

JavaScript – Programmation avancée

03/06/2024 – 05/06/2024

Glodie Tshimini



m2information.fr



Glodie Tshimini

Consultant Formateur et développeur Web depuis 2017

Développeur fullStack en Freelance

Développeur fullStack à l'INRAE

Tech Lead agence digitale Comkwatt Saint-Raphaël

Certifié Coderbase IT Expert Trainer

Certifié Coderbase IT Expert Trainer At POE

Certifié PSD I

Email : contact@tshimini.fr



Présentations

Merci de bien vouloir vous présenter individuellement.

HORAIRES ET PAUSES

Lundi

- 9h00 - 12h30
- 13h30 - 17h30

Mardi

- 09h00 - 12h30
- 13h30 - 17h30

Mercredi

- 09h00 - 12h30
- 13h30 - 16h00

Les pauses

- Matin : 15 minutes
- Déjeuner : 1 heure
- Après-midi : 15 minutes



Émargement et évaluation formateur

Chaque matin et chaque après-midi, vous devrez signer les feuilles d'émargement. Les "X" et initiales ne sont pas autorisés, merci de bien vouloir émarger avec une signature.

Le dernier jour de la formation, avant 14h00, vous aurez à saisir une évaluation formateur. Cette évaluation est obligatoire, il doit être rempli consciencieusement.

OBJECTIFS PÉDAGOGIQUES

- Utiliser tous les outils de débogage à disposition
- Décrire les contextes d'exécution
- Structurer le code JavaScript en modules
- Implémenter les concepts objets en JavaScript et les concepts fonctionnels
- Identifier les aspects avancés des "closures" et les promises
- Mémoriser jQuery
- Identifier les différences avec Node.js et expliquer le rôle de chacun.

NIVEAU REQUIS ET PUBLIC CONCERNÉ

Niveau requis

- Avoir suivi le cours [JVS-IN "JavaScript - Fondamentaux"](#)
- Ou avoir une connaissance pratique du langage JavaScript ainsi que des connaissances de HTML5 et CSS3

Public concerné

- Développeurs
- Architectes
- Chefs de projets techniques.

Déroulement de la formation

- 40 % théorie 60% pratique
- Daily meeting
- Exercices immédiats d'application du cours
- Les énoncés des exercices et corrections seront accessibles depuis le [dépôt public GitHub](#)

GitHub de la formation



<https://github.com/glo10/jvs-av-03062024>

Un formateur à votre écoute



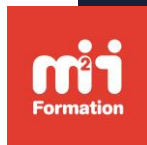
Google Forms

<https://docs.google.com/forms/d/e/1FAIpQLSdbXxnWyjWmc3u05V4rHvvqi2mMHoMUBlaG7HjAVIuv65gEBw/viewform>

Plan du cours



- I. Rappels POO et DOM
- II. Modules
- III. Débogage
- IV. Asynchronie
- V. API HTML5
- VI. JQuery
- VII. NODE JS



I. RAPPELS POO ET DOM



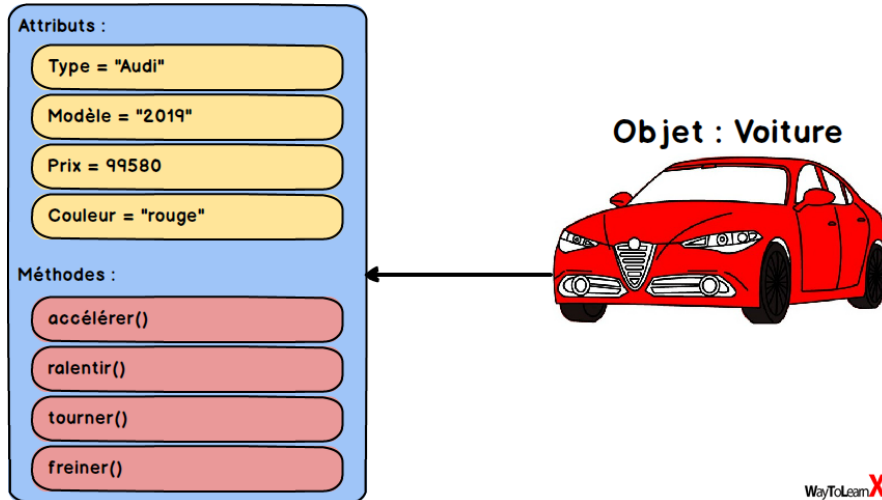
m2iinformation.fr

Généralités



Rappels programmation orientée objet

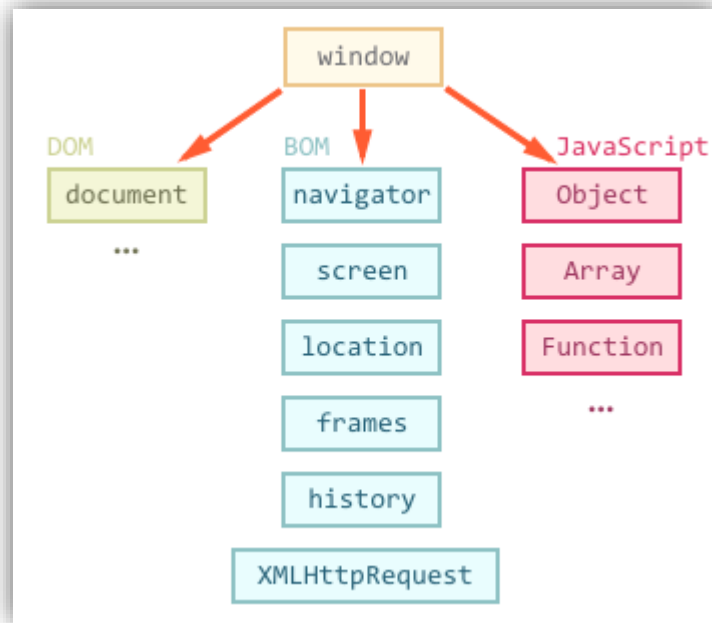
[Source image waytolearnx.com](http://waytolearnx.com)



- La POO est une méthode de programmation informatique née dans les années 1960.
- **Collaboration** entre des objets pour résoudre un problème ou un besoin.
- Un **objet** est une entité qui possède des **caractéristiques** et des **comportements**.
- Les caractéristiques ou propriétés ou données constituent un ensemble d'attributs définissant l'apparence de l'objet.
- Les comportements ou méthodes constituent un ensemble d'actions réalisables par l'objet.

En JavaScript, tout est objet

- DOM : objets spécifiques à la manipulation du document HTML.
- **BOM**: objets spécifiques à la manipulation du navigateur.
- APIs JavaScript : tous les autres objets
- La notation pointée (avec le point) permet d'accéder aux objets, propriétés, méthodes de même hiérarchie ou inférieure.
 - Par exemple *window.location.href* permet de récupérer l'URL de la page courante.



[Source image medium.com/@fknuessel](https://medium.com/@fknuessel)

Opérateur this et la portée des variables



Opérateur this

- Mot clé qui permet de faire référence à l'objet courant
- Selon les contextes cet objet courant peut-être
 - Dans le contexte globale, c'est l'objet **window**
 - Dans le contexte d'une fonction ou d'un **objet**, c'est la fonction ou l'objet lui-même
 - Dans le contexte d'un événement, c'est l'objet **event** que l'on verra plus tard

Exemple

```
36 ∨ const glodie = {
37   firstName: 'Glodie',
38   myself: function () {
39     return this
40   },
41   mySelfArrow: () => {
42     return this
43   }
44 }
45 console.log(this === window) // true
46 console.log(glodie === glodie.myself()) // true
47 ∨ console.log(
48   glodie.mySelfArrow(), // retourne window (objet global englobant)
49   glodie === glodie.mySelfArrow() // false car window !== glodie
50 )
```

Portée (scope) des données

- Avec beaucoup de simplification, le fonctionnement des scopes s'effectue avec des blocs repérés par `{}`
- Une variable est *locale*, si elle est définie et accessible dans le bloc dans laquelle elle a été déclarée. On va privilégier le mot clé *let* ou *const* pour les variables locales.
- Une variable est dite *globale*, lorsqu'elle est accessible dans les autres blocs « internes » du bloc dans laquelle elle a été déclarée.
- Le mot clé *var* a une portée associée à la *fonction* ou une portée *globale*
- [Documentation sur let et les scopes](#)
- [Document avec var et les scopes](#)

Schématisation de la portée [Source image dasha.ai](https://dasha.ai)

```
const userName = "Peter";

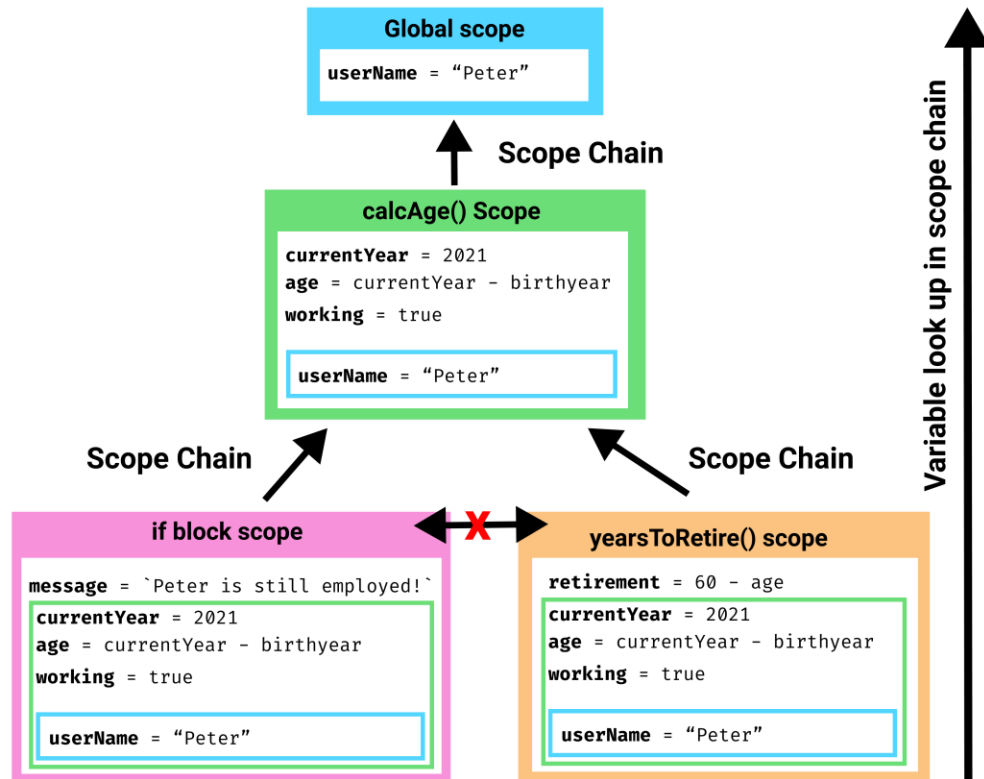
function calcAge(birthyear) {
  const currentYear = 2021;
  const age = currentYear - birthyear;

  if (age ≤ 60) {
    var working = true;
    const message = `Peter is still employed!`;
    console.log(message);
  }

  function yearsToRetire() {
    const retirement = 60 - age;
    console.log(`${userName} will be retired in ${retirement} years!`);
  }

  yearsToRetire();
}

calcAge(1975);
```



Schématisation de la portée

Source image reddit

keyword	const	let	var
global scope	NO	NO	YES
function scope	YES	YES	YES
block scope	YES	YES	NO
can be reassigned	NO	YES	YES

Source image levelup

Var	Let	Const
Function Scope	Block Scope	Block Scope
Reassignment	Reassignment	Reassignment
Redeclaration	Redeclaration	Redeclaration

INSTALLATIONS





Guide d'installation depuis GitHub

Suivez les instructions du fichier d'installation de notre environnement de travail depuis le ***README.md*** le dépôt GitHub du cours

Nouveautés ES6 sur les objets et classes



Focus nouveautés ES6 sur les classes et objets

- *Import*: import d'un module avec le mot-clé *import* au lieu de *require()*.
- *Export*: export d'un module avec le mot-clé *export* au lieu de *modules.export=a*.
- *Class*: structure similaire aux autres langages objets.
- *Objet littéral*: création des objets avec des accolades `{}`
- *Notation fléchée* des fonctions =>

```
hello() => { console.log('hello world') }
```
- *Déstructuration* des tableaux et objets : créer des variables à partir des propriétés d'un objet ou des éléments d'un tableau

