

M2CCI – M2 GEOMAS 2026-2027

cours TW (Technologies du Web)

Introduction au langage JavaScript (1ère partie)

Philippe Genoud

Philippe.Genoud@univ-grenoble-alpes.fr

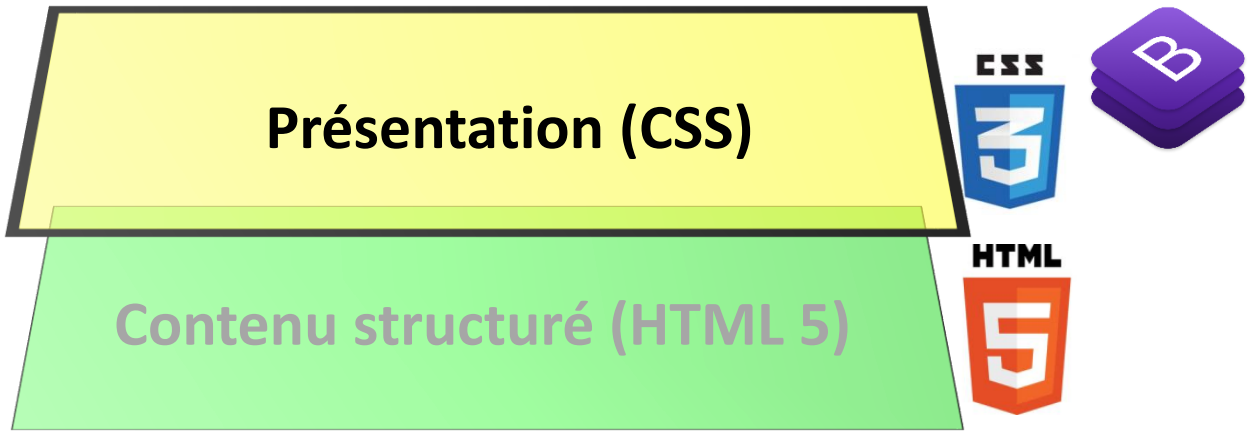


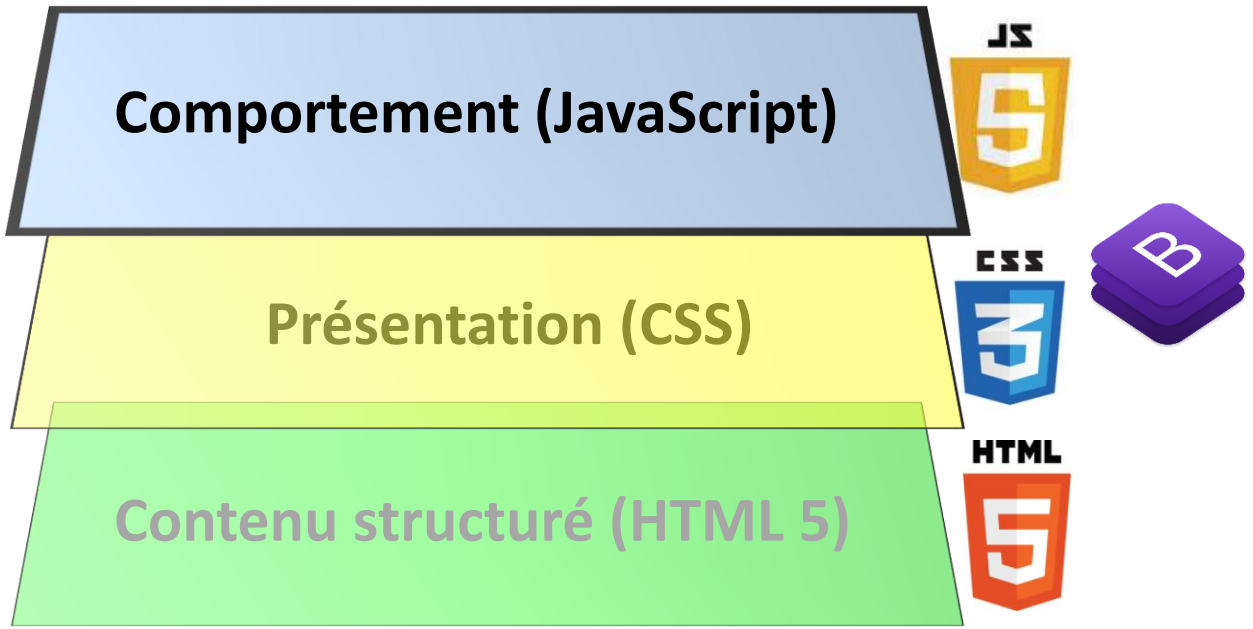
This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Dernière modification : 15/10/2026 16:52

Contenu structuré (HTML 5)



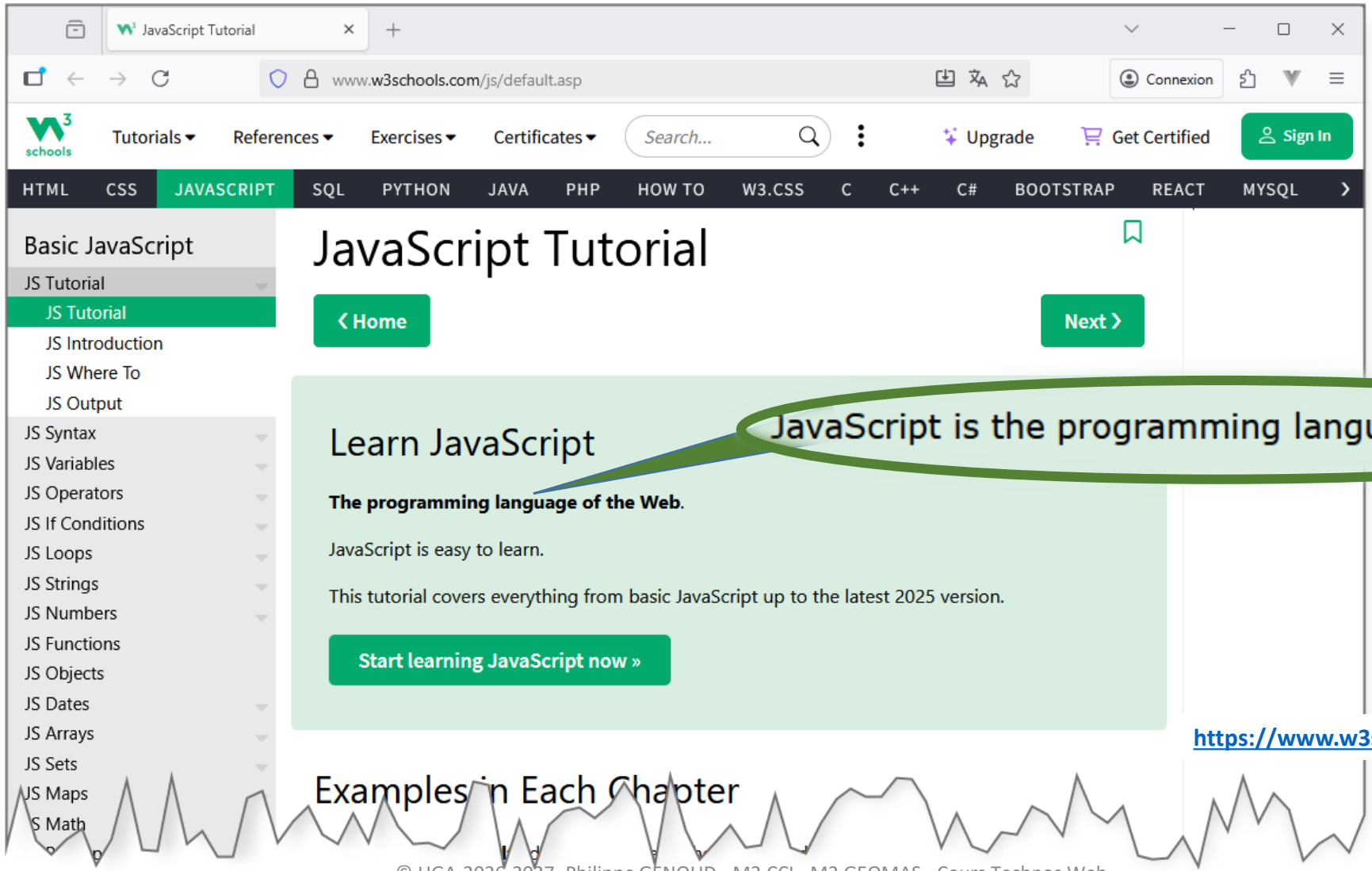

Présentation (CSS)
Contenu structuré (HTML 5)
CSS
HTML


Comportement (JavaScript)
Présentation (CSS)
Contenu structuré (HTML 5)
JS
CSS
HTML

- Historique et évolution de JavaScript
- JavaScript côté client et JavaScript côté serveur
  - JavaScript dans le navigateur
  - JavaScript côté serveur (node.js)
  - (WebAssembly)
- Intégration de JavaScript dans une page Web
- Types de bases et variables
  - déclaration de variables
  - constantes
  - type d'une variable
  - opérateurs et expression
- Structures de contrôle

Motivation

- JavaScript un langage incontournable du développement Web



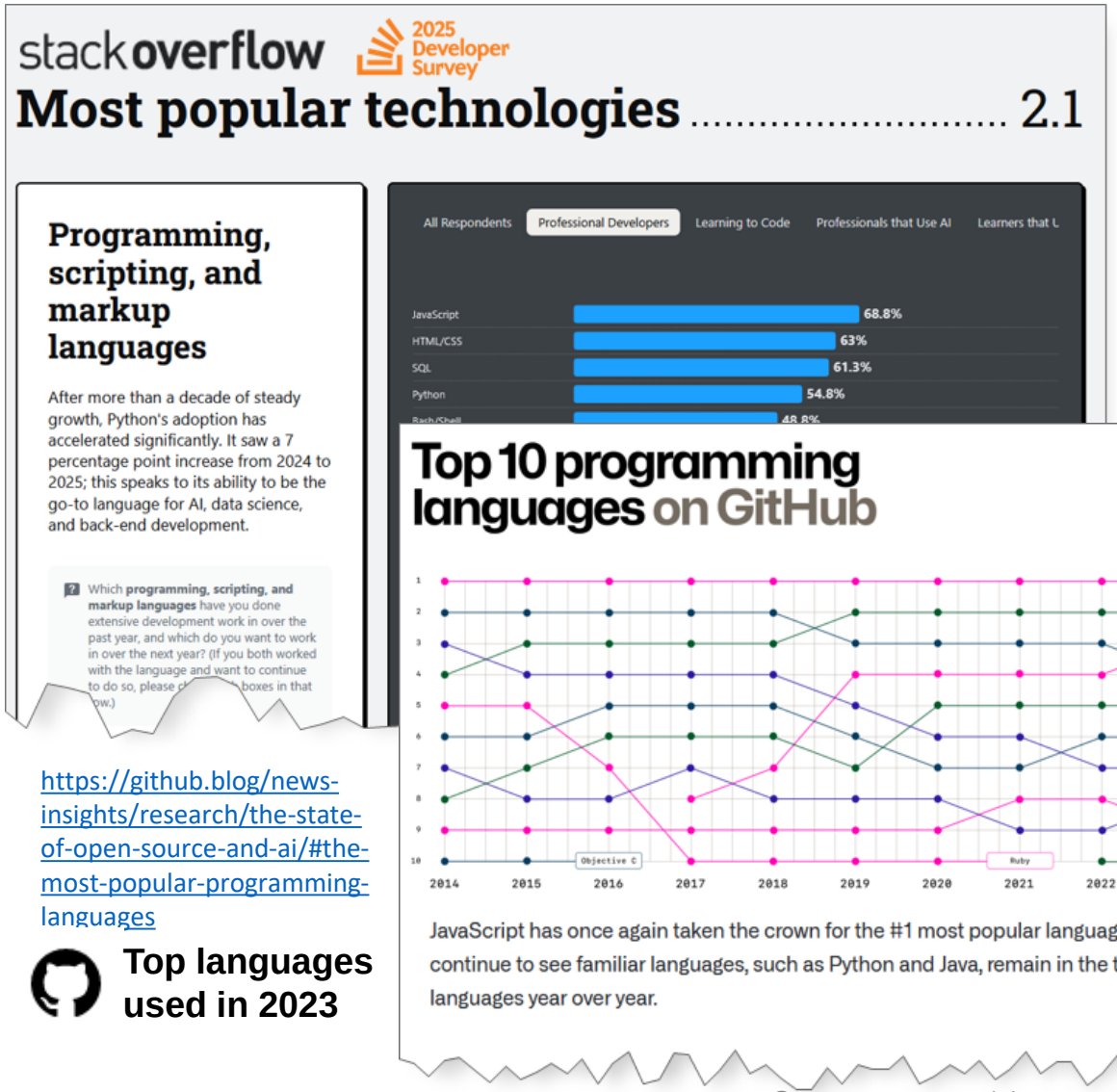
JavaScript is the programming language of the Web.

https://www.w3schools.com/js/default.asp

Motivation

- JavaScript un langage incontournable du développement (web)

