Cours JavaScript

Part 3

PLAN

QCM (30min)

Les Origines (Events Driven)

Correction des TP

Paramètres par défaut

Mini projet

Conclusion

QCM

JS

Envoyé dans votre boite mail.

Les Origines

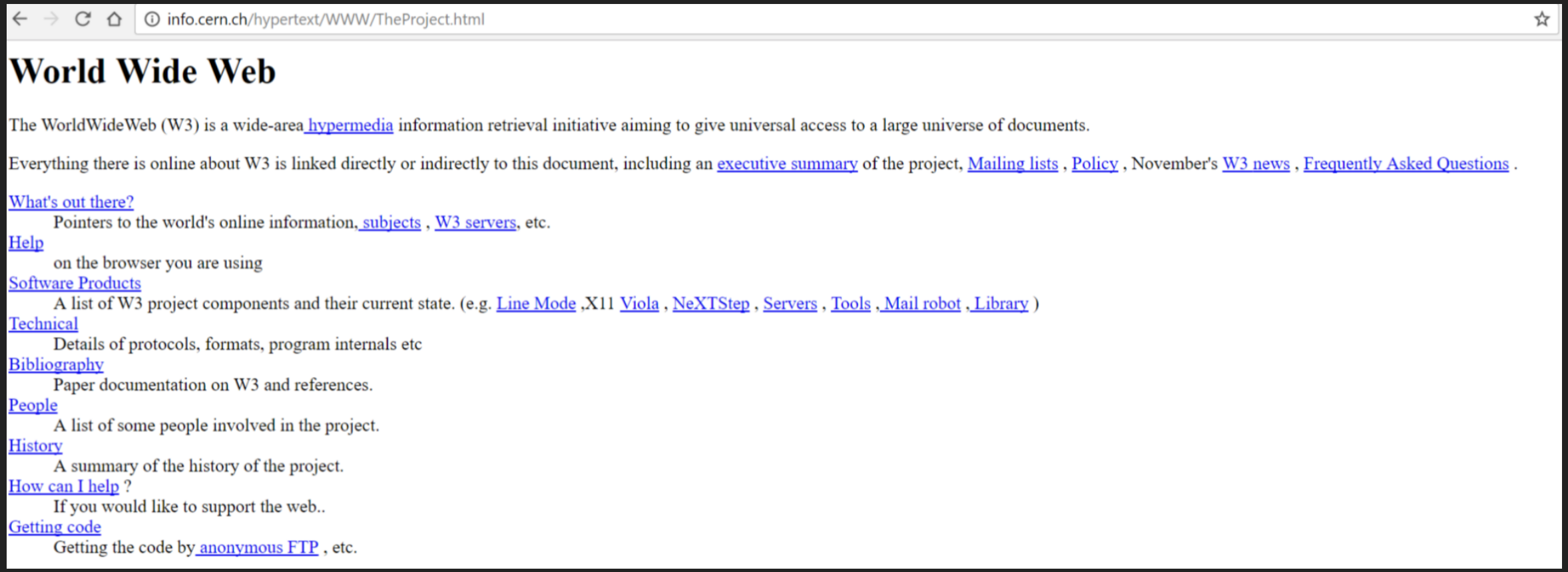
https://www.perdu.com

Les Origines

Tim Berners-Lee 1991

W3C

Pur HTML



Les Origines

JavaScript a été créé en 1995 par Brendan Eich (en 10 J seulement !) 😱

Dynamic HTML

"Vieux" PC => Mono-Cœur

Les applications dans les autres langages sont Multithreads pour simuler le "Multi-tache"