Exemples nouveautés ES6

Class

```
// BEFORE ES6
var Car = function (color) {
  this.color = color
}
Car.prototype.accelerate = function() {
  // Do something here
}
Car.prototype.brake = function() {
  // Do something here
}
modules.export = Car

/*-----*/
// WITH ES6
class Car {
  constructor(color) {
    this.color = color
  }
  accelerate () {
    // Do something here
  }
  brake() {
    // Do something here
  }
}
export default Car
```

Déstructuration

```
const cars = ['twingo', 'clio', 'megane', 'zoe']
const [twingo, clio, ...others] = cars

console.log(twingo) // twingo
console.log(clio) // clio
console.log(others) // ['megane', 'zoe']

const zoe = {
  color: 'black',
  engine: 'electric',
  odomter: 45000
}

const {color} = zoe
console.log('zoe color', color) // black
```

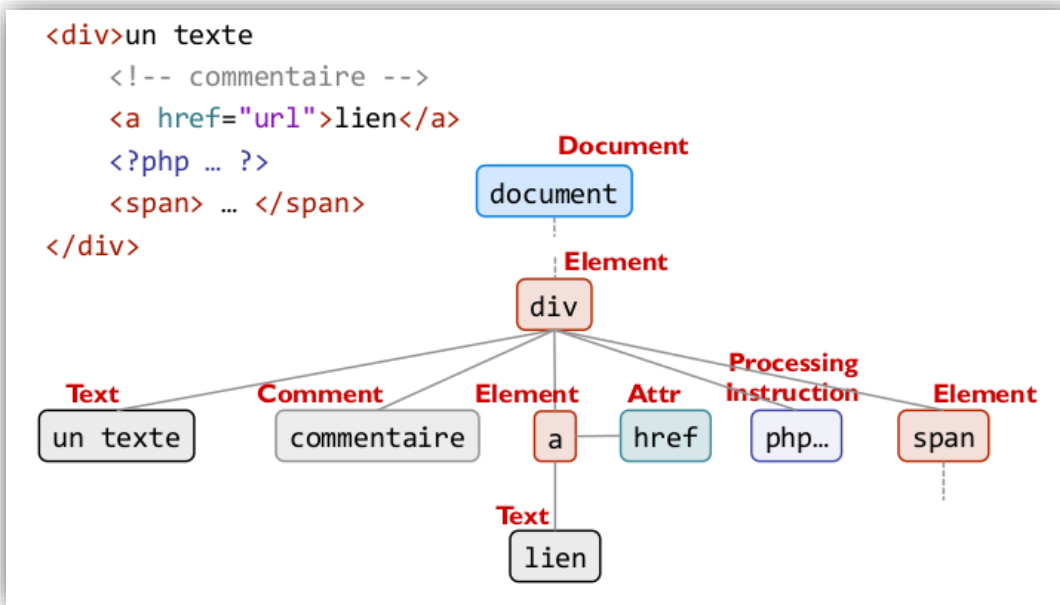
Objet littéral

```
const Car = {
  color: undefined,
  accelerate: () => {
    // Do something here
  },
  brake: () => {
    // Do something here
  },
}
```



DOM : Document Object Model

- Le DOM est une API permettant de représenter et manipuler les éléments constituant une page Web.
- A l'aide des méthodes offertes par l'objet document de JavaScript, une page web peut être modifiée.



Accéder aux éléments du DOM

- Plusieurs méthodes permettent d'accéder à un ou plusieurs éléments du DOM à partir de leur ID, nom de la balise, nom de la classe etc.
- Nous retiendrons dans le cadre de ce cours uniquement les méthodes [querySelector\(\)](#) et [querySelectorAll\(\)](#) qui permettent de tout sélectionner à partir des sélecteurs CSS.
 - *`document.querySelector(selector)`*: retourne le premier élément du document html qui correspond au sélecteur *selector*
 - *`document.querySelectorAll(selector)`*: retourne tous les éléments du document HTML qui correspond au sélecteur *selector* sous forme de tableau.

Les autres méthodes et attributs du DOM

- [*el.createElement\(tag\)*](#) : crée l'élément à partir du tag donné.
- [*el.insertAdjacentHTML\(position, el\)*](#) : insère un nouveau nœud HTML dans l'élément *el*/par rapport à la position spécifiée.
- Position prend les valeurs suivantes :
 - *Beforebegin* : avant l'élément lui-même ;
 - *Afterbegin* : juste à l'intérieur de l'élément, avant son premier enfant ;
 - *Beforeend* : juste à l'intérieur de l'élément, après son dernier enfant ;
 - *Afterend* : après l'élément lui-même.
- [*el.parentElement*](#) : renvoie le parent du nœud (textuel ou html) ou null.
- [*el.replaceWith\(nodeEl\)*](#) : remplace l'élément courant *el* par le nœud *nodeEl*.
- [*el.firstElementChild\(\)*](#) : renvoie le premier nœud enfant de type *element* ou *null*.

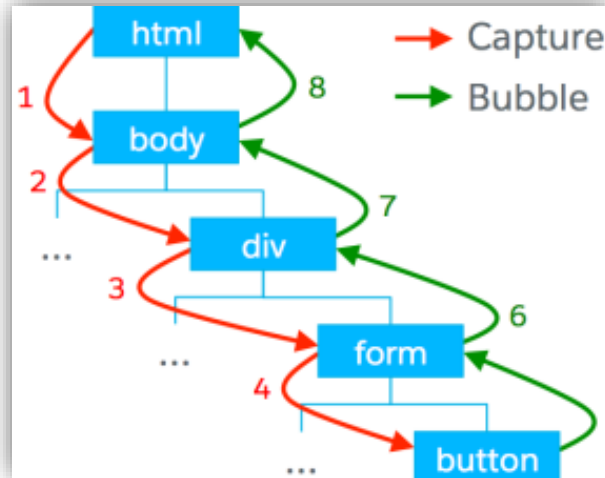
[Source MDN](#)

Propagation des événements

- Le troisième argument de la méthode [addEventListener](#)(*event, callback, boolean*) détermine le flux d'événement.
(*true* => *capture*, *false* => *bubble*).
- Phase de **bouillonnement** (Bubble) : propagation de l'événement en remontant la hiérarchie du DOM.
Comportement par **défaut**.
- Phase de **capture** : propagation de l'événement en **descendant** la hiérarchie du *DOM*.

Propagation des événements

- [e.preventDefault\(\)](#): empêche le comportement par défaut d'un élément de s'exécuter.
- [e.stopPropagation\(\)](#): stoppe la propagation de l'événement



[Source de l'image laptrinhx](#)

Objet event

- À chaque définition d'un écouteur d'événement à l'aide de la méthode `.addEventListener()`, la fonction callback peut prendre un argument en paramètre qu'on nomme communément *e*.
Cet argument est un objet qui contient des propriétés et méthodes correspondant à l'événement qui a été déclenché.
- Les événements de type *click*, *mouse* ou *keyup* n'auront pas les mêmes attributs.
- Nous utiliserons dans la plupart du temps l'attribut ***e.target.value*** pour obtenir la valeur de l'élément HTML sur lequel un événement a été greffé.

Vous pouvez utiliser la fonction [console.dir\(e\)](#) ou *log* pour afficher dans la console toutes les informations disponibles d'un événement e.

```
click { target: h1, buttons: 0, clientX: 34, clientY: 51, layerX: 34, layerY: 51 }
  altKey: false
  bubbles: true
  button: 0
  buttons: 0
  cancelBubble: false
  cancelable: true
  clientX: 34
  clientY: 51
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 1
  eventPhase: 0
  ▶ explicitOriginalTarget: <h1>
  isTrusted: true
  layerX: 34
  layerY: 51
  metaKey: false
  movementX: 0
  movementY: 0
  mozInputSource: 1
  mozPressure: 0
  offsetX: 0
  offsetY: 0
  ▶ originalTarget: <h1>
    pageX: 34
    pageY: 51
    rangeOffset: 0
    rangeParent: null
    relatedTarget: null
    returnValue: true
    screenX: 34
    screenY: 193
    shiftKey: false
  ▶ srcElement: <h1>
  ▶ target: <h1>
    timeStamp: 4792
    type: "click"
  ▶ view: Window http://127.0.0.1:5500/demo/index.html
    which: 1
    x: 34
    y: 51
```

Les événements

Rendre les pages web plus interactives.

1. Sélectionnez un élément HTML.
2. Ajoutez un écouteur événement de type : souris (*click*, *dblclick*, *mouseover*, *mouseout* etc.), formulaire (*focus*, *blur*, *change*, *submit*), clavier (*keydown*, *keyup*, *keypress* etc).
3. Ajoutez une fonction callback qui sera exécutée au moment où l'événement aura lieu.

```
const btn = document.querySelector('button')
btn.addEventListener('click', (e) => {
  console.log('Hi, i am the callback function')
  console.log('e for event, object that contains the details of an event ', e)
})
```

Les événements propres à un formulaire

- **Input**, événement déclenché lorsqu'une valeur est saisie dans le champ
- **Change**, déclenché lorsque la valeur d'un champ change
- **Focus**, déclenché lorsqu'il y a le focus sur le champ
- **Blur**, déclenché lorsqu'il y a une perte du focus sur le champ
- **Submit**, déclenché à la soumission du formulaire
- **Reset**, déclenché lorsque les données d'un formulaire sont tous supprimées (un reset)


Learn JavaScript

DOM Events Cheat Sheet

Event Listeners

```

// register event listener
document.addEventListener('click', (event) => {
  console.log('Click Event', event);
});

// unregister event listener
document.removeEventListener('click', (event) => {
  console.log('Unregistered Event', event);
});

```

Mouse Events

click	left mouse button click
dblclick	left mouse button double click
mousedown	pointing device button is pressed when inside element
mouseup	mouse button is released over an element
mouseover	mouse pointer enters an element
mousemove	mouse pointer moves over an element

Keyboard Events

keydown	key is pressed down
keyup	key is released

Form Events

blur	element has lost focus
change	user modifies value of <input>, <select> or <textarea>
focus	element has received focus
select	text has been selected in an element
submit	fires on <form> when submitted
reset	fires when form is reset

Window Events

abort	resource was not fully loaded, but not due an error
error	error event occurs if resource failed to load or can't be used
load	document has finished loading
unload	document is being unloaded
scroll	document is scrolled
resize	window is resized


David Mráz
@davidm_ai


learning.atheros.ai

Demo : événements

[Référence de tous les événements](#)

EXERCICES



Exercice 1



0-exercices/1-ex/README.md

Ce qu'il faut retenir

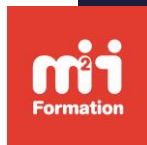


SAVE

- Le DOM (Document Object Model) permet de manipuler du HTML et CSS à travers JavaScript (ajout d'éléments, d'événements, modifications des propriétés CSS etc.).
- La programmation orientée objet (POO) permet de solutionner un problème informatique à l'aide de la collaboration de plusieurs objets ayant des propriétés (caractéristiques) et des méthodes (comportements).
- POO facilite le partage, la maintenance et l'évolution du code.
- La programmation événementielle repose sur la manipulation des éléments du DOM.



- Codez en utilisant les normes (notations) *ECMAScript 6 (ES6 – ES2015)* et +



II. Modules



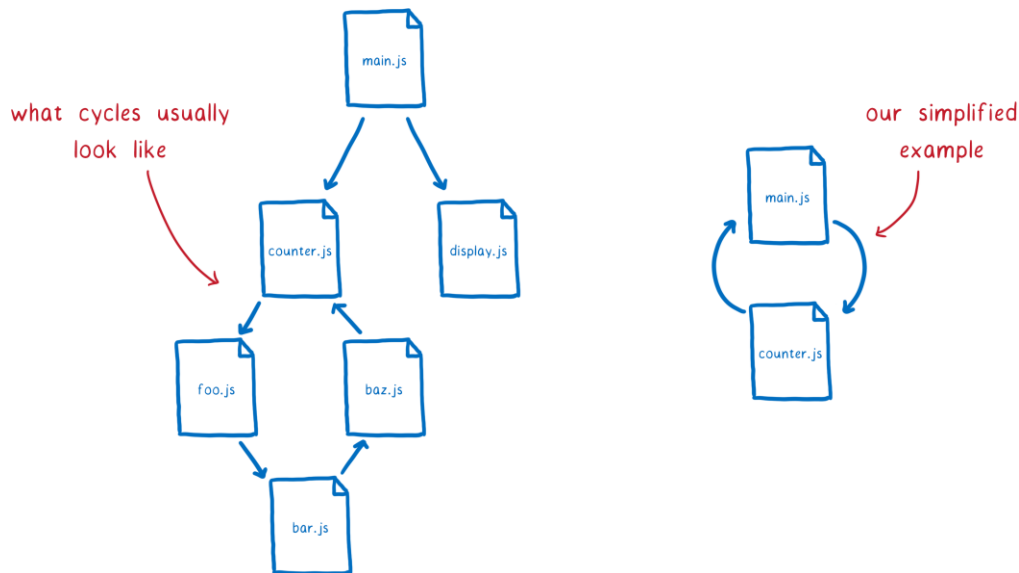
m2iinformation.fr

Définition



Qu'est ce qu'un module ? ([source image tech mozfr](#))

- Un module est un fichier JavaScript contenant du code à partager ou à réutiliser dans d'autres scripts JS.