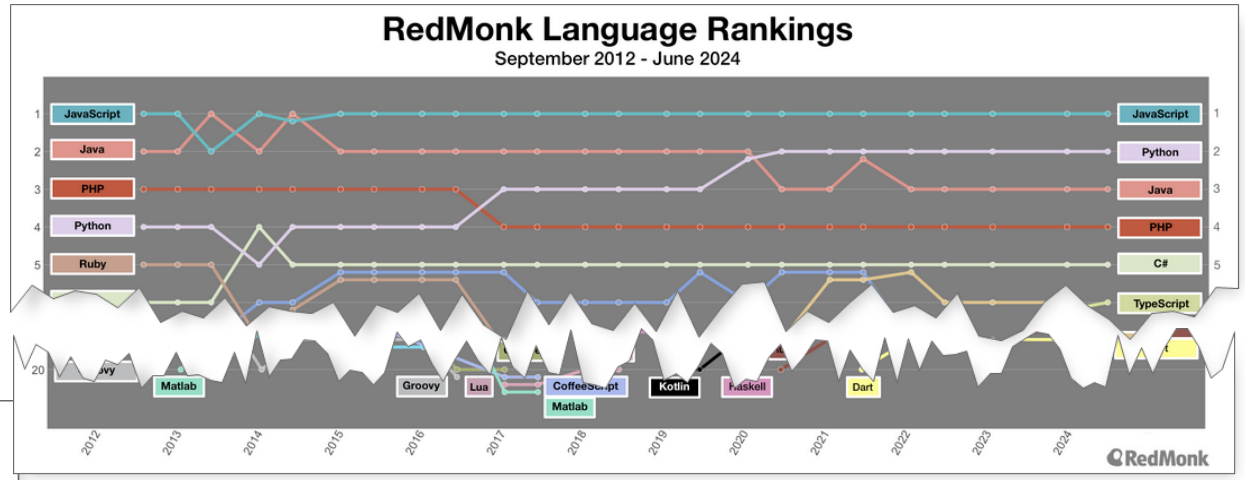
https://survey.stackoverflow.co/2025/technology#most-popular-technologies-language-prof



https://github.blog/news-insights/research/the-state-of-open-source-and-ai/#the-most-popular-programming-languages

Top languages used in 2023

RedMonk
the developer-focused industry analyst firm https://redmonk.com/rstephens/2024/09/12/top20-jun2024/



Fullstack Academy

The 15 Best Programming Languages to Learn in 2024 By David Yang

https://www.fullstackacademy.com/blog/nine-best-programming-languages-to-learn



https ://youtu.be/uCyVa9_gUWs

1. JavaScript

With increasing demand for dynamic web applications, it's nearly impossible to become a professional software developer without learning JavaScript. According to Stack Overflow's 2024 Developer's Survey, JavaScript is the most popular programming language among developers for the eleventh year in a row with 62% of survey respondents reporting that they had used it in the past year.

A majority of the web's most popular sites, from Facebook and Twitter to Gmail and

et bien d'autres encore https://tiobe.com/tiobe-index/ ...

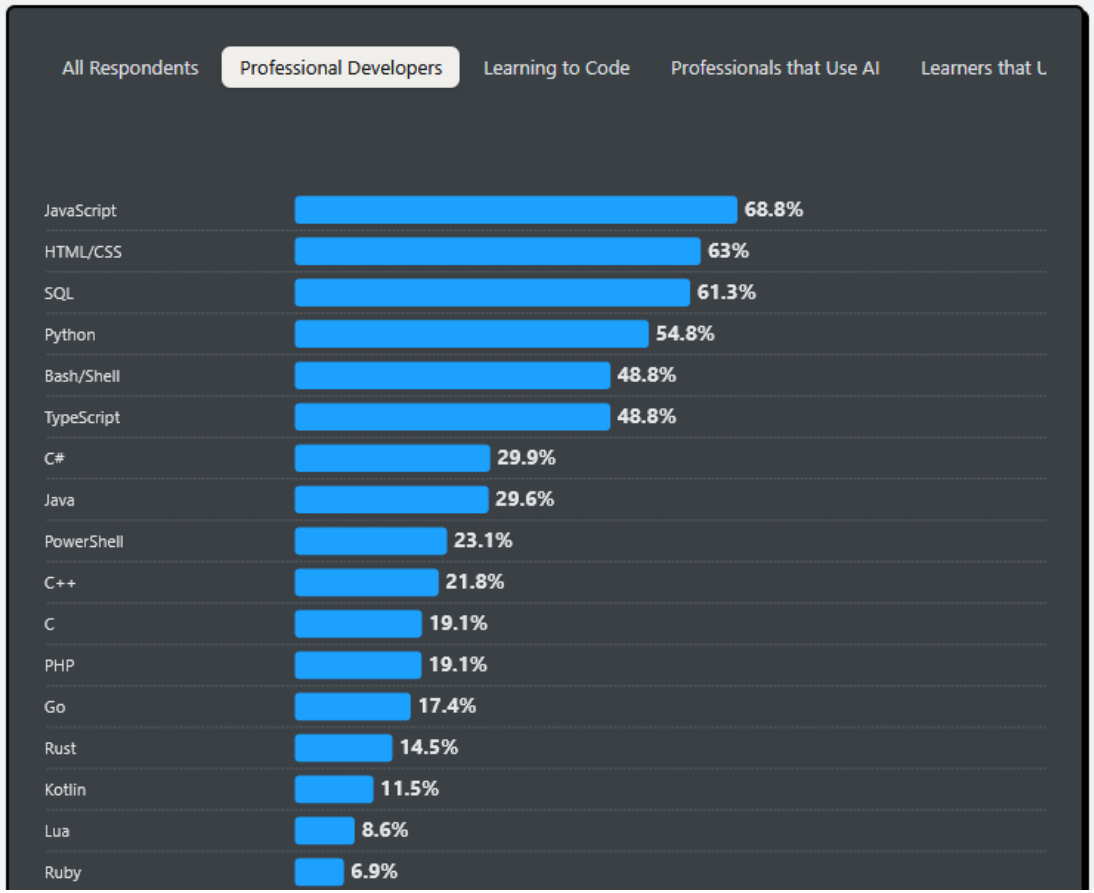
stackoverflow 2025 Developer Survey

Most popular technologies 2.1

Programming, scripting, and markup languages

After more than a decade of steady growth, Python's adoption has accelerated significantly. It saw a 7 percentage point increase from 2024 to 2025; this speaks to its ability to be the go-to language for AI, data science, and back-end development.

Which **programming, scripting, and markup languages** have you done extensive development work in over the past year, and which do you want to work in over the next year? (If you both worked with the language and want to continue to do so, please check both boxes in that row.)



Quelques idées fausses



Quelques idées fausses

- JavaScript est la version script de Java.
- Non ! JavaScript n'a (presque) rien à voir avec Java !





http://www.seguetech.com/blog/2013/02/15/java-vs-javascript

JavaScript	Java
• Créé par Netscape (dec. 1995)	• Créé par Sun Microsystems (mai 1995)

• Syntaxe dérivée du C/C++
`for (var i = 0; i < 10; i++) {`
`  console.log("Hello World");`
`}`
	• Syntaxe dérivée du C/C++
	`for (int i = 0; i < 10; i++) {`
	`  System.out.println("Hello World");`
	`}`

• Langage interprété :
  • Pas de compilation
  • Exécution directe dans un interpréteur
	• Langage compilé :
	  • Compilation vers du ByteCode
	  • Exécution nécessite machine virtuelle Java

• Typage dynamique
  • Plus de souplesse mais moins de contrôles
	• Type statique
	  • Contrôles à la compilation, moins souple mais plus robuste

• Modèle objet à base de Prototypes	• Modèle objet à base de Classes

• Un vrai langage fonctionnel
  • Les fonctions sont des objets de 1er ordre (1rst class objects)
	• Un langage impératif
	  • lambda expressions introduites dans Java 8 (2014)

Quelques idées fausses

- JavaScript est **le** langage du Web

JavaScript serait donc la seule technologie applicative côté client ?



> JavaScript is the programming language of HTML and the Web.

https://www.w3schools.com/js/default.asp

- JavaScript intégré en standard aux navigateurs Web



- D'autres technologies ont été développées pour s'exécuter côté client
  - Adobe Flash
  - applets Java



- Mais avec :
  - le développement d'AJAX (Asynchronous JavaScript and XML)
  - la normalisation du langage et un meilleur support par tous les navigateurs
  - l'introduction de HTML5 (et de ses nombreuses API JavaScript)

JavaScript est revenu au tout 1er plan



Mais plus tout seul

TypeScript

WebAssembly

(Type Script et WebAssembly

TypeScript (TS)

- Langage de programmation open source développé par Microsoft.
- **Objectif :** améliorer la productivité des développeurs JavaScript et la qualité du code.

- **Surensemble de JavaScript** qui ajoute un typage statique optionnel et des fonctionnalités avancées
  - **Typage statique** : Contrairement à JavaScript, TypeScript permet de définir les types des variables, des paramètres de fonction et des valeurs de retour,
    - aide à détecter les erreurs de type dès la phase de développement
  - **Classes et interfaces** : TypeScript prend en charge les concepts de programmation orientée objet comme les classes et les interfaces
    - facilite la création de structures de code robustes et maintenables
  - **Interopérabilité avec JavaScript** : Tout code JavaScript valide est également valide en TypeScript
    - permet une adoption progressive et une compatibilité avec les bibliothèques JavaScript existantes
  - **Transpilation :** Le code TypeScript est compilé en JavaScript,
    - permet son exécution dans n'importe quel environnement JavaScript, comme les navigateurs web ou Node.js
  - **Outils de développement :** TypeScript est bien intégré dans les environnements de développement modernes, offrant des fonctionnalités comme l'autocomplétion, la vérification des types en temps réel et la refactorisation de code

TypeScript (TS) : exemples simples*

- Adoptez TypeScript graduellement

1

No editor warnings in JavaScript files

This code crashes at runtime!

```
function compact(arr) {
    if (orr.length > 10)
        return arr.trim(0, 10)
    return arr
}
```

2

Adding this to a JS file shows errors in your editor

```
// @ts-check

function compact(arr) {
  if (orr.length > 10)
Cannot find name 'orr'.
      return arr.trim(0, 10)
  return arr
}
```

the param is arr, not orr!

3

Using JSDoc to give type information

```
// @ts-check

/** @param {any[]} arr */
function compact(arr) {
    if (arr.length > 10)
      return arr.trim(0, 10)
Property 'trim' does not exist on type 'any[]'.
    return arr
  }
```

Now TS has found a bad call. Arrays have slice, not trim.

4

TypeScript adds natural syntax for providing types

```
function compact(arr: string[]) {
  if (arr.length > 10)
    return arr.slice(0, 10)
  return arr
}
```

- Décrivez la forme des données (objets).

```
interface Account {
  id: number
  displayName: string
  version: 1
}

function welcome(user: Account) {
  console.log(user.id)
}
```

```
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

- Transpilation

```
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

Fichier TypeScript.

```
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

Les types sont retirés.

```
function verify(result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

Fichier JavaScript.

*Exemples tirés de https://www.typescriptlang.org/

WebAssembly

- JavaScript n'est plus tout seul dans le navigateur





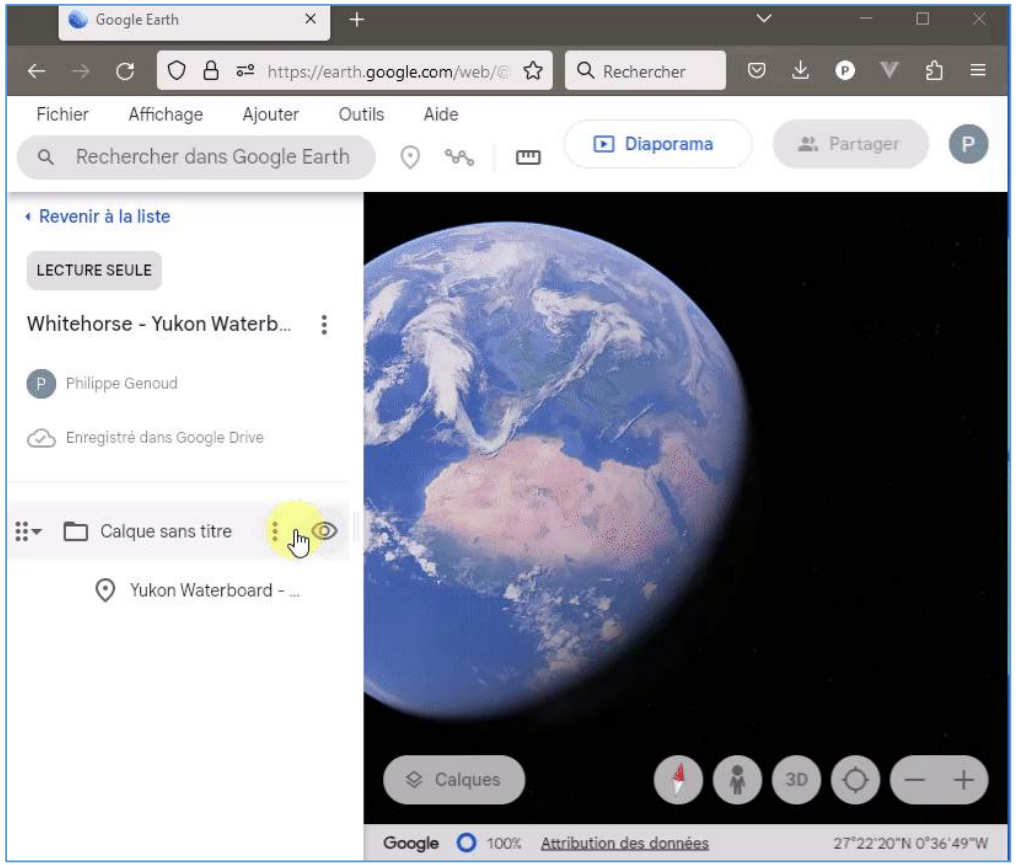
https://www.xenonstack.com/insights/guide-webassembly/

