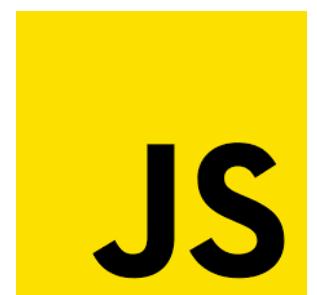


Développement Web

JAVASCRIPT



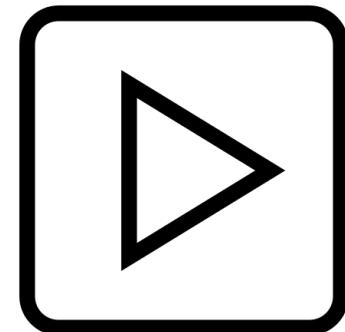
Objectifs pédagogiques

- Comprendre et maîtriser le JavaScript moderne et adopter les bonnes pratiques



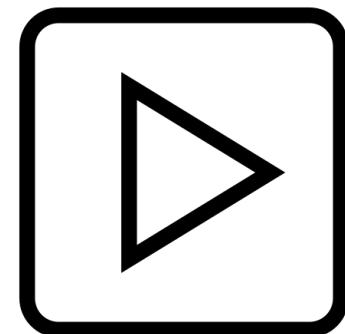
Sommaire (1/2)

- Introduction
- Historique
- Installation
- Fondamentaux
- Bonnes pratiques JavaScript
- JavaScript moderne : DOM, événements
- Ajax, Promises, Fetch vs Axios, Async et Await



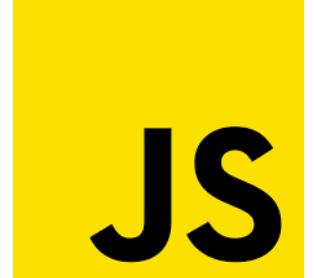
Sommaire (2/2)

- Webpack
- TypeScript
- Mise en production : performances et sécurité
- Web Components
- APIs pour les applications
- Choisir son framework



Développement Web

DÉCOUVERTE DE JAVASCRIPT



JS

Introduction 1/2

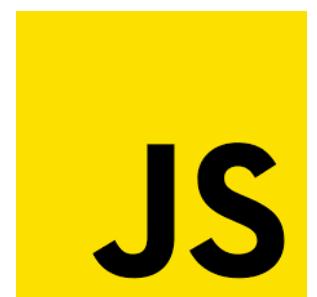
- Langage de script léger, orienté objet, souvent abrégé en « JS »
- Principalement connu comme le langage de script des pages web
- Utilisé dans de nombreux environnements extérieurs aux navigateurs web
 - Node.js
 - Apache CouchDB
 - Adobe Acrobat
- Interprété ou compilé à la volée (JIT)
- JavaScript est **dynamique et non fortement typé**
 - **Dynamique** : Pas nécessaire de définir précisément le type de la variable utilisée
 - **Non fortement typé** : une variable d'un certain type peut devenir une variable d'un autre type au cours de l'exécution du programme

Introduction 2/2

- Langage à objets utilisant le concept de prototype
- Le standard pour JavaScript est **ECMASCRIPT**
- **En 2012** : tous les navigateurs supportent ECMASCIPT 5.1 (**ES2009**)
- Les anciens navigateurs supportent **ES3**
- **En 2015** : une version majeur est finalisée et s'intitule **ES2015 (ES6)**

Développement Web

HISTORIQUE JAVASCRIPT



Histoire 1/4

- En **1995**, création de JavaScript par Brendan Eich
 - Langage de script sur Netscape 2.0
- En **1996**, ajout de Jscript dans Internet Explorer 3.0 par Microsoft
 - Version de Microsoft, connue sous le nom de Jscript
 - Permet d'afficher des alertes, d'étendre des fonctionnalités du navigateur
- En **1997**, sortie du premier standard du langage
 - Spécification ECMA-262 (ECMAScript 1 ère Edition)
 - Standard théorique à cause de l'utilisation majoritaire de IE avec une implémentation incomplète
- Cette guerre des navigateurs a été un frein considérable
 - Favorise l'utilisation d'alternatives comme Flash

Histoire 2/4

- En **2004**, sortie de Firefox par la fondation Mozilla
 - Disparition du monopole de IE
 - Contributions de Brendan Eich
- En **2006**, création de la bibliothèque Jquery (<https://jquery.com>)
 - Permet de développer, générer des applications avec une cross compatibilité
 - Simplifie la syntaxe du JavaScript, avec la proposition d'animations comme celles de Flash
- En **2009**, sortie de Node.js et AngularJS (<https://nodejs.org> & <https://angularjs.org>)
 - Node.js : interpréteur côté serveur développé par Ryan Dahl
 - AngularJS : Framework JavaScript côté client, développé par Google
- À partir de cette date, de nombreux Framework vont voir le jour
 - Développement complet en JavaScript, sans Flash ni Jquery

Histoire 3/4

- En **2013**, sortie de React par Facebook (<https://fr.reactjs.org>)
 - Résout les soucis de performances d'AngularJS
 - Développement d'application mobile avec React Native
- Sortie de Electron par Github (<https://www.electronjs.org>)
 - Permet de développer des applications de bureau, comme Slack
- En **2016**, sortie de Angular 2+ (<https://angular.io>)
 - Rupture totale avec AngularJS et nécessite l'utilisation de Node.js
- L'évolution de JavaScript continue avec ECMAScript
 - Nouvelle syntaxe, écosystème loin des débuts avec les popups

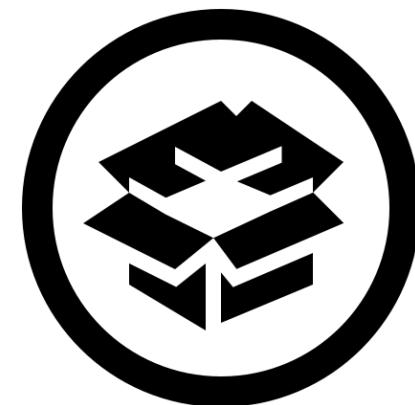
Histoire 4/4

- L'environnement Node.js permet le développement d'un écosystème riche
 - Frameworks
 - ORMs
 - Transpileurs
 - Applications mobiles/ bureautiques
 - Gestionnaire de packages



Développement Web

INSTALLATION

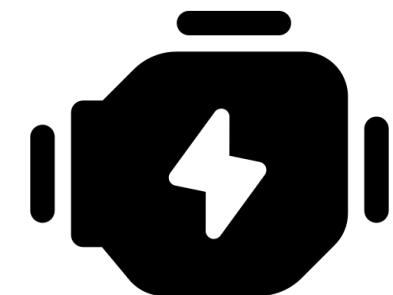


Outils 1/1

- **IntelliJ WebStorm ou VS Code**
 - Gèrent nativement la syntaxe JS
- **Firefox ou Chrome**
 - Outils de développement intégrés
 - Permet d'exécuter le JavaScript
- **NodeJS**
 - Pour générer une application interprétable par un navigateur à partir de plusieurs fichiers organisés
 - **Pas indispensable mais fortement pratique (NPM ou Yarn)**
 - <https://nodejs.org/en/download/>

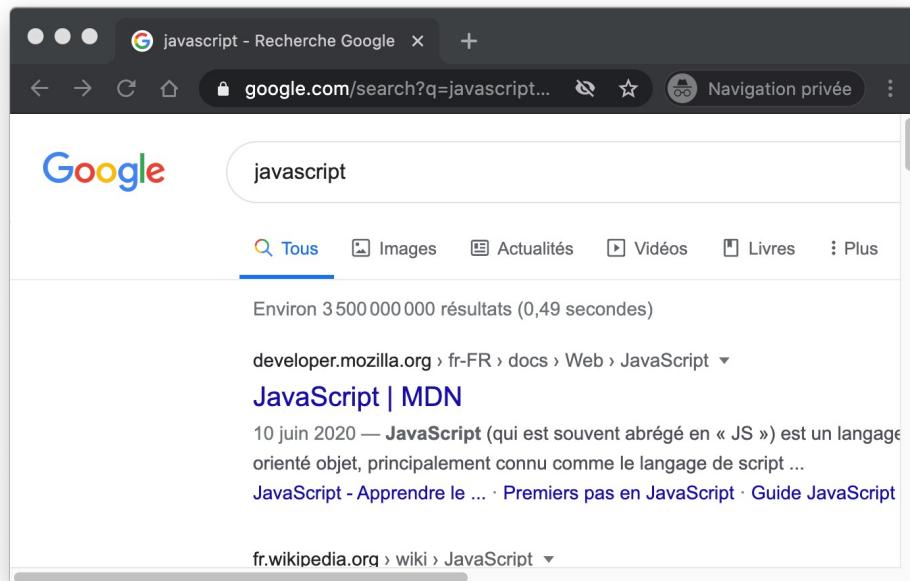
JavaScript & Développement Web

FONDAMENTAUX



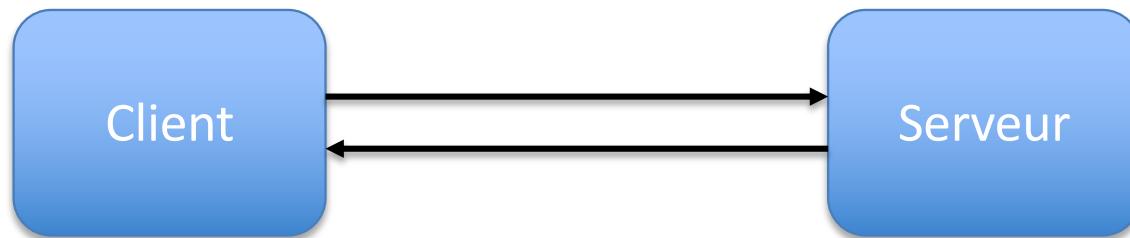
Notions essentielles

- Notions essentielles : Client et Serveur
 - Un programme Client envoie des requêtes à un programme Serveur
 - Un programme Serveur envoie une réponse à son tour
- Exemple : le serveur google nous renvoie une réponse sous forme de code HTML/CSS/JavaScript

A screenshot of a web browser window showing the raw HTML source code of a Google search result. The address bar at the top shows "view-source:https://www.google.com...". The main content area displays the HTML code for the search results page, including the doctype, head, and body sections. The code is heavily obfuscated with many random characters and URL-encoded strings, typical of Google's anti-crawling measures.

Notions essentielles

- **JavaScript** permet de réaliser des applications côté client et serveur
 - **Côté Client** : Langage pour interagir avec notre page web
 - Interaction, Animation
 - Application : messagerie instantanée, outils de gestion, jeu vidéo

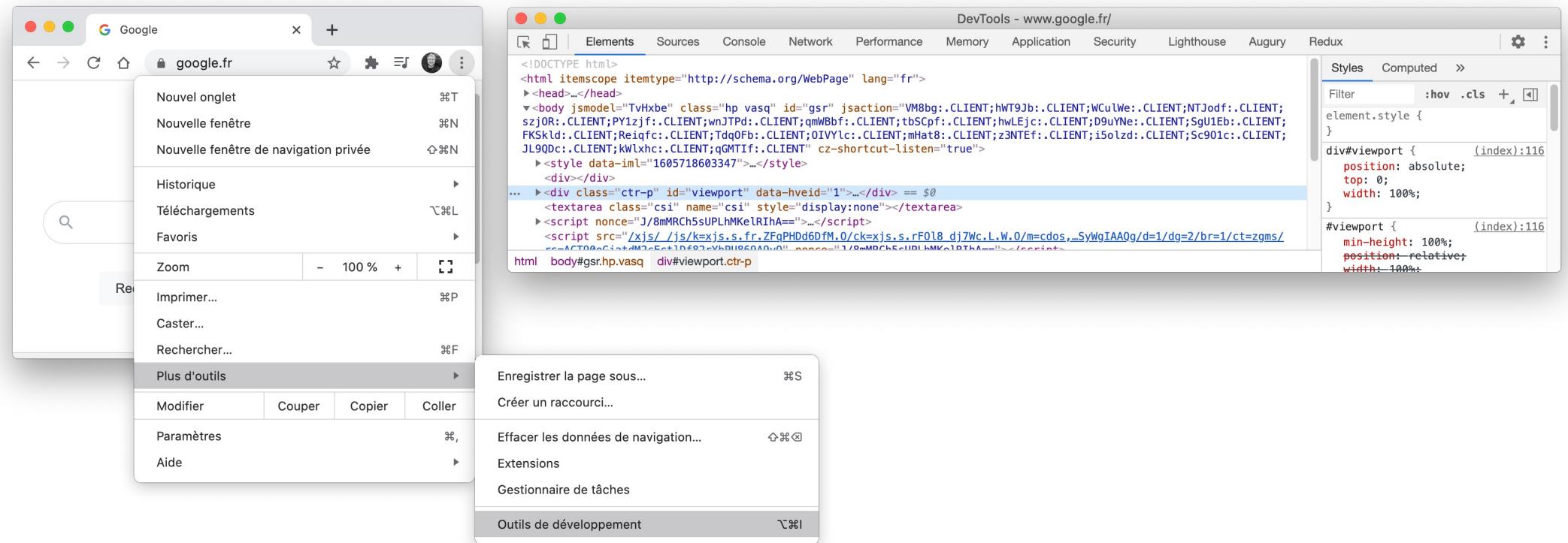


- **Côté Serveur :**
 - Serveur web, communiquer avec des base de données
 - Node.JS, Deno

JavaScript

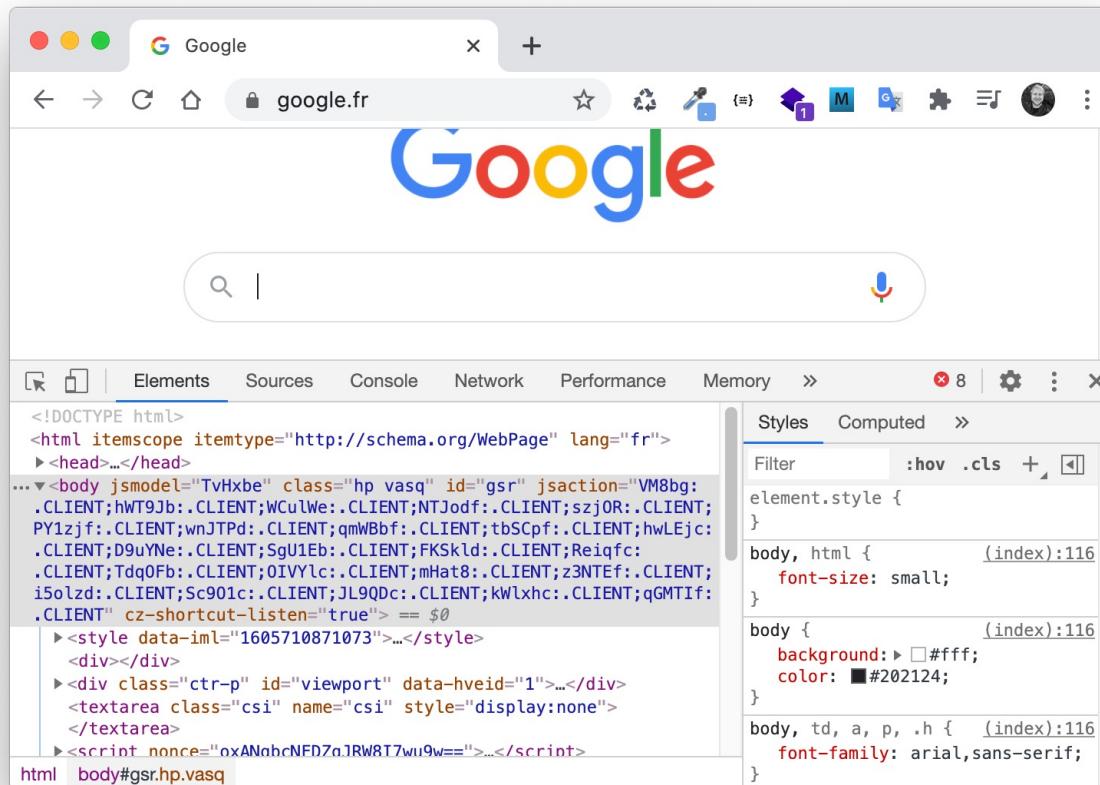
Notions essentielles

- Utilisation des outils essentiels sur le navigateur :
 - **Elements, Sources, Console, Network**



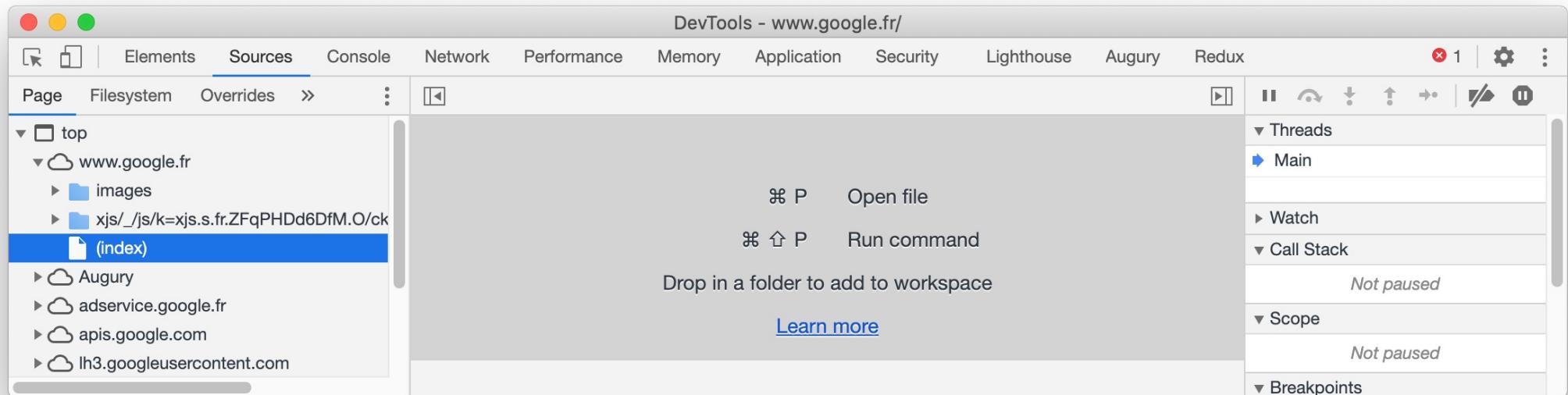
Notions essentielles

- Utilisation de l'inspecteur du navigateur avec **Elements**
 - Permet de sélectionner des éléments visuel et de voir le code source associé



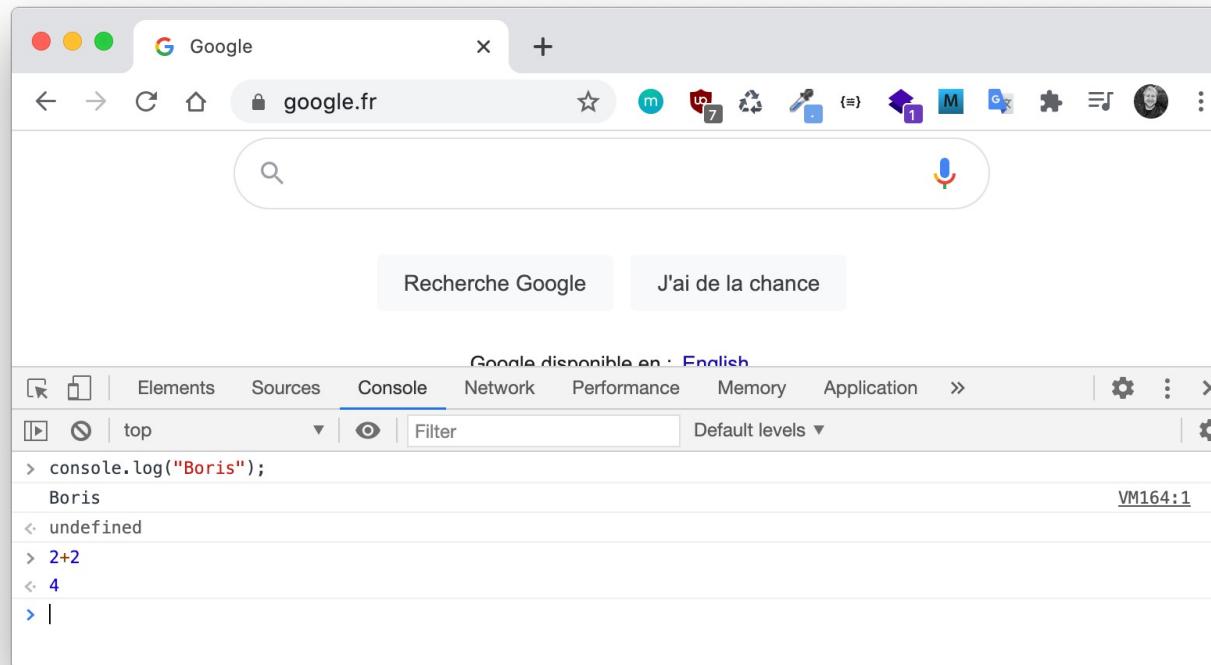
Notions essentielles

- Utilisation de **Sources** du navigateur
 - Permet de voir des éléments visuel et de voir le code source associé



Notions essentielles

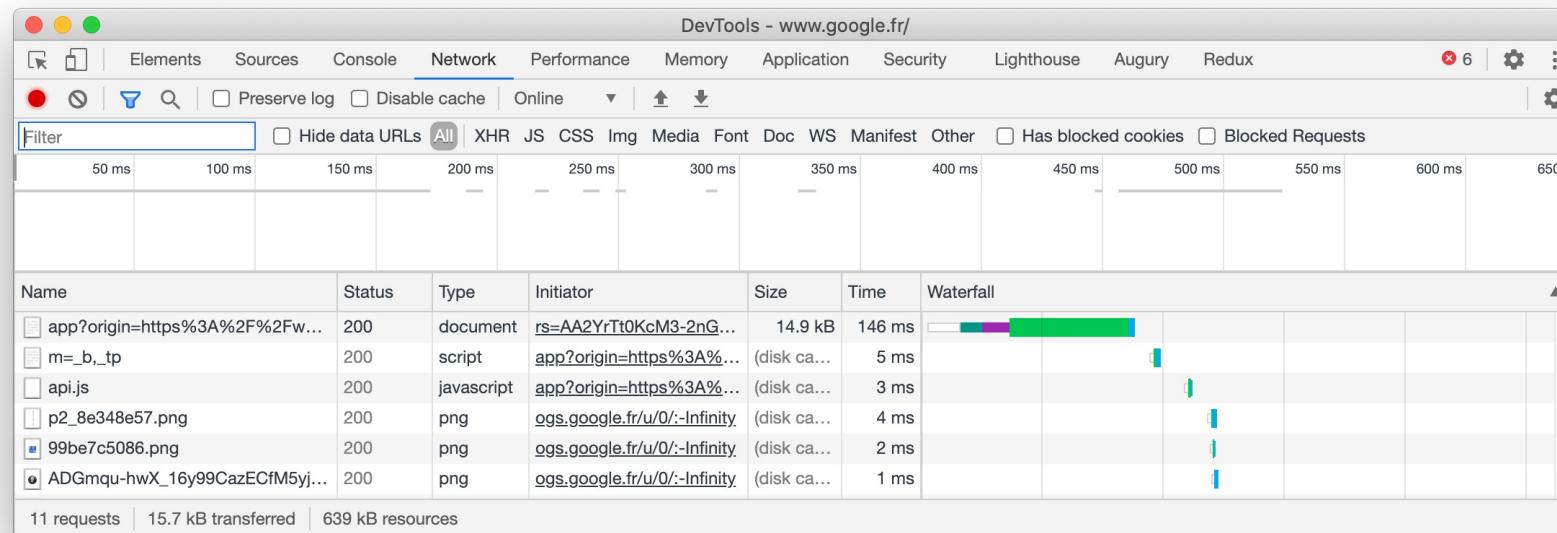
- Utilisation de la **Console** du navigateur
 - Interpréteur du JavaScript



- La flèche retourne le type de donnée renvoyée

Notions essentielles

- Utilisation de **Network** du navigateur
 - Permet de visualiser tous les appels réseau d'une page internet



Notions essentielles

- Les codes de statut de réponse HTTP indiquent si une requête HTTP a été exécutée avec succès ou non.
- Les réponses sont regroupées en 5 classes:
 - Les réponses informatives (100 - 199)
 - Les réponses de succès (200 - 299)
 - Les redirections (300 - 399)
 - Les erreurs du client (400 - 499)
 - Les erreurs du serveur (500 - 599)
- <https://developer.mozilla.org/fr/docs/Web/HTTP/Status>

