



JavaScript

SOMMAIRE



I. Introduction

1. Javascript, c'est quoi ?

II. Mise en place de l'environnement

1. Installer GIT et Bash
2. Installer et paramétrer Babel
3. Incorporer du JS dans un fichier HTML
4. Installer et configurer Webpack
5. La console Javascript

III. Les bases du Javascript

1. Les commentaires et les variables
2. Les Types de variables
3. Les opérateurs arithmétiques
4. Les conversions implicites
5. Les opérateurs de comparaison
6. Référence et valeur

IV. Les Conditions et Boucles

1. If, else, else if
2. Opérateur ternaire
3. Switch
4. Boucle for
5. Boucle while et do while

V. Les nombres

1. Déclarer un nombre
2. Quelques méthodes de Number
3. L'objet Math

VI. Les chaînes de caractère

1. Déclarer une chaîne de caractères
2. Propriétés et quelques méthodes de String
3. Les Regex

SOMMAIRE



VII. Les objets

1. Un objet c'est quoi ?
2. Les propriétés des objets
3. La décomposition d'objets
4. Tester l'existence d'une propriété
5. Supprimer une propriété
6. Itérer sur un objet
7. Le format JSON

VIII. Les fonctions

1. Déclarer une fonction
2. Paramètre
3. Valeur de retour d'une fonction
4. Contexte global et contexte d'exécution
5. Le mot clé this et le mode strict
6. Définir ou lier this
7. Les fonctions fléchées
8. Fonctions de rappel (callback)

IX. Les tableaux

1. Introduction aux tableaux
2. Accéder aux éléments d'un tableau
3. Imbrication de tableaux
4. Ajouter des éléments à un tableau
5. Supprimer des éléments à un tableau
6. Trouver des éléments dans un tableau
7. Copier un tableau
8. Fusionner des tableaux
9. Trier un tableau
10. Itérer sur un tableau
11. Introduction à la programmation fonctionnelle

X. Les modules

1. Exporter les modules
2. Importer les modules
3. Réexporter et les imports dynamiques

SOMMAIRE



XI. Le DOM

1. Introduction
2. Sélectionner des éléments du DOM
3. Modifier les éléments du DOM
4. Attributs et propriétés
5. Utilisation de CSS avec Webpack
6. Modification du style et des classes
7. Créer des nœuds et les positionner
8. Supprimer et remplacer des noeuds

XII. Les évènements

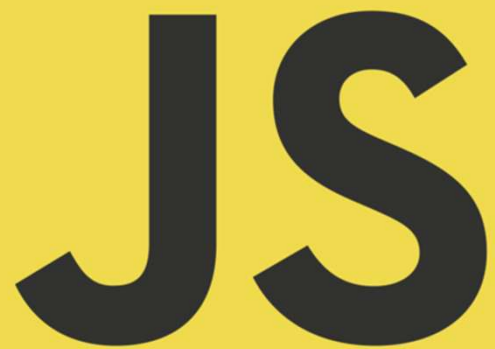
1. Introduction
2. Utilisation des propriétés du DOM on*
3. La méthode addEventListener
4. Supprimer un gestionnaire d'évènement et déclencher un évènement
5. Empêcher le comportement par défaut et l'objet Event

XIII. Gérer l'asynchrone sur un navigateur

1. Introduction à l'asynchrone et timer
2. Les promesses
3. Les méthodes des promesses
4. Utilisation de polyfills avec Webpack
5. Les fonctions asynchrones avec async / await
6. Event loop

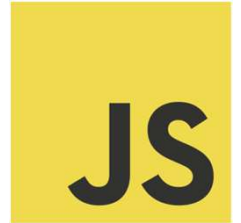
XIV. Le réseau

1. Introduction aux requêtes
2. Première requête HTTP avec fetch
3. Effectuer une requête POST
4. Les CORS
5. Annuler des requêtes en cours
6. Les objets FormData
7. La Web API URL
8. XMLHttpRequest



I. Introduction

1- Javascript, c'est quoi



C'est un langage de programmation qui permet de générer et de modifier dynamiquement du contenu HTML et CSS.

Il permet également d'interagir avec les API (Application Programming Interface) des navigateurs.

Exemple: API vidéo, API audio, API canvas, API Geolocalisation...

Ce langage est un des seules langages utilisées par tous les navigateurs (avec le HTML et le CSS)

1- Javascript, c'est quoi

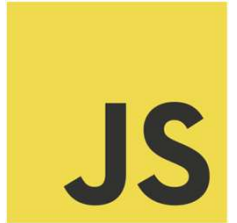


Initialement, il s'agissait d'un langage exclusivement côté client.
Il peut maintenant, à l'aide de Node.js, l'être également côté serveur.

En 2013, AirBNB est la première grande entreprise à l'utiliser côté client et serveur.

Dans ce cours, nous étudierons les bases de Javascript et son utilisation côté client.

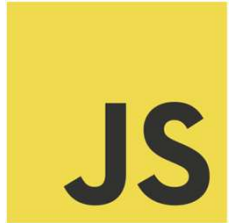
1- Javascript, c'est quoi



Les principales caractéristiques de Javascript

- Langage interprété => le moteur des navigateurs va l'interpréter et l'exécuter. Il n'a pas besoin d'être compilé en langage machine avant d'être exécuté.
- Langage faiblement typé => lors de la déclaration des variables, le type de variable n'est pas spécifié. Il le sera lors de l'exécution du langage.
- Langage orienté objet
- ...

1- Javascript, c'est quoi



Les différentes versions de Javascript et les problèmes de compatibilité

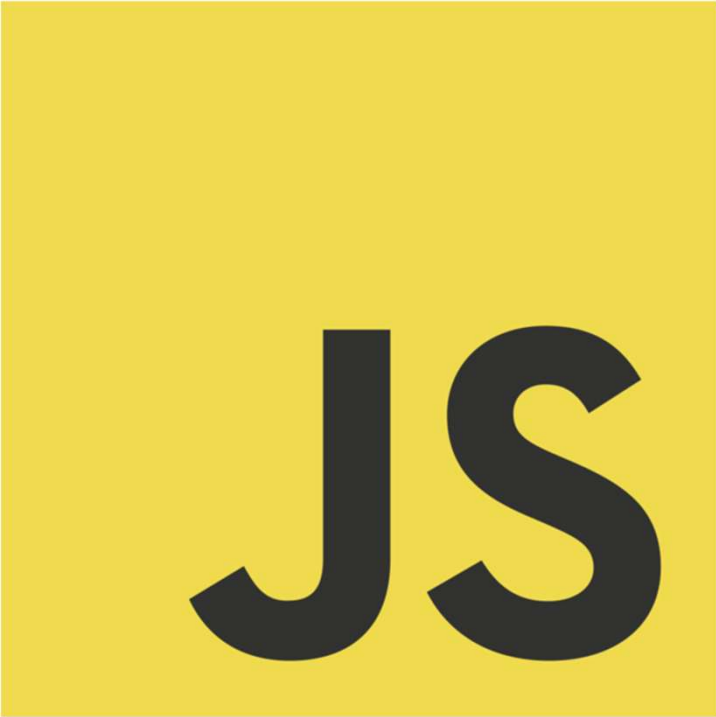
Depuis ECMAScript6 (ES6), une nouvelle version du langage sort chaque année.

Comme nous l'avons vu, Javascript est interprété par les navigateurs via leur moteur.

Malheureusement, les moteurs n'intègrent pas systématiquement et à la même vitesse les évolutions du langage Javascript.

Par contre, tous ont implémenté la version ES6.

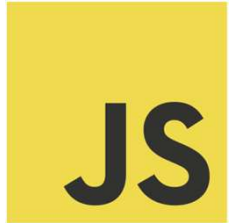
Ainsi si nous voulons utiliser les dernières évolutions du langage, il faudra utiliser un outil appelé Babel qui permet de transpiler votre code Javascript en une version compatible avec le moteur utilisé par votre navigateur.

A yellow square containing the letters 'JS' in a large, bold, dark grey sans-serif font.

JS

II. Mise en place de l'environnement

1- Installer GIT et Bash

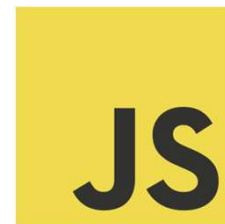


GIT, c'est quoi ?

C'est un système de contrôle de version. Il permet de versionner le code afin de connaître, durant la vie d'un projet, les étapes qui ont mené à la dernière version par exemple.

Bash c'est quoi ?

C'est un interpréteur en ligne de commande. Il permet d'exécuter des actions à l'aide de lignes de commande.



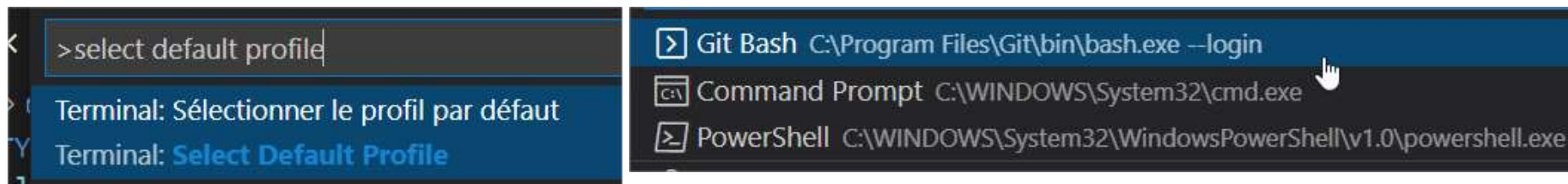
1- Installer GIT et Bash

Installer GIT

<https://git-scm.com/>

Installer Bash sur VSCode

CTRL+SHIFT+P puis taper Select Default Profile puis sélectionner GIT Bash



2- Installer et configurer Babel

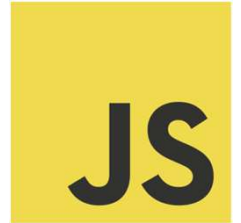


Babel c'est quoi ?

Babel est un transpileur JavaScript.

Il permet de transpiler du JavaScript très récent en JavaScript plus ancien pour qu'il soit compatible sur toutes les versions des navigateurs.

2- Installer et configurer Babel



- Créer un dossier – chapitre 2
- Ouvrez ce dossier dans VSCode
- Dans le terminal, initialisez npm à l'aide de la ligne de commande suivante:

`npm init` ou `npm init -y` Pour mettre les options par défaut

Un fichier package.json a été créé.

- Créez un fichier index.html
- Créez un fichier babel.config.js
- Installez les 3 packages de Babel à l'aide de la ligne de commande suivante

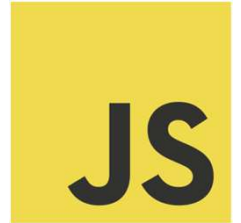
`npm install @babel/core @babel/cli @babel/preset-env`

On peut remarquer que des dépendances ont été ajoutées dans le fichier package.json.

Qu'un fichier package-lock.json s'est créé contenant la liste des dépendances

Qu'un dossier node_modules s'est créé contenant tous les packages que nous avons ajoutés ainsi que leurs dépendances

2- Installer et configurer Babel



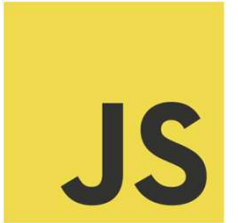
Si par erreur le dossier `node_modules` est supprimé. Il suffit de taper la ligne de commande `npm install` pour que npm lise le fichier `package.lock.json` afin d'installer toutes les dépendances nécessaires

- On va maintenant paramétrer le fichier `babel.config.js` en indiquant les lignes de commandes suivantes:

```
module.exports = {  
  presets: [  
    ["@babel/preset-env"]  
  ]  
}
```

- On va maintenant créer un dossier `src` où on mettra l'ensemble de nos dossiers js notamment avant passage dans Babel

2- Installer et configurer Babel



- Nous allons maintenant modifier notre fichier package.json afin de créer un script qui exécutera Babel en créant à partir du dossier src un dossier dist contenant le javascript qui aura été traité par Babel

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "babel": "babel src --out-dir dist"  
},
```

- Créons maintenant un exemple en créant un fichier myfile.js dans le dossier src avec le code suivant:

```
let test = "123";
```

- Tapons ensuite la ligne de commande suivante :

```
npm run babel
```

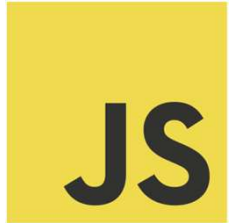

2- Installer et configurer Babel



- Un dossier dist s'est créé avec un fichier myfile.js
- Quand nous ouvrons ce fichier, nous voyons du code js mais différent car compréhensible par tous les navigateurs

```
"use strict";  
  
var test = "123";
```

3- Incorporer du JS dans un fichier HTML



- Il y a 2 manières de mettre du JS dans un fichier HTML
 - La moins utilisée

```
<title>Document</title>
<script>
  console.log('Hello');
</script>
</head>
```

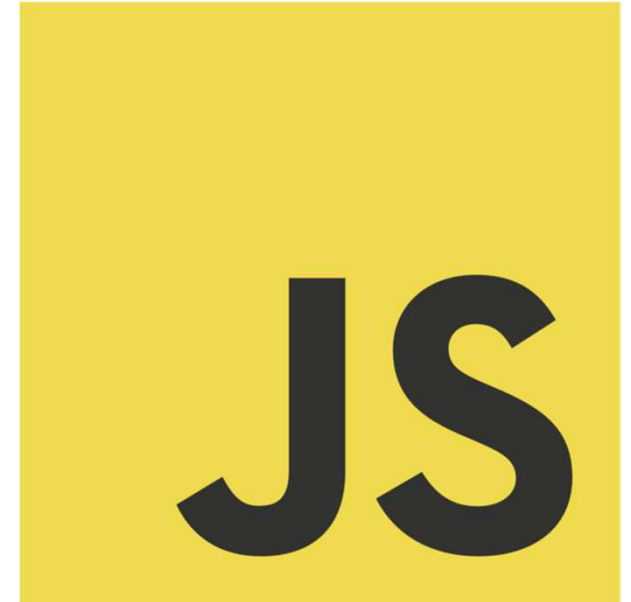
- La plus utilisée (et celle à privilégier)

```
<script src="dist/myfile.js"></script>
```

Si vous utilisez Babel, il est important d'aller chercher le fichier dans le dossier dist et non pas src, sinon le travail réalisé par Babel ne sert à rien

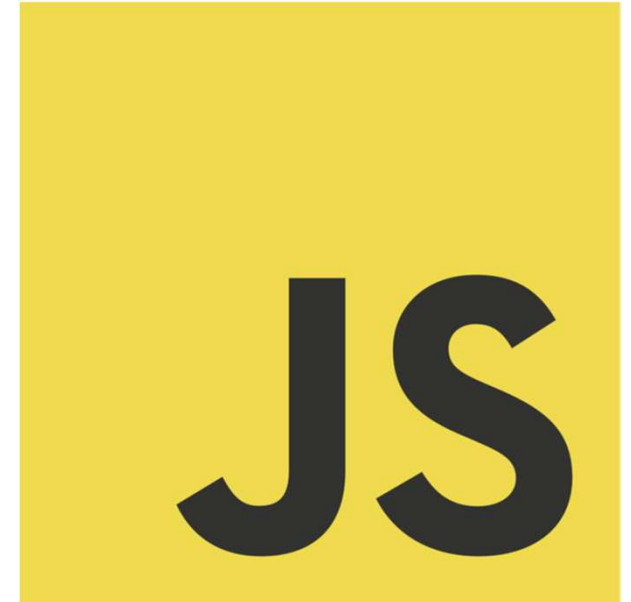
3- Incorporer du JS dans un fichier HTML

- Incorporer un fichier Javascript monjs.js dans un fichier HTML
- Durée 5 minutes



3- Incorporer du JS dans un fichier HTML

- Correction



4- Installer et configurer Webpack



Webpack, c'est quoi ?

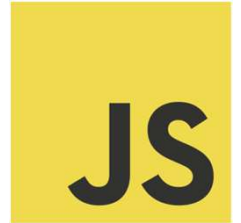
Il permet de rassembler les fichiers de votre application (fichiers JavaScript, HTML, CSS, les images etc) en un ou plusieurs modules.

Cela va donc limiter les requêtes HTTP et accélérer donc les pages.

Mais Webpack ne permet pas seulement de modulariser votre code, c'est un outil très puissant qui peut effectuer grâce à des loaders et des plugins un très grand nombre de tâches essentielles en développement Web :

- Il peut utiliser Babel pour transpiler votre JavaScript pour le rendre compatible.
- Il peut minifier et compresser le code, pour réduire la taille des fichiers et donc la latence réseau (temps nécessaire pour télécharger les fichiers avec HTTP).
- Il peut générer des noms de fichier pour optimiser la mise en cache.
- Il peut compiler le Sass en CSS.
- Il peut optimiser les images.
- ...

4- Installer et configurer Webpack

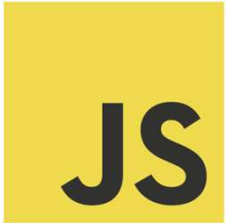


- On va commencer par quelques modifications dans notre projet:
 - On va renommer notre fichier myfile.js dans src en index.js
 - On va mettre le fichier index.html dans le dossier src
 - On supprime le dossier dist
 - Dans le fichier package.json, on va supprimer le script babel
- Maintenant on va installer Webpack en tapant la ligne de commande suivante :

```
npm install webpack webpack-cli webpack-dev-server babel-loader
```
- On va également installer un autre plugin en tapant la ligne de commande suivante :

```
npm install html-webpack-plugin
```

4- Installer et configurer Webpack



- On va ensuite créer notre fichier de configuration webpack.config.js avec le code suivant:

Fichier

Fichier webpack.config.js

- On va ensuite créer les scripts dans le fichier package.json

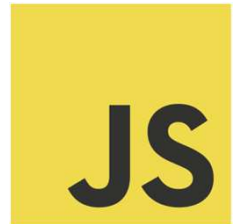
```
"webpack": "webpack",  
"start": "webpack server"
```

- Puis exécuter le premier script en utilisant la ligne de commande suivante :

```
npm run webpack
```

Un nouveau dossier dist s'est créé avec plusieurs fichiers à l'intérieur

4- Installer et configurer Webpack



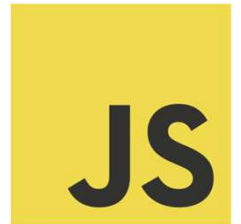
- On va maintenant exécuter le deuxième script en utilisant la ligne de commande :

```
npm run start
```

Au bout de quelques secondes la page HTML va s'ouvrir à l'aide du serveur qui s'est créé



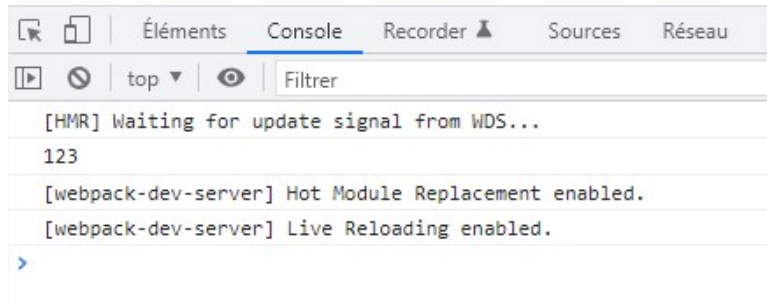
5- La console Javascript



La console permet d'avoir des informations sur le code javascript. On peut également afficher des informations dans cette console en tapant dans le code js le code suivant :

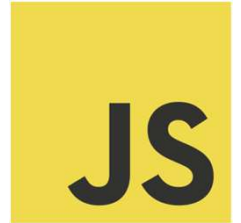
```
let test = "123";  
  
console.log(test);
```

Dans ce cas, on affiche la valeur de la variable test



Sur chrome, clic droit puis inspecter puis onglet « Console »

5- La console Javascript



En faisant un zoom sur la ligne suivante:

123 index.js:3

Vous avez à droite l'information d'où vient cet affichage et si vous cliquez sur `index.js:3`, la page suivante s'affichera

```
index.js x
1 let test = "123";
2
3 console.log(test);
```

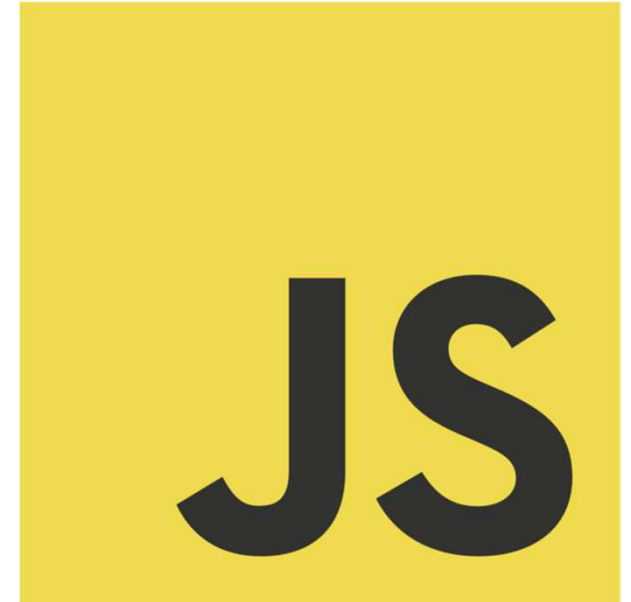
On peut remarquer que le 3 de `index.js:3` correspond à la ligne du code.

On peut remarquer également une autre chose, c'est que le code qui s'affiche est celui du fichier contenu dans le dossier `src` et pas `dist`.

Cela est dû au fichier `main.bundle.js.map` qui permet de faire le lien entre le fichier traité et le fichier d'origine. Cela facilite le débogage.

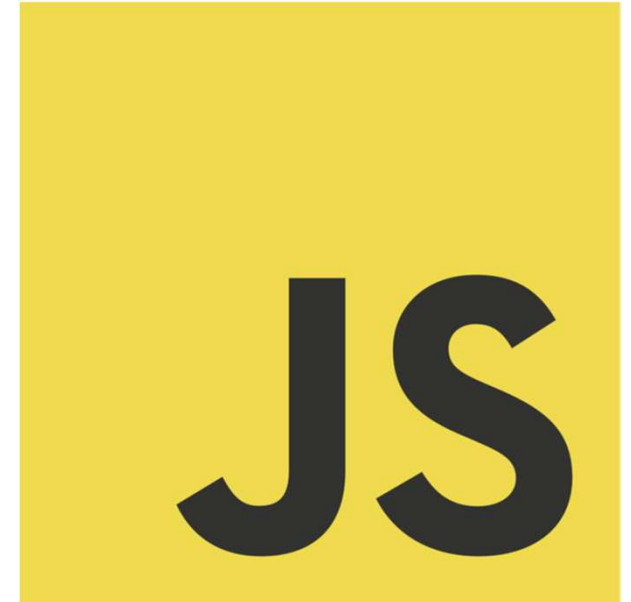
5- La console Javascript

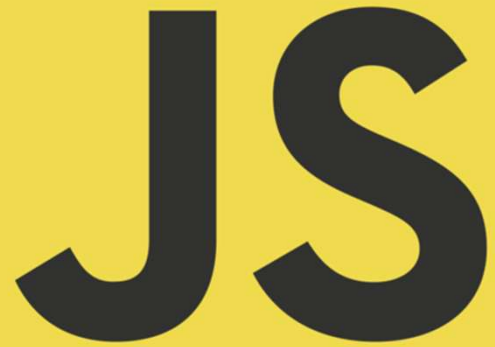
- Afficher dans la console du navigateur le texte "Hello World"
- Durée 5 minutes



5- La console Javascript

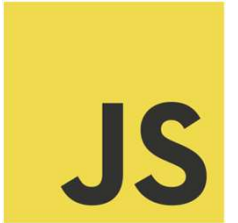
- Correction





III. Les bases du Javascript

1- Les commentaires et les variables



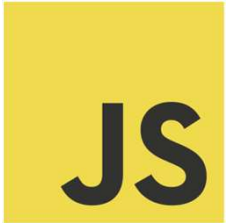
Voici les 2 manières pour écrire un commentaire dans un fichier Javascript

Exemple Fichier commentaires_et_variables.js

```
// Un commentaire sur une seule ligne

/*
Un commentaire sur
plusieurs lignes
*/
```

1- Les commentaires et les variables



Voici les 3 manières de déclarer une variable

Les déclarations de variables sont traitées avant que le code soit exécuté

Déclaration de la variable
Assignation d'une valeur à
la variable

```
var maVariable1;  
maVariable1 = 8;
```

La portée d'une variable déclarée avec var est le contexte d'exécution courant, c'est-à-dire : la fonction qui contient la déclaration ou le contexte global si la variable est déclarée en dehors de toute fonction.

```
let maVariable2;  
maVariable2 = -5;
```

let permet de déclarer des variables dont la portée est limitée à celle du bloc dans lequel elles sont déclarées tout comme const

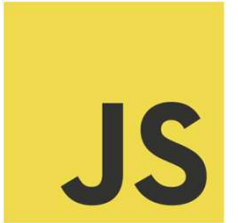
```
const maVariable3 = 6;
```

Les variables déclarés avec const doivent être assigné et déclaré en même temps
Elle est utilisé lorsque la valeur ne change pas.

Exemple

Fichier commentaires_et_variables.js

1- Les commentaires et les variables



Var et let peuvent être assigné et déclaré dans le même temps comme ici par exemple