- WebAssembly
  - Nouveau type de code pouvant être exécuté dans les navigateurs modernes
  - Format binaire compact, rapide à charger et à exécuter
    → performances proche du code natif
  - Conçu pour être un cible de compilation efficace pour des langages source de bas niveau (C, C++, Rust, …)

WebAssembly

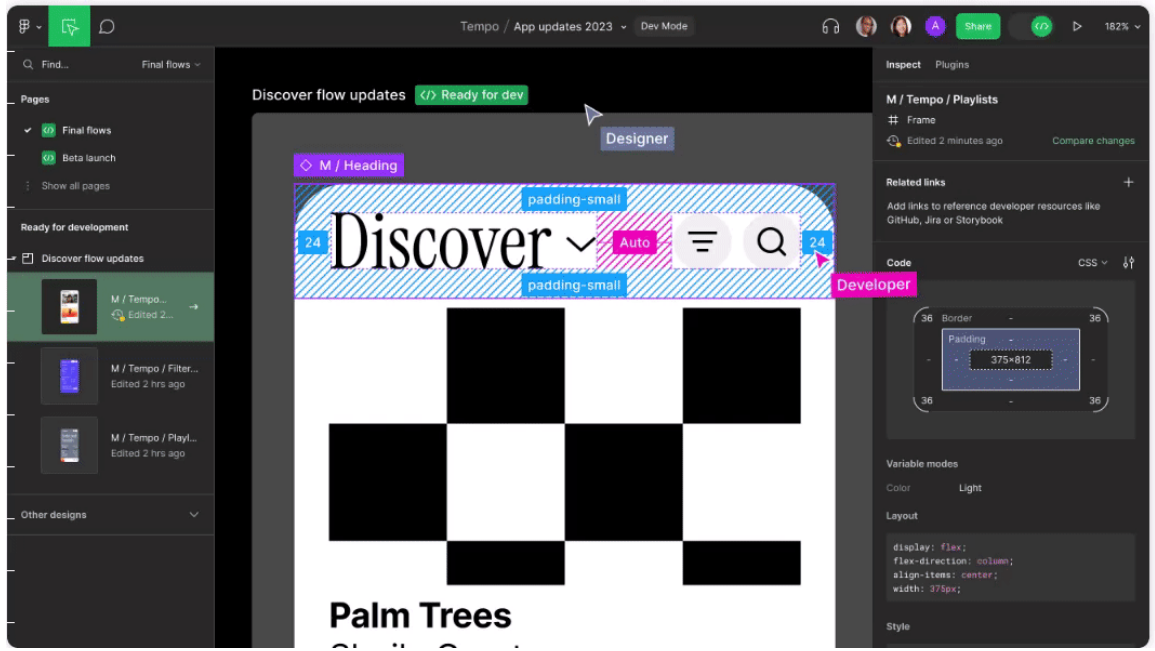
- exemples d'applications utilisant WebAssembly

https://earth.google.com/web/



Google Earth : cartographie 3D

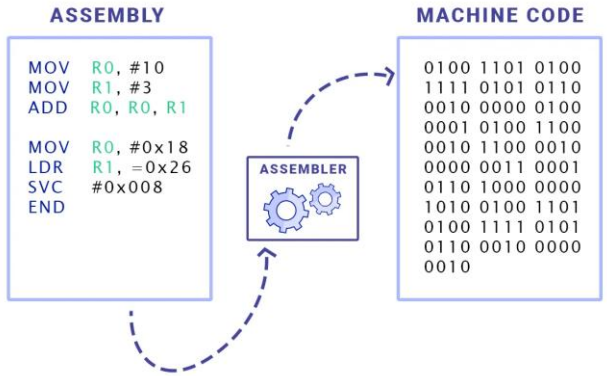
https://www.figma.com/



Figma : outil collaboratif de conception d'interfaces utilisateur (UI)

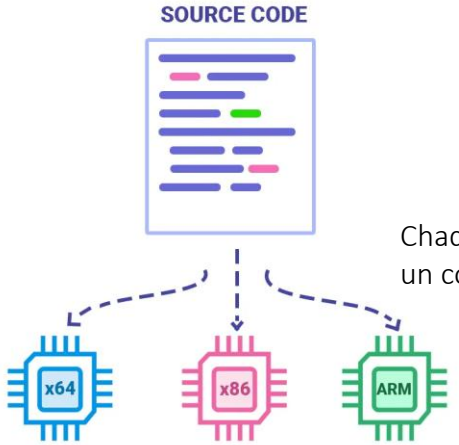
WebAssembly

Assembleur : représente le langage machine sous une forme textuelle lisible par un humain



*

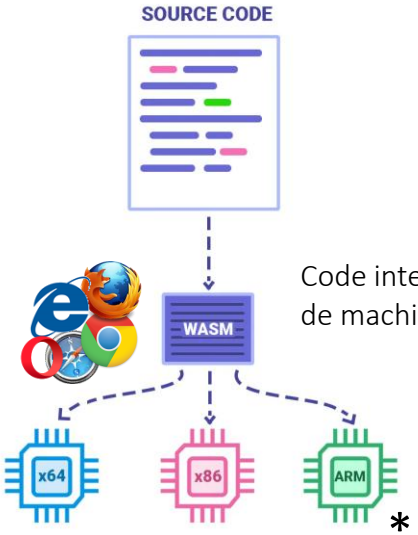
Pour être exécutés les programmes écrits dans un langage de haut niveau doivent être traduits en langage machine



Chaque type de processeur nécessite un code machine (assembleur) spécifique

*

WebAssembly : pas un véritable langage d'assemblage



Code intermédiaire ne ciblant pas de machine spécifique

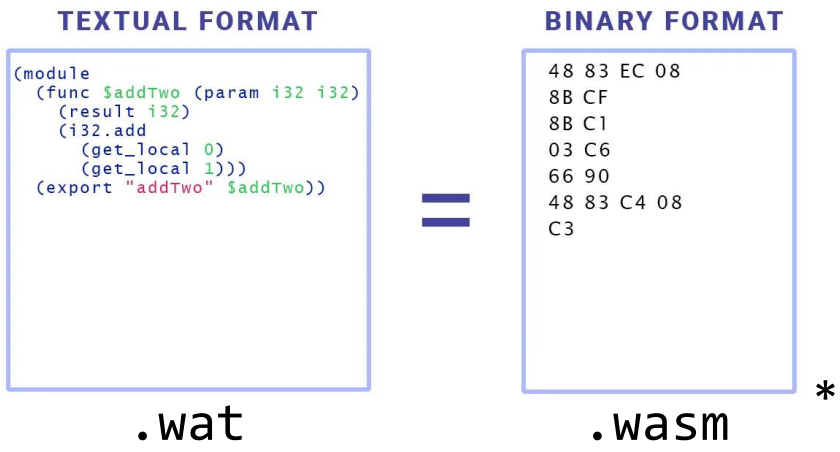
Quand le navigateur charge du code WebAssembly, il peut rapidement le traduire en code machine

*

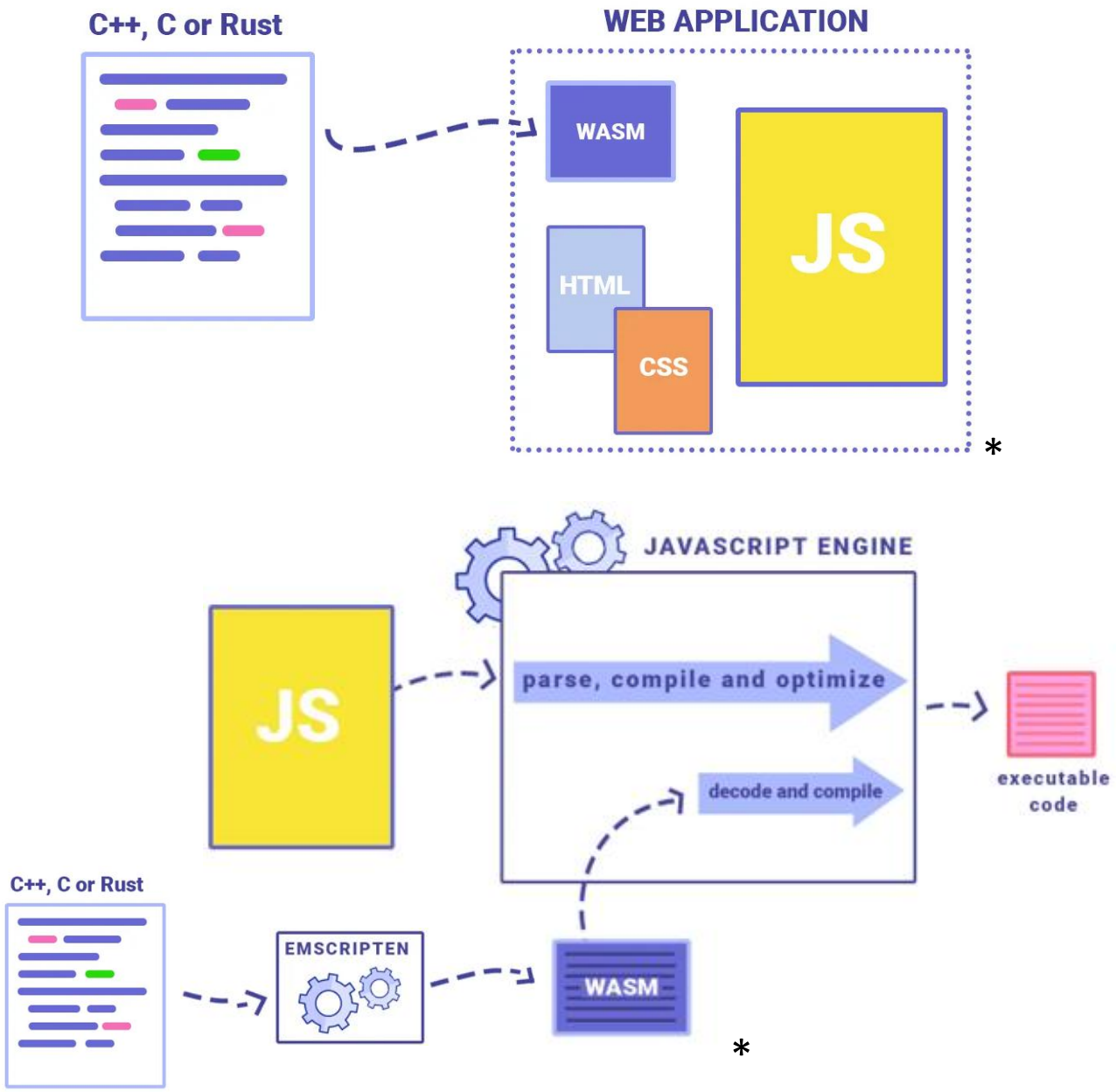
* WebAssembly: How and why - Milica Mihajlija – Août 2018
https://blog.logrocket.com/webassembly-how-and-why-559b7f96cd71/

WebAssembly

- Code C, C++, Rust peut être compilé vers du WebAssembly
(WASM : WebAssembly module)



- Celui-ci peut être chargé dans une application web et appelé depuis JavaScript (et inversement)



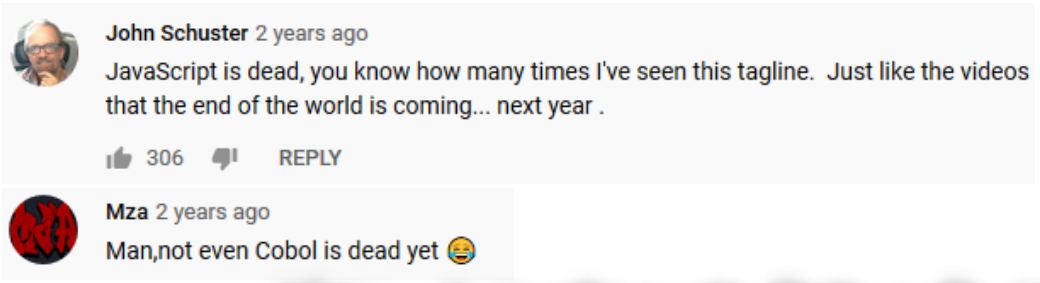
* WebAssembly: How and why - Milica Mihajlija – Août 2018
https://blog.logrocket.com/webassembly-how-and-why-559b7f96cd71/

WebAssembly

- Pour résumer WebAssembly apporte
  - Vitesse
  - Portabilité
  - Flexibilité
    - Développeurs web pourront choisir d'autre langages que JavaScript



https://www.youtube.com/watch?v=3LWgbjVWLug
WebAssembly and the Death of JavaScript ?
Colin Eberhardt (Scott Logic)



> web developers will be able to choose other languages and more developers will be able to write code for the web. JavaScript will still be the best choice for most use cases but now there will be an option to drop down to a specialized language once in a while when you really need a boost. Parts like UI and app logic could be in JavaScript, with the core functionality in WebAssembly. When optimizing performance in existing JS apps, bottlenecks could be rewritten in a language that is better suited for the problem.

