

JavaScript – Fondamentaux

24/01/2024 – 26/01/2024

Glodie Tshimini



Glodie Tshimini

Formateur et développeur depuis 2017

Certifié Codebase IT Expert Trainer

Certifié Codebase IT Expert Trainer At POE

Certifié Professional SCRUM Developer (PSD)

Email : contact@tshimini.fr

HORAIRES ET PAUSES

Mercredi

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

Jeudi

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

Vendredi

- 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.

Déroulement de la formation

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

GitHub de la formation



<https://github.com/glo10/jvs-in-24012024.git>

Un formateur à votre écoute



Google Forms

<https://docs.google.com/forms/d/e/1FAIpQLSfe2ajgGol2orlEtJ4VTX5pZDUKWI2Qf0LQxdw5AD81r4L4bg/viewform>

OBJECTIFS PÉDAGOGIQUES

- Mémoriser les bases de JavaScript et de son utilisation pour le DOM
- Gérer les **événements** et les manipulations dynamiques
- Identifier les règles d'or de la programmation avec JavaScript
- Réaliser des appels synchrones et Asynchrones
- Exploiter les media queries en JavaScript pour une interface responsive.

NIVEAU REQUIS ET PUBLIC CONCERNÉ

Niveau requis

- Avoir connaissance de HTML5 et CSS3.
- Il est également nécessaire de connaître la programmation structurée.

Public concerné

- Développeurs
- Architectes
- Chefs de projets techniques.



Présentations

Merci de bien vouloir vous présenter individuellement.

Merci de bien vouloir m'indiquer quelles sont vos attentes suite à cette formation.

PLAN DU COURS



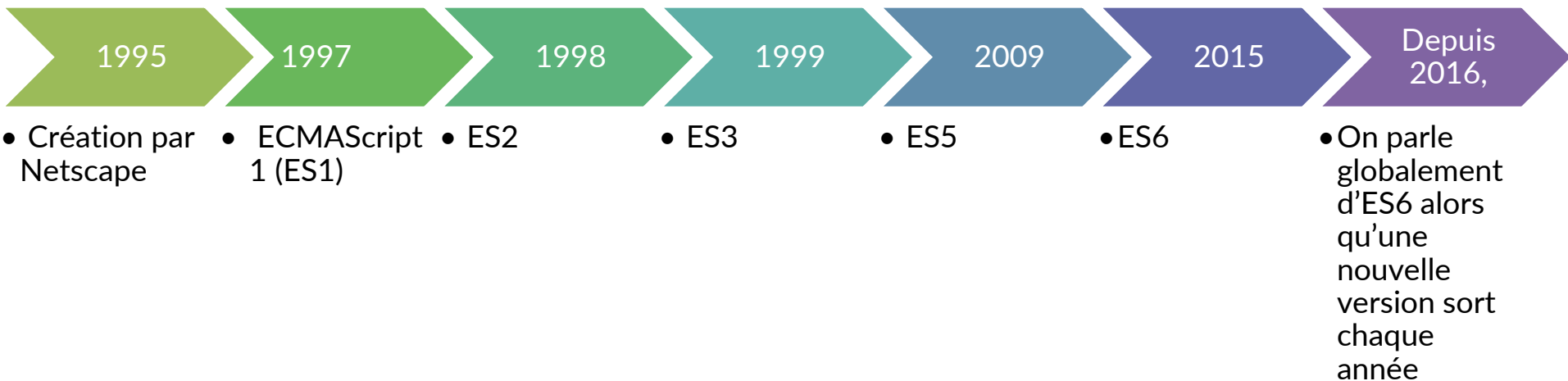
Plan

- I. Histoire de JavaScript
- II. Bases du langage
- III. DOM
- IV. Manipuler le CSS avec JavaScript
- V. Asynchronie
- VI. Nouveautés depuis ES6

I. HISTOIRE DE JAVASCRIPT



HISTORIQUE (documentation versions)

