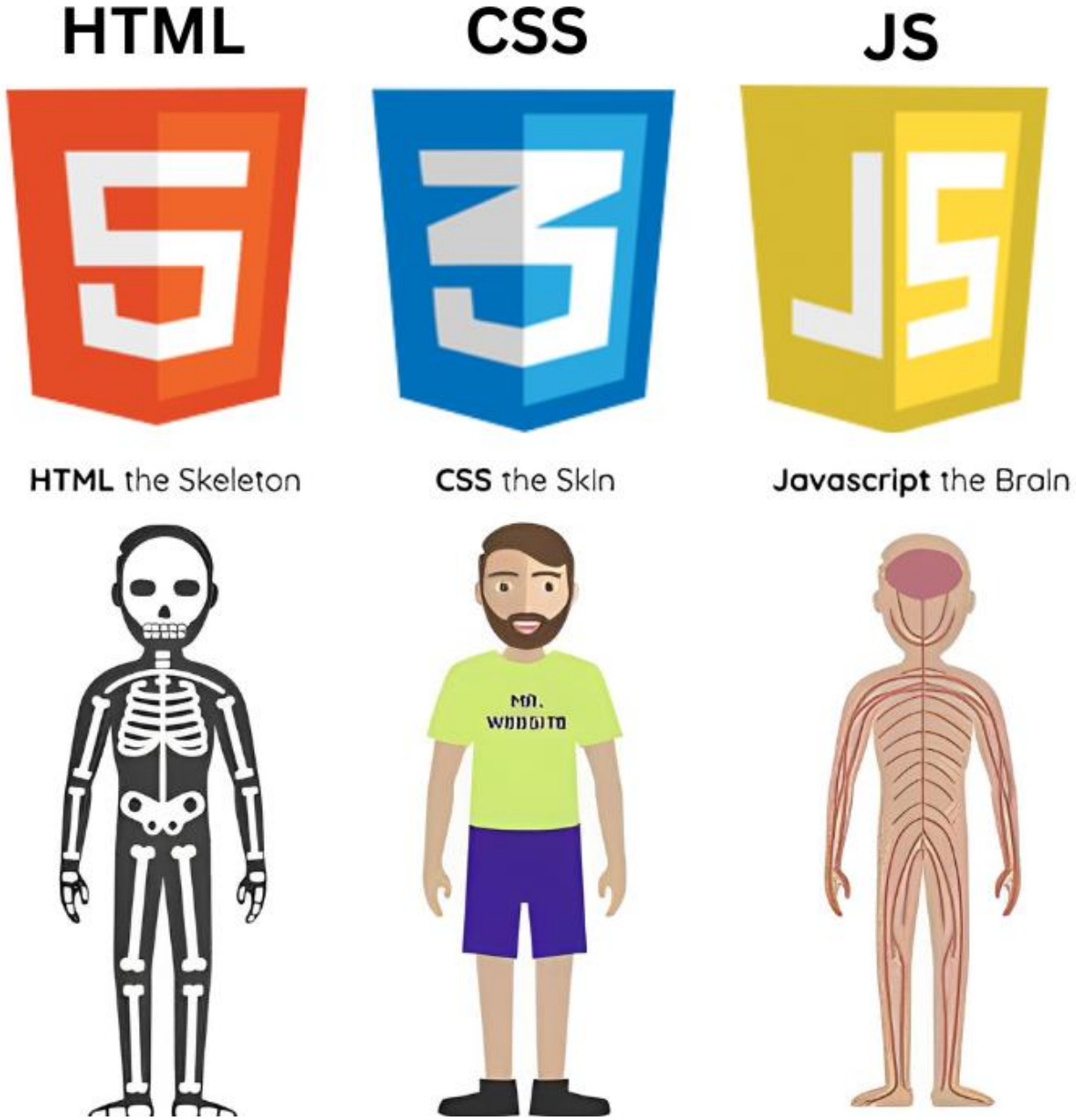


Cours JS

BC DEV ITIS

Evry-Courcouronnes

GLODIE TSHIMINI

Consultant Formateur et Développeur depuis 2017

- Avant-hier, développeur chez **INRAE** à Paris
- Hier, **Tech Lead** dans une agence digitale à Saint Raphaël
- Aujourd'hui, **consultant indépendant** dans la formation et le développement d'applications Web
- Certifié Professional Scrum Developer (PSD I)
- student@tshimini.fr

Avant de commencer

Les ressources

- Cours, exercices et corrections disponibles en ligne depuis ce lien GitHub
- Les démonstrations de code et les corrections des exercices seront envoyées au fur à mesure de l'avancement du cours sur GitHub

Validation des acquis

- Exercices
- QCM/Quiz
- Devoirs maisons

Avant de commencer

Objectifs pédagogiques

- Développer des pages Web dynamiques
- Communiquer avec d'autres applications pour partager des données

Programme

- Variables
- Fonctions
- Objets
- DOM
- AJAX

▶ Durée de la formation : 14H00

Plan du cours

I. Généralités

II. Les bases de la programmation avec JavaScript

III. Le DOM

IV. AJAX

Comment utiliser le JS dans une page Web ?