- Plusieurs standards d'utilisation existent en JavaScript : *CommonJS* et *ESM* sont les deux les plus utilisés.

Standards



Différences entre CommonJS et ESM

Les Bundlers (Esbuild, Parcel, WebPack, Vite.js, etc.) permettent de transformer du code JavaScript afin d'éviter les problèmes d'incompatibilité entre modules et d'interprétation des nouvelles syntaxes par les anciens navigateurs.

	CommonJS	ESM
Standard		ES6
Import	<i>require('path/file.js')</i>	<i>import { add } from 'path/file.js'</i>
Export	<i>modules.export = add</i>	<i>export add</i>
Extension	<i>.cjs ou .js</i>	<i>.mjs ou .js</i>
Mode de chargement	synchrone	asynchrone et synchrone
Chargement en HTML côté navigateur	<i><script type="script" src=" ... "></script></i> et utiliser un bundler pour convertir le code	<i><script type="module" src=" ... "></script></i>

Différences entre CommonJS et ESM ([source image lenguajejs](https://lenguajejs.com))

CJS

CommonJS

```
const elem = require('module');

module.exports = {
};
```



ESM

ES Modules

```
import { elem } from './module.js';

export const elem = {
};
```



LenguajeJS.com

Avantages et inconvénients

Avantages

- ✓ Logique divisée en plusieurs fichiers
- ✓ Moins de responsabilité
(responsabilité unique)
- ✓ Traitement propre à un besoin
- ✓ Plus facile à partager et à réutiliser
- ✓ Plus facile à tester
- ✓ Code plus court et plus lisible

Inconvénient

- X Non compatibilité avec tous les navigateurs (anciennes versions)
- X Nécessite un compilateur pour transformer du code JavaScript avec les nouveaux standards en code JavaScript anciens compatibles avec tous les navigateurs
- X Débogage un peu plus difficile à partir du code compilé

Gestionnaire de package



Le gestionnaire de packages NPM



- *Node Package Manager* : gestionnaire de paquets de Node.
- Permet de **télécharger** et de **déployer** les **modules** développés par la communauté des développeurs JavaScript.
- Initialisation d'un nouveau projet : `npm init`
 - Répondre aux questions posées depuis le terminal.
 - A la fin du processus, les fichiers *package.json* et *package-lock.json* contenant les informations requises et les dépendances sont générés.
- Installation de toutes les dépendances d'un projet contenu dans le fichier *package.json* : `npm install`

Le gestionnaire de packages NPM

- *Installation d'un module*
 - De manière globale : `npm install -g node`
 - De manière locale : `npm install --save typescript`
 - Une dépendance de développement en locale : `npm install -D axios`
- Désinstallation d'un module : `npm uninstall node`
- Mise à jour d'un module : `npm update [--save/--save-dev/-g] typescript`
- [Déploiement d'un module sur npmjs.com](#)
- Il existe plusieurs autres gestionnaire de packages pour Node tels que
 - [Yarn de Facebook](#)
 - [PNPM](#)

Quelques modules



Quelques autres modules utiles

- [ESLINT](#): analyse statique du code JS avec un fichier de configuration.
- [StandardJS](#): vérification syntaxe JS sans fichier de configuration (alternatif à *ESLINT*).
- [Vitest](#) ou [Jest](#): pour les tests unitaires.
- [Cypress](#): pour les tests end to end.
- [TypeScript](#): superset (surcouche) de JavaScript qui apporte des fonctionnalités supplémentaires tels que le typage, les classes abstraites, etc.
- [Concurrently](#): lancer plusieurs commandes simultanément.
- [Icons.js.org](#): n'est pas un module, c'est un hub regroupant plusieurs icônes de plusieurs fournisseurs différents

Demo : Modules

Exercice 2



0-exercices/2-ex/README.md

EXERCICES



Exercice 2



0-exercices/2-ex/README.md

Ce qu'il faut retenir



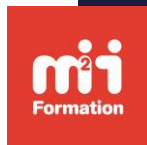
SAVE

- *Les modules facilitent le partage et la réutilisation d'une partie de son code.*
- *NPM* et *YARN* sont des gestionnaires de paquets de JavaScript permettant d'ajouter des modules externes dans son projet.
- Les *Bundlers* permettent d'écrire un code à partir d'un standard et le compiler en sortie dans un autre. A l'aide de plugins intrinsèque ou complémentaire, on peut minifier, compresser, transpiler, etc.

Conseils du formateur



- Codez de façon modulaire.
- Utilisez les modules de la communauté pour éviter de réinventer la roue.
- Créez vos propres modules pour les partager au sein de la communauté ou en intra.



III. Débogage

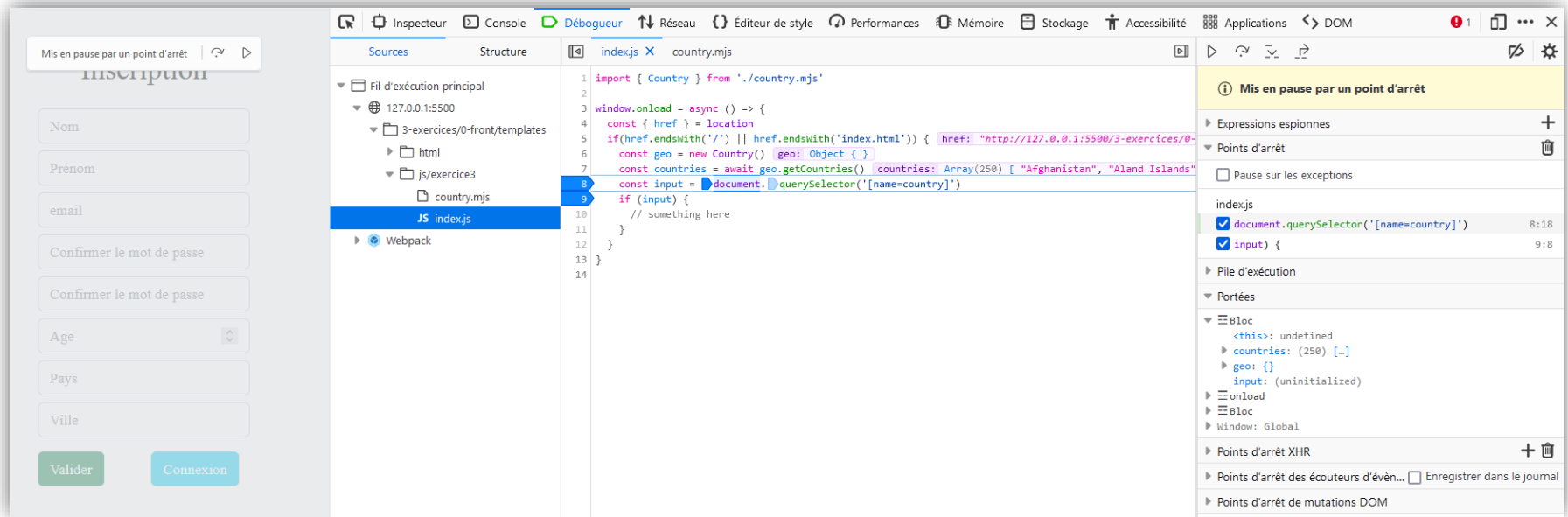


m2iinformation.fr

- Parmi les outils de développement (devTools), la console permet d'exécuter du code JavaScript en direct et de voir les messages d'information, de warning ou d'erreur sur le CSS et le JavaScript.
- Principalement utilisé pour déboguer le DOM et analyser les erreurs JS.
- Tous les outils de la *devTools* sont accessibles à partir de :
 - Windows/Linux avec la touche F12 ou CTRL + Shift + C ;
 - MacOS avec CTRL + CMD + C .

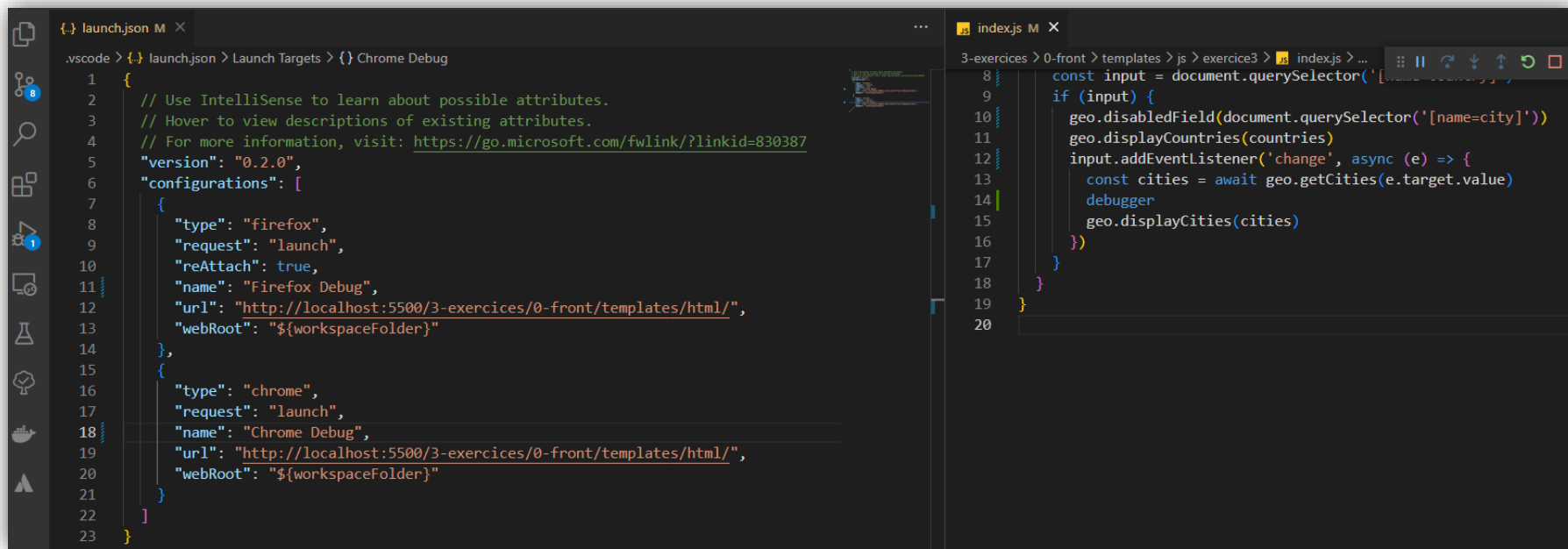
Débogueur

- Outil de la devTools permettant de suivre l'exécution de son code ligne par ligne.
- Avec le suivi pas à pas, vous pouvez voir en temps réel les valeurs de vos variables.
- Vous pouvez également arrêter votre script à des endroits précis.