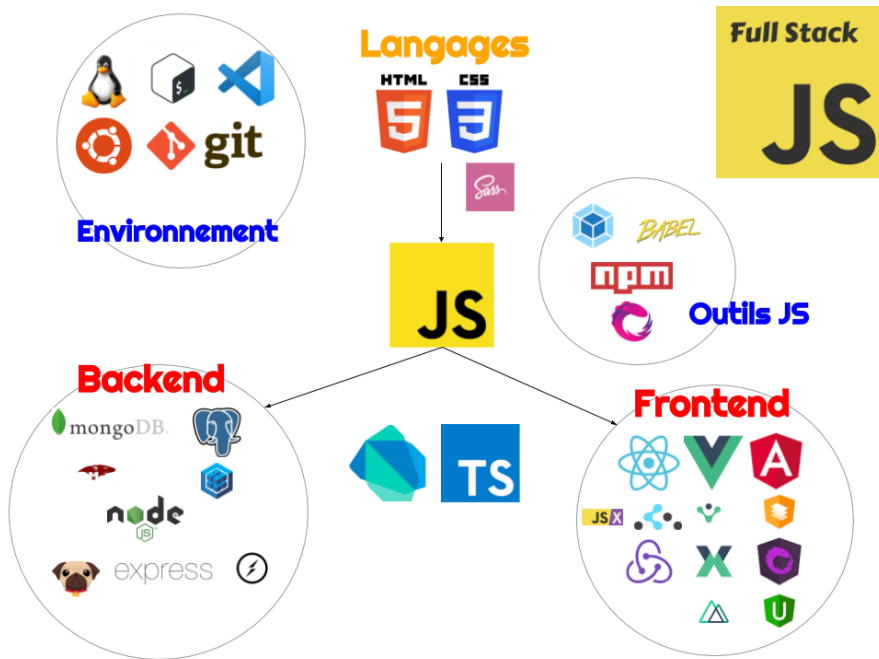


Caractéristiques

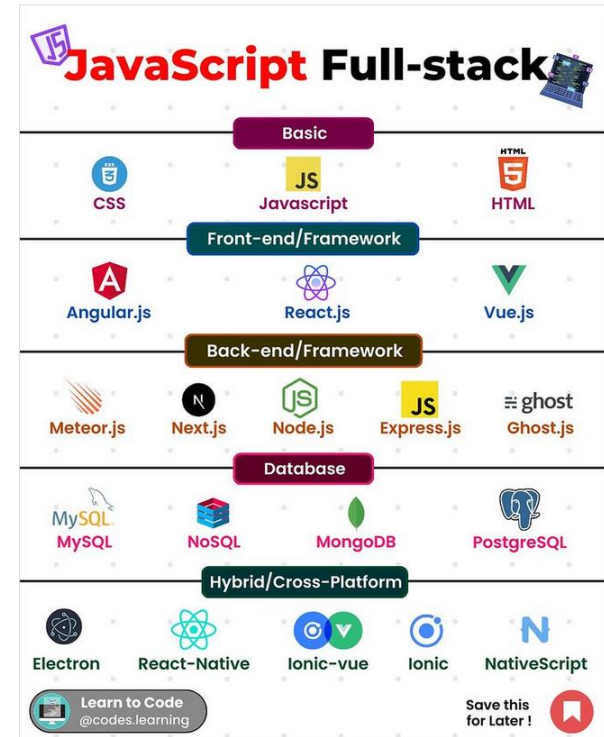
- Langage de programmation et de script
- Standardisé par **ECMAScript** (European Computer Manufacturers Association)
- Faiblement typé
- Case insensitive (sensible aux majuscules et miniscules)
- A ne pas confondre avec Java qui est un autre langage de programmation complètement différent
- Langage à la fois
 - Pour un usage côté client (Navigateur)
 - Et serveur avec l'environnement de développement NodeJS ou encore Deno et Bun

Tout l'écosystème, librairies, Framework, etc.

[Source image blog.dyma](#)



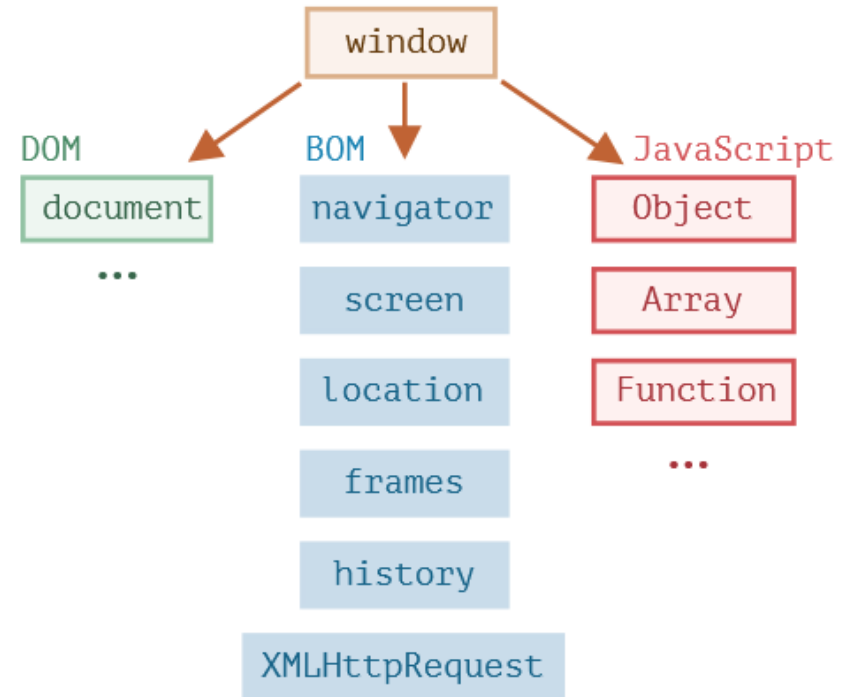
[Source image codes.learning](#)



JavaScript côté client

- Exécuté et interprété par le navigateur
- Permet de rendre les pages Web (HTML et CSS) dynamiques
 - Manipuler les éléments du document HTML (balises et propriétés CSS)
 - Gérer les formulaires de contact (validation)
 - Manipuler la fenêtre (window)
 - Géolocalisation
 - Gestion de l'onglet (dimension, ouverture, fermeture, etc.)
 - API HTML5
 - Storage (stockage côté navigateur)
 - Canvas (dessin)
 - Drag & Drop
 - Offline
 - Web Workers
 - Etc

[Source image fr.javascript.info](http://fr.javascript.info)





- 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*

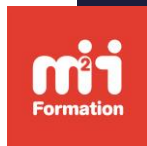
[Liste exhaustive de toutes les dépendances](#)

INSTALLATIONS





- `install.md`



II. BASES DU LANGAGE



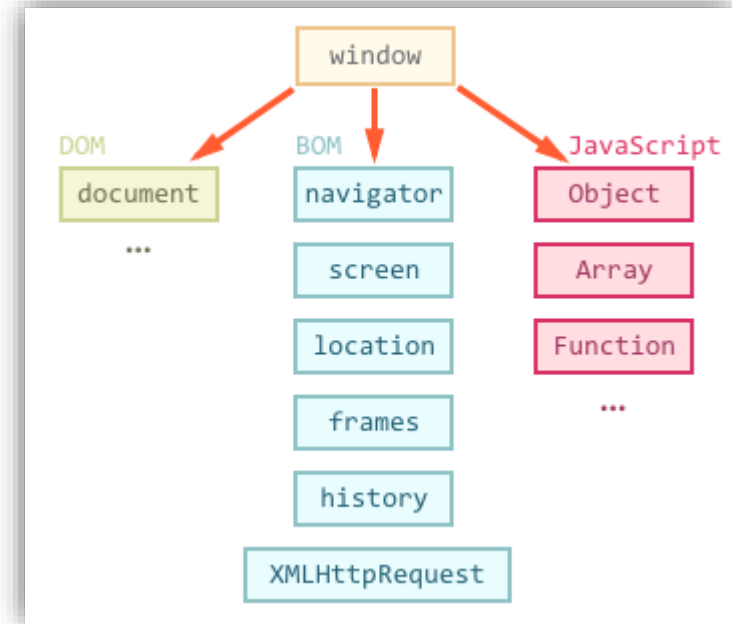
m2information.fr

RAPPELS ET GÉNÉRALITÉS



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)

Rappels programmation orientée objet

- La POO est une méthode de programmation informatique née dans les années 1960.
- Le programme est une collaboration entre des objets.
- Un objet est une entité qui possède des caractéristiques et des comportements.
- Par exemple, une voiture est un objet qui a :
 - Caractéristiques :
 - Nombre de porte
 - Couleur
 - Comportements
 - Accélérer
 - Freiner
- Les caractéristiques ou propriétés constituent un ensemble d'attributs définissant l'apparence de l'objet.
- Les comportements ou méthodes constituent un ensemble d'actions réalisables par l'objet.