```
var maVariable1 = 8;
```

var ne devrait plus être utilisé dans la Javascript moderne, il faut donc éviter de le faire. Compte tenu de ses caractéristiques, il peut entraîner des comportements inappropriés.

L'ordre de priorité d'utilisation des différentes déclarations des variables est :

1. const
2. let
3. var

Attention, il ne faut pas oublier à chaque fin d'instruction d'ajouter un point-virgule. Ce n'est pas forcément obligatoire en Javascript mais c'est recommandé.

1- Les commentaires et les variables

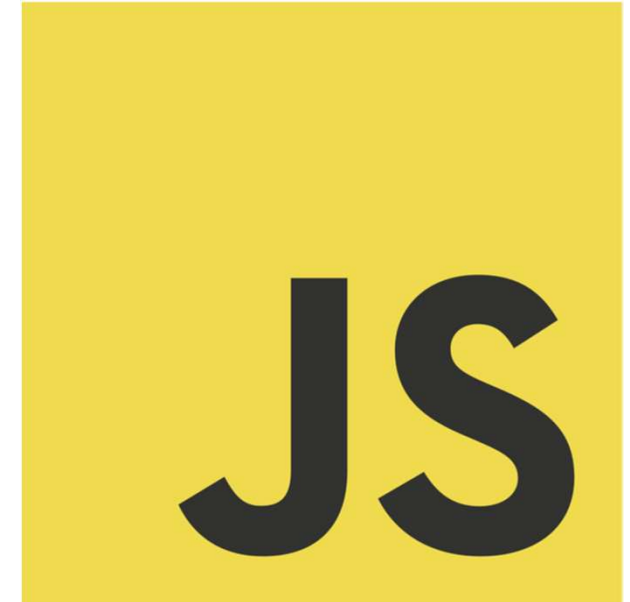


Règle de déclaration d'une variable

- Le nom d'une variable ne doit pas commencer par un chiffre
- Le nom d'une variable peut contenir les caractères suivants:
 - Des chiffres
 - Des lettres
 - Des \$
 - Des _
 - Pas d'accent ni espace
- Lorsque la variable contient plusieurs mots, il est, par convention, demandé de les écrire en camelCase
 - `maVariableTresLongue`

1- Commentaires et variables

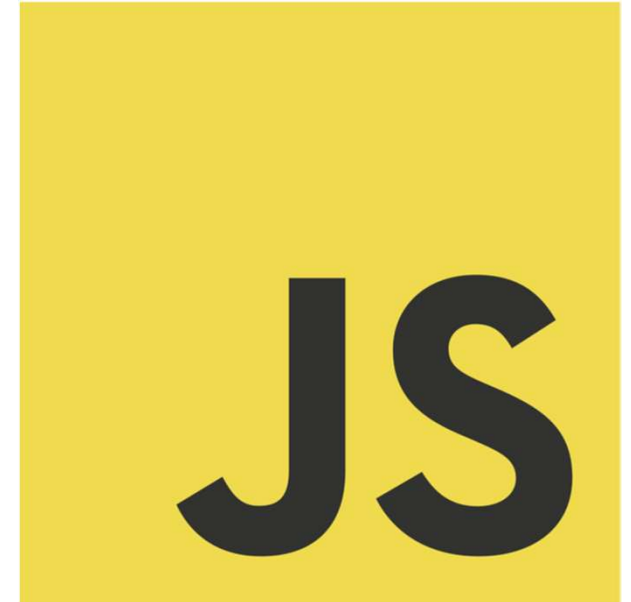
- Ecrivez dans un fichier javascript, un commentaire en utilisant l'écriture pour une ligne et celle pour plusieurs lignes
- Déclarez une constante "age" avec la valeur 24 et une variable "quantite" avec la valeur 3
- Et affichez-les dans la console de votre navigateur
- Durée 10 minutes



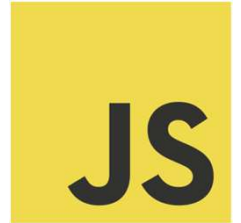
1- Commentaires et variables

- Correction

Exercice



2- Les types de variables



2 types de variables

- les primitives
- Les objets

Les principales variables primitives	Les principales variables objets
Booléen (true ou false)	Objets
Chaînes de caractère (string)	Fonction
Nombres (number)	Tableau (array)
Null	...
Undefined	...
...	...

L'opérateur typeof renvoie une chaîne qui indique le type de son opérande

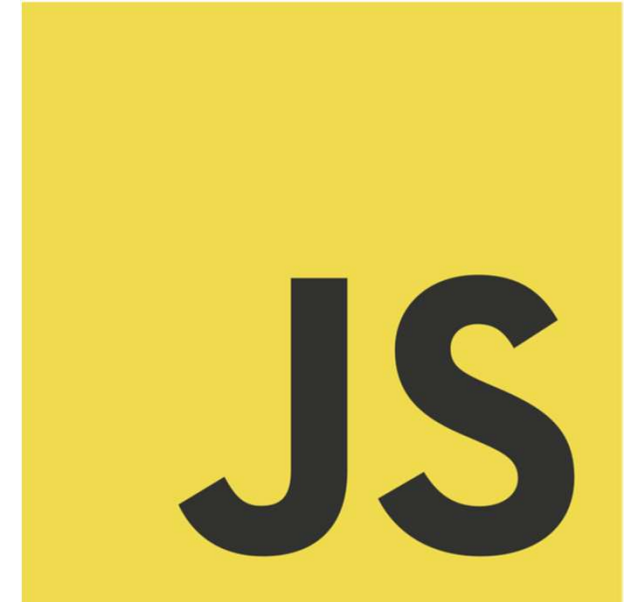
Syntaxe :
typeof operande

Exemple

Fichier types_de_variable.js

2- Les types de variables

- Déclarer et assigner dans un fichier javascript,
 - Une variable de type boolean
 - Une variable de type chaîne de caractère
 - Une variable de type nombre
 - Une variable de type null
 - Une variable de type undefined
 - Une variable de type objet
 - Une variable de type tableau
 - Une variable de type fonction
- Consignes: donner le nom et la valeur que vous voulez
- Durée 10 minutes



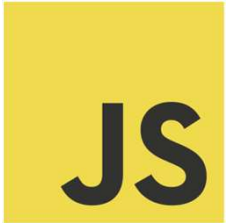
2- Les types de variables

- Correction

Exercice



3- Les opérateurs arithmétiques



Les opérateurs arithmétiques

Opérateurs	Résultat
... = ...	Permet d'assigner une valeur
... + ...	Addition
... - ...	Soustraction
... * ...	Multiplication
... / ...	Division
... % ...	Modulo (retourne le reste d'une division)
...++	Incrémententation
...--	Décrémententation
... += ...	Addition et Assignment
.....

Exemple

Fichier operators.js

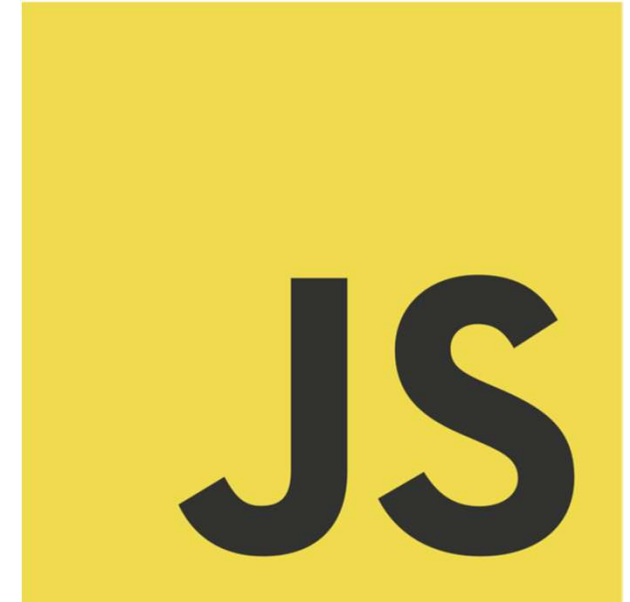
https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Expressions_and_Operators

L'ordre d'exécution des opérateurs arithmétiques en Javascript est identique à celui en mathématique

() puis * et / puis + et -

3- Les opérateurs arithmétiques

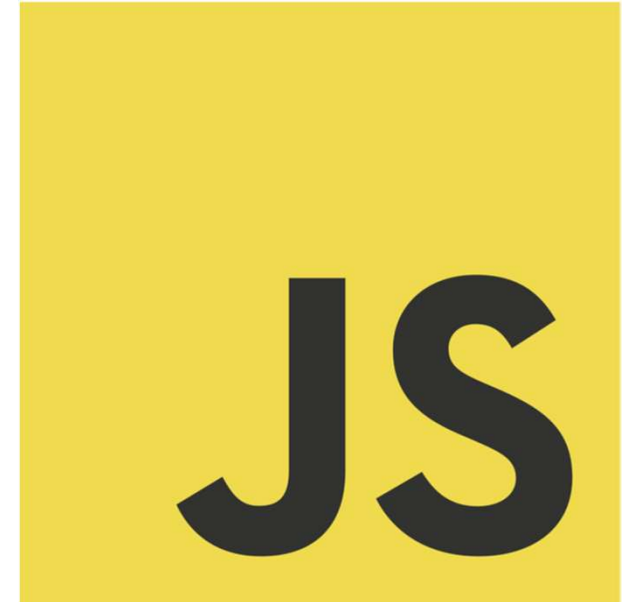
- Créez les variables suivantes et opérations:
 - a avec la valeur 8
 - b avec la valeur 5
 - c résultat de l'addition de a et de b
 - d résultat de la soustraction de a et de b
 - e résultat de la multiplication de a et de b
 - f résultat de la division de a et de b
 - g reste de la division de a et de b
 - Incrémentez f
 - Décrémentez g
 - Ajoutez 5 à a
 - Concatenez la variable h avec la valeur "Bonjour" et la variable i avec la valeur " tous le monde" et mettre le résultat dans une variable j
- Durée 15 minutes



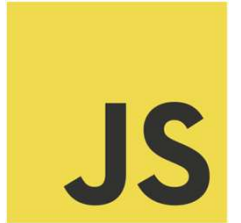
3- Les opérateurs arithmétiques

- Correction

Exercice



4- Les conversions implicites

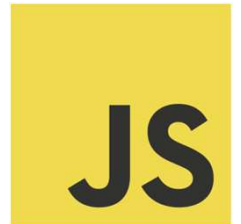


Que se passe-t-il si par exemple nous additionnons un nombre avec une chaîne de caractère ?

Pour résoudre ce dilemme, Javascript va devoir convertir l'une des 2 valeurs dans le type de l'autre. C'est ce qu'on appelle la conversion implicite.

Il faut faire attention à éviter d'additionner des variables de types différents car cela peut entraîner des résultats surprenants.

5- Les opérateurs de comparaison



Ces opérateurs permettent de comparer 2 valeurs et retournent le résultat:

- true si la comparaison est vraie
- false si elle est fautive

Opérateurs	Résultat
a == b	Vérifie si 2 valeurs sont égales (attention si les valeurs sont de 2 types différents, Javascript utilisera la conversion implicite)
a === b	Vérifie si 2 valeurs sont strictement égales (Javascript ne convertira pas les valeurs)
a > b	Vérifie si a est supérieur à b
a >= b	Vérifie si a est supérieur ou égal à b

Opérateurs	Résultat
a < b	Vérifie si a est inférieur à b
a <= b	Vérifie si a est inférieur ou égal à b
a != b	Vérifie si les 2 valeurs sont différentes (attention si les valeurs sont de 2 types différents, Javascript utilisera la conversion implicite)
a !== b	Vérifie si 2 valeurs sont strictement différentes (Javascript ne convertira pas les valeurs)

5- Les opérateurs de comparaison



Il est également possible de retourner un résultat true ou false de plusieurs comparaisons. C'est le rôle des opérateurs suivants :

- && signifie « et » (vérifie si les 2 conditions sont vrais)
- || signifie « ou » (vérifie si l'une ou l'autre des 2 conditions est vrai)

Résultat condition 1	Résultat condition 2	Résultat avec &&	Résultat avec
true	true	true	true
false	false	false	false
true	false	false	true
false	true	false	true

Exemple

Fichie operateurs_de_comparaison.js

5- Les opérateurs de comparaison

- Créer 2 variables “a” et “b” (a = 2 et b = 6)
- Puis retourner dans la console la réponse au comparaison suivante :
 - Est-ce que “a” est égal à “b” ?
 - Est-ce que “a” est supérieur ou égale à “b” ?
 - Est-ce que “a” est inférieur à “b” ?
 - Est-ce que “a” est different de “b” ?
- Durée 10 min



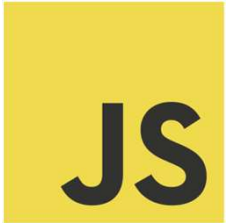
5- Les opérateurs de comparaison

- Correction

Exercice



6- Référence et valeur



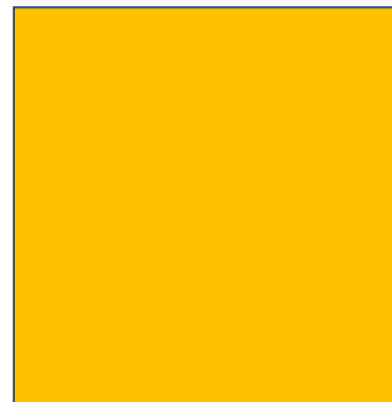
Une variable Javascript est stockée sur la RAM de l'ordinateur. Pour cela le moteur Javascript va leur attribuer une adresse (que l'on appelle la référence) puis enregistre sa valeur.

La référence des variables est stocké dans la stack.

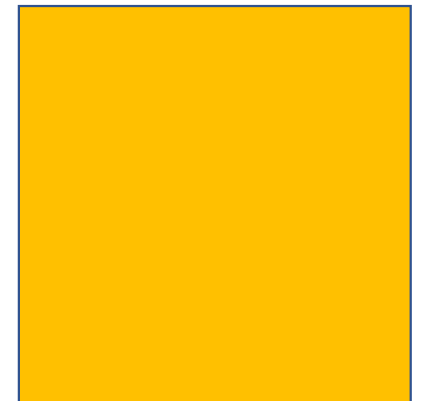
Pour les primitives, sa valeur est également stocké dans la stack.

Par contre les valeurs des variables de type objets ne sont pas stockées de la même manière. En effet, comme la taille d'un objet est difficilement prévisible, sa valeur va être stocké dans une autre zone moins rapidement accessible mais capable d'enregistrer des informations plus volumineuse (c'est la heap)

Stack



Heap



6- Référence et valeur



Script

```
let a = 'Salut';  
let b = { nom : 'Henri'}  
let c = a;  
let d = b;
```

Stack

```
a => x1 | 'Salut'  
b => x2 | x3  
c => x4 | 'Salut'  
d => x5 | x3
```

Heap

```
x3 | {nom : 'Henri'}
```

x^* : adresse (référence)

c copie a par valeur car a est une primitive

d copie b par référence car b est une variable de type objet

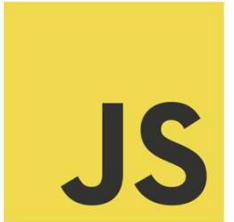
Exemple

Fichier `reference_et_valeur.js`



IV. Les conditions et boucles

1- if, else, else if



Ces mots clés permettent de réaliser un programme conditionnel.

```
// if va vérifier que la condition retourne true
if (condition){
    //code à exécuter
}
```

```
/*
else permettra d'exécuter un code si la condition
if n'est pas true
*/
if (condition){
    //code à exécuter si true
}else {
    //code à exécuter si false
}
```

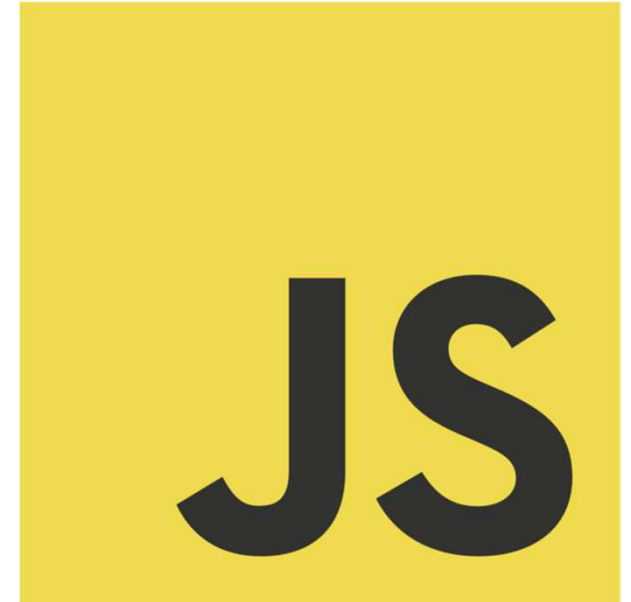
```
/*
else if permettra de vérifier une nouvelle condition
si if n'est pas true
*/
if (condition){
    //code à exécuter si condition est true
}else if(condition2){
    //code à exécuter si condition est false
    //et condition2 est true
}else {
    //code à exécuter si condition et condition2 sont
    //false
}
```

Exemple

Fichier conditionnel.js

1- if, else, else if

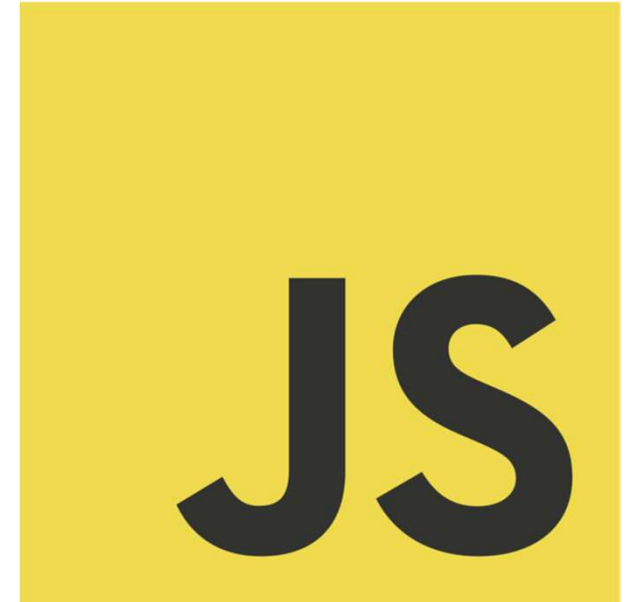
- Exercice 1:
 - créer un programme qui permettra de vérifier si je suis majeur ou mineur à partir d'une valeur saisie dans une variable
 - Afficher dans la console : "Je suis majeur" ou "Je suis mineur"
- Exercice 2 :
 - Créer un programme qui affichera "Je suis majeur" ou "J'ai plus de 16ans" ou "J'ai moins de 16 ans"



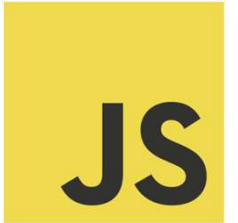
1- if, else, else if

- Correction Exercice age

exercice



2- Opérateur ternaire



L'opérateur ternaire est une autre manière de vérifier des conditions.

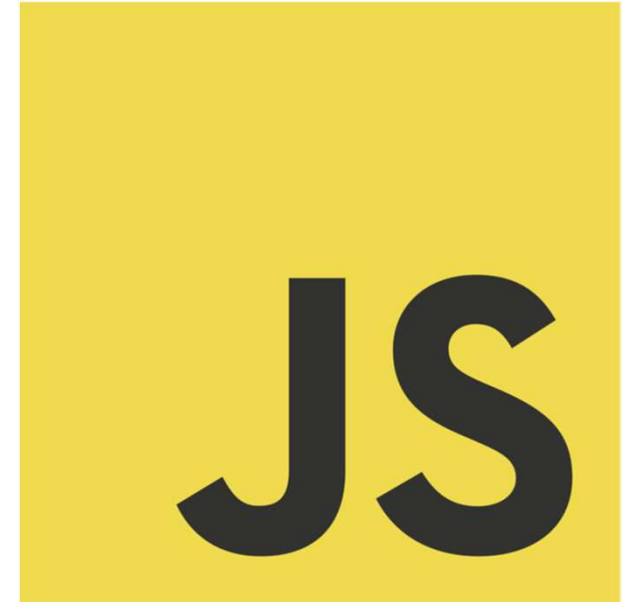
```
condition ? true : false;
```

Exemple

Fichier operateur_ternaire.js

2- Opérateur ternaire

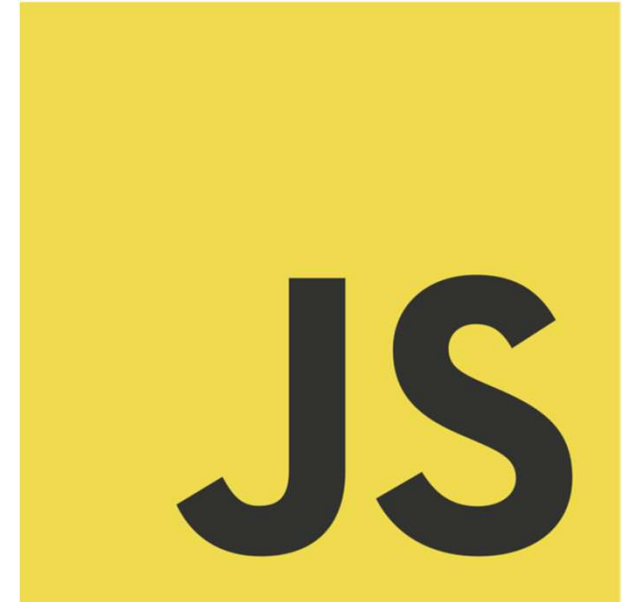
- Exercice :
 - En utilisant les ternaires, vérifier si la variable `firstname` est égale à 'Henri' et afficher dans ce cas "Tu es Henri".
 - Dans le cas contraire, afficher



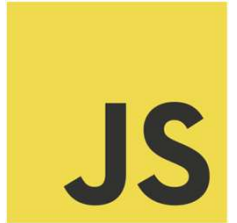
2- Opérateur ternaire

- Correction Exercice Ternaire

exercice



3- Switch



Switch est une autre manière d'exécuter du code selon la valeur d'une variable.

Syntaxe

```
const a = 5;
switch (a) {
  case 9:
    // code à exécuter
    break;
  case 7:
    // code à exécuter
    break;
  case 5:
    // code à exécuter
    break;
  default:
    // code à exécuter
}
```

On indique, dans le mot clé `switch`, la variable à vérifier

Après chaque mot clé `case`, j'indique l'une des valeurs possibles de la variable
Si la variable est égale à cette valeur, j'exécute le code sinon je continue.

Si la valeur de la variable ne correspond à aucun des `case` répertoriés alors le code après `default` va s'exécuter.

ATTENTION : il est important à la fin de chaque `case`, d'indiquer le mot clé `break` qui permet de sortir du bloc (ici le bloc `switch`).

En effet, sans cela, même si la `case` a été trouvée dès le début, la fonction continuera à vérifier chaque autre `case` alors que cela ne sert à rien.

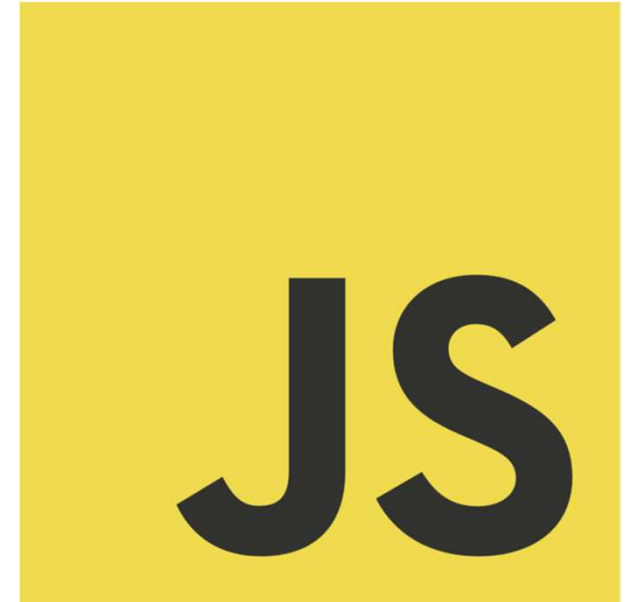
De plus, si aucun `break` n'ai mis dans les `case`, le code indiqué après `default` s'exécutera aussi.

Exemple

Fichier switch.js

3- Switch

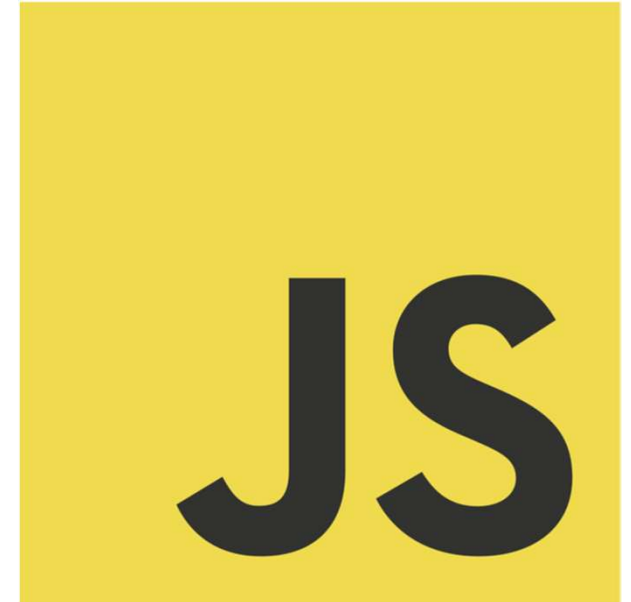
- Exercice :
 - En utilisant un switch, verifier qu'une variable roue puisse être égale à 4 et afficher dans ce cas "C'est une voiture" ou bien 2 et afficher dans ce cas "C'est une moto"
 - Dans tout autres cas, afficher "C'est un camion"



3- Switch

- Correction Exercice Switch

exercice



4- Boucle for



La boucle for permet de répéter un programme un certain nombre de fois

Syntaxe

```
for (initialisation des compteurs; conditions; incrémentations) {  
    // instructions à exécuter;  
}
```

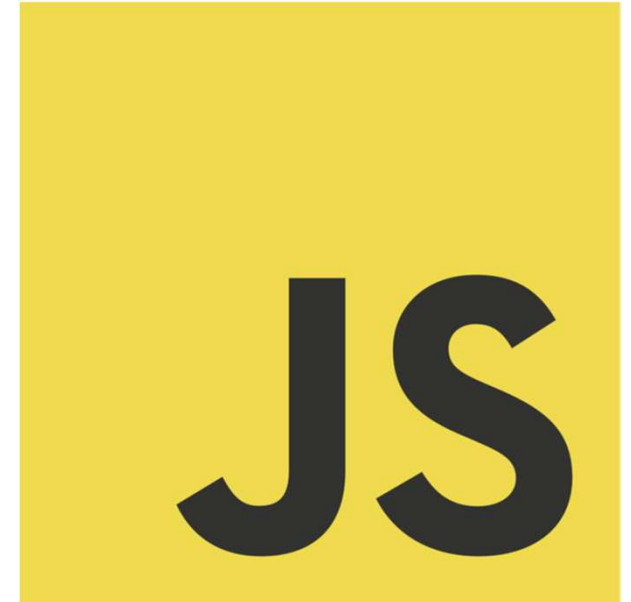
ATTENTION aux boucles infinies

Exemple

Fichier for.js

4- Boucle for

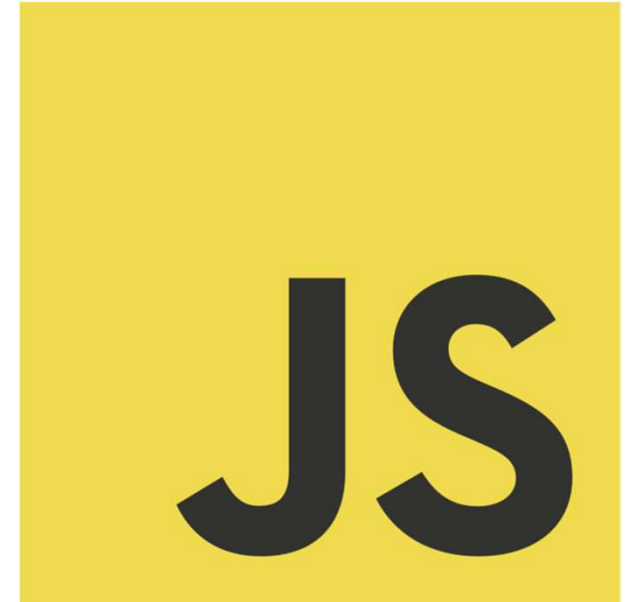
- Exercice :
 - Faire une boucle for permettant d'afficher les nombres de 0 à 9 dans la console
 - L'affichage des nombres doit se faire 1 par ligne
- Durée : 5 min

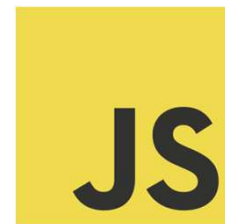


4- Boucle for

- Correction

exercice





5- Boucle while et do while

La boucle while permet d'exécuter un code tant que la condition indiquée n'est pas atteinte

Syntaxe

```
while (condition){  
    //code à exécuter  
}
```

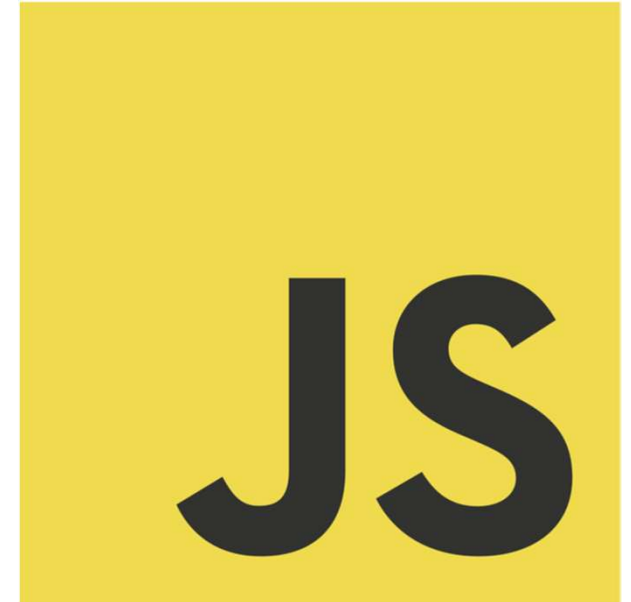
ATTENTION aux boucles infinies

Exemple Fichier while.js

La boucle do while permet d'exécuter un code au moins une fois puis rentre dans la boucle

5- Boucle while et do while

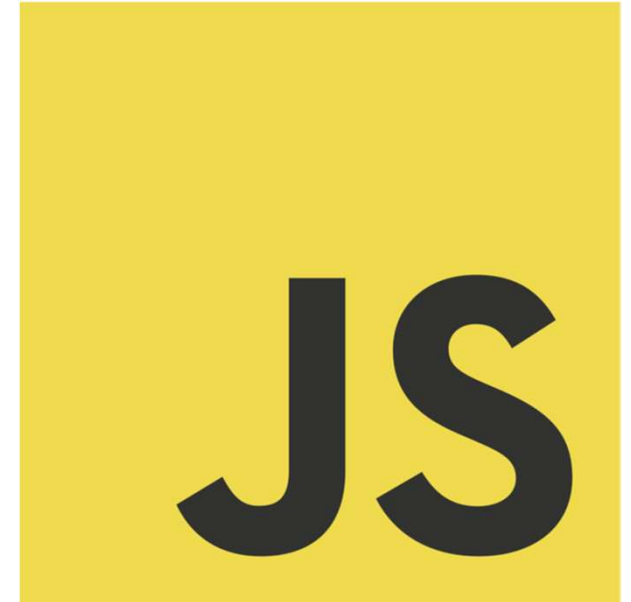
- Exercice :
 - Afficher dans la console à l'aide d'une boucle while le résultat de la table de multiplication des 7
 - Durée 5 minutes



5- Boucle while et do while

- Correction

exercice





V. Les nombres

1- Déclarer un nombre



Il y a 3 manières de déclarer un nombre

```
const a = 45;  
  
const b = Number(45);  
  
const c = new Number(45);
```

Les 2 premières sont équivalentes et nous privilégierons donc la 1^{ère} car plus simple à écrire.

La dernière manière est intéressante car elle permet d'écrire un équivalent objet au nombre.

L'avantage d'un objet est qu'elles peuvent avoir des méthodes (c'est-à-dire des fonctions) permettant de manipuler l'objet.

Exemple

Dossier déclarer_un_nombre

2- Quelques méthodes de Number



La méthode `toFixed` permet d'arrondir un nombre et de le retourner en chaîne de caractère
Le paramètre sera alors un nombre correspondant au nombre de chiffre après la virgule (pour l'arrondi)

```
variable.toFixed(2);
```

Par contre, cette méthode retourne une chaîne de caractère.

Si vous voulez transformer cette chaîne de caractère, vous pouvez utiliser les fonctions natives `parseInt` ou `parseFloat` de la manière suivante :

```
parseInt(variable);
```

`parseInt` transformera en nombre entier

```
parseFloat(variable);
```

`parseFloat` transformera en nombre décimal

Exemple

Dossier déclarer_un_nombre

2- Quelques méthodes de Number



Javascript est un langage qui est bien fait et il permet donc d'utiliser les méthodes de l'objet number directement sur un nombre déclaré de la manière suivante:

```
const a = 45;  
a.toFixed(2)
```

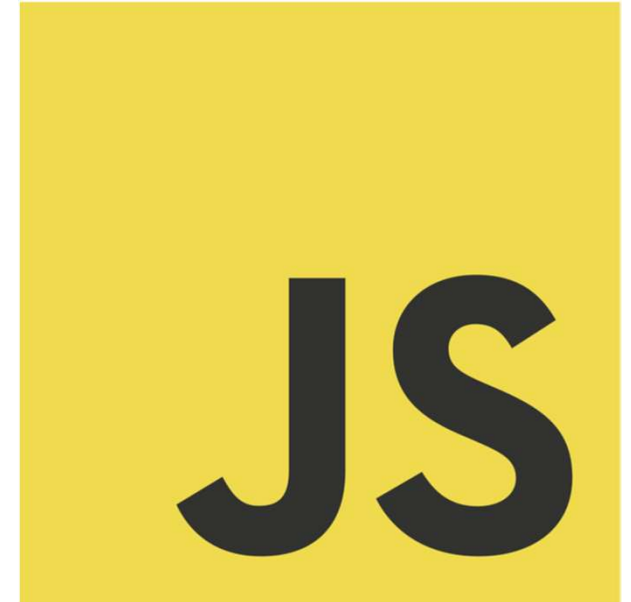
Dans ce cas, Javascript transforme momentanément la const a en objet number le temps d'utiliser la méthode.

Exemple

Dossier déclarer_un_nombre

2- Quelques méthodes de Number

- Exercice :
 - Déclarer une variable avec un nombre decimal avec au moins 4 chiffres après la virgule.
 - Arrondir cette variable à 2 chiffres après la virgule puis afficher là dans la console en tant que nombre



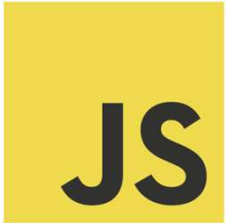
2- Quelques méthodes de Number

- Correction

Exercice



3- L'objet Math



L'objet Math possède de nombreuses méthodes permettant de gérer les mathématiques facilement.

Voici quelques exemples de méthode :

Math.ceil() => arrondi supérieur du nombre en paramètre

Math.floor() => arrondi inférieur du nombre en paramètre

Math.round() => arrondi au plus proche du nombre en paramètre

Math.PI => retourne le nombre Pi

Math.max() => retourne la valeur max des nombres en paramètre

Math.min() => retourne la valeur min des nombres en paramètre

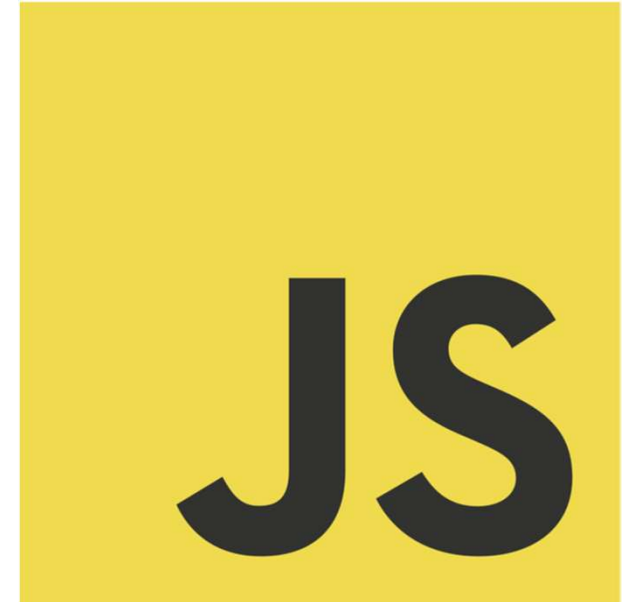
Math.random() => retourne un chiffre aléatoire entre 0 et 1

Exemple

Dossier Objet Math

3- L'objet Math

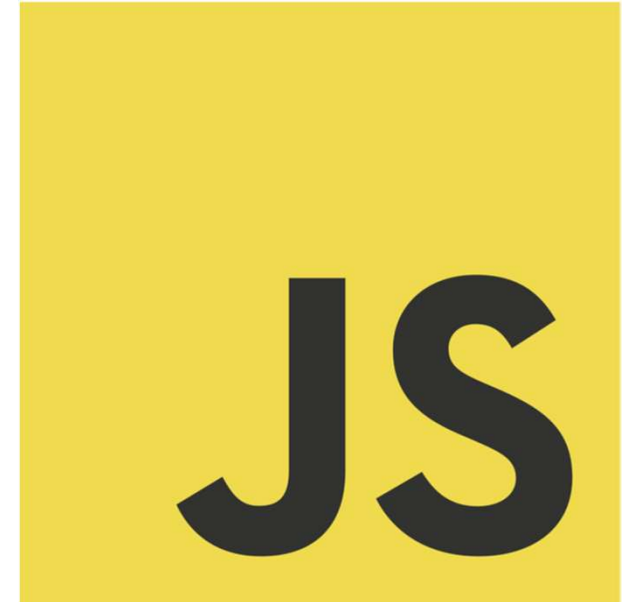
- Exercice :
 - Déclarer une variable "num" avec la valeur 3,46
 - En utilisant l'objet Math, affichez l'arrondi à l'entier supérieur de "num"
 - Affichez ensuite dans la console le nombre le plus grand parmi les nombres suivants : 5, 65, 123, 4 en utilisant l'objet Math
 - Affichez dans la console un nombre aléatoire entre 0 et 1
- Durée : 5 min

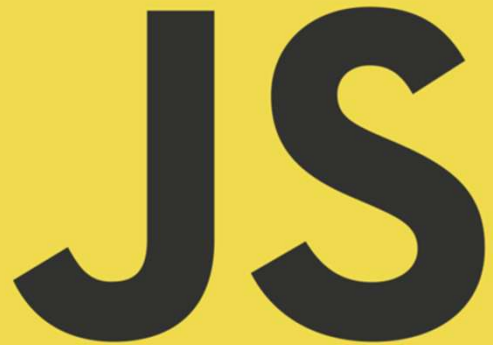


3- L'objet Math

- Correction

Exercice





VI. Les chaînes de caractères

1- Déclarer une chaîne de caractères



Une chaîne de caractère peut-être déclaré de 3 manières différentes :

