true) {
  // Instruction
}

do {
  // Instruction
} while (i < 5)
```

TP 3

Itérations & Conditions

1. Créer une page HTML avec 3 div carrés ayant un fond noir. Au clic sur une div, elle prend la couleur rouge. Si vous recliquez dessus, elle redevient noir.
2. Créer une page HTML vide. Grâce à une boucle for, généré aléatoirement entre 5 et 15 div de 50px de haut et une largeur de 100% avec une couleur aléatoire.
3. Copier l'exercice précédent. Au clic d'une des div, changer la couleur de fond en rouge. Lorsque une div rouge est de nouveau cliquée, elle récupère sa couleur d'origine
4. Créer une page HTML, coder un script qui permet d'enregistrer les touches tapées par l'utilisateur sous forme de liste. Chaque fois qu'une touche est pressée sur le clavier, le nom de la touche sera ajouter une liste ul. La liste ne doit pas faire plus de 20 entrées, si jamais la liste est plus longue vous pouvez enlever les éléments les plus anciens.
5. Créer une page HTML qui affiche la taille de la fenêtre dynamiquement. Lorsque vous redimensionner la page, la largeur et la hauteur en pixel sont affichés.

Manipuler les strings

Introduction

Javascript permet une manipulation facile des Strings avec un ensemble de méthode. On peut déclarer les strings avec des doubles quotes ou des simple quotes, selon la configuration de vos IDE ou de vos linters, l'un ou l'autre sera imposé. Il est recommandé d'utiliser un seul type de quote par projet.

```
let text = "Toto";

let name1 = "Tata";  // Doubles quotes
let name2 = 'Titi';  // Simples quotes

let length = text.length; // Récupère la longueur
```

Manipuler les strings

Méthodes : Slice

Pour extraire une partie d'une string on utilise **slice(start, end)**

```
let str = "Apple, Banana, Kiwi";
let part = str.slice(7, 13);

// Output : Banana


// Si un des index est négatif, la position est comptée à partir de la fin
let str = "Apple, Banana, Kiwi";
let part = str.slice(-12, -6);

// Output : Banana


// Si vous ne précisez pas de 2ème argument, le reste de la String est slicée
let part = str.slice(7);

// Output : Banana, Kiwi
```

Manipuler les strings

Méthodes : substring & substr

substring fonctionne comme slice mais ne gère pas les index négatifs.

substr fonctionne comme slice, mais le 2ème paramètre est la longueur de la String qu'on veut extraire

```
let str = "Apple, Banana, Kiwi";
let part = str.substring(7, 13);

// Banana

let str = "Apple, Banana, Kiwi";
let part = str.substr(7, 6);

// Banana
```

Manipuler les strings

Méthodes : replace

Vous pouvez remplacer une partie de la chaîne de caractère avec la méthode replace

```
let text = "Hello world !";
let newText = text.replace("world", "everyone");

// Hello everyone !
```

Manipuler les strings

Méthodes : liste complète

String possède pleins de méthode, la liste complète des méthodes disponibles est sur MDN

Documentation des méthodes de String

TP 4

Manipulation des Strings

1. Créer une fonction **revert(str)** qui inverse une chaîne de caractère et la retourne
2. Ecrire une fonction **ucFirst(word)** qui transforme la première lettre d'une chaine de caractère en majuscule
3. Ecrire une fonction **capitalize(str)** qui transforme chaque première lettre de chaque mot en majuscule
4. Ecrire une fonction **pascalCase(str)** qui transforme chaque première lettre de chaque mot en majuscule et supprime les espaces
5. Créer une fonction **palindrome(str)** qui vérifie si un mot est un palindrome et retourne un booléen
6. Écrire une fonction **findLongestWord(str)** qui permet de trouver le mot le long plus d'une chaîne de caractère