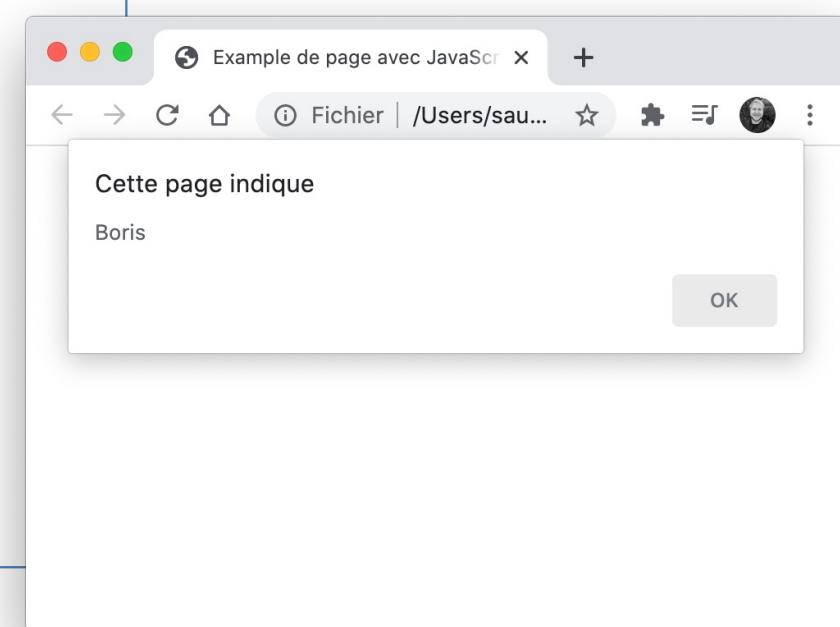
Utilisation de JavaScript

- Création d'un fichier HTML dans lequel on utilise JavaScript :

```
<!DOCTYPE html>
<html>
<head>
  <title>Example de page avec JavaScript</title>

  <script type="text/javascript">
    let firstname = "Boris";
    alert(firstname);
  </script>

</head>
<body>
</body>
</html>
```



 index.html

Utilisation de JavaScript

- Création d'un fichier HTML dans lequel on utilise JavaScript :

```
<!DOCTYPE html>
<html>
<head>
    <title>Example de page avec JavaScript</title>

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

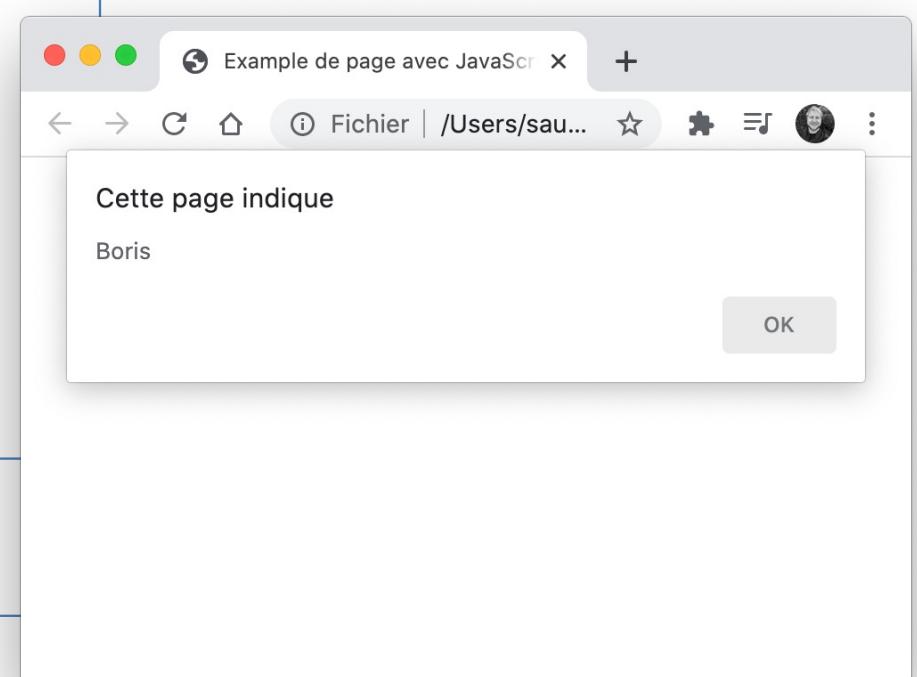
</head>
<body>
</body>
</html>
```



```
let firstname = "Boris";
alert(firstname);
```



example.js



JavaScript : les variables

- **var** est une variable avec une portée globale
 - elle est remontée en haut du fichier lors de l'exécution du fichier
- **let** est une variable avec une portée locale au bloc dans lequel elle est déclarée
- **const** est une variable constante, accessible en lecture

```
var a = 1;  
let b = 2;  
const c = 3;
```

JavaScript : les block

- Un **block** est utilisé afin de grouper zéro ou plusieurs instructions
 - Le bloc est délimité par une paire d'accolades

```
var x = 1;  
{  
    var x = 2;  
}  
console.log(x); // affiche 2
```

```
let x = 1;  
{  
    let x = 2;  
}  
console.log(x); // affiche 1
```

```
const c = 1;  
{  
    const c = 2;  
}  
console.log(c); // affiche 1
```

JavaScript : les conditions

- L'instruction **if** exécute une instruction si une condition donnée est vraie
 - Si la condition n'est pas vérifiée, il est possible d'utiliser une autre instruction
 - Le bloc est délimité par une paire d'accolades

```
function testNum(a) {  
    let result;  
  
    if (a > 0) {  
        result = 'positive';  
    } else {  
        result = 'NOT positive';  
    }  
    return result;  
}  
  
console.log(testNum(-5));  
// expected output: "NOT positive"
```

JavaScript : les boucles **for**

- L'instruction **for** crée une boucle
 - composée de trois expressions optionnelles séparées par des points-virgules
 - encadrées entre des parenthèses qui sont suivies par une instruction

```
let str = "';  
  
for (let i = 0; i < 9; i++) {  
    str = str + i;  
}  
  
console.log(str);  
// expected output: "012345678"
```

- Alternatives :
 - `for...in`
 - `for...of`

JavaScript : les boucles while

- L'instruction **while** créer une boucle qui s'exécute tant qu'une condition de test est vérifiée
 - La condition est évaluée avant d'exécuter l'instruction contenue dans la boucle

```
let n = 0;  
  
while (n < 3) {  
    n++;  
}  
  
console.log(n);  
// expected output: 3
```

- Alternatives :
 - do...while

JavaScript : les switch

- L'instruction **switch** évalue une expression et, selon le résultat obtenu et le cas associé, exécute les instructions correspondantes.

```
const expr = 'Pomme';
switch (expr) {
  case 'Pomme':
    console.log('0.59 euros');
    break;
  case 'Fraises':
  case 'Cerises':
    console.log('1.00 euros');
    break;
  default:
    console.log('Aucun prix');
}
// expected output: "0.59 euros"
```

JavaScript : les fonctions

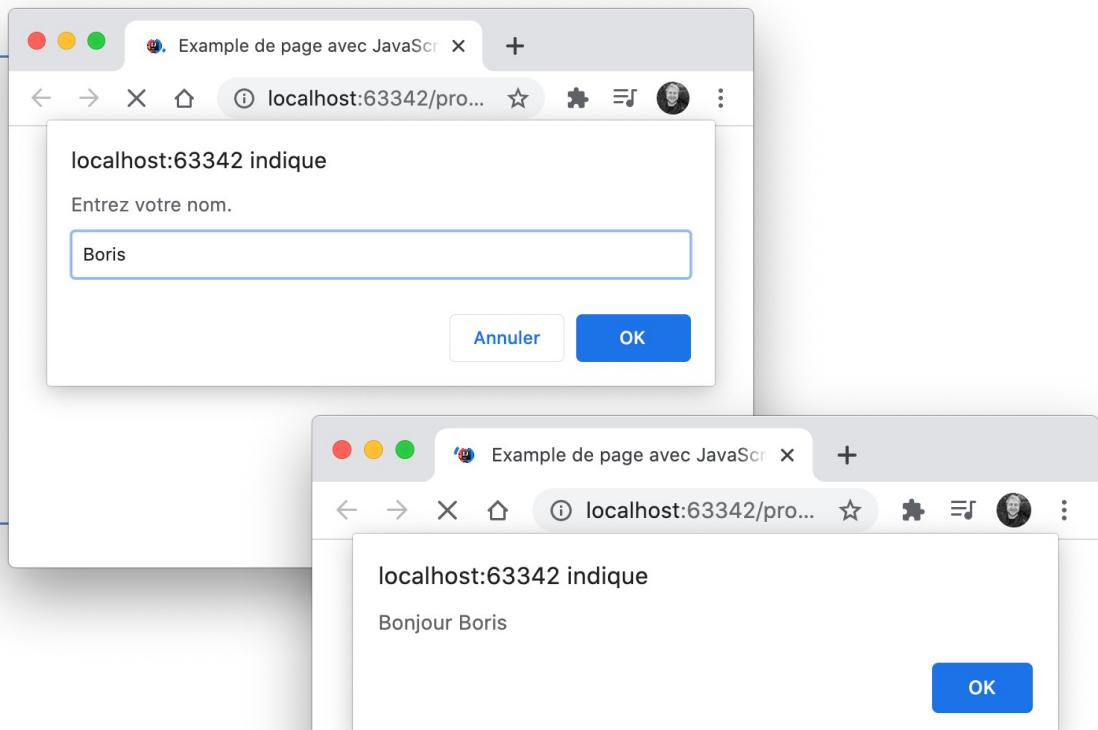
- La déclaration **function** permet de définir une fonction et les paramètres que celle-ci utilise.

```
function calcRectArea(width, height) {  
    return width * height;  
}  
  
console.log(calcRectArea(5, 6));
```

JavaScript : les fonctions de rappel

- Une fonction de rappel (**callback**) est une fonction passée dans une autre fonction en tant qu'argument
 - est ensuite invoquée à l'intérieur de la fonction externe

```
function sayHello(name) {  
    alert('Bonjour ' + name);  
}  
  
function processUserInput(callback) {  
    let name = prompt('Entrez votre nom.');//  
    callback(name);  
}  
  
processUserInput(sayHello);
```



JavaScript : les prototypes

- Les prototypes sont des propriétés appartenant à des objets instanciés
 - JavaScript est un langage prototypé

```
let firstname = "Boris";
console.log(firstname.__proto__);
```

- Cette chaîne de caractère est une **String**
- String hérite d'objet

```
let firstname = "Boris";
console.log(Object.getPrototypeOf(firstname));
```

```
▼ String ⓘ
  ► anchor: f anchor()
  ► big: f big()
  ► blink: f blink()
  ► bold: f bold()
  ► charAt: f charAt()
  ► charCodeAt: f charCodeAt()
  ► codePointAt: f codePointAt()
  ► concat: f concat()
  ► constructor: f String()
  ► endsWith: f endsWith()
  ► fixed: f fixed()
  ► fontcolor: f fontcolor()
  ► fontsize: f fontsize()
  ► includes: f includes()
  ► indexOf: f indexOf()
  ► italics: f italics()
  ► lastIndexOf: f lastIndexOf()
  ► length: 0
  ► link: f link()
  ► localeCompare: f localeCompare()
  ► match: f match()
  ► matchAll: f matchAll()
  ► normalize: f normalize()
  ► padEnd: f padEnd()
  ► padStart: f padStart()
  ► repeat: f repeat()
  ► replace: f replace()
  ► replaceAll: f replaceAll()
  ► search: f search()
  ► slice: f slice()
  ► small: f small()
  ► split: f split()
  ► startsWith: f startsWith()
  ► strike: f strike()
  ► sub: f sub()
  ► substr: f substr()
  ► substring: f substring()
  ► sup: f sup()
  ► toLocaleLowerCase: f toLocaleLowerCase()
  ► toLocaleUpperCase: f toLocaleUpperCase()
  ► toLowerCase: f toLowerCase()
  ► toString: f toString()
  ► toUpperCase: f toUpperCase()
  ► trim: f trim()
  ► trimEnd: f trimEnd()
  ► trimLeft: f trimStart()
  ► trimRight: f trimEnd()
  ► trimStart: f trimStart()
  ► valueOf: f valueOf()
  ► Symbol(Symbol.iterator): f [Symbol.iterator]()
  ► __proto__: Object
  [[PrimitiveValue]]: ""
```

JavaScript : les objets

- Un **objet** possède plusieurs propriétés qui lui sont associées
 - Une propriété peut être vue comme une variable attachée à l'objet
 - Les propriétés d'un objet représentent ses caractéristiques sont accessibles avec le point « . »

```
let person = {
  firstname: "Boris",
  eat: function () {
    console.log(this.firstname + " mange");
  }
};
person.eat();
```



JavaScript : objet et constructeur

- Déclaration d'un objet avec un constructeur

```
const Person = function (firstname) {
    this.firstname = firstname;
};

Person.prototype.eat = function () {
    console.log(this.firstname + " mange");
};

let person = new Person("Boris");
console.log(person);
person.eat();
```

The screenshot shows the DevTools Console tab with the following output:

```
DevTools - localhost:63342/projet%20redaction%20cours%20JS/index.html?_jtt=15isf115qsqtjnv0pj1502njdg
```

Console tab selected.

```
Person {firstname: "Boris"} example.js:10
  firstname: "Boris"
  ▾ __proto__:
    ▷ eat: f ()
    ▷ constructor: f (firstname)
    ▷ __proto__: Object
Boris mange example.js:6
```

The object `Person` has been expanded to show its properties: `firstname` (value "Boris") and a `__proto__` object containing methods `eat` and `constructor`, and another `__proto__` object pointing to `Object`. The message "Boris mange" is also present in the console.

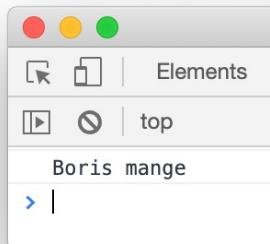
JavaScript : prototype vs méthode

```
const Person = function (firstname) {  
    this.firstname = firstname;  
};
```

```
let person = new Person("Boris");
```

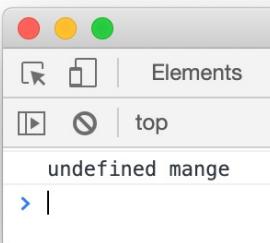
- **La fonction d'un prototype** est une méthode
 - Nécessite d'être utilisée sur une instanciation

```
person.eat();
```

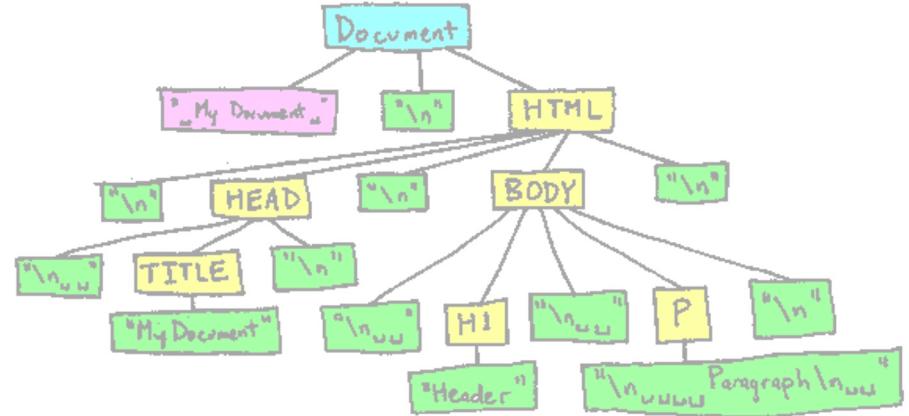


- **La méthode d'un objet** est une méthode statique
 - Ne nécessite pas d'être instanciée pour fonctionner

```
Person.eat();
```



JavaScript : le DOM



- **Le DOM** (Document Object Model) est l'API du navigateur qui permet de manipuler une page web
- Un document HTML est semi-structuré sous forme d'arborescence
 - Chaque nœud est parent ou enfant d'un autre nœud
- Le nœud racine est HTML
- La manipulation du DOM avec JavaScript se fait avec l'interface **document**

JavaScript : l'interface Document

- L'interface **Document** représente n'importe quelle page Web chargée dans le navigateur et sert de point d'entrée dans le contenu de la page Web, qui est l'arborescence DOM
- **Document** possède de propriétés pour manipuler ou récupérer des informations :
 - **document.images** : renvoie la liste des images du document courant
 - **document.forms** : retourne une collection des éléments <form> présent dans le document actuel
- **Document** hérite des interfaces **Node** et **EventTarget**
 - **document.querySelector(#myId)** : retourne le premier élément correspondant au sélecteur
 - **document.querySelectorAll()** : renvoie une **NodeList** correspondant au sélecteur
- <https://dev.w3.org/2006/webapi/selectors-api2/#findelements>

Exercice 1/2

- Faire les exercices suivants :

Convertisseur minuscules en majuscules

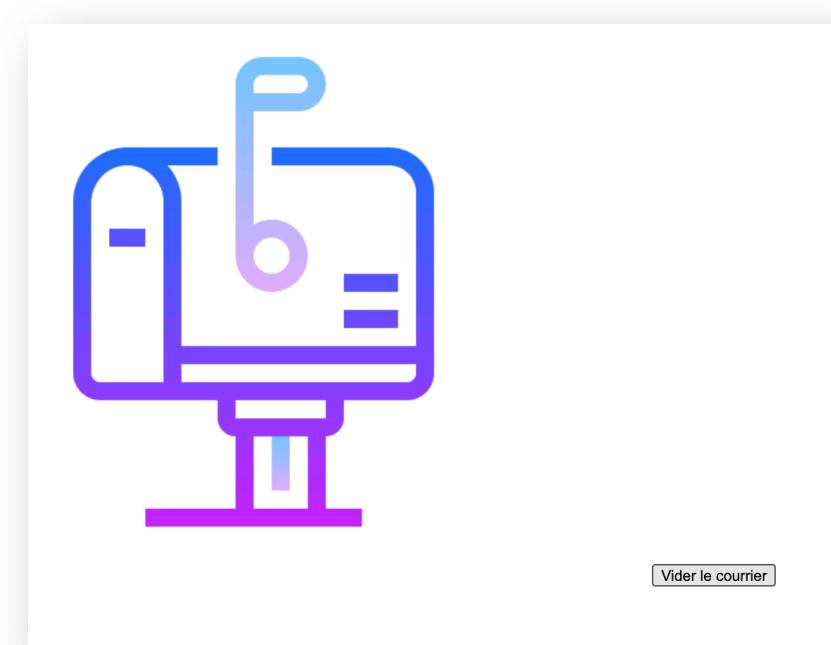
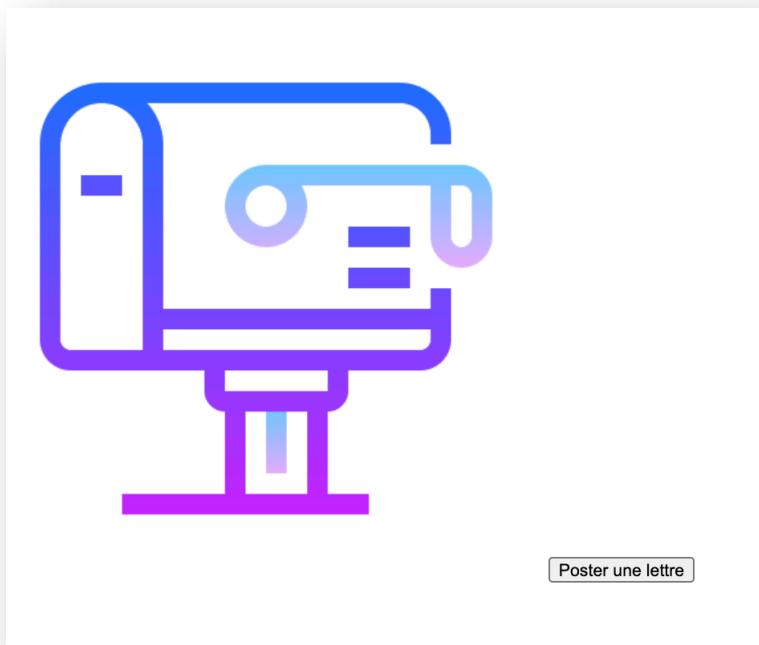
Saisir votre texte en minuscules

Convertir

JavaScript

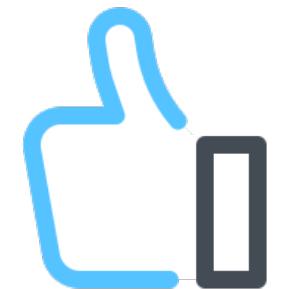
Exercice 2/2

- Faire les exercices suivants :



Développement Web

BONNES PRATIQUES : INTRO



Les bonnes pratiques

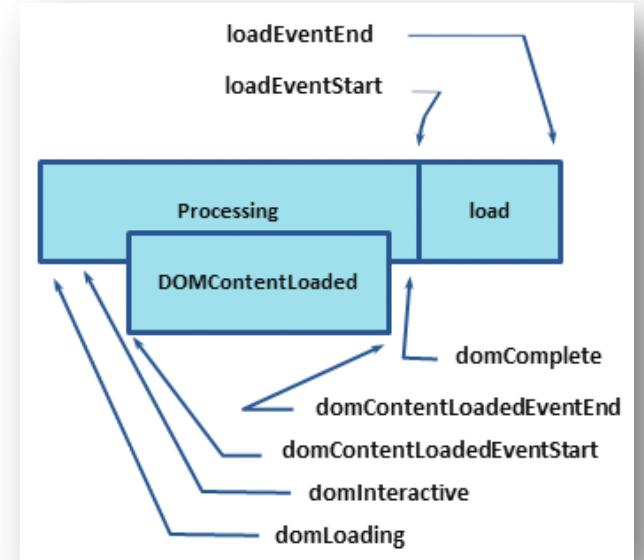
Introduction

- Pour afficher la page, les arborescences DOM et CSSOM doivent être créées
- Le HTML/CSS doivent être transmis au navigateur le plus rapidement possible
 - Le **balisage HTML** est transformé en **DOM** (modèle d'objet de document)
 - Le **balisage CSS** est transformé en **CSSOM** (modèle d'objet CSS)
 - Les modèles DOM et CSSOM sont des structures de données indépendantes
- **Chrome DevTools Timeline** nous permet de capturer et de contrôler les coûts de construction et de traitement des modèles DOM et CSSOM
- Le JavaScript doit aussi être chargé rapidement sans bloquer le processus
- <https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model>

Les bonnes pratiques

Cycle de vie et événements 1/2

- Le cycle de vie d'une page HTML comporte 3 événements importants
- **DOMContentLoaded**
 - l'arborescence DOM est construite et chargée
 - les ressources externes (images, feuilles de styles) ne sont peut être pas chargées
- **load**
 - le navigateur a chargé toutes les ressources (image, styles, autre)
- **beforeunload/unload**
 - l'utilisateur quitte la page



Cycle de vie et événements 2/2

1. Le navigateur analyse le HTML
 2. Il arrête la construction du DOM lorsqu'il rencontre une balise `<script></script>`
 3. Il exécute le script instantanément et continue la construction du DOM
-
- **DOMContentLoaded** est déclenché lorsque tous les script ont été exécutés
 - L'emplacement des balises script à son importance
 - Le même comportement a lieu avec des scripts externes `<script src="">`
 - Nous allons voir qu'il existe des solutions pour optimiser le chargement des scripts

Les bonnes pratiques

DOMContentLoaded 1/2

- L'événement **DOMContentLoaded** est exécuté une fois que le document HTML de base est chargé et que son analyse a eu lieu
 - Cet événement n'attend pas la fin du chargement des modules complémentaires tels que les feuilles de style, les sous-cadres et les images / images
 - <https://developer.mozilla.org/fr/docs/Web/Events/DOMContentLoaded>

```
window.addEventListener("DOMContentLoaded", (event) => {
  console.log("DOM entièrement chargé et analysé");
});
```

```
switch (document.readyState) {
  case "loading":
    log.textContent = 'readyState=Encore en chargement.\n';
    break;

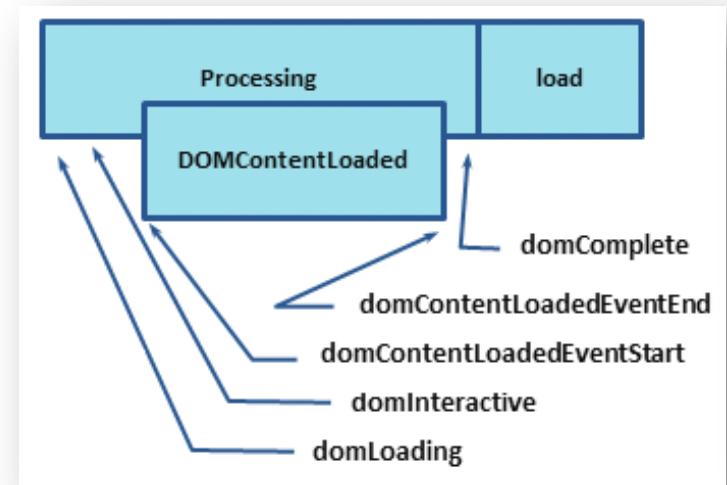
    // alternative à DOMContentLoaded
  case "interactive":
    log.textContent = 'readyState=DOM construit et accessible.\n';
    break;
```

Les bonnes pratiques

DOMContentLoaded 2/2

DOMContentLoaded est composé de :

- **domLoading**
 - Le navigateur démarre l'analyse du HTML
- **domInteractive**
 - L'analyse et la construction du DOM sont terminées
- **DOMContentLoaded (EventStart et EventEnd)**
 - le DOM est prêt, aucun style ne bloque l'exécution de JS
 - EventStart et EventEnd marquent le début et la fin de **DOMContentLoaded**
- **domComplete**
 - la totalité du traitement et le téléchargement des ressources (images, etc.) sont terminés
 - le bouton fléché en cours de chargement a cessé de tourner



Les bonnes pratiques

Load, beforeLoad, unload 1/2

load est composé de :