WebAssembly: How and why - Milica Mihajlija – Août 2018
https://blog.logrocket.com/webassembly-how-and-why-559b7f96cd71/

TypeSCript et WebAssembly

- Pour en savoir plus

  - TypeScript :
    - https://www.typescriptlang.org/fr/

  - WebAssembly
    - A Beginner’s guide to WebAssembly – Mars 2020
      https://www.xenonstack.com/insights/guide-webassembly/
    - WebAssembly: How and why - Milica Mihajlija – Août 2018
      https://blog.logrocket.com/webassembly-how-and-why-559b7f96cd71/
    - Transforming the Tech Landscape: The Top WebAssembly Use Cases for 2023
      https://hybrowlabs.com/blog/the-top-use-cases-for-webassembly-in-2023
    - WebAssembly
      https://developer.mozilla.org/en-US/docs/WebAssembly
    - Le site officiel
      https://webassembly.org/
    - https://www.ionos.fr/digitalguide/sites-internet/developpement-web/webassembly-quest-ce-que-cest/

TypeScript et WebAssembly)

Quelques idées fausses

- JavaScript est pour les clients Web
- oui… mais pas que.



> JavaScript is the programming language of HTML and the Web.

https://www.w3schools.com/js/default.asp



- Historiquement JavaScript intégré aux navigateurs



- JSON s'impose de plus en plus comme un format d'échange de données



- Depuis 2009/2010 de plus en plus présent côté serveur

Quelques idées fausses

- JavaScript est un langage simple et facile à apprendre
- Cela se discute… JavaScript n'est pas un langage sans défauts et il peut être parfois délicat à utiliser



JavaScript is easy to learn.

https://www.w3schools.com/js/default.asp

Because JavaScript has a forgiving, flexible syntax and works across all major browsers, it is one of the friendliest programming languages for beginners.

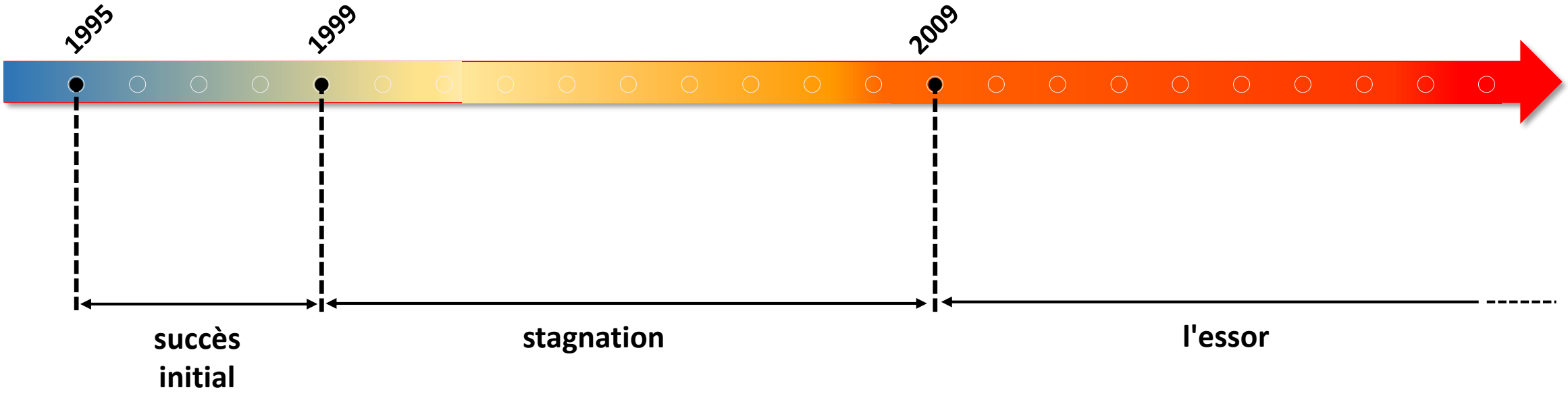
https://www.fullstackacademy.com/blog/nine-best-programming-languages-to-learn

> *"La plupart des langages contiennent des bons et des mauvais éléments. … Le JavaScript est un langage particulièrement bien loti en ce qui concerne les mauvais éléments… Mais le JavaScript contient heureusement un certain nombre d'éléments exceptionnellement bons…"*

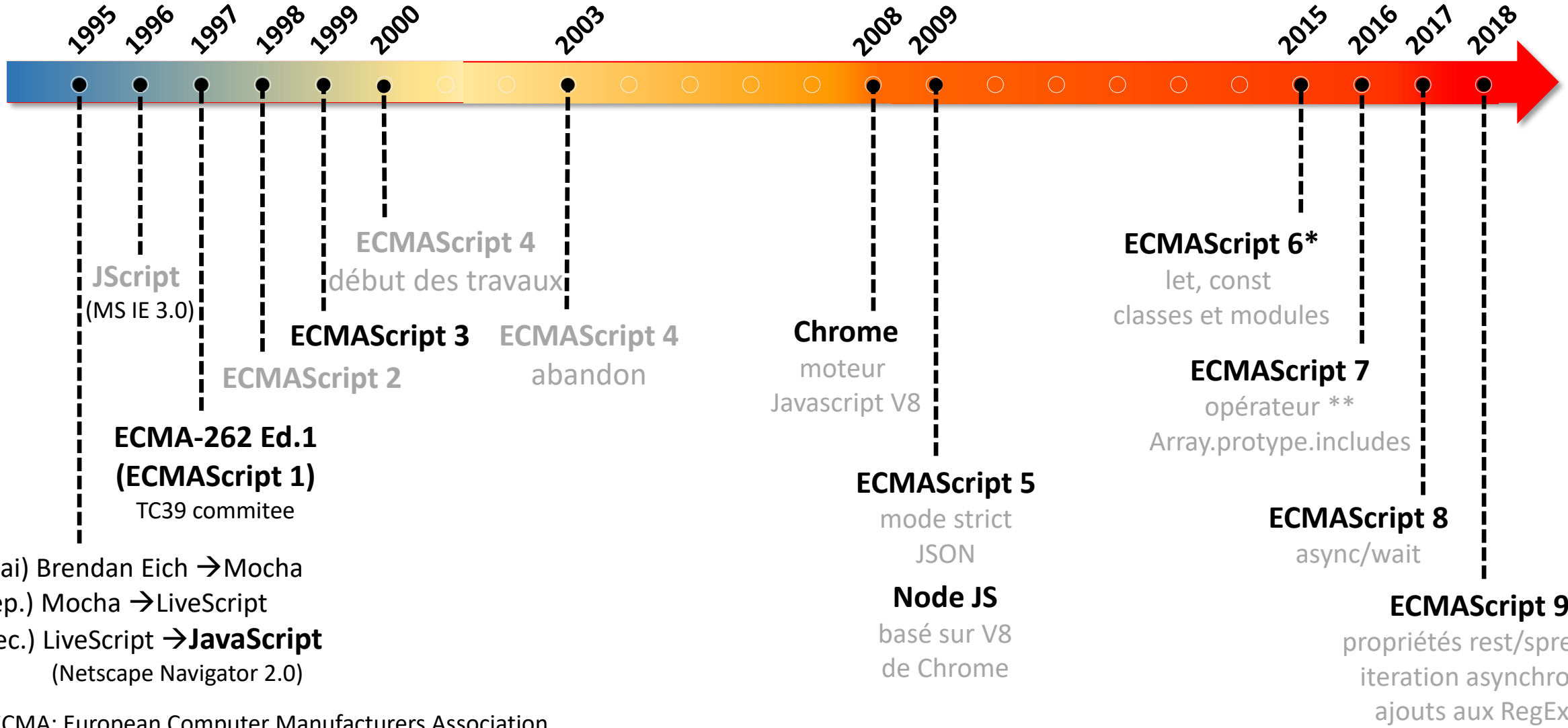
Douglas Crockford
JavaScript, les bons éléments
Ed. Pearson France, 2013



Historique



Historique

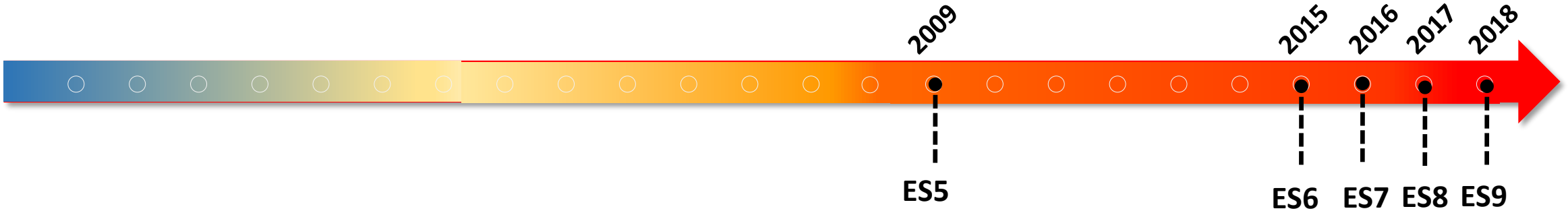


(Mai) Brendan Eich →Mocha
(Sep.) Mocha →LiveScript
(Dec.) LiveScript →**JavaScript**
(Netscape Navigator 2.0)

ECMA: European Computer Manufacturers Association
ECMA-262: nom officiel du standard
ECMAScript: nom du langage

*à partir de 2015 les versions de ECMAScript sont numérotées par leur année de publication ECMAScript 2015, ECMAscript 2016...

Support par les différents navigateurs



	ECMAScript 5		ECMAScript 6		ECMAScript7	
Navigateur	Version	Date	Version	Date	Version	Date
Chrome	23	Sept. 2012	68	Avr. 2017	68	Mai 2018
Firefox	21	Avr. 2013	54	Juin 2017		
IE	9*	Mars 2011	-	-	-	-
Edge	10	Sept. 2012	14	Août 2016		
Safari	6	Juil. 2012	10	Sept. 2016		
Opera	15	Juil. 2013	55	Août 2017	47	Juil. 2018

*IE 9 ne supporte pas mode strict

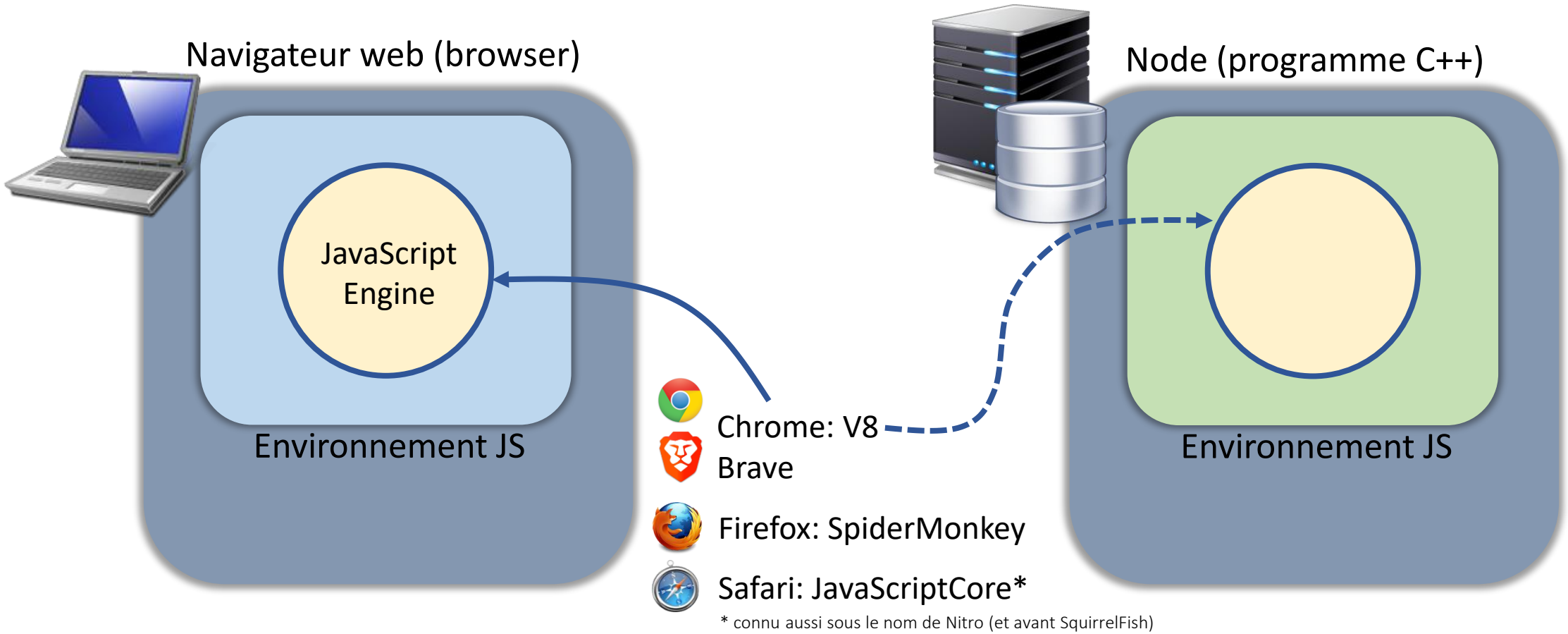
d'après https://www.w3schools.com/js/js_versions.asp