```
revert("Hello I'm Robert !") // treboR m'I olleH

ucFirst("hello") // Hello

capitalize("hello i'm robert") // Hello I'm Robert

pascalCase("sentence in pascalCase") // SentenceInPascalCase

palindrome('kayak') //true

findLongestWord("Le chemin le plus cours n'est pas toujours le meilleur"); // toujours
```

Manipuler des tableaux et des objets

Les tableaux et les objets

Introduction

Les objets sont une structure de données mutable en javascript qui est utilisée pour représenter une « chose ». Cela peut être n'importe quoi comme des voitures, des plantes, une personne, une communauté, etc.

Les tableaux sont des objets uniquement en javascript. La principale différence est qu'ils stockent les données dans une collection ordonnée dans laquelle les données sont accessibles à l'aide d'un index numérique. Ils sont également modifiables et les données peuvent être modifiées à n'importe quel index.

Les méthodes des tableaux

Créer un tableau

La création d'un tableau est identique à la majorité des autres langages de programmation.

```
const fruits = ['Apple', 'Banana'];

console.log(fruits.length);
// 2
```

On peut accéder à un élément du tableau via son index

```
let first = fruits[0];
// Apple

let last = fruits[fruits.length - 1];
// Banana
```

Les méthodes des tableaux

Parcours de tableau

Pour parcourir un tableau, on évitera dans la majorité des cas d'utiliser une boucle **for.**

Javascript permet d'utiliser des méthodes appropriées pour le parcours de tableau.

```
let fruits = ['Apple', 'Banana'];

fruits.forEach(function(item, index) {
  console.log(item, index);
});
// Apple 0
// Banana 1
```

Les méthodes des tableaux

Parcours de tableau

Pour parcourir un tableau, on évitera dans la majorité des cas d'utiliser une boucle **for.**

Javascript permet d'utiliser des méthodes appropriées pour le parcours de tableau : la méthode **forEach**

```
let fruits = ['Apple', 'Banana'];

fruits.forEach(function(item, index) {
  console.log(item, index);
});
// Apple 0
// Banana 1
```

Les méthodes des tableaux

Manipulation d'éléments

```
fruits.push('Orange'); // Ajoute à la fin
// ["Apple", "Banana", "Orange"]


fruits.pop(); // Supprime le dernier élément
// ["Apple", "Banana"];

fruits.shift(); // Supprime le premier élement
// ["Banana"];

fruits.unshift('Strawberry') // Ajoute un élément au début
// ["Strawberry", "Banana"];

fruits.indexOf('Strawberry'); // Trouve l'index d'un élément
// 0
```

Les méthodes des tableaux

Array.prototype.map()

```
const animals = ['Horse', 'Dog', 'Cat'];

const redAnimals = animals.map(function(element) {
  return 'Red ' + element;
});

console.log(redAnimals);
// expected output: Array ['Red Horse', 'Red Dog', 'Red Cat'];
```

La méthode **map()** crée un nouveau tableau rempli avec les résultats de l'appel d'une fonction fournie sur chaque élément du tableau appelant.

Les méthodes des tableaux

Array.prototype.filter()

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]
```

La méthode **filter()** crée un nouveau tableau avec tous les éléments qui réussissent le test implémenté par la fonction fournie.

Les méthodes des tableaux

Array.prototype.find()

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found);
// expected output: 12
```

La méthode **find()** renvoie le premier élément du tableau fourni qui satisfait la fonction de test fournie. Si aucune valeur ne satisfait la fonction de test, *undefined* est renvoyé.

Les méthodes des tableaux

Array.prototype.join()

```
const elements = ['Fire', 'Air', 'Water'];

console.log(elements.join());
// expected output: "Fire,Air,Water"

console.log(elements.join(''));
// expected output: "FireAirWater"

console.log(elements.join('-'));
// expected output: "Fire-Air-Water"
```

La méthode **join()** crée et renvoie une nouvelle chaîne en concaténant tous les éléments d'un tableau (ou d'un objet de type tableau), séparés par des virgules ou une chaîne de séparation spécifiée. Si le tableau ne contient qu'un seul élément, cet élément sera renvoyé sans utiliser de séparateur.

Fonctions fléchées

Les fonctions fléchées ont été introduites dans ES6.

Les fonctions fléchées nous permettent d'écrire une syntaxe de fonction plus courte

```
// Avant
hello = function() {
  return "Hello World!";
}

// Maintenant
hello = () => {
  return "Hello World!";
}

// Si la méthode retourne une valeur
// On peut directement écrire comme ça
hello = () => "Hello World!";
```

Toutes les fonctions doivent tenir en 1 ligne

TP 5

Manipulation des tableaux

1. Écrivez une fonction **getHashTags(str)** qui récupère les trois mots les plus longs d'une chaîne de caractère et les transforme en hashtags. Si plusieurs mots sont de la même longueur, récupérez le mot qui apparaît en premier.