Déboguer depuis l'éditeur du code

- Extension VSCode debugger.
- Configuration des navigateurs dans le fichier `.vscode/launch.json`
- Depuis votre code, vous pouvez ajouter le mot-clé *debugger* qui permet de stopper l'exécution du code à l'endroit à un endroit précis ou les boutons dédiées aux débogages



The screenshot shows the VS Code interface with two files open. The left file is `launch.json`, which is a JSON configuration for the VS Code debugger. It defines two launch targets: one for Firefox and one for Chrome. The right file is `index.js`, which contains JavaScript code. A `debugger` statement is placed on line 14, just before the `geo.displayCities(cities)` call. The code in `index.js` is as follows:

```

8  const input = document.querySelector('[name=city]');
9  if (input) {
10     geo.disabledField(document.querySelector('[name=city]'));
11     geo.displayCountries(countries);
12     input.addEventListener('change', async (e) => {
13         const cities = await geo.getCities(e.target.value);
14         debugger;
15         geo.displayCities(cities);
16     });
17 }
18 }
19 }
20

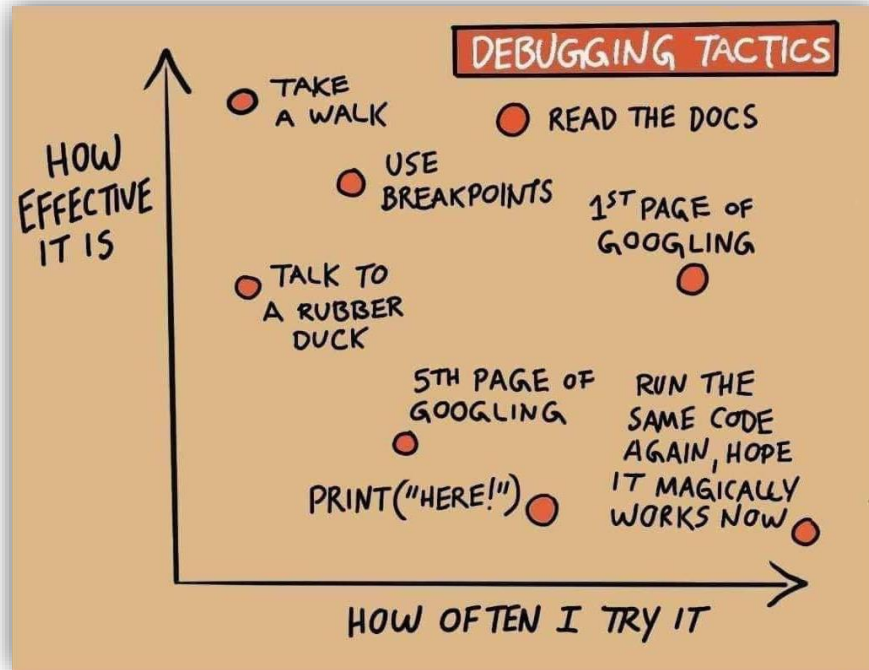
```

- Le réseau est l'outil de la *devTools* consacré à l'analyse des requêtes *HTTP* effectuée par une page web, pour charger les différentes ressources telles que les fichiers CSS, JS, images etc.
- Grâce à cet onglet, vous pouvez également voir le statut (code *HTTP* de réponse) de retour, le type de fichier, la taille et la durée de chaque élément ainsi que la durée totale.

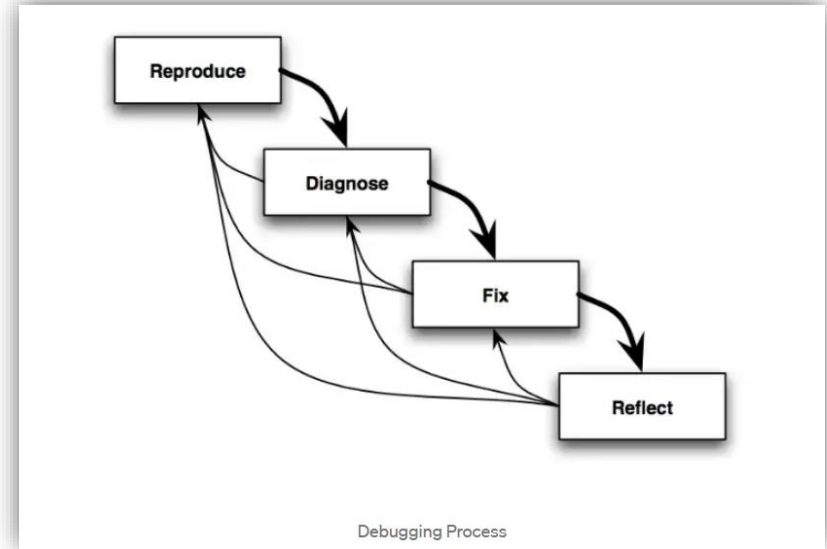
Stockage

- Onglet de la devTools permettant d'explorer les types de stockage supportés par le navigateur, ainsi que les données enregistrées pour chaque type de stockage.
- Des solutions de stockage local (dans le navigateur)
 - [LocalStorage](#)
 - [SessionStorage](#)
 - [Cookie](#)
 - [IndexedDB](#)

Méthodes de débogage



[Source de l'image codetravelled](#)



[Source de l'image medium.com/@gonzalookh](https://medium.com/@gonzalookh)

Demo : débogage avec Visual Studio Code

Ce qu'il faut retenir



SAVE

- La console sert à explorer les données d'information, de sensibilisation ou d'erreur liées à vos fichiers multimédia, CSS et JavaScript.
- Le réseau permet de vérifier que tout se charge correctement.
- Le debugger permet de suivre l'exécution de son code pas à pas.
- Le stockage permet de voir tous les types de stockage supportés par le navigateur et les données stockées.
- L'inspecteur permet d'explorer et parcourir l'arborescence du code HTML.

Conseils du formateur



ADVICE

- Utilisez les extensions disponibles sur les éditeurs pour un débogage plus efficace et plus rapide.
- La correction des bugs ou l'évolution du code peut induire d'autres bugs, en testant votre code à l'aide des tests unitaires, d'intégration et/ou fonctionnelle, vous assurez très rapidement qu'il n'y a pas d'effet de bord.

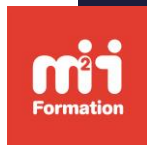
EXERCICES



Exercice 3



`exercices/3-ex/README.md`



IV. ASYNCHRONIE



m2iformation.fr

Généralités



Définition

- JavaScript est **synchrone**, c'est-à-dire que l'exécution de la ligne suivante du script s'effectue uniquement après la fin de l'exécution de la ligne précédente. On dit que le script est **bloquant**.
- L'**asynchrone** détermine un ensemble de technologies permettant d'effectuer des opérations **non-bloquantes**, c'est-à-dire que les autres instructions (tâches) sont exécutées malgré le fait que la tâche en cours ne soit pas terminée.
- Pour désigner l'asynchrone en JavaScript, on emploie communément le terme ***AJAX*** (***Asynchronous JavaScript And XML***), apparu en **2005**, qui regroupe l'ensemble des solutions offertes par le langage JavaScript.

Synchrone VS Asynchrone

Operations to be executed

FETCHING DATA
FROM API

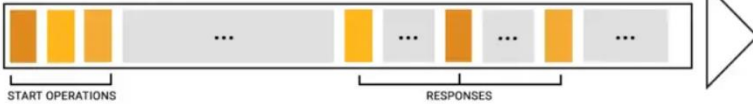
SET TIMEOUT
FOR 5 SECONDS

DATABASE
OPERATION

Synchronous

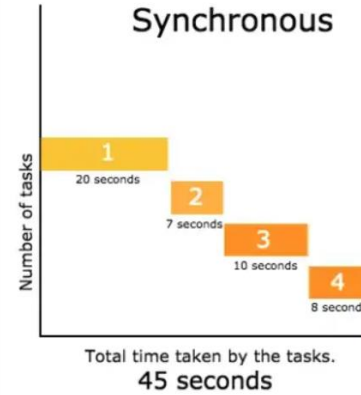


Asynchronous



[Source de l'image JavaScript.plainenglish.io](https://www.javascript.plainenglish.io/)

Synchronous



Asynchronous



[Source de l'image medium.com/@vivianyim](https://medium.com/@vivianyim/)

Intérêts de l'asynchrone

- Recharger partiellement des pages web
- Exécuter des tâches lourdes sans ralentir l'exécution du reste du Script
- Communiquer avec un serveur externe sans interruption ou blocage de la pile d'exécution principale
- Meilleure expérience utilisateur (Navigation plus flexible)
- Temps de chargement du site Web plus court

Initialement les données étaient envoyées/reçues sous le format *XML*. Aujourd'hui le format *JSON* a pris le dessus grâce à ses caractéristiques que nous verrons juste après.

Formats d'échange



Format textuel XML

- **eXtensible Markup Language**. Comme le HTML, c'est un langage de balisages.
Les balises **XML** ne sont pas prédéfinies, l'auteur du fichier doit définir ses propres balises en respectant les normes du langage.
- L'extension du fichier est *.xml*.
- *XML* est principalement utilisé pour stocker des données structurées ou les échanger entre des applications.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <users>
    <item>
      <name>tshimini</name>
      <age>31</age>
      <email>contact@tshimini.fr</email>
      <adresses>
        <city>Paris</city>
        <country>France</country>
      </adresses>
      <adresses>
        <city>Nice</city>
        <country>France</country>
      </adresses>
    </item>
    <item>
      <name>john</name>
      <age>19</age>
      <email>john@doe.com</email>
      <adresses>
        <city>london</city>
        <country>UK</country>
      </adresses>
      <adresses>
        <city>Nice</city>
        <country>France</country>
      </adresses>
    </item>
  </users>
</root>
```

Format textuel JSON

- JSON pour *JavaScript Object Notation*.
- Stocker les informations sous forme de couple de *clé : valeur* ; les valeurs pouvant elles-mêmes être des clés contenant un autre sous-ensemble de clés et valeurs.
- L'extension du fichier est *.json*.
- *Plus léger, plus lisible pour l'homme, plus rapide à traiter et proche de la notation objet de JS.*
- C'est le format de prédilection des échanges des données entre les applications.

```
{
  "users": [
    {
      "name": "tshimini",
      "age" : 31,
      "email": "contact@tshimini.fr",
      "adresses": [
        {
          "city": "Paris",
          "country": "France"
        },
        {
          "city": "Nice",
          "country": "France"
        }
      ]
    },
    {
      "name": "john",
      "age" : 19,
      "email": "john@doe.com",
      "adresses": [
        {
          "city": "london",
          "country": "UK"
        },
        {
          "city": "Nice",
          "country": "France"
        }
      ]
    }
  ]
}
```




Promesse

Promise

- Objet permettant d'effectuer une requête asynchrone depuis ES6.
- La réponse (une promesse) est disponible ou non dans le futur.
- L'objet *Promesse* prend en paramètre deux fonctions *resolve()* et *reject()*.
 - *Resolve()* exécutée en cas de succès (réponse positive de la promesse).
 - *Reject()* exécutée en cas d'échec.
- En cas de succès, la réponse peut être récupérée à l'aide de la méthode *.then()* de l'objet *Promesse*.
- En cas d'échec, l'erreur peut être capturée par la méthode *.catch()* de l'objet *Promesse*.

Implémentation Promise

Implémentation en JS

```
const nb = 11
const myPromise = new Promise((resolve, reject) => {
  if (nb % 2 === 0) {
    resolve('Nombre pair')
  } else {
    reject('Nombre impair')
  }
})

myPromise
  .then((pair) => {
    console.log(pair) // Nombre pair
  })
  .catch(
    error => console.log(error) // Nombre impair
  )
```

Les états et enchaînements de la promesse [Source de l'image scoutapm](#)



Promise all et Promise allSettled

- [Promise.all](#) renvoie une promesse lorsque l'ensemble des promesses données en arguments sous forme de tableau ont été toutes résolues avec succès ou échec.
- Si toutes les promesses sont résolues avec succès, la promesse retournée est un succès, par contre, si une seule échoue, la promesse finale échoue entièrement.
- [Promise.allSettled](#) est similaire à Promise.all, sauf qu'elle renvoie un objet contenant
 - En cas de succès, une propriété ***status*** dont la valeur est « ***fulfilled*** » et une propriété ***value***
 - En cas d'échec, ***status*** vaut « ***rejected*** » et on a une propriété ***reason*** à la place de ***value***

```
var p1 = Promise.resolve(3);
var p2 = 1337;
var p3 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, "foo");
});

Promise.all([p1, p2, p3]).then((values) => {
  console.log(values); // [3, 1337, "foo"]
});
```



Fetch

- Méthode de **prédilection** permettant d'effectuer des traitements asynchrones vers un serveur local ou distant.
- La méthode ***fetch*** renvoie une promesse. En cas de succès, la méthode ***.then()*** de l'API ***fetch*** permet de manipuler le résultat. En cas d'échec, un traitement spécifique peut être effectué dans la méthode ***.catch()***.
- La méthode ***fetch*** prend :
 - En **premier** paramètre, un **URL (obligatoire)** ;
 - En **second** paramètre, un objet contenant des **options**. Dans les options, on peut indiquer la **méthode HTTP**, les informations d'en-têtes (**headers**), le **cache** etc.

Implémentation de l' API fetch

