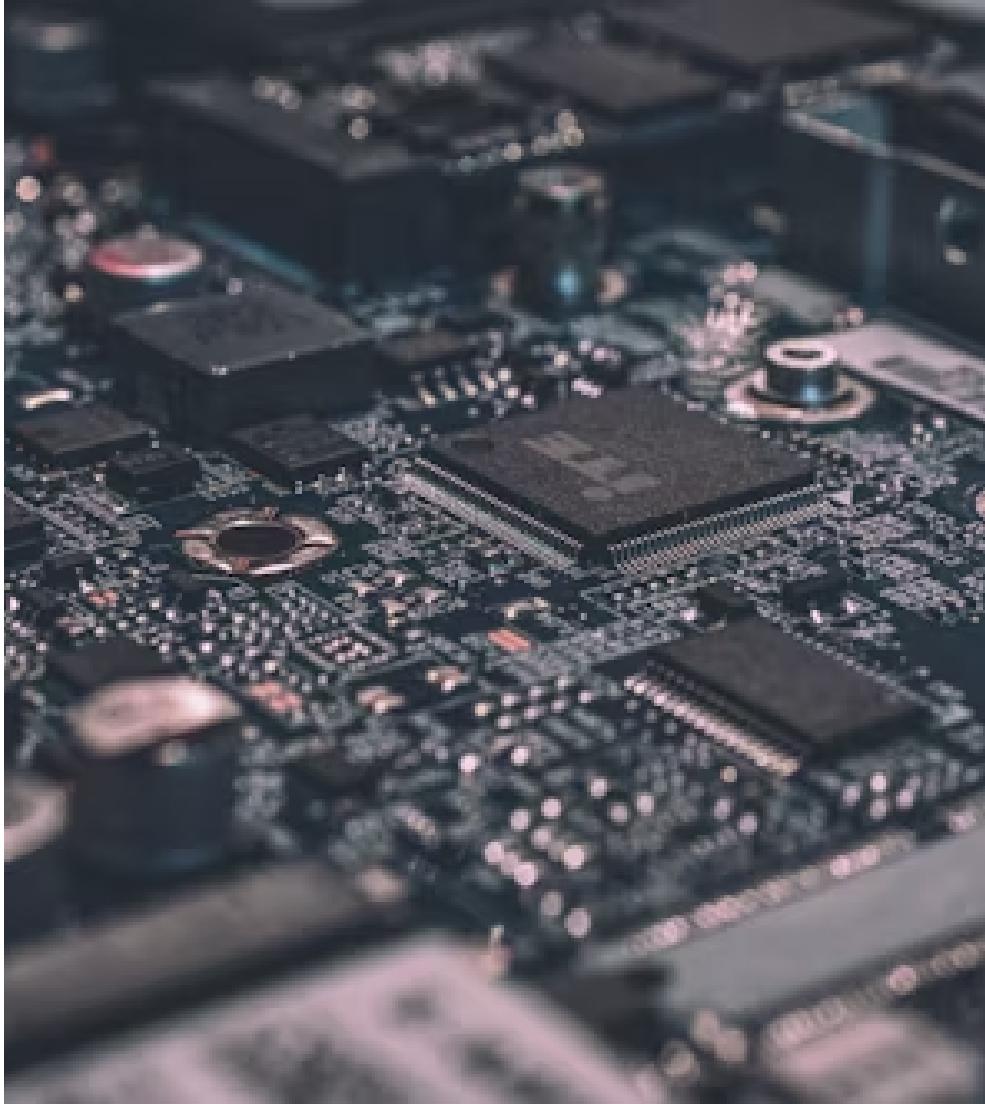
HNU 6058

Université de Montréal

Technologies du web

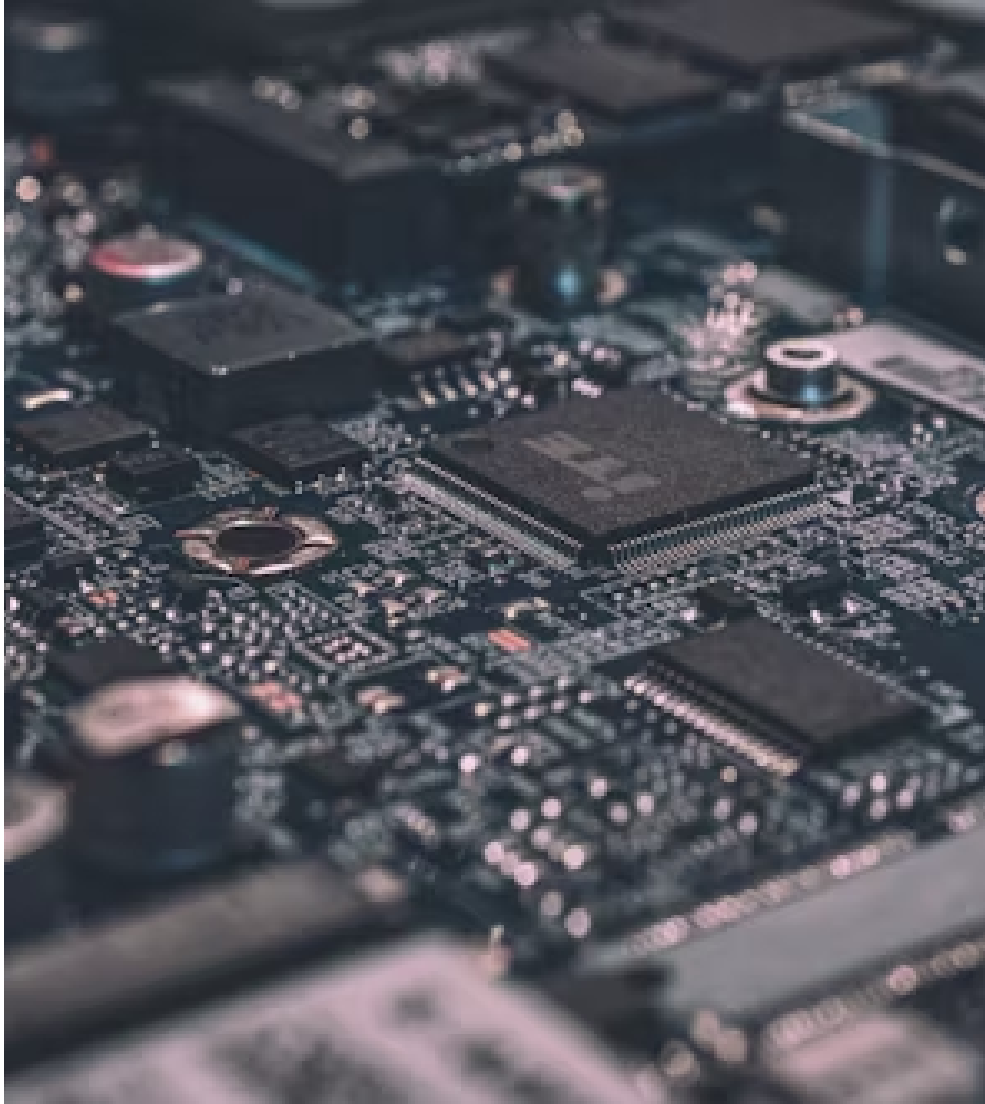
Séance du 21 février 2024

Le site web du cours



Rappel

1. Lier une page css à une page css
2. Les trois principes d’un document css
3. Les trois "règles" possibles pour les boîtes css
4. Changer la couleur d’arrière-plan en css
5. Ajouter un font non-prédéfini à un fichier html
6. Display grid



Pour le rendu final : GitHub Pages

Objectifs de la séance

21 février

- La logique de la programmation en JavaScript
- La logique des feuilles de style en cascade
- La syntaxe de JavaScript
- Atelier JS : création d’un jeu/devinette

- Atelier JS (partie 2) : création d’un jeu/devinette
- Utiliser la fonction fetch pour récupérer des données d’autres sites
- Comprendre la notion de générateur

Un peu de contexte

Historique de JavaScript

Pensé pour permettre d’ajouter des fonctions et des comportements interactifs.