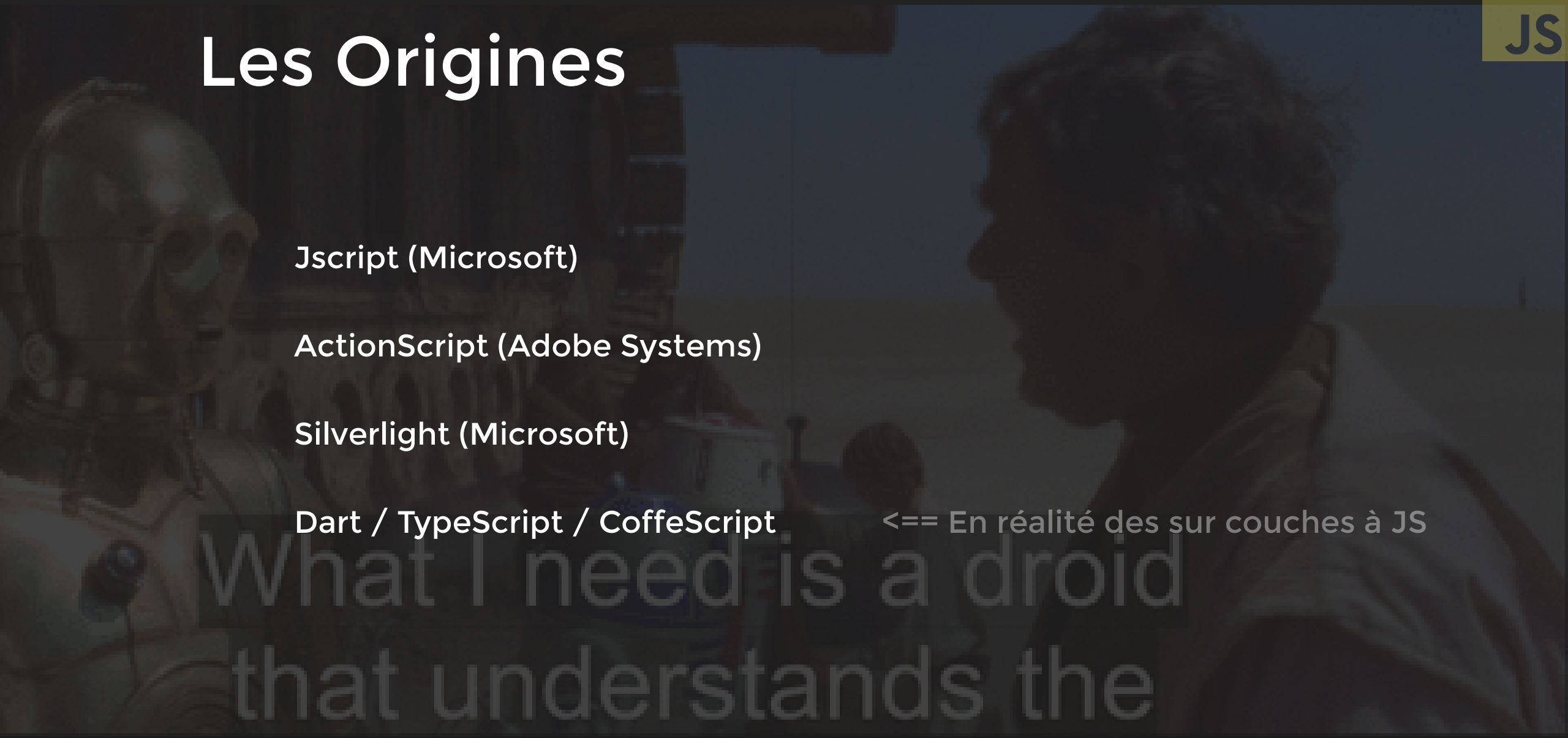
Les Origines

Jscript (Microsoft)

ActionScript (Adobe Systems)

Silverlight (Microsoft)