```
fetch(  
  'https://jsonplaceholder.typicode.com/photos',  
  {  
    method: 'GET',  
    headers: new Headers({'Content-Type': 'text/json'}),  
    mode: 'cors',  
    cache: 'default'  
  })  
  .then((res) => res.json())  
  .then((photos) => {  
    console.log('photos : ', photos)  
  })  
  .catch((err) => console.log('error : ', err ))
```

Async / Await



Async / Await

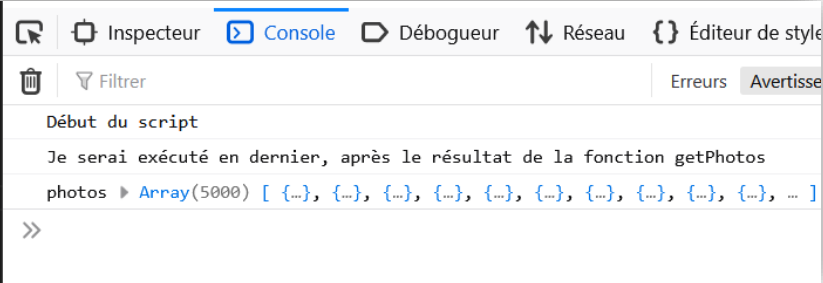
Les mots-clés *async* et *await* fonctionnent de pair. Ces mots-clés permettent d'attendre la résolution d'une promesse avant d'exécuter la suite de votre script. Autrement dit l'exécution de la suite du script est bloquée en attendant d'obtenir le résultat.

Sans async/await

```
console.log('Début du script')

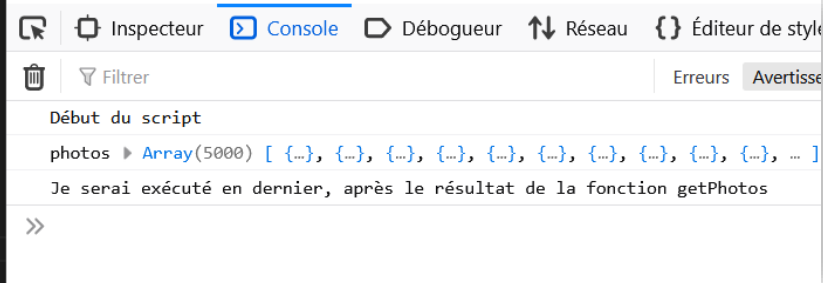
const getPhotos = () => {
  fetch('https://jsonplaceholder.typicode.com/photos')
    .then((res) => res.json())
    .then((photos) => console.log('photos', photos))
    .catch((err) => console.log(err))
}
```

```
getPhotos()
console.log('Je serai exécuté en dernier, après le résultat de la fonction getPhotos')
```



Avec async/await

```
(async () => {
  console.log('Début du script')
  const getPhotos = async () => {
    await fetch('https://jsonplaceholder.typicode.com/photos')
      .then((res) => res.json())
      .then((photos) => console.log('photos', photos))
      .catch((err) => console.log(err))
  }
  await getPhotos()
  console.log('Je serai exécuté en dernier, après le résultat de la fonction getPhotos')
})();
```



Demo : Asynchrone

EXERCICES



Exercice 4



0-exercices/4-ex/README.md

Ce qu'il faut retenir

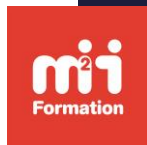


SAVE

- L'asynchrone permet d'effectuer des traitements lourds ou externes sans bloquer le cours d'exécution principal de notre script
- L'asynchrone permet de mettre à jour une partie ou l'intégralité du code HTML sans recharger la page.
- L'asynchrone en Vanilla JS (JavaScript natif) s'effectue à travers les objets Promise, les mots-clés `async/await`, la méthode `fetch` et les fonctions `callback` des événements du DOM.



- Pour les traitements asynchrones, utilisez de préférence *fetch* et les *Promises*.
- *Il existe d'autres méthodes comme l'objet XMLHttpRequest déprécié depuis ES6, mais il continue d'être supporté par tous les navigateurs.*
- Il y a également la librairie *axios* qui a des fonctionnalités supplémentaires comme l'interception des requêtes via des middleware qu'on peut mettre en place.



V. API HTML5



m2iformation.fr

Stockage



LocalStorage

- Solution de stockage offerte par les navigateurs. On parle plus communément de Web Storage.
- Le *localStorage* permet de stocker des informations sous-forme de clé-valeur avec une grande capacité de stockage allant jusqu'à **5 Mo – 10Mo** selon les navigateurs.
- La particularité du *localStorage* par rapport aux autres moyens de stockage du côté client (navigateur) est de pouvoir stocker les informations « sans limite dans le temps ».

```
// Enregistrer des données dans le localStorage
localStorage.setItem('firstname', 'glodie')
localStorage.setItem('age', 31)
localStorage.setItem('email', 'contact@tshimini.fr')

// Récupérer les données depuis le localStorage
localStorage.getItem('firstname') // glodie
localStorage.getItem('age') // 31

// Supprimer les données depuis le localStorage
localStorage.removeItem('age')

// Supprimer toutes les données du localStorage
localStorage.clear()
```


SessionStorage

- Similaire au *localStorage* avec une capacité de stockage moins importante.
- Les informations stockées ne sont disponibles que durant la session en cours.
- *Un onglet = une session.*
- Les informations ne sont pas partagées entre les différentes sessions de l'utilisateur.
- Avec le *localStorage*, les informations sont partagées avec les différents onglets appartenant au même domaine.

```
// Enregistrer des données dans le sessionStorage
sessionStorage.setItem('firstname', 'glodie')
sessionStorage.setItem('age', 31)
sessionStorage.setItem('email', 'contact@tshimini.fr')

// Récupérer les données depuis le sessionStorage
sessionStorage.getItem('firstname') // glodie
sessionStorage.getItem('age') // 31

// Supprimer les données depuis le sessionStorage
sessionStorage.removeItem('age')

// Supprimer toutes les données du sessionStorage
sessionStorage.clear() // ou fermez l'onglet du site
```

Sérialisation et désérialisation

- On parle de sérialisation lorsqu'il s'agit de transformer un objet en string (chaîne de caractères) pour le stocker, par exemple dans le *localStorage*.
La méthode ***JSON.stringify()*** transforme l'objet en chaîne de caractères sérialisés.
- A l'inverse, la désérialisation consiste à reconstruire l'objet d'origine à partir d'une chaîne de caractères sérialisés.
La méthode ***JSON.parse()*** reforme l'objet à partir de la chaîne de caractères sérialisés.

```
// Stockage d'un objet sérialisé

const person = {
  lastname: 'tshimini',
  firstname: 'glodie',
  age: 31,
  city: 'paris'
}
localStorage.setItem('glodie', JSON.stringify(person)) // transforme l'objet en JSON

/*
Récupérer un texte JSON depuis le localStorage
pour reconstruire l'objet d'origine (désérialisation)
*/
const glodie = JSON.parse(localStorage.getItem('glodie'))
```

Cookies

- Petits fichiers stockés sur le disque dur d'un visiteur d'une taille inférieure à 4 Ko.
- Des informations supplémentaires sont ajoutées pour permettre au navigateur de sécuriser le cookie.
- **Seul le nom (clé) et la valeur du cookie sont obligatoires.**
- Autres informations du cookie :
 - **Expires** : date d'expiration du cookie.
 - **Secure** : transmission avec le HTTPS
 - **SameSite** : prévient des attaques d'usurpation d'identité sur le web ([CRSF](#))

```
tz : "Europe%2FParis"  
Date de création : "Sun, 13 Nov 2022 13:39:01 GMT"  
Dernier accès : "Sun, 13 Nov 2022 21:16:16 GMT"  
Domain : ".github.com"  
Expiration / Durée maximum : "Session"  
HostOnly : false  
HttpOnly : false  
Path : "/"  
SameSite : "Lax"  
Secure : true  
Taille : 16
```

Exemple création, lecture et suppression d'un cookie

La valeur du cookie ne peut être modifiée en dehors du domaine où il a été créée.

```
// CRÉATION
document.cookie = `course=dom; expires=Thu, 01 DEC 2022 00:00:01 GMT; sameSite=Lax; secure;`

// LECTURE
const getCookie = (name) => {
  | return document.cookie.split('; ').find((row) => row.startsWith(`${name}=`)).split('=')[1];
}
console.log(getCookie('course')) // dom

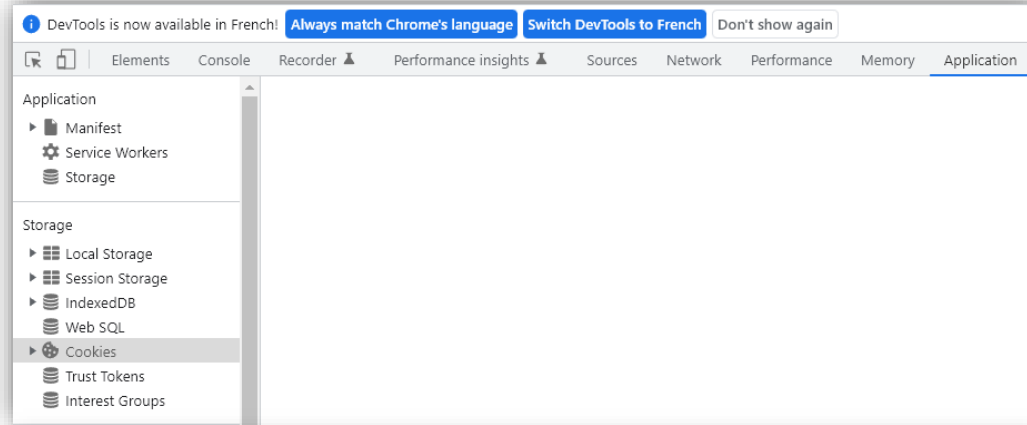
// MODIFICATION
document.cookie = 'course=storage; sameSite=Lax; secure;'
console.log(getCookie('course')) //storage

// SUPPRESSION
document.cookie = 'course=;' // or document.cookie('course=dom;expires=Thu, 07 09 2022 00:00:01 GMT;')
console.log(getCookie('course')) // undefined
```

DevTools et stockage local

Les données stockées en local sont accessibles depuis l'onglet **storage** ou **stockage** du navigateur.

Chrome



Firefox



EXERCICES



Exercice 5



0-exercices/5-ex/README.md



Workers

Web Workers

- Le cours d'exécution principal **délègue la réalisation** d'une ou plusieurs tâches aux *web workers*.
 - Un *web worker* s'exécute sur un **nouveau thread** pour effectuer les opérations qui lui ont été assignés en arrière plan.
 - A la fin des opérations, *le web worker* notifie le *thread* principal.
 - Le thread principal exécute la réponse renvoyée par le *web worker*.
- Gains :
 - ✓ Les traitements lourds n'impactent pas l'interactivité du site (interface utilisateur).
 - ✓ L'utilisateur peut effectuer d'autres actions en parallèle.
 - ✓ Interactions plus fluides.
 - Inconvénients
 - ✗ Pas accès aux objets du document (cookie, history, location, DOM, etc.), il faut notifier le fichier principal pour que ce dernier puisse effectuer les opérations précédentes

Demo Web Workers

EXERCICES



Exercice 6



0-exercices/6-ex/README.md

Composants Web



Composants Web : CustomElement

- Élément HTML personnalisé et réutilisable.
- *CustomElement* fonctionne comme les autres éléments natifs du langage HTML.
- Encapsule son propre code JavaScript pour définir ses propres comportements.
- Pour créer un *CustomElement*, il faut créer une classe qui étend la classe **HTMLElement** ou ses sous-classes *HTMLInputElement*, *HTMLDivElement* etc.
- Implémenter les méthodes
 - **.construct()** : appeler la méthode parent à l'aide de la méthode **super()**
 - **.connectedCallback()** : exécutée au moment où l'élément sera **ajouté** au **DOM**.
 - **.disconnectedCallback()** : exécutée au moment où l'élément sera **supprimé** du **DOM**.
- Enregistrer l'élément personnalisé sur la page :
 - Méthode statique *customElements.define("nom-avec-au-moins-un-tiret", MaClass)*

Implémentation CustomElement