Généralités syntaxe

- Une instruction JS ne se termine pas obligatoirement par un point virgule comme dans la plupart des langages de programmation. Autrement dit finir l'instruction par un ; est optionnelle.
- Les commentaires s'écrivent de deux façons :
 - // commentaire sur une ligne
 - /* commentaires sur plusieurs lignes */

À l'intérieur de la balise HEAD

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Document</title>
7    <!-- (1) Entre les balises head, le fichier JS sera chargé avant le body -->
8    <script src="beforeBody.js"></script>
9    <!-- (2) L'attribut defer de la balise script permet de
10     différer le chargement du fichier à la fin du document
11     -->
12    <script src="betweenHeadButLoadAfterBody.js" defer></script>
13    <script>
14      <!-- (3) directement dans le document HTML -->
15      let firstName = 'Glodie';
16      let age = 32;
17    </script>
18  </head>
19  <body>
20
21  </body>
22  </html>

```

Ou avant la fermeture de la balise body

```

19  <body>
20    <script src="index.js"></script>
21    <script>
22      let lastName = 'Glodie';
23      let city = 'Paris';
24    </script>
25  </body>
26  </html>

```

LES DONNÉES



Variables et constantes

Variable

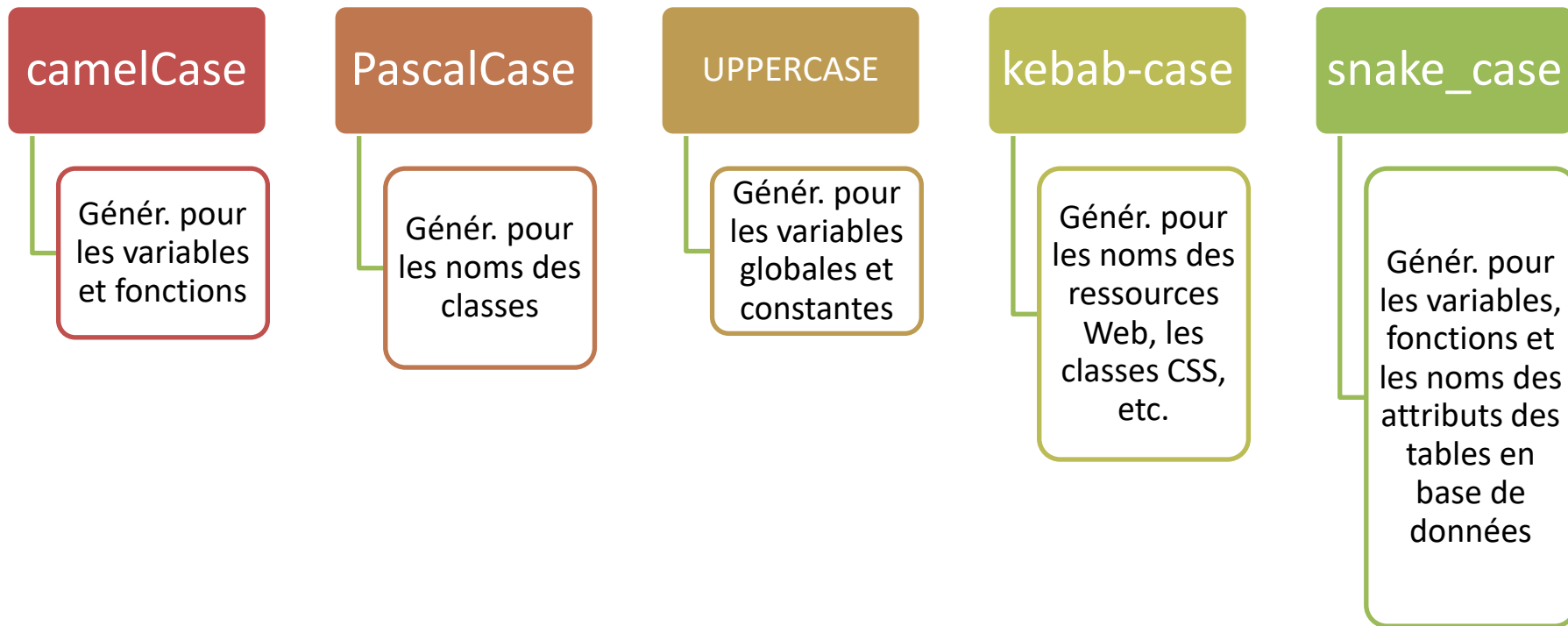
- Une information (donnée) qui est stockée dans la RAM de la machine qui va exécuter le programme.
- Manipuler les variables
 - Mot-clé **var** / **let** suivi du nom de la **variable**
 - Initialiser ou déclarer une variable :
let firstName;
 - Affecter ou assigner une valeur : let
lastName = 'Tshimini'
 - Modifier la valeur : lastName = 'Doe'

Constante

- Variable non-modifiable
- Mot-clé **const**
- Déclarer une constante: const varName
= 0;
- Les déclarations de variables ou des constantes doivent être précises - Attention à la casse !
 - myVar et MyVar sont 2 variables différentes.

Convention de nommage

Les conventions ci-dessous varient en fonction des langages de programmation et des entreprises



- Faiblement typé
- Le typage est dynamique
- Pour avoir un typage fort, on peut utiliser le mode strict ou avoir recours à [TypeScript](#) (*Superset* de JavaScript, une surcouche des fonctionnalités développée par *Microsoft* pour ajouter en outre un typage fort)

7 types primitifs

String	<code>let ville = "Toulouse";</code>
Number	<code>let prix = 55;</code>
Boolean	<code>let existe = true; / let existe = false;</code>
Null	<code>let varNull = null;</code>
Undefined	<code>let ville;</code>
Object	<code>let ville = { nom: "Toulouse", cp:"31100" }</code>

Concaténation, Transtypage

concaténation

- Mettre bout à bout des chaînes de caractère

```
const firstName = 'Glodie'
const lastName = 'Tshimini'
// concaténation avec l'opérateur de concaténation +
console.info(firstName + ' ' + lastName)
// Concaténation avec ${} et ``
console.info(`${firstName} ${lastName}`)
// Glodie Tshimini
```

Transtypage

Changer le type d'une information par un autre type

```
let num = '3' // string
num = parseInt(num) // number
num = parseFloat(num) // number
num = num.toString() // string
```