```
const a = "Bonjour tous le monde";  
  
const b = String("Bonjour tous le monde");  
  
const c = new String("Bonjour tous le monde");
```

Les 2 premières sont équivalentes et nous privilégierons donc la 1^{ère} car plus simple à écrire.

La dernière manière est intéressante car elle permet d'écrire un équivalent objet à la chaîne de caractères.

L'avantage d'un objet est qu'elles peuvent avoir des méthodes (c'est-à-dire des fonctions) permettant de manipuler l'objet.

Exemple

Dossier chaîne de caractères

1- Déclarer une chaîne de caractères



Lors de la déclaration d'une chaîne de caractères, il est possible que vous vouliez déclarer à l'intérieur de cette chaîne des guillemets

Lorsque c'est le cas, nous pouvons utiliser le caractère d'échappement : `\"`

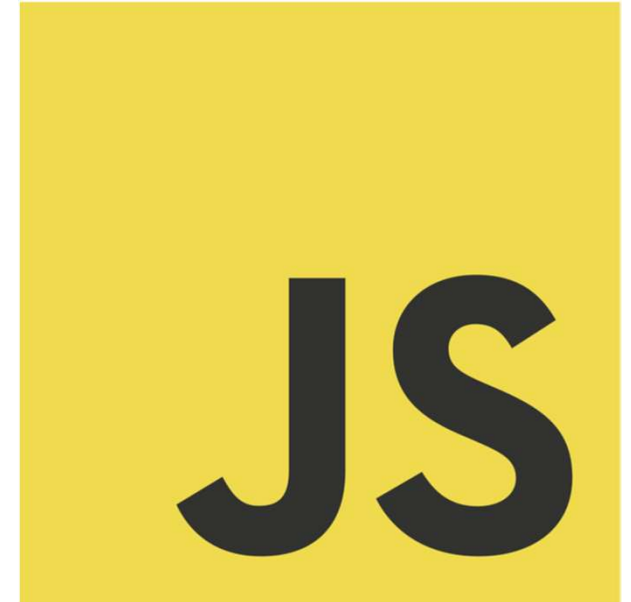
Si vous voulez que le texte apparaisse sur plusieurs lignes, vous devez utiliser l'écriture multiligne.
Les simples ou doubles guillemets sont alors remplacés par des accents (utiliser sur windows ctrl+alt+7)

Exemple

Dossier chaîne de caractères

1- Déclarer une chaîne de caractères

- Exercice :
 - Déclarer une chaîne de caractères “chaîne” avec la valeur “Hello World” en utilisant des guillemets doubles
 - Déclarer une deuxième chaîne de caractères “chaîne2” en utilisant les guillemets simples et avec la valeur “Message d’aujourd’hui”
 - Déclarer une chaîne de caractères “chaîne3” sur plusieurs lignes avec la valeur
 - Quand le ciel bas et lourd pèse comme un couvercle
Sur l'esprit gémissant en proie aux longs ennuis,
Et que de l'horizon embrassant tout le cercle
Il nous verse un jour noir plus triste que les nuits
- Durée : 5 min



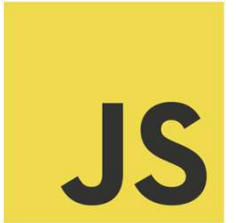
1- Déclarer une chaîne de caractères

- Correction

Exercice



2- Propriétés et quelques méthodes de l'objet String



Tout comme pour les Number, Javascript est un langage qui est bien fait et il permet donc d'utiliser les méthodes de l'objet String directement sur une chaîne de caractères déclarée de la manière suivante:

```
const a = "Bonjour tous le monde";
```

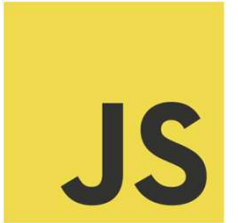
Dans ce cas, Javascript transforme momentanément la const a en un objet String le temps d'utiliser la méthode.

Propriété length => retourne le nombre de caractères de la chaîne de caractères

Exemple

Dossier chaîne de caractères

2- Propriétés et quelques méthodes de l'objet String



La notion d'index

Une chaîne de caractère se comporte un peu comme un tableau dont chaque caractère a son propre index. Ainsi on peut sélectionner un des caractères d'une chaîne de caractères à l'aide de celui-ci

```
const a = "Bonjour tous le monde";
```

`a[2]` Retournera alors n

Attention, les index commence par 0.

Exemple

Dossier chaîne de caractères

2- Propriétés et quelques méthodes de l'objet String



indexOf(param)

Le paramètre est la sous-chaîne que l'on recherche dans la chaîne de caractères où cette méthode est utilisée.
Retourne la valeur de l'index du 1^{er} caractère de la sous-chaîne
Si la sous-chaîne n'existe pas, retourne -1

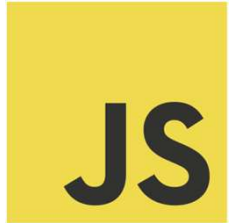
replace(param1, param2)

Param1 est la sous-chaîne que nous allons rechercher et remplacer
Param2 est la sous-chaîne que nous allons utiliser pour le remplacement
ATTENTION, La méthode retourne une chaîne de caractère avec le remplacement mais elle ne modifie en rien la variable d'origine

Exemple

Dossier chaîne de caractères

2- Propriétés et quelques méthodes de l'objet String



trim()

Retourne une chaîne de caractères sans les espaces de début et de fin

slice(param1, param2)

Retourne une sous-chaîne de la chaîne d'origine dont le début correspond au param1 et la fin au param2 de la chaîne où s'applique la méthode

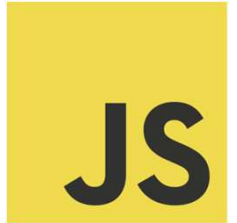
Param1 correspond à l'index de début

Param2 correspond à l'index de fin

Exemple

Dossier chaîne de caractères

2- Propriétés et quelques méthodes de l'objet String



toUpperCase()

Retourne la chaîne de caractère où s'applique la méthode mais entièrement en majuscule

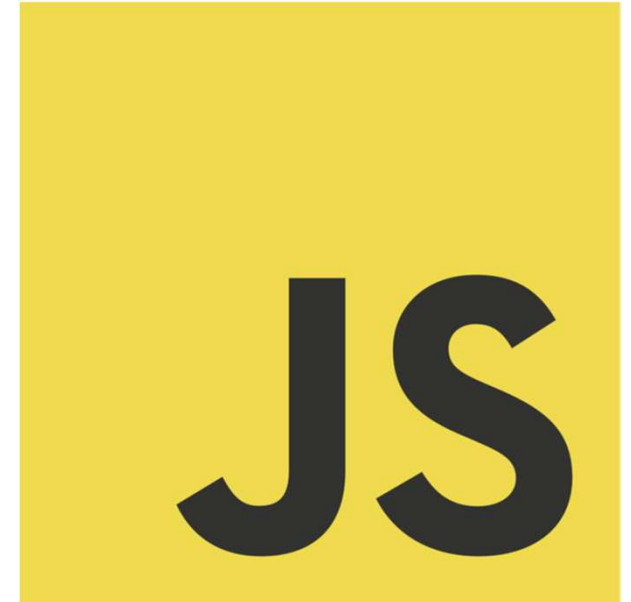
split(param)

Retourne un tableau de la chaîne de caractères suivant le paramètre. Cela peut permettre de retourner un tableau avec chaque mot en utilisant comme paramètre l'espace

Exemple Dossier chaîne de caractères

2- Propriétés et quelques méthodes de l'objet String

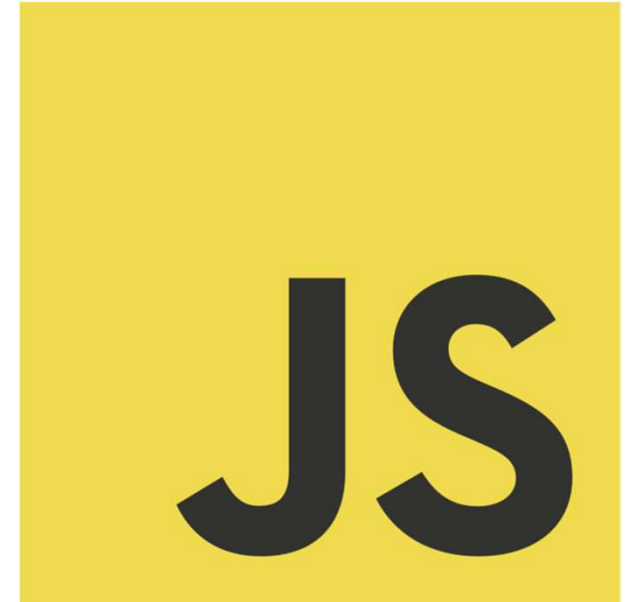
- Exercice :
 - Déclarer une chaîne de caractères "chaîne" avec la valeur "Hello World"
 - Afficher dans la console le nombre de caractères de cette chaîne de caractères "chaîne"
 - Récupérer l'index de début du mot "World" et l'afficher dans la console.
 - Remplacer "World" par "tous le monde" à l'aide d'une fonction et enregistrer dans une variable "chaîne2"
 - Afficher dans la console la valeur de "chaîne2" en majuscule
- Durée : 5 min



2- Propriétés et quelques méthodes de l'objet String

- Correction

Exercice



3- Les expressions rationnelles Regex



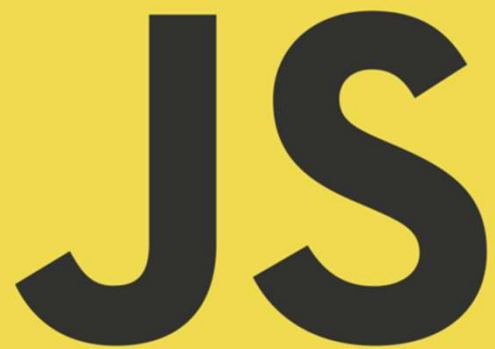
Les Regex servent à repérer dans une chaîne de caractère si celle-ci possède un modèle de chaîne ou bien si cette chaîne correspond à un modèle.

Ce sera utilisé, par exemple, pour vérifier qu'un champ rempli correspond bien à une adresse mail ou à un numéro de téléphone ...

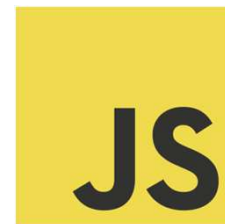
Déclarer une regex

```
const h = / /;  
const i = new RegExp();
```

Exemple Dossier chaîne de caractères



VII. Les objets



1- Un objet c'est quoi

Un objet est en Javascript toutes variables qui n'est pas de type primitive.

Déclarer un objet

```
const a = {};  
const b = Object();  
const c = new Object();
```

Ces 3 manières de déclarer un objet n'ont aucune différence

Un objet contient une ou plusieurs paires clé – valeur.

Ces paires clés – valeurs sont appelés « propriété »

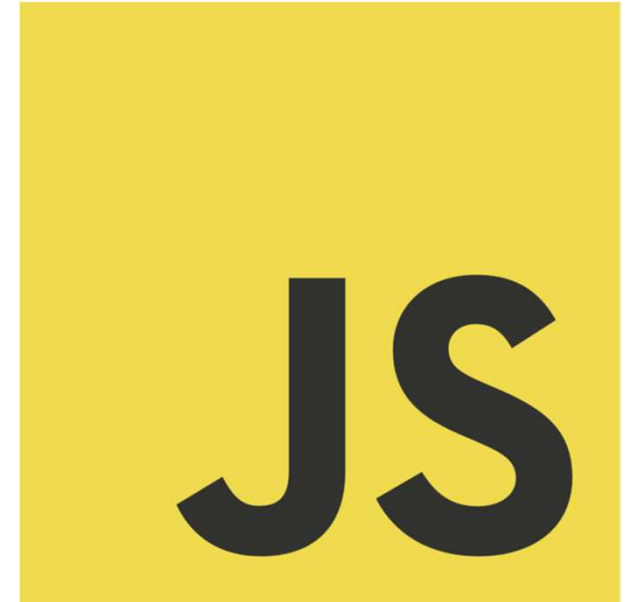
Les clés sont des chaînes de caractères.

Les valeurs peuvent être des variables de type primitive ou bien d'autres objets.

Exemple Dossier les objets

1- Un objet c'est quoi

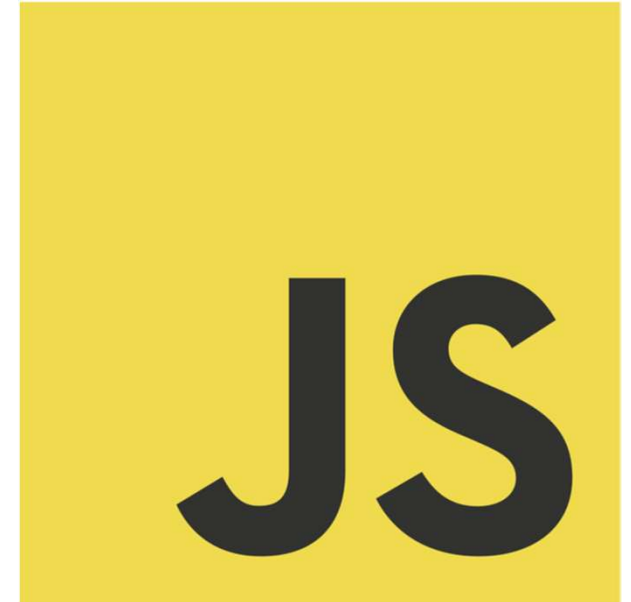
- Exercice :
 - Déclarer un objet "obj" avec les informations suivantes, nom et prénom et les valeurs relatives : votre nom et prénom
 - Durée : 5 min



1- Un objet c'est quoi

- Correction

Exercice





2- Les propriétés des objets

Les propriétés des objets peuvent être déclarées de plusieurs manières.

```
const people2 = {  
  nom: "Jimmy",  
  prenom: "Connors"  
}
```

Directement dans l'objet

```
people2.age = 69;
```

En utilisant le point

```
people2["trophee"] = 8;
```

En utilisant les crochets

Exemple Dossier les objets

2- Les propriétés des objets

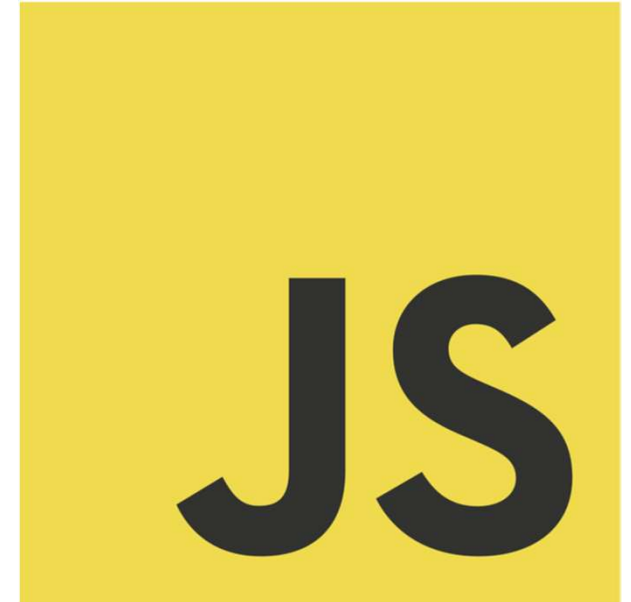


Les valeurs des propriétés des objets peuvent contenir être des variables de type primitive mais également d'autres objets comme des fonctions (on parle alors pour des fonctions d'objets de méthodes).

Exemple Dossier les objets

2- Les propriétés des objets

- Exercice :
 - A l'aide de l'exercice précédent, affichez dans la console la propriété nom de l'objet
 - Ajouter ensuite dans cet objet la propriété age et sa valeur (prenez votre âge en tant que Valeur)
- Durée : 5 min



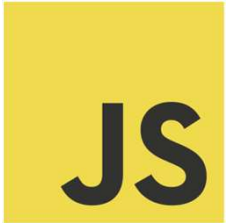
2- Les propriétés des objets

- Correction

Exercice



3- La décomposition d'objets



La décomposition d'objet est une nouvelle manière de récupérer des propriétés à partir d'un objet. Nous avons vu que nous pouvions récupérer une propriété d'un objet de la manière suivante

`people1.age` On récupère alors la propriété age de l'objet people1

La décomposition d'objet permet de récupérer en 1 ligne plusieurs propriétés d'un objet. Voici la syntaxe.

```
const { prop1, prop2, ...lereste } = objet;
```

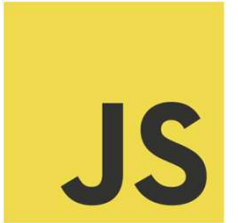
Cela retournera dans ce cas pour chaque propriété indiquée qui doivent avoir le même nom que celui dans l'objet plusieurs variables correspondantes.

L'indication `...lereste`, permettra de stocker toutes les autres propriétés qui ne seraient pas indiqués dans une variable `lereste`.

Cette dernière indication n'est pas obligatoire si vous n'avez pas besoin de les récupérer.

Exemple Dossier décomposition d'objet

4- Tester l'existence d'une propriété

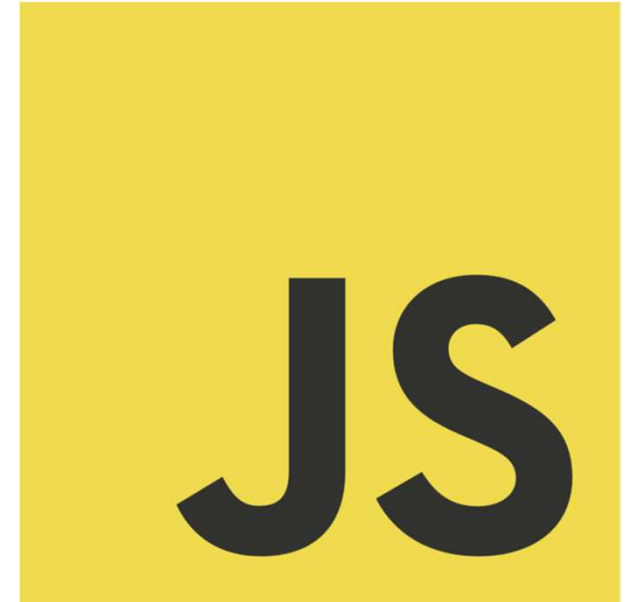


Il y a plusieurs manières de vérifier qu'une propriété existe :

Exemple Dossier existence propriete

4- Tester l'existence d'une propriété

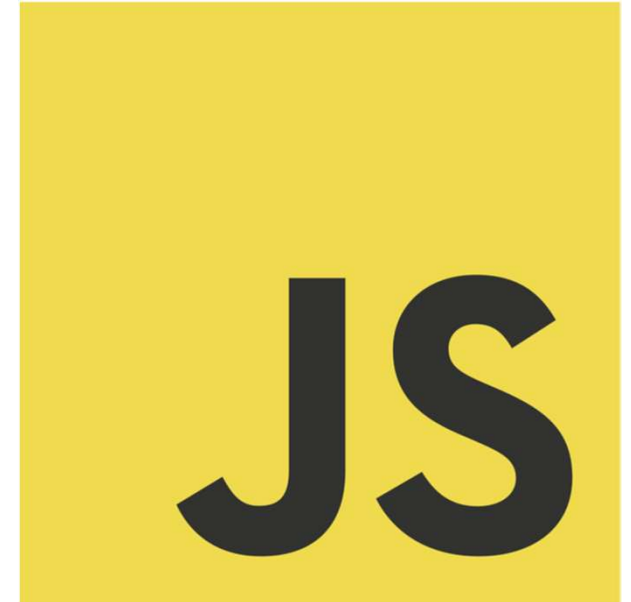
- Exercice :
 - Vérifier que la propriété age existe (afficher le résultat dans la console)
 - Vérifier que la propriété hobbies n'existe pas (afficher le résultat dans la console)
- Durée : 5 min



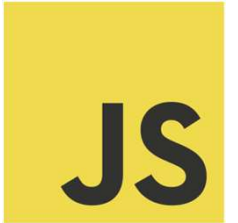
4- Tester l'existence d'une propriété

- Correction

Exercice



5- Supprimer une propriété



Si vous voulez supprimer une propriété d'un objet, il existe un mot clé qui le permet

```
delete objet.prop;
```

Vous supprimerez dans ce cas la propriété ainsi que la valeur associée.

Exemple

Dossier existence propriete

6- Itérer sur un objet



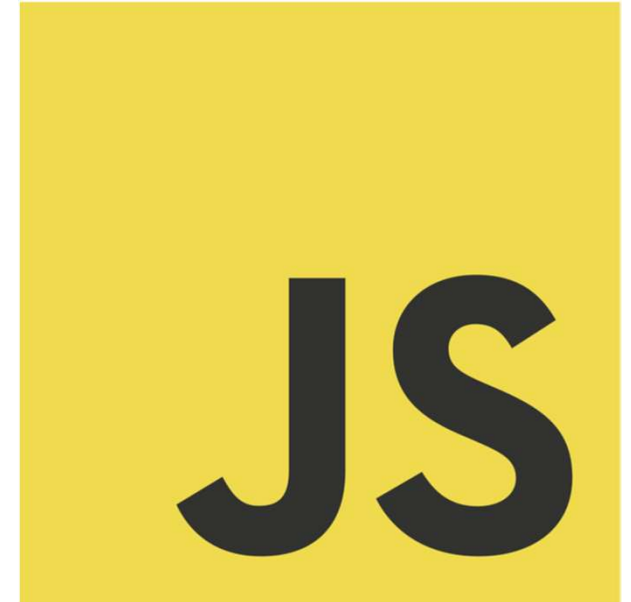
Itérer sur un objet signifie que nous allons passer sur chaque propriété d'un objet, l'une après l'autre. Pour cela, nous allons pouvoir utiliser la syntaxe suivante:

```
for(prop in variable){  
    // code à exécuter  
}
```

Exemple Dossier itérer sur un objet

6- Itérer sur un objet

- Exercice :
 - Itérer sur l'objet `obj` pour afficher dans la console la valeur de ces différentes propriétés
 - L'affichage dans la console doit être de la manière suivante:
 - `nom: Filliot`
 - `prenom: Aymeric`
 - `age: 43`
- Durée : 5 min



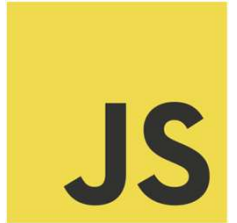
6- Itérer sur un objet

- Correction

Exercice



7- Le format JSON



Le format JSON est le format le plus utilisé pour échanger des informations.

JSON: signifie JavaScriptObjectNotation et sa syntaxe est proche de celle d'un objet Javascript.

JSON n'est pas utilisé qu'en Javascript, vous le trouverez dans toutes les applications Web modernes.

L'objet JSON et ses méthodes

Javascript possède un objet JSON qui permettra facilement de transformer du JSON en objet Javascript et inversement.

- La méthode `JSON.stringify`

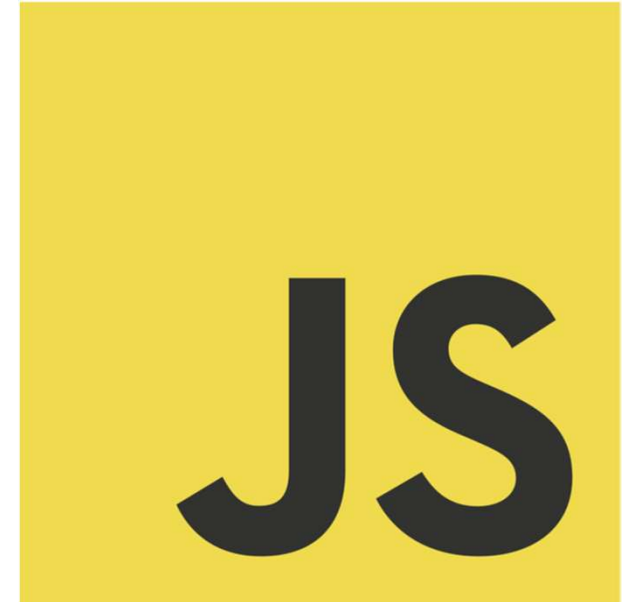
Permet de transformer tout objet ou primitive Javascript au format JSON. Il retourne une chaîne de caractère au format JSON

- La méthode `JSON.parse`

Permet d'analyser du format JSON pour reconstruire un objet Javascript

7- Le format JSON

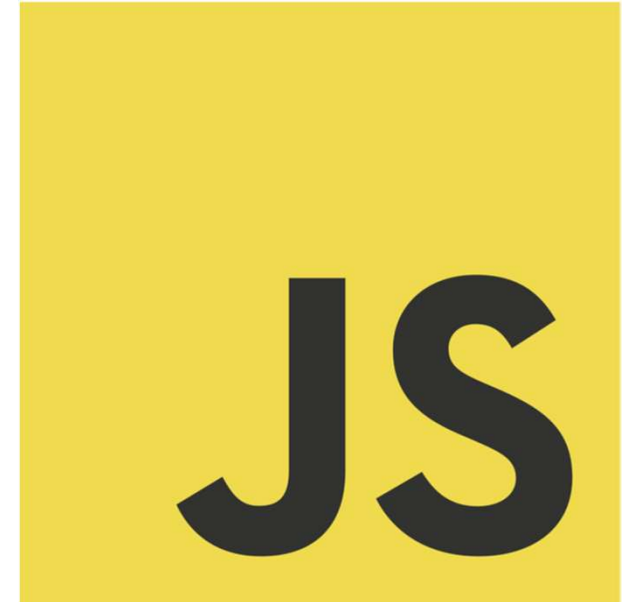
- Exercice :
 - Utiliser l'objet "obj" precedent (avec nom, prénom et âge
 - Créer une variable "monJSON" où sera stocké la transformation de "obj" au format JSON.
 - Créer une variable "obj2" et stocker dedans la transformation de "monJSON" en objet
- Durée : 10 min

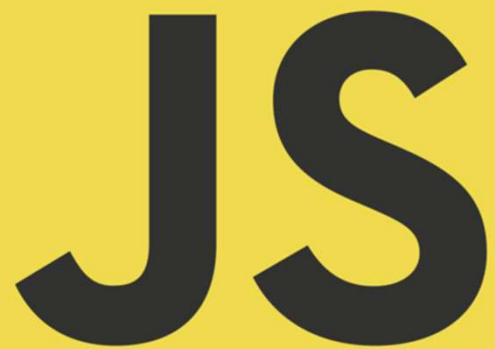


7- Le format JSON

- Correction

Exercice





VIII. Les fonctions



1- Déclarer une fonction

Une fonction est un bloc d'instructions qui peut être exécuté.

Déclaration d'une fonction

```
function nomDeLaFonction(){  
    //Code à exécuter  
}
```

Déclaration d'une fonction sans paramètre

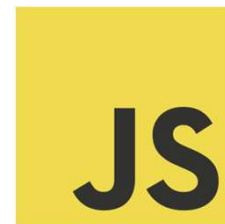
```
function nomDeLaFonction(param1, param2){  
    //Code à exécuter  
}
```

Déclaration d'une fonction avec des paramètres.
Il peut y avoir autant de paramètres que voulus

Lorsque vous déclarez une fonction, celle-ci ne s'exécute pas automatiquement. Pour qu'elle s'exécute, il faut l'appeler.

```
nomDeLaFonction();  
  
nomDeLaFonction(3,5);
```

Exemple Dossier fonction



1- Déclarer une fonction

L'expression de fonction

L'expression de fonction est le fait de déclarer une fonction dans une variable.