- 1993 : Mosaic
- 1994 : Netscape vs. Microsoft (première browser war – 1995-2001)
- 1995 : nécessité d’un langage de script côté client pour le Web – Java vs. Scheme
- Brendan Eich
- 1996 : JScript "copié" par Microsoft
- 1997 : Netscape vers la standardization avec ECMA International (European Computer Manufacturers Association)

- 1998 : JScript est le standard *de facto*
- 2022 : report Octoverse réalisé par GitHub – « JavaScript continues to reign supreme »
- Frameworks : ex. Angular, React, Vue, Svelte – des boîtes à outils pour le développement Web avec des décisions déjà faites et réfléchies
- ECMA Editions

Dans la pratique

Que peut-on faire avec JS ?

Manipulation des pages Web, Interactions avec l’utilisateur et le serveur Web.

Par exemple, in-browser JavaScript est capable de :

- Ajouter du nouveau HTML à la page, changer le contenu existant, modifier les style
- Réagir aux actions de l’utilisateur, s’exécuter sur les clics de souris, les mouvements du pointeur, les pressions sur les touches

Que peut-on faire avec JS ?

- Envoyer des requêtes sur le réseau à des serveurs distants, télécharger et téléverser des fichiers, charger le contenu d’une page web sans rafraîchir la page
- Obtenir et définir des cookies, poser des questions au visiteur, afficher des messages
- Mémoriser les données du côté client (*Local Storage*)