```
// mot clé typeof sur une variable ou constante retourne son type
console.log(typeof num)
```

Liste indexée (ordonnée) d'éléments normalement appartenant au même domaine de définition (type), en JavaScript un tableau peut contenir en même temps un *string*, un *number*, un *boolean*, etc.

Déclaration d'un tableau :

- `const monTableau = [];`
- `const monAutreTab = new Array();`

Remplissage d'un tableau :

- `monTableau[0] = 4;` : Mettre 4 à la première position 0
- `console.log(monTableau[1])` : Accès à l'élément situé à l'indice 2 du tableau ici *undefined*

Exemples

```
170  const numbers = [1, 2, 3, 4, 5]
171  const names = ['Fatou', 'Maria', 'Solange', 'Sarah']
172  // Tableau à 2 dimensions
173  const persons = [
174      ['Fatou', 'Paris', 30, true],
175      ['Eric', 'Nancy', 56]
176  ]
177  console.log(numbers[0], numbers[5]) // 1 et undefined
178  console.log(names[1]) // Maria
179  console.log(persons[1]) // ['Eric', 'Nancy', 56]
```

Quelques méthodes associées aux tableaux

Source Igor Gonchar

Creates new Array:

`[▲●■○].map(☐→■) → [■■ ■■]`

`[▲●■○].filter(☐===●) → [●●]`

`[▲●■○].join("-") → "▲-●-■-○"`

`[▲●].concat([■○]) → [▲●■○]`

`[▲●[■○]].flat() → [▲●■○]`

`[▲●■○●○].slice(2, 4) → [■○]`

Edits current Array:

`[▲●■].forEach(☐→■) → [■ ■ ■]`

`[▲●■○].push(●) → [▲●■○●]`

`[▲●■○●].pop() → [▲●■○]`

`[●○■○●].shift() → [○■○●]`

`[▲●■○].sort() → [▲●○■]`

`[▲●■○].fill(■, 1) → [▲■■■]`

Searches in current Array:

`[▲●■○].find(☐===●) → ●`

`[▲●■○].findIndex(☐===■) → 2`

`[▲●■○].indexOf(●) → 1`

`[▲●■○].some(☐===■) → true`

`[▲●■○].every(☐===●) → false`

`[▲●■○].includes(■) → true`

-
- ☐ - each item in array, one by one
 - (☐===●) - accepts callback function
 - (●) - accepts value
 - [...] [...] - different arrays
 - "..." - string

<https://igorgo.nl>

STRUCTURES CONDITIONNELLES



IF-ELSEIF-ELSE

- Exécution d'une seule instruction de la structure parmi les différentes « branches » possibles
- L'ordre des instructions à une importance
- La première instruction qui vérifie la condition sera exécutée (pas les autres)

```
69  const age = 100
70  if(age < 18) {
71      console.log('Mineur')
72  } else if(age >= 18 && age < 55) {
73      console.log('Majeur')
74  } else if(age >= 55 && age < 100) {
75      console.log('Senior')
76  } else {
77      console.log('Centenaire')
78  }
```

SWITCH

- Identique à *if*, plus lisible dans les cas où il y a une vérification sur le contenu d'une chaîne de caractère
- Plus performant au niveau de l'exécution

```
80  const season = 'winter'
81  switch(season) {
82      case 'winter':
83          console.log('Manteau')
84          break
85      case 'summer':
86          console.log('Tee-shirt')
87          break
88      case 'autumn':
89          console.log('Parapluie')
90          break
91      case 'spring':
92          console.log('Trench')
93          break
94      default:
95          console.log('Autre')
96          break
97  }
```

STRUCTURES ITÉRATIVES



Structures itératives (boucles)

- Parcourir des objets en exécutant un bloc de code tant qu'une certaine condition est vérifiée
- [Documentation de toutes les boucles itératives avec JavaScript](#)
- FOR : `for (let i=0; i<10; i++) { ... };`
 - Boucle adaptée lorsqu'on connaît le nombre d'itération à effectuer
- FOR ... IN : `for (index in tab) { tab[index] };`
 - Dérivée de la boucle For et adaptée pour parcourir des objets
- FOR ... OF : `for (index of tab) { index };`
 - Dérivée de la boucle for et adaptée pour parcourir des objets
- WHILE : `while (condition) { ... };`
 - Boucle adaptée pour parcourir des éléments lorsqu'on ignore le nombre d'itération à effectuer
- DO ... WHILE : `do { ... } while {condition};`
 - Similaire à while sauf qu'il s'exécute au moins une fois
- Le mot-clé **break** permet de sortir d'une boucle prématurément.

```

100 const cities = ['Paris', 'Marseille', 'Lille', 'Lyon', 'Nantes']
101
102 for(let i = 0; i < cities.length; i++) {
103   console.log('Ville avec la boucle for : ', cities[i])
104 }
105
106 let i = 0
107 while(cities.length > i) {
108   console.log('Ville avec la boucle while : ', cities[i])
109   i++// attention en cas d'oubli de l'incrémentation => boucle infini
110 }
111
112 do {
113   console
114   .log('Je m'exécute au moins une fois même si la condit. est fausse')
115 } while(false)

```

```

114 const cities = ['Paris', 'Marseille', 'Lille', 'Lyon', 'Nantes']
115 const france = {
116   capital: 'Paris',
117   continent: 'Europe',
118   cities
119 }
120
121 for(let i in cities) {
122   console.log('Ville avec for in : ', i) // 0, 1, 2, 3, 4
123 }
124 for(let city of cities) {
125   console.log('Ville avec for in : ', city) // Paris, Marseille, Lille, etc.
126 }
127
128 for(let prop in france) {
129   console.log('france avec for in : ', prop) // capital, continent
130 }
131 for(let val of france.cities) {
132   console.log('france avec for of : ', val) // Paris, Marseille, Lille, etc.
133 }

```


FONCTION



- *"programme dans le programme"*
- On utilise des fonctions pour **regrouper des instructions et les appeler sur demande** :
chaque fois qu'on a besoin de ces instructions, il suffira d'appeler la fonction au lieu de répéter toutes les instructions.
- Pour accomplir ce rôle, le **cycle de vie d'une fonction comprend 2 phases** :
 1. Une phase unique dans laquelle la fonction est **déclarée**
On définit à ce stade toutes les instructions qui doivent être groupées pour obtenir le résultat souhaité.
 2. Une phase qui peut être répétée une ou plusieurs fois dans laquelle la fonction est appelée puis **exécutée**.