Dart / TypeScript / CoffeScript <== En réalité des sur couches à JS



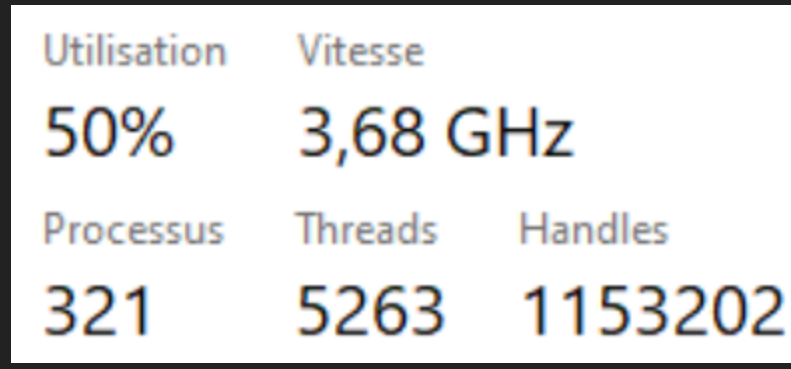
Callbacks

Les autres languages sont MultiThreads

Javascript est MonoTread, mais il est "Event Driven"

Les Callbacks sont un moyen de JS de s'assurer que notre code sera exécuter au bon moment !

Un Callback est une fonction qui sera appelé plus tard



Syntaxe

```
arr.forEach(callback);
arr.forEach(callback, thisArg);
```

Paramètres

`callback`

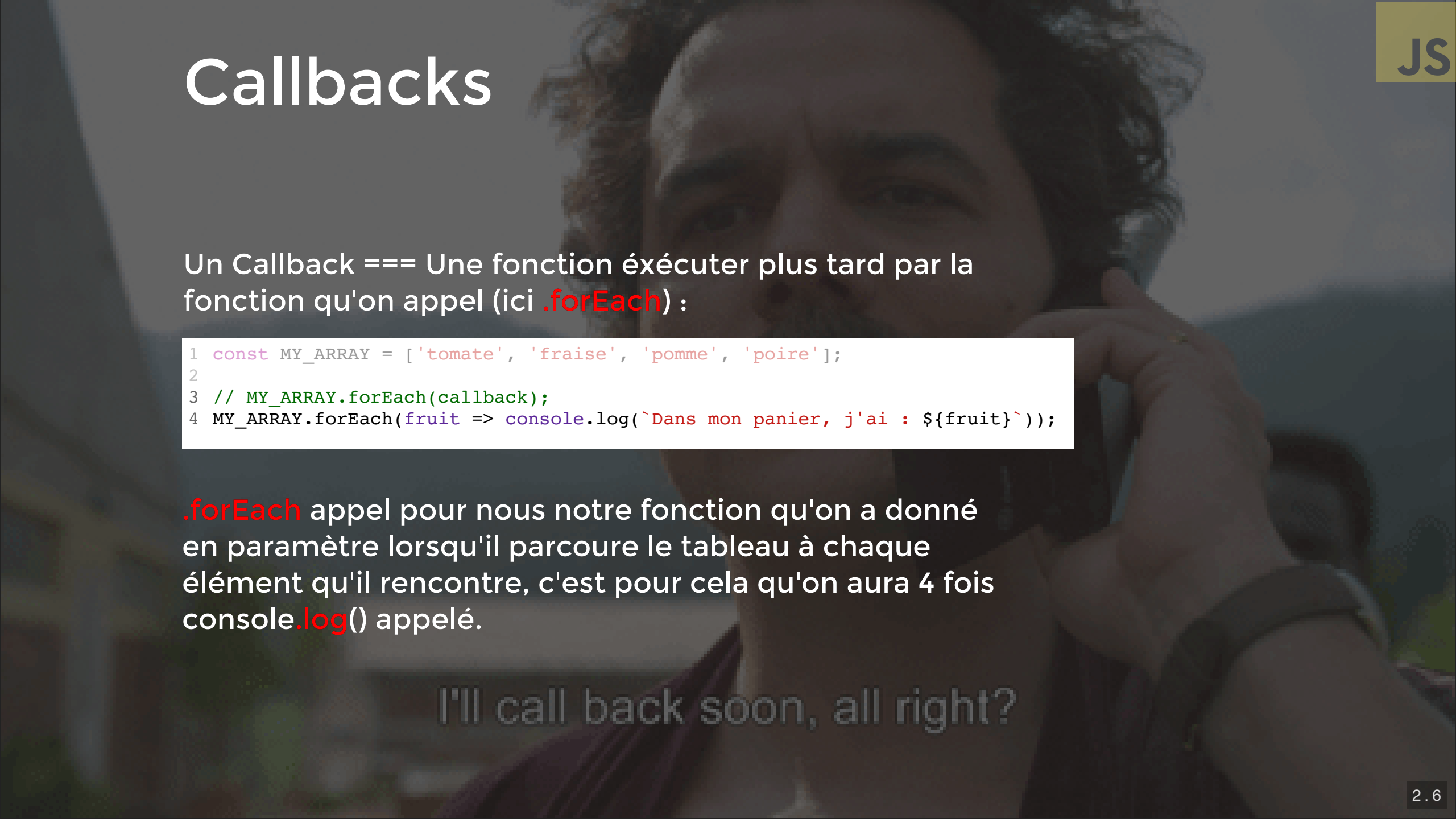
La fonction à utiliser pour chaque élément du tableau. Elle prend en compte trois arguments