- **loadEvent**
 - La page est entièrement chargée
- **beforeunload**
 - Déclenché avant que la page et les ressources ne soient déchargées
- **unload**
 - déclenché lorsque la page est complètement déchargée
(utilisable pour envoyer les données analytiques ou pour nettoyer les ressources)

```
window.addEventListener('load', () => {
  console.log("Toutes les ressources sont chargées !");
});

window.addEventListener('beforeunload', () => {
  console.log("Êtes-vous sûr de vouloir quitter la page ?");
});

window.addEventListener('unload', () => {
  console.log("Fermeture de la page");
});
```

Les bonnes pratiques

Load, beforeLoad, unload 2/2

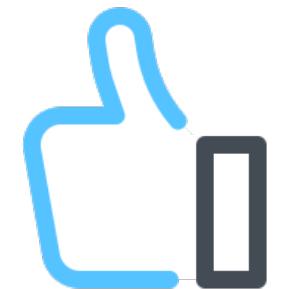
- L'évènement **load** doit être utilisé uniquement pour détecter qu'une page est entièrement chargée
 - C'est une erreur répandue d'utiliser load là où **DOMContentLoaded** serait beaucoup plus approprié

<https://developer.mozilla.org/fr/docs/Web/Events/load>

```
window.addEventListener("load", function (event) {  
    console.log("Toutes les ressources sont chargées !");  
});
```

Développement Web

BONNES PRATIQUES : SCRIPTS



Les types de chargement

- Il est possible de charger nos scripts JavaScript de plusieurs manières différentes
 - Dans le `<head></head>`
 - Dans le `<body></body>`

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

Fonctionnement :

- L'analyseur HTML lance une requête GET lors de la lecture de `<script></script>`
 - Une fois récupéré, le script est exécuté
- Une fois terminé, l'analyse poursuit l'analyse du code HTML restant

Il est important de ne pas nuire aux performances de chargement de page

- Si le script est lent

Chargement dans le footer

- La position de la balise `<script></script>` est importante
 - Lorsque vous apprenez le HTML, les script se situent dans la `<head>`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Example</title>
    <script src="script.js"></script>
</head>
..
```

- Cette pratique retarde l'analyse et le chargement de la page
- Une solution très courante consiste à placer la balise en bas de la page
 - Amélioration considérable
 - Est compatible avec les anciens navigateurs qui ne sont pas compatible avec **async** et **defer**

Les bonnes pratiques

Defer et Async

- Il est possible d'utiliser 2 attributs pour optimiser le chargement :
 - **async** : charger/exécuter les scripts de façon asynchrone
 - **defer** : différer l'exécution à la fin du chargement du document
- Ces attributs sont reconnus par les navigateurs modernes actuels et versions mobiles
 - Si un moteur ne comprend pas l'un ou l'autre, ceci ne se révélera pas bloquant pour l'interprétation du document, les performances resteront simplement "non optimisées"

But : lancer l'interprétation de code JavaScript **sans bloquer le rendu HTML**

<https://caniuse.com/#feat=script-async>

<https://caniuse.com/#feat=script-defer>

Les bonnes pratiques

Defer

- Indique que le navigateur charge les scripts en **parallèle** sans stopper le rendu
 - Attribut existant depuis les "anciennes" versions d'Internet Explorer

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Example</title>
  <script src="script.js" defer></script>
</head>
..
```

- L'ordre d'exécution des scripts est préservé, en fonction de l'apparition dans le HTML
 - reporté à la fin du *parsing* du DOM (avant l'événement DOMContentLoaded)
- De nos jours, cet attribut ne présente plus beaucoup d'intérêt
 - les navigateurs modernes téléchargent les ressources en parallèle sans nécessairement attendre les autres

Les bonnes pratiques

Async

- Signifie que le script pourra être exécuté de façon **asynchrone**
 - le navigateur n'attendra pas de suivre un ordre particulier si plusieurs balises <script> sont présentes
 - Le script ne bloquera pas le chargement du reste des ressources, notamment la page HTML
- L'exécution aura lieu avant l'événement load lancé sur window

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Example</title>
    <script src="script.js" async></script>
</head>
..
```

Rétrocompatibilité : defer + async

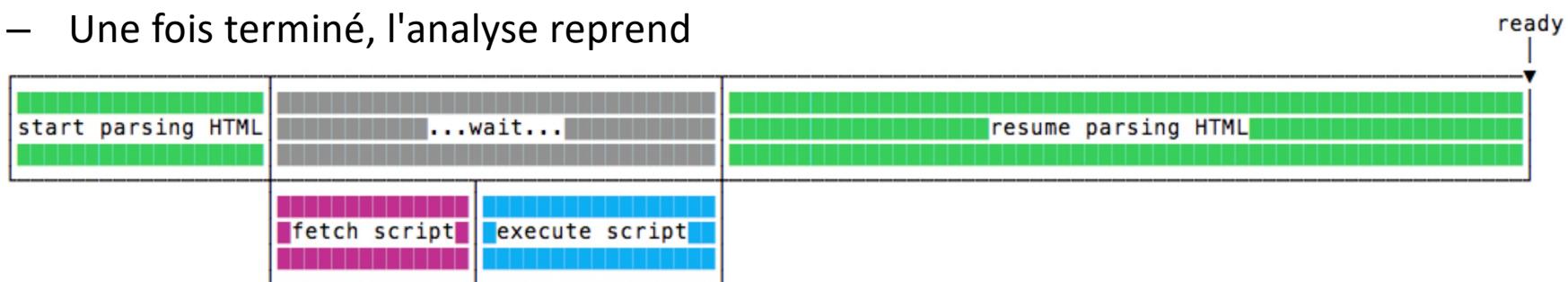
- **defer** peut être spécifié même si l'attribut **async** est spécifié
- Si le navigateurs Web ne prend pas en charge **async**, alors **defer** sera utilisé

```
<script src="example.js" async defer></script>
```

Les bonnes pratiques

Comparatif : sans defer/async

- script dans la partie <head></head>
 - L'analyse est suspendue jusqu'à ce que le script soit récupéré et exécuté
 - Une fois terminé, l'analyse reprend



- script dans la partie <body></body>
 - L'analyse s'exécute sans pause
 - Une fois terminé

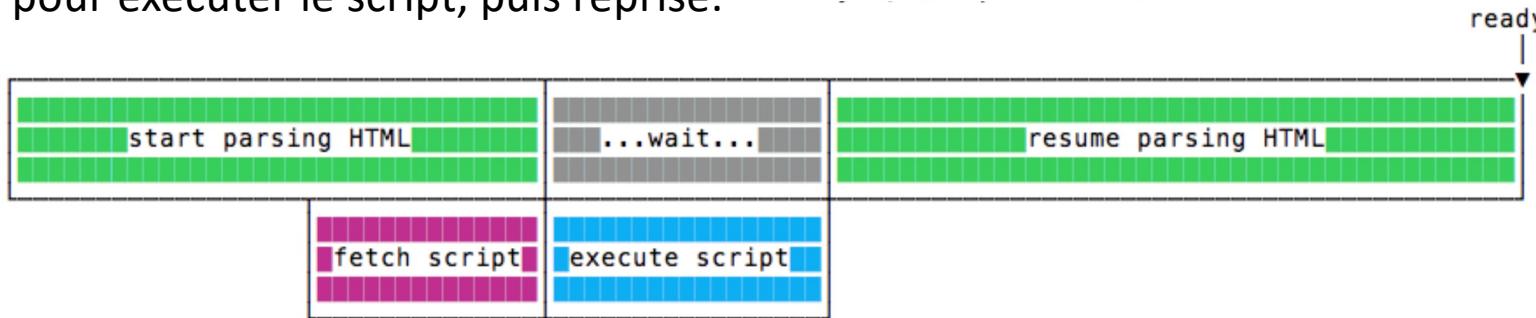


Les bonnes pratiques

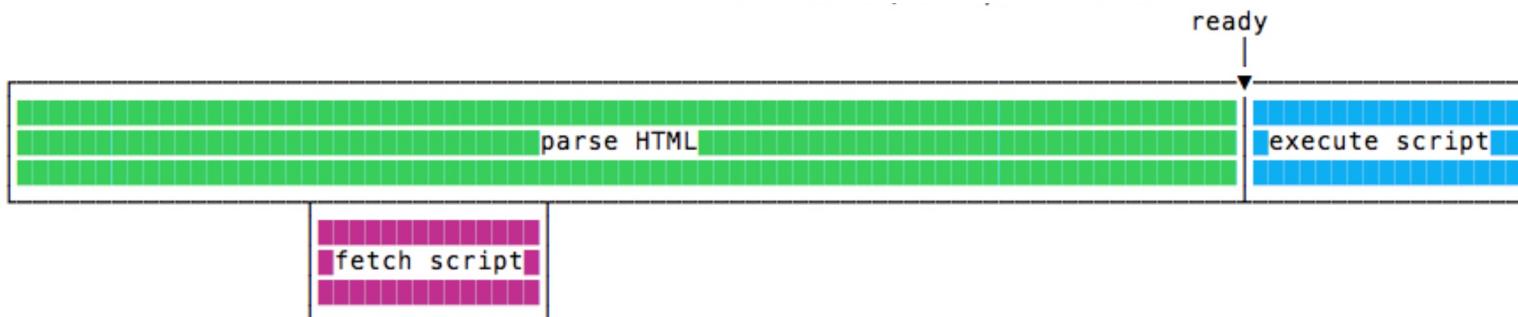
Comparatif : defer/async dans <head>

Avec **defer** ou **async** dans <head>, le script est récupéré de manière asynchrone

- Avec **async**, le script est récupéré de manière asynchrone. Une fois prêt, l'analyse est suspendue pour exécuter le script, puis reprise.



- Avec **defer**, le script a lieu qu'une fois l'analyse HTML terminée



Les bonnes pratiques

Comparatif final

	Analyse HTML	Ordre	DOMInteractive
Async	Bloquant	Chargé en premier. Ordre sans importance.	Non prédictible.
Defer	Non-Bloquant	Ordre des documents respecté.	Exécuté au DOMInteractive (après le chargement l'analyse HTML), juste avant le DOMContentLoaded

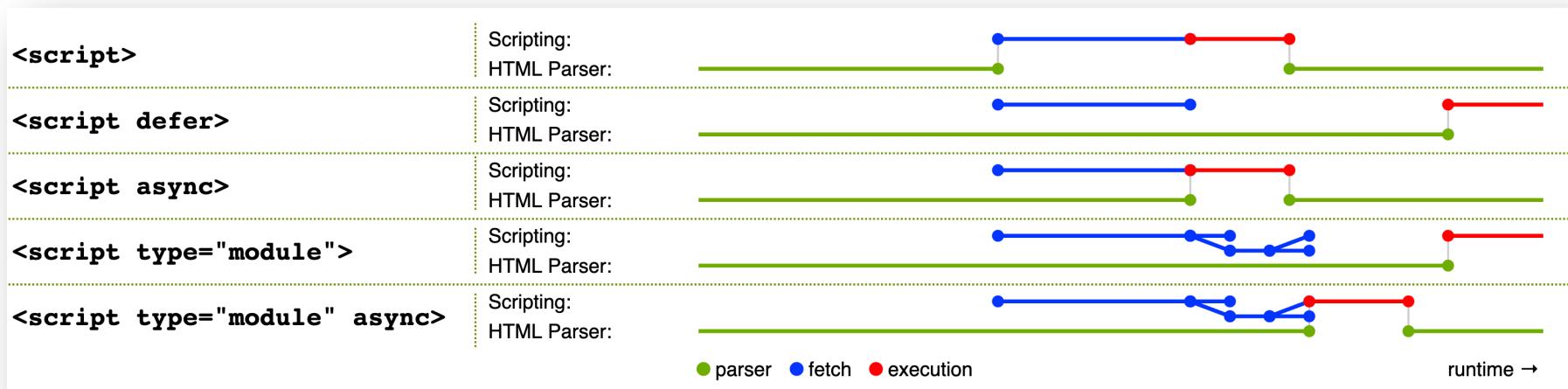
Sources supplémentaires :

- <https://calendar.perfplanet.com/2016/prefer-defer-over-async/>
- <https://javascript.info/script-async-defer>
- <https://html.spec.whatwg.org/multipage/scripting.html#attr-script-async>

Les bonnes pratiques

Alternatives

- Defer.js
 - <https://github.com/shinsenter/defer.js>
- Les modules ES6 :
 - <https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Modules>
 - <https://html.spec.whatwg.org/multipage/scripting.html#attr-script-async>



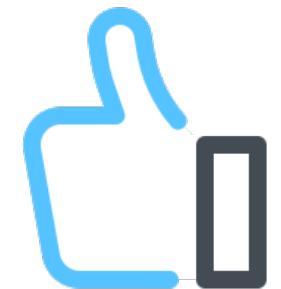
Les bonnes pratiques

Mesurer

- Il est possible de mesurer de nombreuses métriques de vos évènements de vos documents hypertextes
- https://developer.mozilla.org/en-US/docs/Web/API/Navigation_timing_API/Using_Navigation_Timing
- <https://developers.google.com/web/fundamentals/performance/critical-rendering-path/measure-crp?hl=fr>

Développement Web

BONNES PRATIQUES : CDN, MINIFICATION, CACHE...



Les bonnes pratiques

Introduction et vocabulaire

- **Minification** : utilisée pour améliorer les performances web en réduisant la taille du fichier
 - <https://developer.mozilla.org/fr/docs/Glossaire/minification>
- **Concaténation (Regrouper)**: Vous n'aurez plus « N » fichiers CSS, mais un seul
 - <https://www.gdm-pixel.fr/creasite/configurer-autooptimize/>
- **TreeShaking** : Suppression de code mort
 - <https://github.com/rollup/rollup#tree-shaking> ou <https://webpack.js.org/>
- **Cache** : Utilisation du fichier .htaccess afin de configurer la mise en cache de fichiers
 - <https://www.hostinger.fr/tutoriels/exploitation-cache/>
- **CDN** : Utilisation de serveurs pour délivrer du contenu en le livrant à partir de l'emplacement le plus proche du visiteur du site internet
 - <https://www.hostinger.fr/tutoriels/cdn/>

Les bonnes pratiques

Polyfills

- Un polyfill est un bout de code (généralement en JavaScript sur le web) utilisé pour fournir des fonctionnalités récentes sur d'anciens navigateurs qui ne les supportent pas nativement.
- Il est possible de charger un polyfill personnalisé avec <https://polyfill.io/v3/>

Références :

- <https://developer.mozilla.org/fr/docs/Glossaire/Polyfill>
- <https://remysharp.com/2010/10/08/what-is-a-polyfill>
- <https://caniuse.com/>

Les bonnes pratiques

Minification VanillaJS

Outils disponibles en ligne :

- Google : <https://closure-compiler.appspot.com/home>
- MinifyCode : <https://minifycode.com/>

Outils disponibles en paquets :

- cssmin-js : <https://www.phpied.com/cssmin-js/>
- YUI-Compressor: <https://yui.github.io/yuicompressor/>

Mode strict et debugger 1/3

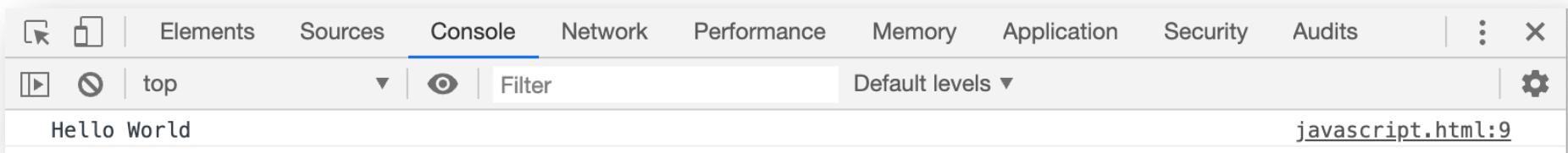
- Le mode strict de ES2009 permet de choisir une variante restrictive pour le JavaScript
 - apporte quelques changements à la sémantique « normale »
 - Elimine certaines erreurs silencieuses en levant une exception
 - Optimise les performances d'exécution
 - Interdit les mot-clés réservés pour les futures versions de ECMAScript
- Pour invoquer le **mode strict** pour un script entier, on ajoutera l'instruction :

```
"use strict";
```

- **REMARQUE : le mode strict est automatiquement appliqué aux corps des classes et aux modules ES2015 (ES6)**

Mode strict et debugger 2/3

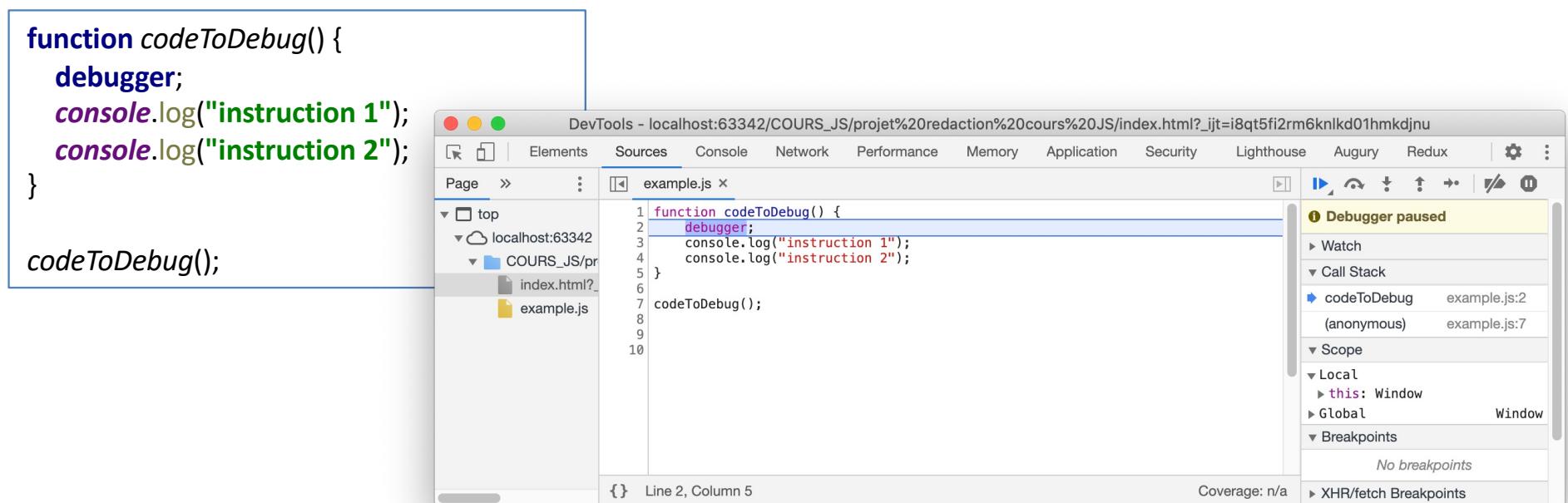
- Il existe de nombreuses méthodes pour afficher des informations dans la console
- **console.log()** est une fonction JavaScript permettant d'afficher dans la console du navigateur
- **console.table()** qui affiche sous forme de tableaux
- **console.dir()** qui affiche sous forme de répertoires
- **console.count()** qui compte le nombre d'appel de la méthode
- **console.time()** qui démarre le chrono
- **console.timeLog()** qui affiche la valeur actuelle d'un **console.time()**



Les bonnes pratiques

Mode strict et debugger 3/3

- L'instruction debugger permet de faire appel à un outil de débogage
 - Si cette fonctionnalité de débogage n'est pas disponible, l'instruction n'aura aucun effet
- Pour utiliser le **debugger** on ajoutera l'instruction :



The screenshot shows the Google Chrome DevTools interface. On the left, the Sources tab is selected, displaying the file `example.js`. The code contains the following:`function codeToDebug() {
 debugger;
 console.log("instruction 1");
 console.log("instruction 2");
}

codeToDebug();`In the middle pane, the code is shown with the line containing `debugger;` highlighted in blue. In the bottom right corner of this line, there is a small yellow square icon with a question mark, indicating a warning or inspection point. The status bar at the bottom of the DevTools window shows "Line 2, Column 5".

The right side of the DevTools window shows the Debugger sidebar, which displays the message "Debugger paused" in green. It also lists the call stack, showing the entry for `codeToDebug` at line 2 of `example.js`, and the current scope, which is the global object `Window`.

Les bonnes pratiques

Conventions de codage

- **Google Guidelines :** <https://google.github.io/styleguide/jsguide.html>
- **Il existe de nombreux outils d'analyse statique de code source**
 - Déetectent les erreurs et problèmes de syntaxe et de style (tabulation, espaces, indentation)
 - **JSLint**, créé en 2002 par Douglas Crockford (<https://www.jslint.com>)
 - Obsolète et présentant de nombreux défauts
 - **JSHint**, fork de JSLint créé en 2010 (<https://jshint.com>)
 - Version JSLint maintenue et configurable
 - **ESLint**, fusionné avec JSCS en 2013 (<https://eslint.org>)
 - Il supporte de nombreux standards récents et frameworks
 - Propose l'utilisation des règles de Airbnb, Google, standardJS

Les bonnes pratiques

La librairie standard

- JavaScript est un vieux projet sans site officiel
- Mozilla possède un site avec toutes les informations de la librairie standard
 - MDN : <https://developer.mozilla.org/fr/>
 - Disponible dans plusieurs langues
- Langage haut niveau, JavaScript nécessite d'être encadré
 - https://developer.mozilla.org/fr/docs/Web/JavaScript/Language_Resources

Développement Web

ECMASCRIPT



L'ECMAScript c'est quoi ?

- Ensemble de normes concernant les langages de type script
 - Standardisé par Ecma International dans le cadre de la spécification ECMA-262
 - Standard dont les spécifications sont mises en œuvre dans différents langages de script
 - Langage de programmation orienté prototype
- L'ES5 date de 2009
- L'ES6 date de 2015
- Les différences entre l'ES5 (ES2009) et l'ES2015 sont majeures
- ECMAScript a décidé de sortir une mise à jour par an
 - 2015 (**ES2015 ou ES6**)
 - 2016 (**ES2016 ou ES7**)
 - 2017 (**ES2017 ou ES8**)
 - 2018 (**ES2018 ou ES9**)
 - 2019 (**ES2019 ou ES10**)
 - 2020 (**ES2020 ou ES11**)

Déclaration des variables

- Il est possible de définir une variable avec la portée locale à un bloc
- Le **mot clef let** permet de déclarer une variable visible (uniquement dans le bloc courant)

`let x = 1;`
- **Quelle est la différence entre let et var ?**
 - La portée (**le scope**) de la variable
 - **Avec let**, la variable est locale, elle est donc accessible uniquement dans le bloc où elle a été déclarée
 - **Avec var**, la déclaration de la variable est remontée au début du code et sa portée est globale

Déclaration constantes

- Permet de figer la référence
- Une erreur sera produite si on tente de modifier val

```
const val = 42;
```

- Par convention on peut l'écrire en majuscules, cela permet de les repérer rapidement

```
const FIRSTNAME = 'Boris';
```

Les fonctions fléchées 1/2

- Il existe de nouvelles manières d'écrire les fonctions
- La nouvelle syntaxe se rapproche de la syntaxe du C#, Java 8 ou coffeescript

```
// ES5 (2009)
var addition = function (x) {
    return x + 1;
}
```

```
// ES6 (2015)
let addition = (x) => {
    return x + 1;
}
```

// Avec un seul argument, pas besoin de parenthèses

```
let addition = x => {
    return x + 1;
}
```

// Avec une seule instruction, pas besoin d'accolades et du return

```
let addition = x => x + 1;
```

Les fonctions fléchées 2/2

- Les fonctions fléchées utilisent un **this** lexical
 - la fonction n'aura pas son **this**
 - elle utilisera le **this** du scope parent
- Les fonctions fléchées **ne sont pas totalement identiques** aux fonctions classiques
 - Elles ne changent pas les valeurs **this**, **arguments** et **super**
 - Ces valeurs correspondent à celles de l'objet englobant (**scope parent**)

```
const objet = {};
objet.m1 = function () {
    return this
};
objet.m2 = () => this;
window.console.log(objet.m1()); // affiche l'objet

window.console.log(objet.m2()); // affiche window
```

Template string

- Les chaînes de caractères peuvent être contenues :
 - entre des apostrophes : 'Example'
 - entre des guillemets doubles : "Example"
- Avec ES2016, les chaînes de caractères peuvent être contenues
 - entre des backquotes `
- Plus besoin de concaténer avec des + les chaînes de caractères

```
// Avant
var hello = function (name) {
    return "Your welcome " + name;
}
hello("Jean");
```

```
// Avec ES2016
let hello = (name) => {
    return `Your welcome ${name}`;
}
hello("Jean");
```

Les paramètres par défaut

- Désormais possible de définir des paramètres par défaut

```
// AVANT ES2016 :  
// Sans paramètre  
var hello = function(name) {  
    return "Your welcome " + name;  
}  
hello(); // return "Your welcome undefined"  
  
// On était donc obligé de faire  
var hello = function(name) {  
    if (name === "undefined") {  
        name = "Inconnu";  
    }  
    return "Your welcome " + name;  
}  
  
hello(); // return "Your welcome inconnu"
```

```
// AVEC ES2016  
let hello = (name="inconnu") => {  
    return "Your welcome " + name;  
}
```

Rest parameters

- permet de représenter un nombre indéfini d'arguments sous forme d'un tableau

```
let logValues = (...values) => {
    // values devient un tableau avec toutes les valeurs.
    console.log(values);
}
logValues("Jean", 25, "Example");
// values == ['Jean', 25, 'Example']
```

Exemple 1