```

136 // Déclaration sans paramètre et retourne une valeur
137 function helloWorld() {
138     return 'Hello World'
139 }
140 // Déclaration avec un paramètre et ne retourne pas une valeur
141 function getStatus(age) {
142     if(age >= 18) console.log('Majeur')
143     else console.log('Mineur')
144 }
145
146 const hello = helloWorld()
147 console.log(hello) // Hello World
148 getStatus(20) // Majeur

```

```

150 /**
151  * Déclaration
152  * avec un nombre indefini d'arguments(paramètres)
153  * et retourne une valeur
154  * Notation sous forme de variable
155  */
156 let mySum = function(...args) {
157     let sum = 0
158     args.forEach(nb => sum += nb)
159     return sum
160 }
161
162 console.log(mySum(10,20,30,40)) // 100
163 console.log(mySum()) // 0
164 console.log(mySum(1,2)) // 3

```

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 global, 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, s'il est défini et accessible dans le bloc dans lequel il a été déclaré. 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 lequel il est déclaré.
- 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](#)

Portée des variables let et var

Const et let

```
function varTest() {
  var x = 31;
  if (true) {
    var x = 71; // c'est la même variable !
    console.log(x); // 71
  }
  console.log(x); // 71
}

function letTest() {
  let x = 31;
  if (true) {
    let x = 71; // c'est une variable différente
    console.log(x); // 71
  }
  console.log(x); // 31
}
```

Var

```
var x = 0; // Déclare x comme variable globale du fichier, on lui affecte 0

console.log(typeof z); // "undefined", car z n'existe pas encore

function a() {
  var y = 2; // Déclare y dans la portée de la fonction a
  // Affecte 2 comme valeur à y

  console.log(x, y); // 0 2

  function b() {
    x = 3; // Affecte 3 à la variable globale x
    // Ne crée pas une nouvelle variable globale
    y = 4; // Affecte 4 à la variable externe y,
    // Ne crée pas une nouvelle variable globale
    z = 5; // Crée une nouvelle variable globale
    // et lui affecte la valeur 5.
  } // (lève une ReferenceError en mode strict.)

  b(); // Crée z en tant que variable globale
  console.log(x, y, z); // 3 4 5
}

a(); // l'appel à a() entraîne un appel à b()
console.log(x, z); // 3 5
console.log(typeof y); // "undefined" car y est local à la fonction a
```


EXERCICES



Exercice 1



- 0-exercices/1-ex.md



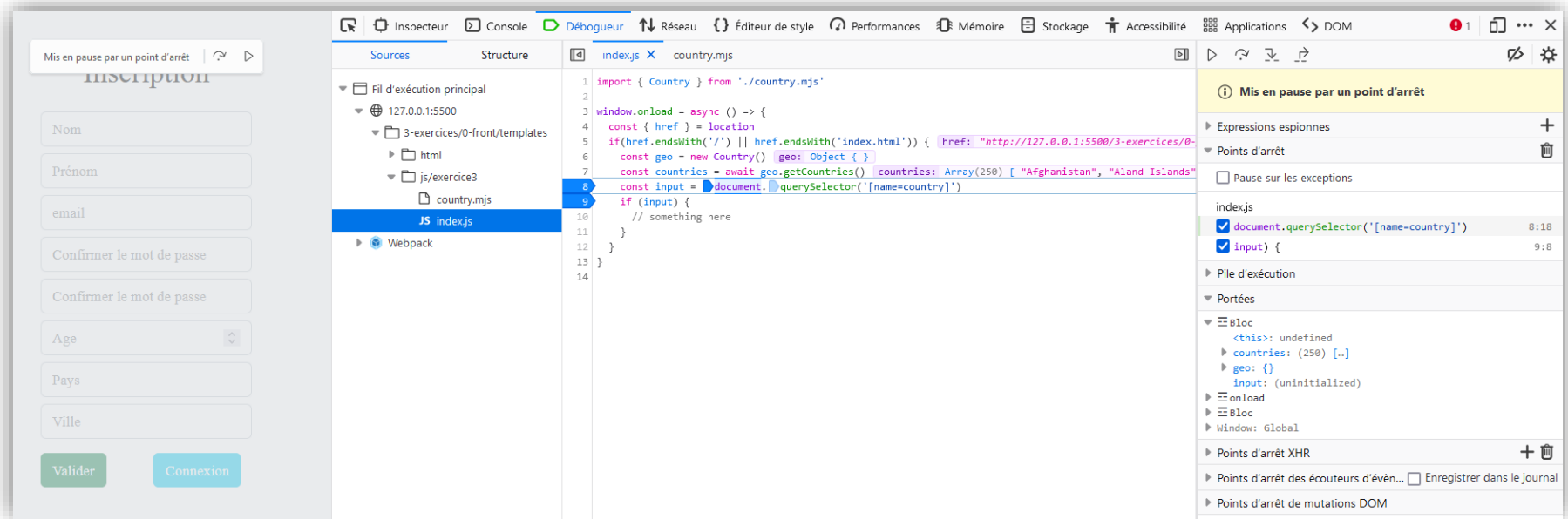
DEBOGAGE



- 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.



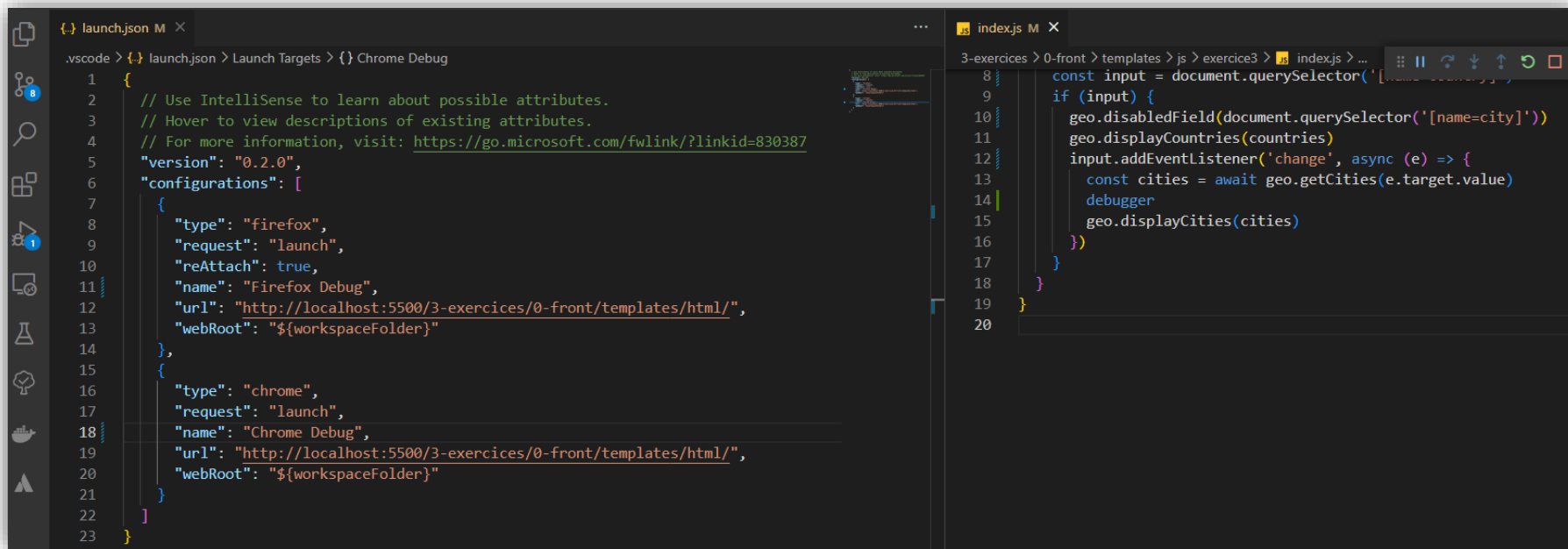
The screenshot displays the Chrome DevTools interface with the Debugger tab active. On the left, the 'Sources' panel shows the file structure of a project, with 'index.js' selected under 'js/exercice3'. The main editor shows the code of 'index.js', with a breakpoint set at line 8. The right sidebar shows the 'Mis en pause par un point d'arrêt' (Paused at breakpoint) panel, which lists the breakpoints and the current execution stack. The execution stack shows the following frames:

- `document.querySelector` (8:18)
- `input` (9:8)

The 'Portées' (Scopes) panel shows the current scope, including variables like `countries` (Array(250)), `geo` (Object), and `input` (uninitialized).

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.



The screenshot shows the VS Code interface with two files open. The left file is `launch.json`, which is a JSON configuration for launching debuggers. It defines two configurations: 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, to pause execution at that point.