- 2 méthodes :

  1. Dans un fichier externe *index.js* via une liaison au sein du HTML par l'intermédiaire de la balise enfant ***script*** du ***head :***

     <head>

     <script type="text/javascript" src="index.js"></script>

     </head>

  2. Ou directement dans le document HTML avec l'utilisation de la balise ***<script></script>*** placé juste avant la fermeture la balise </*body*>

- Il existe d'autres méthodes que nous ne verrons pas dans le cadre de ce cours.

- **Nous utiliserons uniquement la deuxième méthode pour ce cours**

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <!--Méthode 1-->
    <script src="index.js"></script>
</head>
<body>
    <!--Méthode 2-->
    <script src="main.js"></script>
</body>
</html>
```

I. Généralités

Caractéristique du langage

- A la différence du *HTML* et *CSS* (langages de structure) que nous avons vu précédemment, JavaScript est un **langage de programmation**
- Langage de programmation = créer un **programme** (liste d'instructions écrite par un humain) **exécuté** sur une **machine** (ordinateur)
- *JavaScript* est **standardisé par *ECMASCRIPT*** (European Computer Manufacturers Association)
- **A ne pas confondre avec *Java*** qui est un autre langage de programmation
- Langage qui **s'exécute** à la fois côté **client** (*Front-end*) et côté **serveur** (*Back-end*)
  - Pour un usage côté client avec le navigateur que nous verrons dans le cadre de ce cours
  - Et serveur avec l'environnement de développement ***NodeJS***

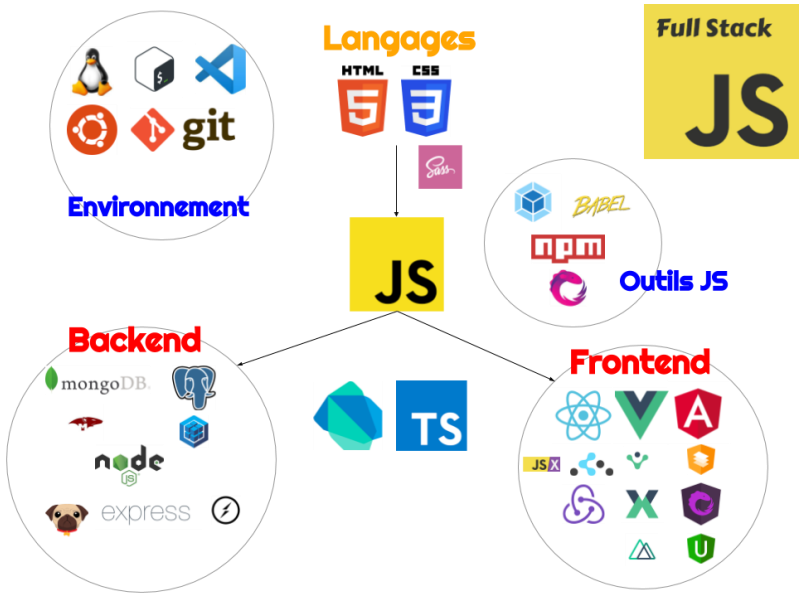
Un peu d'histoire

1995	1997	1998	1999	2009	2015	Depuis 2016
• Création par Netscape	• *ECMAScript* 1 (*ES1*)	• *ES2*	• *ES3*	• *ES5*	• *ES6*	• On parle globalement toujours d'*ES6* pour toutes les versions après 2016

On rattache depuis plusieurs années la version à l'année, *ES2017*, *ES2022*, etc.

Tout l'écosystème JavaScript

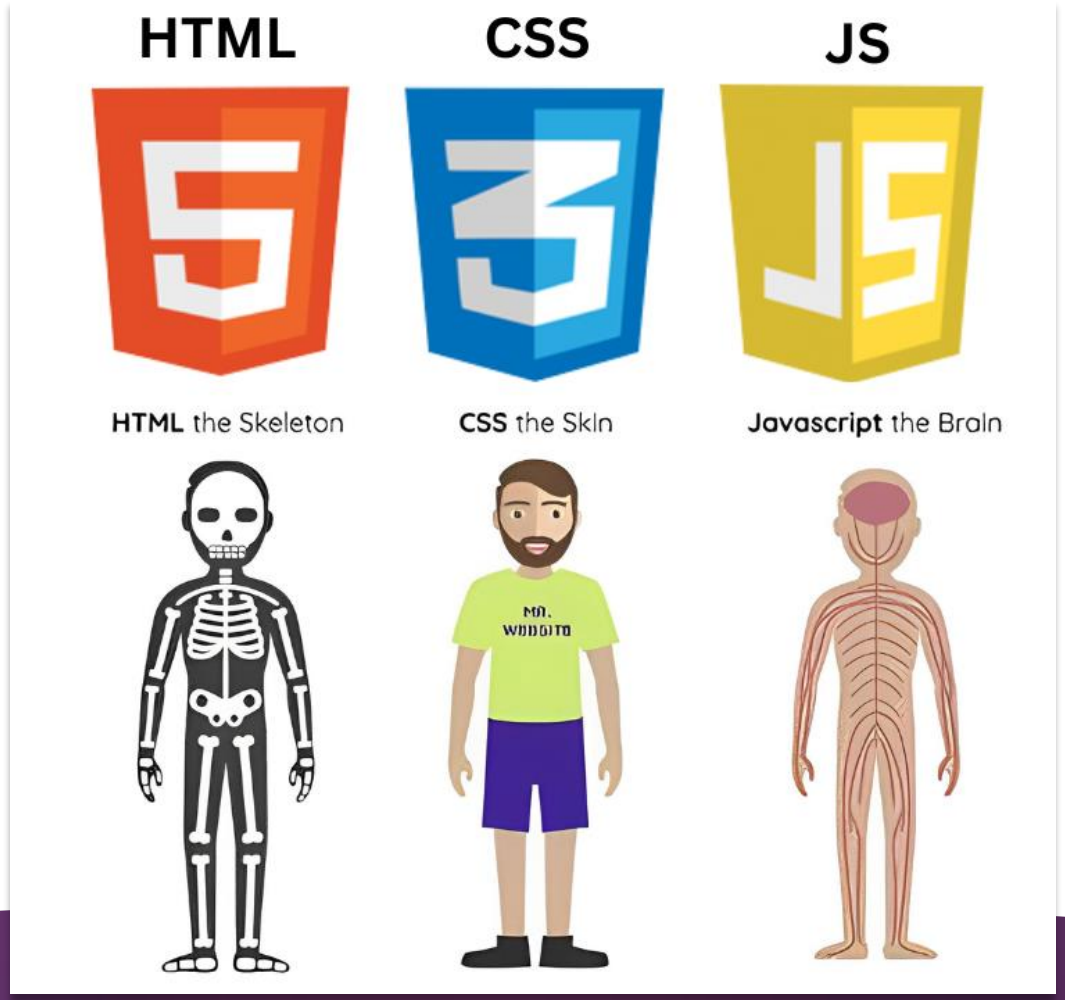
Source image blog.dyma



Source image codes.learning

Concrètement à quoi sert JavaScript ?

- **Exécuté et interprété par le navigateur**
- Permet de **rendre les pages Web dynamiques**
  - Manipuler (créer, modifier et supprimer) les éléments de la page web (balises HTML, propriétés et valeurs CSS)