```
const argsToObject = (keys, ...values)=> {
    let object = {};
    object[keys] = values;
    return object;
}

let o = argsToObject('fruits', 'pomme', 'poire', 'abricot')
// {fruits: [ 'pomme', 'poire', 'abricot' ]}
```

Exemple 2

Spread operator 1/2

- Découpe un tableau/objet en plusieurs valeurs sous forme d'un tableau

```
let ingredients = ["Tomate", "Poivron", "Farine"];
```

```
let recette = [...ingredients, "Poisson", "Sel"];
```

```
// recette == ["Tomate", "Poivron", "Farine", "Poisson", "Sel"]
```

Exemple 1

```
let args = ["var 1", "var 2", "var 3"];
```

```
console.log(...args);
```

```
// == console.log('var 1', 'var 2', 'var 3')
```

Exemple 2

Spread operator 2/2

- Depuis ES2018, on peut le faire aussi avec des objets :

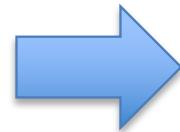
```
let hero = {  
    name: "Moriarty",  
    vie: 97,  
    xp: 11,  
    attaque(id) {  
        //...  
    }  
};  
  
let hero2 = {...hero};
```

Object Destructuring 1/2

- Javascript permet de décomposer certaines propriétés d'un objet

```
let person = {  
    firstName: 'John',  
    lastName: 'Doe'  
};
```

```
let firstName = person.firstName;  
let lastName = person.lastName;
```



```
let { firstName: fname, lastName: lname } = person;  
console.log(fname);  
console.log(lname);
```

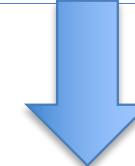
Object Destructuring 2/2

- Javascript permet de décomposer certaines propriétés d'un objet
 - Fonctionne aussi avec des paramètres de fonctions

```
let person = {  
  firstName: 'John',  
  lastName: 'Doe'  
};
```

```
sayHello(person);
```

```
const sayHello = (person) => {  
  console.log(`Hey ${person.firstName} ${person.lastName}`);  
}
```



```
const sayHello = ( {firstName, lastName} ) => {  
  console.log(`Hey ${firstName} ${lastName}`);  
}
```

Les objets 1/3

- ES2015 simplifie l'initialisation des objets
 - Simplification pour les propriétés

```
let lastname = "SAUVAGE";
let firstname = "Boris";
let email = "email@email.fr";
```

```
let identity = {
  firstname: firstname,
  lastname : lastname,
  other : {
    email: email
  }
};
```



```
let identity = {
  firstname,
  lastname,
  other : {
    email
  }
};
```

Les objets 2/3

- ES2015 simplifie l'initialisation des objets
 - Simplification pour les méthodes

```
let lastname = "SAUVAGE";
let identity = {
  lastname,
  getLastname: function() {
    return this.lastname;
  }
};
console.log(identity.getLastname()); // SAUVAGE
```

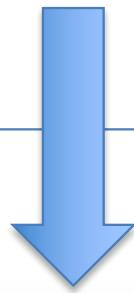


```
let lastname = "SAUVAGE";
let identity = {
  lastname,
  getLastname() {
    return this.lastname;
  }
};
console.log(identity.getLastname()); // SAUVAGE
```

Les objets 3/3

- ES2015 simplifie l'initialisation des objets
 - Simplification pour les méthodes

```
let value = "last";  
  
let identity = {  
  [value + "name"]: "SAUVAGE"  
};  
  
console.log(identity.lastname); // SAUVAGE
```



```
MBP-de-Boris:Desktop sauvageb$ node test.js  
SAUVAGE
```

Les objets : méthode assign

- ES2015 introduit de nouvelles méthodes pour Object
 - **Object.assign** : est utilisée pour copier les valeurs de toutes les propriétés non héritées d'un objet

```
const identity = { lastname: "SAUVAGE" };
console.log(identity);
Object.assign(identity, { firstname: "Boris" });
console.log(identity);
```

```
MBP-de-Boris:Desktop sauvageb$ node test.js
{ lastname: 'SAUVAGE' }
{ lastname: 'SAUVAGE', firstname: 'Boris' }
```

- **Surcharge de propriétés** : les propriétés ont le même nom

```
const identity = { lastname: "SAUVAGE", firstname: "Bo" };
console.log(identity);
Object.assign(identity, { firstname: "Boris" });
console.log(identity);
```

```
MBP-de-Boris:Desktop sauvageb$ node test.js
{ lastname: 'SAUVAGE', firstname: 'Bo' }
{ lastname: 'SAUVAGE', firstname: 'Boris' }
```

- **Avec des accesseurs** : propriété dont la valeur est celle renvoyée par l'accesseur

```
const identite = { lastname: "SAUVAGE", firstname: "Boris",
  get infos() { return this.firstname + " " + this.lastname }
};
console.log(identite);
let copie = Object.assign({}, identite);
console.log(copie);
```

```
MBP-de-Boris:Desktop sauvageb$ node test.js
{ lastname: 'SAUVAGE', firstname: 'Boris', infos: [Getter] }
{ lastname: 'SAUVAGE', firstname: 'Boris', infos: 'Boris SAUVAGE' }
```

Les classes

- JavaScript permet d'utiliser les **classes**
 - syntaxe plus simple pour des objets

```
class Cat {  
  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    meow() {  
        console.log("Miaouu");  
    }  
}  
  
let myCat = new Cat("Réglisse", 2);  
myCat.meow();      // Miaouu
```

L'héritage de classe

- Il est possible de faire de l'héritage de classe
 - Héritage prototypal

```
class SayHello extends Hello {  
    constructor() {  
        super();  
    }  
  
    getHello2() {  
        return `Bienvenue ${this.name}`  
    }  
}  
  
let say = new SayHello("Jean");  
console.log(say.getHello()); //return "Your welcome Jean"  
console.log(say.getHello2()); //return "Bienvenue Jean"
```

- La fonction **super()** récupère les propriétés de la classe parente

Exercice : OOP JavaScript 1/4

- **Créez une classe Pokémon :** Un Pokémon possède un nom, un numéro d'identification dans le Pokédex, une taille, un poids, un type, un certain nombre de points de vie et deux attaques différentes qui peuvent infliger un certain nombre de points de dégâts.
- **Une attaque est une classe** composée de deux attributs : un libellé et un nombre de dégâts.
- À partir de ces deux classes, instanciez deux Pokémon : Pikachu et Evoli.
 - **Pikachu** est le Pokémon numéro 025. Il est de type électrique, il mesure 40 cm et pèse 6 kilos. Il a 82 points de vie et ses deux attaques sont **statik**, qui inflige 10 points de dégâts, et **paratonnerre**, qui en inflige 25
 - **Evoli** est le Pokémon numéro 133. Il est de type normal, il mesure 30 cm et pèse 6,5 kilos. Il a 70 points de vie et ses deux attaques sont **adaptabilité**, qui inflige 15 points de dégâts, et **anticipation**, qui en inflige 30.

Exercice : OOP JavaScript 2/4

Attaque

- Ajoutez une méthode attaque() dans la classe Pokemon. Celle-ci permet à un Pokémon d'en combattre un autre

Pokémon attaquant :

- S'il n'a plus que 20 % de points de vie
 - le Pokémon lance la deuxième attaque, plus puissante
- Sinon, la première attaque est lancée
 - (celle qui inflige le moins de dégâts) qui est lancée

Exercice : OOP JavaScript 3/4

Combat

- Organisez le combat des deux Pokémons Pikachu et Evoli
- Chaque Pokémons attaque à tour de rôle. Un Pokémons est KO lorsque son total de points de vie est inférieur à zéro. Le premier attaquant est décidé par tirage au sort

La méthode `Math.random()` retourne un nombre flottant aléatoire compris entre 0 et 1

- Chaque attaque d'un Pokémons a 10 % de chances d'être un coup critique et d'infliger 100 % de dégâts supplémentaires.
- L'application déclare le combat, journalise chaque attaque et affiche le vainqueur en indiquant le nombre de points de vie restants.

Exercice : OOP JavaScript 4/4

Pikachu VS Evoli

Le tirage au sort a décidé que Evoli attaquait en premier.

Evoli a 70 points de vie.

Pikachu a 82 points de vie.

Evoli a attaqué. Il a fait 9 de dégâts. Pikachu a attaqué. Il a fait 10 de dégâts.

Evoli a attaqué. Il a fait 9 de dégâts. Pikachu a attaqué. Il a fait 10 de dégâts.

Evoli a attaqué. Il a fait 9 de dégâts. Pikachu a attaqué. Il a fait 10 de dégâts.

Evoli a attaqué. Il a fait 9 de dégâts. Pikachu a attaqué. Il a fait 10 de dégâts.

Evoli a attaqué. Il a fait 9 de dégâts. Pikachu a attaqué. Il a fait 10 de dégâts.

Evoli a attaqué. Il a fait 18 de dégâts. Pikachu a attaqué. Il a fait 10 de dégâts.

Evoli a attaqué. Il a fait 15 de dégâts. Pikachu a attaqué. Il a fait 25 de dégâts.

Pikachu a gagné le combat. Il lui restait 4pv.

ECMAScript : Les nouveautés

FICHIERS ET MODULES

Les modules

- Mécanisme pour diviser les programmes JavaScript en plusieurs modules qu'on pourrait importer les uns dans les autres
- Présents dans de nombreux Frameworks comme Node.js : **CommonJS**
- Nous allons voir les systèmes de modules **CommonJS** et **ES6**

Pourquoi les modules ?

- Lorsque nous utilisons une variable, elle est disponible dans le contexte global
 - C'est problématique d'avoir d'avoir toutes nos variables dans le contexte globale

```
<script>
  var a = 'a';
</script>
```

The screenshot shows a browser's developer tools console tab labeled "Console". It displays the following output:
> a
< "a"
> window.a
< "a"

- Lorsque nous utilisons une variable dans une fonction, la portée est locale :
 - C'est une solution limitée à un fichier

```
const fn1 = () => {
  var a = 'a';
  console.log(a, a);
};

fn1();
```

```
() => {
  var a = 'a';
  console.log(a, a);
})();
```

Exemple d'une IIFE

The screenshot shows a browser's developer tools console tab labeled "Console". It displays the following output:
a a
> window.a
< undefined

- Historiquement, JavaScript a créé **CommonJS** pour faciliter la gestion des variables
 - Récemment, les modules ES6

CommonJS : Système Node.js 1/3

- Chaque fichier est un module, totalement isolé des autres modules
- Chaque élément que nous souhaitons partager avec l'extérieur doit être exporté

Exemple : export avec module.exports

```
const getFullName = (name, surname) => {
  return name + " " + surname;
}
const getSurName = (name, surname) => surname;

module.exports = {getFullName, getSurName};
```



```
const lib = require("./lib");

console.log('getFullName', lib.getFullName('Boris S', 'sauvageb'));
console.log('getSurName', lib.getSurName('Boris S', 'sauvageb'));
```

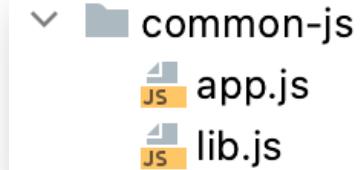


```
Terminal: Local +
MacBook-Pro-de-Boris:common-js sauvageb$ node app.js
getFullName Boris S sauvageb
getSurName sauvageb
MacBook-Pro-de-Boris:common-js sauvageb$
```

CommonJS : Système Node.js 2/3

Exemple : export sélectif avec exports.maFonction

```
exports.getFullName = (name, surname) => {  
    return name + " " + surname;  
}  
exports.getSurName = (name, surname) => surname;
```



```
const lib = require("./lib");  
  
console.log('getFullName', lib.getFullName('Boris S', 'sauvageb'));  
console.log('getSurName', lib.getSurName('Boris S', 'sauvageb'));
```



Terminal: Local +

```
MacBook-Pro-de-Boris:common-js sauvageb$ node app.js  
getFullName Boris S sauvageb  
getSurName sauvageb  
MacBook-Pro-de-Boris:common-js sauvageb$
```

CommonJS : Système Node.js 3/3

La fonction **require** est utilisable dans des conditions :

```
let lib;
if (1 === 1) {
  lib = require("./lib");
}
console.log('getFullName', lib.getFullName('Boris S', 'sauvageb'));
console.log('getSurName', lib.getSurName('Boris S', 'sauvageb'));
```

La déstructuration ES6 peut-être recommandée :

```
const {getFullName, getSurName} = require("./lib");

console.log('getFullName', getFullName('Boris S', 'sauvageb'));
console.log('getSurName', getSurName('Boris S', 'sauvageb'));
```

ES6 Modules 1/3

- Comme tout système de module, ES6 permet d'effectuer les mêmes actions
 - En revanche, les Modules ES6 ne fonctionnent pas en local, ils nécessitent un serveur web (**npm serve**)
 - Il est nécessaire d'indiquer l'extension du fichier lors de l'import

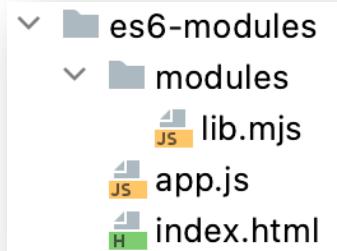
Exemple : export avec export default

```
const getFullName = (name, surname) => {    lib.mjs
    return name + " " + surname;
}
const getSurName = (name, surname) => surname;

//exports.modules {getFullName, getSurName};
export default {getFullName, getSurName};
```

```
// const lib= require("./modules/lib");
import lib from './modules/lib.mjs'

console.log('getFullName', lib.getFullName('Boris S', 'sauvageb'));
console.log('getSurName', lib.getSurName('Boris S', 'sauvageb'));
```



```
<html>
<head>
<meta charset="UTF-8">
<title>Projet Module ES6</title>
<script src="app.js" type="module"></script>
</head>
<body>
<h1>Exemple</h1>
</body>
</html>
```

The screenshot shows a browser's developer tools console tab. It displays two log entries: 'getFullName Boris S sauvageb' and 'getSurName sauvageb'. Below the console are standard developer tools controls for Elements, Console, and Filter.

ES6 Modules 2/3

Exemple : export avec export maFonction

```
export const getFullName = (name, surname) => {
    return name + " " + surname;
}
export const getSurName = (name, surname) => surname;
```



```
// const {getFullName, getSurName} = require("./modules/lib");
import {getFullName, getSurName} from './modules/lib.mjs'

console.log('getFullName', getFullName('Boris S', 'sauvageb'));
console.log('getSurName', getSurName('Boris S', 'sauvageb'));
```



```
es6-modules
  modules
    lib.mjs
    app.js
    index.html
```

```
<html>
<head>
<meta charset="UTF-8">
<title>Projet Module ES6</title>
<script src="app.js" type="module"></script>
</head>
<body>
<h1>Exemple</h1>
</body>
</html>
```



```
Elements Console
top ▾ Filter
getFullName Boris S sauvageb
getSurName sauvageb
```

ES6 Modules 3/3

- L'utilisation des modules ES6 est mieux que CommonJS
- **require** est une fonction simple qui peut être utilisée dans du code VanillaJS
 - Vérifié au runtime
 - La fonction require peut contenir des paramètres classiques de fonction
- **Import** n'est pas une fonction mais est statique
 - Vérifié par TypeScript durant l'analyse syntaxique
 - La déclaration de l'instruction sera toujours remontée en haut du fichier
 - Les imports peuvent contenir que des chaînes de caractères

Exercice : OOP JavaScript

- Mettre en place les modules ES6 sur le projet Pokémon précédent

REMARQUE : Lors des imports, veillez à bien renseigner le .js que l'IDE a tendance à oublier

ECMAScript : Les nouveautés

TABLEAUX

Les tableaux : initialisation 1/2

- Depuis le début JavaScript, il est possible de déclarer des tableaux :

```
const ex1 = [12, 5, 8, 130, 44];
```

- Avec ES2009, il est possible de déclarer un tableau comme ci-dessous :

```
const ex2 = new Array(12, 13, 14, 15);
```

- Initialisation d'un tableau de 5 éléments :

```
const ex3 = new Array(5);
```

```
MBP-de-Boris:PROJETS sauvegeb$ node /Users/sauvegeb/Desktop/test.js
[ <5 empty items> ]
```

- Utilisation de **Array.of** :

```
const ex4 = Array.of("Pomme", "Pêche", "Cerise");
```

```
MBP-de-Boris:PROJETS sauvegeb$ node /Users/sauvegeb/Desktop/test.js
[ 'Pomme', 'Pêche', 'Cerise' ]
```

Les tableaux : initialisation 2/2

- Utilisation d'une fonction exécutée pour chaque élément :

```
const asArray = {0: "Boris", 1: "Rodolphe", 2: "Maximilien", 3: "Louise", length: 4};

const myArray = Array.from(asArray, x => x + " !");
for (const item of myArray) {
    console.log(item);
}
```

```
MBP-de-Boris:PROJETS sauvageb$ node /Users/sauvageb/Desktop/test.js
Boris!
Rodolphe!
Maximilien!
Louise!
```

ECMAScript : Les tableaux

9 MUTATEURS DES TABLEAUX

Les tableaux : fill

- La méthode **fill** permet de remplir ou remplacer les valeurs de nos tableaux

```
let example = new Array(5);
example.fill(1);
console.log(example);
```

```
MBP-de-Boris:PROJETS sauvageb$ node /Users/sauvageb/Desktop/test.js
[ 1, 1, 1, 1, 1 ]
```

- Il est possible d'utiliser **fill(value, start?, end?)**

```
let example = [1, 2, 3, 4];
example.fill(100, 0, 1);
console.log(example);
```

```
MBP-de-Boris:PROJETS sauvageb$ node /Users/sauvageb/Desktop/test.js
[ 100, 2, 3, 4 ]
```

```
let example = [1, 2, 3, 4];
example.fill(100, 2);
console.log(example);
```

```
MBP-de-Boris:PROJETS sauvageb$ node /Users/sauvageb/Desktop/test.js
[ 1, 2, 100, 100 ]
```

Les tableaux : copyWithin

- La méthode **copyWithin** effectue une copie superficielle d'une partie du tableau
 - Fait la copie sur le tableau courant et le renvoie, sans modifier la taille

```
const startArray = ["Boris", "Rodolphe", "Maximilien", "Louise"];
console.log(startArray);

// copie l'element de l'index 0 ("Boris") dans la dernière position du tableau ("Louise")
const finalArray = startArray.copyWithin(startArray.length - 1, 0, 1);
console.log(finalArray);
```

```
[ 'Boris', 'Rodolphe', 'Maximilien', 'Louise' ]
[ 'Boris', 'Rodolphe', 'Maximilien', 'Boris' ]
```

Les tableaux : push, unshift, shift, pop

- Les fonctions **push()** et **unshift()** permettent d'ajouter des éléments au tableau
 - **push()** ajoute en fin de tableau
 - **unshift()** ajoute au début

```
let v = ["Ferrari", "Porsche", "VW"];
v.push("Renault");
```

► (4) ["Ferrari", "Porsche", "VW", "Renault"]

```
let v = ["Ferrari", "Porsche", "VW", "Renault"];
v.unshift("Fiat");
```

► (5) ["Fiat", "Ferrari", "Porsche", "VW", "Renault"]

- Les fonctions **pop()** et **shift()** permettent de retirer un élément du tableau
 - **pop()** supprime le dernier élément du tableau
 - **shift()** retire le premier du tableau

```
let v = ["Ferrari", "Porsche", "VW", "Renault", "Fiat"];
v.pop();
```

► (4) ["Ferrari", "Porsche", "VW", "Renault"]

```
let v = ["Ferrari", "Porsche", "VW", "Renault", "Fiat"];
v.shift();
```

► (4) ["Porsche", "VW", "Renault", "Fiat"]

Les tableaux : reverse

- La méthode **reverse** transpose les éléments d'un tableau
 - Le premier élément devient le dernier et le dernier devient le premier
- **Utilisation de la fonction reverse()**

```
const example = ["Boris", "Rodolphe", "Maximilien", "Louise"];
console.log(example);

example.reverse();
console.log(example);
```

['Boris', 'Rodolphe', 'Maximilien', 'Louise']
['Louise', 'Maximilien', 'Rodolphe', 'Boris']

Les tableaux : sort

- La méthode **sort** trie les éléments d'un tableau
 - Le tri s'effectue selon les unités de code UTF-16 des caractères
- **Utilisation de la fonction sort()**

```
const example = ["Boris", "Rodolphe", "Maximilien", "Louise"];
console.log(example);
example.sort();
console.log(example);
```

['Boris', 'Rodolphe', 'Maximilien', 'Louise']
['Boris', 'Louise', 'Maximilien', 'Rodolphe']

- **Utilisation d'une fonction de comparaison avec sort()**

```
example.sort((a, b) => a.length - b.length);
console.log(example);
```

['Boris', 'Rodolphe', 'Maximilien', 'Louise']
['Boris', 'Louise', 'Rodolphe', 'Maximilien']

Les tableaux : splice

- La méthode **splice** permet de vider ou remplacer une partie d'un tableau
 - Le tri s'effectue selon les unités de code UTF-16 des caractères
- **Utilisation de la fonction splice(début, nombre à supprimer , tableau)**

```
const example = ["Boris SAUVAGE", "Rodolphe", "Maximilien"];
console.log(example);

example.splice(0, 1, "Boris");
console.log(example);
```

['Boris Sauvage', 'Rodolphe', 'Maximilien']
['Boris', 'Rodolphe', 'Maximilien']

```
example.splice(example.length, 1, "Louise");
console.log(example);
```

['Boris', 'Rodolphe', 'Maximilien']
['Boris', 'Rodolphe', 'Maximilien', 'Louise']

ECMAScript : Les tableaux

8 ACCESSEURS DES TABLEAUX

Les tableaux : concat

- La méthode **concat** est utilisée afin de fusionner un ou plusieurs tableaux
 - Renvoie un nouveau tableau qui est le résultat de la concaténation
- **Utilisation de la fonction concat()**

```
const example1 = ["Boris", "Rodolphe"];
const example2 = ["Maximilien", "Louise"];
console.log(example1);
console.log(example2);

const result = example1.concat(example2);
console.log(result);
```

```
[ 'Boris', 'Rodolphe' ]
[ 'Maximilien', 'Louise' ]
[ 'Boris', 'Rodolphe', 'Maximilien', 'Louise' ]
```

Les tableaux : includes

- La méthode **includes** permet de déterminer si un tableau contient une valeur
 - Renvoie un booléen
- **Utilisation de la fonction includes()**

```
const example1 = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example1);
```

```
const includeBoris = example1.includes("Boris");
console.log(includeBoris);
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
true
```

Les tableaux : join

- La méthode **join** concatène les éléments d'un tableau avec l'élément de votre choix
 - Renvoie une nouvelle chaîne de caractères
- **Utilisation de la fonction join(séparateur)**

```
const example1 = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example1);
```

```
const result = example1.join(" et ");
console.log(result);
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
Louise et Boris et Maximilien et Rodolphe
```

Les tableaux : slice

- La méthode **slice** renvoie un tableau contenant une copie superficielle d'une portion du tableau d'origine
 - La portion est définie par un indice de début et un indice de fin (exclus)
- **Utilisation de la fonction slice(début, fin)**

```
const people = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(people);
```

```
let result1 = people.slice(1, people.length);
console.log(result1);
```

```
result2 = result1.slice(0, 1);
console.log(result2);
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
[ 'Boris', 'Maximilien', 'Rodolphe' ]
[ 'Boris' ]
```

Les tableaux : `toString`

- La méthode **`toString`** renvoie une chaîne de caractères représentant le tableau spécifié et ses éléments
- **Utilisation de la fonction `toString()`**

```
const people = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(people);
```

```
let peopleStr = people.toString();
console.log(peopleStr);
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
Louise,Boris,Maximilien,Rodolphe
```

Les tableaux : `toLocaleString`

- La méthode **`toLocaleString`** renvoie une chaîne de caractères représentant le tableau spécifié et ses éléments, utilisant la méthode **`toLocaleString`** pour chacun d'eux
- **Utilisation de la fonction `toLocaleString()`**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe", new Date()];
console.log(example);
```

```
let result = example.toLocaleString();
console.log(result);
```

```
[  
  'Louise',  
  'Boris',  
  'Maximilien',  
  'Rodolphe',  
  2020-11-14T21:42:40.168Z  
]  
Louise,Boris,Maximilien,Rodolphe,11/14/2020, 10:42:40 PM
```

Les tableaux : indexOf

- La méthode **indexOf** renvoie le premier indice pour lequel on trouve un élément donné dans un tableau
 - Retourne -1 si l'élément cherché n'est pas présent dans le tableau
- **Utilisation de la fonction indexOf()**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

let result = example.indexOf("Boris");
console.log(result);
```

['Louise', 'Boris', 'Maximilien', 'Rodolphe']
1

Les tableaux : lastIndexOf

- La méthode **lastIndexOf** renvoie le dernier indice pour lequel on trouve une valeur donnée est présente dans un tableau
 - Retourne -1 si la valeur cherchée n'est pas présente dans le tableau
- **Utilisation de la fonction lastIndexOf()**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe", "Boris"];
console.log(example);
```

```
let result = example.lastIndexOf("Boris");
console.log(result);
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe', 'Boris' ]
4
```

ECMAScript : Les tableaux

12 MÉTHODES D'ITÉRATION

Les tableaux : entries

- La méthode **entries** renvoie un nouvel objet de type **Array Iterator**
 - Contient la clef/valeur pour chaque éléments du tableau
- **Utilisation de la fonction entries()**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const iterator = example.entries();

let value = iterator.next().value;
console.log(value);

value = iterator.next().value;
console.log(value);
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
[ 0, 'Louise' ]
[ 1, 'Boris' ]
```

Les tableaux : every

- La méthode **every** permet de tester si tous les éléments d'un tableau vérifient une condition donnée
- **Utilisation de la fonction every()**