```
class MyCustomElement extends HTMLElement {
  constructor() {
    super() // Création de l'élément
  }

  connectedCallback() {
    // Appelée au moment de l'ajout de l'élément dans le DOM
  }

  disconnectedCallback() {
    // Appelée au moment de la suppression de l'élément du DOM
  }

  /**
   * Il existe d'autres méthodes de la classe mère que vous pouvez implémenter
   * Vous pouvez également ajouter vos propres méthodes pour définir le comportement de l'élément
   */
}

customElements.define("my-tag", MyCustomElement);

/**
 * Implémentation dans le HTML
 * <body>
 * <my-tag></my-tag>
 * </body>
 */
```

Démo CustElement

EXERCICES



Exercice 7



0-exercices/7-ex/README.md

Ce qu'il faut retenir



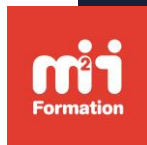
SAVE

- Contrairement aux **cookies** qui permettent de stocker une information sous la forme de clé-valeur avec une capacité très limitée de l'ordre de quelques Ko.
- Le **localStorage** permet de stocker la plus grande quantité d'informations sur le navigateur de l'utilisateur.
- Le **sessionStorage** est similaire. Cependant, la capacité de stockage est moins importante et les informations sont supprimées à la fin de la session (fermeture de l'onglet).
- Les webworkers sont utiles pour effectuer des traitements lourds sans bloquer le cours d'exécution principal.
- Utilisez les *CustomElement* dès que vous avez des éléments interactifs pouvant disparaître lors d'une modification du *DOM*.
Attention, ce n'est pas supporté par tous les navigateurs cf. [can i use](https://caniuse.com/custom-elements).

Conseils du formateur



- Choisissez la solution de stockage en fonction du volume d'informations à stocker
- Ne stockez jamais les informations sensibles (mot de passe, coordonnées bancaires etc.) sur les solutions de stockage du navigateur



VI. JQUERY



m2iinformation.fr

Généralités



- Développé en 2006 par *John Resig* (Mozilla).
- Bibliothèque JavaScript « **write less, do more** » :
 - Code **moins verbeux (moins de code)** pour effectuer des opérations sur le *DOM*.
 - **JQuery** règle le problème de compatibilité de JavaScript avec les navigateurs, ce qui justifie son succès et sa popularité auprès des développeurs JS.
- Très populaire durant la décennie précédente, aujourd'hui surpassé par les multiples frameworks ou librairies tels que *React, Vue JS, Angular, Astro, Anime, Three.js, Velocity* etc. et l'évolution de JavaScript lui-même.
- Dernière version **3.7.1** d'Aout 2023
- *Une version 4.0 en BETA*

Téléchargement

1. Via le site officiel <https://jquery.com/download/> en sauvegardant la librairie en local dans un fichier JavaScript.

```
<script src="jquery.js"></script>
```

2. Ou via les Content Delivery Network (CDN) de [Google](#) ou [CloudFare](#) ou [JQuery](#) lui même
 - Lien à ajouter dans le HTML soit dans le *head* ou avant la fermeture de la balise *body*

3. Ou via un gestionnaire de package comme *NPM* ou *YARN*
 - Avec *NPM* ou *YARN*, pour utiliser JQuery, importez le module depuis *node_modules/*

```
$ npm i jquery
```

```
$ yarn add jquery
```

```
import * as $ from 'jquery'
```


Fonctions



Fonction \$()

La variable globale JQuery et son alias \$ sont disponibles de manière globale ou via un import à partir du fichier source de la librairie.

```
JQuery(function(){  
    // script here !  
    // Déclenche le script avec JQuery une fois que le document HTML a été chargé.  
    // Évite les conflits dus à l'utilisation $ dans des nombreuses librairies JavaScript.  
})
```

Sélecteurs JQuery

```
$("#id") // par l'attribut ID de l'élément HTML  
  
$(".class") // par l'attribut CLASS  
  
$("p") // par le nom de la BALISE (TAG)  
  
$("h1, header, h2") // Combinaison de plusieurs éléments  
  
$("a[href=/contact.html]") // par l'attribut HREF
```

Tous les sélecteurs CSS peuvent être utilisés en argument de la fonction `$(/)` pour sélectionner des éléments du **DOM**.

La méthode css

- Affecte une valeur à la propriété CSS pour tous les éléments sélectionnés à l'aide de la fonction `$()`.

```
$("h1").css("color:red")
$("h2").css({"color:blue", "font-size": "20px"})
```

- Retourne la valeur de la propriété CSS du premier élément.
- [.hasClass\(name\)](#): retourne un booléen après avoir identifié au moins un élément de la liste des éléments possédant la classe *name*.
- [.addClass\(name\)](#): ajoute la classe *name* à tous les éléments.
- [.removeClass\(\)](#): supprime la classe *name* à tous les éléments.
- [.toggleClass\(\)](#): ajoute/supprime la classe *name* à tous les éléments.

```
$("h3").css("display")
```



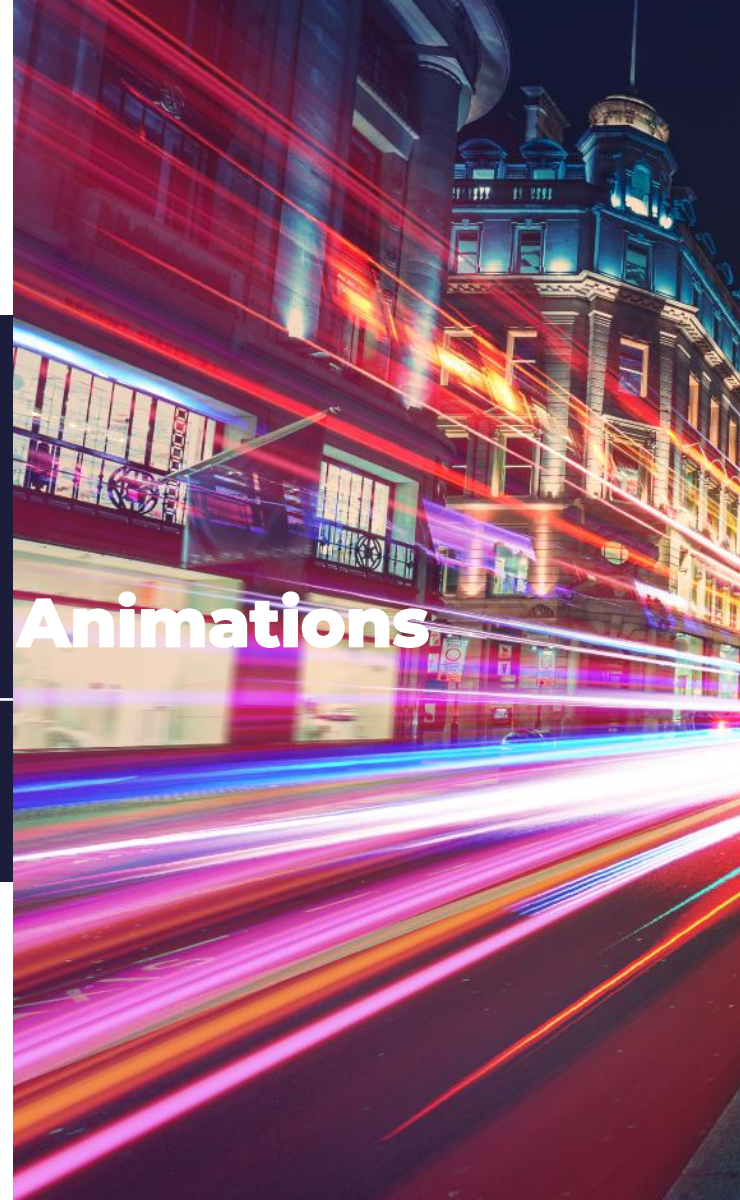
Le DOM

- [.html\(\)](#) : récupère le contenu HTML du premier élément de la liste.
- [.html\(code\)](#) : remplace le contenu des éléments de la liste par le *code* HTML.
- [.val\(\)](#) : récupère la valeur du premier élément.
- [.val\(value\)](#) : affecte la valeur *val* à tous les éléments.
- [.text\(\)](#) : récupère les nœuds textuels de tous les éléments descendants.
- [.text\(content\)](#) : affecte le contenu textuel *content* à tous les éléments.
- [.append\(code\)](#) : ajoute le *code* HTML à la fin de chaque élément de la liste.
- [.prepend\(code\)](#) : ajoute le *code* HTML au début de chaque élément de la liste.
- [.attr\(name, value\)](#) : affecte la valeur *value* à l'attribut HTML *name* de tous les éléments.

Les événements

- [.on\(event, callback\)](#) : identique à *document.addEventListener()*
- [.off\(\)](#) : supprime les gestionnaires d'événements de tous les éléments
- [.trigger\(event\)](#) : déclenche l'événement *event* sur tous les éléments

Animations



Les effets et animations

- [.show\(duration?\)](#) : apparition progressive de l'élément durant la durée duration
- [.hide\(duration?\)](#) : disparition progressive de l'élément durant la durée duration
- [.toggle\(duration?\)](#) : apparition/disparition alternée selon la valeur de la propriété CSS display
- [.fadeIn\(duration?\)](#) : similaire à show mais avec l'évolution de la propriété CSS opacity de 0 à 1. Effet de fondu
- [.fadeOut\(duration?\)](#) : opacité de 1 à 0
- [.fadeToggle\(duration?\)](#) : alterne selon la valeur de la propriété CSS opacity.

PS : *duration?* signifie que le paramètre est **optionnel**. Par défaut il vaut 0ms, ce qui provoque un effet instantané.

Les effets et animations

- [.animate\(cssNewProperties, options\)](#):
 - cssNewProperties un objet contenant les nouvelles propriétés CSS à remplacer progressivement par les anciennes valeurs en tenant compte des options.
 - options (objet) ayant les propriétés :
 - Durée
 - Elasticité de l'animation
 - etc.

```
$('#div').animate({height: '500px', opacity: '0.7'}, "slow")
```

Asynchrone



Voir la documentation globale sur la méthode ajax()

```
var menuId = $( "ul.nav" ).first().attr( "id" );
var request = $.ajax({
    url: "script.php",
    method: "POST",
    data: { id : menuId },
    dataType: "html"
});

request.done(function( msg ) {
    $( "#log" ).html( msg );
});