Callbacks

Un Callback === Une fonction éxécuter plus tard par la fonction qu'on appel (ici .forEach) :

```
1 const MY_ARRAY = ['tomate', 'fraise', 'pomme', 'poire'];
2
3 // MY_ARRAY.forEach(callback);
4 MY_ARRAY.forEach(fruit => console.log(`Dans mon panier, j'ai : ${fruit}`));
```

.forEach appel pour nous notre fonction qu'on a donné en paramètre lorsqu'il parcoure le tableau à chaque élément qu'il rencontre, c'est pour cela qu'on aura 4 fois console.log() appelé.



Callbacks

Ici on appel deux fonction à la suite, comme attendu, les résultats sont affiché dans l'ordre de l'appel :

```
1  function first() {
2    console.log(1);
3  }
4
5  function second() {
6    console.log(2);
7  }
8
9  first();
10 second();
11
12 // 1
13 // 2
```

Avec setTimeout(callback, timeout) on peux simuler un event qui sera déclenché après x milliseconde :

```
1  function first() {
2    setTimeout(() => {
3          console.log(1);
4    }, 10); // milliseconde
5  }
6
7  function second() {
8    console.log(2);
9  }
10
11 first();
12 second();
13
14 // 2
15 // 1
```

Callbacks

Parce qu'on ne peux pas espérer que lorsque deux fonction appelé l'une après l'autre qu'elle soit exécuté dans l'ordre que vous souhaitez /!\

Le seul moyen en Javascript d'avoir un ordonnancement voulu, c'est de passer par des Callbacks /!\

Ici callback est une référence à notre fonction second()

```
1  function first(callback) {
2    setTimeout(() => {
3          console.log(1);
4
5      // appel de la fonction
6      // passé en paramètre
7      callback();
8    }, 10); // millisecondes
9  }
10
11 first(function second() {
12   console.log(2);
13 });
14
15 // 1
16 // 2
```

Events & Callbacks

Création d'un élément span avec l'id "clickme"

On récupère en JS notre élément par son ID

On créer notre listener et on fournis notre callback

```
1  <html>
2    <head></head>
3    <body>
4      <span id="clickme">Cliquez-moi !</span>
5
6      <script>
7        const element = document.getElementById('clickme');
8
9        element.addEventListener('click', () => {
10         alert("Vous m'avez cliqué !");
11       });
12     </script>
13   </body>
14 </html>
```