```
const example = ["Louise", "Boris", "Rodolphe"];
console.log(example);

const test1 = example.every((x) => x.includes("i"));
console.log(test1);

const test2 = example.every((x) => x.includes("o"));
console.log(test2);
```

```
[ 'Louise', 'Boris', 'Rodolphe' ]
false
true
```

Les tableaux : filter

- La méthode **filter** retourne un nouveau tableau contenant tous les éléments du tableau d'origine qui remplissent une condition déterminée
- **Utilisation de la fonction filter(fonction de test)**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const result = example.filter(x => !x.includes("o"));
console.log(result);
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
[ 'Maximilien' ]
```

Les tableaux : find

- La méthode **find** renvoie la valeur du **premier élément** trouvé dans le tableau qui respecte la condition donnée par la fonction de test
 - Sans résultat trouvé, **undefined** est renvoyé
- **Utilisation de la fonction find(fonction de test)**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const test1 = example.find(x => x.includes("r"));
console.log(test1);
```

['Louise', 'Boris', 'Maximilien', 'Rodolphe']
Boris

Les tableaux : `findIndex`

- La méthode **`findIndex`** renvoie l'indice du premier élément du tableau qui respecte une condition donnée par la fonction de test
 - Sans résultat trouvé, **-1** est renvoyé
- Utilisation de la fonction `findIndex(fonction de test)`

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);
```

```
const test1 = example.findIndex(x => x.includes("r"));
console.log(test1);
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
1
```

Les tableaux : map

- La méthode **map** crée un nouveau tableau avec les résultats de l'appel d'une fonction fournie sur chaque élément du tableau appelant
- **Utilisation de la fonction map(fonction)**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const test1 = example.map(x => x.concat(" !"));
console.log(test1);
```

['Louise', 'Boris', 'Maximilien', 'Rodolphe']
['Louise !', 'Boris !', 'Maximilien !', 'Rodolphe !']

Les tableaux : forEach

- La méthode **forEach** permet d'exécuter une fonction donnée sur chaque élément du tableau
- **Utilisation de la fonction forEach(fonction)**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);
```

```
example.forEach(x => console.log(x));
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
```

```
Louise
```

```
Boris
```

```
Maximilien
```

```
Rodolphe
```

Les tableaux : some

- La méthode **some** teste si au moins un élément du tableau passe le test implémenté par la fonction fournie. Elle renvoie un **booléan** indiquant le résultat du test
- **Utilisation de la fonction some(fonction)**
 - Ci-dessous, retourne true si un élément a une taille de 10

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const test = example.some(x => x.length == 10);
console.log(test);
```

['Louise', 'Boris', 'Maximilien', 'Rodolphe']
true

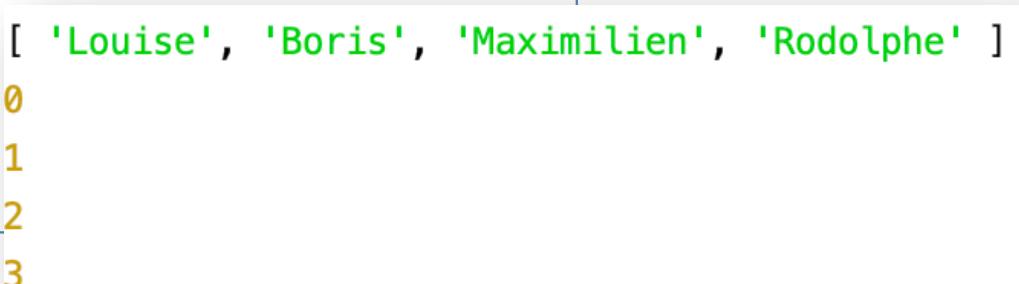
Les tableaux : keys

- La méthode **keys** renvoie un nouvel objet de type **Array Iterator**
 - Contient les clefs pour chaque indice du tableau
- **Utilisation de la fonction keys()**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const iterator = example.keys();

for (const key of iterator) {
  console.log(key);
}
```



```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
0
1
2
3
```

Les tableaux : values

- La méthode **values** renvoie un nouvel objet de type **Array Iterator**
 - Contient les valeurs pour chaque indice du tableau
- **Utilisation de la fonction values()**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const iterator = example.values();

for (const value of iterator) {
  console.log(value);
}
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
Louise
Boris
Maximilien
Rodolphe
```

Les tableaux : reduce

- La méthode **reduce** applique une fonction qui est un accumulateur
 - Traite chaque valeur de la liste afin de la réduire à une seule valeur
- **Utilisation de la fonction reduce()**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const result = example.reduce((accumulator, currentValue) => {
  console.log("accumulator =" + accumulator + ", value =" + currentValue);
  return accumulator + currentValue;
});
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
accumulator =Louise, value = Boris
accumulator =LouiseBoris, value = Maximilien
accumulator =LouiseBorisMaximilien, value = Rodolphe
```

Les tableaux : reduceRight

- La méthode **reduceRight** applique **reduce** de droite à gauche
 - Traite chaque valeur de la liste afin de la réduire à une seule valeur
- **Utilisation de la fonction reduceRight()**

```
const example = ["Louise", "Boris", "Maximilien", "Rodolphe"];
console.log(example);

const result = example.reduce((accumulator, currentValue) => {
  console.log("accumulator =" + accumulator + ", value =" + currentValue);
  return accumulator + currentValue;
});
```

```
[ 'Louise', 'Boris', 'Maximilien', 'Rodolphe' ]
accumulator =Rodolphe, value = Maximilien
accumulator =RodolpheMaximilien, value = Boris
accumulator =RodolpheMaximilienBoris, value = Louise
```

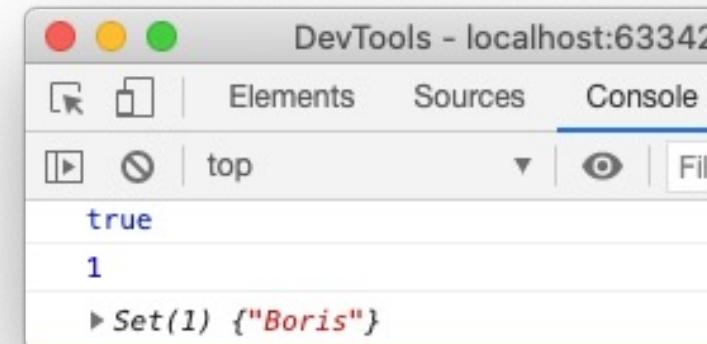
ECMAScript : Les Set et Map

LES SET ET LES MAP

Les Sets

- L'objet **Set** permet de stocker des valeurs *uniques*, de n'importe quel type, que ce soit des valeurs d'un type primitif ou des objets.
- **Utilisation de l'objet Set**

```
const set1 = new Set(["Louise", "Boris", "Maximilien", "Rodolphe"]);
console.log(set1.has("Louise"));
set1.clear();
set1.add("Louise");
set1.add("Louise");
set1.add("Boris");
set1.delete("Louise");
console.log(set1.size);
console.log(set1);
```



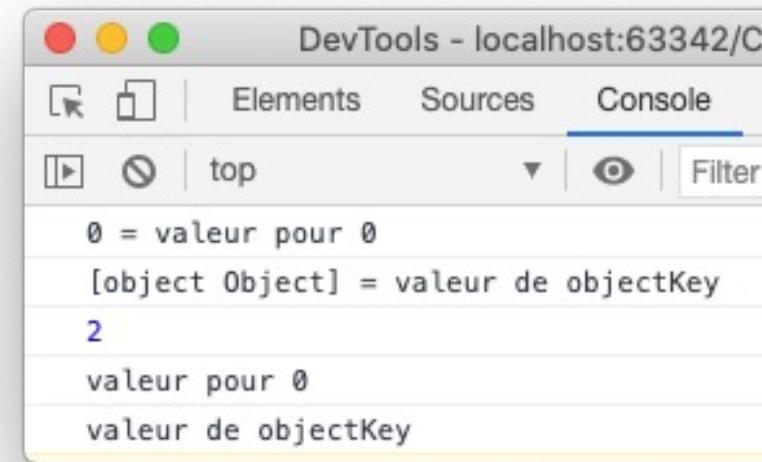
Les Map

- L'objet **Map** représente un dictionnaire (une carte de clés/valeurs)
 - N'importe quelle valeur valable en JavaScript peut être utilisée comme clé ou comme valeur
 - L'ordre d'insertion des clés est mémorisé et les boucles sur les Map parcourront les clés dans cet ordre
- **Utilisation de l'objet Map**

```
const myMap = new Map();
const objectKey = {};
myMap.set(0, "valeur pour 0");
myMap.set(objectKey, "valeur de objectKey");

for (const [key, value] of myMap.entries()) {
  console.log(`[${key}] = ${value}`);
}

console.log(myMap.size);
console.log(myMap.get(0));
console.log(myMap.get(objectKey));
```



Exercice 1/3

À partir du tableau suivant, effectuez les traitements ci-dessous :

```
["william", "jones", "aaron", "seppe", "frank", "gilliam"]
```

- Transformer les valeurs du tableau en majuscules
- À partir du tableau, créez un tableau contenant les noms de plus de 5 lettres

Exercice 2/3

À partir du code suivant, effectuez les traitements ci-dessous :

```
class Person {  
    constructor(name, age){  
        this.name = name;  
        this.age = age;  
    }  
}
```

```
let persons = [  
    new Person('Sarah', 4),  
    new Person('Eva', 32),  
    new Person('Victor', 40),  
    new Person('Arthur', 29),  
    new Person('Maeva', 55),  
    new Person('Alix', 25),  
];
```

Toujours en partant du tableau initial

- Afficher le tableau dans l'ordre décroissant en fonction de l'âge
- Afficher un nouveau tableau ne contenant pas les personnes entre 18 et 26 ans
- Affichez un nouveau tableau contenant uniquement les âges des personnes
- Créez un nouveau tableau contenant les noms des personnes de plus de 30 ans
- Calculez la somme des âges de toutes les personnes du tableau initial

ECMAScript

Exercice 3/3

- Pour aller plus loin :

<https://github.com/kentcdodds/es6-workshop>

Autres ressources...

- ECMAScript 5 (2009) :
 - https://www.w3schools.com/js/js_es5.asp
- ECMAScript 6 (2015) :
 - https://www.w3schools.com/js/js_es6.asp

Le Support ECMAScript Next

- ECMAScript Next fait référence au fonctionnalités ajoutées au standard ECMA-262 depuis ES2015
- <https://kangax.github.io/compat-table/esnext/>
- https://developer.mozilla.org/fr/docs/Web/JavaScript/Language_Resources
- <https://caniuse.com/es6>

Transpileur JavaScript

- Permet de transformer un code ES6/ES7/ES8 vers une version précédente
- Certaines syntaxe ES+ ne sont pas encore supportée par tous les navigateurs

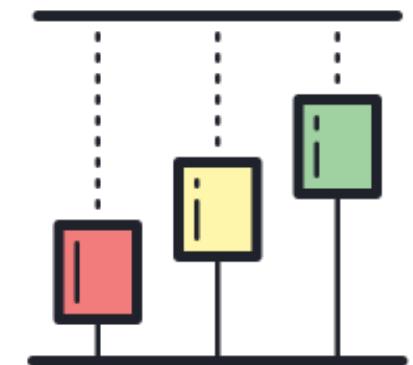
Exemple avec Babel.js :



- Babel.js permet de profiter des standards à venir (ES7, ES8)

Développement Web

CONTEXTE ASYNCHRONE

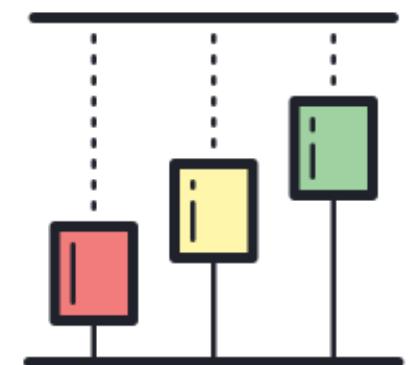


Les promesses, `async` et `await`

- Dans un contexte asynchrone, il est important de connaître les notions suivantes :
- Promesses :
 - https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise
- Async :
 - https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/async_function
- Await :
 - <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/await>

Promise 1/5

- **Promise** est utilisé pour réaliser des traitements de façon asynchrone
- Représente une valeur qui peut être disponible maintenant, dans le futur voire jamais
- Une Promise est dans un de ces états :
 - pending (en attente) : état initial, la promesse n'est ni remplie, ni rompue
 - fulfilled (tenue) : l'opération a réussi
 - rejected (rompue) : l'opération a échoué



Promise 2/5

- Représente la complétion ou l'échec d'une opération asynchrone
 - on « consomme » des promesses
- Elle remplace l'utilisation des fonctions avec des callbacks en paramètres

```
function myOperation(successCallback, failureCallback) {  
    console.log("Traitement en cours ....");  
  
    if (Math.random() > .5) {  
        successCallback("Réussite");  
    } else {  
        failureCallback("Échec");  
    }  
}  
  
myOperation(successCallback, failureCallback);
```

```
function successCallback(resultat) {  
    console.log("Opération réussi : " + résultat);  
}  
  
function failureCallback(erreur) {  
    console.error("Opération échoué : " + erreur);  
}
```

Promise 3/5

- Désormais, nous aurons le code suivant :
 - une fonction qui renvoie une promesse
 - Les callbacks attachées à cette promesse

```
function myOperation() {  
    return new Promise((successCallback, failureCallback) => {  
        console.log("Traitement en cours ....");  
  
        if (Math.random() > .5) {  
            successCallback("Réussite");  
        } else {  
            failureCallback("Échec");  
        }  
    })  
}  
  
myOperation().then(successCallback, failureCallback);
```

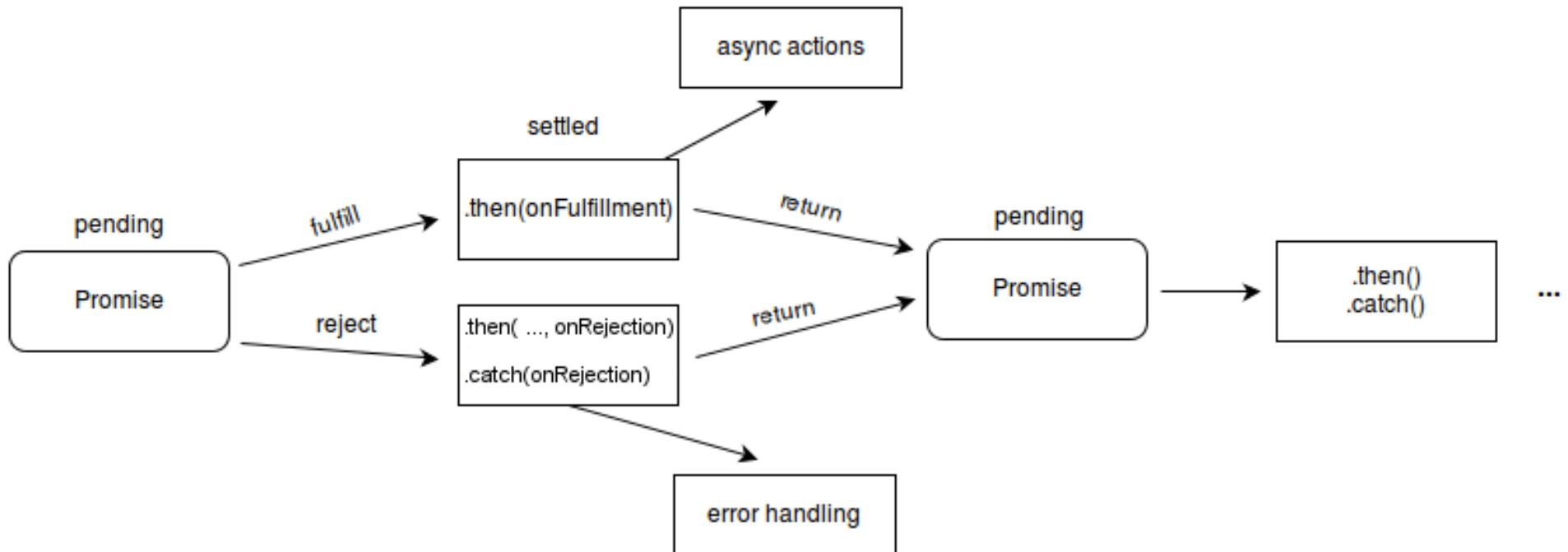
```
function successCallback(resultat) {  
    console.log("Opération réussi : " + resultat);  
}  
  
function failureCallback(erreur) {  
    console.error("Opération échoué : " + erreur);  
}
```

Promise 4/5

- Il est possible d'exécuter plusieurs opérations les unes à la suite des autres
 - Possible en utilisant une chaîne de promesses

```
myOperation()  
  .then(result => myOperation2(result))  
  .then(newResult => myOperation3(newResult))  
  .then(finalResult => {  
    console.log('Résultat final : ' + finalResult);  
  })  
  .catch(failureCallback);
```

Promise 5/5



Async function

- Définit une fonction asynchrone qui renvoie un objet **AsyncFunction**
- Une fonction **async** peut contenir **await** qui interrompt l'exécution de la fonction
 - Attend la résolution de la promesse, puis reprend, puis renvoie la valeur

```
function resolveAfter2Seconds() {  
  
    return new Promise(resolve => {  
        setTimeout(() => {  
            resolve('resolved');  
        }, 2000);  
    });  
  
}
```

```
async function asyncCall() {  
  
    console.log('calling');  
  
    const result = await resolveAfter2Seconds();  
  
    console.log(result);  
    // expected output: "resolved"  
}  
  
asyncCall();
```

Await function

- Permet d'attendre la résolution d'une promesse
 - Utilisable uniquement au sein d'une fonction asynchrone

```
function resolveAfter2Seconds() {  
  
    return new Promise(resolve => {  
        setTimeout(() => {  
  
            resolve('resolved');  
  
        }, 2000);  
    });  
  
}
```

```
async function asyncCall() {  
  
    console.log('calling');  
  
    const result = await resolveAfter2Seconds();  
  
    console.log(result);  
    // expected output: "resolved"  
}  
  
asyncCall();
```

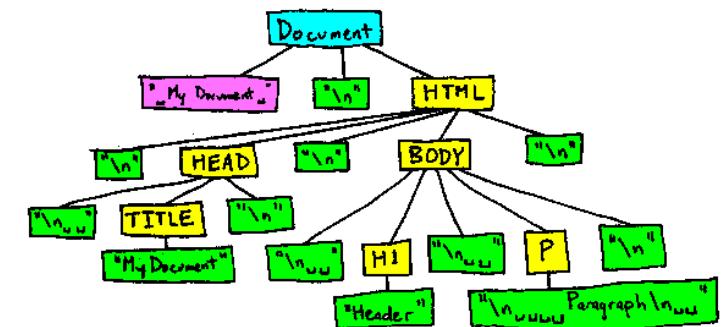
Exemple

- Utilisation d'une fonction asynchrone

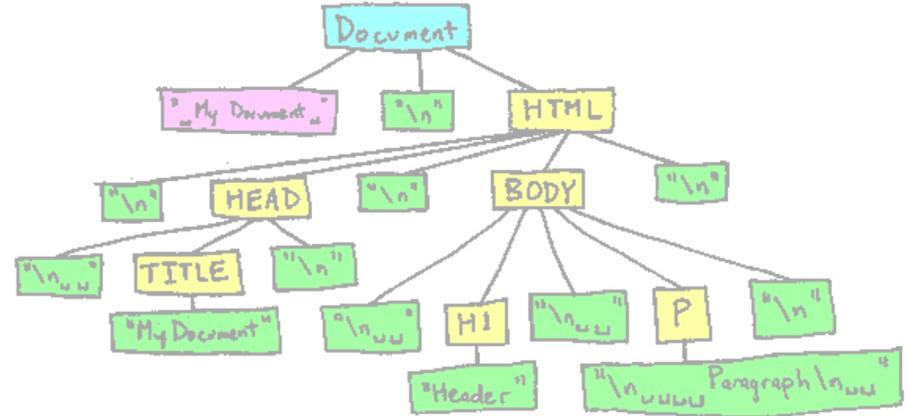
```
async function fetchCars(){  
  
    const response = await fetch('http://localhost:3000/cars');  
    const cars = await response.json();  
    return cars;  
}  
  
fetchCars().then(cars => {  
    console.log(cars);  
});
```

Développement Web

JAVASCRIPT MODERNE : DOM, ÉVÉNEMENTS



JavaScript : le DOM



- **Le DOM** (Document Object Model) est l'API du navigateur qui permet de manipuler une page web
- Un document HTML est semi-structuré sous forme d'arborescence
 - Chaque nœud est parent ou enfant d'un autre nœud
- Le nœud racine est HTML
- La manipulation du DOM avec JavaScript se fait avec l'interface **document**

JavaScript : API Selectors

- Définit des méthodes pour accéder aux nœuds DOM (éléments HTML présents)
- Facilitent la manipulation des nœuds en JavaScript
- Plus puissant et souple que les fonctions des premières versions de l'interface DOM :
 - **getElementById()** : recherche un élément d'après son identifiant (attribut id),
 - **getElementsByName()** : recherche des éléments d'après leur type (balise),
 - **getElementsByClassName()** : recherche des éléments d'après leur classe (attribut class)
- Ces **3 méthodes sont** performantes mais **limitées**
 - Avec des pages ou applications complexes, elles se révèlent peu souples car nécessitent de faire appel à des boucles, diverses opérations algorithmiques et des filtres sur les résultats obtenus pour pouvoir obtenir une liste d'éléments

JavaScript : API Selectors

- **2 nouvelles fonctions introduites par l'API Selectors**
- Syntaxe qui fait appel aux **sélecteurs CSS**
 - On applique en général ces deux méthodes à partir de la racine document
- **querySelector()**
 - retourne le 1^{er} élément trouvé satisfaisant au sélecteur (type de retour : Element), ou null si aucun objet correspondant n'est trouvé
- **querySelectorAll()**
 - retourne tous les éléments satisfaisant au sélecteur, dans l'ordre dans lequel ils apparaissent dans l'arbre du document (type de retour : NodeList), ou un tableau NodeList vide si rien n'est trouvé

JavaScript : Comparaison

- Recherche de toutes les cellules <td> contenues dans un élément <table id="mytable">

/ Version DOM native

```
var table = document.getElementById("mytable");
var tbodies = table.tBodies;
var rangees = null;
var cellules = [];

for (var i = 0; i < tbodies.length; i++) {
    rangees = tbodies[i].rows;
    for (var j = 0; j < rangees.length; j++) {
        cellules.push(rangees[j].cells[1]);
    }
}
```

versus

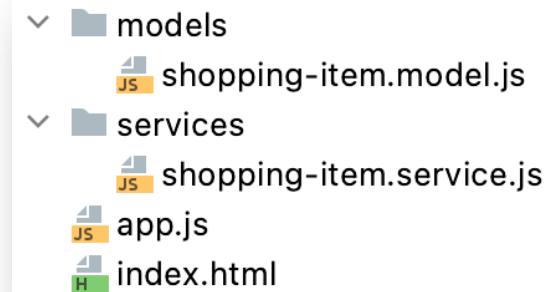
// Version API Selectors

```
let cellules =
document.querySelectorAll('#mytable td');
```

Exercice

- Faire les exercices suivants :
 - Mettre en place une structure modulaire ES6
 - Définir un CRUD dans un Service
 - Il utilisera un tableau et les fonctionnalités vues précédemment

```
class ShoppingItemService {  
  
    getItems = () => {...}  
  
    createItem = (name) => {...}  
  
    deleteItem = ...  
  
    updateItem = ...  
  
}  
  
export {ShoppingItemService};
```



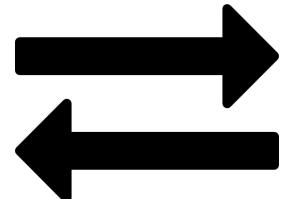
My shopping list

Enter a new item : Add

- Sugar delete
- Chocolate delete
- Cranberries delete

Développement Web

AJAX, PROMISES, FETCH, AXIOS, ASYNC, AWAIT

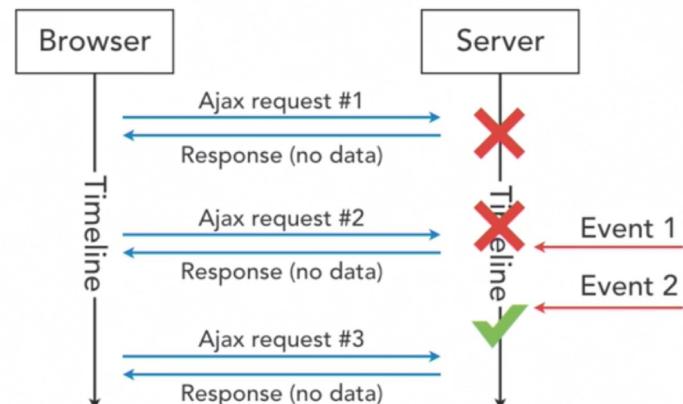


Historique

- **Avant 2001** : requêtes HTTP synchrones
 - Attentes des utilisateurs longue
- **2001** : requêtes HTTP asynchrones avec XMLHttpRequest (XHR)
- **2005** : Les Framework AJAX mettent en avant XHR (Prototype, Dojo, Jquery)
 - <https://www.chandlerproject.org/ajaxlibraries/>
 - Attentes des utilisateurs plus courtes
- **2011** : HTML5 : Cross-Document Messaging, Server-Sent Events, **WebSocket**
 - moins d'attente et plus d'interactions possibles
- Les changements technologiques ont pour but d'améliorer l'expérience des utilisateurs et le temps d'attente