request.fail(function( jqXHR, textStatus ) {
    alert( "Request failed: " + textStatus );
});
```

- [\\$.ajax\(\)](#): globale pour toute requête asynchrone
- [\\$.get\(\)](#): spécifique pour une requête via la méthode HTTP GET
- [\\$.getJSON\(\)](#): Récupère du JSON depuis un serveur
- [\\$.post\(\)](#): spécifique pour une requête via HTTP POST
- Toutes les méthodes précédentes, retourne une promesse.
 - La méthode [.done\(callback\)](#) permet d'obtenir le résultat en cas de succès.
 - [.fail\(callback\)](#) permet de capturer une erreur survenue au cours du traitement de la requête.
 - [.always\(callback\)](#) s'exécute dans les cas

JQuery UI (User Interface)

- Extension de la librairie JQuery dédiée aux :
 - Interactions
 - Drag and drop
 - Redimensionnement des éléments de l'IU
 - Trie des listes
 - Etc.
 - Animations
- Widgets : éléments d'interface prêt à l'emploi
 - Accordeon
 - DatePicker
 - Tooltip
 - Etc.
 - Thèmes :
 - Plusieurs thèmes à disposition pour les éléments constituant cette bibliothèque
 - [Demos](#)
 - [Téléchargement de la librairie](#)

Démo JQuery

EXERCICES



Exercice 8



0-exercices/8-ex/README.md

Ce qu'il faut retenir



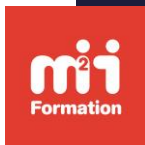
SAVE

- *JQuery* est une librairie de JavaScript compatible avec tous les navigateurs et très populaire lors de la décennie précédente, aujourd'hui on peut s'en passer.
- JQuery UI est une librairie dédiée à l'interface utilisateur proposant des animations, interactions, widgets, etc.

Conseils du formateur



- Utilisez *Vanilla JS* (JavaScript natif) à la place de *jQuery* pour les nouveaux projets.
- [Youmightnotneedjquery](#) est une bonne alternative pour transformer du code JQuery en Vanilla JS



VII. NODE JS



m2iinformation.fr

ÉCOSYSTÈME





- Node est un environnement (technologie) crée en **2009** par **Ryan Dahl**.
- Composé de :
 - Moteur **V8** (C++) de chrome qui **interprète** et **exécute** le code **JavaScript**
 - **NPM** (Node Package Module)
 - *OpenSSL*
 - *Zlib* (compression/décompression fichiers)
 - Modules permettant de gérer des protocoles *TCP*, *UDP*, **HTTP**, *QUIC*

[Liste exhaustive de toutes les dépendances](#)

Caractéristiques

- Programmation avec le langage JavaScript
- Multi-plateforme (Linux, Windows, Mac OS).
- Facile à prendre en main.
- Robuste : gestion d'un grand nombre de connexions.
- Rapide grâce à la gestion asynchrone des requêtes.
- Modulaire : développement des paquets par la communauté très actif.
- Faire du développement de plus bas niveau orienté serveur.
- Attention, certains modules côté front (navigateur) ne sont pas disponibles ou existent sous d'autres noms côté serveur.

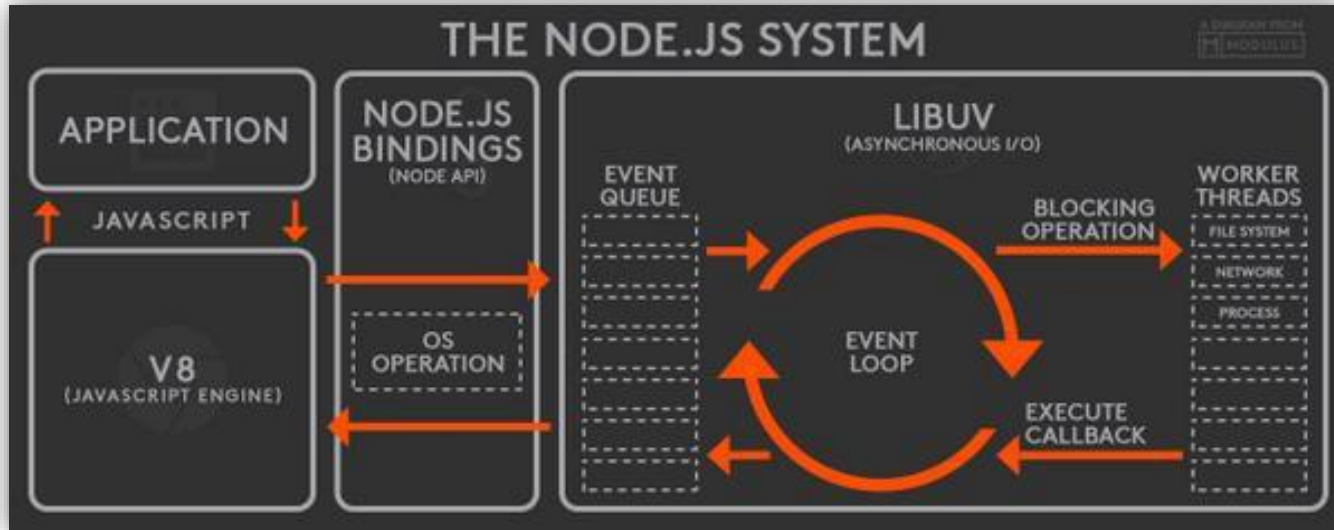
Principales utilisations

- Serveur Web
- API REST
- Streaming
- Chat
- SPA (Single Page Application)
- IoT (internet des objets)

- [Lien d'installation de Node.js selon son système \(OS\)](#)
- Version **LTS**: Long Time Support, version stable pour la production.
Actuellement 20.14.0 LTS.
- Version **Current**: version en cours de développement avec les dernières nouveautés.
Actuellement 22.2.0 Current.

Event Driven Model

- *NODE.JS BINDINGS*: librairies de communication entre les systèmes.
- *LIBUV*: gestion des entrées/sorties de manière asynchrone.
 - *Event Queue*: tâches à effectuer.
 - *Event Loop*: système de gestion des tâches.
 - *Worker Threads*: pile d'exécution des tâches (traitements).





Demo Event Loop

- [Simulateur loupe](#)

Application depuis le terminal



Notre premier script

1. Créez un fichier index.js
2. Ecrivez dans ce fichier
`console.log(process.versions)`
3. Exécutez ce fichier depuis un terminal avec la commande
`node index.js`

```
console.log(process.versions)
```

MENTS

▼ TERMINAL

```
Glodie@Glodie MINGW64 /e/formations
$ node index.js
{
  node: '19.0.1',
  v8: '10.7.193.13-node.16',
  uv: '1.43.0',
  zlib: '1.2.11',
  brotli: '1.0.9',
  ares: '1.18.1',
  modules: '111',
  nghttp2: '1.47.0',
  napi: '8',
  llhttp: '8.1.0',
  openssl: '3.0.7+quic',
  cldr: '41.0',
  icu: '71.1',
  tz: '2022b',
  unicode: '14.0',
  ngtcp2: '0.8.1',
  nghttp3: '0.7.0'
```

Objet process

- Gestion et contrôle du **processus** d'exécution du programme en cours.
- Objet *process* disponible de manière **globale** dans l'environnement Node.
- Gestion des **interactions** avec **3 propriétés** :
 - 1. *stdin***: entrées des données depuis le **flux standard des entrées** (input) ;
 - 2. *stdout***: sortie des données depuis le **flux standard des sorties** (output) ;
 - 3. *stderr***: sortie des erreurs depuis le **flux standard des erreurs**.

Objet process

- *process.env* : contient toutes les **variables d'environnement** de votre système où est installé Node.
- *process.argv*: arguments passés en ligne commande pour lancer le processus courant.

```
index.js demo\node\index.js
console.log(process.argv)

MENTS
▼ TERMINAL

Glodie@Glodie MINGW64 /e/formations/coderbase/orsys/js-perf-28112022/demo/node
$ node index.js argument1 argument2
[
  'C:\\Program Files\\nodejs\\node.exe',
  'E:\\formations\\coderbase\\orsys\\js-perf-28112022\\demo\\node\\index.js',
  'argument1',
  'argument2'
]
```

Interaction depuis le terminal : package readline

- *Objet Readline* : interaction avec un utilisateur à partir du terminal.
 - `.createInterface()` crée une interface en ligne de commande en prenant en paramètre les propriétés *stdin* et *stdout* de l'objet *process*.
 - `.close()` met fin au dialogue.
 - `.question()` permet à l'utilisateur de saisir des données depuis le flux standard d'entrée et qui peuvent être récupérées dans une variable.

```
import { createInterface } from "node:readline/promises"
const app = createInterface({input: process.stdin,output: process.stdout})
app.question('Quel est votre langage de programmation préféré ? ')
.then(res => {
  console.log(`langage préféré : ${res}`)
  app.close()
})
```

MINAL PROBLEMS OUTPUT COMMENTS

TERMINAL

```
Glodie@Glodie MINGW64 /e/formations/coderbase/orsys/js-perf-28112022 (feature/back/typesc
$ node demo/node/index.mjs
Quel est votre langage de programmation préféré ? JavaScript
langage préféré : JavaScript
```

EXERCICES



Exercice 9



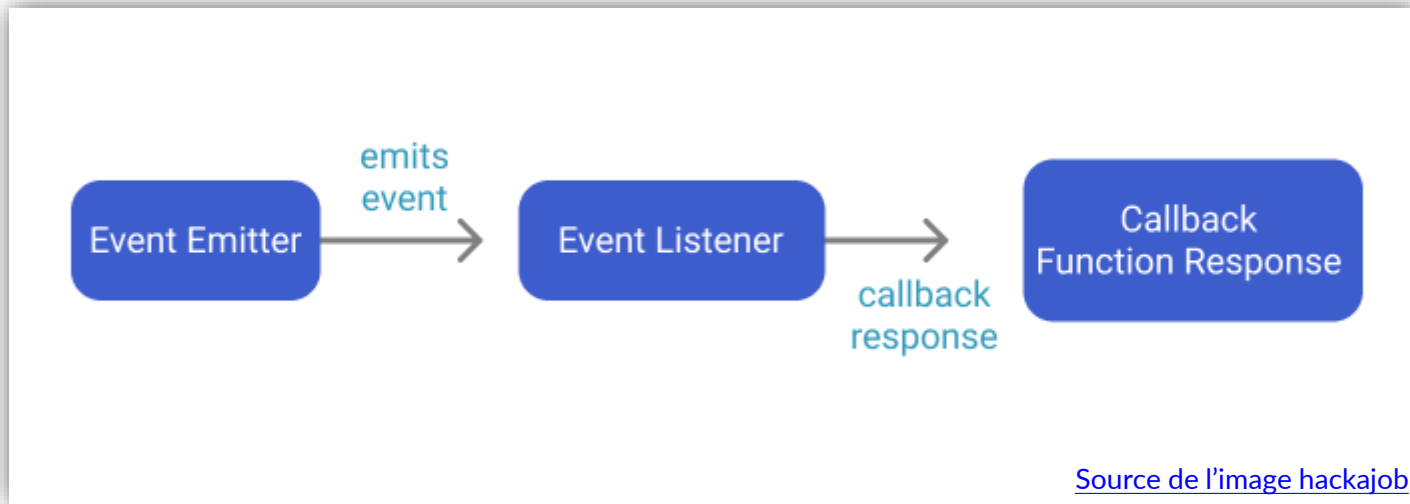
0-exercices/9-ex/README.md

Gestion des événements



Classe Events : package events

- Responsable de la gestion des événements de tout type, entrée, sortie, serveurs, données, fin de processus, etc.
- Un bon nombre des classes de Node dérive (hérite) de cette classe.
- Lorsqu'un événement a lieu, un écouteur d'événement exécute du code défini depuis sa fonction de callback (retour asynchrone).



Méthodes

- *obj.addListener(eventName, listenerCallback)*: ajout d'un gestionnaire d'événement à l'objet *obj*.
 - Paramètres
 - *eventName*: nom de l'événement
 - *listenerCallback*: fonction de rappel exécuté lorsque l'événement se produit.
- *obj.on(eventName, listenerCallback)*: alias de la fonction *.addListener()*.
- *obj.emit(eventName)*: émission de l'événement *eventName* sur l'objet *obj*.

```

index.mjs demo\node\index.mjs app

import { EventEmitter } from 'events'
const app = new EventEmitter()
app.on('app_start', () => {console.log('app started on http://localhost:3000')})
// Something else here
app.emit('app_start')
  
```

COMMENTS

▼ TERMINAL

```

Glodie@Glodie MINGW64 /e/formations/coderbase/orsys/js-perf-28112022/demo/node (feature/ba
$ node index.mjs
app started on http://localhost:3000
  
```

EXERCICES



Exercice 10



0-exercices/10-ex/README.md

Serveur Web



Serveur Web : package http

- Gestion des connexions via le protocole HTTP
- Méthodes
 - *createServer([requestListener]):*
crée un serveur et retourne un objet *http.server* qui hérite de la classe *events.EventEmitter*
 - *.listen(port, [callback]):*
définit le **port d'écoute** sur lequel un client doit se connecter pour communiquer avec le serveur.
La callback est facultative.

```
index.mjs demo\node\index.mjs\webServer\createServer() callba
import { createServer } from 'http'
console.info('App on http://localhost:8080')
const webServer = createServer(() => {
  console.log('Welcome!')
  process.exit(1)
})
webServer.listen(8080)
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-2
App on http://localhost:8080
Welcome!