  - Gérer les formulaires de contact (validation)
  - Manipuler la fenêtre (*window*)
    - Géolocalisation
    - Gestion de l'onglet (dimension, ouverture, fermeture, etc.)
- Et Interagir avec d'autres outils mise à disposition par le navigateur
  - *Storage* : stockage côté navigateur)
  - *Canvas* : dessin
  - *Drag & Drop* : glisser, déposer des éléments sur la page
  - *Offline* : travailler en hors ligne
  - *Web Workers* : pour des traitements lourds gagner en fluidité lors de la navigation sur la page
  - *Etc*



Source image Anass Essadikine

Installation

- Nous allons ajouter 2 extensions à notre éditeur de code VSCode
  1. JavaScript Debugger Companion Extension de Microsoft
  2. JavaScript (ES6) code snippets de Charalampos Karypids

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 mais je vous recommande fortement de toujours l'utiliser

- Les commentaires s'écrivent de deux façons :
  - *// commentaire sur une ligne*
  - */* commentaires*
    sur plusieurs
    lignes
    **/*

Toute la documentation

- W3schools : documentation basique HTML, CSS et JS
- MDN : documentation HTML, CSS et JS plus poussée et détaillée
- Tous les événements JS pour la partie dédiée au DOM (Document Object Model)

II. Les bases de la programmation avec JavaScript

Variables et constantes : stockage des données du programme

Variable

- Une information (donnée) stockée dans la RAM de la machine qui execute le programme.
- Information modifiable
- Manipuler les variables
  - Mot clé **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.

Typage : domaine de définition de la donnée (nature)

- Langage **faiblement typé**
  - Permet de manipuler facilement des données de nature différente sans les convertir et d'effectuer des opérations entre elles
  - Un langage fortement typé impose de convertir la nature des données avant d'effectuer certaines opérations
- Le typage est **dynamique** à l'interprétation (exécution) du code
- 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 autre un typage fort)
- JavaScript possède 7 types primitifs

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

Concaténation, Transtypage

- Concaténation = mettre bout à bout des chaines 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)
```