https://caniuse.com/

http://kangax.github.io/compat-table/es6/

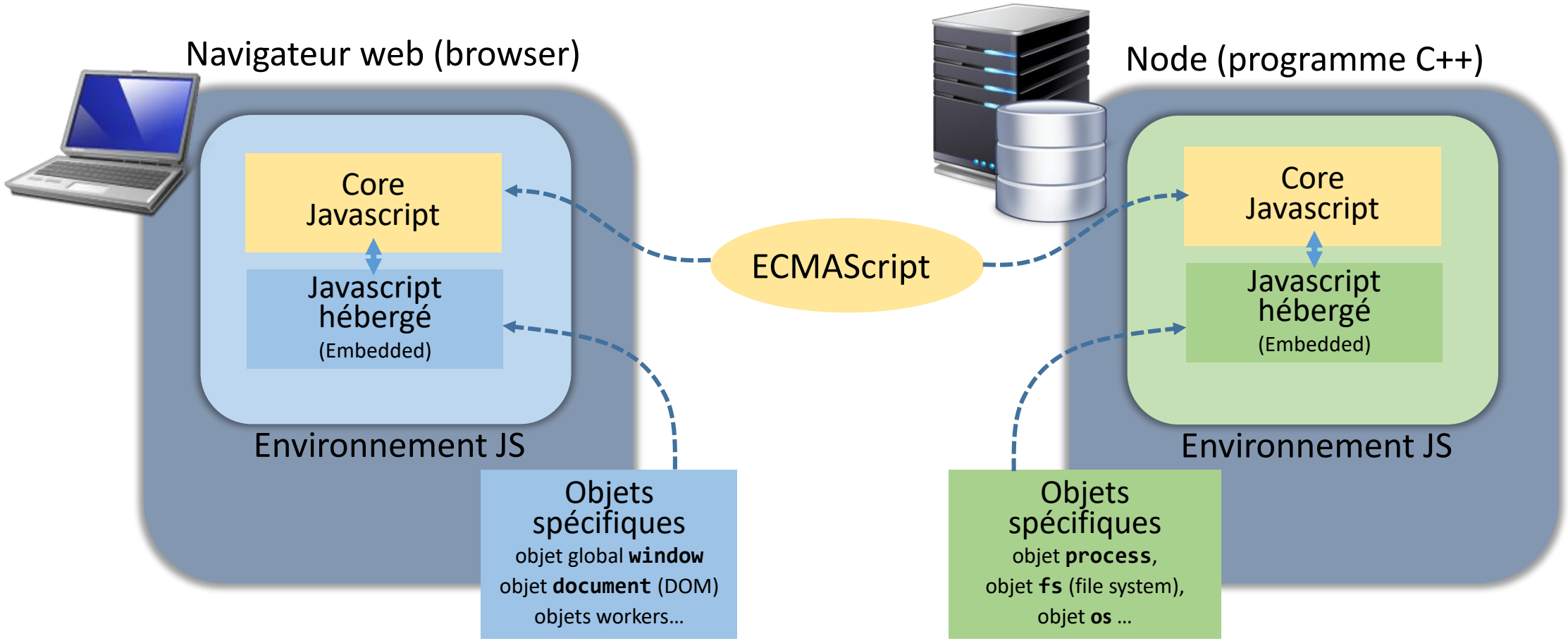
Environnements d'exécution JavaScript

- analyse et exécution du code JavaScript sont effectué par un moteur Javascript (JavaScript Engine) qui s'exécute dans un environnement hôte



Environnements d'exécution JavaScript

- dans l'environnement JavaScript on distingue
  - core javascript : correspond aux spécifications de la norme ECMAScript
  - javascript hébergé: intègre des objets spécifiques à l'environnement



JavaScript dans un navigateur

- Code JavaScript peut être inclus dans des pages HTML

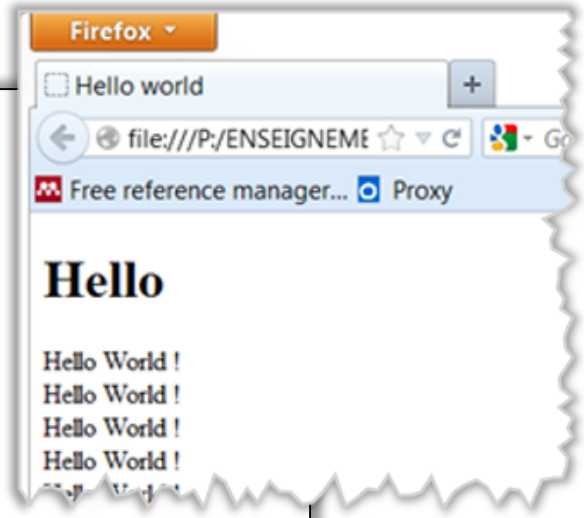


Et alors ? En quoi JavaScript est-il si différent d'autres langages comme PHP, JSP, ASP ?



[HelloWorld.html](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello world</title>
  </head>
  <body>
    <h1>Hello</h1>
    <p>
      <script>
         for (i=0; i < 5; i++) {
            document.write("Hello World !</br>");
         }
      </script>
    </p>
  </body>
</html>
```

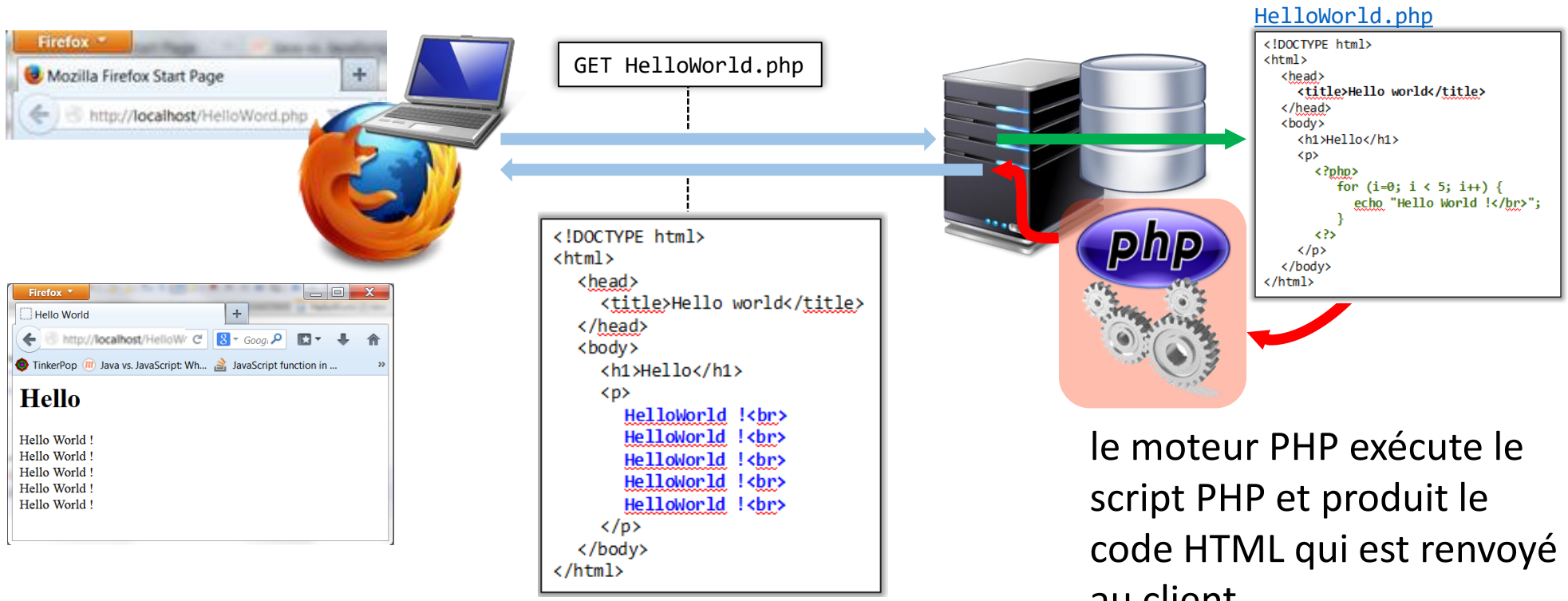


[HelloWorld.php](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello world</title>
  </head>
  <body>
    <h1>Hello</h1>
    <p>
      <?php
         for ($i=0; $i < 5; $i++) {
           echo "Hello World !</br>";
         }
      ?>
    </p>
  </body>
</html>
```

JavaScript dans le navigateur

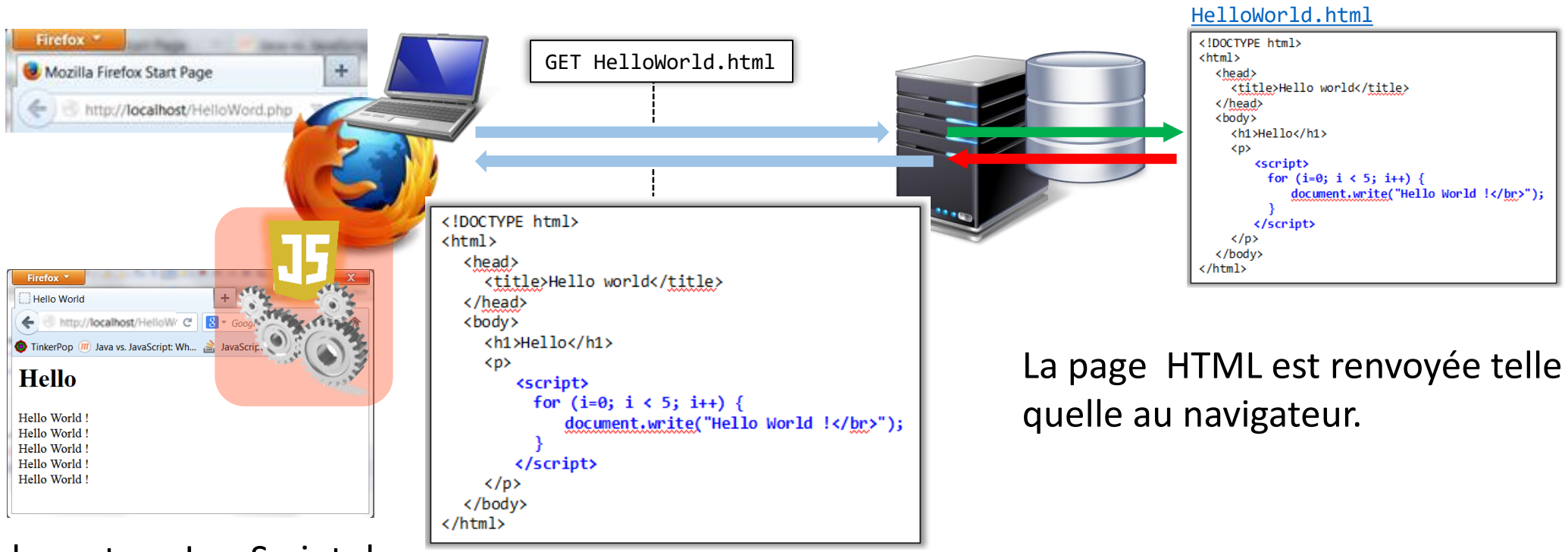
- un script PHP s'exécute côté serveur



le moteur PHP exécute le script PHP et produit le code HTML qui est renvoyé au client

JavaScript dans le navigateur

- un script JavaScript s'exécute côté client (dans le navigateur)



La page HTML est renvoyée telle quelle au navigateur.

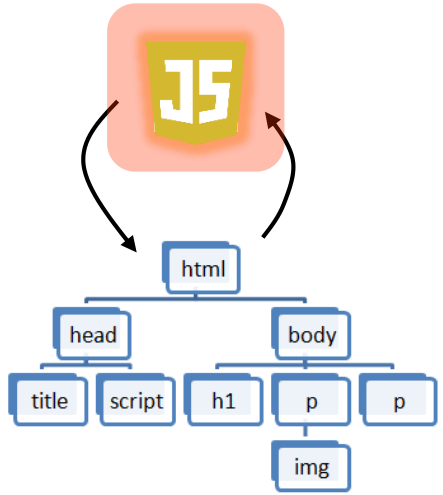
C'est le moteur JavaScript du navigateur qui interprète le script et modifie la page HTML