```
console.log(getHashTags("How the Avocado Became the Fruit of the Global Trade")); // ["#avocado",
"#became", "#global"]
console.log(getHashTags("Why You Will Probably Pay More for Your Christmas Tree This Year")); //
["#christmas", "#probably", "#will"]
console.log(getHashTags("Hey Parents, Surprise, Fruit Juice Is Not Fruit")); // ["#surprise",
"#parents", "#fruit"]
console.log(getHashTags("Visualizing Science")); // ["#visualizing","#science"]
```

2. Créer une fonction qui supprime les valeurs dupliquées dans un tableau (Number, String)

```
const tab = [0, 2, 4, 6, 8, 8];
removeDuplicate(tab) // [0,2,4,6,8]
```

Toutes les fonctions doivent tenir en 1 ligne

TP 5

Manipulation des tableaux

3. Créer une function **intersection(arr,arr2),** fonction qui calcule l'intersection entre 2 tableaux (Number, String)

```
const first = [0, 2, 4, 6, 8, 8];
const second = [1, 2, 3, 4, 5, 6];
intersection(first,second) // [2,4,6]
```

4. Créer une fonction **arrayDiff(first,second)** qui calcule la différence entre 2 tableaux (Number, String)

```
const first = [0, 2, 4, 6, 8, 8];
const second = [1, 2, 3, 4, 5, 6];
arrayDiff(first,second) // [0,1,3,5,8
```

Toutes les fonctions doivent tenir en 1 ligne

TP 5

Manipulation des tableaux

5. Créez une fonction combinations qui prend un nombre variable d'arguments, chaque argument représentant le nombre d'articles dans un groupe, et qui renvoie le nombre de permutations (combinaisons) d'articles que vous pourriez obtenir en prenant un article de chaque groupe.

```
combinations(2, 3) // 6
combinations(3, 7, 4) // 84
combinations(2, 3, 4, 5) // 120
```

Les méthodes des objets

Création d'un objet

```
let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Les méthodes des objets

Itération

```
for (const key in person) {
    console.log(`${key}: ${person[key]}`);
}
```

Les méthodes des objets

Méthodes à connaître

```
Object.keys(person); // Permet de récuperer les clefs
Object.values(person); // Permet de récuprer les valeurs
Object.entries(person); // Retourne l'objet sous forme de tableau itérable
```

TP 6

Manipulation d’objets

Exercice 1 :

En Italie, chaque personne possède un code d'identification unique délivré par l'administration fiscale nationale après l'enregistrement de la naissance : le code fiscal (Codice Fiscale). Consultez la page https://en.wikipedia.org/wiki/Italian_fiscal_code pour plus d’informations

Vous aurez à disposition un objet contenant les données personnelles d'une personne (nom, prénom, sexe et date de naissance), Créer une fonction fiscalCode qui renvoie les 11 caractères du code sous forme de chaîne de caractères en suivant ces étapes :

Générez 3 lettres majuscules à partir du nom de famille, s’il y a:

Au moins 3 consonnes,prendre les 3 premières consonnes. (Newman -> NWM).

Moins de 3 consonnes, les voyelles remplaceront les caractères manquants dans le même ordre qu'ils apparaissent (Fox -> FXO | Hope -> HPO). Moins de trois lettres, "X" prendra le troisième emplacement après la consonne et la voyelle (Yu -> YUX).

Générez 3 lettres majuscules à partir du nom, s’il y a :

Exactement 3 consonnes ,les consonnes sont utilisées dans l'ordre où elles apparaissent (Matt -> MTT). Plus de 3 consonnes , la première, troisième et quatrième consonnes sont utilisées (Samantha -> SNT | Thomas -> TMS).