```

{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "type": "firefox",
      "request": "launch",
      "reAttach": true,
      "name": "Firefox Debug",
      "url": "http://localhost:5500/3-exercices/0-front/templates/html/",
      "webRoot": "${workspaceFolder}"
    },
    {
      "type": "chrome",
      "request": "launch",
      "name": "Chrome Debug",
      "url": "http://localhost:5500/3-exercices/0-front/templates/html/",
      "webRoot": "${workspaceFolder}"
    }
  ]
}

```

```

const input = document.querySelector('[name=city]');
if (input) {
  geo.disabledField(document.querySelector('[name=city]'));
  geo.displayCountries(countries);
  input.addEventListener('change', async (e) => {
    const cities = await geo.getCities(e.target.value);
    debugger;
    geo.displayCities(cities);
  });
}

```

- 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](#)

Demo : débogage avec Visual Studio Code



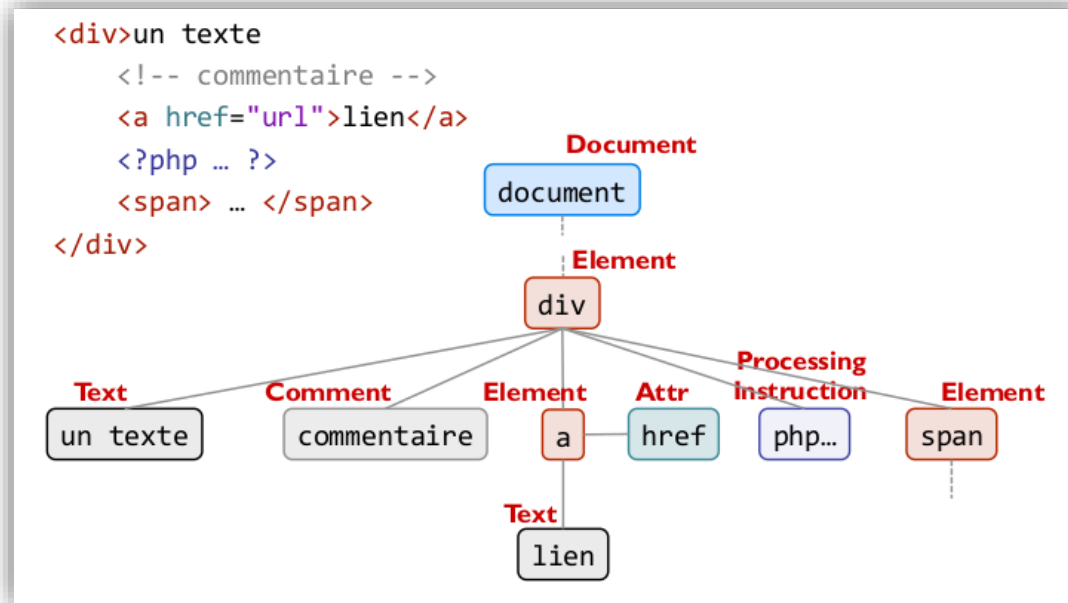
III. DOM



m2information.fr

DOM : Document Object Model

- Le DOM est une API permettant de représenter et de 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

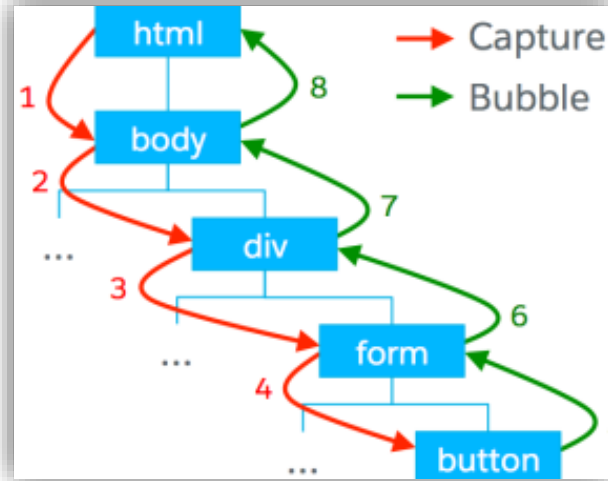
- *el.createElement(tag)*: crée l'élément à partir du tag donné.
- *el.insertAdjacentHTML(position, el)*: insère un nouveau nœud HTML dans l'élément *el*/par rapport à la position spécifiée.
- Position prend les valeurs suivantes :
 - *Beforebegin*: avant l'élément lui-même ;
 - *Afterbegin*: juste à 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*.

Les événements

- Pour gérer les événements, il faut à partir d'un élément qu'on récupère avec *document.querySelector*, utiliser la méthode *.addEventListener* sur cet élément.
- La méthode *addEventListener* prend 3 paramètres (arguments)
 1. *Event* : nom de l'événement (exemple *click* ou *mouseenter* ou *keypress*, etc.)
 2. Une fonction de *callback* qui sera exécuté au moment où l'événement aura lieu
 3. Un *booléen* qui détermine la direction de la propagation de l'événement. (*true* => *capture*, *false* => *bubble*).
- Phase de **bouillonnement** (*Bubble*) : propagation de l'événement en remontant la hiérarchie du DOM. C'est le 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*.
C'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 au formulaire

- Input : événement déclenché lorsqu'une valeur est saisie dans le champ
- Change : événement déclenché lorsque la valeur d'un champ change
- Focus : événement déclenché lorsqu'il y a le focus sur le champ
- Blur ; événement déclenché lorsqu'il y a une perte du focus sur le champ
- Submit : événement déclenché à la soumission du formulaire
- Reset : déclenché lorsque les données d'un formulaire sont toutes 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

EXERCICES



Exercice 2



- 0-exercices/2-ex.md

IV. MANIPULER LE CSS AVEC JAVASCRIPT



RAPPELS CSS



Historique

1996

- Naissance du **CSS1**

2001

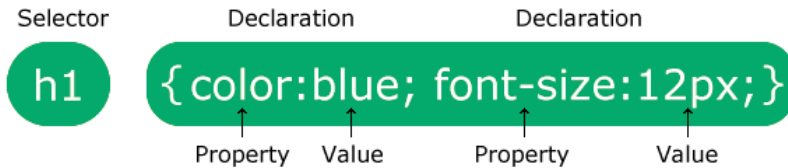
- **CSS 2.1**

1998

- **CSS2**

Version
actuelle
CSS3

Source image W3Schools



- Sélecteur
 - Élément *HTML* ciblé
- Propriété
 - Élément de style ciblé
- Valeur
 - Valeur à appliquer à l'élément de style ciblé
- */*Commentaires en CSS*/*