Historique : XHR vers Fetch

- Depuis longtemps, les développeurs s'appuient sur le protocole asynchrone XHR pour créer des interfaces utilisateur beaucoup plus réactives
- XHR peut rendre le développement d'applications web complexe
 - Interrogation délicates pour envoyer des données entre le navigateur et le serveur
 - Chaque requête doit envoyer la demande, l'en-tête HTTP, entre le navigateur et le serveur
 - Demandes souvent réitérées lorsqu'une réponse souhaitée n'est pas reçue dans les délais
- Apparition de l'API Fetch
 - Meilleur alternative à XHR utilisant des Promesses
 - Moyen facile et logique de récupérer des ressources à travers le réseau de manière asynchrone

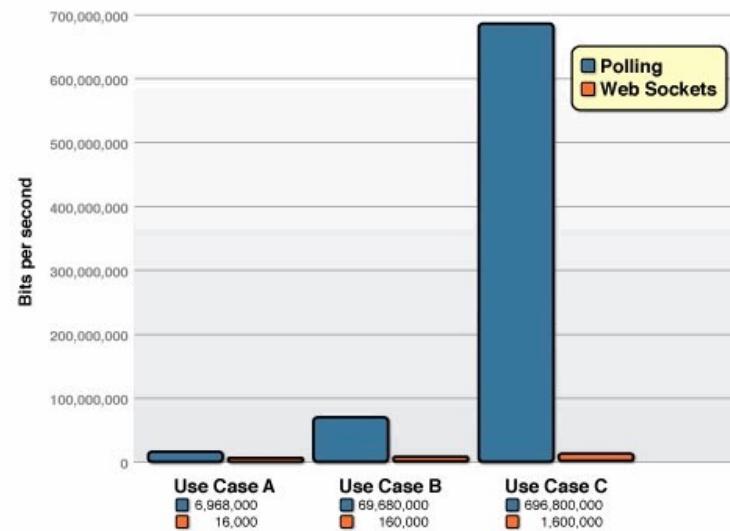
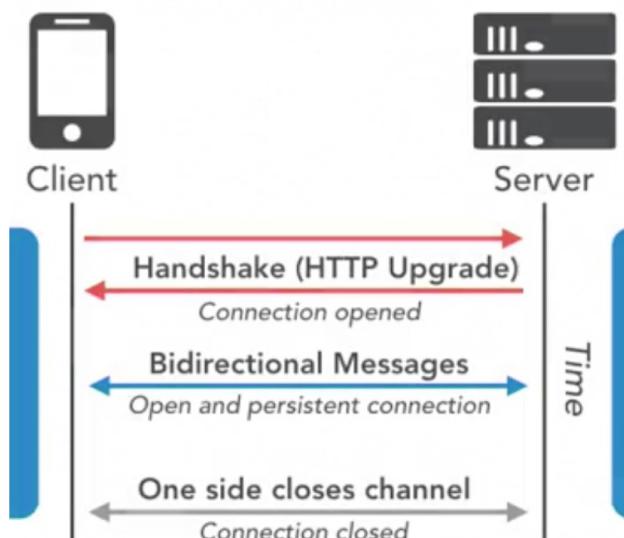


Historique : vers les WebSockets

- Avec l'introduction des WebSockets, on obtient une connexion bidirectionnelle
 - Les navigateurs web peuvent désormais établir une connexion permanente
 - Le navigateur web peut envoyer des messages au serveur web
 - Le serveur web peut initier une conversation et envoyer des messages au navigateur web
- L'ordre des messages n'est pas garanti
 - Protocole non séquentiel
 - contrairement à XHR qui garanti l'ordre

Historique : vers les WebSockets

- HTTP et WS ont une taille de connexion similaire lors de l'initialisation (handshake)
- Pour les communications subséquentes, un message WebSocket contient un surcoût d'entête (**overhead**) minimal de deux octets
 - Avec HTTP, un surcoût largement supérieur est maintenu tout au long de la communication



<http://www.websocket.org/quantum.html>

Historique : vers les WebSockets

- **WebSocket** définit un sous-protocol standard d'application pour toute la durée de la communication
 - concernent davantage les règles de communication entre le navigateur et le serveur
 - peut être utilisé comme transport pour des protocoles application de haut niveau largement normalisés
- **Exemples de protocoles : XMPP, STOMP, AMQ, RFB, WAMP, MBWS**
 - <https://www.iana.org/assignments/websocket/websocket.xml>
- **Focus sur STOMB (Simple Text-Oriented Messaging Protocol)**
 - Protocole de messagerie léger permettant de créer un code client/serveur adapté
 - Fournit la sémantique de la messagerie et définit les types de trame (mappés avec les WebSockets)
 - Permet d'effectuer des tâches simples

STOMP Methods
connect()
subscribe()
send()
onmessage()

Premiers pas avec XMLHttpRequest

- XMLHttpRequest permet d'interagir avec des serveurs
 - On peut récupérer des données à partir d'une URL sans avoir à rafraîchir complètement la page
 - Cela permet à une page web d'être mise à jour sans perturber les actions de l'utilisateur
 - XMLHttpRequest est beaucoup utilisé par l'approche **AJAX**

Exemple :

```
shoppingItemService.getItems(displayItems, displayError);
```

My Shopping list

Enter a new item :

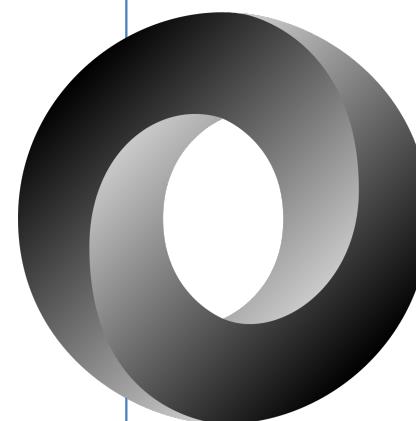
- Oeufs
- Sel
- Farine

```
getItems(onSuccess, onError) {  
    const xhr = new XMLHttpRequest();  
    xhr.open('GET', 'http://localhost:3000/items', true);  
    xhr.send();  
    xhr.onreadystatechange = () => {  
        if (xhr.readyState == xhr.DONE) {  
            if (xhr.status == 200) {  
                this.items = JSON.parse(xhr.response);  
                onSuccess(this.items);  
            } else {  
                onError();  
            }  
        }  
    };  
};
```

Le JSON

- <https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON>

```
{  
  "squadName": "Super hero squad",  
  "formed": 2016,  
  "active": true,  
  "members": [  
    {  
      "name": "Molecule Man",  
      "age": 29,  
      "powers": [ "Turning tiny", "Radiation blast"]  
    },  
    {  
      "name": "Madame Uppercut",  
      "age": 39,  
      "powers": [ "Superhuman reflexes »"]  
    }  
  ]  
}
```



...Vers les Promises

- Objet qui représente la complétion ou l'échec d'une opération asynchrone
 - La plupart du temps, on « consomme » des promesses
- Une promesse est un objet qui est renvoyé et auquel on attache des *callbacks*
 - plutôt que de passer des *callbacks* à une fonction

VERSUS

```
const successCallback = (résultat) => console.log("Succès : " + résultat);
const failureCallback = (erreur) => console.error("Echec : " + erreur);

const operationALancienne = (onSuccess, onFailure) => {
  console.log("C'est fait");
  // réussir une fois sur deux
  Math.random() > .5 ? onSuccess("Réussite") : onFailure("Échec");
}

operationALancienne(successCallback, failureCallback);
```

```
const operation = ()=> {

  return new Promise((onSuccess, onFailure) => {
    console.log("C'est fait");
    // réussir une fois sur deux
    Math.random() > .5 ? onSuccess("Réussite") : onFailure("Échec");
  })
}

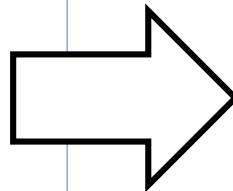
const promise = operation();
operation.then(successCallback, failureCallback);
```

AJAX Nouvelle génération et Promises

Fetch API

- Fournit une interface pour la récupération de ressources à travers le réseau
 - Familière à tout utilisateur de XMLHttpRequest
 - Nouvelle API proposant un ensemble de fonctionnalités plus souples et plus puissantes

```
shoppingItemService.createItem(itemInput.value)
  .then(itemAdded => {
    displayItem(itemAdded);
    itemInput.value = "";
    itemInput.focus();
  }).catch(error => {
    alert("error");
})
```



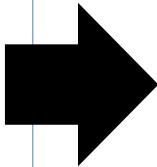
```
createItem = (name) => {
  let item = new ShoppingItem(++this.itemCount, name);

  return fetch('http://localhost:3000/items', {
    method: 'post',
    headers: { 'Content-type': 'application/json' },
    body: JSON.stringify(item)
  }).then(res => {
    if (res.ok) {
      this.items = [...this.items, item];
      return item;
    }
    throw new Error(res.type);
  }).catch(e => Promise.reject(e));
}
```

Async & Await

- Mise en place des mots clés `async` et `await`
 - **Await** : Indique à une fonction de retourner une promesse
 - **Async** : Indique de mettre en pause le code pour attendre le résultat de la promesse

```
fetch('coffee.jpg')
  .then(response => {
    if (!response.ok) {
      throw new Error(`Erreur HTTP : ${response.status}`);
    }
    return response.blob();
  })
  .then(myBlob => {
    let objectURL = URL.createObjectURL(myBlob);
    let image = document.createElement('img');
    image.src = objectURL;
    document.body.appendChild(image);
  })
  .catch(e => {
    console.log('Problème avec la récupération : ' + e.message);
 });
});
```



```
async function myFetch() {
  let response = await fetch('coffee.jpg');
  if (!response.ok) {
    throw new Error(`Erreur HTTP : ${response.status}`);
  }
  return await response.blob();
}
```



```
myFetch().then((blob) => {
  let objectURL = URL.createObjectURL(blob);
  let image = document.createElement('img');
  image.src = objectURL;
  document.body.appendChild(image);
}).catch(e => console.log(e));
```

AJAX Nouvelle génération et Promises

Exercice 1/2

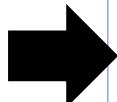
Faire les exercices suivants :

1. Mettre en place un CRUD qui utilisera json-server et l'API Fetch
 - <https://github.com/typicode/json-server>
 - Les opérations CRUD devront effectuer des requêtes HTTP GET, POST, DELETE...

REMARQUE :

- pour résoudre l'erreur du CORS, vous devrez utiliser une entête spécifique
 - <https://developer.mozilla.org/fr/docs/Web/HTTP/CORS>

```
headers: {  
  'Accept': 'application/json',  
  'Content-type': 'application/json',  
  'Access-Control-Allow-Origin': '*'  
},
```



Exercice 2/2

Améliorations :

1. A l'aide du code source initial, mettre en place une structure avec Bootstrap
2. Le site devra afficher au minimum des produits avec les caractéristiques suivantes :
 - nom
 - prix
 - type
 - image du produit
3. Une recherche de produit par nom est possible
4. Un tri par prix est également possible

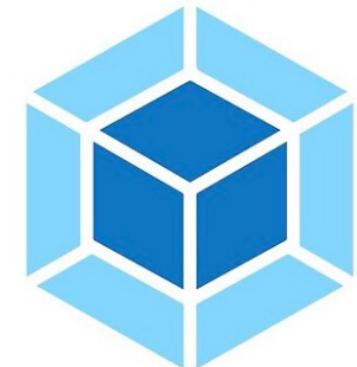
Bonus : la librairie Axios

- Il est possible d'utiliser l'API Fetch intégrée aux navigateurs moderne
 - <https://caniuse.com/fetch>
- Il existe également Axios, une librairie basée sur les Promises
 - <https://github.com/axios/axios>

```
const axios = require('axios');
// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    console.log(response);      // handle success
  })
  .catch(function (error) {
    console.log(error);        // handle error
  })
  .then(function () { // always executed })
```

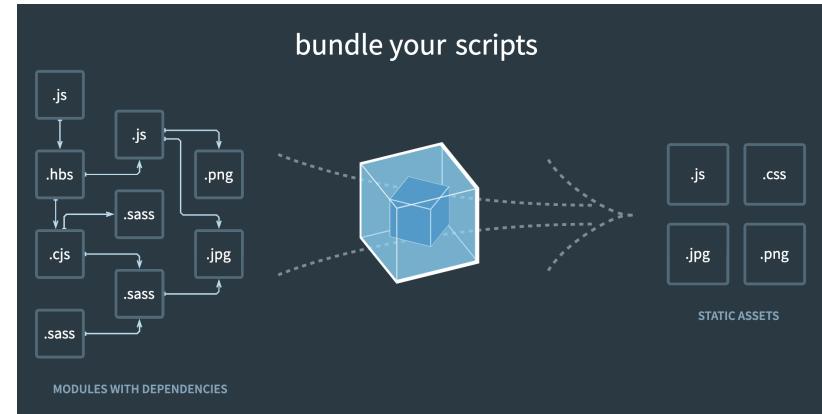
Développement Web

WEBPACK



Automatisation Webpack

- Modules Bundler open source
 - <https://webpack.js.org/>
- Permet de regrouper des fichiers JavaScript pour les utiliser dans un navigateur
 - capable de transformer, regrouper ou empaqueter à peu près n'importe quelle ressource
- Il existe de nombreux Modules Bundler sur le marché :
 - Rollup (<https://github.com/rollup/rollup>)
 - Browserify (<https://browserify.org/>)
 - Parcel (<https://v2.parceljs.org/>)
 - Esbuild (<https://github.com/evanw/esbuild/>)
 - Microbundle (<https://github.com/developit/microbundle>)



Mettre en place Webpack 1/4

- <https://webpack.js.org/guides/getting-started/>

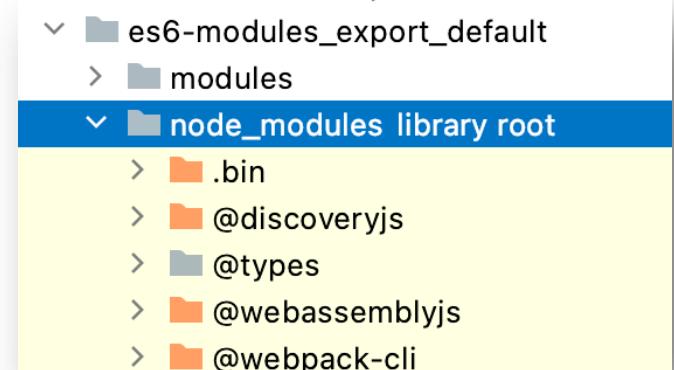
- Initialisation d'un projet avec NPM :
 - Création automatique d'un package.json

```
npm init -y
```



- Installation du module webpack :
 - Création automatique du dossier node_modules

```
npm install webpack webpack-cli --save-dev
```



Mettre en place Webpack 2/4

- Définition de la visibilité du paquet npm en privé pour éviter les publications accidentielles

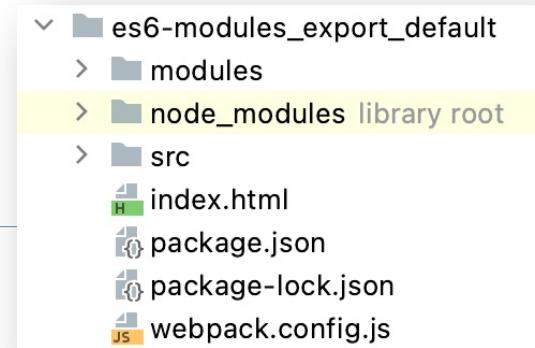
```
{  
  "name": "es6-modules_export_default",  
  "version": "1.0.0",  
  "description": "",  
  "private": true,  
  "main": "src/app.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "webpack": "^5.43.0",  
    "webpack-cli": "^4.7.2"  
  }  
}
```



package.json

Mettre en place Webpack 3/4

- Création d'un dossier `/src` qui contiendra nos sources



- Mise en place de la configuration Webpack :

```
const path = require('path');

module.exports = {
  entry: {
    app: './src/app.js',
  },
  output: {
    filename: '[name].bundle.js',          /* Génération d'un fichier app.bundle.js, défini par la clé de entry */
    path: path.resolve(__dirname, 'dist'),
    clean: true                           /* Nettoyage du dossier dist à chaque génération webpack */
  },
  mode: 'development',
  devtool: 'inline-source-map',           /* Définition du mode dev et de la map pour avoir des erreurs verbose avec map*/
};
```

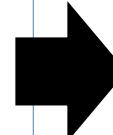
Mettre en place Webpack 4/4

- Génération du projet avec webpack
 - L'option **-watch** permet relance webpack dès qu'un changement est détecté
 - <https://webpack.js.org/guides/development/>

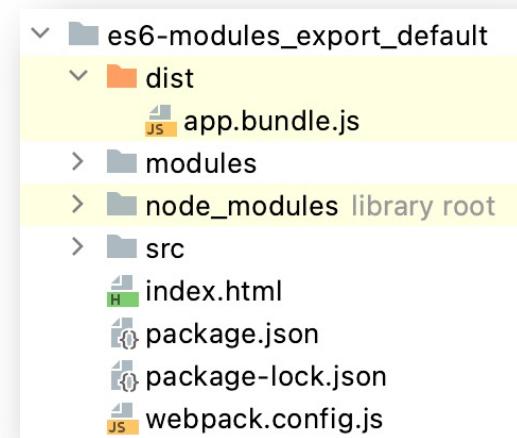
```
npx webpack --config webpack.config.js --watch
```

- Simplifier le lancement de webpack :

```
{  
  "name": "es6-modules_export_default",  
  "version": "1.0.0",  
  "description": "",  
  "private": true,  
  "main": "src/app.js",  
  "scripts": {  
    "dev": "webpack --watch"  
  },  
  ...  
}
```



```
npm run dev
```



Webpack et Babel 1/2

- Mise en place d'un loader Webpack
 - <https://webpack.js.org/concepts/#loaders>

- Installation du loader Babel :

```
npm install -D babel-loader @babel/core @babel/preset-env webpack
```

- Configuration de webpack :
 - <https://webpack.js.org/loaders/babel-loader/>

- **Désormais, let est converti en var**

```
module: {
  rules: [
    {
      test: /\.m?js$/,
      exclude: /(node_modules|bower_components)/,
      use: [
        {
          loader: 'babel-loader',
          options: {
            presets: ['@babel/preset-env']
          }
        }
      ]
    }
  ]
}
```



webpack.config.js

Webpack et Babel 2/2

- Babel est configurable pour mettre en place des polyfills automatiquement
 - <https://babeljs.io/docs/en/babel-preset-env#targets>
- Il est possible de configurer **le plugin preset-env** pour cibler spécifiquement des versions précises de navigateurs

```
module: {  
  rules: [  
    {  
      test: /\.m?js$/,  
      exclude: /(node_modules|bower_components)/,  
      use: {  
        loader: 'babel-loader',  
        options: {  
          presets: ['@babel/preset-env'],  
          targets: ['defaults'],  
          plugins: ['@babel/plugin-proposal-object-rest-spread']  
        }  
      }  
    }  
  ]  
}
```



webpack.config.js

Webpack et Serveur 1/1

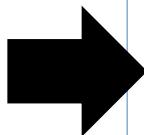
- Webpack fournit un serveur web pour lancer votre projet
 - <https://webpack.js.org/configuration/dev-server/>

- Installation et configuration :

```
npm install -D webpack-dev-server
```

- Lancement avec :

```
npx webpack serve
```



```
module.exports = {  
  entry: {  
    app: ['./src/js/css.js', './src/js/app.js']  
  },  
  output: {  
    ...  
  },  
  devServer: {  
    contentBase: path.join(__dirname, 'dist'),  
    compress: true,  
    port: 9000,  
  },  
};
```

 webpack.config.js

```
MacBook-Pro-de-Boris:es6-modules_webpack4_4 eslint sauvageb$ npx webpack serve  
i [wds]: Project is running at http://localhost:9000/  
i [wds]: webpack output is served from /  
i [wds]: Content not from webpack is served from /Users/sauvageb/Desktop/WORKSPAC
```

Webpack et Minification 1/1

- Webpack nous permet de minifier nos sources
 - <https://webpack.js.org/guides/production/#minification>
- Depuis Webpack v4+, la minification est activée en **mode production**
 - Utilise le plugin TerserPlugin
- Il est possible d'utiliser un autre plugin comme ClosureWebPackPlugin

Webpack et Debugging

- Il est possible de mettre une cartographie de code source
 - Habituellement nommés sources-map
 - https://developer.mozilla.org/fr/docs/Tools/Debugger/How_to/Use_a_source_map
- Fichier grâce auquel le débogueur peut faire le lien entre le code étant exécuté et les fichiers sources originaux
- Permettent au navigateur de reconstruire la source originale et de l'afficher dans le Débogueur

Webpack et Code Splitting

- Permet de diviser votre code en bundles utilisables à la demande ou en parallèle
- Utilisé pour réduire les bundles et contrôler la priorisation des chargements
 - Lazy Loading (<https://webpack.js.org/guides/lazy-loading/#root>)
- Procédé qui peut avoir un impact majeur sur le temps de chargement

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  entry: {
    index: './src/index.js',
    another: './src/another-module.js',
  },
  output: {
    filename: 'main.js',
    filename: '[name].bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
};
```

Séparation simple avec **entry**

```
module.exports = {
  entry: {
    index: './src/index.js',
    another: './src/another-module.js',
    index: {
      import: './src/index.js',
      dependOn: 'shared',
    },
    another: {
      import: './src/another-module.js',
      dependOn: 'shared',
    },
    shared: 'lodash',
  },
  ...
```

Séparation avec **partage de modules**

Webpack et Lazy Loading 1/2

- Pour faire suite au **Code Splitting**, le **Lazy Loading** permet le chargement différé
- Permet de déclencher le chargement d'un module si besoin
- Accélère le chargement initial de l'application
 - allège son poids global

```
button.onclick = e => {
  import(/* webpackChunkName: "lib" */ './modules/lib').then(module => {
    const lib = module.default;

    lib.init();
  });
}
```

Webpack et Lazy Loading 2/2

- Exemple :

```
import _ from 'lodash';

function component() {
  const element = document.createElement('div');
  const button = document.createElement('button');
  const br = document.createElement('br');

  button.innerHTML = 'Click me and look at the console!';
  element.innerHTML = _.join(['Hello', 'webpack'], ' ');
  element.appendChild(br);
  element.appendChild(button);

  button.onclick = e => {
    import(/* webpackChunkName: "lib" */ './modules/lib').then(module => {
      const lib = module.default;

      lib.init();
    });
  }
  return element;
}

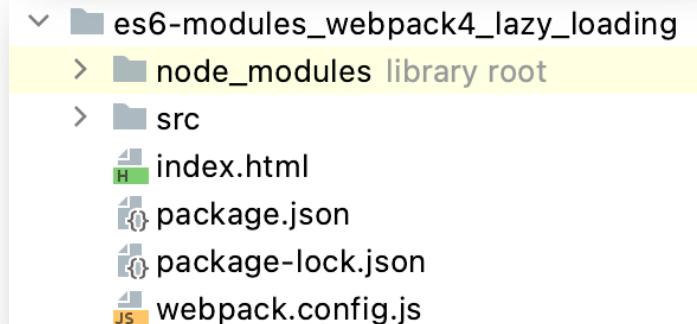
document.body.appendChild(component());
```

```
const init = () => {
  console.log('Hey');
}

export default {init};
```

Hello webpack

Click me and look at the console!

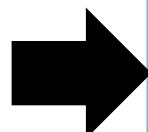


Webpack et Template HTML

- Il est possible d'utiliser un template HTML pour la génération du dossier dist
 - <https://webpack.js.org/guides/output-management/#setting-up-htmlwebpackplugin>

```
npm install --save-dev html-webpack-plugin
```

- Ajouter le require plugin
- Ajouter la configuration
- Exécuter Webpack
 - Le fichier html est copié dans /dist



```
const path = require('path');
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  ...
},
plugins: [
  new HtmlWebpackPlugin({
    template: "./index.html"
  })
];
```

Webpack et CSS 1/2

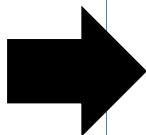
- Il est utile de gérer le Style CSS avec webpack
 - <https://webpack.js.org/loaders/css-loader/#root>

`npm install --save-dev css-loader`



`npm install --save-dev style-loader`

- Ajouter et configurer le module