Lier une page HTML à un script JS

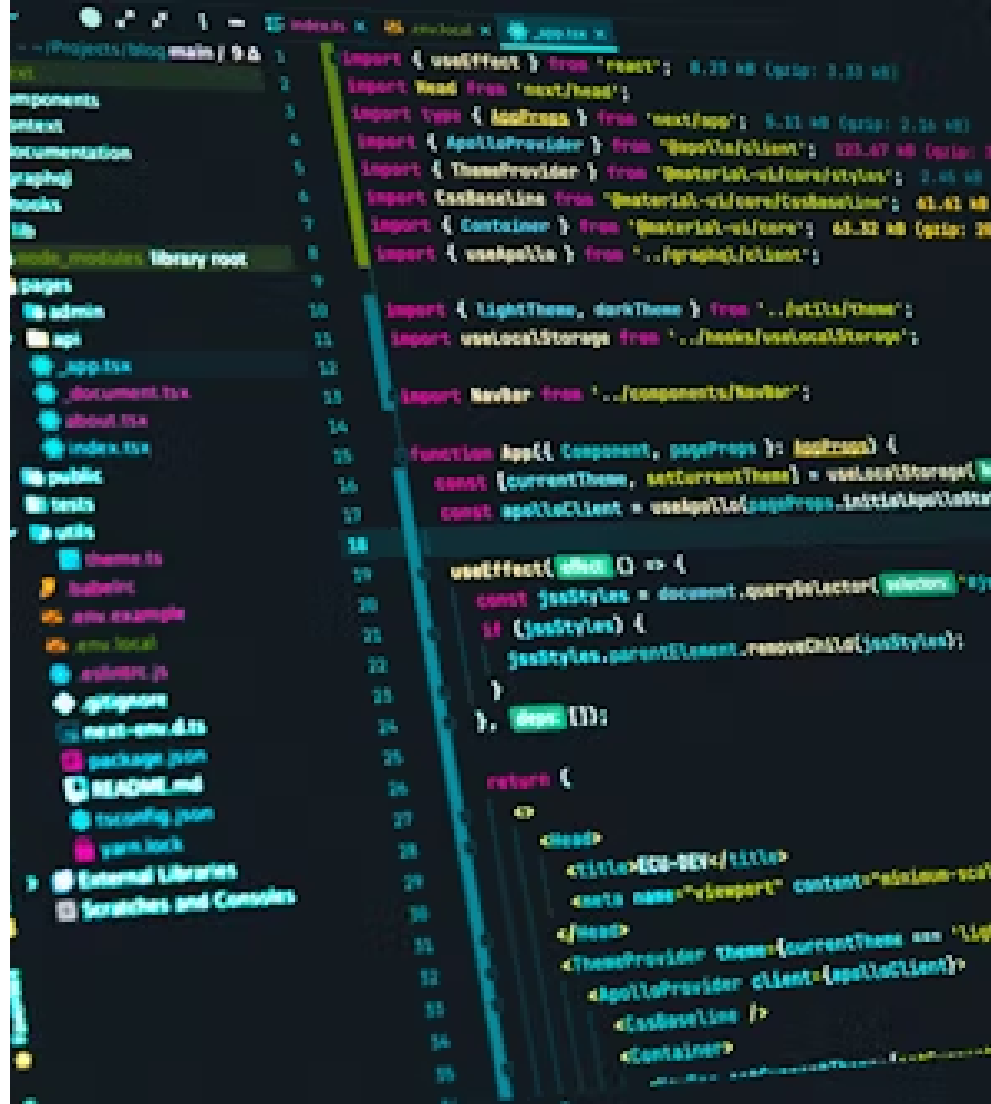
`<script src="/path/to/script.js">`

OU

`<script src="https://link.js">`

OU

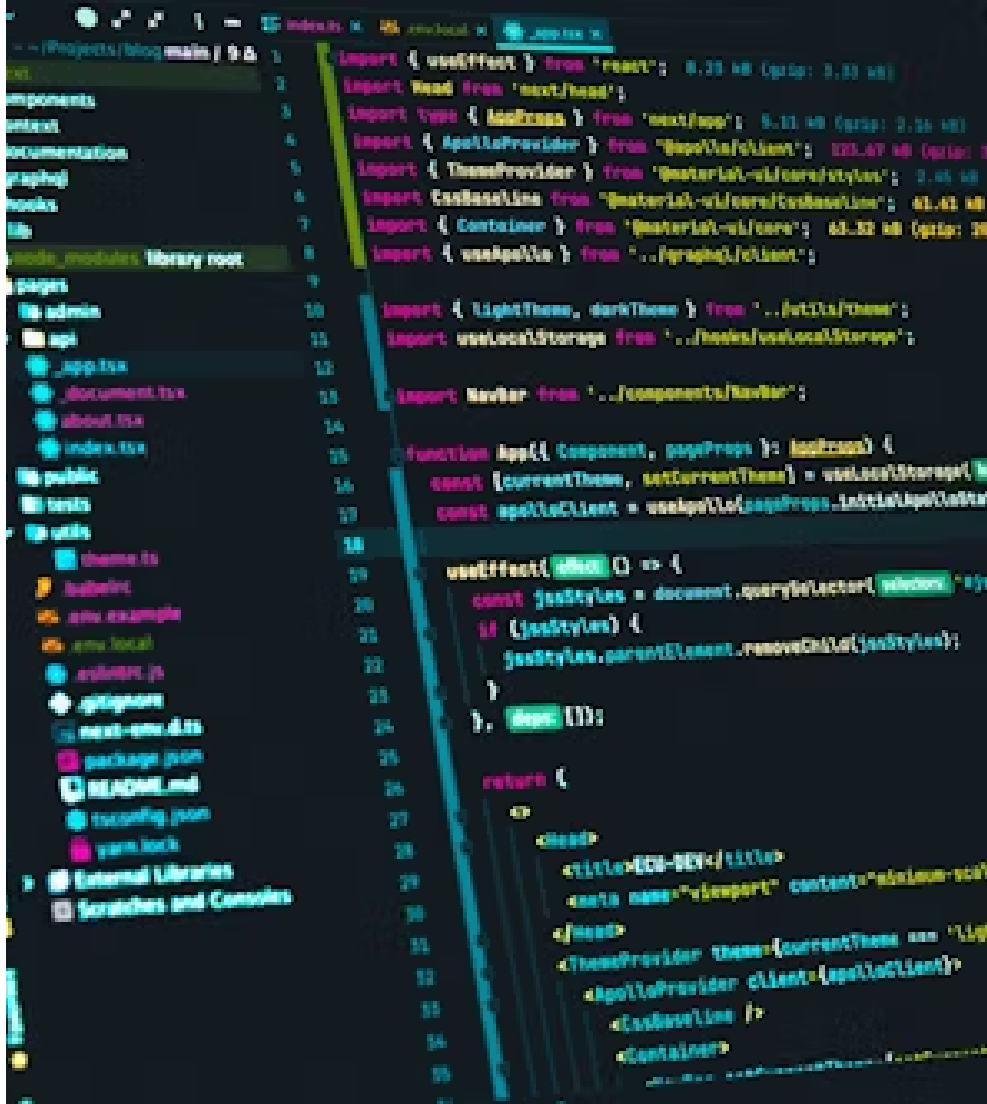
style *embedded* dans l'HTML avec le tag `<script>` Plusieurs scripts peuvent être liés à un seul document HTML