Tableaux : définition

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 :

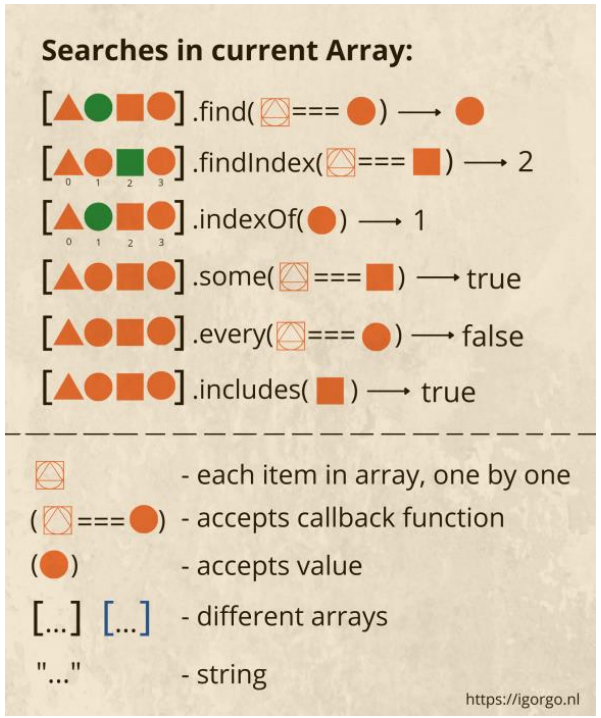
- *monTableau[0] = 4;* : Mettre 4 à la première position 0
- *console.log(monTableau[1])* : Accès à l'élément situé à l'indice 2 du tableau ici *undefined*