JavaScript dans le navigateur

- La manière dont JavaScript fonctionne*

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello world</title>
  </head>
  <body>
    <h1>Hello</h1>
    <p>
        <script>
          for (i=0; i < 5; i++) {
             document.write("Hello World !</br>");
          }
        </script>
    </p>
  </body>
</html>
```





1 Ecriture

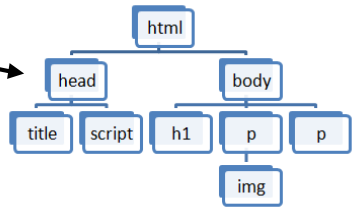
Vous créez vos pages HTML et votre code JavaScript que vous mettez dans un ou plusieurs fichiers

2 Chargement

Le navigateur récupère la page et charge la page en analysant (parsing) son contenu de haut en bas.

Lorsque le navigateur rencontre du code JavaScript il l'analyse, vérifie sa correction puis l'exécute

Le navigateur construit un modèle interne de la page HTML : le DOM



3 Exécution

Le navigateur affiche la page.
JavaScript continue à s'exécuter, en utilisant le DOM pour examiner la page, la modifier, recevoir des événements depuis celle-ci, ou pour demander au navigateur d'aller chercher d'autres données sur le serveur web

*d'après "Head First HTML 5 programming"
Eric Freeman, Elisabeth Robson
Ed. O'Reilly, 2011

Intégration de code JavaScript à une page HTML

- la balise **<script>** délimite code JavaScript
- peut être insérée soit dans l'entête (**<head>**) soit dans le corps (**<body>**) de la page HTML
- les scripts sont exécutés dans leur ordre d'apparition dans la page

Placer les scripts dans l'en tête pour qu'ils soient chargés et exécutés avant la construction et l'affichage de la page

Par défaut, les scripts placés dans le body sont exécutés au fur et à mesure que celui-ci est chargé et que le DOM est construit.
→ met en attente le moteur d'analyse HTML/CSS

Souvent les scripts sont placés à la fin du corps de la page (juste avant </body>) → améliorer la vitesse de chargement de la page.

Images et css en provenance de différents sites peuvent être téléchargées simultanément (en parallèle) mais une fois que le navigateur a rencontré une balise script ce n'est (par défaut) plus possible → les scripts bloquent les téléchargements parallèles

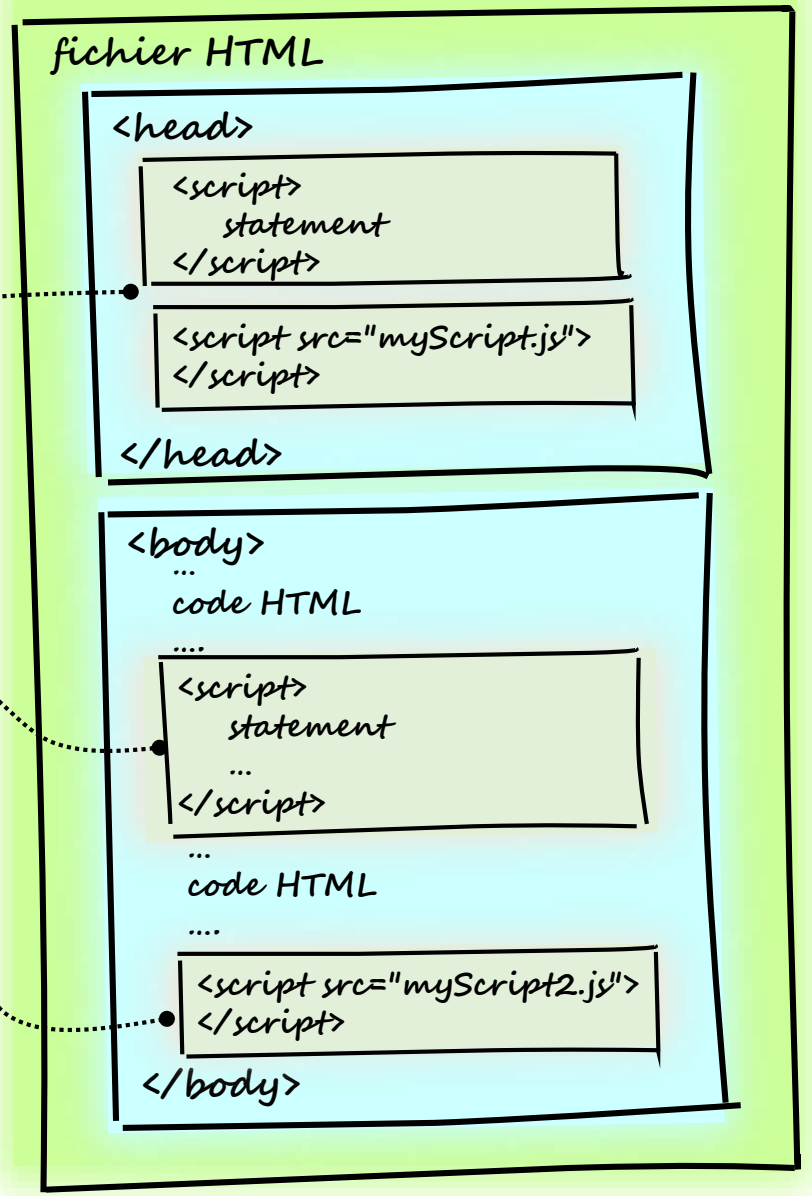


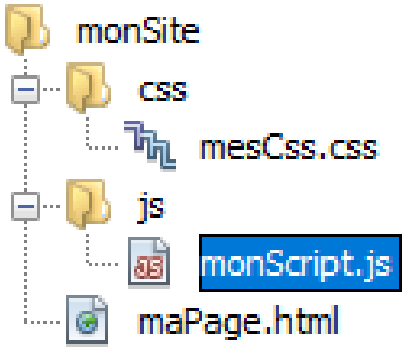
figure d'après "Head First HTML 5 programming"
Eric Freeman, Elisabeth Robson- Ed. O'Reilly, 2011

Intégration de code JavaScript à une page HTML

- deux possibilités pour un élément **<script>**
  - soit il contient des instructions JavaScript 1
  - soit il pointe vers un fichier de script externe au travers de son attribut **src** 2
    - url fichier externe, permet de localiser un fichier texte qui contient le code JavaScript.
      - url vers un fichier du site : chemin relatif **./chemin**
      - url vers un fichier d'un site externe: **https:// ….**
    - fonctionnement équivalent à une intégration directe du code JavaScript à cet endroit.



quand l'attribut **src** est utilisé il ne faut rien écrire entre les balises **<script src="…">** et **</script>**



maPage.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello Good bye</title>
  </head>
  <body>
    <h1>Hello</h1>
    <p>
      <script>                                          1
         for (i = 0; i < 5; i++) {
            document.write("Hello World !</br>");
         }
      </script>
    </p>
    <h1>Bye Bye</h1>
    <p>
      <script src="./js/monScript.js"></script>        2
    </p>
  </body>
</html>
```

chemin relatif

monScript.js

```
for (j = 0; j < 5; j++) {
  document.write("Bye Bye World !</br>");
}
```

Intégration de code JavaScript à une page HTML

- Autres attributs de la balise **`<script>`**

- **`type`**
  - ex : **`<script type="text/javascript"> ... </script>`**
  - utilisé par certains "vieux codes".
  - inutile avec les navigateurs modernes et HTML5 : JavaScript est le langage par défaut.
- **`async`**
  - utilisable que pour des scripts externes (attribut **`src`** présent)
  - indique si le script peut être évalué de manière asynchrone (chargé en parallèle à l'analyse de la page (parsing) sans interrompre celle-ci, et évalué dès que son chargement est terminé). Ordre des scripts n'est pas préservé.
  - `false` par défaut

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



Le deuxième script risque d'être évalué avant que le premier ait fini d'être chargé → erreur

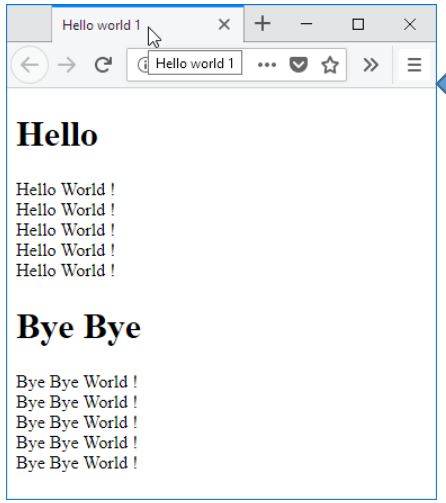
- **`defer`**
  - utilisable que pour des scripts externes (attribut **`src`** présent)
  - indique si le script doit être évalué de manière différée (une fois que la page aura été entièrement analysée (parsée), avant événement `DOMContentLoaded`), l'ordre des scripts est préservé
  - `false` par défaut

https://www.alsacreations.com/astuce/lire/1562-script-attribut-async-defer.html

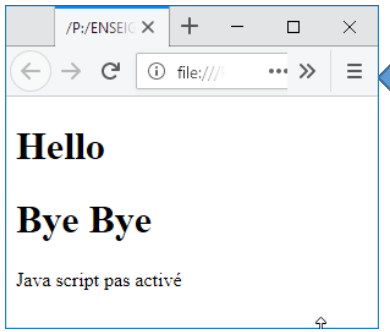
Intégration de code JavaScript à une page HTML

- possibilité de désactiver JavaScript dans le navigateur

HelloByeByeNoJS.html

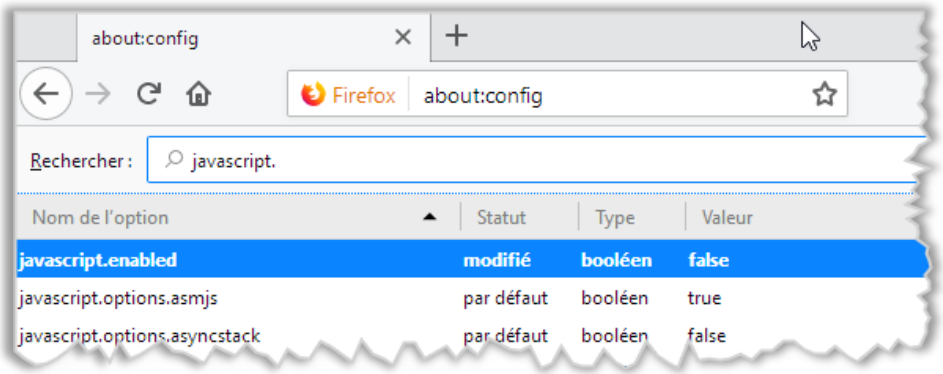


JavaScript activé (par défaut dans les navigateurs)



JavaScript désactivé

désactiver JavaScript dans Firefox



commentaires HTML : masquent code pour navigateurs ne supportant pas JavaScript (on peut rencontrer cela dans du très très vieux code)

balise <noscript> définit alternative si JavaScript désactivé

```
<!DOCTYPE html>
<html>
   <head>
      <script>
        document.write("<title>Hello world 1</title>");
      </script>
   </head>
   <body>
      <h1>Hello</h1>
      <p>
        <script>
           for (i=0; i < 5; i++) {
               document.write("Hello World !</br>");
           }
        </script>
      </p>
      <h1>Bye Bye</h1>
      <p>
        <script>
        <!--
           for (i=0; i < 5; i++) {
             document.write("Bye Bye World !</br>");
           }
        // -->
        </script>
        <noscript>
           Java script pas activé
        </noscript>
      </p>
   </body>
</html>
```

Intégration de code JavaScript à une page HTML

Affichage

- Core JavaScript
  - `console.log(message)`
    - affiche un message dans la console
- JavasScript dans un navigateur web
  - `document.write(markupText)`
    - écrit un flux de texte dans le document ouvert
  - `window.alert(message)`
    - affiche un dialogue d'alerte contenant le texte spécifié.
  - `result = window.prompt(message, default)`
    - affiche un message et permet la saisie d'une valeur (chaîne de caractères)
  - `result = window.confirm(message)`
    - affiche une alerte avec un message et deux bouton OK (true) ou Cancel (false)
  - en modifiant le contenu d'un élément HTML
    - `document.getElementById("id1").innerHTML= "Nouveau contenu";`
    - `document.querySelector("#id1").innerHTML= "Nouveau contenu";` ES6 ECMACScrip2015

IoJavaScript.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title> une page simple </title>
5      </head>
6      <body>
7          <h1> Bonjour <span id="nom"></span></h1>
8          <script>
9              alert('bonjour');
10             let nom = prompt('quel est votre nom ?', 'Indiquer votre nom ici');
11             let rep = confirm('Afficher le nom dans une alerte ?');
12             if (rep) {
13                 alert('le nom saisi est ' + nom);
14             }
15             console.log('le nom saisi est ' + nom);
16             document.write('<h2>Bonsoir ' + nom + '<h2>');
17             document.getElementById("nom").innerHTML = nom;
18         </script>
19     </body>
20 </html>
```

Variables

- utilisées pour stocker temporairement des données dans la mémoire de l'ordinateur

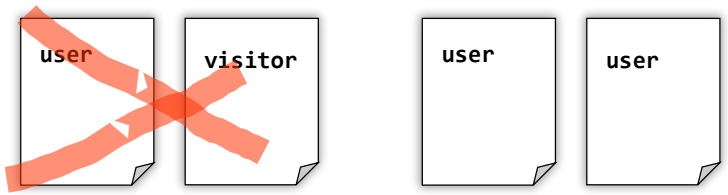


En JavaScript deux types de valeurs

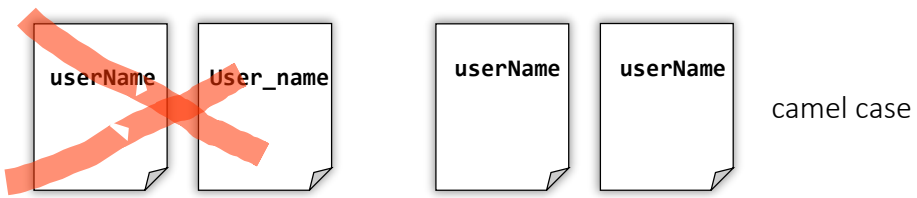
- valeurs primitives
  - notation littérales
    - de nombres (`14`, `3.14159`, `3.25e-6`)
    - de chaînes de caractères (`'Ceci est une chaîne'`, `"cela c'est aussi une chaîne"`)
  - valeurs natives (built-in) (`true`, `false`, `undefined`, `null`, `NaN`, `Infinity`)
- objets
  - c'est-à-dire des conteneurs (par exemple un tableau), la valeur de la variable dans ce cas est l'adresse (référence) du conteneur.

Variables : règles de nommage

- une variable doit être désignée avec un nom unique (dans sa portée) : identifiant
- règles similaires à celles de la plupart des langages
  - noms ne peuvent contenir d'espace, d'opérateurs arithmétiques (`+` , `-`, `/`, `*`) ou de caractère de ponctuation (`. , ;`)
  - noms peuvent contenir des lettres, chiffres, `_` ou `$`
  - noms ne peuvent commencer par un chiffre
  - noms sensibles à la casse (case sensitive) `maVariable` ≠ `mavariable`
  - noms doivent être différents des mots réservés (par ex. `var`, `if`, `while`, `function`…).
- bonnes pratiques
  - choisir des noms descriptifs
    - par exemple pour un variable correspondant à une moyenne on préfèrera **`moyenne`** à `m`
  - ne pas hésiter à utiliser plusieurs mots pour décrire avec précision le rôle d'un variable
    - par exemple pour une pour un variable correspondant à un prix unitaire d'un produit on préfèrera **`prixUnitaire`** à **`pu`**
  - adopter des règles de nommage cohérentes dans tout le code
    - Définir un glossaire, réutiliser les mêmes termes partout



- Conserver les mêmes conventions d'écriture



camel case

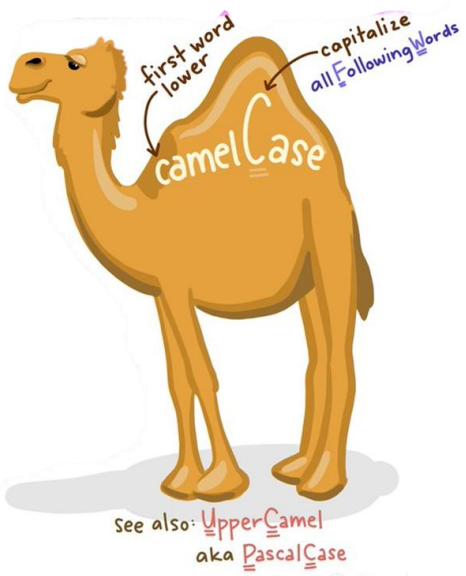
Variables : règles de nommage

- Comment regrouper plusieurs mots dans un identifiant
  - Ex : variable représentant l'âge du capitaine.

~~ac~~ ~~agec~~ ~~agecap~~ ~~ageducapitaine~~

- Conventions d'écriture couramment utilisées

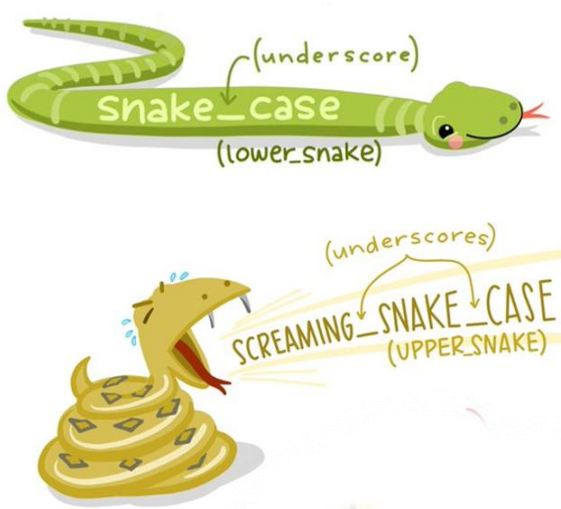
Camel case	Pascal case	Snake case	Kebab case
`ageDuCapitaine`	`AgeDuCapitaine`	`age_du_capitaine` `AGE_DU_CAPITAINE`	`age-du-capitaine`



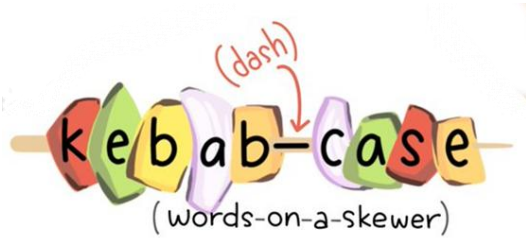
Identifiants de variables ou fonctions



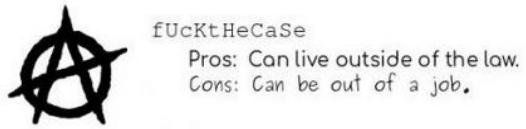
Identifiants de Classes



Identifiants de Constantes



Identifiants de classes de styles CSS
URL



Images d'après @allison_horst

Variables: déclaration

- historiquement déclaration par `var nomDeVariable ;`
- mais depuis ES6 la bonne pratique et d'utiliser `let` au lieu de `var`