Mobile First et Responsive Design

[Source image Frederic Gonzalo](#)



Responsive Web Design

Mobile First Web Design



- Le **Mobile First** est la méthodologie de conception inventée par *Luke Wroblewski* en 2011.
- **Concevoir** son application d'abord sur **mobile** puis adapter le contenu pour les écrans plus grands.
- Le **Responsive Web Design** est une méthode qui permet de créer une page Web qui **s'adapte automatiquement** par rapport à la **taille de l'écran** d'affichage.
- Ces méthodes sont possible grâce à la mise en place des *medias queries* dans le style **CSS**.

1 seul code **HTML** et des règles **CSS** différentes selon les tailles d'écran.

Tailles des écrans

Tailles	
Maximum 600px de large	Smartphone, montre
Minimum 600px de large	Orientation portrait tablette et phablette (entre smartphone et tablette)
Minimum 768px de large	Orientation paysage tablette
Minimum 992px de large	Ordinateurs portable et de bureau
Minimum 1200px de large	Écrans de télévision , ordinateurs portables, ordinateurs de bureau très large

- Les tailles des écrans peuvent varier en fonction des librairies CSS.

Ici, les tailles sont celles définies par le site [W3schools](https://www.w3schools.com/css/css3_media_queries.asp)

Mise en place des Media Queries côté CSS

Source image Christian Lisangola

```
@media screen and (min-width: 480px) {
  /* Votre code CSS */
}

@media screen and (min-width: 768px) {
  /* Votre code CSS */
}

@media screen and (min-width: 992px) {
  /* Votre code CSS */
}

@media screen and (min-width: 1200px) {
  /* Votre code CSS */
}
```

Source image Megawebdesign



Quelques méthodes et attributs du DOM associées au CSS

- *el.attributes.class.value* : valeur de l'attribut class.
- *el.classList.add('alert-danger')* : ajoute la valeur *alert-danger* dans la liste des valeurs de l'attribut class de l'élément.
- *el.classList.remove('alert-danger')* : supprime la class *alert-danger* dans la liste des valeurs de l'attribut class de l'élément.
- *el.textContent* : contenu textuel d'un nœud et de ses descendants.
- *el.outerHTML* : contenu textuel d'un élément et de ses descendants incluant la balise HTML de l'élément.
- *el.remove()* : retire (supprime) l'élément du DOM.
- *elements.length* : nombre des éléments.

[Source MDN](#)

Mise en place des Media Queries côté JavaScript

- La gestion des *media queries* peut s'effectuer côté JavaScript avec la méthode *matchMedia* combinée au DOM et les événements
- [Documentation des éléments paramètres que l'on peut associer à la méthode matchMedia](#)

Demo : matchDemo

EXERCICES



Exercices 3 et 4



- 0-exercices/3-ex.md
- 0-exercices/4-ex.md



V. ASYNCHRONIE



m2iformation.fr

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

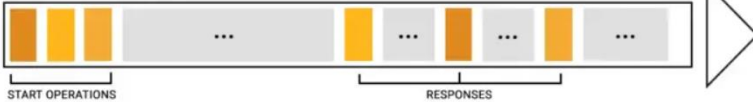
SET TIMEOUT
FOR 5 SECONDS

DATABASE
OPERATION

Synchronous



Asynchronous

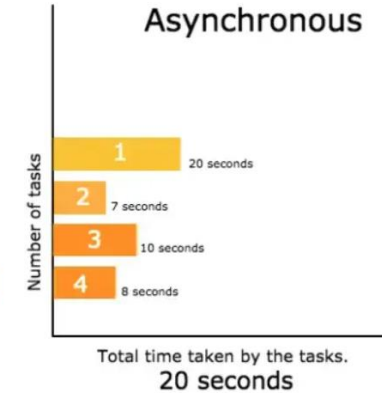


[Source de l'image JavaScript.plainenglish.io](https://www.javascriptplainenglish.io/)

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.

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"
        }
      ]
    }
  ]
}
```

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*.