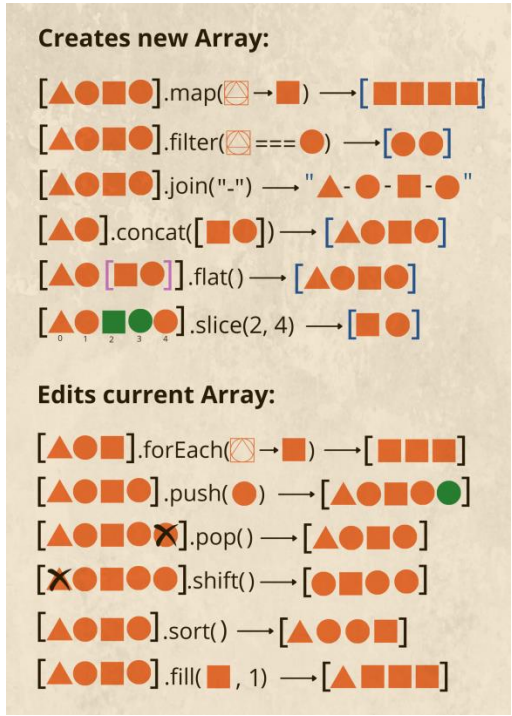
Tableaux : exemples

```
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

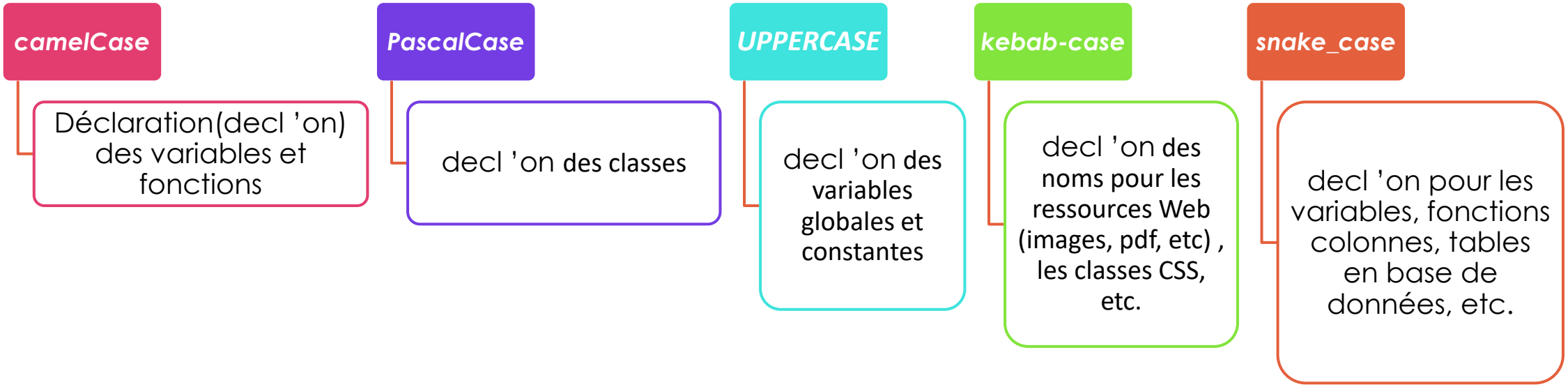
Source images Igor Gonchar





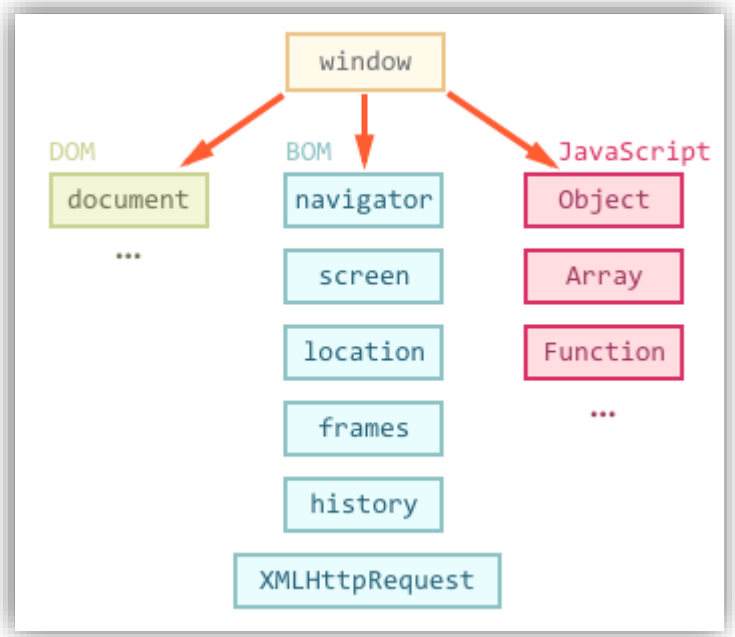
Convention de nommage des variables, fonctions, etc.

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



En JavaScript, tout est objet

- ***DOM*** : objets spécifiques à la manipulation du document HTML.
- *BOM* : objets spécifiques à la manipulation du navigateur.
- *APIs JavaScript* : tout le reste
- 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.

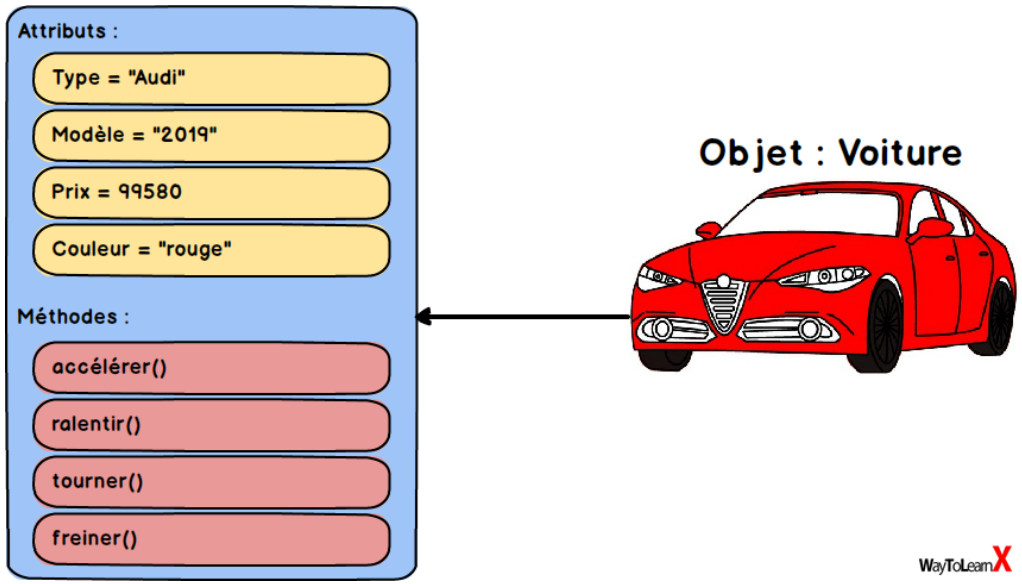


Source image medium.com/@fknussel

Rappels programmation orientée objet

- 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.

Source image waytolearnx.com



Objets en JavaScript : notation littérale {}

Pour accéder au attribut ou méthodes d'un dans l'objet, on y fait référence via le mot clé **this**.

Pour y accéder aux attributs:

- article.designation
- article.calculerPrixTTC()