Quelques exemples

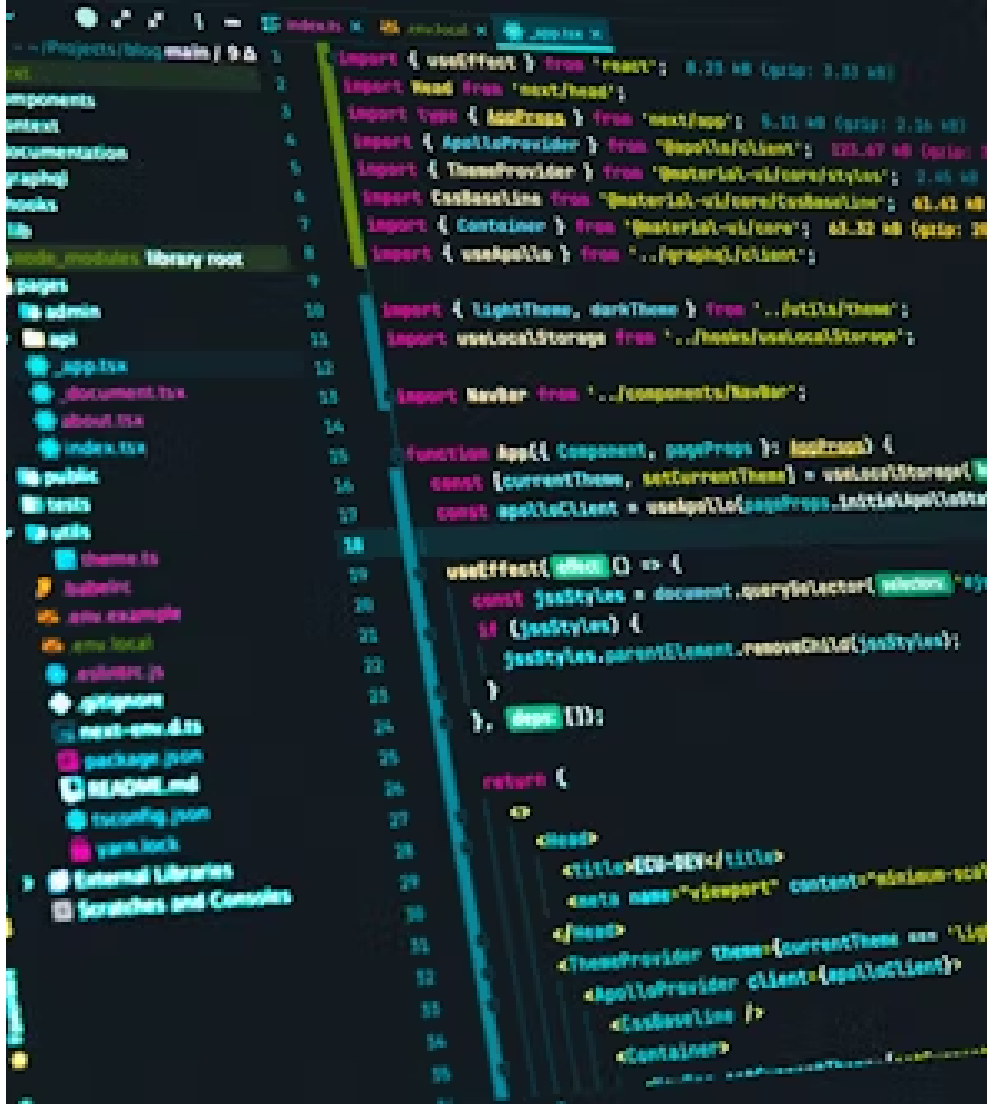
- `alert('hello')`
- `console.log('hello')`
- `document.body.innerHTML = 'hello'`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Un document avec JavaScript</title>

  <!-- référence à un fichier externe -->
  <script src="script.js"></script>
</head>
<body>