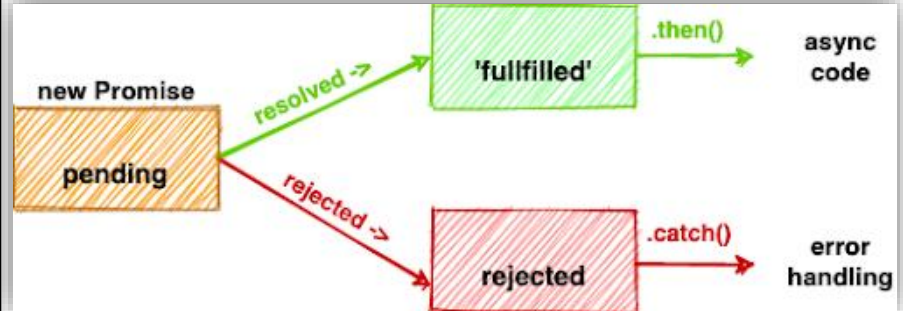
Implémentation Promise

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 ))
```

XMLHttpRequest

- Objet développé initialement par **Microsoft** pour effectuer des traitements asynchrones. De nos jours, cet objet est **largement moins utilisé**, mais est toujours supporté par les navigateurs.
- Les méthodes :
 - **.abort()** : annule la requête en cours ;
 - **.getResponseHeaders()** : headers (en-têtes) ;
 - **.open(method, url, boolean)** : démarre la requête avec une méthode *HTTP*, l'*URL* du serveur et le mode synchrone (false) ou asynchrone (true) ;
 - **.send(data)** : envoie des données au serveur.

Les propriétés de l'objet XMLHttpRequest

- Propriété *.readyState*
 - *0*: non initialisé, la méthode *.open()* n' a pas encore été appelée.
 - *1*: initialisé, la méthode *.send()* n'a pas encore été appelée.
 - *2*: *headers* (en-têtes de réponse) disponibles.
 - *3*: *body* (données reçues) partiellement disponible.
 - *4*: *body* entièrement disponible.
- Propriété *.status* (Code HTTP du retour du serveur sur la requête effectuée)
 - 20x: OK
 - 40x : erreur côté client (navigateur)
 - 50x : erreur côté serveuravec x, un chiffre.
- Propriété *.responseText* : obtenir les données renvoyées par le serveur au format texte brut.
- Propriété *.responseXML* : obtenir les données renvoyées par le serveur au format XML.

Implémentation en JavaScript

```
const getUsers = () => {  
  const myPromise = new Promise((resolve, reject) => {  
    const xhr = new XMLHttpRequest()  
    xhr.onreadystatechange = function () {  
      if (xhr.readyState === 4 && (xhr.status === 200 || xhr.status === 0)) {  
        const res = xhr.responseText  
        resolve(res)  
      } else if (xhr.status === 404) {  
        reject('404 not found')  
      }  
    };  
    xhr.open('GET', 'https://jsonplaceholder.ir/users')  
    xhr.send()  
  })  
  myPromise  
    .then((res) => console.log("result", res))  
    .catch((e) => console.log("error message", e))  
}  
  
getUsers()
```

EXERCICES



Exercice 5



- `0-exercices/5-ex.md`

VI. NOUVEAUTÉ DEPUIS ES6



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 `=>`
- Déstructuration des tableaux et objets : créer des variables à partir des propriétés d'un objet ou des éléments d'un tableau

```
hello() => { console.log('hello world') }
```

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
}

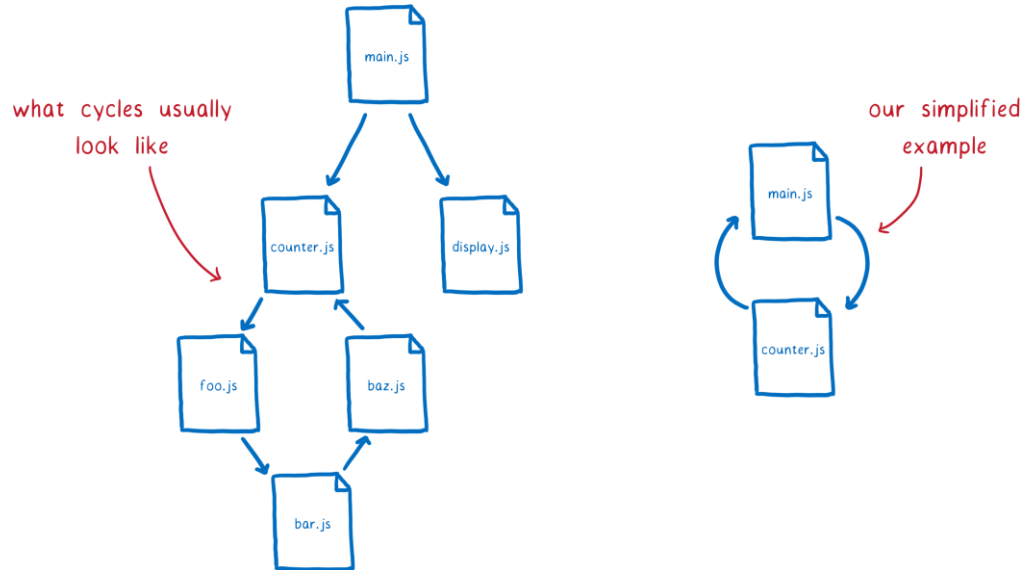
const {color} = zoe
console.log('zoe color', color) // black
```

Objet littéral

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

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.

Différences entre CommonJS et ESM

Les *Bundlers*, que nous verrons plus tard, permettent de résoudre les problèmes d'incompatibilité entre modules.

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

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 de l'utilisation des modules

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

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.
- L'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 gestionnaires de packages pour Node tels que
 - [Yarn de Facebook](#)
 - [PNPM](#)


Émargements

Pensez à vérifier les feuilles d'émargement

Chao Manu

LEVI STRAUSS Claude

HULOT Nicolas

A large rectangular box containing a handwritten signature in black ink. The signature is 'Levi', written in a cursive style. The 'L' is large and loops around. The 'e' and 'v' are connected, and the 'i' has a small dot. The 'S' is also connected to the 'v'.

A small blue icon of an eraser with a pencil, located at the bottom right corner of the signature box.

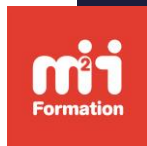
ANNULER

ENREGISTRER

Évaluations formateur

N'oubliez pas les évaluations formateur avant de partir.





FIN



m2information.fr