```
const maFonction = function(){  
    //Code à exécuter  
}  
  
maFonction();
```

```
let test;  
  
if(condition){  
    test = function(){  
        //code à exécuter  
    }  
}else{  
    test = function(){  
        //code à exécuter  
    }  
}
```

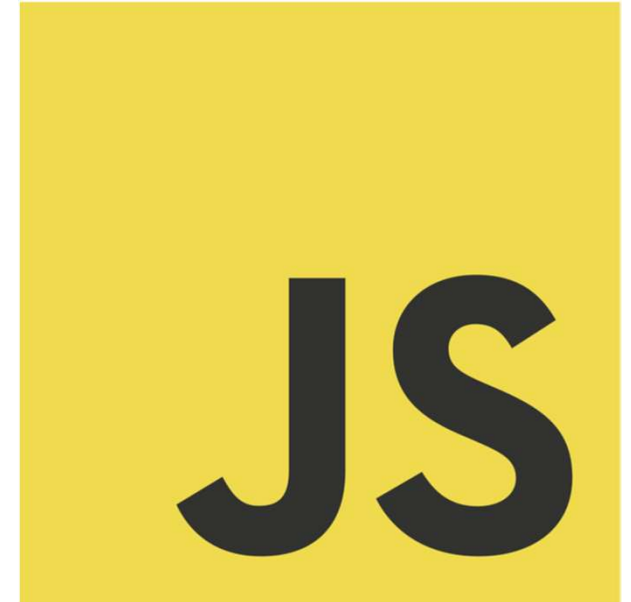
Nous voyons ici que nous ne mettons pas de nom après function, c'est ce qu'on appelle une fonction anonyme.

Exemple

Dossier fonction

1- Déclarer une fonction

- Exercice :
 - Déclarer une fonction “maFonction” sans parametre qui affiche dans la console “Hello” lorsqu’elle est exécutée
 - Exécuter cette fonction
- Durée : 5 min



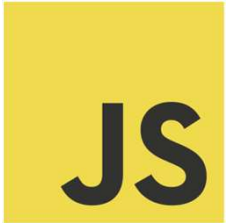
1- Déclarer une fonction

- Correction

Exercice



2- Paramètres



Comme nous l'avons déjà vu, une fonction peut avoir des paramètres

```
function nomDeLaFonction(param1, param2){  
    //Code à exécuter  
}
```

Selon le code qu'il faut exécuter, il est parfois utile de protéger la fonction afin qu'elle ne retourne pas d'erreur.

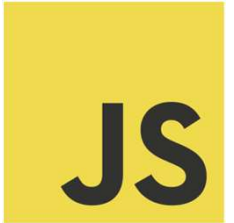
Pour cela, il y a plusieurs façon de la protéger. Nous allons voir 2 d'entre elles:

- La vérification d'un/des paramètres
- La mise en place d'une valeur par défaut

Exemple

Dossier parametre

2- Paramètres



Il est possible également que vous vouliez créer une fonction mais que sur cette fonction vous ne savez pas le nombre d'arguments.

Dans ce cas, nous pouvons utiliser un opérateur qui est l'opérateur rest

```
function maFonction(...rest){  
    //Code à exécuter  
}
```

Comme son nom l'indique celui-ci récupère le reste des arguments et donc la totalité si il est le seul paramètre.

```
function maFonction(param, ...rest){  
    //Code à exécuter  
}
```

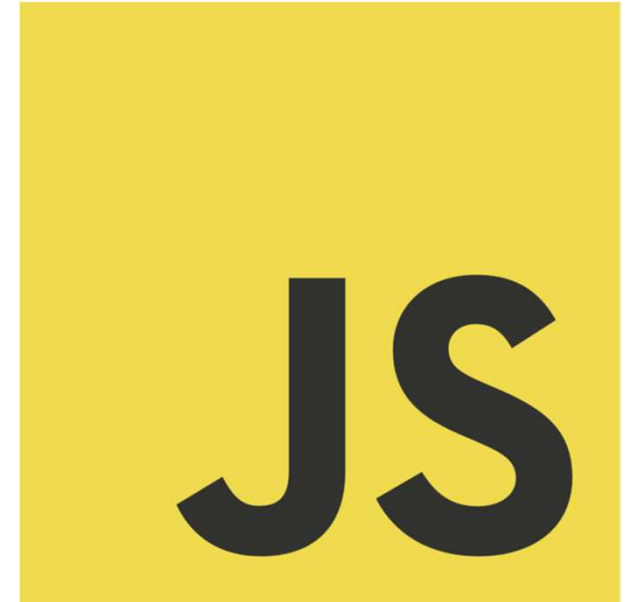
Dans le cas, où nous voulons récupérer également un autre paramètre, il est important que l'opérateur rest soit en dernier.

Exemple

Dossier parametre

2- Paramètres

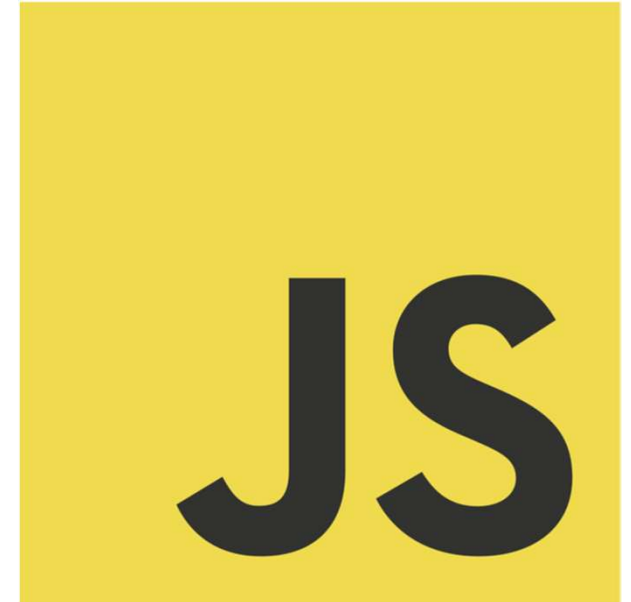
- Exercice :
 - Déclarer une fonction “maFonction” qui permettra d’additionner 2 chiffres mis en paramètre et d’afficher le résultat dans la console.
 - Exécuter cette fonction
 - Déclarer une fonction “maFonction2” qui permettra d’afficher sur la console selon l’âge mis en paramètre si la personne est majeur ou mineur.
 - Exécuter cette fonction
- Durée : 5 min



2- Paramètres

- Correction

Exercice



3- Valeur de retour d'une fonction



Pour le moment, nous avons utilisé à chaque fois la méthode `console.log` pour voir le résultat d'une fonction afin de rapidement voir le résultat.

Cependant, vous ne voudrez pas faire ça normalement mais vous voudrez récupérer un résultat

Pour récupérer un résultat, vous utilisez alors à la fin de la fonction un mot clé `return`

```
function maFonction(){  
    //code à exécuter pour calculer le résultat  
    return result;  
}
```

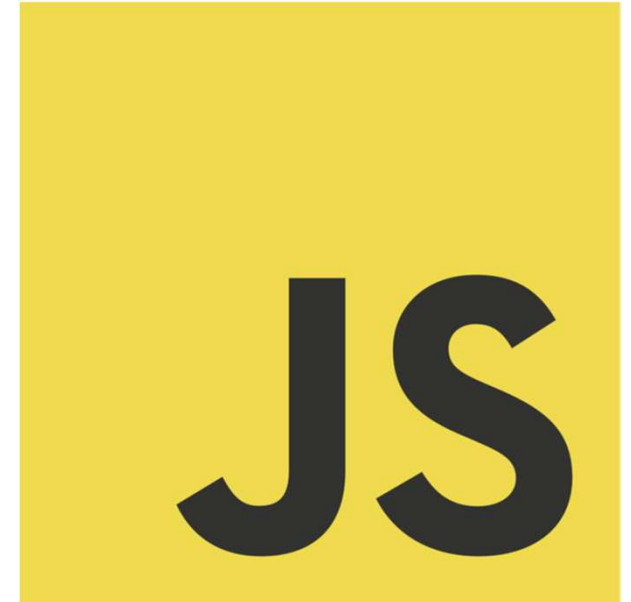
Nous allons donc dans ce cas exécuter des instructions pour retourner un résultat.

Attention, nous ne pouvons retourner qu'une seule valeur avec une fonction. Par contre, cette valeur peut-être un tableau, un objet...

Exemple Dossier return

3- Valeur de retour d'une fonction

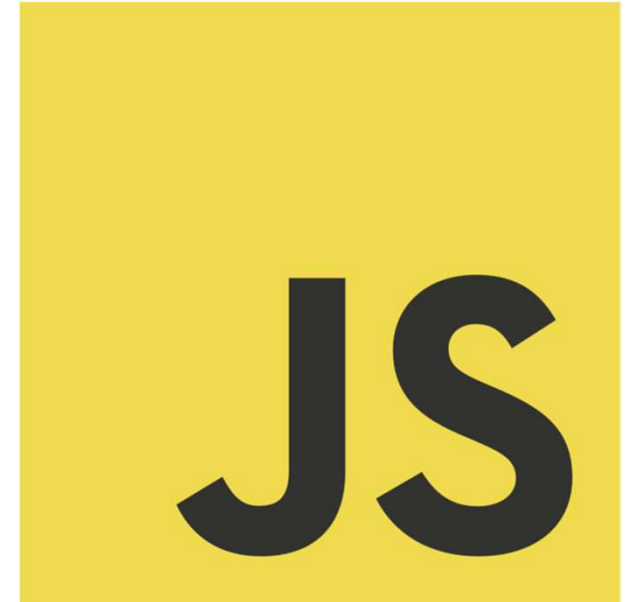
- Exercice :
 - Déclarer une fonction “maFonction” qui permettra de retourner la valeur “Je suis mineur” ou “Je suis majeur”
 - Exécuter cette fonction et stocker son résultat dans une variable “result”
- Durée : 10 min



3- Valeur de retour d'une fonction

- Correction

Exercice



4- Contexte global et contexte d'exécution



Lorsqu'un fichier Javascript est lu, le moteur crée automatiquement un contexte global contenant lorsqu'il s'agit d'un navigateur un Objet Window.

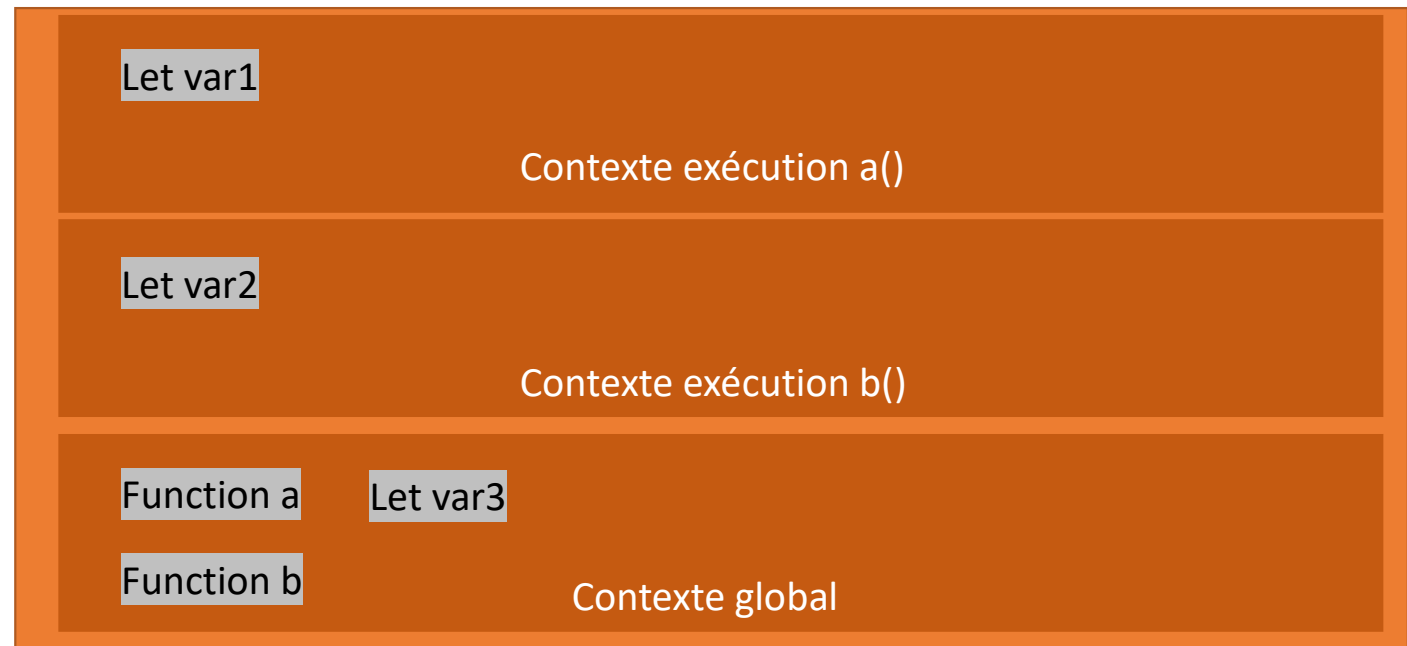
Il crée ensuite pour chaque fonction un contexte d'exécution qu'il mets dans la pile d'exécution.

Stack ou pile d'execution

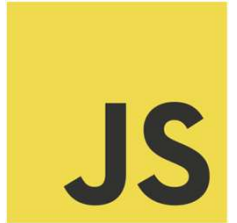
```
function a() {  
    let var1;  
}
```

```
function b() {  
    let var2  
    a();  
}
```

```
b();  
let var3;
```



4- Contexte global et contexte d'exécution

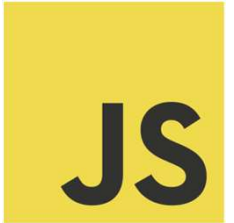


Le code est ensuite exécuté dans l'ordre de la pile.

C'est-à-dire ici

1. Le contexte d'exécution a() puis il le supprime
2. Le contexte d'exécution b() puis il le supprime
3. Puis le Contexte global

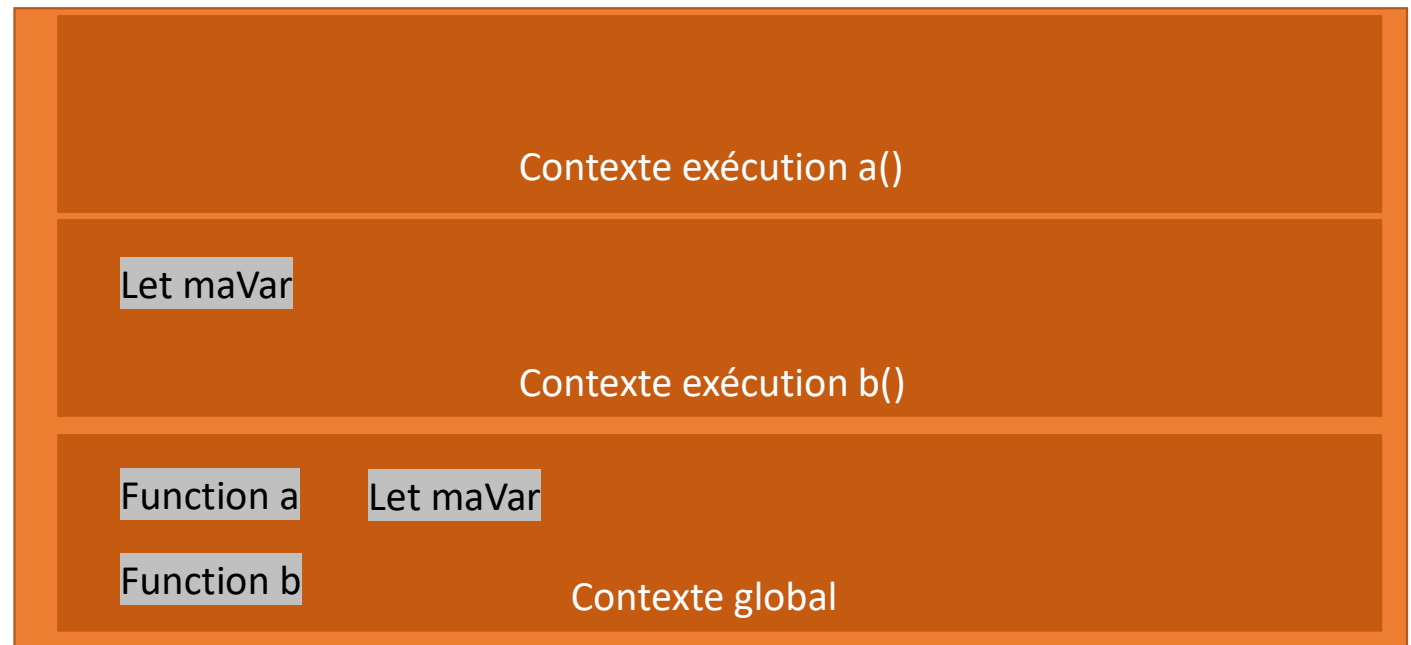
4- Contexte global et contexte d'exécution



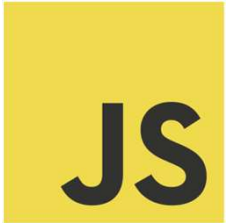
Mais que se passe-t-il dans le cas suivant, quelle valeur retournera a, 1 ou 2 :

```
function a() {  
    console.log(maVar)  
;  
}  
  
function b() {  
    let maVar = 1;  
    a();  
}  
let maVar = 2;  
  
b();
```

Stack ou pile d'exécution



4- Contexte global et contexte d'exécution



Dans ce cas, l'exécution de la fonction a se fera de la manière suivante:

1. Si la variable maVar est déclaré dans a alors il utilisera celle-ci (ici il n'y en a pas)
 2. Dans le cas contraire, il vérifiera si une variable maVar est déclaré dans le contexte où la fonction a a été déclaré.
- Ici la fonction a a été déclaré dans le contexte global.

Il y a bien une variable maVar dans le contexte global, il utilise donc cette variable dont la valeur est 2

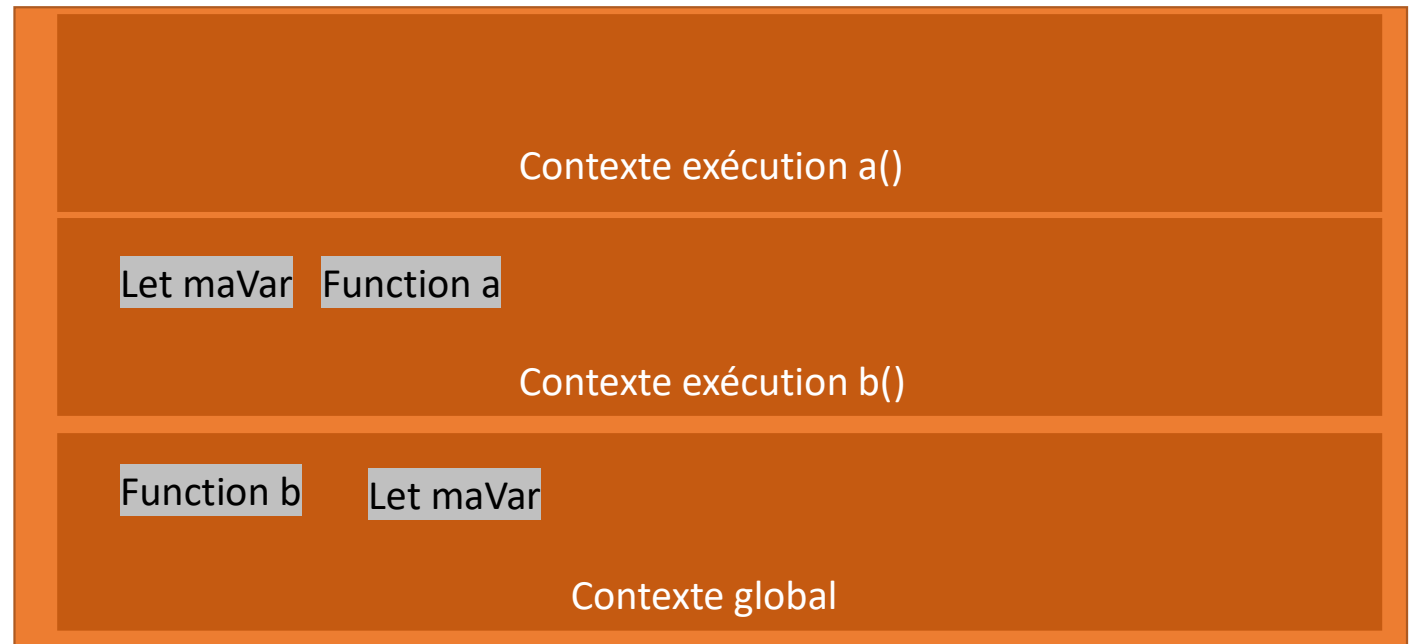
4- Contexte global et contexte d'exécution



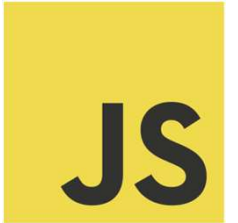
Autre exemple, que se passe-t-il dans le cas suivant, quelle valeur retournera a, 1 ou 2 :

```
function b() {  
  function a() {  
    console.log(maVar);  
  }  
  let maVar = 1;  
  a();  
}  
let maVar = 2;  
  
b();
```

Stack ou pile d'exécution



4- Contexte global et contexte d'exécution

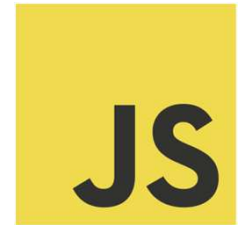


Dans ce cas, l'exécution de la fonction a se fera de la manière suivante:

1. Si la variable maVar est déclaré dans a alors il utilisera celle-ci (ici il n'y en a pas)
 2. Dans le cas contraire, il vérifiera si une variable maVar est déclaré dans le contexte où la fonction a a été déclaré.
- Ici la fonction a a été déclaré dans le contexte d'exécution de b.
Il y a bien une variable maVar dans le contexte d'exécution de b, il utilise donc cette variable dont la valeur est 1

Exemple Dossier return

5- Le mot clé this et le mode strict



`this` est un mot clé Javascript permettant d'accéder à l'objet représentant le contexte d'exécution et donc de l'endroit où il est appelé.

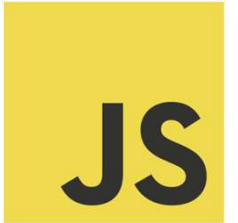
Le `mode strict` de Javascript est une amélioration recommandée du langage. Il permet notamment d'éliminer des erreurs silencieuses de Javascript, c'est-à-dire des erreurs qui ne stoppent pas l'exécution du code.

Webpack utilise le mode strict par défaut.

`this` aura un comportement différent selon que le code est en mode strict ou pas:

- Dans le contexte global, il n'y a pas de différence
- Dans un appel simple d'une fonction, en mode normal `this` représentera le contexte global alors que dans le mode strict il retournera `undefined`. Cela permet d'éviter des failles de sécurité lors de l'utilisation de bibliothèques tierces en leur interdisant l'accès à l'objet global. (risque d'injection de code malveillant ou de récupération d'informations)
- Lorsque la fonction invoquée est une méthode d'un objet, `this` aura pour valeur la référence de l'objet, en mode strict et en mode normal
- ...

6- Définir ou lier this



Dans certains cas, vous souhaitez modifier le comportement par défaut de `this`, c'est à cela que va servir les fonctions que nous allons voir ici.

Il est en effet possible de définir la valeur de `this` lorsqu'on évoque une fonction. Il suffit pour cela d'utiliser les méthodes `call()` ou `apply()` et de passer en premier argument la valeur de `this`

Une autre méthode existe également, la méthode `bind()`.

La méthode `bind()` permet de créer un clone d'une fonction en liant définitivement la valeur de `this` à l'argument passé en premier paramètre.

Exemple

Dossier définir et lier this

7- Les fonctions fléchées



Il existe une autre manière d'écrire les fonctions, il s'agit des fonctions fléchées

```
const a = () => {  
  //instructions  
}
```

L'une des particularités des fonctions fléchées est qu'elle ne possède pas leur propre `this`

Les fonctions fléchées seront privilégiées dans la plupart des cas, à l'exception des cas suivants:

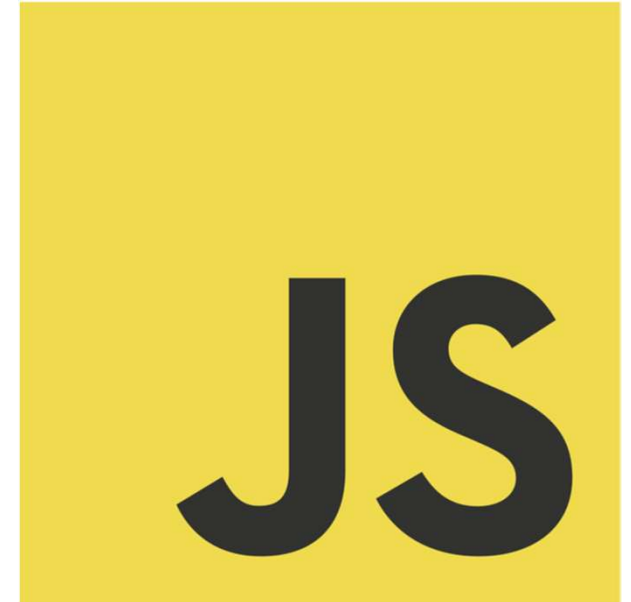
- Déclaration d'une méthode dans un objet (comme les fonctions fléchées n'ont pas leur propre `this`, si on appelle `this` dans celle-ci, `this` correspondra au `this` de l'objet)
- Les constructeurs

Exemple

Dossier fonctions flechees

7- Les fonctions fléchées

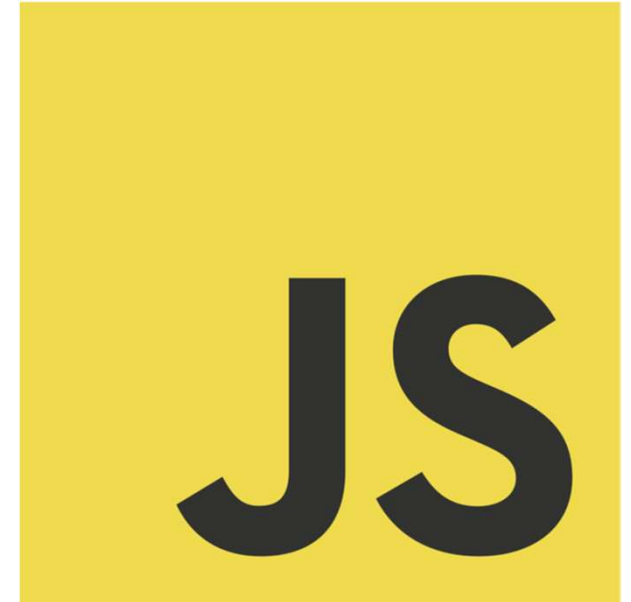
- Exercice :
 - Déclarer une fonction fléchée “maFonction” qui retournera le résultat d’une multiplication de 2 chiffres
 - Exécuter cette fonction et stocker son résultat dans une variable “result”
- Durée : 5 min



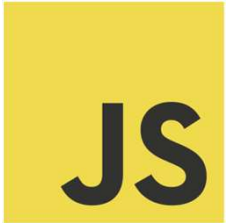
3- Les fonctions fléchées

- Correction

Exercice



8- Fonctions de rappel (callback)

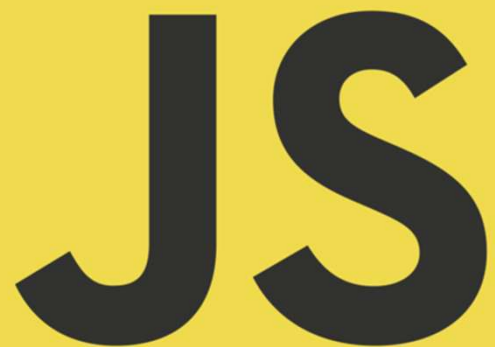


Une fonction de rappel ou callback est une fonction qui est passée en argument d'une autre fonction et qui est exécutée dans cette dernière.

C'est quelque chose qui est très utilisé en Javascript, notamment pour gérer les évènements (exemple: déclencher une action lorsqu'un utilisateur clique sur un bouton)

Exemple

Dossier callback



IX. Les tableaux

1- Introduction aux tableaux



Un tableau peut-être déclaré de 3 manières différentes :

```
const arr = [2, "deux", { lastname: "Dupont" }, [1, 2, 3]];
const arr2 = Array(1, 2, 3);
const arr3 = new Array(4, 5, 6);
```

Les 3 manières de déclarer un tableau sont équivalentes.

On peut mettre à l'intérieur d'un tableau tous ce que l'on veut (données primitives et données de type objet).

La propriété length

Cette propriété permet de récupérer le nombre d'élément présent dans le tableau.

Exemple

Dossier tableau

1- Introduction aux tableaux



La méthode from de l'objet Array

Cette méthode permet de retourner un tableau à partir d'une variable itérable (que l'on peut parcourir).

La méthode isArray de l'objet Array

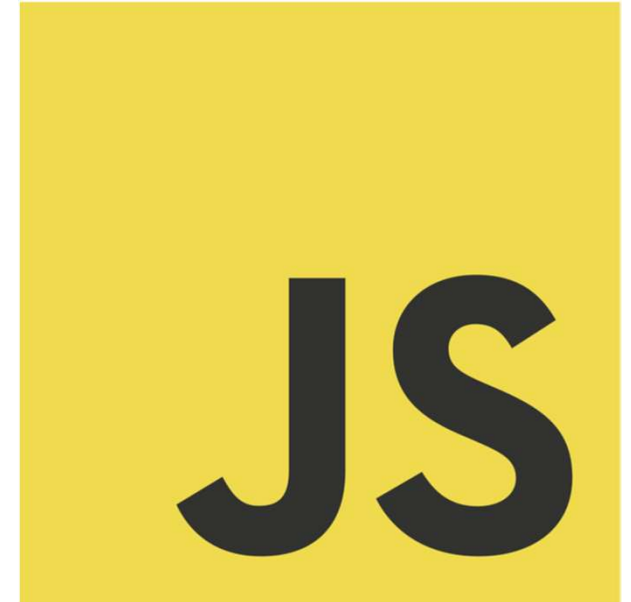
Permet de vérifier que la variable mise en argument est un tableau. Retourne true si c'est le cas.

Exemple

Dossier tableau

1- Introduction aux tableaux

- Exercice :
 - Déclarer un tableau “tab” contenant les valeurs 1,2,3.
 - Retourner le nombre d’élément dans une variable “result”
- Durée : 5 min



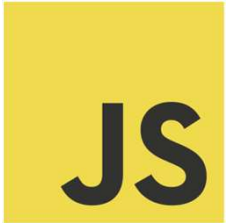
1- Introduction aux tableaux

- Correction

Exercice



2- Accéder aux éléments d'un tableau



Chaque élément d'un tableau possède un index. Le 1^{er} élément de ce tableau ayant l'index 0. Cet index va nous permettre de récupérer facilement la valeur d'un élément et la modifier si nécessaire.

`arr[0];` Permet ici de récupérer l'élément dont l'index est 0 du tableau arr

`arr[0] = "Billy";` Permet ici de remplacer la valeur de l'élément ayant l'index 0 par la valeur « Billy »

La propriété `length` permettra ainsi de récupérer le dernier élément du tableau de la manière suivante:

`arr[arr.length - 1];`

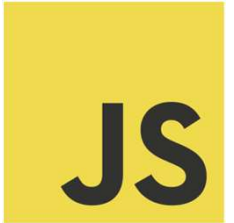
On peut également utiliser la propriété `length` pour réinitialiser un tableau

`arr.length = 0;`

Exemple

Dossier tableau

2- Accéder aux éléments d'un tableau



La propriété `length` va permettre également d'itérer sur un tableau à l'aide d'une boucle.