```
const article = {
  description: "T-shirt",
  reference: 2864738,
  priceExVAT: 25,
  computedPriceInVAT: function () {
    return this.priceExVAT * 1.2;
  },
};
```

Objets en JavaScript : notation classe ES6

- Notation similaire à l'implémentation de la programmation objet des autres langages de programmation tels que Java, PHP, etc.

```
class Article {
  constructor(id, designation, reference, priceExVAT) {
    this.id = id;
    this.designation = designation;
    this.reference = reference;
    this.priceExVAT = priceExVAT;
  }
  computedPriceInVAT() {}
}
```

Structure conditionnelle *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 exécuté (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  }
```

Structure conditionnelle SWITCH

- Identique à *if*, plus lisible dans les cas ou 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 (boucles)

- Parcourir des objets en executant un bloc de code tant qu'une certaine condition est vérifiée
- ***FOR*** : *for (let i=0; i<10; i++) { … };*
  - Boucle adapté **lorsqu'on connait le nombre d'iteration à effectuer**
- ***WHILE*** : while (condition) { … };
  - Boucle adapté pour parcourir des éléments **lorsqu'on ignore le nombre d'iteration à effectuer**
- ***DO … WHILE*** : do {… } while {condition};
  - Similaire à while sauf qu'il s'execute au moins une fois
- Le mot clé **break** permet de sortir d'une boucle prématurément.
- Il existe d'autres structures itératives avec JavaScript, *for in* et *for of*

Documentation de toutes les boucles itératives avec JavaScript

Exemples des structures itératives

```
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)
```

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.**

Exemples des fonctions

```
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
```

```
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
```

A vous de jouer

Jeux Labyrinthe de Blockly

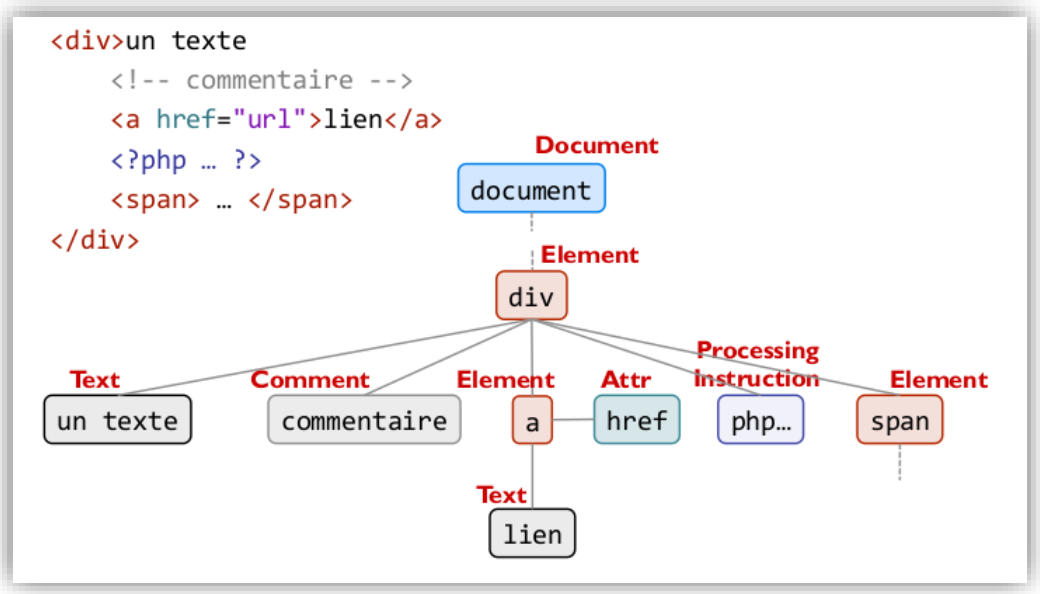
Créez un fichier index.js, à chaque tour copier/coller la fonction affiché par le site dans le fichier JavaScript précédent

III. Le DOM

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**.