```
module: {
  rules: [
    {
      ...
    },
    {
      test: /\.css$/i,
      use: ["style-loader", "css-loader"],
    },
  ],
},
```

Webpack et CSS 2/2

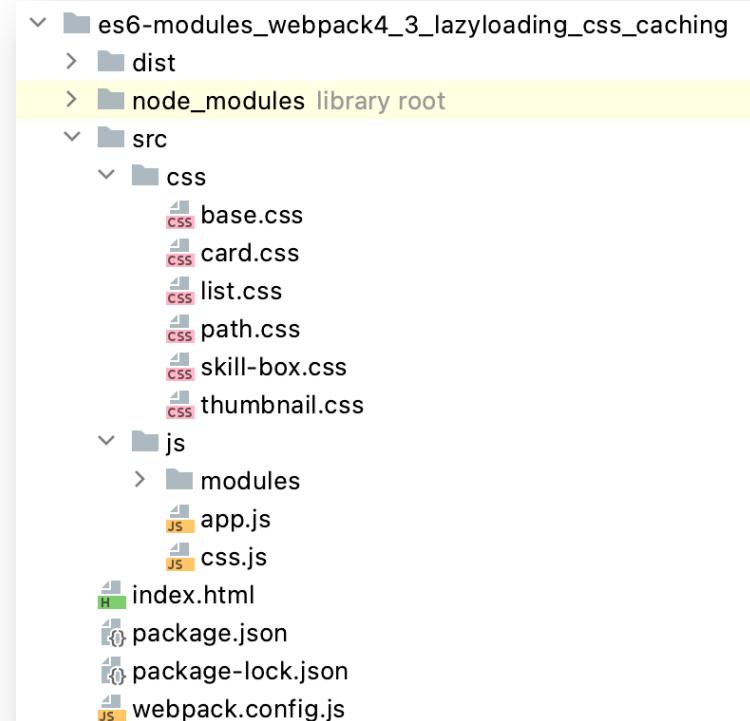
- Il suffit ensuite de charger les styles css grâce à un fichier **css.js** :

```
css.js
```

```
import base from './css/base.css';
import card from './css/card.css';
import list from './css/list.css';
import path from './css/path.css';
import skillbox from './css/skill-box.css';
import thumbnail from './css/thumbnail.css';
```

```
webpack.config.js
```

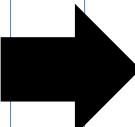
```
module.exports = {
  entry: {
    app: ['./src/js/css.js', './src/js/app.js']
  },
  output: {
    filename: 'assets/js/[name].[contenthash].bundle.js',
    path: path.resolve(__dirname, 'dist'),
    clean: true
  },
}
```



Webpack et Mise en cache

- La mise en cache permet d'améliorer la vitesse de chargement
 - Réduit le traffic réseau
- Il est possible de configurer un hash dans le nom des fichiers de sortie :
 - <https://webpack.js.org/configuration/output/#outputfilename>

```
<!DOCTYPE html>
<html>
<head>
  <title>Example ES6 - WEBPACK</title>
  <meta charset="utf-8"/>
  <script src="assets/js/app.b3924c4988003212a559.bundle.js"
defer></script>
</head>
...
```



```
module.exports = {
  entry: {
    app: './src/js/app.js'
  },
  output: {
    filename: 'assets/js/[name].[contenthash].bundle.js',
    path: path.resolve(__dirname, 'dist'),
    clean: true
  },
  mode: 'development',
}
```

Webpack et ESLint 1/2

- Il est possible de mettre en place un outil d'analyse de code
 - Permet d'identifier les modèles problématiques trouvés dans le code
- Installation du plugin Webpack eslint

```
npm install eslint-webpack-plugin --save-dev
```

- Installation d'ESLint
 - Un fichier `.eslintrc.js` est créé

```
npx eslint --init
```

- Ajout d'ignorePatterns

```
module.exports = {  
  "env": {  
    "browser": true,  
    "es2021": true  
  },  
  "extends": "eslint:recommended",  
  "parserOptions": {  
    "ecmaVersion": 12,  
    "sourceType": "module"  
  },  
  "ignorePatterns": ["css.js"],  
  "rules": {}  
};
```



Webpack et ESLint 2/2

- Avec EsLint, il est possible d'utiliser différentes extensions :
- Utilisation de **standardjs**

– <https://github.com/standard/eslint-config-standard>

```
npm install eslint-config-standard
```

```
module.exports = {  
  ...  
  "extends": ["standard"],  
  ...  
};
```



- Utilisation de **airbnb-base**

– <https://www.npmjs.com/package/eslint-config-airbnb>

```
npx install-peerdeps --dev eslint-config-airbnb-base
```

```
module.exports = {  
  ...  
  "extends": ["airbnb-base"],  
  ...  
};
```



- **Correction automatique :**

```
npx eslint --fix
```

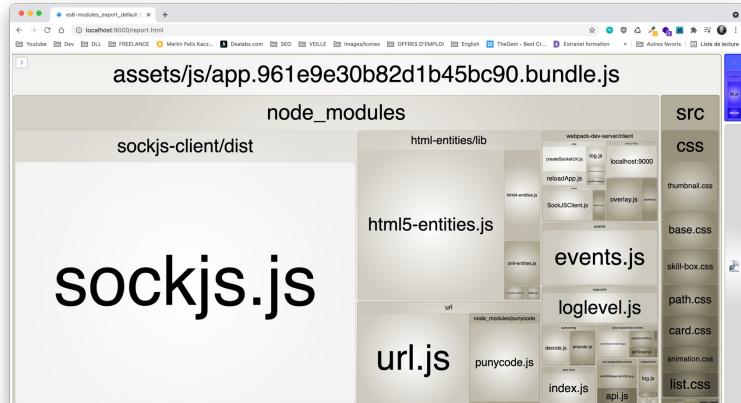
Webpack et Analyzer 1/3

- Il est possible de visualiser la proportion de vos dépendances
- Installation du plugin et configuration du fichier webpack :

```
npm install -D webpack-bundle-analyzer
```

- Lancement :

```
npx webpack serve --analyze
```



```
const BundleAnalyzerPlugin = require('webpack-bundle-analyzer')  
.BundleAnalyzerPlugin;  
module.exports = {  
  ...  
},  
plugins: [  
  ...  
  new BundleAnalyzerPlugin({  
    openAnalyzer: false,  
    defaultSizes: 'gzip',  
    analyzerMode: 'static'  
  })  
]  
};
```

 webpack.config.js

Webpack et Analyzer 2/3

- Mise en place d'un script pour démarrer notre analyse avec Node.JS :

```
"scripts": {  
  "test": "echo \\\"Error: no test specified\\\" && exit 1",  
  "dev": "webpack serve",  
  "report": "webpack serve --analyze"  
},
```



package.json

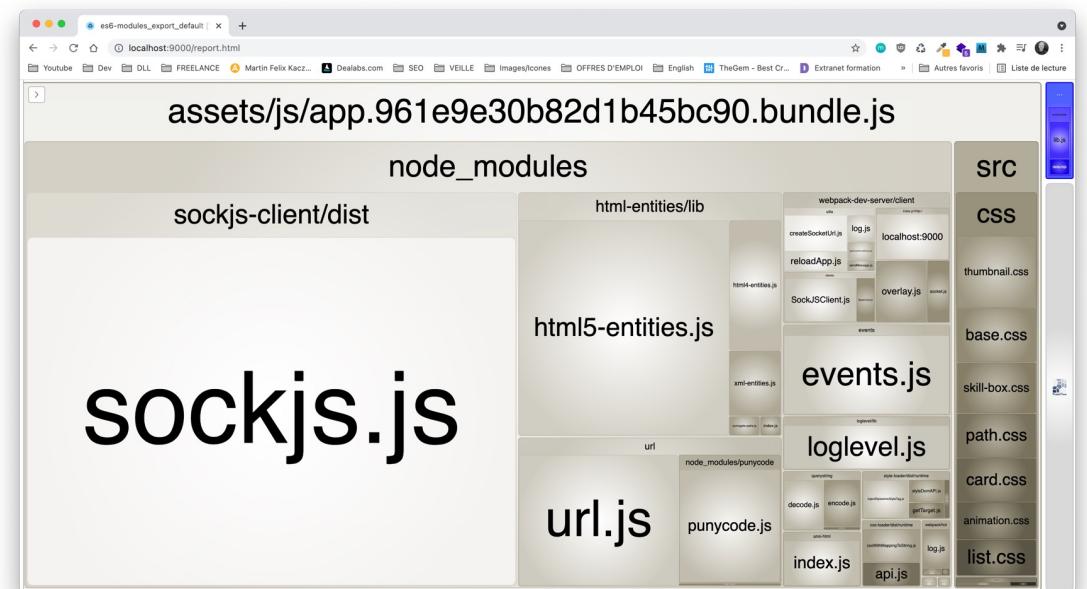
- Modification du fichier webpack pour activer ou non le plugin :

```
let config = {  
  ... plugins: [...]  
};  
  
let analyzerMode = process.argv.find(param => param === '--analyze');  
config.plugins.push(new BundleAnalyzerPlugin({  
  openAnalyzer: false,  
  defaultSizes: 'gzip',  
  analyzerMode: analyzerMode ? 'static' : 'disabled'  
}));  
module.exports = config;
```

Webpack et Analyzer 3/3

- Lancement du serveur, génération d'un rapport, visualisation sur :
 - <http://localhost:9000/report.html>
- Avec la commande suivante :

```
npm run report
```



Automatisation jsDoc

Class: hideLoader

hideLoader()

new hideLoader()

Represents a book.

Source: app.js, line 5

[Home](#)

Classes

hideLoader

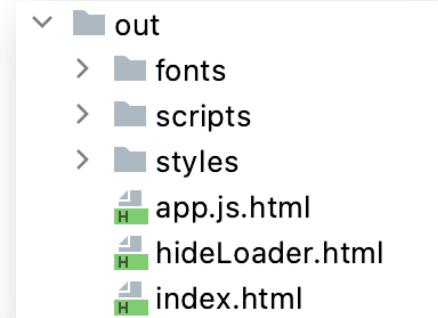
- Il est possible de documenter le code JavaScript avec la bibliothèque jsDoc :
 - <https://jsdoc.app/>

- Installation possible avec : `npm install -g jsdoc`

- Documentation du code :

```
/**  
 * Will hide your loader defined by .loader class  
 */  
const hideLoader = () => {  
  const loader = document.querySelector('.loader');  
  loader.remove();  
};
```

- Exécution possible avec : `jsdoc src/js/`



Exercice

Faire les exercices suivants :

- Mettre en place Webpack sur votre projet
 - Génération du template HTML dans le dossier /dist
 - Transpilation avec Babel
 - Lancement du serveur Web avec le /dist
 - Minification du code
 - Lint du code avec ESLint
- Documentation avec JsDoc

Développement Web

TYPESCRIPT



Introduction

- TypeScript est une surcouche du langage JavaScript
- Contributeur principal : Anders Hejlsberg (créateur de Delphi et de Turbo Pascal)
 - Souhait de conserver ce qui fait la force de JavaScript en supprimant les imperfections dépeintes par certains réfractaires au langage
- Il propose :
 - Annotations
 - Enumérations
 - un typage des variables et fonctions
- Se rapproche de ses concurrents célèbres et populaires tels que Java et C++
- Le TypeScript est transcompilé
 - code TypeScript transformé en JavaScript et ensuite être intégré à une page HTML

Typage sur TypeScript 1/2

- **TypeScript** permet d'indiquer au compilateur et aux développeurs quel est le type attendu dans une variable et même dans le retour d'une fonction
- Pour typer une variable, il suffit d'indiquer son type : `let uneVariable: string;`
- Pour une fonction, c'est le même principe :
`function uneFonction(): string {
 return 'une chaîne de caractères';
}`
- Il existe 7 types :
 - Une chaîne de caractères (**string**)
 - Un nombre (**number**)
 - Un booléen (**boolean**)
 - Un tableau (**string[]**)
 - Une énumération (**myEnum**)
 - Aucun type (**void**)
 - Un type, non défini pour le moment (**any**)

Typage sur TypeScript 2/2

- Exemple avec les différents types :

```
let uneChaineDeCaractere: string;  
// Nombre  
let unNombre: number;  
// Booléen  
let unBoolean: boolean;  
// Tableaux  
let unTableauDeChaineDeCaracteres: string[];  
let unTableauDeNombres: number[];  
// Enumération  
enum uneEnumeration {}  
let uneVariableDeTypeEnum: MonEnum;  
// Sans aucun type  
let uneVariableSansType: void;  
// Sans type défini pour l'instant  
let uneVariableSansTypeDefiniPourLinstant: any;
```

```
enum MonEnum {  
    Valeur1,  
    Valeur2,  
    Valeur3,  
}
```

Les énumérations

- Une **énumération** est une liste de constantes
- Ressemble énormément au type tableau, mais diffère beaucoup dans sa déclaration

```
enum PointsCardinaux {  
    N,  
    S,  
    E,  
    O  
}
```

```
// Déclaration  
let uneDirection: PointsCardinaux;  
// Utilisation  
uneDirection = PointsCardinaux.N; // uneDirection = 0  
uneDirection = PointsCardinaux.S; // uneDirection = 1  
uneDirection = PointsCardinaux.E; // uneDirection = 2  
uneDirection = PointsCardinaux.O; // uneDirection = 3
```

Les fonctions

- Déclarer une fonction en utilisant **TypeScript** ajoute quelques éléments à sa signature

```
function traducteurMajuscule(mot: string): string {  
    return mot.toUpperCase();  
}  
  
let phrase = 'Il était une fois';  
let tabMots: string[] = phrase.split(' ');  
for (let i = 0; i < tabMots.length; i++) {  
    console.log(traducteurMajuscule(tabMots[i]));  
}
```

tsc fichier.ts

node fichier.js

IL
ÉTAIT
UNE
FOIS



- Cette fonction prend un mot de type chaîne de caractères en argument et le traduit en majuscules

Les fonctions particulières

- **Fonction anonyme** : Une fonction peut-être directement affectée à une variable

```
let motTraduit = function traducteurMajuscule(mot: string): string {  
    return mot.toUpperCase();  
};
```

- **Fonction lambda ou fléchées** : grâce à ES6, il est possible de raccourcir avec =>

```
let motTraduit = (mot: string): string => {  
    return mot.toUpperCase();  
};
```

```
let value = motTraduit("example")
```

- Inutile de préciser que c'est une fonction
- Fonction anonyme donc pas besoin de nom
- Le type de retour de la fonction est implicite

Les classes

- **Classe minimale :** Ressemble à la syntaxe d'une classe Java
 - contient des attributs, un constructeur, des méthodes et des accesseurs

```
class Voiture {  
}  
// Déclaration d'une nouvelle voiture  
let maVoiture = new Voiture();
```

- **Attributs :** précédés du mot-clé **private** afin de respecter l'encapsulation

```
class Voiture {  
    private _kilometrage: number;  
    private _proprietaire: string;  
    private _couleur: string;  
}
```

Le Constructeur

- Permettent d'accéder aux attributs privé d'une classe et de les modifier

```
class Voiture {  
    // Attributs  
    private _kilometrage: number;  
    private _proprietaire: string;  
    private _couleur: string;  
    // Le constructeur  
    constructor(kilometrage: number, proprietaire: string, couleur: string) {  
        this._kilometrage = kilometrage;  
        this._proprietaire = proprietaire;  
        this._couleur = couleur;  
    }  
}
```

```
let maVoiture = new Voiture(180000, 'CapitaineFlam', 'Rouge');
```

Les getters/setters

- Permettent d'accéder aux attributs privé d'une classe et de les modifier

```
// Getters
get kilometrage() { return this._kilometrage; }
get proprietaire() { return this._proprietaire; }
get couleur() { return this._couleur; }
```

```
// Setters
set kilometrage(km: number) { this._kilometrage = km; }
set proprietaire(nom: string) { this._proprietaire = nom; }
set couleur(couleur: string) { this._couleur = couleur; }
```

```
// Déclaration d'une nouvelle voiture
let maVoiture = new Voiture();
```

```
// Utilisation des setters
maVoiture.proprietaire = 'Actarus';
maVoiture.kilometrage = 120000;
maVoiture.couleur = 'grise';
```

```
// Affichage de la voiture grâce aux getters
console.log('Ma voiture : ' + maVoiture.proprietaire + ' - ' + maVoiture.kilometrage + ' km - ' + maVoiture.couleur);
```

Les Méthodes

- Une méthode est une fonction qui représente une action possible de l'objet
- **Exemple :** La voiture peut **avancer** ou **reculer**

```
avance(nbreDeMetres: number) { this._kilometrage += nbreDeMetres; }
recule(nbreDeMetres: number) { this._kilometrage -= nbreDeMetres; }
```

- **Utilisation :**

```
let maVoiture = new Voiture(180000, 'CapitaineFlam', 'Rouge');
console.log('kilométrage au départ : ' + maVoiture.kilometrage);
console.log('J'avance ...');
maVoiture.avance(100);
console.log('Je recule ...');
maVoiture.recule(50);
console.log('kilométrage à l'arrivée : ' + maVoiture.kilometrage);
console.log('Si quand j'avance, tu recules, ...');
```

Les interfaces

- Il est possible de définir des interfaces pour typer un regroupement de données :

```
interface Person {  
    firstname: string;  
    lastname: string;  
    age?: number;  
}
```

```
function sayHello(person: Person) {  
    return `Hello ${person.firstname} ${person.lastname}`;  
}
```

- Des propriétés facultatives sont définissables avec ? Dans l'interface

```
const greet = sayHello({firstname: "Boris", lastname: "Sauvage"});  
console.log(greet);
```

L'héritage

- **Une classe** peut hériter d'une autre classe
 - La classe qui hérite de l'autre possède les mêmes attributs et méthodes que celle-ci, mais peut en posséder d'autres
 - Permet de factoriser les attributs et méthodes redondants

```
class TracteurTondeuse extends Voiture {
  private _nbHelices: number;

  constructor(kilometrage: number, propriétaire: string, couleur: string) {
    super(kilometrage, propriétaire, couleur);
    this._nbHelices = 2;
  }

  abaisse(nbreDeCm: number) {
    console.log('hélice abaissée de ' + nbreDeCm + 'centimètres');
  }

  remonte(nbreDeCm: number) {
    console.log('hélice remontée de ' + nbreDeCm + 'centimètres');
  }
}
```

```
let tt = new TracteurTondeuse(12, 'Jean Dupont', 'Vert');
console.log('Kilométrage : ' + tt.kilometrage);
tt.abaisse(10);
tt.avance(0.1)
tt.remonte(10);
tt.reucle(0.1);
console.log('Kilométrage : ' + tt.kilometrage);
```

Les modules

- **Un module** est un ensemble de classes
- **Chaque élément du module a une portée limitée au module**, sauf s'il est **exporté**
 - Il peut y avoir une classe Voiture dans le module **monModule01** et une autre classe Voiture dans le module **monModule02**

```
module monModule {  
  
    class Vehicule { ... }  
  
    class Voiture extends Vehicule { ... }  
  
    class TracteurTondeuse extends Vehicule { ... }  
}
```

Exporter les modules

- Pour utiliser les classes et méthodes d'un module, elle doivent être exportées

```
module monModule {  
  
    export class Vehicule { ... }  
  
    class Voiture extends Vehicule { ... }  
  
    class TracteurTondeuse extends Vehicule { ... }  
}
```

```
let voiture = new monModule.Voiture(0, 'Boris', 'blanche');
```

```
voiture.methodeDuVehicule();
```

Mettre en place TypeScript

- À la racine du projet, installez **TypeScript** avec : `npm install -g typescript`
- Sur **VS Code**, il est possible de créer des fichiers avec **l'extension .ts**
- Afin de transcompiler les fichiers TypeScript en fichiers JavaScript : `tsc fichier.ts`

`tsc -v`
- Il est ensuite possible d'intégrer ces fichiers JavaScript via le `<script></script>`
- Il est aussi possible de lancer le fichier directement avec node : `node fichier.js`

Typage statique avec TypeScript

- **JavaScript est un langage non fortement typé mais dynamique**
 - Le type des données n'est identifié qu'à l'exécution du programme
 - On peut ainsi modifier le type des données à la volée au cours de l'exécution
- **TypeScript introduit la notion d'un typage plus fort que pour le langage JavaScript**
 - Typage manifesté par l'ajout de plusieurs types de bases ainsi qu'un typage statique
- **Un langage de typage statique vérifie les types des variables à la compilation**
- **Un langage de typage dynamique effectue cette vérification à l'exécution**

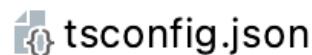
TypeScript avec Webpack 1/3

- Mettre en place TypeScript sur le projet

```
npm install --save-dev typescript ts-loader
```

- Adapter la configuration :

```
{  
  "compilerOptions": {  
    "outDir": "./dist/",  
    "noImplicitAny": true,  
    "module": "ES2020",  
    "target": "esnext",  
    "allowJs": true,  
    "moduleResolution": "node",  
    "lib": [ "es2020", "dom" ]  
  }  
}
```



```
module: {  
  rules: [  
    // LOADER TYPESCRIPT  
    {  
      test: /\.tsx?$/,  
      use: 'ts-loader',  
      exclude: /node_modules/,  
    },  
  ]  
}
```



TypeScript avec Webpack 2/3

- Renommer tous les fichiers .js en .ts
- Activer l'import dynamique qui bloque la compilation :

```
resolve: {  
    extensions: ['.tsx', '.ts', '.js'],  
},
```



```
{  
    "compilerOptions": {  
        "outDir": "./dist/",  
        "noImplicitAny": true,  
        // "module": "es6",  
        "module": "ES2020",  
        "target": "es5",  
        "jsx": "react",  
        "allowJs": true,  
        "moduleResolution": "node"  
    }  
}
```



TypeScript avec Webpack 3/3

- Utilisez les types pour vos variables et vos fonctions

```
const button: HTMLElement = document.querySelector('...');  
button.onclick = () => { ... };
```

```
let picture:HTMLImageElement = document.querySelector("#person-picture");  
picture.src = user.picture.large;
```

```
let email: HTMLAnchorElement = document.querySelector("#person-email");  
email.href = "mailto:" + user.email;
```

ESLint avec TypeScript & Webpack

- Il est continué à utiliser ESLint avec TypeScript

```
npm install --save-dev @typescript-eslint/parser @typescript-eslint/eslint-plugin
```

```
module.exports = {  
  ...  
  "root": true,  
  "parser": "@typescript-eslint/parser",  
  "plugins": [  
    "@typescript-eslint"  
,  
    "extends": [  
      "eslint:recommended",  
      "plugin:@typescript-eslint/eslint-recommended",  
      "plugin:@typescript-eslint/recommended"  
    ]  
  }  
}
```



Typings et autocomplétion 1/2

- Avec TypeScript, il existe des fichiers définissant des types statiques
- Fichiers contenant l'ensemble des signatures de méthode d'une bibliothèque
- Permet d'alimenter l'auto-complétion d'avoir l'API complète d'une bibliothèque
 - Il n'est plus nécessaire de connaître par cœur l'API des librairies
- Source principale : <https://github.com/DefinitelyTyped/DefinitelyTyped>
 - Contient un grand nombre de fichiers de déclaration des librairies connues
 - Fichiers de déclaration sont mis à jour par la communauté
 - il y a des décalages entre la version de la librairie et celle du fichier de déclaration
- Il est possible de déclarer et publier ses propres fichiers de déclaration
 - <https://www.typescriptlang.org/docs/handbook/declaration-files/introduction.html>

Typings et autocomplétion 2/2

- Il est possible de rechercher des définitions de types spécifiques sur :
 - <https://www.typescriptlang.org/dt/search?search>
- Une fois installé, vous aurez l'auto-complétion des commandes
 - <https://www.typescriptlang.org/docs/handbook/declaration-files/consumption.html>

Exemple :

```
import * as _ from "lodash";
_.padStart("Hello TypeScript!", 20, " ")
```



```
"devDependencies": {
...
"@types/lodash": "^4.14.171",
```



Webpack

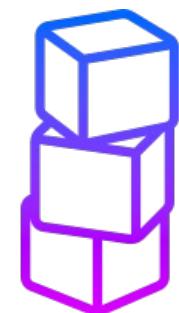
Exercice

Faire les exercices suivants :

- Mettre en place TypeScript sur votre projet
 - Transformer la configuration webpack

Développement Web

WEB COMPONENTS



Introduction : les Composants Web

- Technologies permettant de créer des composants d'interface graphique personnalisés et réutilisables
- Intégré aux navigateurs
 - ne nécessitent pas de bibliothèque externe comme jQuery ou Dojo
- Utilisables grâce à une simple déclaration d'importation à une page HTML
 - utilisent les nouvelles capacités standard de navigateur, ou celles en cours de développement
- Les composants Web ont pour but de résoudre la problématique de réutilisation de code
 - <https://caniuse.com/?search=custom%20element>
 - <https://caniuse.com/?search=shadow%20dom>

Approche basique

1. Créer une classe ECMAScript 2015
 - Définira la fonctionnalité du composant Web en utilisant la syntaxe de classe ECMAScript 2015
2. Enregistrer l'élément personnalisé en utilisant `CustomElementRegistry.define()`
 - avec en paramètre le nom de l'élément à définir
 - la classe ou la fonction dans laquelle la fonctionnalité est spécifiée
3. Connecter un shadow DOM à l'élément personnalisé avec `Element.attachShadow()`
 - Ajout des éléments-fils, des écouteurs d'événements, etc., au shadow DOM avec les méthodes usuelles
4. Définir un template HTML en utilisant `<template>` et `<slot>`
 - Utilisation des méthodes DOM usuelles pour cloner le template et le connecter au shadow DOM
5. Utiliser l'élément personnalisé sur une page, comme tous les autres éléments HTML

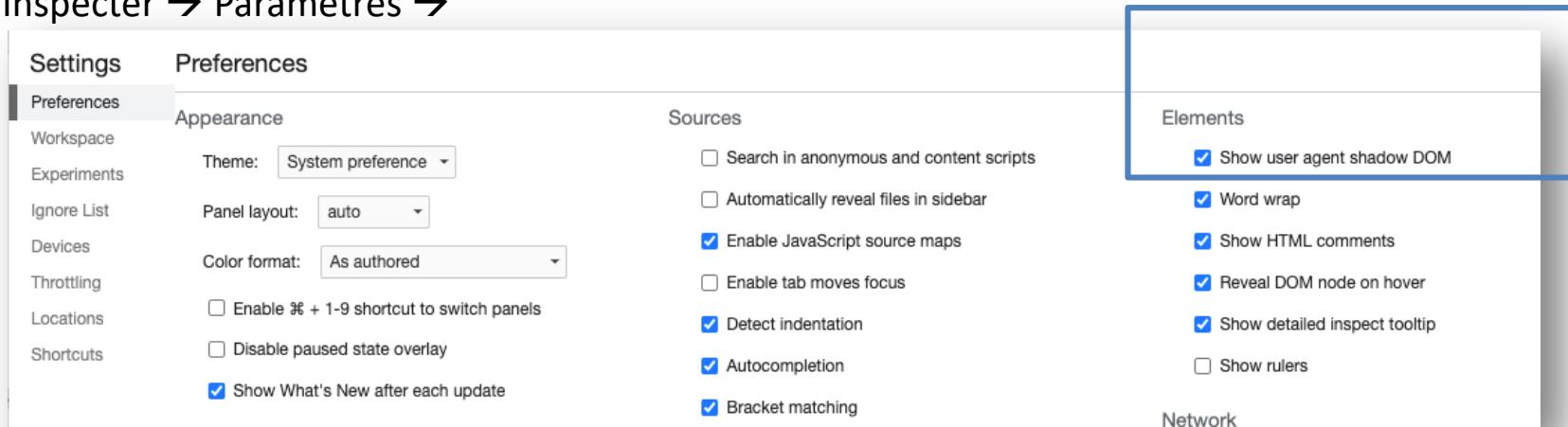
Custom elements v1

- Il est possible de créer des éléments qui encapsulent vos fonctionnalités
- Le contrôleur des éléments personnalisés est **CustomElementRegistry**
 - permet d'enregistrer un élément personnalisé sur une page
 - permet de renvoyer des informations sur les éléments personnalisés enregistrés
- Pour enregistrer un élément sur la page, vous utilisez `CustomElementRegistry.define()`
 - `DOMString` représente le nom que vous donnez à l'élément (comportant un tiret)
 - Un Objet de classe définissant le comportement de l'élément
 - Un Objet d'options qui indique l'élément intégré dont votre élément hérite