```
for(let i = 0; i < arr.length; i++){  
    // on itere sur chaque indexen utilisant  
    // arr[i]  
}
```

Un tableau peut contenir un autre ou plusieurs autres tableaux, c'est ce qu'on appelle des tableaux imbriqués. Pour récupérer les valeurs des éléments, c'est très simple il suffit d'indiquer les index successifs.

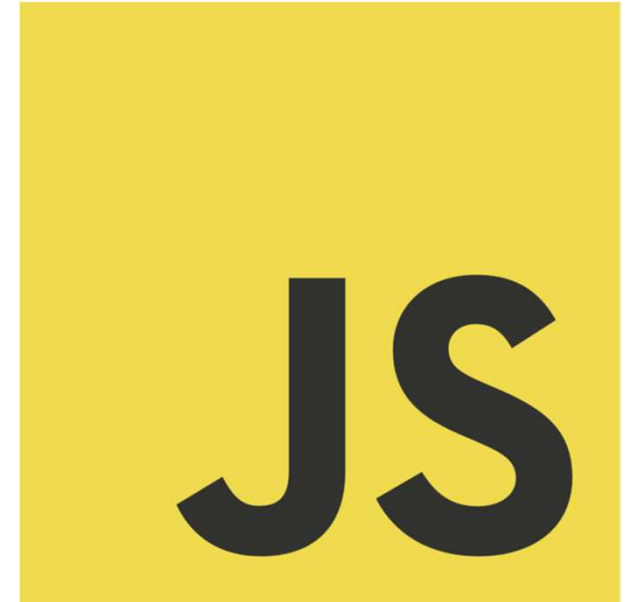
```
arr[2][1];
```

Exemple

Dossier tableau

2- Accéder aux éléments d'un tableau

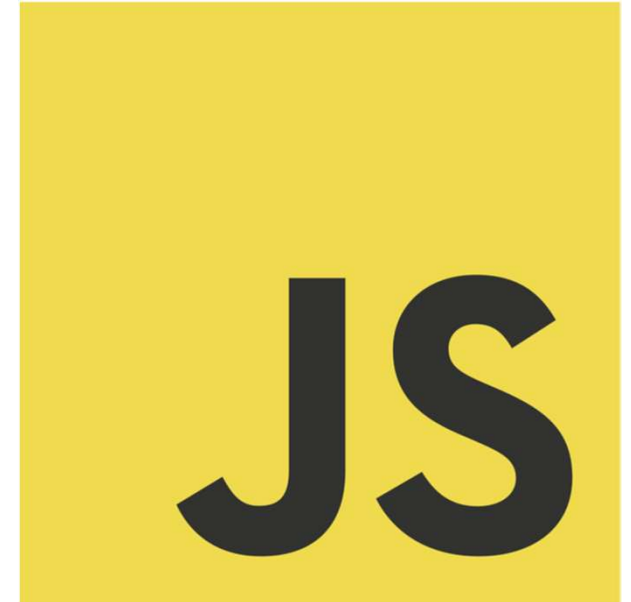
- Exercice :
 - Déclarer un tableau "tab" contenant les valeurs "Jean", "Robert", "Marcel".
 - Récupérer dans une variable "result" la valeur "Jean" dans le tableau à l'aide de son index.
- Durée : 5 min



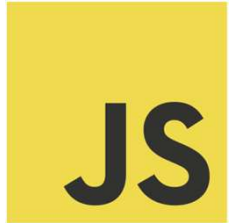
2- Accéder aux éléments d'un tableau

- Correction

Exercice



3- Décomposition de tableau et opérateur ...rest



Il est possible de décomposer un tableau et récupérer les valeurs dans des variables

Exemple

Dossier décomposition tableau

4- Ajouter des éléments à un tableau



Il y a plusieurs manières qui permettent d'ajouter un élément à un tableau

La méthode push

```
const arr = [];  
arr.push("a", "b", "c");
```

Cette méthode permet d'ajouter un ou plusieurs éléments à la fin du tableau

La méthode unshift

```
arr.unshift("a", "b", "c");
```

Cette méthode permet d'ajouter un ou plusieurs éléments au début du tableau

Exemple

Dossier décomposition tableau

4- Ajouter des éléments à un tableau



La méthode splice

Cette méthode permet de modifier un tableau. Elle peut à la fois ajouter et supprimer des éléments d'un tableau. Voici sa syntaxe:

```
const tableau = [1, 2, 3];  
tableau.splice(index_de_depart, element_a_supprimer, element_aajouter);
```

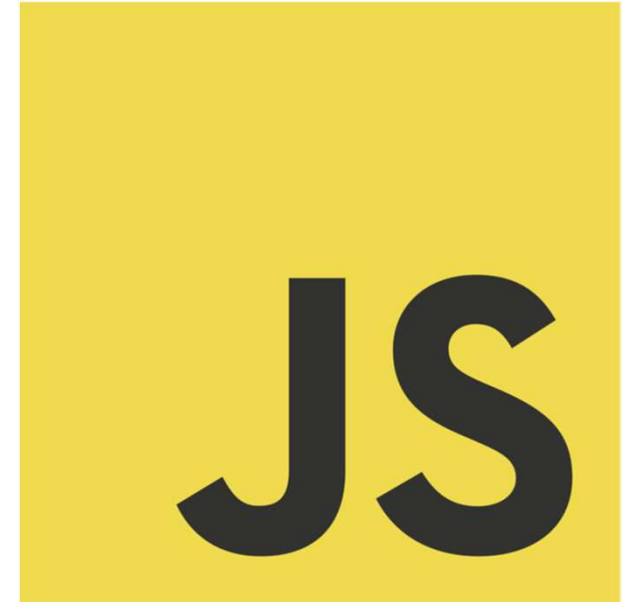
Cette méthode retourne un tableau des éléments supprimer

Exemple

Dossier décomposition tableau

4- Ajouter des éléments à un tableau

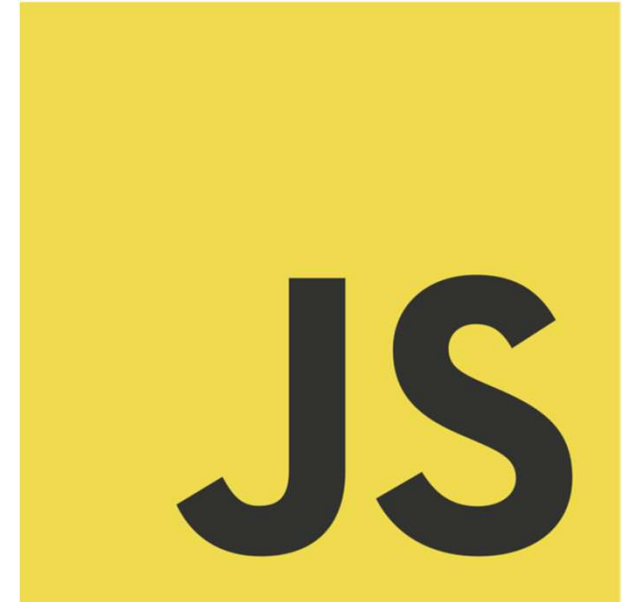
- Exercice :
 - Déclarer un tableau "tab" contenant les valeurs "Jean", "Robert", "Marcel".
 - Ajouter à la fin du tableau la valeur "René"
 - Ajouter au début du tableau la valeur "Gislain"
- Durée : 10 min



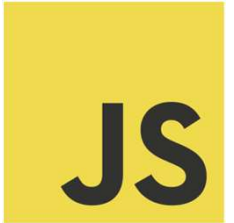
4- Ajouter des éléments à un tableau

- Correction

Exercice



5- Supprimer des éléments d'un tableau



La méthode shift

Cette méthode permet de supprimer le 1^{er} élément d'un tableau et de le retourner

La méthode pop

Cette méthode permet de supprimer le dernier élément d'un tableau et de le retourner

La méthode splice

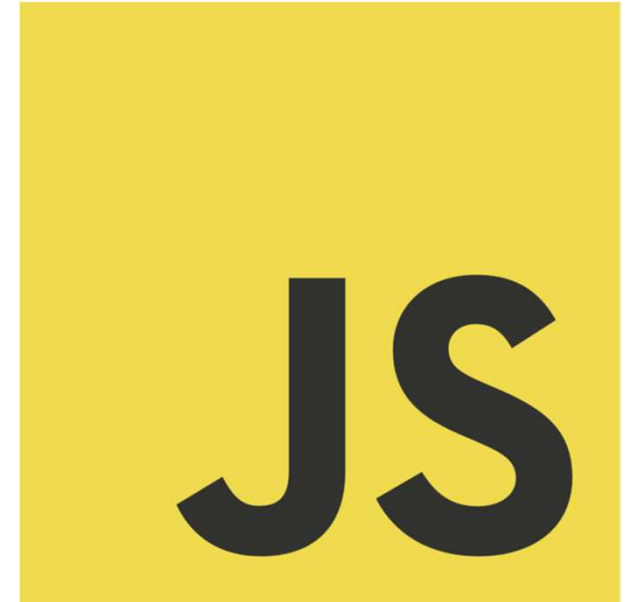
Cette méthode permet de se placer à un endroit du tableau en utilisant l'index et d'indiquer le nombre de valeur à supprimer à partir de cet index. Cette méthode retourne également les éléments supprimés dans un tableau.

Exemple

Dossier supprimer élément tableau

5- Supprimer des éléments d'un tableau

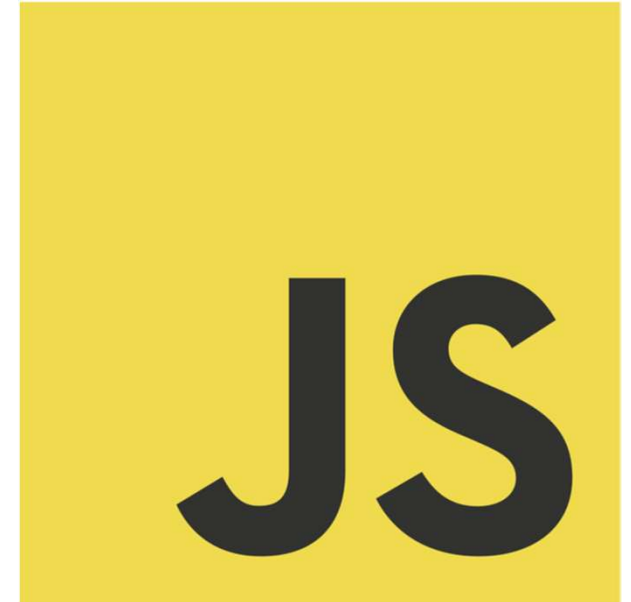
- Exercice :
 - A partir du tableau de la leçon précédente
 - Supprimer le 1er élément du tableau
 - Supprimer le dernier élément du tableau
 - Remplacer "Robert" par "Gilles" et "Yohann"
- Durée : 5 min



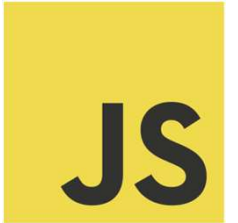
5- Supprimer des éléments d'un tableau

- Correction

Exercice



6- Trouver des éléments dans un tableau



La méthode indexOf

Cette méthode retourne l'index de l'élément mis en argument si celui-ci est trouvé dans le tableau. Sinon, la méthode retourne -1.

Si plusieurs éléments ont la même valeur, il retournera le premier du tableau.

La méthode lastIndexOf

Cette méthode retourne l'index de l'élément mis en argument si celui-ci est trouvé dans le tableau. Sinon, la méthode retourne -1.

Si plusieurs éléments ont la même valeur, il retournera le dernier du tableau.

Exemple

Dossier supprimer élément tableau

6- Trouver des éléments dans un tableau



La méthode includes

Cette méthode retourne true ou false si l'élément mis en argument a été trouvé dans le tableau

La méthode findIndex

Cette méthode retourne l'index d'un élément selon une fonction callback mise en argument. Utile pour retrouver des objets stockés dans le tableau. Si plusieurs éléments peuvent correspondre, il retourne le 1er

La méthode find

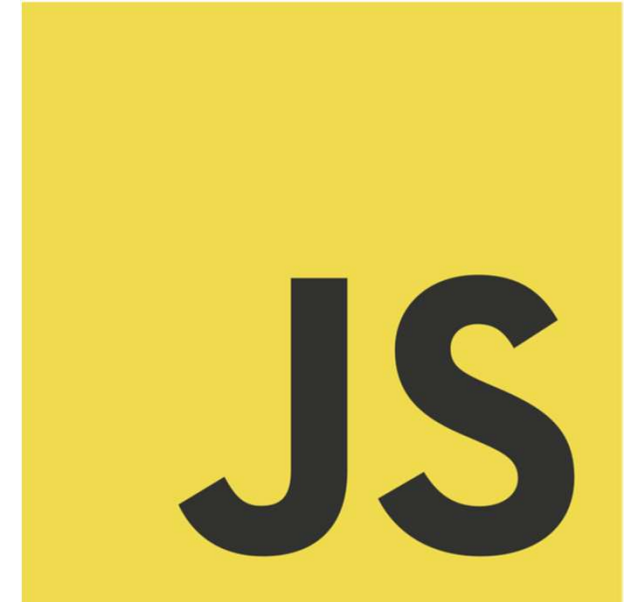
Cette méthode retourne la valeur d'un élément selon une fonction callback mise en argument. Utile pour retrouver des objets stockés dans le tableau. Si plusieurs éléments peuvent correspondre, il retourne le 1er

Exemple

Dossier supprimer élément tableau

6- Trouver des éléments dans un tableau

- Exercice :
 - Déclarer un tableau “tab” avec les valeurs Jean, Robert et Marcel
 - Créer une fonction qui permettra de vérifier que l’élément mis en argument existe dans un tableau mis en argument et afficher le résultat dans la console
- Durée : 10 min



15
0

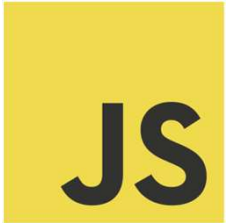
6- Trouver des éléments dans un tableau

- Correction

Exercice



7- Copier un tableau



Un tableau en Javascript est un objet.

Donc si nous le copions en faisant `a = b` par exemple, nous le copions en référence. Ainsi si on modifie `b`, `a` sera modifié également et inversement.

Les copies superficielles

Il y a plusieurs manières de copier un tableau de manière superficielle:

- Avec `[...arr]`
- Avec la méthode `slice()`
- Avec la méthode `Array.from()`

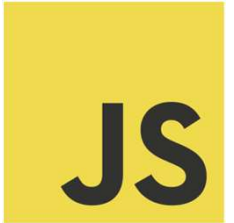
Les copies profondes

Pour réaliser une copie profonde et donc copier tout en valeur et non en référence, nous allons être obligé d'utiliser une transformation JSON.

Malheureusement, cette méthode pose problème s'il y a par exemple des fonctions dans le tableau d'origine.

Exemple Dossier `copier_un_tableau`

8- Fusionner des tableaux



Pour fusionner 2 tableaux, nous pouvons utiliser 2 méthodes différentes

La méthode concat

Cette méthode retourne un nouveau tableau.

Syntaxe:

```
const c = a.concat(b);
```

 a et b étant 2 tableaux. La méthode concat retournera un tableau avec les éléments du tableau a puis les éléments du tableau b

Avec [...]

Exemple

Dossier fusionner des tableaux

9- Trier un tableau



Pour trier un tableau nous allons utiliser une méthode, la méthode sort

La méthode sort

Attention, la méthode sort sans argument va trier selon l'ordre des caractères ASCII, ce qui est généralement pas ce que l'on veut.

Mais nous pouvons, lui mettre en argument une fonction callback qui permettra de trier correctement.

Le résultat de cette fonction callback permettra de trier, s'il est négative le 1^{er} élément est avant le 2^{ème} si positif c'est l'inverse.

La méthode localeCompare sur une chaîne de caractère

Cette méthode permettra, couplée à la méthode sort, de trier des chaînes de caractères

La méthode reverse

Cette méthode permet d'inverser un tableau.

Exemple

Dossier fusionner des tableaux

9- Trier un tableau

- Exercice :
 - Déclarer un tableau “tab” avec les valeurs Jean, robert et Marcel
 - Trier le tableau (attention à robert avec un r minuscule)
- Durée : 5 min



9- Trier un tableau

- Correction

Exercice

A yellow square containing the letters 'JS' in a large, bold, black sans-serif font.

10- Itérer sur un tableau



Pour itérer sur tableau, il y a plusieurs manières de faire.

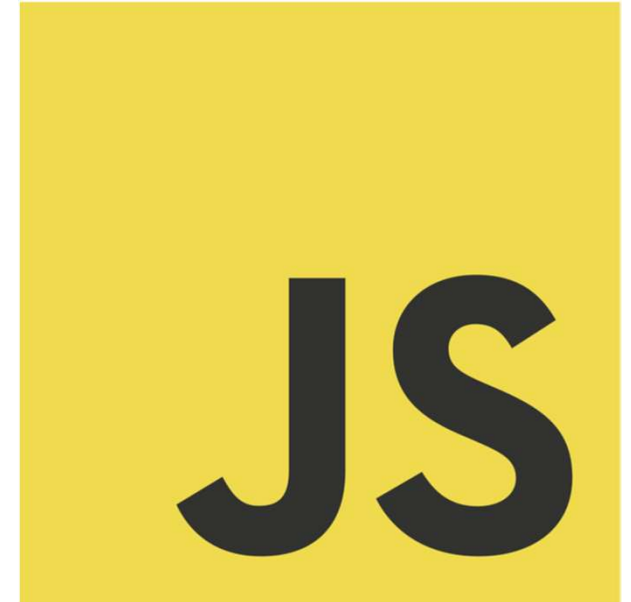
- Utilisation d'une boucle for
- Utilisation de la méthode forEach (utile si nous avons besoin de récupérer l'index)
- Utilisation de la boucle for ... of (utile si nous n'avons pas besoin de récupérer l'index)

Exemple

Dossier iterer_sur_un_tableau

10- Itérer sur un tableau

- Exercice :
 - Déclarer un tableau “tab” avec les valeurs 1,2,3
 - Itérer sur le tableau en multipliant la valeur par 2 puis afficher cette valeur sur la console
- Durée : 5 min



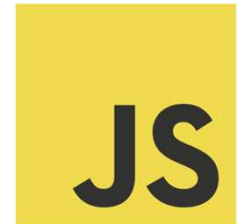
10- Itérer sur un tableau

- Correction

Exercice



11- Introduction à la programmation fonctionnelle



La programmation fonctionnelle est un paradigme de programmation qui consiste à concevoir ses programmes comme un ensemble de fonctions mathématiques que l'on compose entre elles.

Les fonctions sont des objets de première classe en JavaScript, elles peuvent donc être passées en argument et donc être composées. C'est notamment le principe des fonctions de rappel ou callback

La méthode map

La méthode map retourne un tableau avec le résultat de l'exécution d'une fonction callback sur chacun des éléments du tableau d'origine

La méthode filter

La méthode filter retourne un tableau contenant les éléments qui passe la fonction de rappel

Exemple

Dossier `iterer_sur_un_tableau`

11- Introduction à la programmation fonctionnelle



La méthode reduce

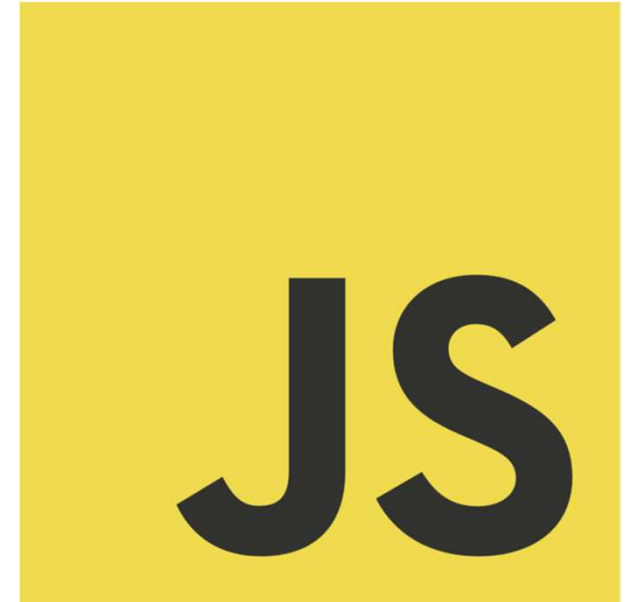
Cette méthode permet de réduire un tableau à une seule valeur

Exemple

Dossier `iterer_sur_un_tableau`

11- Introduction à la programmation fonctionnelle

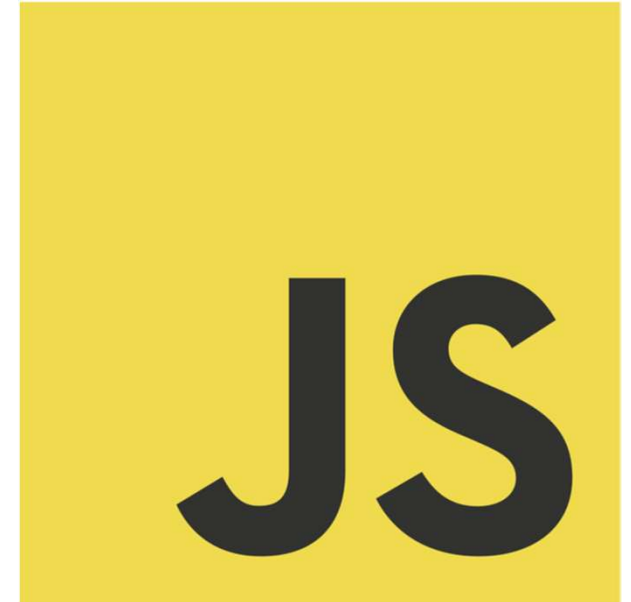
- Exercice :
 - Déclarer un tableau "tab" avec les valeurs 1,2 et 3
 - Créer à partir du tableau tab un nouveau tableau avec des valeurs multipliées par 10
 - Déclarer un tableau "tab2" avec les valeurs suivantes : 5,56,32,46,4,15
 - Créer un tableau à partir du tableau "tab2" avec uniquement les valeurs >à20
- Durée : 10 min

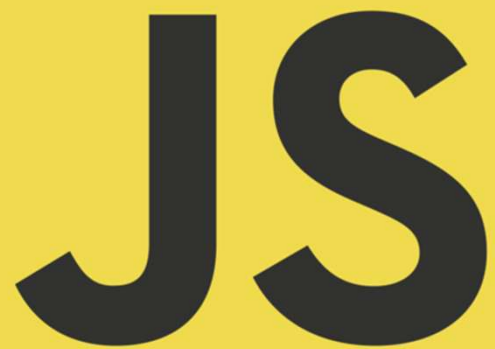


11- Introduction à la programmation fonctionnelle

- Correction

Exercice





X. Les modules

1- Exporter les modules



Lorsque notre application devient plus importante, nous voulons séparer notre application en plusieurs fichiers qui sont appelés des modules.

Pour cela, il faut indiquer dans le fichier js, ce que vous voulez exporter à l'aide du mot clé export. Exporter signifie que vous rendez disponible la fonction ou la variable pour être importée dans un ou plusieurs modules (donc fichiers).

```
function maFonction() {  
    // Instructions;  
}  
export { maFonction };
```

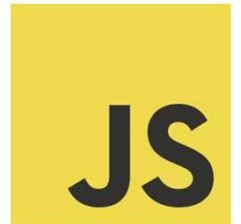
```
const a = "Ma variable a";  
export { a };
```

Vous pouvez également importer les 2 éléments en même temps de la manière suivante

```
import { maFonction, a } from 'module';
```

Exemple Dossier les_modules

1- Exporter les modules



Il est également possible lors de l'export de renommer ce que l'on veut exporter

```
const b = "Ma variable b";  
export { b as varB };
```

Dans ce cas la variable b lors de l'export s'appellera varB et il faudra donc l'importer sous ce nouveau nom

Il est également possible d'exporter un élément par fichier par défaut.

Il est tout de fois déconseiller d'utiliser l'export par défaut

```
function maFonction() {  
    // Instructions  
}  
export default maFonction;
```

Exemple Dossier les_modules

2- Importer les modules



Une fois que des éléments d'un fichier ont été exporté, on peut les importer.
Pour cela, nous allons utiliser la syntaxe suivante

```
import { a } from './monfichier.js';
```

S'il y a plusieurs éléments à importer à partir d'un même fichier, nous pouvons les importer de la manière suivante:

```
import { a, b } from './monfichier.js';
```

Il est également possible d'importer la totalité des éléments exportés d'un module de la manière suivante:

```
import * as monModule from 'chemin-vers-module';
```

Exemple Dossier les_modules

2- Importer les modules

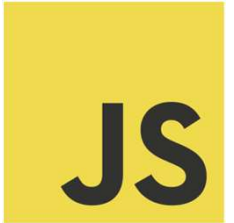


Il est recommandé de ne pas utiliser les imports de tout un module avec *.Premièrement, parce qu'en faisant de la sorte vous perdez l'autocomplétion automatique de l'IDE.

Deuxièmement, car en important explicitement des fichiers, Webpack pourra effectuer des optimisations : notamment du tree-shaking en ne mettant dans le bundle que le code qui est vraiment utilisé. Tout ce qui n'est pas explicitement importé sera écarté.

Exemple Dossier les_modules

3- Réexporter et les imports dynamiques



Il est possible d'importer et de réexporter directement des éléments de la manière suivante :

```
export { monExport } from 'chemin-du-module';
```

Cela permet lorsque votre application commence à être volumineuse de pouvoir mettre en place une architecture qui permettra d'y voir plus clair.

Voici un exemple d'architecture

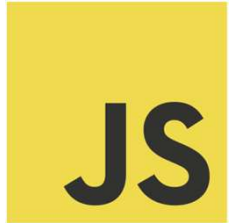
Exemple Dossier architecture

A large yellow square containing the letters 'JS' in a bold, dark grey, sans-serif font.

JS

XI. Le DOM

1- Introduction



Qu'est-ce qu'une API (Application Programming Interface) ?

C'est un ensemble d'éléments (fonctions, objets, méthodes) qu'un programme met à disposition d'un autre.

Qu'est-ce qu'une Web API ?

Ce sont des API disponibles par exemple dans les navigateurs et directement accessibles via Javascript.

Qu'est-ce que l'objet Window?

Window est l'objet global dans un navigateur. Il représente la fenêtre de l'onglet utilisé par votre application.

Il y a donc un objet window par onglet ouvert.

Window a de nombreuses propriétés dont le document qui représente le DOM.

Qu'est-ce que le DOM (Document Object Model)?

Le DOM est une API qui permet d'interagir avec des documents et notamment le HTML.



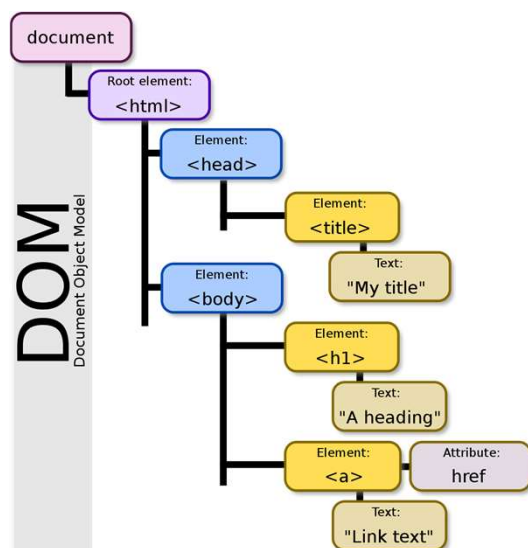
1- Introduction

Qu'est-ce que le DOM (Document Object Model)?

Le DOM est une API qui permet d'interagir avec des documents et notamment le HTML.

Le DOM est un arbre de nœud qui représente le HTML.

C'est le DOM qui nous permettra de manipuler les éléments HTML(les modifier, les supprimer, en ajouter,...), de manipuler le CSS, d'ajouter des évènements (déclenchement d'action selon le comportement de la souris par exemple, du scroll,...)



2- Sélectionner des éléments du DOM



Pour sélectionner les éléments du DOM, il y a plusieurs méthodes, nous allons en voir quelques unes.

Les méthodes getElement(s)

`document.getElementById()` => permet de sélectionner l'élément dont l'id est celui indiqué en argument

`document.getElementsByName()` => permet de sélectionner les éléments ayant un attribut name dont la valeur est celle mise en argument

`element.getElementsByClassName()` => permet de sélectionner les éléments descendants qui possèdent la classe indiquée en argument

`element.getElementsByTagName()` => permet de sélectionner les éléments descendants correspondant à la balise passé en argument

Remarquez que si les deux premières méthodes s'utilisent uniquement sur le document, les deux autres peuvent s'utiliser sur tout élément du DOM.

2- Sélectionner des éléments du DOM



Les méthodes querySelector

Ces méthodes ont la particularité de permettre de sélectionner des éléments comme nous le faisons avec du CSS

`element.querySelector()` => permet de sélectionner le 1^{er} élément dont le sélecteur CSS mis en argument correspond.

`element.querySelectorAll()` => permet de sélectionner tous les éléments dont le sélecteur CSS mis en argument correspond.

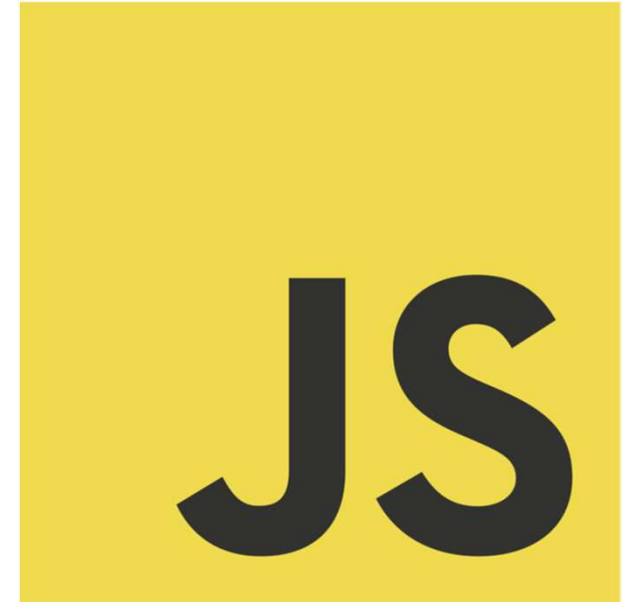
Les méthodes `querySelector` retournent des collections statiques et les méthodes `getElementsBy` des collections live.

La différence est que les collections live se mettent à jour automatiquement lorsqu'il y a des changements dans le document, alors que les collections statiques non.

Exemple Dossier `selection_dom`

2- Sélectionner des éléments du DOM

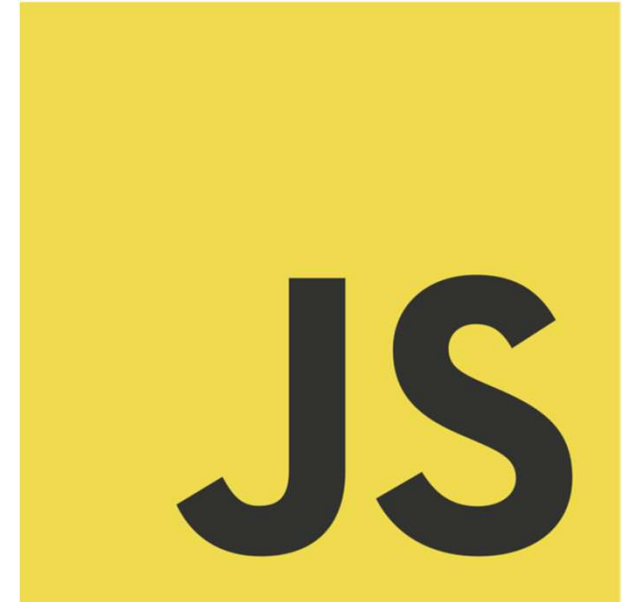
- Exercice :
 - Créer un fichier HTML avec à l'intérieur les éléments suivants :
 - Une section avec l'id maSection
 - Un paragraphe avec la classe mesPar
 - Sélectionner la section à l'aide de son ID avec une méthode get
 - Sélectionner le paragraphe à l'aide de sa classe mesPar avec une méthode get
 - Faire la même chose en utilisant querySelector
- Durée : 10 min



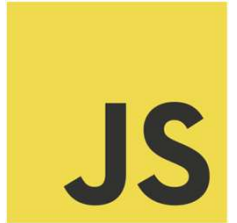
2- Sélectionner des éléments du DOM

- Correction

Exercice



3- Modifier des éléments du DOM



Les éléments du DOM peuvent être modifiés à l'aide de Javascript une fois qu'ils sont sélectionnés. Voici quelques exemples de propriétés que nous allons pouvoir utiliser.

La propriété innerHTML

Cette propriété permet de modifier le contenu HTML des éléments descendants de l'élément ciblé.

La propriété outerHTML

Cette propriété permet de modifier le contenu HTML de l'élément ciblé et de ses éléments descendants.

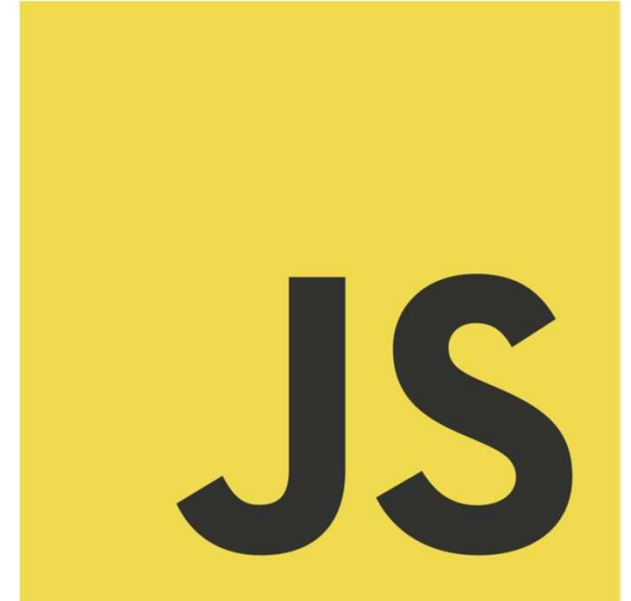
La propriété innerText

Cette propriété permet de modifier le contenu texte de l'élément ciblé et de ses éléments descendants.

Exemple Dossier modifier_element_dom

3- Modifier des éléments du DOM

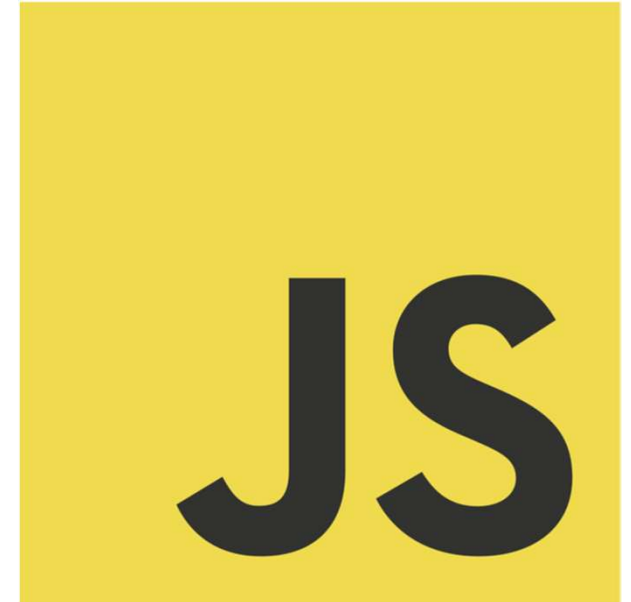
- Exercice :
 - A l'aide de la même structure HTML que précédemment, modifier le text du paragraphe par le texte : "Nouveau texte de mon paragraphe"
- Durée : 5 min



3- Modifier des éléments du DOM

- Correction

Exercice



4- Attributs et propriétés



En HTML, les balises HTML peuvent avoir des attributs.

Tous les attributs se trouvant dans les spécifications HTML Living Standard sont automatiquement reconnues lors du parsing par le navigateur et les propriétés correspondantes sont créés sur le DOM.

En revanche, si l'attribut n'est pas standard, aucune propriété ne sera créé sur le DOM.

Accéder aux attributs

On peut avoir accéder aux attributs standards très facilement de la manière suivante:

```
element.nomDelAttribut;
```

Mais cette syntaxe n'est pas utilisable pour les attributs non standards.

On pourra pour ces éléments (mais également pour les attributs standards) utiliser les méthodes suivantes:

Exemple Dossier attributs_et_proprietes

4- Attributs et propriétés



element.hasAttribute(nom)

Retourne true ou false selon que l'élément ciblé a ou pas l'attribut mis en argument.

element.getAttribute(nom)

Retourne la valeur de l'attribut mis en argument.

element.setAttribute(nom, valeur)

Crée l'attribut mis en premier argument et sa valeur mise en 2^{ème} argument

Si l'attribut existe, cette méthode modifie la valeur.

S'il n'existe pas, la méthode crée l'attribut et lui attribue sa valeur.

Exemple Dossier attributs_et_proprietes

4- Attributs et propriétés



element.removeAttribute(nom)

Supprime l'attribut mis en argument

element.attributes()

Retourne une collection des attributs d'un élément

Attributs et propriétés

Dans la plupart des cas, la modification d'un attribut entraînera la mise à jour de la propriété du DOM, et la modification d'une propriété du DOM entraînera la mise à jour de l'attribut.

Les attributs et les propriétés sont donc synchronisés dans la plupart des cas.

Mais il y a des spécificités à connaître absolument.

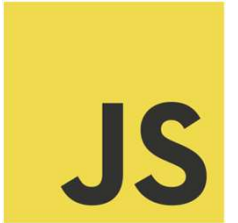
Par exemple, dans le cas d'un input. Il n'est possible de modifier que son attribut value pour voir sa propriété value mise à jour.

Si vous essayez de modifier sa propriété value, son attribut ne sera pas mis à jour :

La raison est que ce sont des éléments HTML sur lequel une action utilisateur est attendue. Or l'utilisateur modifie la valeur de l'attribut en rentrant du texte dans l'input, il ne modifie pas la valeur de la propriété !

Exemple Dossier attributs_et_proprietes

4- Attributs et propriétés



Les attributs data-* et la propriété dataset

Il est fortement conseillé de définir des attributs personnalisés dans le HTML en respectant l'interface prévue pour.

Pour cela, nous allons utiliser les attributs data-*

Ces attributs spécifiques permettent de créer des attributs personnalisés qui pourront correctement être utilisés.

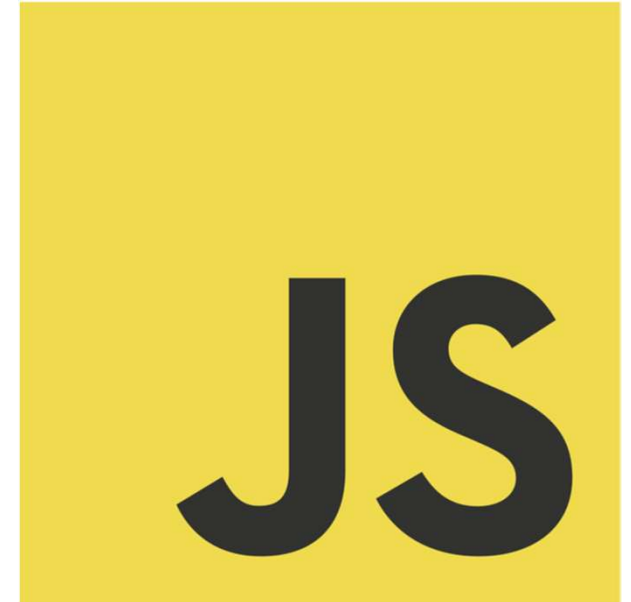
Il est par la suite possible de récupérer la valeur de m'attribut de la manière suivante :

Par exemple, pour un attribut data-id, on récupérera sa valeur avec la propriété dataset

```
element.dataset.id;
```

4- Attributs et propriétés

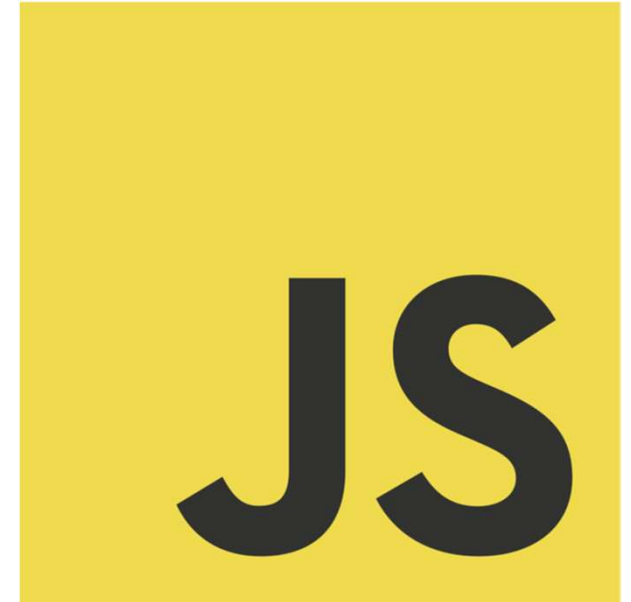
- Exercice : Toujours à partir du HTML créé précédemment
 - Ajouter une balise `<a>` donnez lui les attributs `href=https://www.wf3.fr/` (en utilisant Javascript) avec le texte "Mon lien" et afficher là après le paragraphe
 - Vérifier à l'aide d'une méthode que ce lien a bien l'attribut `href`
 - Puis récupérer cette valeur et afficher là dans la console.
 - Ensuite supprimer cet attribut de ce lien
 - Ajouter ensuite sur la section l'attribut `data-section1` avec la valeur "section1"
- Durée : 10 min



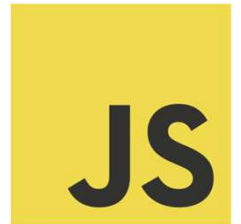
3- Attributs et propriétés

- Correction

Exercice



5- Utilisation de CSS avec Webpack



Nous allons tous d'abord créer un fichier style.css dans le dossier src

Ensuite comme le point d'entrée de notre application est le fichier index.js, nous allons importer style.css dans ce fichier de la manière suivante :

```
import "./style.css";
```

Par contre, comme le fichier est un fichier .css importé dans un .js, il faudra la transformer en .js pour l'importer.

Pour cela, nous allons utiliser webpack et ajouter des loaders en saisissant la commande suivante dans le terminal:

npm install css-loader style-loader -D

```
module: {  
  rules: [  
    {  
      test: /\.js$/,  
      exclude: /node_modules/,  
      use: ["babel-loader"]  
    },  
    {  
      test: /\.css$/,  
      use: ["style-loader", "css-loader"]  
    }  
  ]  
},  
plugins: [
```

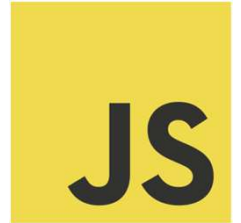
Une fois cela fait, nous allons modifier le fichier de config de webpack, webpack.config

Une fois cela fait, vous pouvez démarrer le serveur avec la ligne de commande: npm start

Exemple

Dossier Le dom

6- Modification du style et des classes



Pour modifier le CSS d'un élément en Javascript, il y a 2 manières:

- Modifier directement le style à l'aide de Javascript mais cela n'est pas recommandé
- Modifier les classes d'un élément afin que le fichier css puisse lui appliquer le style voulu (recommandé)

La propriété style (non recommandée)

Cette propriété peut-être utilisé pour récupérer les styles de l'élément, par contre elle n'est pas recommandé pour cela car elle ne prends pas en compte le style calculé et donc les infos récupérées ne sont pas toujours les bonnes.

Cette propriété peut permettre de modifier le style d'un élément mais cela n'est pas non plus recommandé car c'est fastidieux de le faire de cette manière et il est préférable de modifier le css via les feuilles de styles css et donc notamment à l'aide des classes.

Exemple Dossier `modification_du_style_et_des_classes`

6- Modification du style et des classes



Voici les propriétés et méthodes que nous allons utiliser pour modifier les classes d'un élément

La propriété className

Cette propriété permet de récupérer les classes d'un éléments ou bien de définir la ou les classes d'un élément en écrasant celle actuellement mise en place.

Lorsqu'il récupère les classes, ces classes se présentent sous la forme d'une seule chaîne de caractères.

La propriété classList

Cette propriété permet de récupérer les classes d'un éléments sous forme de liste.

De plus, elle possède de nombreuses méthodes très utile pour manipuler les classes d'un élément

La méthode classList.add()

Cette méthode ajoute la classe mis en argument

```
element.classList.add('nomClasse');
```

Exemple Dossier modification_du_style_et_des_classes

6- Modification du style et des classes



La méthode `classList.remove()`

Cette méthode enlève la classe mis en argument de l'élément ciblé.

```
element.classList.remove('nomClasse');
```

La méthode `classList.toggle()`

Cette méthode enlève ou ajoute la classe à l'élément ciblé selon qu'il la possède ou pas.

Si l'élément possède la classe, cette méthode va l'enlever.

Si l'élément ne possède pas la classe, cette méthode va l'ajouter.

```
element.classList.toggle('nomClasse');
```

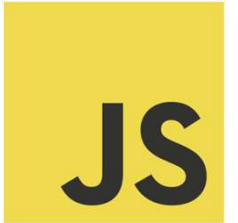
La méthode `classList.contains()`

Cette méthode, qui prends en argument le nom d'une classe, retourne true ou false selon que cette classe existe dans l'élément ciblé ou pas.

```
element.classList.contains('nomClasse');
```

Exemple Dossier `modification_du_style_et_des_classes`

6- Modification du style et des classes



La fonction globale `getComputedStyle()`

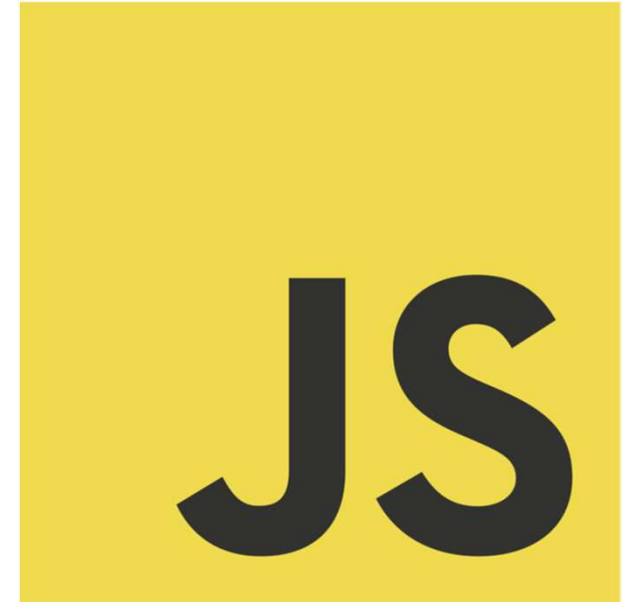
Cette fonction aura comme argument un élément à cibler.
Elle permet de récupérer le style calculé de cet élément.

```
getComputedStyle(element);
```

Exemple Dossier `modification_du_style_et_des_classes`

6- Modification du style et des classes

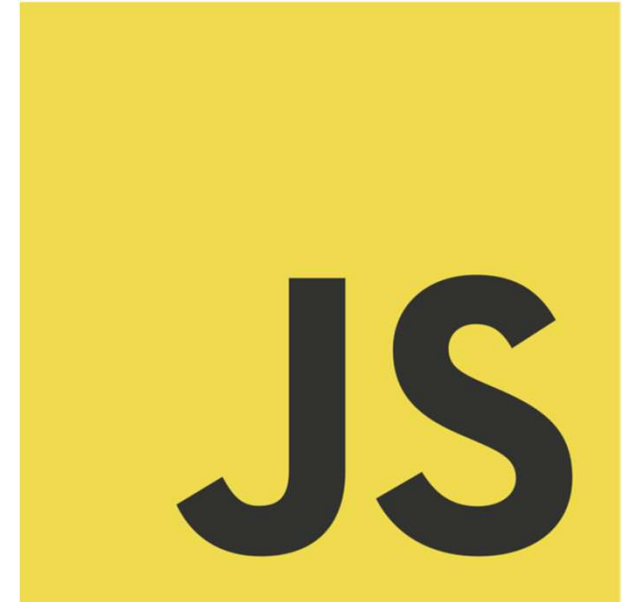
- Exercice : Toujours à partir du HTML créé précédemment
 - Mettre dans le fichier css la classe red pour que le texte soit rouge
 - Ajouter la classe “red” au paragraphe
 - A l’aide d’une méthode, vérifier que ce paragraphe a bien la classe red
 - Terminer par enlever la classe red de ce paragraphe
- Durée : 10 min



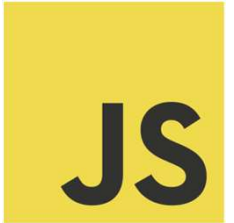
6- Modification du style et des classes

- Correction

Exercice



7- Créer des nœuds et les positionner



Il y a plusieurs méthodes qui permettent de créer des nœuds selon leur nature.

La méthode document.createElement()

Cette méthode prend en argument une chaîne de caractère correspondant à la balise HTML que nous voulons créer.

La méthode document.createTextNode()

Cette méthode prend en argument une chaîne de caractère du texte que vous voulez créer.

La méthode document.createAttribute()

Cette méthode prend en argument une chaîne de caractère de l'attribut que vous voulez créer.

Pour cette méthode, il est préférable de mettre des attributs dans un élément à l'aide de la méthode `element.setAttribute()`

Exemple Dossier `creer_des_noeuds_et_les_positionner`

7- Créer des nœuds et les positionner



Une fois que des nœuds ont été créés, il faut les positionner dans le document.
Vous trouverez ici un choix de méthodes qui permettent le positionnement des nœuds.

La méthode `nœud.prepend()`

Permet d'insérer plusieurs nœuds ou chaînes de caractères en argument avant le premier enfant du nœud ciblé.

La méthode `nœud.append()`

Permet d'insérer plusieurs nœuds ou chaînes de caractères en argument après le dernier enfant du nœud ciblé.

La méthode `nœud.before()`

Permet d'insérer plusieurs nœuds ou chaînes de caractères en argument avant le nœud ciblé.

La méthode `nœud.after()`

Permet d'insérer plusieurs nœuds ou chaînes de caractères en argument après le nœud ciblé.

Exemple Dossier `creer_des_noeuds_et_les_positionner`

7- Créer des nœuds et les positionner



Une autre méthode permet directement d'insérer du contenu HTML à l'intérieur ou à l'extérieur d'un élément.

La méthode `element.insertAdjacentHTML()`

Cette méthode permet d'insérer du contenu HTML à différent endroit par rapport à l'élément ciblé.

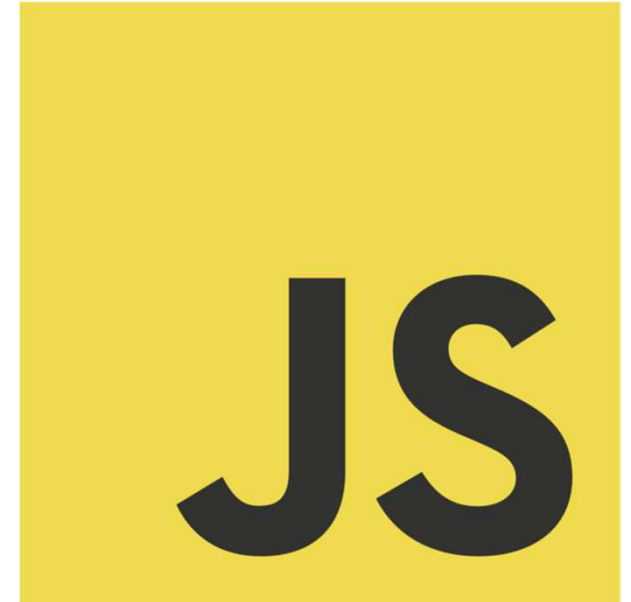
Pour cela, il prend 2 arguments:

- L'endroit où on va mettre le contenu HTML par rapport à l'élément ciblé, il s'agira alors d'une chaîne de caractères parmi les suivantes :
 - `beforebegin` => se positionnera avant l'élément ciblé
 - `afterbegin` => se positionnera à l'intérieur de l'élément ciblé au début
 - `beforeend` => se positionnera à l'intérieur de l'élément ciblé à la fin
 - `afterend` => se positionnera après l'élément ciblé
- Une chaîne de caractères avec le code HTML

Exemple Dossier `creer_des_noeuds_et_les_positionner`

7- Créer des nœuds et les positionner

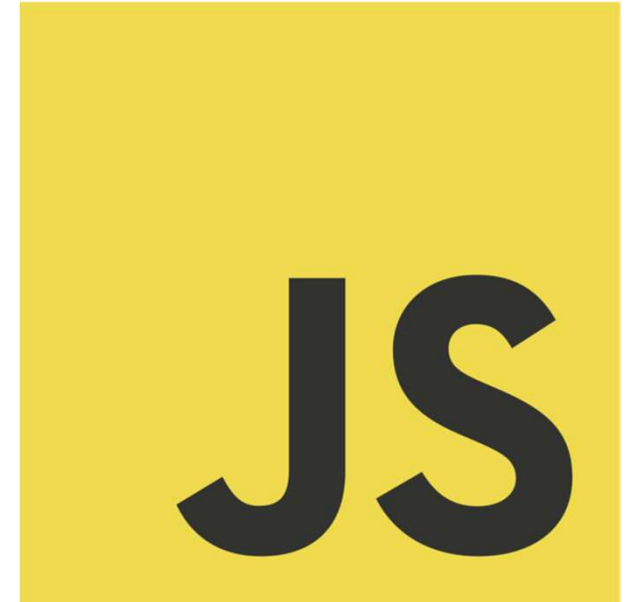
- Exercice :
 - Reprendre le fichier HTML avec la section et le paragraphe
 - Créer un paragraphe avec javascript avec le texte "Mon nouveau paragraphe"
 - Mettez-le après le 1er paragraphe
 - Ensuite placer une nouvelle section après la section actuelle en utilisant la méthode insertAdjacentHTML() avec à l'intérieur le code HTML suivant :
"`<p>The paragraphe</p>`"
- Durée : 10 min



7- Créer des nœuds et les positionner

- Correction

Exercice



8- Supprimer et remplacer des nœuds



Nous allons voir ici 2 méthodes permettant de supprimer ou bien de remplacer un nœud.

La méthode `noeud.remove()`

Cette méthode permet de supprimer le nœud ciblé.

La méthode `noeud.replaceWith()`

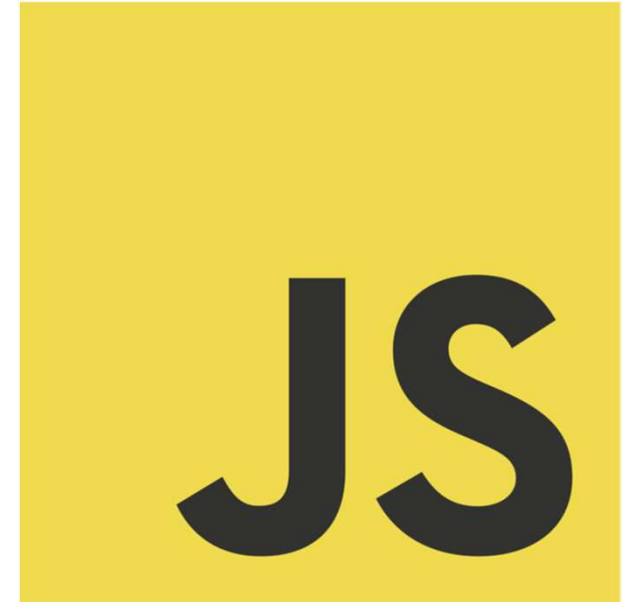
Cette méthode permet de remplacer le nœud ciblé par les nœuds mis en argument(séparé par une virgule) ou bien par des chaînes de caractères.

Exemple

Dossier `supprimer_et_remplacer_des_noeuds`

8- Supprimer et remplacer des nœuds

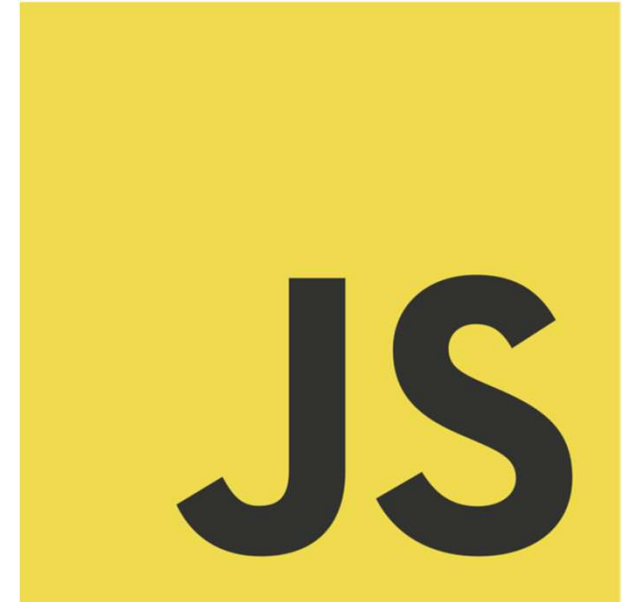
- Exercice :
 - Reprendre le fichier HTML avec la section et le paragraphe
 - Supprimer la paragraphe à l'aide d'une méthode Javascript
 - Remplacer à l'aide d'une méthode Javascript, la section par un titre H1 "Supprimer et remplacer les noeuds"
- Durée : 5 min

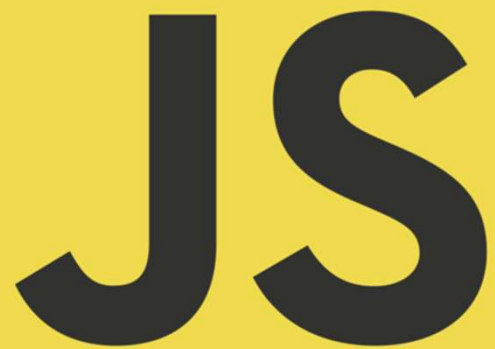


8- Supprimer et remplacer des nœuds

- Correction

Exercice





XII. Les évènements

1- Introduction



Qu'est ce qu'un évènement ?

Un évènement est un signal indiquant que quelque chose est arrivé.

Cela peut-être un clic de souris, une touche pressée,... (il y a de très nombreux évènements qui existent)

Vous trouverez une liste à cette adresse https://www.w3schools.com/jsref/dom_obj_event.asp

Cela est très utile pour avoir de l'interaction avec l'utilisateur.

Pour qu'un évènement soit traité, il faut 2 choses :

- Mettre en place un écouteur d'évènement
- Déclencher une instruction lorsque l'évènement est émis

2- Utilisation des propriétés du DOM on*



Tous les nœuds ont des propriétés on* qui sont à null par défaut mais sur lesquelles on peut passer un gestionnaire d'évènement.

La syntaxe est dans ce cas la suivante:

```
noeud.onclick = () => //instruction ;
```

Ici, nous avons pris l'exemple de la propriété onclick mais il y en a beaucoup d'autres

Ou bien :

```
noeud.onclick = function(){  
    //instructions;  
}
```

ATTENTION à la valeur de this, en effet celle-ci est différente si vous utilisez une fonction classique et une fonction fléchée.

Exemple Dossier utilisation_des_propriétés_du_dom_on

2- Utilisation des propriétés du DOM on*



Les fonctions à exécuter peuvent prendre un argument qui représentera l'objet événement.

Cela peut permettre notamment de contourner le problème de valeur de this et permettre de récupérer à l'aide de l'objet événement sa cible avec la propriété currentTarget

Exemple de code utilisant currentTarget:

```
noeud.onclick = function(e){  
    console.log(e.currentTarget);  
}
```

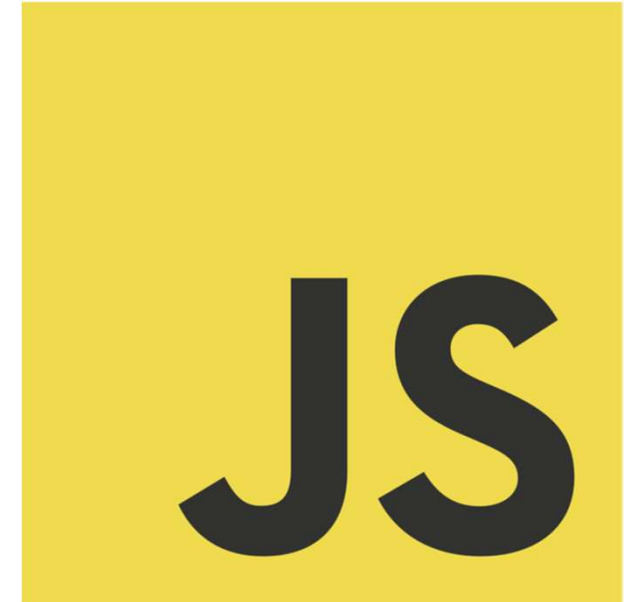
En faisant cela e.currentTarget permet de récupérer le nœud comme un this avec les fonctions classiques.

Exemple

Dossier utilisation_des_propriétés_du_dom_on

2- Utilisation des propriétés du DOM on*

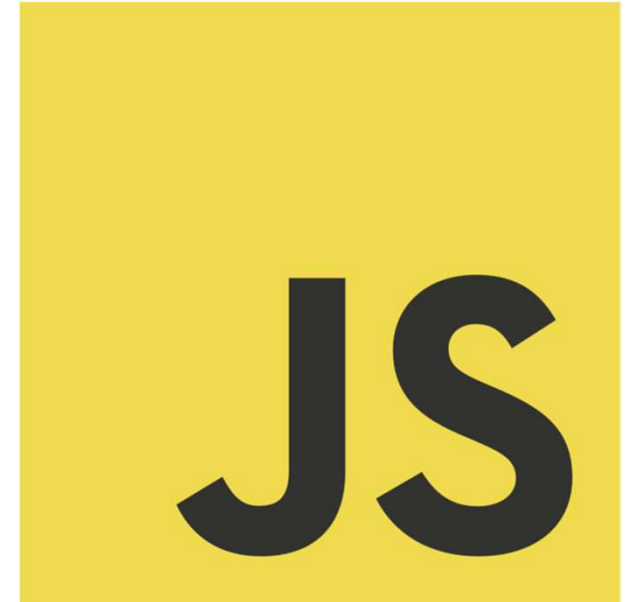
- Exercice :
 - Créer un fichier HTML (qui sera réutilisé plus tard) avec les éléments suivant : un titre H1 et 2 boutons
 - Créer un évènement qui permet d'afficher dans la console au clic sur le premier bouton le texte suivant : "L'évènement fonctionne"
 - Créer un évènement qui permet au clic sur le deuxième bouton de changer le background du bouton en rouge s'il ne l'est pas et revenir à sa couleur d'origine si il l'est
- Durée : 15 min



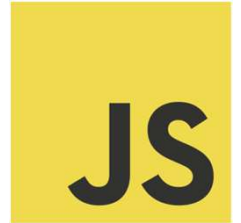
2- Utilisation des propriétés du DOM on*

- Correction

Exercice



3- La méthode addEventListener



Il existe une limite à l'utilisation des propriétés DOM `on*`, en effet, il n'est possible pour chaque évènement que d'exécuter une seule fonction.

Pour résoudre ce problème, il existe une méthode qui s'appelle `node.addEventListener()`

La méthode `node.addEventListener()`

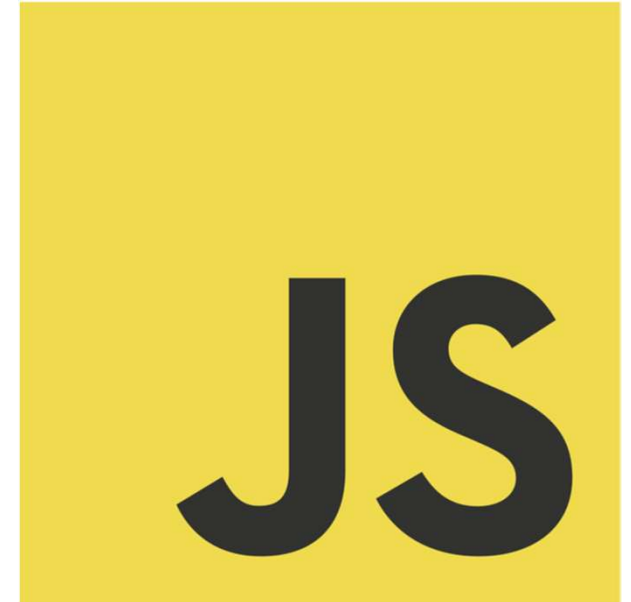
Cette méthode possède plusieurs arguments:

- Le premier est le type d'évènement (exemple `click`,...). Il s'agit des mêmes évènements qu'avec `on*` mais sans le `on`
- Le deuxième est la fonction qui doit s'exécuter (le gestionnaire d'évènement)
- Le troisième est optionnel et permet de passer des options
 - Exemple l'option `once` qui permet de retirer le gestionnaire d'évènement dès la première émission de l'évènement ciblé.

Exemple Dossier `addEventListener`

3- La méthode addEventListener

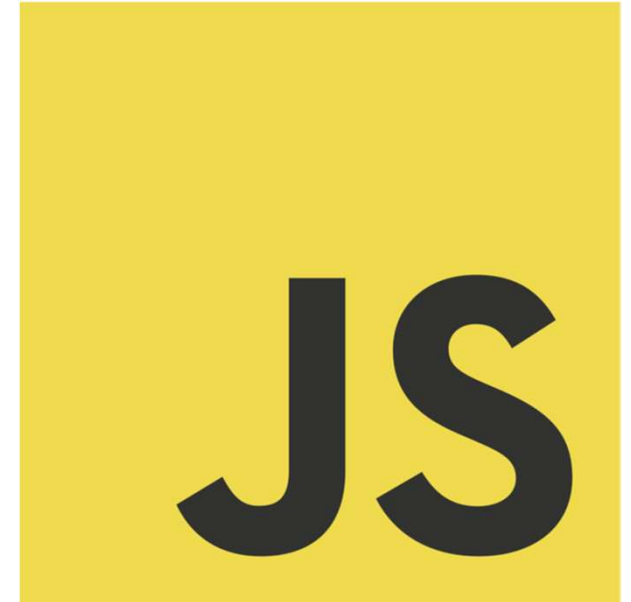
- Exercice :
 - Reprendre le code précédent et modifier le pour utiliser `addEventListener()` au lieu de `onclick`
- Durée : 5 min



3- La méthode addEventListener

- Correction

Exercice



4- Supprimer un gestionnaire d'évènement et déclencher un évènement



La méthode `nœud.removeListener()`

Cette méthode permet de supprimer un gestionnaire sur un nœud.

Elle prends comme arguments les éléments suivants:

- Le type d'évènement (click...)
- Le gestionnaire d'évènement à supprimer
- Des options (facultatif)

La méthode `nœud.dispatchEvent()`

Cette méthode permet de déclencher un évènement de manière artificielle.

Elle prends comme arguments :

- Un objet constructeur Event avec comme argument le type d'évènement (exemple click)

Exemple

Dossier `supprimer_evenement`

4- Supprimer un gestionnaire d'évènement et déclencher un évènement

- Exercice :
 - Créer un 3ème bouton qui au click supprimera l'évènement déclenché par le bouton1
 - Modifier ce bouton3 pour qu'il déclenche également l'évènement du bouton2
- Durée : 15 min

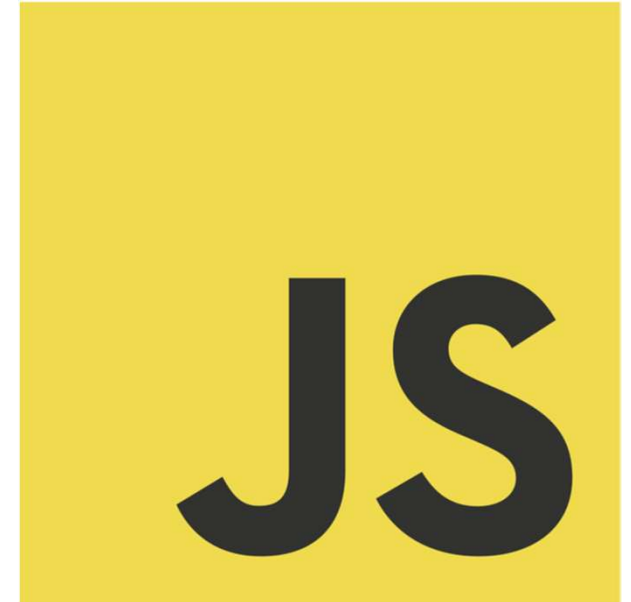


21
1

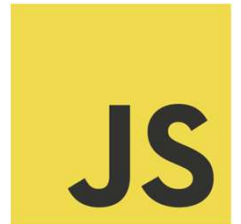
4- Supprimer un gestionnaire d'évènement et déclencher un évènement

- Correction

Exercice



5- Empêcher le comportement par défaut et l'objet Event



L'objet Event

Les événements émis par les noeuds sont des objets qui ont de nombreuses propriétés. Certaines sont communes à la plupart des événements et d'autres sont spécifiques au type d'événement.

La méthode stopPropagation()

Il se peut que lors d'un déclenchement d'un événement, d'autres se déclenchent également par exemple dans le cas d'élément HTML imbriqué.

La méthode stopPropagation() permet de résoudre cela.

Exemple Dossier event

5- Empêcher le comportement par défaut et l'objet Event



La méthode preventDefault()

Beaucoup d'éléments HTML ont des comportements par défaut extrêmement utiles.

Par exemple, le clic sur un lien permet d'initier une navigation sur l'URL ciblée.

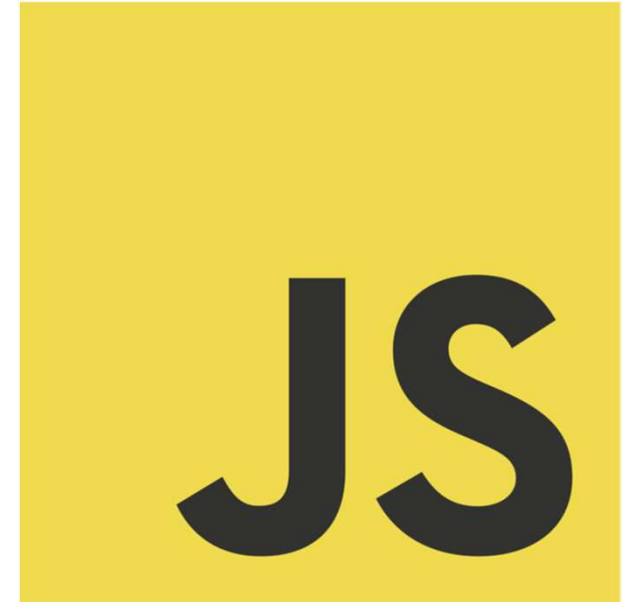
Mais parfois, nous voulons empêcher ces comportements par défaut pour les remplacer par des comportements personnalisés.

C'est ici qu'entre en jeu la méthode `preventDefault()`. Comme son nom l'indique, elle va prévenir le comportement par défaut.

Exemple Dossier event

5- Empêcher le comportement par défaut et l'objet Event

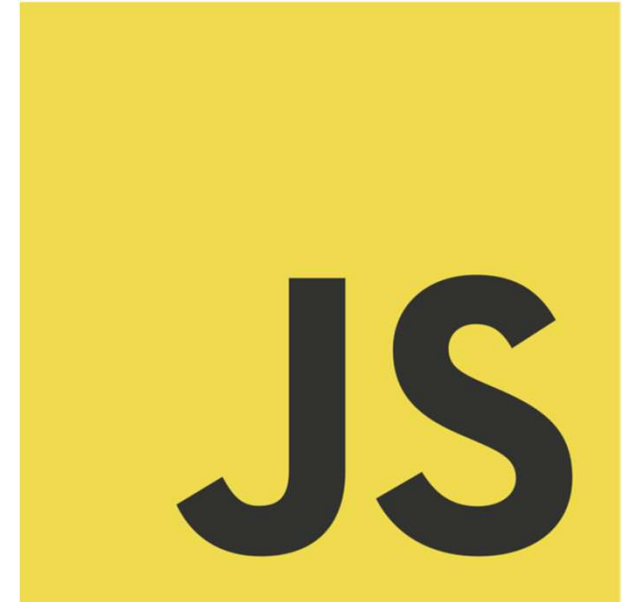
- Exercice :
 - Dans un fichier HTML, mettre une section et à l'intérieur un formulaire avec juste un input texte et un bouton submit
 - Créer 2 événements, le 1er au survol avec la souris de la section affiche dans la console "Survol de la section" et un 2ème lorsqu'on survol l'input qui affiche dans la console "Survol de l'input"
 - Modifier le code pour que lorsqu'on survol l'input, le message "Survol de la section" ne s'affiche pas
 - Modifier le code pour que lorsqu'on clic sur le bouton submit, la page ne se recharge pas
- Durée : 15 min



5- Empêcher le comportement par défaut et l'objet Event

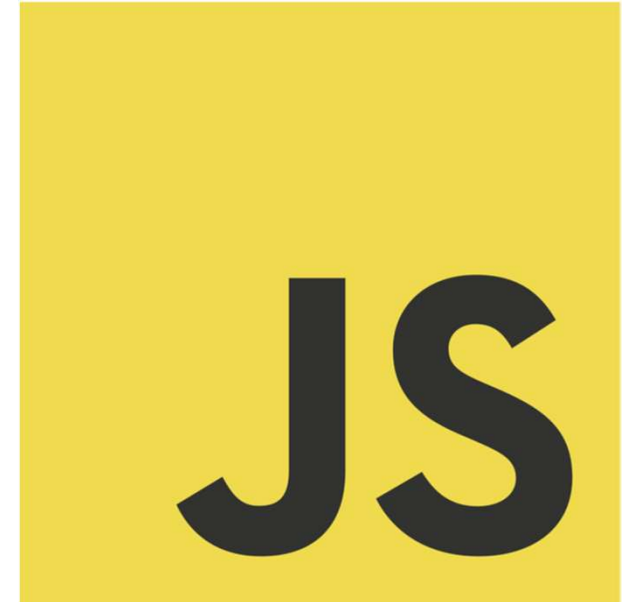
- Correction

Exercice



PROJET : TODO List

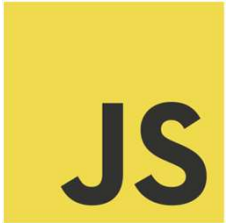
- Exercice :
- Durée : 15 min





XIII. Gérer l'asynchrone sur un navigateur

1- Introduction à l'asynchrone et timers



Une instruction est asynchrone quand elle n'est pas exécutée immédiatement.

La méthode setTimeout()

Cette méthode permet de définir un minuteur qui va déclencher l'exécution d'une fonction de rappel mise en 1^{er} argument lorsque le temps est écoulé.

En 2^{ème} argument, nous avons la durée en ms avant l'exécution de la fonction de rappel.

La méthode retourne un identifiant.

La méthode clearTimeout()

Cette méthode permet d'annuler le minuteur créé à l'aide de la méthode setTimeout().

Pour cela, cette méthode a besoin d'avoir en argument l'identifiant retourné par la fonction setTimeout()

Exemple

1- Introduction à l'asynchrone et timers



La méthode setInterval()

Cette méthode permet d'exécuter une fonction de rappel mise en 1^{er} argument de façon répétée selon l'intervalle spécifiée en 2^{ème} argument.

Cette méthode retourne un identifiant.

La méthode clearInterval()

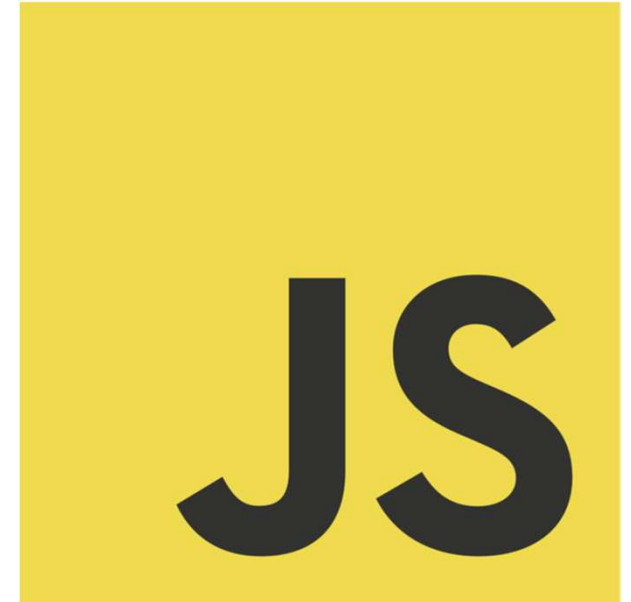
Cette méthode permet d'arrêter l'exécution d'une fonction de rappel créé à l'aide de la méthode setInterval().

Pour cela, cette méthode a besoin d'avoir en argument l'identifiant retourné par la fonction setInterval().

Exemple

1- Introduction à l'asynchrone et timers

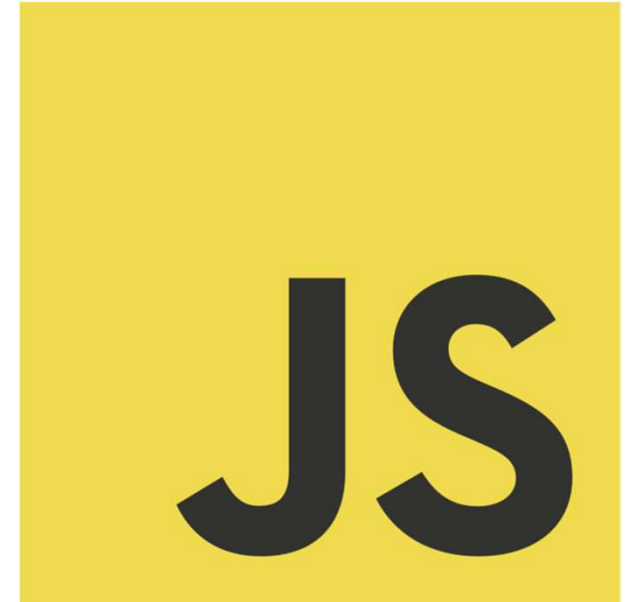
- Exercice:
 - Afficher un message de bienvenue ("Bienvenue sur notre nouveau site") dans un pop-up après 2 seconde.
 - Afficher dans la page un texte : "Vous êtes sur cette page depuis x secondes". Remplacer x par le nombre de seconde depuis le début de la connexion sur la page
- Durée: 40 min



1- Introduction à l'asynchrone et timers

- Correction

Exercice



2- Les promesses



Définition des promesses

Les promesses représentent une valeur qui sera disponible dans le futur.

Elles permettent de gérer l'asynchrone de manière plus simple que les fonctions de rappel.

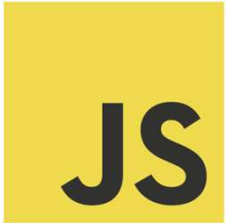
Les états d'une promesse

Une promesse peut être dans un des trois états suivants.

- Son état initial est pending (en attente). La promesse est en cours d'exécution.
- Ensuite la promesse peut être tenue (fulfilled)
- ou rompue (rejected).

Lorsqu'elle a atteint un de ces deux derniers états on dit qu'elle est acquittée (settled). Mais ce n'est pas un état, c'est une manière de dire qu'elle n'est plus en cours.

2- Les promesses



Déclaration d'une promesse

Une promesse est un objet natif JavaScript Promise qui se déclare comme ceci :
`new Promise((resolve, reject) => {});`

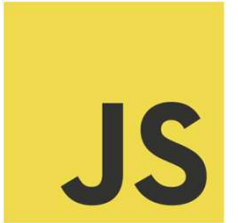
La fonction passée à une promesse n'est pas une fonction de rappel.
Elle est exécutée immédiatement avant que l'objet Promise soit retourné.
Elle est appelée fonction exécuteur.

Elle reçoit deux fonctions en arguments `resolve()` et `reject()` qui permettent, soit de tenir, soit de rejeter une promesse et de retourner une valeur passée en argument.

Une promesse va donc exécuter du code asynchrone dans sa fonction exécuteur, ensuite il faut obligatoirement appeler l'une des deux fonctions de rappel passées en argument en fonction du résultat des tâches asynchrones.

Exemple

2- Les promesses



Consommer une promesse

La méthode `then()`

La méthode `then()` s'utilise sur une promesse et prend un ou deux arguments. Elle permet de gérer la valeur de retour en cas de rejet ou de succès.

Généralement, nous l'utiliserons que pour le succès.

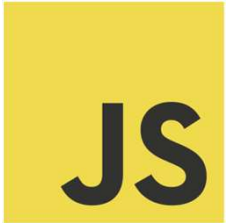
La méthode `catch()`

La méthode `catch()` est un raccourci de la méthode `then()` lorsque nous ne sommes intéressés que par le retour du `rejected()`.

Il est recommandé d'utiliser la méthode `then()` avec un unique paramètre pour gérer le succès de la promesse, et la méthode `catch()` pour gérer la gestion du rejet. Le code est plus lisible si vous procédez de cette manière.

Exemple

2- Les promesses



La méthode `finally()`

La méthode `finally()` prend une seule fonction de rappel en argument qui est appelée que la promesse soit tenue ou rompue.

La fonction de rappel ne reçoit aucune valeur.

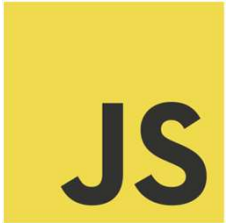
Elle est utile pour exécuter du code lorsque la promesse est terminée, par exemple pour retirer l'affichage d'un GIF de chargement.

Chaîner des promesses

Toutes les méthodes que nous avons vues retournent une nouvelle promesse.

Cela signifie que vous pouvez chaîner les promesses pour réaliser autant de traitements asynchrones que vous souhaitez à la suite.

3- Les méthodes des promesses



Raccourci syntaxique avec Promise.reject() et Promise.resolve()

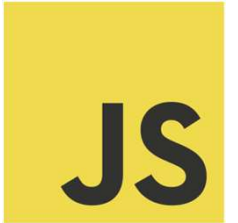
Ces méthodes permettent de créer directement un objet Promise acquitté :

- soit tenue avec `resolve()`,
- soit rompue avec `reject()`.

Elles peuvent être utiles lorsqu'une API a besoin d'utiliser une promesse mais que vous pouvez obtenir la valeur de manière synchrone (par exemple si vous l'avez mise en cache).

Exemple

3- Les méthodes des promesses



La méthode `Promise.all()`

Cette méthode permet d'effectuer des traitements asynchrones en parallèle et d'attendre que tous ces traitements soient terminés pour retourner une valeur.

La méthode prend en argument un tableau de promesses et va attendre soit qu'une des promesses échoue, soit que toutes les promesses soient résolues.

Elle retourne une nouvelle promesse et il est donc possible de chaîner avec `then()`, `catch()` ou `finally()`.

Si une promesse est rejetée, elle n'attend pas les autres promesses et renvoie immédiatement une erreur.

Si toutes les promesses sont tenues, elle renvoie un tableau de résultats qui contient les valeurs retournées par les promesses dans le même ordre que la déclaration des promesses (et non pas dans l'ordre d'arrivée des valeurs).

Exemple

3- Les méthodes des promesses



La méthode `Promise.allSettled()`

La méthode `Promise.allSettled()` renvoie une promesse qui est résolue une fois que l'ensemble des promesses passées en argument sont réussies ou rejetées.

Cette méthode prend également un tableau de promesse.

Cette méthode retourne un tableau de résultats contenant des objets.

- Pour les promesses tenues, l'objet est de la forme `{status:"fulfilled", value:42}`.
- Pour les promesses rejetées, l'objet est de la forme `{status:"rejected", reason: "erreur..."}`.

La méthode `Promise.race()`

La méthode `Promise.race()` renvoie une promesse qui est résolue ou rejetée dès qu'une des promesses passées en argument est résolue ou rejetée.

Elle prend en argument un tableau de promesses comme pour les autres méthodes.

Cette méthode fait la course entre les promesses, d'où son nom !

Elle retourne uniquement le résultat de la promesse la plus rapide à être acquittée.

Exemple

4- Utilisation de polyfills avec Webpack



Les polyfills

Un polyfill est du code d'une version plus ancienne de JavaScript qui permet de simuler une fonctionnalité récente du langage et qui n'est pas encore disponible dans toutes les versions des navigateurs.

Configuration de Webpack pour utiliser des polyfills

Heureusement nous n'avons pas à ajouter nous-même tous les polyfills nécessaires !

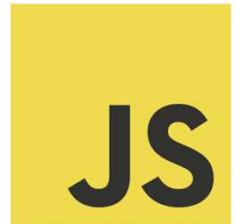
Webpack et Babel vont le faire pour nous. Mais ils ont besoin d'une librairie `core-js`

Cette librairie est une immense liste de polyfills.

Nous avons également besoin de `regenerator-runtime` qui est une librairie maintenue par Facebook et qui permet notamment de permettre le polyfill en ES5 des fonctions asynchrones que nous allons utiliser.

```
npm i -D core-js@3 regenerator-runtime
```

4- Utilisation de polyfills avec Webpack



Modification du fichier de configuration babel

Maintenant que nous avons les librairies nécessaires, nous allons modifier le fichier `babel.config.js`

```
const presets = [  
  [  
    "@babel/preset-env",  
    {  
      useBuiltIns: "usage",  
      debug: true,  
      corejs: 3,  
      targets: "> 0.25%, not dead"  
    }  
  ]  
];  
  