```
var nomDeVariable;
```



```
let nomDeVariable;
```

on peut même utiliser une variable sans la déclarer !



Patient tu dois être. Les différences (subtiles parfois) plus tard seront expliquées



Variables: déclaration

- JavaScript langage avec **typage dynamique**
  - pas de définition de type à la déclaration d'une variable
  - type d'une variable défini qu'au moment de l'exécution lorsqu'une valeur est affectée à la variable (opérateur =)
  - possibilité de changer le type d'une variable à l'exécution
  - une variable non initialisée a la valeur `undefined`

- possibilité d'initialiser une variable à sa déclaration en utilisant l'opérateur = (affectation)
  - à l'aide d'expressions littérales
  - à l'aide d'expressions complexes

- possibilité d'effectuer plusieurs déclarations simultanément
  - séparées par ,
  - à éviter pour une plus grande lisibilité

```
// Déclaration d'une variable sans initialisation
let a;
console.log('a = ' + a);    → a = undefined
a = "une chaîne de caractères";
console.log('a = ' + a);    → a = une chaîne de caractères
a = 1234;
console.log('a =' , a);     → a = 1234


// Déclaration de variables avec initialisation
let prixHT = 150;
let tauxTVA= 19.6;
let prixTTC = prixHT + (tauxTVA / 100 * prixHT);
console.log(`prix TTC : ${prixTTC}`);  → prix TTC : 179.4


// Déclaration simultanée de plusieurs varaibles
let xCentre = 10.5, yCentre = 14, rayon = 10;
let surface =  Math.PI * rayon * rayon;
console.log(`surface du cercle de centre\
 (${xCentre},${yCentre})\
 et rayon ${rayon} : ${surface.toFixed(2)}`);
```

→ surface du cercle de centre (10.5,14) et rayon 10 : 314.16

Constantes

- parfois il peut être intéressant d'empêcher la modification d'une variable

```
let tauxTVA = 0.20;  // TVA à 20 %
let tauxReduc = 0;
let nbProduitsAchetes = 0;

...
if (nbProduitsAchete >= 5) {
   // appliquer un taux de réduction de 25 %
   tauxTVA = 0.25;
}
...
```



va provoquer des résultats d'exécution erronés,
bug peut être difficile à corriger lorsque la taille du code est importante

Constante

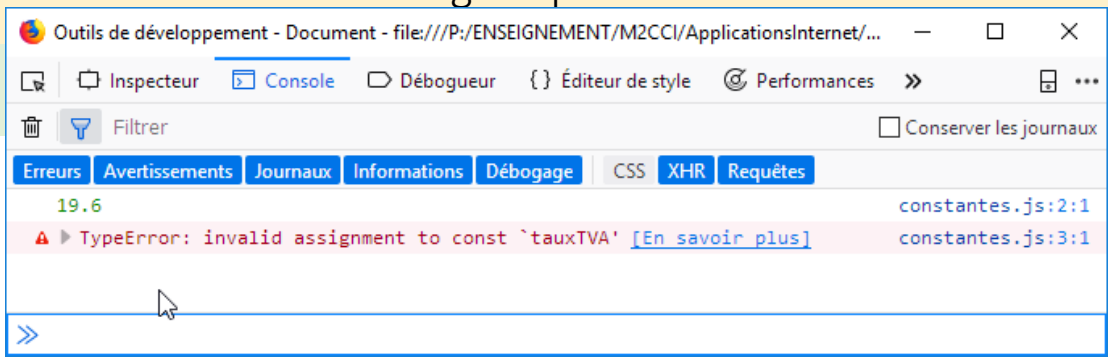
- possibilité de déclarer des constantes en JavaScript
- utiliser `const` au lieu de `let`

```
const tauxTVA = 0.2;
let tauxReduc = 0;
let nbProduitsAchetes = 0;


...
if (nbProduitsAchete >= 5) {
   // appliquer un taux de réduction de 25 %
   tauxTVA = 0.25;
}
...
```

obligation d'initialiser la variable à sa déclaration

une tentative de modification de la constante va provoquer une erreur d'exécution avec un message explicite sur la console



utiliser des majuscules pour les noms de constantes

```
const TAUX_TVA = 0.2;
```

Type d'une variable

- JavaScript est un langage dynamique : le type d'une variable peut changer au cours de l'exécution d'un programme.
- opérateur `typeof` permet de déterminer le type d'une variable.

```
let a;
console.log(typeof a);-------------------------> undefined

a = "une chaîne de caractères";
console.log(typeof a);--------------------------> string

a = 1234;
console.log(typeof a);--------------------------> number

a = 12.34;
console.log(typeof a); --------------------------> number

a = false;
console.log(typeof a);--------------------------> boolean

a = [ 12, 13, 14, 17];
console.log(typeof a);---------------------------> object

a = {
    nom : "Joe",
    prenom : "Doe",
    age : 34
};
console.log(typeof a);---------------------------> object

a = null;
console.log(typeof a);---------------------------> object
```

Types de base

- **types primitifs** : définissent des valeurs primitives (nombres, chaînes de caractères, booléens) qui sont stockées dans une variable
  - `number` : permet de représenter les nombres (nombre flottants double précision sur 64 bits, format IEEE 754)
    - Nombres décimaux compris entre 2^{-1074} (`Number.MIN_VALUE`) et 2^{1074} (Number.MAX_VALUE) pour les nombres positifs et -2^{1074} et -2^{-1074} pour les nombres négatifs
    - Nombres entiers entre $-(2^{53} - 1)$ (`Number.MIN_SAFE_VALUE`) et $2^{53} - 1$ (`Number.MAX_SAFE_VALUE`)
    - Valeurs particulières vers lesquelles sont automatiquement converties les valeurs en dehors de 2^{-1074} et 2^{1074} sont automatiquement converties vers des valeurs particulières
      - `+Infinity` : valeurs positives > `Number.MAX_VALUE`
      - `-Infinity` : valeurs négatives < `-Number.MAX_VALUE`
      - `+0` : valeurs positives < `Number.MIN_VALUE`
      - `-0` : Valeurs négatives > `-Number.MIN_VALUE`
    - `NaN` (Not a Number)
      - valeur rencontrée quand le résultat d'une opération arithmétique ne peut pas être représentée par un nombre
      - C'est la seule valeur en JavaScript qui n'est pas égale à elle même

```
» 34 / 0
← Infinity
» 34 / -0
← -Infinity
» +0 == -0
← true
```

```
» let x = parseInt("azerty")
← undefined
» x
← NaN
```

```
» let y = parseInt("azerty")
← undefined
» y
← NaN
» x == y
← false
```

Types de base

- **types primitifs** (suite)
  - `string`
    - Représente des données textuelles (chaînes de caractères) encodées comme une séquence d'entiers non signés sur 16 bits (2 octets) représentant une unité de code UTF-16
    - Chaque lettre de la chaîne occupe une position définie par une position (*index*) : 0 pour la première lettre, 1 pour la deuxième, …
    - La longueur de chaîne (*length*) est le nombre d'unités de codes UTF-16 (mots de 2 octets).
      - Ce n'est pas nécessairement le nombre de caractères Unicode (certains caractères exotiques peuvent être codés sur plus de 2 octets).
    - Expressions littérales pour les chaînes délimitées par des guillemets (*quotes*) : `"` ou `'` ou `` ` `` (*backticks*)



Pas de type **`char`** !

```
let nom = "DUPONT";
let prenom = 'Marie Louise';
```

```
let s1 = "Voilà l'automne";
let s2 = 'Il a dit : "Ah bon !"';
```

```
let nbHeures = 3;
let s3 = `durée en secondes : ${nbHeures * 60} s`;
```

→ `"durée en secondes : 180 s"`

`${ … }` permet d'inclure des variables et expressions dans une chaîne
La valeur de l'expression est évaluée et son résultat devient partie de la chaîne

Types de base

- types primitifs (suite et fin)
  - `boolean`
    - deux valeurs littérales : `true` (vrai) et `false` (faux).
  - `undefined`
    - possède une unique valeur, `undefined` affectée à une variable déclarée dont le contenu n'a jamais été initialisé.

```
let age;
```

age | undefined

  - `null`
    - La valeur spéciale `null`, n'appartient à aucun des types précédents, mais au type `object`

```
let jedi = null;
```

jedi | null

jedi est un variable particulière (une **référence**) qui permet de désigner un **objet** (regroupement de plusieurs variables en une structure complexe).

Pour le moment elle ne désigne aucun objet particulier (mais comme son nom le laisse à penser, elle devrait être utilisée ultérieurement pour désigner un objet représentant un guerrier Jedi)

Types de base

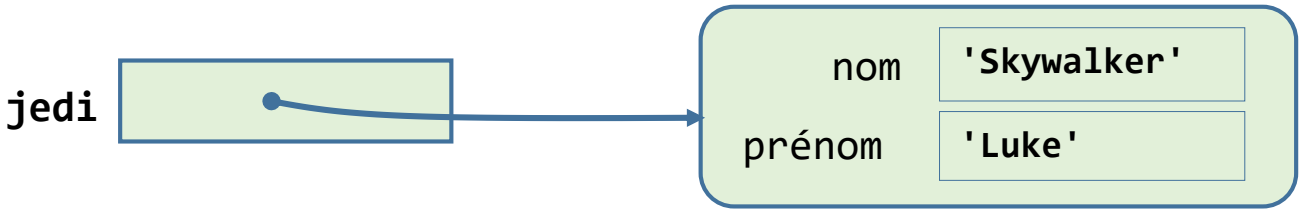
- Type objet
  - `object`
    - variable ne contient plus une simple valeur mais un **référence** (adresse) vers un objet complexe (valeur non primitive qui est un regroupement de variables)
    - un objet est défini par un ensemble de **propriétés** (ou **attributs**) qui peuvent être soit des valeurs primitives, soit elles mêmes des références d'objets
    - en plus de propriétés, des fonctions spécifiques (**méthodes**) peuvent être définies pour un objet
    - mais de tout cela on reparlera plus tard…

```
let nom = 'Skywalker';
```



variable de type primitif
la variable stocke la valeur