[Done] exited with code=1 in 3.217 seconds
```


Objet Request

- Objet de la classe *http.incomingMessage* qui dérive de la classe *stream.readable*.
- Contient les informations transmises par un client *HTTP* au serveur *HTTP*, on peut y trouver les propriétés suivantes :
 - *request.url* : URL de la requête sans le nom de domaine et le port ;
 - *request.method* : méthode HTTP associée à la requête ;
 - *request.headers* : en-têtes de la requête.

Objet Response

- Objet de la classe *http.ServerResponse* qui dérive de la classe *stream.writable*.
- Possède les méthodes suivantes :
 - *response.write(chunk, [encoding], [callback])* : écrit la donnée *chunk* sur le flux d'écriture du corps du message ;
 - *response.end([chunk], [encoding], [callback])* : ferme le flux d'écriture ;
 - *response.writeHead(statusCode, [headers])* : indique le statut HTTP de retour et ajoute les informations d'en-têtes.

Implémentation d'un serveur Web

```
import { createServer } from 'http'
console.info('App on http://localhost:3000')
createServer((req, res) => {
  const headers = { 'Content-Type': 'application/html; charset=utf-8' }
  if (/\/$/.test(req.url)) {
    res.writeHead(200, headers)
    res.write('<h1>Welcome</h1>')
  } else {
    res.writeHead(404, headers)
    res.write('not found')
  }
  res.end()
}).listen(3000)
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\demo\node\index.mjs"
App on http://localhost:3000

GET http://localhost:3000

http://localhost:3000

GET http://localhost:3000

Params Authorization Headers (1)

Query Params

KEY
x
Key

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize

Welcome

EXERCICES



Exercice 11



0-exercices/11-ex/README.md

Gestion des fichiers



Gestion des fichiers : package fs

- Lire/écrire :
 - *`fs.readFile('file.md', {encoding: 'utf-8'}, (err, data) => ({...});`*
 - *`fs.writeFile('file.md', 'my content', (err) => ({...}).`*
- Supprimer fichiers et dossiers :
 - *`fs.unlink(path, callback)`, `fs.mkdir(path, callback)` et `fs.rmdir(path, callback)`*
- Informations sur les fichiers et dossiers, ouvrir/fermer, lister, etc.
- La gestion des fichiers peut se faire de manière :
 - Asynchrone ;
 - Synchrone : les méthodes ont le suffixe Sync ex: *`readFileSync()`*.
 - Avec des promesses depuis les packages avec un suffixe *`/promises`*
Par exemple ***`import { readFile } from 'node:readFile/promises'`***

Implémentation écriture et lecture d'un fichier

```
37 import {readFile, writeFile} from 'node:fs'
38
39 writeFile('hello.md', 'hello world', (err) => {
40   if(!err) {
41     readFile('hello.md', {encoding: 'utf-8'}, (er, data) => {
42       if(!er) console.log('content : ', data)
43     })
44   }
45 })
46
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\demo\node.js" -- --
content : hello world

[Done] exited with code=0 in 0.203 seconds

Les streams : package fs

- Utile pour la gestion des fichiers très volumineux.
- L'écriture ou la lecture s'effectue par flux de données (bout d'information découpé de plus petite taille).
- La méthode *createReadStream()* permet de gérer un flux de lecture.
- La méthode *createWriteStream()* permet de gérer un flux d'écriture.
- La méthode *pipe()* appliqué à un *stream* permet de synchroniser 2 flux.

Par exemple un flux de lecture avec un flux d'écriture pour copier un fichier d'un *disque A SSD* vers un *disque B SATA* (ils n'ont pas la même vitesse de transfert des données).

Implémentation streams lecture et écriture

```
import { createWriteStream } from 'node:fs'
import { request } from 'node:https'
const hostname = 'raw.githubusercontent.com'
const path = '/dr5hn/countries-states-cities-database/master/countries%2Bstates%2Bcities.json'
const countries = createWriteStream('country-states.json')
request({hostname, path, method: 'GET'}, (res) => {
  res.pipe(countries)
  res.on('end', () => { console.log('fin de lecture de la page web')})
  countries.on('close', (err) => {
    if(err) console.log('erreur', err)
    console.log('fin de la copie !')
  })
}).end()
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\demo\node\index.mjs"
fin de lecture de la page web
fin de la copie !

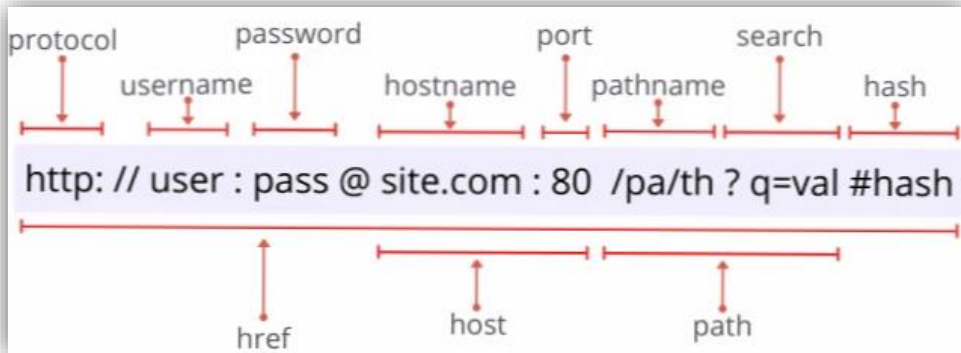
[Done] exited with code=0 in 34.148 seconds
```

Package url

- Gestion des différents composants d'un URL.

- Propriétés

- *href*: URL
- *Protocol*
- *Hostname*
- *hash*: l'ancre (après #)
- *pathname*: partie après le port



[Source de l'image usbforwindows](#)

- Méthodes

- *resolve(from, to)*: construit un URL relative ou absolue en combinant les informations du *from* (URL de base) et *to* (URL relatif à résoudre par rapport à la l'URL de base)

EXERCICES



Exercice 12



0-exercices/12-ex/README.md

Base de données SQLite 3



Installation et création d'une base de données

- Installation `npm i sqlite3`
- Mode de lecture
 - *OPEN_READONLY*
 - *OPEN_READWRITE*
 - *OPEN_CREATE*
- Création de la base de données sur :
 - Mémoire (*:memory:*);
 - Ou disque en indiquant le chemin vers le fichier de la base de données cf. image ci-dessous

```
import sqlite3 from 'sqlite3'
const db = new sqlite3.Database('./database.sqlite', sqlite3.OPEN_READWRITE, (err) => {
  if (err) {
    console.error(err.message)
  }
  console.log('Connected to the database !')
})
```

TERMINAL PROBLEMS 3 OUTPUT COMMENTS

[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\3-exercices\1-back\demo.mjs"

SQLITE_CANTOPEN: unable to open database file

Connected to the database !

[Done] exited with code=0 in 0.264 seconds

Les Méthodes pour effectuer les requêtes SQL

- *Méthodes*
 - *.run(sql, [params], [callback(err, [row])])*: pour les requêtes de création, insertion, suppression et modification.
 - *.get(sql, [params], callback(err, row))*: pour les requêtes de sélection avec la récupération du premier résultat.
 - *.all(sql, [params], callback(err, row))*: pour les requêtes de sélection d'une table entière.
 - *.close([callback(err)])*: ferme la connexion à la base de données
- *Propriétés*
 - *lastID*: ID du dernier élément inséré dans la table lors d'une requête d'insertion
 - *changes*: informations sur les lignes affectés par une requête.

Mode d'exécution

- *.serialize()*
 - Exécution des requêtes de manière séquentielle, une requête après l'autre selon l'ordre d'écriture dans le code.
- *.parallelize()*
 - Exécution des requêtes en parallèle.

Implémentation base de données SQLite3

Attention avec `.parallelize()`, tout s'exécute en même temps donc l'ordre n'est pas garanti

```
import sqlite3 from 'sqlite3'
const db = new sqlite3.Database(':memory:', sqlite3.OPEN_READWRITE, (err) => {
  if (err) {console.error(err.message)}
  console.log('Connected to the database !')
})
db.serialize(() => {
  db
    .run('CREATE TABLE product(id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(50))', (err) => {
      if(!err) console.log('Table created !')
    })
    .run('INSERT INTO product(name) VALUES (?,?), (??), (??)', ['tomato', 'apple', 'orange'], function (err) {
      if(!err) console.log(`${this.changes} affected and lastID ${this.lastID}`)
    })
  })
db.parallelize(() => {
  db
    .all('SELECT * FROM product', function (err, rows) {
      if(!err) console.log('results before update', rows)
    })
    .run('UPDATE product SET name=? WHERE id=?', ['salad', 3], function (err){
      if(!err) console.log(`updated, ${this.changes} row affected`)
    })
    .all('SELECT * FROM product', function(err, rows) {
      if(!err) console.log('results after update', rows)
    })
  })
})
db.close(() => {console.info('database closed !')})
```

```
[Running] node "e:\formations"
Connected to the database !
Table created !
3 affected and lastID 3
results before update [
  { id: 1, name: 'tomato' },
  { id: 2, name: 'apple' },
  { id: 3, name: 'orange' }
]
updated, 1 row affected
results after update [
  { id: 1, name: 'tomato' },
  { id: 2, name: 'apple' },
  { id: 3, name: 'salad' }
]
database closed !
```

EXERCICES



Exercice 13



0-exercices/13-ex/README.md

BCRYPT

Bcrypt : module bcrypt

- Installation : `npm install bcrypt`
- [Documentation](#)
- Cryptage irréversible : impossible de revenir vers la donnée initiale
- Cas d'utilisation des mots de passe avec les méthodes suivantes :
 - `.genSalt([nb, version], callback)` génère un sel, par défaut nb=10 et version=b
 - `.hash(plain, salt, callback)` génère un mot de passe haché à partir du mot de passe brut et du sel
 - `.compare(plain, hashDb, callback)` compare les 2 hash (hash crypté depuis la saisie brut *plain* de l'utilisateur et *hashDb* en base de données.
- Les méthodes *hash*, *compare*, *genSalt* peuvent s'utiliser de manière **asynchrone**, avec des **promesses** ou **synchrone** avec le suffixe *Sync*, exemple `.hashSync()`

Implémentation hachage

```
db.serialize(() => {
  db.run('CREATE TABLE user(id INTEGER PRIMARY KEY AUTOINCREMENT, password VARCHAR(255))', (err) => {
    if(!err) console.log('Table user created !')
  })
  const salt = 10
  const PlainPassword = '>j5LqNgHw*%74Y7'
  hash(PlainPassword, salt, (err1, hashPassword) => {
    if (!err1) {
      db
        .run('INSERT INTO user(password) VALUES (?)', hashPassword, function (err2) {
          if(!err2) console.log(`${this.changes} affected and lastID ${this.lastID}`)
        })
        .all('SELECT id, password FROM user', function (err3, row) {
          if(!err3) console.log('user in database ', row)
        }).close(() => {console.info('database closed !')})
    }
  })
})
```

TERMINAL PROBLEMS OUTPUT COMMENTS

```
[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\3-exercices\1-
Connected to the database !
Table user created !
1 affected and lastID 1
user in database [
  {
    id: 1,
    password: '$2b$10$XGzI3iRaUQA2rAxbVKHUUuTg9xgV7.aT4Rxvrik8tWMP8PjYF4M.K'
  }
]
database closed !
```

Implémentation comparaison

```
const plain = '>j5LqNgHw*%74Y7'  
const fromDb = '$2b$10$Xp2H0N7h.Qe3fBOJJc9h0ef9sm2MknOW2x.kt.6wOGpwrPofltFWC'  
compare(plain, fromDb)  
  .then((ok) => {  
    if(ok) console.log('same password')  
    else throw new Error('not same password')  
  }).catch(ko => console.log('error : ', ko))
```

TERMINAL PROBLEMS OUTPUT COMMENTS

[Running] node "e:\formations\coderbase\orsys\js-perf-28112022\3-exercices\1-back\demo.mjs"
same password

[Done] exited with code=0 in 0.33 seconds

EXERCICES



Exercice 14



0-exercices/14-ex/README.md

Ce qu'il faut retenir



SAVE

- Node est un ensemble de technologie écrit en C, C++ et JavaScript permettant de réaliser des applications Web côté Serveur, SPA, IoT, etc.
- La gestion des fichiers et des *streams* s'effectue à l'aide du paquet *fs*.
- Le paquet *http* permet de réaliser un serveur Web.
- Sécurisez les mots de passe à l'aide du paquet *bcrypt*.

Conseils du formateur



- *Bcrypt* est indispensable pour crypter les mots de passe utilisateur.
- Ne jamais stocker des informations sensibles en utilisant JWT.
- Vous pouvez aller plus loin en utilisant [Express](#), *framework* de *NODE* facile à prendre en main.
- Vous pouvez aller plus loin avec les bases de données en explorant les packages [MySQL2](#) ou [MongoDB](#) pour des échanges avec des bases données distantes.

Évaluations formateur

N'oubliez pas les évaluations formateur avant de partir.



FIN
Merci d'avoir suivi cette formation