module.exports = { presets };
```

L'option `useBuiltIns` permet de gérer l'utilisation des polyfills par Babel. Par défaut l'option est à `false` et Babel ne va pas ajouter de polyfills automatiquement. La valeur `usage` permet d'importer les polyfills nécessaires pour les versions ciblées des navigateurs pour les fonctionnalités utilisées (à la différence d'`entry` qui importera tous les polyfills nécessaires pour les versions des navigateurs ciblées sans distinction de l'utilisation).

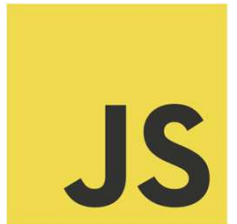
L'option `corejs` permet simplement de spécifier quelle version de la librairie `core-js` nous utilisons.

L'option `debug` permet d'indiquer dans la console.log quels sont les versions des navigateurs ciblés par votre requête et tous les polyfills nécessaires pour votre code.

L'option `targets` est une requête permettant de définir les navigateurs ciblés.

<https://babeljs.io/docs/en/babel-preset-env>

5- Les fonctions asynchrones avec `async` / `await`



Les fonctions asynchrones

Une fonction asynchrone s'exécute de façon asynchrone en utilisant une promesse.

Pour déclarer une fonction asynchrone, il suffit d'utiliser le mot clé `async` devant `function`

Autrement dit le mot clé `async` permet de s'assurer qu'une fonction va toujours retourner une promesse.

Attendre le résultat d'une fonction asynchrone

L'opérateur `await` permet d'attendre la résolution d'une promesse uniquement dans une fonction asynchrone.

C'est une syntaxe extrêmement concise et élégante : elle permet d'attendre l'acquittement d'une promesse et d'obtenir sa valeur si elle est tenue.

Une erreur sera automatiquement levée par le moteur.

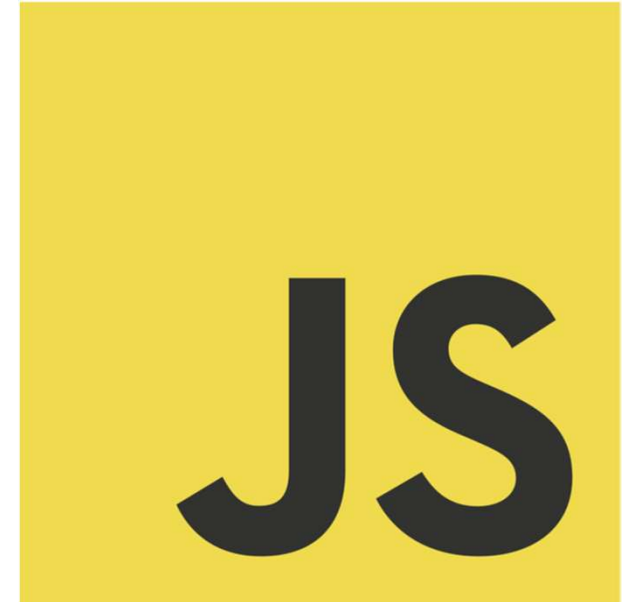
Il est possible de la gérer dans un `catch()`

Mais la méthode recommandée est l'utilisation de blocs `try` / `catch` qui permet de récupérer les erreurs du bloc `try` dans le bloc `catch`

Exemple

5- Les fonctions asynchrones avec async / await

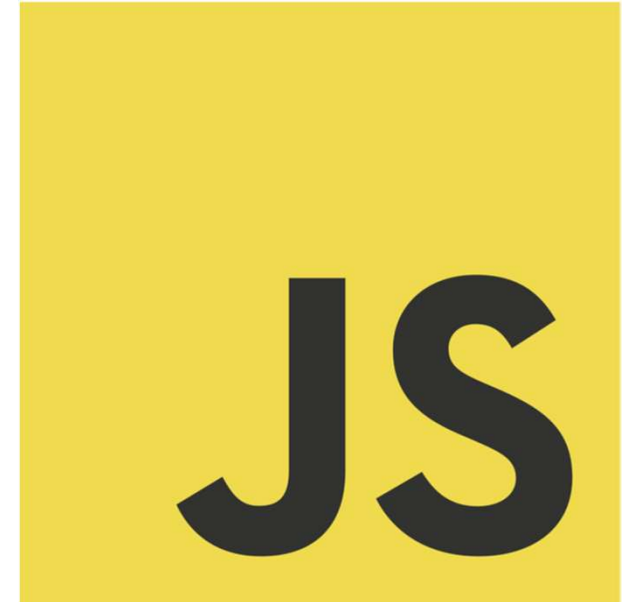
- Exercice:
 - Créer une fonction asynchrone qui appellera une promesse.
 - Cette promesse contiendra un tableau avec 2 valeurs "200" et "500", elles représentent si la requete a fonctionné ou pas.
 - Au bout de 2 secondes, on choisi un résultat aléatoirement parmi les 2 valeurs du tableau
 - Si 200, la promesse est tenue et retourne "John Doe"
 - Sinon on retourne ""
 - La fonction asynchrone affiche dans un span, que vous créerait en javascript, le résultat "Bonjour John Doe" ou bien "Bonjour"
- Durée: 20 min



5- Les fonctions asynchrones avec async / await

- Correction

Exercice



6- Event loop



La boucle d'événements (Event loop)

La boucle d'événements (event loop) est une boucle infinie qui permet de gérer tous les événements.

Les priorités des différentes tâches

Pile d'exécution > File des micro-tâches > File des tâches

Pile d'exécution

Ici va s'exécuter tous le code synchrone dans l'ordre FIFO (First In First Out). La 1^{er} ligne de code lu sera la première exécuté

File des micro-tâches

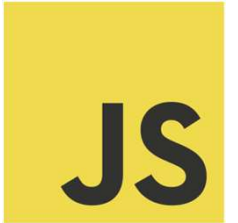
Les promesses utilisent une file différente qui est aussi une file FIFO (first-in-first-out) : la file des micro-tâches.

Ce qui se passe est que les gestionnaires de promesses sont mis sur cette file lorsque la promesse est acquittée.

Ainsi, lorsque nous appelons `then()`, cette méthode est ajoutée à la file des micro tâches lorsque la promesse sur laquelle elle est appelée est acquittée.

Exemple

6- Event loop

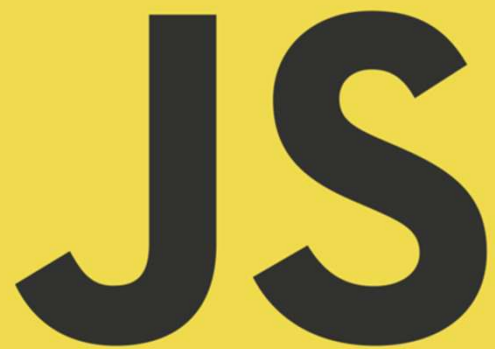


La file des tâches

La file des tâches contient toutes les fonctions passées comme gestionnaire à une Web API asynchrone et qui a terminé son traitement.

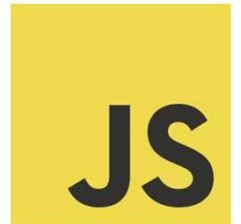
C'est le cas par exemple des `addEventListener()`, des `setTimeout()`, des `setInterval()`

Exemple



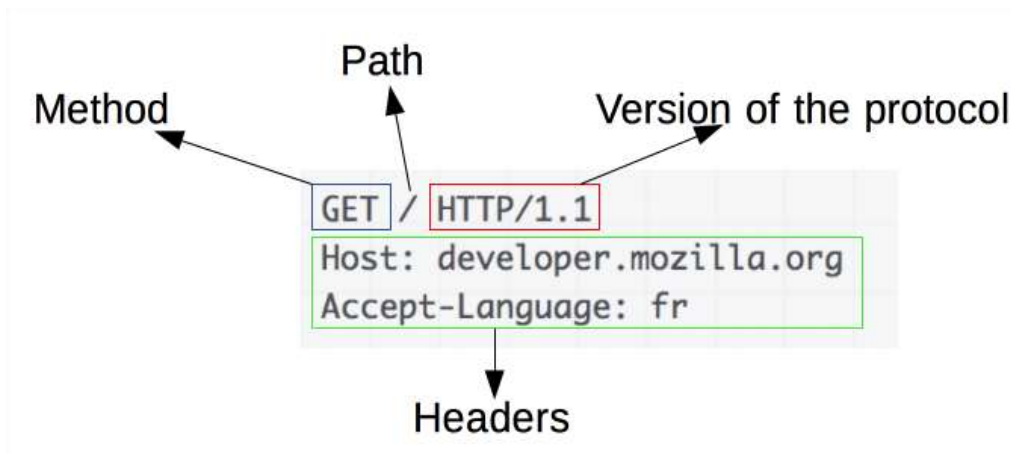
XIV. Le réseau

1- Introduction aux requêtes



Qu'est-ce qu'une requête HTTP ?

Une requête HTTP se décompose comme suit :



La ligne requête (request line)

La première ligne est appelée request line elle comporte trois parties :

- la méthode HTTP : GET, POST, ...
Elle exprime l'intention de la requête.
Par exemple GET signifie donne moi la ressource à l'URI indiqué.
POST signifie je t'envoie des données.
- Request-URI : est l'identifiant de la ressource demandée
- La version du protocole : 1.1 ou 2.

Les Headers

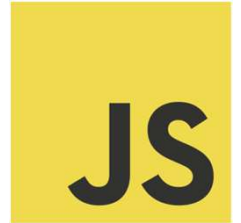
Il existe de nombreux headers HTTP.

Leur forme est toujours constituée du nom du header suivi d'un deux-points puis sa valeur.
Ces headers de requête permettent de contrôler la mise en cache, l'authentification...

Le body

Pour les requêtes dont la méthode est par exemple POST, elles transmettent des données, par exemple le contenu d'un formulaire.

1- Introduction aux requêtes



Faire des requêtes HTTP ?

Pour effectuer des requêtes HTTP vous pouvez utiliser un navigateur.

Par exemple lorsque vous allez sur l'adresse <http://wikipedia.fr> le navigateur va envoyer une requête GET au serveur wikipédia.

Vous pouvez également utiliser du HTML, avec comme nous l'avons vu les formulaires, qui permettent d'envoyer des requêtes POST et GET.

Vous pouvez utiliser des Web API du navigateur en utilisant du JavaScript, que nous allons étudier, et qui vous permettent de faire n'importe quel type de requête HTTP et de gérer la réponse du serveur en JavaScript.

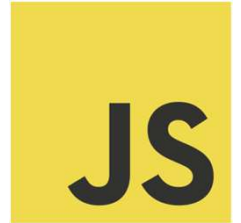
Les requêtes AJAX

Nous appelons requêtes AJAX les requêtes effectuées en JavaScript qui n'entraînent pas un rechargement de la page.

Elles permettent principalement une mise à jour dynamique de certaines parties des pages en JavaScript.

Elles utilisent des Web API du navigateur : `Fetch` ou `XMLHttpRequest`.

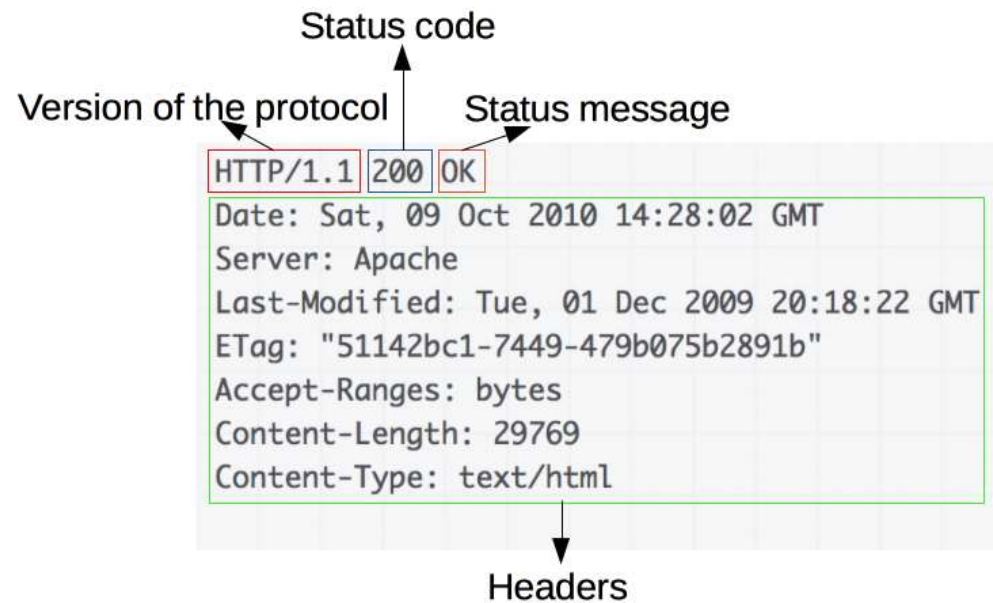
2- Première requête HTTP avec fetch



Avant d'envoyer notre première requête, nous avons besoin d'étudier ce qu'un serveur va nous retourner : une réponse HTTP.

La réponse HTTP

Le serveur après avoir traité la requête et effectué les opérations souhaitées doit renvoyer une réponse au client.



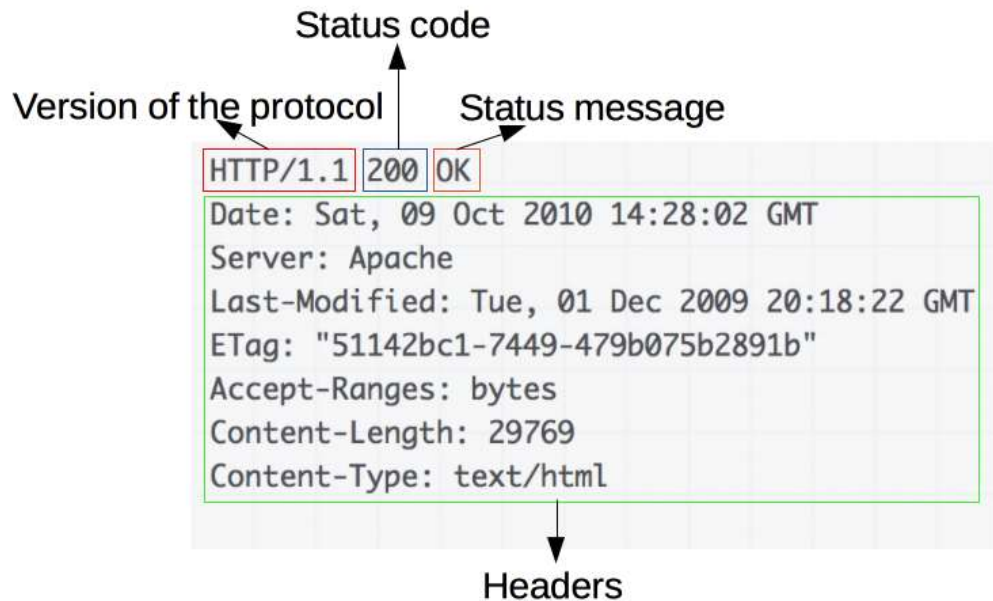
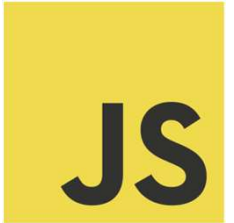
La ligne réponse

Elle comporte la version du protocole utilisé puis le code du statut et le message correspondant.

Les status code et status message : Les codes de status permettent d'indiquer si une requête HTTP a été exécutée avec succès ou non. Un message court l'accompagne.

- Les codes commençant par 2 signifie que la requête a fonctionné.
- Les codes commençant par 3 sont utilisés pour les redirections.
- Les codes commençant par 4 sont utilisés pour les erreurs côté client.
- Les codes commençant par 5 sont utilisés pour les erreurs côté serveur.

2- Première requête HTTP avec fetch



Les Headers

Comme pour la requête il existe des headers spécifiques à la réponse HTTP.

- **Access-Control-Allow-Origin: *** : cela est relié aux requêtes CORS (Cross-origin resource sharing). Le serveur peut contrôler l'accès d'un agent utilisateur à des ressources. Le serveur indique par * que toute origine peut accéder à ses ressources.
- Le **ETag** : est un identifiant unique pour une version spécifique d'une ressource, il est géré par une librairie le plus souvent et permet une mise en cache optimal des ressources pour n'envoyer la ressource que si elle a changé depuis la dernière visite.
- **Date** spécifie juste le moment de la réponse et **Last-Modified** est la date de la dernière modification de la ressource.
- **Content-Type**: text/html; charset=utf-8 : permet d'indiquer à l'agent utilisateur le media-type et l'encoding de la ressource renvoyée.
- **Content-Length** indique la taille en octets (exprimée en base 10) du corps de la réponse envoyée au client.

<https://developer.mozilla.org/fr/docs/Web/HTTP/Headers>

2- Première requête HTTP avec fetch



La Web API fetch

`fetch` est une Web API disponible dans tous les navigateurs qui permet d'envoyer des requêtes HTTP.

Syntaxe

La syntaxe de l'API est très simple :

```
const promesse = fetch(url, [options]);
```

Le premier paramètre est l'URL cible de la requête.

Le second paramètre est un objet d'options.

La promesse est résolue avec un objet `Response`.

Exemple

2- Première requête HTTP avec fetch



Envoyer une requête GET

Sans passer d'option, `fetch` va effectuer une requête HTTP avec la méthode `GET` :

```
const reponse = await fetch("https://jsonplaceholder.typicode.com/users");
```

Vous pouvez à ce stade accéder aux propriétés :

```
console.log(reponse.ok);  
console.log(reponse.status);  
console.log(reponse.statusText);  
...
```

Parser la réponse

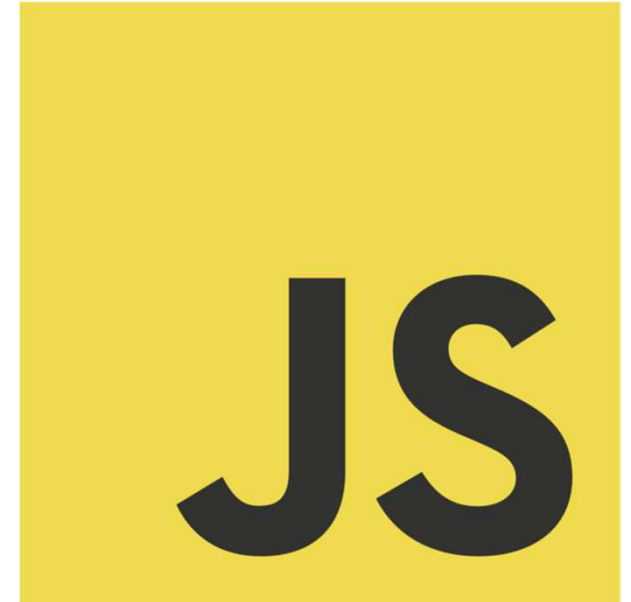
Pour lire le contenu de la réponse, il faut la parser, c'est-à-dire attendre que tout le body soit reçu et le lire dans un format particulier.

- `text()` permet de lire la réponse et de la parser au format text.
- `json()` permet de lire la réponse et de la parser au format json.
- ...

Exemple

2- Première requête HTTP avec fetch

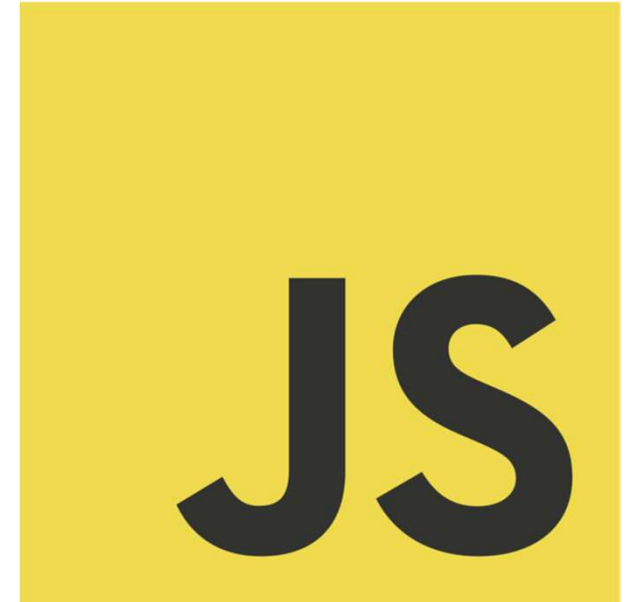
- Exercice:
 - Récupérer une liste de Todo au format json puis afficher dans la console
- Durée: 15 min



2- Première requête HTTP avec fetch

- Correction

Exercice



3- Effectuer une requête POST



Envoyer une requête GET

Sans passer d'option, `fetch` va effectuer une requête HTTP avec la méthode `GET` :

```
const reponse = await fetch("https://jsonplaceholder.typicode.com/users");
```

Vous pouvez à ce stade accéder aux propriétés :

```
console.log(reponse.ok);  
console.log(reponse.status);  
console.log(reponse.statusText);  
...
```

Parser la réponse

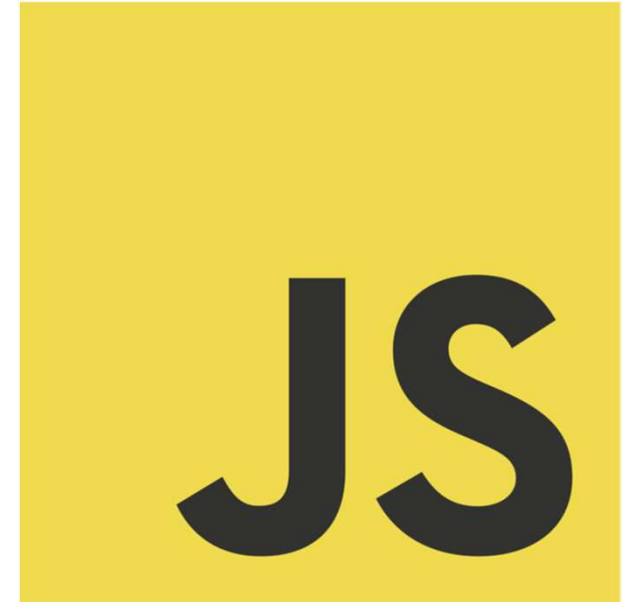
Pour lire le contenu de la réponse, il faut la parser, c'est-à-dire attendre que tout le body soit reçu et le lire dans un format particulier.

- `text()` permet de lire la réponse et de la parser au format text.
- `json()` permet de lire la réponse et de la parser au format json.
- ...

Exemple

3- Effectuer une requête POST

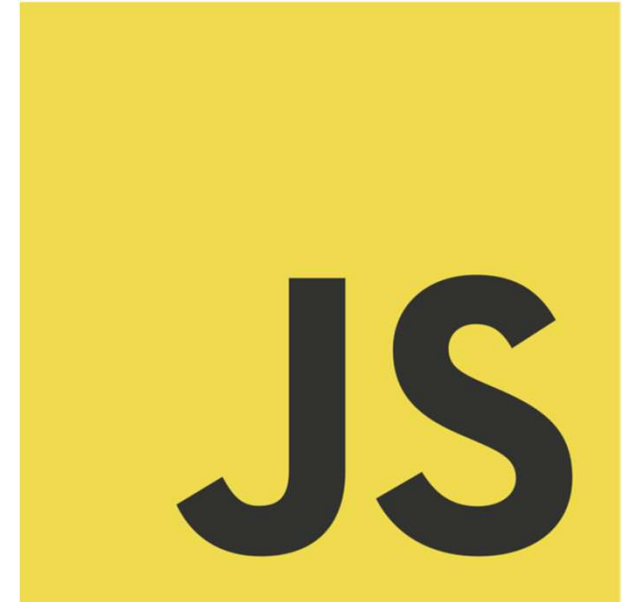
- Exercice:
 - Envoyer une todo via une requête POST
- Durée: 10 min



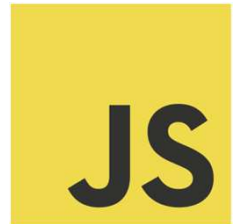
3- Effectuer une requête POST

- Correction

Exercice



4- Les CORS



Le CORS (Cross-origin resource sharing) est un mécanisme de sécurité des navigateurs.

Il permet de définir si un utilisateur peut accéder à des ressources d'un serveur situé sur une autre origine que le site courant.

Cela permet d'éviter que du JavaScript d'un site Web puisse accéder aux données d'un autre site Web.

Sinon des scripts malveillants pourraient très simplement récupérer beaucoup d'informations sur les utilisateurs d'une page.

Ce mécanisme est valable pour les Web APIs API XMLHttpRequest et Fetch.

Une origine est constitué d'un domaine, d'un protocole et d'un port.

Par défaut, du JavaScript ne peut donc effectuer des requêtes que sur la même origine depuis laquelle il a été chargé (c'est ce qu'on appelle la Same-origin policy).

Seul le serveur qui contrôle la ressource peut décider de sa politique CORS, il s'agit donc d'une problématique uniquement serveur.

4- Les CORS



Les requêtes simples

Les requêtes simples sont des requêtes qui utilisent les méthodes GET, POST ou HEAD ET qui utilisent uniquement un ou plusieurs entêtes de la liste suivantes :

Accept, Accept-Language, Content-Language, Content-Type, Last-Event-ID, DPR, Save-Data, Viewport-Width, Width.

Dernière condition, si l'entête Content-Type est utilisé, les seules valeurs possibles sont : application/x-www-form-urlencoded, multipart/form-data et text/plain.

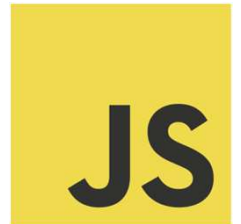
Si la requête respecte ces conditions elle est considérée comme simple.

Dans ce cas, le navigateur envoie directement la requête en spécifiant l'entête Origin.

Le serveur peut ensuite décider selon l'Origin de répondre ou non.

Si il accepte la requête CORS la réponse aura un entête access-control-allow-origin

4- Les CORS



Les autres requêtes

Toutes les autres requêtes nécessitent l'envoi d'une requête CORS préliminaire par le navigateur appelée `prelight request` avec la méthode `OPTIONS`.

Vous n'avez aucun contrôle sur cette requête : c'est le navigateur qui la fait.

Cette requête préliminaire aura trois entêtes importants :

- `origin` qui va permettre au serveur de déterminer si il accepte l'accès.
- `Access-Control-Request-Headers` qui va contenir tous les headers passés en option de la requête et permet au serveur de les contrôler.
- `Access-Control-Request-Method` qui va contenir la méthode de la requête demandée.

Le serveur doit ensuite envoyer une réponse qui contiendra des entêtes que le navigateur analysera pour décider si il effectuera la requête ou renverra une erreur :

- `Access-Control-Allow-Header` qui va permettre au navigateur de comparer.
- `Access-Control-Allow-Headers` qui doit contenir tous les headers demandés.
- `Access-Control-Allow-Method` qui doit contenir la méthode de la requête demandée.
- `Access-Control-Max-Age` qui permet de spécifier un nombre de secondes pour la mise en cache par le navigateur de la politique CORS du serveur spécifié dans les entêtes précédents. Ainsi, le navigateur n'aura pas à faire de requêtes préliminaires si elles respectent les règles spécifiées par le serveur.

Si tous les entêtes correspondent alors le navigateur effectuera la requête, sinon il ne la fera pas.

5- Annuler des requêtes en cours



Les AbortController

Un AbortController permet d'annuler une requête en cours.
Il fait partie des Web API natives.

Créer un AbortController

```
const controller = new AbortController;
```

Les propriétés des AbortController

Un AbortController n'a qu'une propriété `signal` et une méthode `abort()`.

La propriété `signal` retourne un objet `AbortSignal` qui peut être utilisé pour annuler une requête en cours.

Lorsque la méthode `abort()` est appelée, l'événement `abort` est émis sur la propriété `signal` du contrôleur et la propriété `controller.signal.aborted` devient `true`.

Lorsqu'une requête est annulée, une erreur `AbortError` est levée.

Exemple

6- Les objets FormData



L'objet FormData

FormData est un objet natif, il s'agit d'une Web API des navigateurs, qui permet d'envoyer un ensemble de paires clé / valeur et notamment des fichiers.

Les données sont envoyées au format `multipart/form-data` et l'entête `Content-Type` doit contenir `multipart/form-data`.

Le format `multipart/form-data`

Ce format est utilisé pour les formulaires qui envoient des fichiers, des données non ASCII et des données binaires.

Envoyer des données FormData

Pour envoyer un objet FormData rien de plus simple avec `fetch()` :

```
const response = await fetch('/test', {  
  method: 'POST',  
  body: formData  
})
```

`fetch` va automatiquement détecter qu'il s'agit de données FormData. Il va automatiquement encoder les données au format `multipart/form-data` (ajouter les bons entêtes etc).

Exemple

6- Les objets FormData



Les méthodes disponibles

Il existe plusieurs méthodes permettant de manipuler des FormData :

- `formData.append(nom, valeur)` : permet d'ajouter un champ à l'objet avec le nom et la valeur indiqués.
- `formData.set(nom, valeur)` : permet d'ajouter un champ à l'objet avec le nom et la valeur indiqués et de supprimer tous les autres champs avec le même nom. Cela permet de s'assurer qu'il n'y a aucun autre champ avec le même nom sur l'objet.
- `formData.delete(nom)` : permet de supprimer le champ sur l'objet ayant le nom indiqué.
- `formData.get(nom)` : permet de récupérer le champ sur l'objet ayant le nom indiqué.
- `formData.has(nom)` : permet de vérifier que le champ sur l'objet ayant le nom indiqué est présent, la méthode retourne un booléen.
- ...

Exemple

7- La Web API URL



L'objet `URL` est un objet qui permet de gérer facilement des URL en ayant des propriétés et des méthodes adaptées.

Pour le créer, il suffit de taper le code suivant:

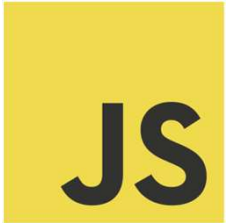
```
const url = new URL("https://wikipedia.fr");
```

Les propriétés des objets `URL`

- `hash` : chaîne de caractères contenant un # suivi de l'identifiant du fragment de l'URL (exemple : #test dans <https://wikipedia.fr/#test>).
- `hostname` : chaîne de caractères contenant le domaine (exemple : wikipedia.fr pour <https://wikipedia.fr/test>).
- `host` : chaîne de caractères contenant l'hôte c'est à dire l'hostname et le port (exemple : wikipedia.fr:3000 pour <https://wikipedia.fr:3000/test>).
- `protocol` : chaîne de caractères contenant le protocole suivi d'un : (exemple : https: pour <https://wikipedia.fr:3000/api/test>).
- `search` : chaîne de caractères contenant les paramètres (exemple : ?user=paul&gender=m pour <https://wikipedia.fr:3000/api/test?user=paul&gender=m>).
- `pathname` : chaîne de caractères contenant le chemin (exemple : /api/test pour <https://wikipedia.fr:3000/api/test>).
- ...

Exemple

7- La Web API URL



La propriété searchParams

La propriété `searchParams` est particulière car elle ne retourne pas une chaîne de caractères mais un objet `URLSearchParams`.

Elle permet de créer des paramètres de recherche et de les encoder correctement.

Les URL doivent respecter le standard `RFC 3986` : les espaces, les caractères non latin et certains caractères spéciaux doivent être encodés en UTF-8, c'est-à-dire que le caractère est remplacé par le code UTF-8.

Par exemple un espace sera encodé en `%20` car son code UTF-8 est 20 en hexadécimal.

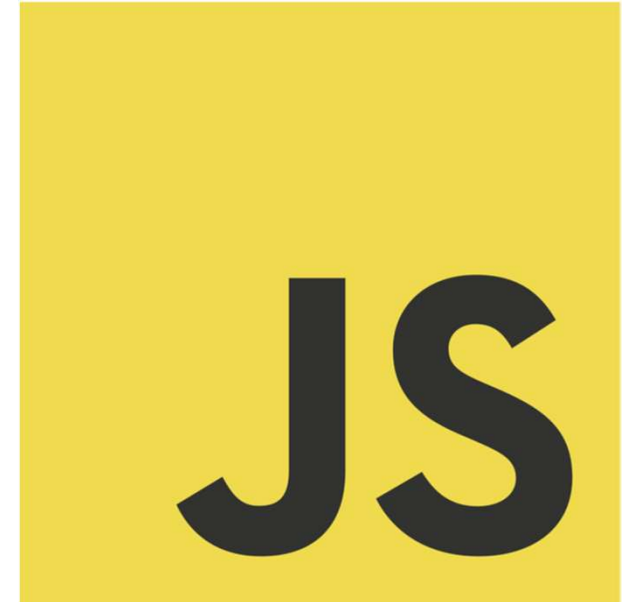
Méthodes disponibles sur l'objet URLSearchParams

- `append(nom, valeur)` permet d'ajouter le paramètre de recherche à l'URL avec le nom et la valeur spécifiés.
- `set(nom, valeur)` permet d'ajouter, ou de remplacer s'il existe, le paramètre de recherche à l'URL avec le nom et la valeur spécifiés. Cela permet de s'assurer qu'il y a un seul paramètre avec le nom.
- `delete(nom)` permet de supprimer le paramètre de recherche de l'URL avec le nom spécifié.
- `get(nom)` permet de récupérer le premier paramètre de recherche de l'URL avec le nom spécifié.
- `has(nom)` permet de vérifier si le paramètre de recherche ayant le nom spécifié est présent sur l'URL.

Exemple

7- La Web API URL

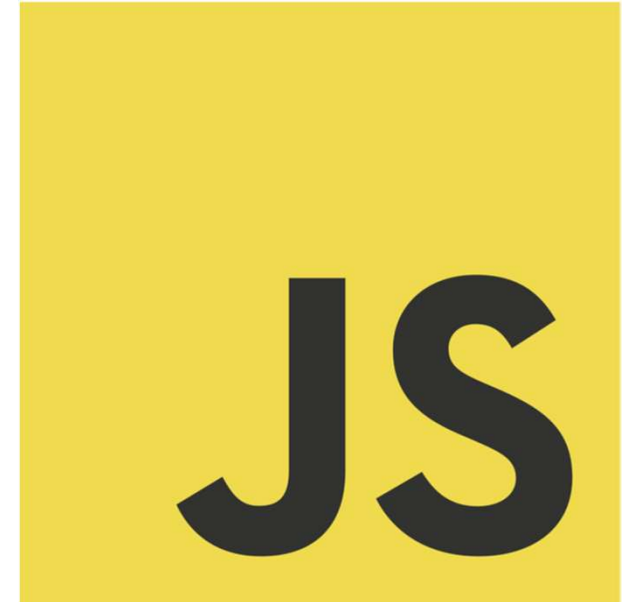
- Exercice:
 - A partir de l'url suivante :
<https://fr.wikipedia.org/wiki/JavaScript>
 - Récupérer les infos suivantes et affichez les dans la console:
 - Le nom de domaine
 - Le chemin
 - Le protocole
- Durée: 5 min



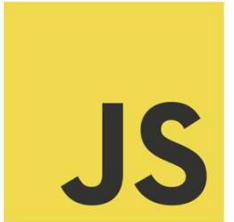
7- La Web API URL

- Correction

Exercice



8- XMLHttpRequest



L'API XMLHttpRequest

La Web API XMLHttpRequest permet également d'effectuer des requêtes AJAX comme fetch.

Il s'agit d'une API plus ancienne qui n'utilise pas les promesses.

Dans la majorité des cas il est conseillé d'utiliser l'API fetch.

Une exception est le besoin d'obtenir l'état d'avancement d'un upload qui n'est pas encore possible avec fetch.

Étapes d'une requête XMLHttpRequest

Étape 1 - Pour commencer il faut créer un objet XMLHttpRequest :

```
const xhr = new XMLHttpRequest();
```

Étape 2 - Il faut ensuite paramétrer la requête avec la méthode open() :

```
xhr.open('GET', 'https://jsonplaceholder.typicode.com/todos');
```

Étape 3 - Il faut ensuite l'envoyer

```
xhr.send();
```

Exemple

8- XMLHttpRequest



Étapes d'une requête XMLHttpRequest suite

Étape 4 - Il faut écouter les événements sur la requête pour les gérer. Les principaux sont load, error et progress.

- L'événement `progress` permet de suivre la progression du transfert de données.
- L'événement `load` permet de savoir lorsque la requête est terminée.
- L'événement `error` permet de savoir si il y a eu une erreur lors de l'envoi.
- L'événement `abort` permet d'être notifié si la requête est annulée.

Les types de réponse

Vous pouvez spécifier le type de la réponse attendue, elle sera alors automatiquement parsée au bon format

Par défaut, si vous ne précisez pas, la réponse sera considérée au format text et vous aurez donc une chaîne de caractères.

Exemple:

```
xhr.responseType = 'json';
```

Il faut bien sûr préciser la réponse avant l'envoi de la requête.

Définir les entêtes de la requête

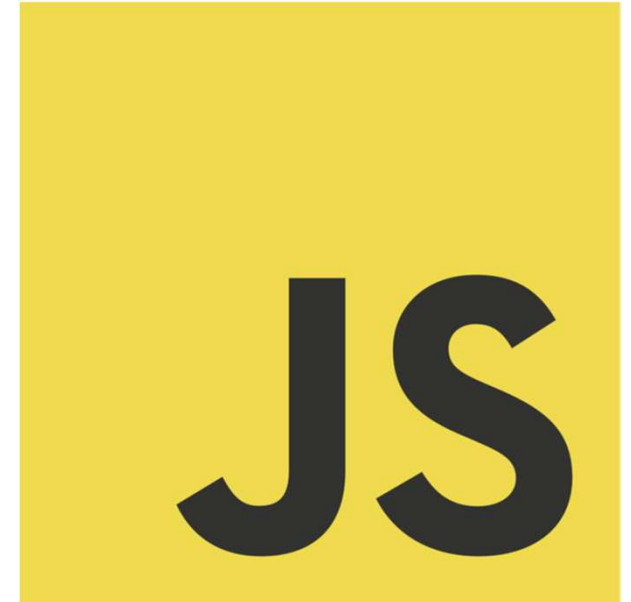
Vous pouvez définir les entêtes de la requête en utilisant la méthode `setRequestHeader()`

```
xhrPost.setRequestHeader('Content-Type', 'application/json');
```

Exemple

8- XMLHttpRequest

- Exercice:
 - Récupérer une liste de todo au format json puis afficher dans la console
- Durée: 10 min



8- XMLHttpRequest

- Correction

Exercice