```
let jedi = {
    nom : 'Skywalker',
    prenom: 'Luke';
}
```



variable de type objet
la variable stocke la référence (adresse) de l'objet

Accès aux membres d'un objet

- Les membres d'un objet sont
  - les propriétés (attributs de l'objet)
  - le méthodes (fonctions) associées à l'objet
- L'accès à un membre d'un objet se fait avec notation pointée
  - `identifiantObjet.nomPropriété` pour accéder à un attribut
  - `identifiantObjet.nomMéthode(…)` pour exécuter une méthode (fonction) de l'objet

```
let unePersonne = {
    nom : 'DUPONT',
    prenom: 'Jean',
    age: 35
}

console.log(unePersonne.nom + " " + unePersonne.prenom);
unePersonne.age = 36;
console.log("age : " + unePersonne.age);
```

accès à un attribut d'un objet dans une expression

affection d'une valeur à un attribut d'un objet



DUPONT Jean
age : 36

Accès aux membres d'un objet

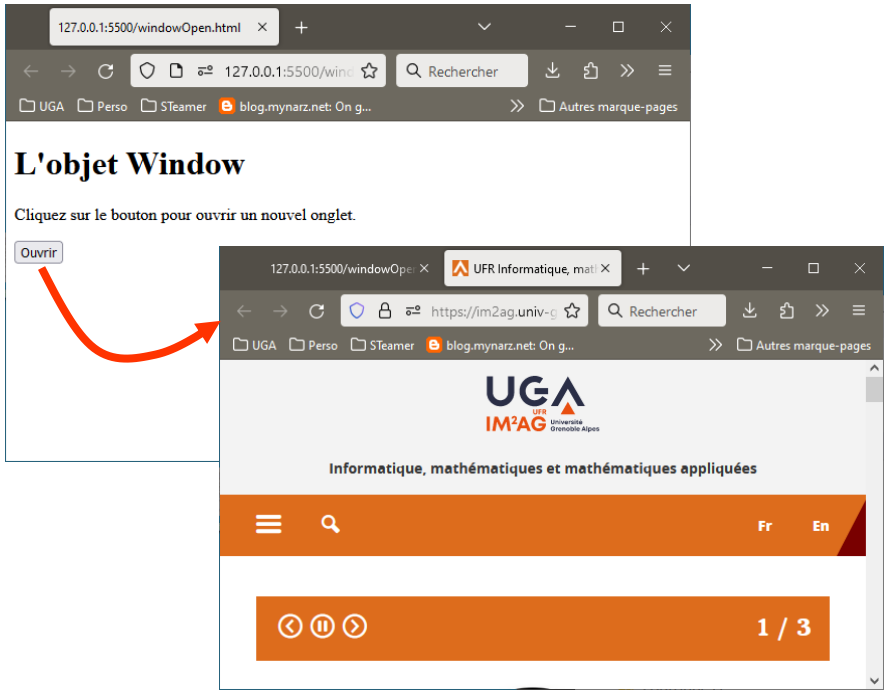
- Les membres d'un objet sont
  - les propriétés (attributs de l'objet)
  - le méthodes (fonctions) associées à l'objet
- L'accès à un membre d'un objet se fait avec notation pointée
  - `identifiantObjet.nomPropriété` pour accéder à un attribut
  - `identifiantObjet.nomMéthode(…)` pour exécuter une méthode de l'objet

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Objet Window </title>
</head>
<body>
    <h1>L'objet Window</h1>
    <p>Cliquez sur le bouton pour ouvrir un nouvel onglet.</p>

    <button onclick="ouvrirTab()">Ouvrir</button>

    <script>
        function ouvrirTab() {
            window.open("https://im2ag.univ-grenoble-alpes.fr/", "_blank");
        }
    </script>
</body>
</html>
```

appel d'une méthode de l'objet **window** pour ouvrir une page dans un nouvel onglet du navigateur



Types de base

- pseudo objets
  - bien que les types `number`, `string` et `boolean` définissent des valeurs primitives directement stockées dans une variable, ils possèdent néanmoins des propriétés et méthodes (on parle parfois de pseudo objets)

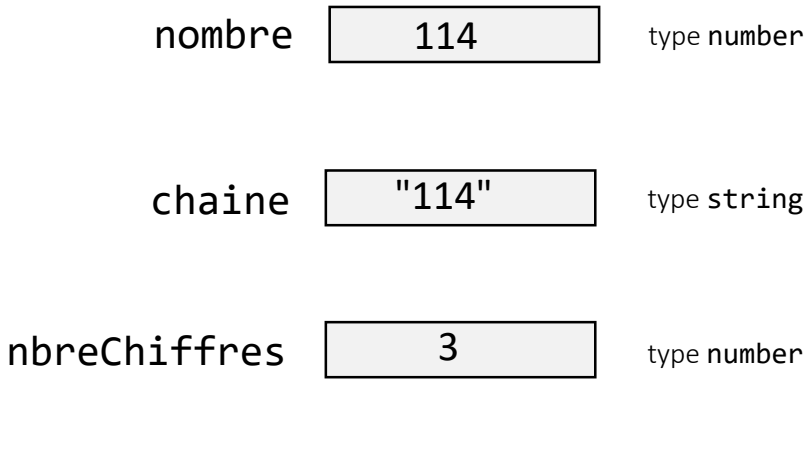
```
let nombre = 114;

let chaine = nombre.toString();

let nbreChiffres = chaine.length;
```

méthode

propriété



Opérateurs et instructions de contrôle

- JavaScript propose les opérateurs et instructions standards de C++/Java
- opérateurs
  - arithmétiques
    `+, -, *, /, %, ++, --, **`
  - comparaisons
    `==, !=, <, >, <=, >=, ===, !==`
  - logiques
    `&&, ||, !`
- instructions
  - conditionnelles
    `if, if-else, switch`
  - itératives
    `for, while, do … while`

```
1  // d'après Dave Reed  js03.html  2/01/04
2  const DISTANCE_TO_SUN = 149597870e3;
3  let thickness = .002;
4
5  let foldCount = 0;
6  while (thickness < DISTANCE_TO_SUN) {
7      thickness *= 2;
8      foldCount++;
9  }
10
11 console.log("Number of folds = " + foldCount);
```

Conversion automatique des types

-  JavaScript effectue des conversions automatiques de types

`50 * 20`	`1000`	
`'50' * '20'`	`1000`	les valeurs de type **`string`** sont converties en valeurs de type **`number`**
`'50' + '20'`	`'5020'`	l'opérateur **`+`** est surchargé quand les opérandes sont de type **`number`** c'est l'addition, quand ils sont de type **`string`** c'est la concaténation de chaînes
`'50' + 20`	`'5020'`	l'un des opérandes est de type **`string`** ⇒ **`+`** est la concaténation de chaînes, la valeur de type **`number`** est convertie en valeur type **`string`**
`50 + '20'`	`'5020'`	l'un des opérandes est de type **`string`** ⇒ **`+`** est la concaténation de chaînes, la valeur de type **`number`** est convertie en valeur type **`string`**
`'50' - 20`	`30`	l'opérateur **`-`** est la soustraction, la valeur de type **`string`** est convertie en valeur de type **`number`**
`50 + null`	`50`	la valeur **`null`** est convertie en la valeur **`0`** de type **`number`**
`'50' + null`	`'50null'`	la valeur **`null`** est convertie en la valeur '**`null`**' de type **`string`**
`50 + undefined`	`NaN`	

Truthy

- En JavaScript, on dit en anglais qu'une valeur est **truthy**[1] lorsqu'elle est considérée comme vraie (`true`) quand elle est évaluée dans un contexte booléen.
- Toutes les valeurs des types autres que booléen sont **truthy** sauf `0`, `""`, `null`, `undefined` et `NaN` qui sont évaluées à `false` dans un contexte booléen.

```
let a;
if (a) {
   console.log('Vrai');
} else {
   console.log('Faux');
}
```

→ Faux

```
let a = 'Hello';
if (a) {
   console.log('Vrai');
} else {
   console.log('Faux');
}
```

→ Vrai

```
let a = parseInt("114z");
if (a) {
   console.log('Vrai');
} else {
   console.log('Faux');
}
```

→ Faux

[1] https://developer.mozilla.org/fr/docs/Glossary/Truthy

Valeurs Falsy

- Les valeurs fausses (**falsy**) sont des valeurs évaluées comme fausses (`false`) quand elles sont évaluées dans un contexte booléen (condition, boucle…).

Table des valeurs **falsy**

Value	Type	Description
null	Null	The keyword `null` — the absence of any value.
undefined	Undefined	`undefined` — the primitive value.
`false`	Boolean	The keyword `false`.
NaN	Number	`NaN` — not a number.
`0`	Number	The `Number` zero, also including `0.0`, `0x0`, etc.
`-0`	Number	The `Number` negative zero, also including `-0.0`, `-0x0`, etc.
`0n`	BigInt	The `BigInt` zero, also including `0x0n`, etc. Note that there is no `BigInt` negative zero — the negation of `0n` is `0n`.
`""`	String	Empty string value, also including `''` and ``` ``.

https://developer.mozilla.org/en-US/docs/Glossary/Falsy

Valeurs *falsy* et *truthly* dans expressions booléennes

```
let a = 0;
let b = "+";
```

```
if (a && b) {
    console.log('a && b évaluée à true');
}
else {
    console.log('a && b évaluée à false');
}

if (a || b) {
    console.log('a || b évaluée à true');
}
else {
    console.log('a || b évaluée à false');
}
```

→ a && b évaluée à false

→ a || b évaluée à true

```
let c = a && b;
let d = a || b;
```

```
console.log('c : ' + c);
```

→ c = 0

```
console.log('d : ' + d);
```

→ d = +



Valeurs *falsy* et *truthly* dans expressions booléennes

```
let a = 0;
let b = "+";
```

```
if (a && b) {
    console.log('a && b évaluée à true');
}
else {
    console.log('a && b évaluée à false');
}

if (a || b) {
    console.log('a || b évaluée à true');
}
else {
    console.log('a || b évaluée à false');
}
```

→ a && b évaluée à false

→ a || b évaluée à true

```
let c = a && b;
let d = a || b;
```

```
console.log('c : ' + c);
```

→ c = 0

```
console.log('d : ' + d);
```

→ d = +



Explication

C'est un comportement fondamental (et souvent surprenant) de JavaScript : les opérateurs **&&** et **||** ne renvoient pas de booléens, mais la valeur de l'**un de leurs deux opérandes** en pratiquant un **court circuit** (*short-circuiting*).

JavaScript évalue le premier opérande, le convertit **implicitement** en booléen pour décider du chemin à prendre,

- s'il trouve une valeur **falsy** l'opérateur **&&** s'arrête là et renvoie telle quelle la valeur du premier opérande; sinon il renvoie la valeur du second opérande.
- s'il trouve une valeur **truthy** l'opérateur **||** s'arrête là et renvoie telle quelle la valeur du premier opérande; sinon il renvoie la valeur du second opérande.

La valeur retournée est la **valeur d'origine** du dernier opérande évalué, pas le résultat de sa conversion en booléen.

Évaluation de **a && b** : **a** est **falsy** → renvoie **a** → **0**

Évaluation de **a || b** : **a** est **falsy** → renvoie **b** → **"+"**

Si on veut un vrai booléen, utiliser **Boolean()** ou une double négation

```
let c = Boolean(a && b); // false
let d = !!(a || b); // true
```

Valeurs *falsy* et *truthly* dans expressions booléennes

- Le comportement précédent n'est pas un bug mais une fonctionnalité souvent utilisée dans du code JavaScript
  - cas d'usage

```
// Valeur par défaut (si a est falsy, prendre b)
let nom = saisie || "anonyme";

// Garde-fou (n'exécute une méthode que si obj existe)
obj && obj.afficher();
```

- Opérateur **??** (*nullish coalescing operator*)
  - opérateur qui ne court-circuite que sur **null** ou **undefined**
  - valeur de l'expression `a ?? b`
    - **a** : si **a != undefined** et **a!= null**
    - **b** : sinon
  - Permet de choisir une valeur à partir d'une liste de variables

```
let a = 0;
let b = '+';

let e = a ?? b; // 0 (car a n'est ni null ni undefined)
```

Conversion automatique des types

- règles nombreuses et pas toujours intuitives

JavaScript Type Conversion Table

https://www.w3schools.com/js/js_type_conversion.asp

This table shows the result of converting different JavaScript values to Number, String, and Boolean:

Original Value	Converted to Number	Converted to String	Converted to Boolean	Try it
false	0	"false"	false	Try it »
true	1	"true"	true	Try it »
0	0	"0"	false	Try it »
1	1	"1"	true	Try it »
"0"	0	"0"	true	Try it »
"000"	0	"000"	true	Try it »
"1"	1	"1"	true	Try it »
NaN	NaN	"NaN"	false	Try it »
Infinity	Infinity	"Infinity"	true	Try it »

Conversion de type et opérateurs d'égalité

- JavaScript possède deux jeux d'opérateurs d'égalité

égalité et différence avec conversion implicite des valeurs si les opérandes sont de type différent.

==

!=



===

égalité **de valeur ET de type**

!==

différence **de valeur OU type**

```
let x = 3;
x == 3   → true
x == '3' → true
```

```
let x = 3;
x === 3   → true
x === '3' → false
```

```
'' == '0'           → false
0 == ''             → true
0 == '0'            → true
false == 'false'    → false
false == '0'        → true
false == undefined  → false
false == null       → false
null == undefined   → true
' \t\r\n' == 0      → true
```

Règles de conversion compliquées et difficiles à mémoriser

utiliser === et !== toujours tu dois !



exemple tiré de *JavaScript, les bons éléments*
Douglas Crockford, Ed. Pearson France, 2013