Events & Callbacks

Il est parfaitement possible de créer des listeners pour le même "event"

```html
<html>
  <head></head>
  <body>
    <span id="clickme">Cliquez-moi !</span>

    <script>
      const element = document.getElementById('clickme');

      element.addEventListener('click', () => {
        alert("Vous m'avez cliqué !");
      });

      element.addEventListener('click', () => {
        alert("Et j'en ai la preuve !");
      });
    </script>
  </body>
</html>
```

Events & Callbacks

JS

L'objet Event nous donne un tas d'informations
(position du curseur, touche pressé, etc...)

```
1  <html>
2    <head></head>
3    <body>
4      <span id="clickme">Cliquez-moi !</span>
5
6      <script>
7        const element = document.getElementById('clickme');
8
9        element.addEventListener('click', e => alert(`${e.clientX}:${e.clientY}`))
10     </script>
11   </body>
12 </html>
```

Events & Callbacks

Liste des events couramment utilisé :

click	Cliquer (appuyer puis relâcher) sur l'élément
dblclick	Double-cliquer sur l'élément
mouseover	Faire entrer le curseur sur l'élément
mouseout	Faire sortir le curseur de l'élément
mousedown	Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
mouseup	Relâcher le bouton gauche de la souris sur l'élément
mousemove	Faire déplacer le curseur sur l'élément

Events & Callbacks

Liste des events couramment utilisé (suite) :

keydown	Appuyer (sans relâcher) sur une touche de clavier sur l'élément
keyup	Relâcher une touche de clavier sur l'élément
keypress	Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
focus	« Cibler » l'élément
blur	Annuler le « ciblage » de l'élément
change	Changer la valeur d'un élément spécifique aux formulaires
input	Taper un caractère dans un champ de texte (pas supporté partout)

Events & Callbacks

JS

Liste des events couramment utilisé (suite) :

select	Sélectionner le contenu d'un champ de texte (input,textarea, etc.)
submit	Envoyer le formulaire
reset	Réinitialiser le formulaire

Allez sur le MDN pour avoir une liste exhaustive

Events & Callbacks

JS

On crée nos éléments html

On les récupère en JS

On ajoute des listener avec des callback pour interagir avec l'utilisateur

.innerHTML permet d'écrire dans notre élément html

```
<html>
  <head></head>
  <body>
    <p id="result"></p>
    <div id="parent1">
      Parent N°1<br /> Mouseover sur l'enfant
      <div id="child1">Enfant N°1</div>
    </div>

    <div id="parent2">
      Parent N°2<br /> Mouseout sur l'enfant
      <div id="child2">Enfant N°2</div>
    </div>

    <script>
      const child1 = document.getElementById('child1');
      const child2 = document.getElementById('child2');
      const result = document.getElementById('result');

      child1.addEventListener('mouseover', e => {
        result.innerHTML = `Le curseur entre sur l'enfant n°1: ${e.relatedTarget.id}`;
      });

      child2.addEventListener('mouseout', e => {
        result.innerHTML = `Le curseur sort sur l'enfant n°2: ${e.relatedTarget.id}`;
      });
    </script>
  </body>
</html>
```

Correction des TP

Les TP ci-dessous sont sélectionne (uniquement sur les JS-Moyen-B2-*) pour la correction en live

Correction des TP

```
const input = [
        { key: 'mArChe', values: ['charme', 'courir', 'macher'] },
        { key: 'rage', values: [ 'amour', 'code', 'source'] },
        { key: 'rage', values: [ 'amour', 'haine', 'gare'] },
        { key: 'rage', values: ['amour', 'code', 'source', 'gare'] },
        { key: 'rame', values: [ 'bateau', 'amer', 'canards', 'mare'] },
        { key: 'tappe', values: [ 'patte'] },
        { key: 'argent', values: ['gendre', 'ganter', 'gare', 'garent', 'gerant'] },
        { key: 'marche', values: ['chArmE', 'CouRir', 'MacHer'] },
        { key: 'marche', values: ['marche', 'Marche', 'MaRchE'] },
];

function getAnagram (list) {
        // Une anagramme est un mot que l'on peut former en changeant
        // de place les lettres d'un autre mot.
        //
        // Pour le mot rage et le tableau ['amour','haine','gare']
        // Je dois retourner le tableau ['gare']

}
```