Source de l'image centralsupelec



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

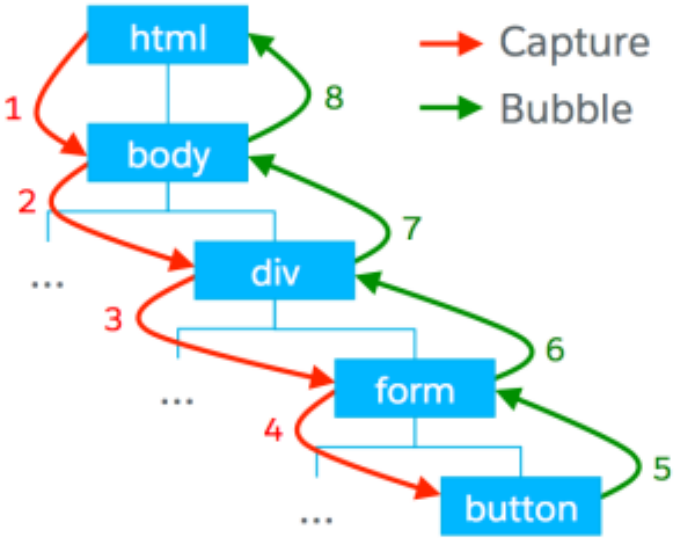
Propagation des événements

- 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

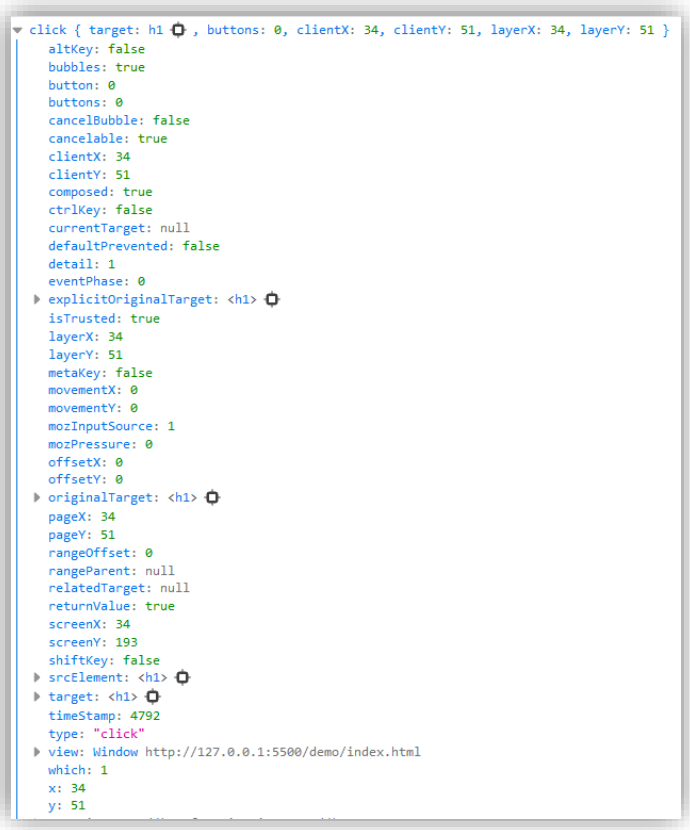


Objet event caractérisant l'événement déclenché

- 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.
- Les événements permettent de **rendre les pages web plus interactives**.

Objet event en détails

- Vous pouvez utiliser la fonction ***console.dir(evt)*** ou ***.log(evt)*** pour afficher dans la console du navigateur toutes les informations disponibles décrivant l'objet *evt* déclenchée
- Toutes informations sont récupérables avec la notation pointée depuis l'objet, par *evt.clientX*, *evt.target.value*, etc.