```
// Define the new element
customElements.define('word-count', WordCount, { extends: 'p' });
```

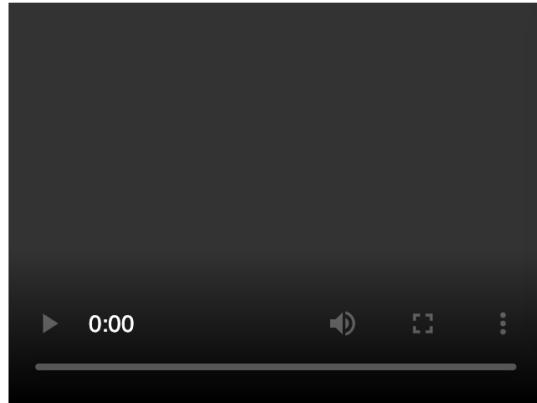
Shadow DOM v1 1/3

- Moyen pour garder la structure de balisage, le style et le comportement cachés et séparés du reste de code de la page (encapsulation)
- Permet à un composant d'avoir sa propre arborescence DOM « fantôme »
 - Ne peut pas être accédé accidentellement à partir du document principal
- Configuration nécessaire pour afficher les DOM Fantômes (Google Chrome) :
 - Inspecter → Paramètres →



Web Components

Exemples de DOM Shadow



```
<!DOCTYPE html>
<html lang="en">
  <head>..</head>
  ..<body> == $0
    <video width="320" height="240" controls>
      #shadow-root (user-agent)
        <div pseudo="-webkit-media-controls" class="use-default-poster sizing-small phase-pre-ready state-no-source">..</div>
          <source src="movie.mp4" type="video/mp4">
          <source src="movie.ogv" type="video/ogg">
            " Your browser does not support the video tag. "
        </video>
    </body>
</html>
```

<video></video>

```
<!DOCTYPE html>
..<html lang="en"> == $0
  <head>..</head>
  <body>
    <input type="range">
      #shadow-root (user-agent)
        <div flex>
          <div pseudo="-webkit-slider-runnable-track" id="track">
            <div id="thumb"></div>
          </div>
        </div>
    </input>
  </body>
</html>
```

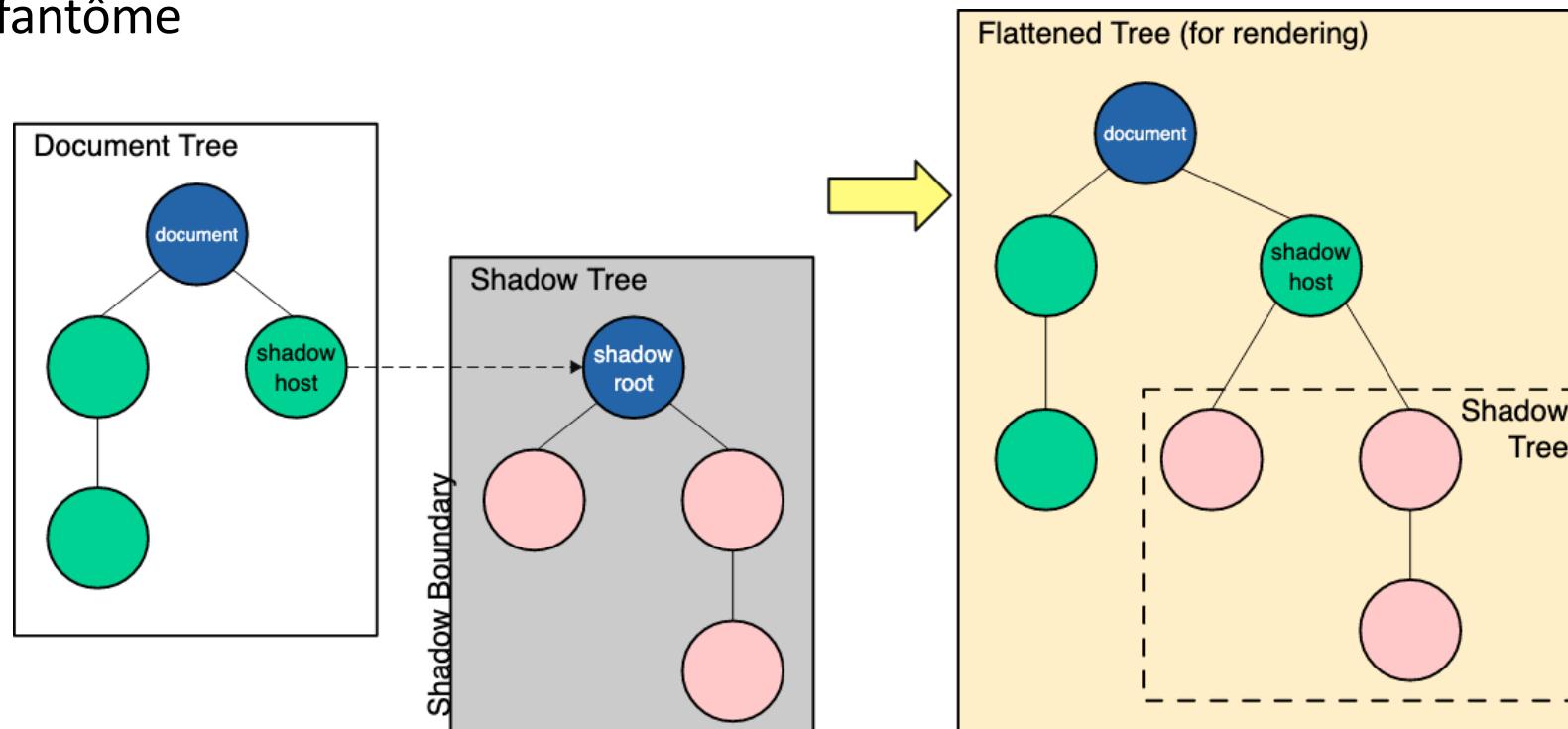
<input type="range"></input>



- Le navigateur utilise le DOM et le CSS en interne pour les dessiner

Shadow DOM v1 2/3

- Les sélecteurs habituels ne permettent pas de récupérer des éléments le DOM fantôme



- Cette technique n'est pas nouvelle
 - les navigateurs l'ont utilisé depuis longtemps pour encapsuler la structure interne d'un élément

Shadow DOM v1 3/3

- Il est possible d'associer une racine fantôme à un élément avec `Element.attachShadow`

```
let fantome = element.attachShadow({mode: 'open'});
```

- L'accès au DOM Fantôme se fait ensuite avec :

```
let monDomFantome = monElementPerso.shadowRoot;
```

- Mode open**

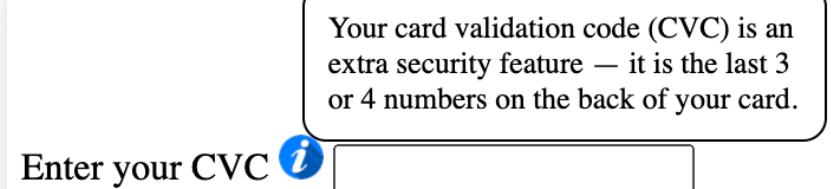
- L'accès au DOM fantôme est possible avec JavaScript dans le contexte de la page principale

- Mode closed**

- L'accès au DOM fantôme n'est pas possible depuis l'extérieur
 - Mode des éléments natifs comme <video>

Exemple de DOM Shadow Custom

```
<popup-info img="img/alt.png"  
text="Your card validation code (CVC) is an extra ...">
```



- Quand l'icône obtient l'attention (:focus), le texte s'affiche dans une fenêtre indépendante pour fournir plus d'informations contextuelles

```
class PopUpInfo extends HTMLElement {  
constructor() {  
super();  
...  
// Create a shadow root  
const shadow = this.attachShadow({mode: 'open'});  
}  
// Define the new element  
customElements.define('popup-info', PopUpInfo);  
}
```

Exemple : Compteur de mots 1/3

- Réalisons un composant **compteur de mots**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Composant : compteur de mots</title>
  </head>
  <body>
    <h1>Compteur de mots</h1>
    <article></article>
    <script src="main.js"></script>
  </body>
</html>
```

Compteur de mots

Titre d'exemple

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc pulvinar sed justo sed viverra. Aliquam ac scelerisque libero. Etiam leo felis, pulvinar et diam id, sagittis pulvinar diam. Nunc pellentesque rutrum sapien, sed faucibus u facilisis sit amet.

Pellentesque ornare tellus sit amet massa tincidunt congue. Morbi cursus, tellus vitae pulvinar dictum, dui turpis faucibus vestibulum vel molestie vitae, mollis vitae eros. Sed lacinia scelerisque diam, a varius urna iaculis ut. Nam lacinia mauris purus eget neque. Donec viverra in ex sed ullamcorper. In ac nisi vel enim accumsan feugiat et sed augue. Integer

Words: 212

```
class WordCount extends HTMLElement {
  constructor() {
    super();
    // Ecrire la fonctionnalité de l'élément ici
  }
}
```

Exemple : Compteur de mots 2/3

- La définition de notre élément word-count personnalisé est la suivante :
 - L'élément **word-count** possède un objet de classe est **WordCount** qui étend de <p>

```
class WordCount extends HTMLParagraphElement {  
    constructor() {  
        super();  
        // Fonctionnalité de l'élément  
    }  
}  
// Définition de l'élément  
customElements.define('word-count', WordCount, {extends: 'p'});
```

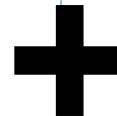
- Mettre du contenu HTML à compter

```
<article contenteditable="">  
    <h2>Titre d'exemple</h2>  
    <p>Lorem</p>  
    <p>Pellentesque ornare tellus sit amet p>  
    <p is="word-count"></p>  
</article>
```

Exemple : Compteur de mots 3/3

- Finalisation de l'élément :

```
class WordCount extends HTMLParagraphElement {  
  constructor() {  
    super();  
  
    const wcParent = this.parentNode;  
    // Create text node and add word count to it  
    const text = document.createElement('span');  
    // count words in element's parent element  
    text.textContent = `Words: ${countWords(wcParent)}`;  
  
    // Create a shadow root and append it to the shadow root  
    const shadow = this.attachShadow({mode: 'open'});  
    shadow.appendChild(text);  
  
    function countWords(node) {  
      const text = node.innerText || node.textContent;  
      return text.split(/\s+/g).length;  
    }  
  }  
}
```



Mise à jour du compteur toutes les 200 ms

```
// Update count when element content changes  
setInterval(function () {  
  const count = `Words: ${countWords(wcParent)}`;  
  text.textContent = count;  
}, 200);
```

Templates 1/2

- Permet de définir une structure de balises utilisable à volonté
 - Évite de réécrire cette structure systématiquement
 - Élément non retranscrit dans le DOM, mais manipulable avec JavaScript

```
<template id="my-paragraph">
  <style>
    p {
      color: white;
      background-color: #666;
      padding: 5px;
    }
  </style>
  <p>Mon texte d'exemple</p>
</template>

<my-paragraph></my-paragraph>
<my-paragraph></my-paragraph>
<my-paragraph></my-paragraph>
```

template d'exemple

Mon texte d'exemple

Mon texte d'exemple

Mon texte d'exemple

```
<html>
  <head>...</head>
  <body cz-shortcut-listen="true">
    <h1>template d'exemple</h1>
    <template id="my-paragraph">
      #document-fragment
    </template>
    <my-paragraph>
      #shadow-root (open)
        <style>
          p {
            color: white;
            background-color: #666;
            padding: 5px;
          }
        </style>
        <p>Mon texte d'exemple</p>
      </my-paragraph>
      <my-paragraph>...</my-paragraph>
      <my-paragraph>...</my-paragraph>
    </body>
  </html>
```

Templates 2/2

- Il est possible de combiner les templates avec les composants web :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>template d'exemple</title>
  <script src="main.js" defer></script>
</head>
<body>
<h1>template d'exemple</h1>
<template id="my-paragraph">
  <style>
    p { ... }
  </style>
  <p>Mon texte d'exemple</p>
</template>
<my-paragraph></my-paragraph>
<my-paragraph></my-paragraph>
<my-paragraph></my-paragraph>
</body>
</html>
```

```
customElements.define('my-paragraph',
  class extends HTMLElement {
    constructor() {
      super();
      const template = document.getElementById('my-paragraph');
      const templateContent = template.content;

      this.attachShadow({mode: 'open'})
        .appendChild( templateContent.cloneNode(true) );
    }
  }
);
```

- La méthode **Node.cloneNode()** renvoie une copie du nœud sur lequel elle a été appelée
- Le contenu est ajouté à un DOM fantôme
 - Il peut contenir du `<style>` encapsulé à l'intérieur de l'élément

Templates et Slots

- **<template>** permet d'afficher de réutiliser une partie HTML statique
- **<slot>** permet de paramétriser chaque instance d'un élément utilisant **<template>**
 - Les slots sont identifiés par leur attribut name
 - Permettent de définir des champs dans le template
- Les Slots peuvent être alimentés avec n'importe quelle structure HTML

```
<p><slot name="my-text">Mon texte par défaut</slot></p>
```

Exemple avec un tableau de slots

```
<template>
<table>
  <tr>
    <th><slot name='title-col-1'></slot></th>
    <th><slot name='title-col-2'></slot></th>
  </tr>
  <tr>
    <td><slot name='col-1.row-1'></slot></td>
    <td><slot name='col-1.row-2'></slot></td>
  </tr>
  <tr>
    <td><slot name='col-2.row-1'></slot></td>
    <td><slot name='col-2.row-2'></slot></td>
  </tr>
</table>
<style> ...</style>
</template>
```

template d'exemple

Titre A	Titre B
Valeur A.1	Valeur A.2
Valeur B.1	Valeur B.2
Voitures	Prix
Tesla Model Y	Ford Mustang Mach-E
51,490 €	44,995 €

```
if ('attachShadow' in document.createElement('div')) {
  let templateContent = document.querySelector('template').content;
  let divs = document.querySelectorAll('div');

  divs.forEach(function (div) {
    div
      .attachShadow({mode: 'open'})
      .appendChild(templateContent.cloneNode(true))
  });
} else {
  console.warn('attachShadow not supported');
}
```

Exercice

Faire les exercices suivants :

- Mettre en place un template sur votre projet

REMARQUE :

- Lorsque vous ajoutez un produit, vous pouvez dupliquer un template, au lieu de créer des éléments avec **document.createElement()**

Développement Web

APIS POUR LES APPLICATIONS



Historique et navigation

- **window** fournit un accès à l'historique du navigateur via l'objet history
 - Permet d'avancer et de reculer dans l'historique de l'utilisateur
 - Permet de manipuler le contenu de l'ensemble de l'historique

```
// affiche le nombre d'entrée dans l'historique
let numberOfEntries = window.history.length;
alert(numberOfEntries);
// Recule dans l'historique
window.history.back();
// Avance dans l'historique
window.history.forward();
// Se déplace à un élément précis de l'historique
window.history.go(-1);
// == forward :
window.history.go(1);
// Ajouter une entrée dans l'historique
var stateObj = {foo: "bar"};
history.pushState(stateObj, "page 2", "bar.html");
// Remplace une entrée dans l'historique
history.replaceState(stateObj, "page 3", "bar2.html");
```

Stockage

- L'interface **Storage** de l'API Web Storage donne accès :
 - au stockage de session (`SessionStorage`)
 - au stockage local (`LocalStorage`) pour un domaine donné
- **Window.localStorage** : Pour manipuler le stockage local pour un domaine
 - <https://developer.mozilla.org/fr/docs/Web/API/Window/localStorage>
- **Window.sessionStorage** : Pour manipuler le stockage de session pour un domaine
 - <https://developer.mozilla.org/fr/docs/Web/API/Window/sessionStorage>

Stockage local 1/2

- **localStorage** vous permet d'accéder à un objet local Storage
 - similaire au sessionStorage
- Les données stockées dans le **localStorage** n'ont pas de délai d'expiration
- Les données stockées dans le **sessionStorage** sont nettoyées quand la session navigateur prend fin — donc quand on ferme le navigateur
- Les clés et les valeurs **sont toujours des chaînes de caractères**
 - Comme pour les objets, les clés entières seront automatiquement converties en chaînes de caractères

Stockage local 2/2

- Exemple de manipulation :

```
// Ajout d'une entrée
localStorage.setItem('myCat', 'Tom');
// Lecture d'une entrée
let cat = localStorage.getItem('myCat');
// Suppression d'une entrée
localStorage.removeItem('myCat');
// Nettoyage total du storage
localStorage.clear();
```

Stockage session 1/2

- **sessionStorage** permet d'utiliser un objet Storage
 - valable pour la session de navigation en cours et pour les pages du même domaine que la page actuelle
 - similaire à window.localStorage
- Les données du sessionStorage ont une durée vie limitée et expirent à la fin de la session de navigation actuelle
 - Une session de navigation dure aussi longtemps que le navigateur est ouvert et s'étend sur plusieurs chargements, rechargements et restaurations de pages
- une session de navigation n'est valable que pour le contexte de navigation actuel,
 - **ouvrir une page dans un nouvel onglet ou dans une nouvelle fenêtre provoquera l'initialisation d'une nouvelle session de navigation**, ce qui diffère du comportement des sessions utilisant des cookies

Stockage session 2/2

- **Exemple de manipulation :**

```
// Enregistre des données dans sessionStorage
sessionStorage.setItem('clé', 'valeur');

// Récupère des données depuis sessionStorage
let data = sessionStorage.getItem('clé');

// Supprime des données de sessionStorage
sessionStorage.removeItem('clé');

// Supprime toutes les données de sessionStorage
sessionStorage.clear();
```

Développement Web

WEB WORKER ET PWA

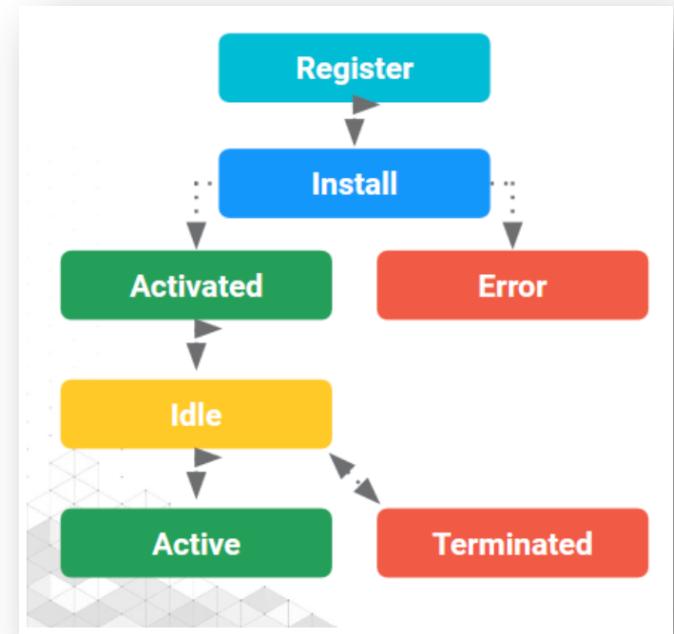
Web Worker 1/2

- l'API Service Worker permet d'offrir une meilleure expérience utilisateur :
 - Mise à disposition d'une expérience offline riche
 - Réalisation de synchronisations en arrière-plan
 - Envoies des notifications push
- Un service worker est un script qui tourne en arrière-plan de la page web dans un web worker
 - Au cours de sa vie, le script du service worker passe par plusieurs états ayant chacun leurs propres spécificités. Son cycle de vie est à part de celui de la page web
- le script du worker se termine lorsqu'il n'est pas utilisé et redémarre lorsque cela est nécessaire

Web Worker 2/2

Le cycle de vie d'un service worker

- L'installation
 - Nécessite un enregistrement avec la mise en cache pour le fonctionnement de la page
- L'activation
 - Le worker prend le contrôle de la page (peut nettoyer des données ou migrer vers un nouveau worker)
- En attente, traitement, terminé
 - Le worker est au repos (idle) et attendra des requêtes pour traiter des actions définies
 - Il peut passer en mode (terminated) selon le bon vouloir du système et de l'utilisation de la mémoire



PWA (Progressive Web App) 1/2

- Application web qui peuvent apparaître à l'utilisateur de la même manière que les applications natives ou les applications mobiles
- Ce type d'applications tente de combiner les fonctionnalités offertes par la plupart des navigateurs modernes avec les avantages de l'expérience offerte par les appareils mobiles
- Une PWA se consulte comme depuis une URL mais permet une expérience utilisateur similaire à celle d'une application mobile, sans les contraintes de cette dernière (soumission aux App-Stores, utilisation importante de la mémoire de l'appareil...)
- https://developer.mozilla.org/fr/docs/Web/Progressive_web_apps

APIs pour les applications

PWA (Progressive Web App) 2/2

Exemples :

- <https://app.starbucks.com/menu>
- <https://2048game.com/>
- <https://housing.com/>

Ressource :

- <https://pwa-workshop.js.org/fr/>
- <https://www.digitalocean.com/community/tutorials/js-vanilla-pwa>

APIs pour les applications

Manifest Web

- fournit des informations concernant une PWA dans un JSON
 - comme son nom, son auteur, une icône et une description
- Le but du manifeste est d'installer des applications sur l'écran d'accueil d'un appareil, offrant aux utilisateurs un accès plus rapide et une expérience plus riche
 - Permet à certains navigateurs d'ajouter au bureau ou à l'écran d'accueil un raccourci vers la PWA
 - Permet à la PWA d'être référencée sur certains apps stores tels que le Windows Store
 - Permet d'afficher la PWA en plein écran, avec un splashscreen lors de l'ouverture de la PWA
- <https://developer.mozilla.org/fr/docs/Web/Manifest>

Cross-site scripting (XSS) 1/2

- Type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page
- Les possibilités des XSS sont très larges
 - l'attaquant peut utiliser tous les langages pris en charge par le navigateur
- De nouvelles possibilités sont régulièrement découvertes
 - Depuis HTML5, Il est possible de rediriger vers un autre site pour de l'hameçonnage ou encore de voler la session en récupérant les cookies
- <https://owasp.org/www-community/attacks/xss/>

Cross-site scripting (XSS) 2/2

- La détection de la présence d'une **faille XSS** peut se faire par exemple en entrant un script Javascript dans un champ de formulaire ou dans une URL :

```
<script> alert("XSS ?")</script>
```

Solutions :

- **obfuscation** : <https://blog.programster.org/add-obfuscation-to-webpack-build>
- **htmlspecialchars()** : https://www.w3schools.com/php/func_string_htmlspecialchars.asp
- **Content Security Policy (CSP)** : <https://developer.mozilla.org/fr/docs/Web/HTTP/CSP>

APIs pour les applications

Les Frameworks...

- Angular
 - <https://angular.io/tutorial>
 - Framework complet (MVC + TypeScript) qui nécessite un temps d'apprentissage non négligeable
- ReactJS
 - <https://fr.reactjs.org/tutorial/tutorial.html>
 - Bibliothèque de base, qui nécessite des bibliothèques tierces (temps d'apprentissage accessible)
- VueJs
 - <https://www.tutorialspoint.com/vuejs/index.htm>
 - Grande personnalisation possible, ressemblant à Angular avec un mode minimal
<https://fr.vuejs.org/v2/guide/comparison.html>

- FIN -