Correction des TP

```
1 function convertRawData (rawData) {
2         // Donnée d'arrivé: BOR675847583748sjt567654;Bordeaux Vendredi
3         // Je dois la transformer en BOR : Bordeaux Vendredi
4
5 }
```

Correction des TP

```
1 function convertRawData (rawData) {
2         let coupage1 = rawData.split(';');
3         let retour = coupage1[0][0] + coupage1[0][1] + coupage1[0][2] + ' : ' + coupage1[1];
4         return retour;
5 }
```

JS-B2-Moyen-2 : https://repl.it/teacher/assignments/4037135/model_solution

```
1 function convertRawData (rawData) {
2         return `${rawData.slice(0, 3)} : ${rawData.split(';')[1]}`;
3 }
```

Correction des TP

```
function armstrong (number) {
        // Implémenter le code necessaire permettant de réaliser toutes les conditions de l'éno
        let chiffre_compo = number.length;
        let nombre_decompo = number.split("");
        i=0;
        i2=0;
        const number_tab = [];

        while(i<chiffre_compo){
         let test = nombre_decompo[i];
         number_tab.push(test);
         i++;
        }

        while(i2<chiffre_compo){
          number_tab[i2]
        }

        return number_tab;


}
```

Paramètres par défaut

Avant ES6, la vérification si l'argument contient une valeur se faisait dans la fonction :

```
1 function multiplier(a, b) {
2   let b = (typeof b !== 'undefined') ? b : 1;
3
4   return a * b;
5 }
6
7 multiplier(5, 2); // 10
8 multiplier(5, 1); // 5
9 multiplier(5);    // 5
```

Avec ES6, il est possible d'ajouter une valeur par défaut à un argument très simplement :

```
1 function multiplier(a, b = 1) {
2   return a * b;
3 }
4
5 multiplier(5, 2); // 10
6 multiplier(5, 1); // 5
7 multiplier(5, undefined); // 5
8 multiplier(5); // 5
```

Paramètres par défaut

TP JS-B2-FACILE-1 (Solution proposé) :

```
1 function say (firstName) {
2         const response = firstName || 'toi';
3 
4         return `Un pour ${response}, un pour moi.`;
5 }
6 
7 console.log(say("Stella"));
8 console.log(say("Jean"));
9 console.log(say());
```

TP JS-B2-FACILE-1 (Avec paramètres par défaut) :

```
1 function say (firstName = 'toi') {
2         return `Un pour ${firstName}, un pour moi.`;
3 }
4 
5 console.log(say("Stella"));
6 console.log(say("Jean"));
7 console.log(say());
```

Mini projet #1

JS

Jeu du pendu (logique en js, affichage en html avec des caractères) :

Le Pendu est un jeu consistant à trouver un mot en devinant quelles sont les lettres qui le composent. Le jeu se joue traditionnellement à deux, avec un papier et un crayon, selon un déroulement bien particulier. Pour plus d'info sur le déroulement: https://fr.wikipedia.org/wiki/Le_Pendu_(jeu)

Ici le deuxième joueur sera votre ordinateur. Vous devez programmer les deux mode de jeu (vous cherchez le mot a deviner, ce dernier sera dans une liste pré-définit en dur dans le code / l'ordinateur dois chercher le mot à deviner sans chercher dans la liste de mots pré-définit en dur). Le mode de jeu doit être choisi avant chaque partie.