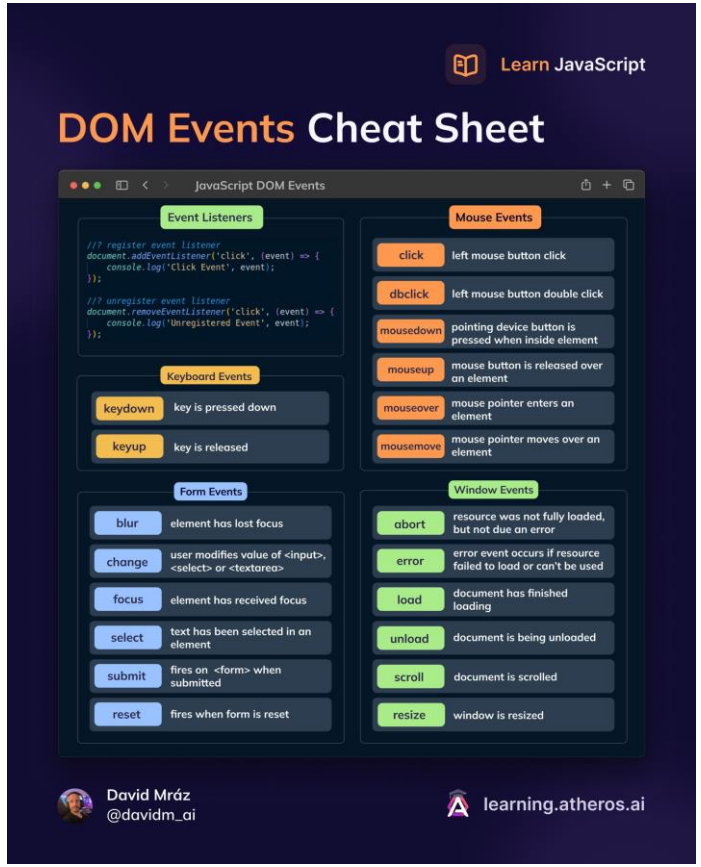
Méthodologie utilisation des événements

1. **Sélectionnez** un élément HTML.
2. **Ajoutez un écouteur événement** de type : souris (***click***, *dbclick*, *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 HTML

- ***Input*** : événement déclenché lorsqu'une **valeur est saisi** 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 tous supprimées (un reset)



Quelques méthodes et attributs du DOM associées à la manipulation du CSS

- ***el.attributes.class.value*** : **valeur** de l'attribut **class**
- ***el.classList.add('alert-danger')*** : **ajoute** la **classe** *alert-danger* à l'élément *el*
- ***el.classList.remove('alert-danger)*** : **supprime** la **class** *alert-danger* de l'élément *el*
- ***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

IV. AJAX

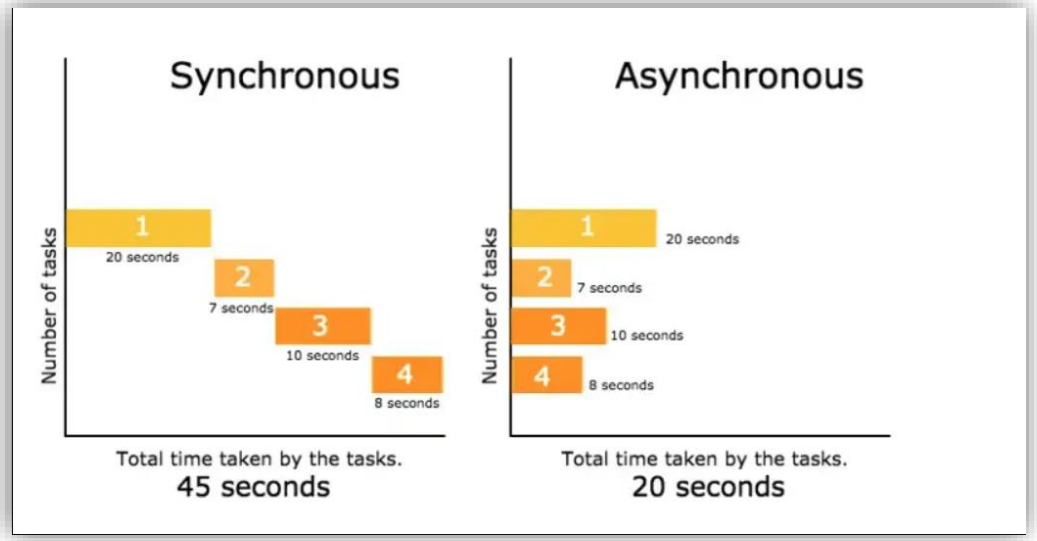
Définition AJAX

- 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 pour effectuer des traitements asynchrones.

Synchrone VS Asynchrone

Source de l'image JavaScript.plainenglish.io

Source de l'image 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 (**chargement plus rapide de la page**)
- Initialement les données étaient envoyées/reçues sous le format ***XML***. Depuis une décennie, le format ***JSON*** a pris le dessus grâce à ses caractéristiques que nous verrons juste après.

Format textuel XML pour partager les données entre applications

- **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
<?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 pour partager les données entre applications

- **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.

Fetch : une méthode pour faire de l'asynchrone

- 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.

Exemple 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 ))
```

FIN

MERCI POUR VOTRE ATTENTION

GLODIE TSHIMINI – ITIS 2025/2026