Moins de 3 consonnes, les voyelles remplaceront les caractères manquants dans le même ordre qu'ils apparaissent (Bob -> BBO | Paula -> PLA).

TP 6

Manipulation d'objets

Moins de trois lettres, "X" prendra le troisième emplacement après la consonne et la voyelle (Al -> LAX).
Générez 2 chiffres, 1 lettre et 2 nombres à partir de la date de naissance et du sexe :
Prenez les deux derniers chiffres de l'année de naissance (1985 -> 85).
Générez une lettre correspondant au mois de naissance (janvier -> A | décembre -> T) en utilisant la table de conversion incluse dans le code.
Pour les hommes, prenez le jour de naissance en ajoutant un zéro au début s'il est inférieur à 10 (tout 9e jour -> 09 | tout 20e jour -> 20).
Pour les femmes, prenez le jour de naissance et ajoutez 40 (tout 9e jour -> 49 | tout 20e jour -> 60).

Les lettres de code doivent être en majuscules.
La date de naissance est indiquée dans le format J/M/AAAA.
Y n'est pas une voyelle.

La table de conversion des mois est la suivante :
Janvier = A
Février = B
Mars = C
Avril = D
Mai = E
Juin = H
Juillet = L
Août = M
Septembre = P
Octobre = R
Novembre = S
Décembre = T

TP 6

Manipulation d'objets

Exemple d'output attendu

```
fiscalCode({
  name: "Matt",
  surname: "Edabit",
  gender: "M",
  dob: "1/1/1900"
}) //"DBTMTT00A01"

fiscalCode({
  name: "Helen",
  surname: "Yu",
  gender: "F",
  dob: "1/12/1950"
}) //"YUXHLN50T41"

fiscalCode({
  name: "Mickey",
  surname: "Mouse",
  gender: "M",
  dob: "16/1/1928"
}) // "MSOMKY28A16"
```

TP 6

Manipulation d'objets

Exercice 2 : Créer une méthode merge qui prend en paramètres des objets et les fusionnent.

```
const object = {
  a: [{ x: 2 }, { y: 4 }],
  b: 1
};
const other = {
  a: { z: 3 },
  b: [2, 3],
  c: 'foo'
};
console.log(merge(object, other)); // {"a":[{"x":2},{"y":4},{"z":3}],"b":[1,2,3],"c":"foo"}
```

Javascript avancé

Les promesses

Une promesse est un objet (Promise) qui représente la complétion ou l'échec d'une opération asynchrone. La plupart du temps, on « consomme » des promesses.

En résumé, une promesse est un objet qui est renvoyé et auquel on attache des callbacks plutôt que de passer des callbacks à une fonction. Ainsi, au lieu d'avoir une fonction qui prend deux callbacks en arguments

Les promesses

```
function faireQqcALAncienne(successCallback, failureCallback){
  console.log("C'est fait");

  if (Math.random() > .5) {
    successCallback("Réussite");
  } else {
    failureCallback("Échec");
  }
}

function successCallback(résultat) {
  console.log("L'opération a réussi avec le message : " + résultat);
}


function failureCallback(erreur) {
  console.error("L'opération a échoué avec le message : " + erreur);
}

faireQqcALAncienne(successCallback, failureCallback);
```

Sans promesse

Les promesses

```
function faireQqc() {
  return new Promise((successCallback, failureCallback) => {
    console.log("C'est fait");

    if (Math.random() > .5) {
      successCallback("Réussite");
    } else {
      failureCallback("Échec");
    }
  })
}

function successCallback(résultat) {
  console.log("L'opération a réussi avec le message : " + résultat);
}

function failureCallback(erreur) {
  console.error("L'opération a échoué avec le message : " + erreur);
}

const promise = faireQqc();
promise.then(successCallback, failureCallback);
```

Avec promesse

Les promesses

Try catch, await / async

Pour gérer les erreurs en JS, vous pouvez utilisez un try catch. Les try catch fonctionnent très bien avec les promesses et permet de rendre son code plus lisible.