Critères de notation :

- Respect des consignes	===> 10pts	- Utilisation de "var"	===> -1pts
- Le projet fonctionne comme attendu	===> 10pts	- Ne pas déclarer les variables	===> -3pts / ligne
- Bonnus	===> 1pts / bonus	- Le projet contient un bug (ou cas non géré)	===> -2pts / bug

Bonus : Ajout d'un compteur de point et il sera possible de le reset avec un bouton. **Designer** (CSS) le projet. **Affichez** le pendu (en caractères) dynamiquement suivant l'évolution de la partie. **Code** soigné, concis et efficace. **Mode** 2 joueur (humain).

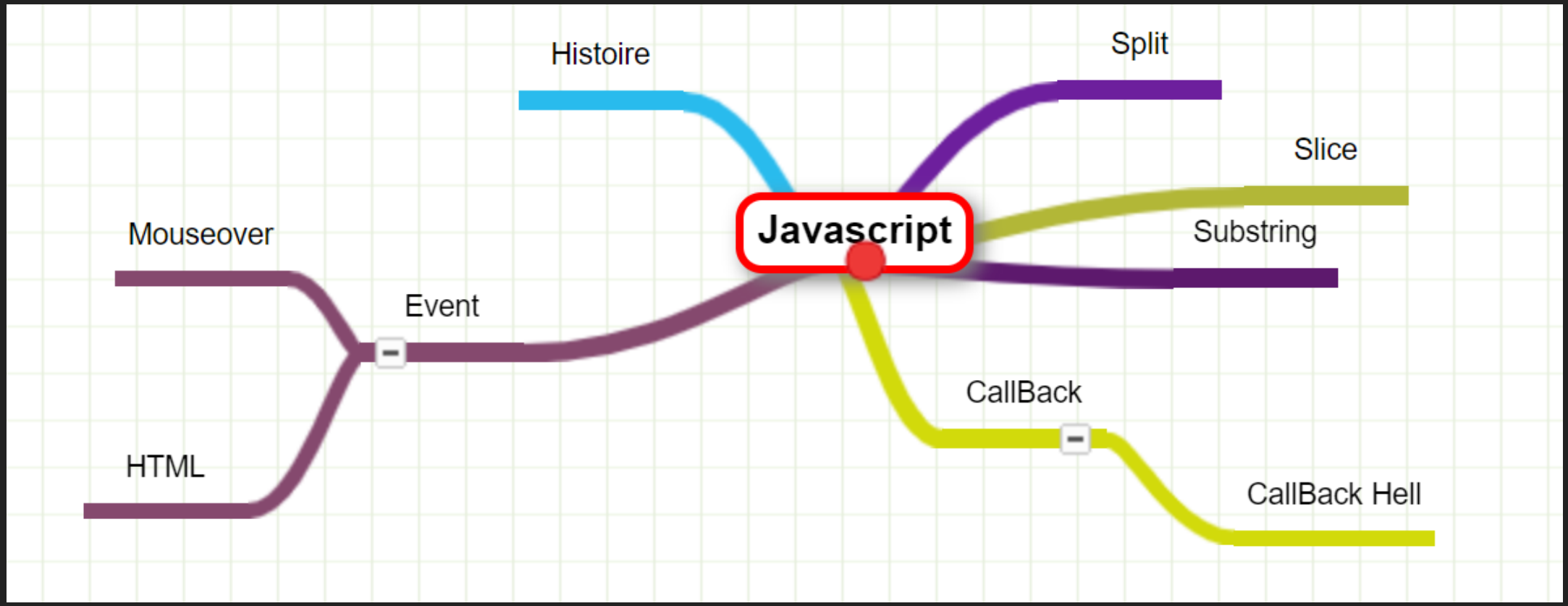
A faire pour le prochain cours

Conclusion

Nous avons vu :

- Les origines de JS
- La notion d'Event Driven
- Les paramètres pas défauts

MindMap time !!!


Histoire
Split
Slice
Javascript
Substring
Mouseover
Event
CallBack
HTML
CallBack Hell