Ici on exécute une promesse basique, et si jamais elle échoue on affiche l'erreur.

```
function execute() {
  try {
    myPromise(1).then(res => console.log(res));
  } catch(e) {
    console.log(e);
  }
};

const myPromise = (value) => {
  return new Promise((resolve, failure) => {
    if(value === 1) {
      resolve('Hey ! This is my promise');
    } else {
      failure('Oops');
    }
});
}

execute();
```

Les promesses

Try catch, await / async

Utiliser des then et des catch après une promesse est compliqué et pas forcément pratique quand on veut récupérer la valeur de cette promesse. Pour cela, on peut utiliser la syntaxe **async / await.**

Await permet d'attendre que la promesse s'exécute avant de continuer à exécuter la suite du code.

Pour pouvoir utiliser **await**, il faut que la fonction soit marqué comme "async" (sinon vous aurez une erreur).

```
async function execute() {
  try {
    const firstPromise = myPromise(1);
    console.log(firstPromise);
    const secondPromise = myPromise(2);
    console.log(secondPromise);
  } catch(e) {
    console.log(e);
  }
};

const myPromise = (value) => {
  return new Promise((resolve, failure) => {
    if(value === 1) {
      resolve('Hey ! This is my promise');
    } else {
      failure('Oops');
    }
});
}

execute();
```

TP 7

1. Créez une fonction qui prend en paramètre une chaîne de caractère, si la chaîne de caractère fait plus de 20 caractères, la promesse échoue, sinon la fonction renvoie **true**
2. Créez une fonction qui prend en paramètre deux int, si la première variable est supérieure à la seconde, la promesse renvoie la différence entre les deux variables, sinon elle échoue.
3. Créez une fonction qui prend en paramètre une date de naissance d'une personne au format DD/MM/YYYY. Si la personne est mineure, la promesse échoue, sinon elle renvoie **true**
4. Exécutez toutes les fonctions créées ci-dessus avec des then & catch
5. Exécutez toutes les fonctions créées ci-dessus dans un try catch avec des awaits
6. Avec fetch et https://swapi.dev/, recupérer la liste de tous les caractères, vaisseaux et planète et affichez les dans une liste. Vous devez utilisez des promesses, async / await & des try catch.

IIFE

IIFE (Immediately Invoked Function Expression) (Expression de fonction invoquée immédiatement) est une fonction JavaScript qui est exécutée dès qu'elle est définie.

```
(function () {
    console.log('Hello world !');
})();

(() => {
  console.log('Hello mars !');
})();
```

Template strings

Les templates strings permettent d'intégrer des expressions directement dans une chaîne de caractère. Avec eux, on peut utiliser des chaînes de caractères multi-lignes et des fonctionnalités d'interpolation.

```
const firstName = "John";
const lastName = "Doe";

const text = `Welcome ${firstName}, ${lastName}!`;
```

Déstructuration

La déstructuration est une expression JavaScript qui permet de décompresser des valeurs de tableaux, ou des propriétés d'objets, dans des variables distinctes

```
const user = {
  id: 1,
  firstname: 'Foo',
  lastname: 'Bar',
}


// Avant ES6

console.log(user.id);
console.log(user.firstname);
console.log(user.lastname);


// Syntaxe ES6

const { id, firstname, lastname } = user;

console.log(id);
console.log(firstname);
console.log(lastname);
```

Spread Operator

```
const user = {
  id: 1,
  firstname: 'Foo',
  lastname: 'Bar',
}


// Avant ES6

console.log(user.id);
console.log(user.firstname);
console.log(user.lastname);


// Syntaxe ES6

const { id, firstname, lastname } = user;

console.log(id);
console.log(firstname);
console.log(lastname);
```

Le spread operator permet de développer les strings, les tableaux ou les objets.

Rest operator

Le rest operator permettent de passer un nombre infini d'arguments à une fonction.

```
function display(arg1, arg2, ...args){
   console.log(arg1);
   console.log(arg2);
   console.log(args);
}

display(2, 3, 4, 5, 6);
```