  <h1>Un document avec du JavaScript</h1>

  <p>(rechargez-moi pour lancer à nouveau le code)</

</body>
</html>
```



Types de données

- variables : `var message = 'Hello'` ou `let` ou `const`
- `var` / `let` : pour déclarer des variables
- `const` : pour déclarer un valeur qui ne devra plus changer
- chaînes de caractères (strings)
- opérateurs arithmétiques : `= - * /`
- booléennes : `true` ou `false`
- liste de valeurs (arrays), par exemple : `let objets = [tasse, livre, ordinateur, table]`

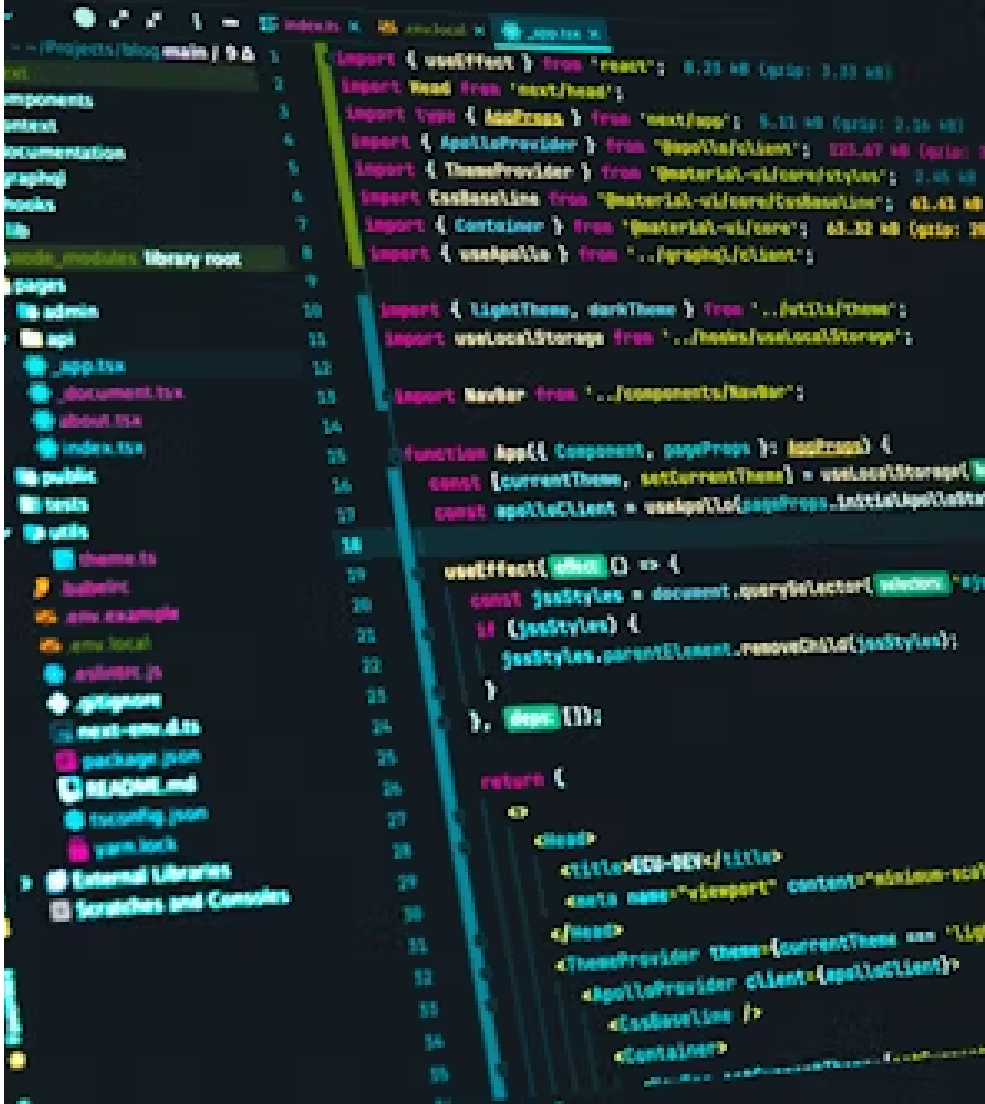


Types de données

- objets : liste de clés associés à des valeurs

```
monObjet = {
  "klé": "valeur",
  "nom": "objet",
  "type" : "typeDeDonnees"
}
```

- `null`
- `undefined`



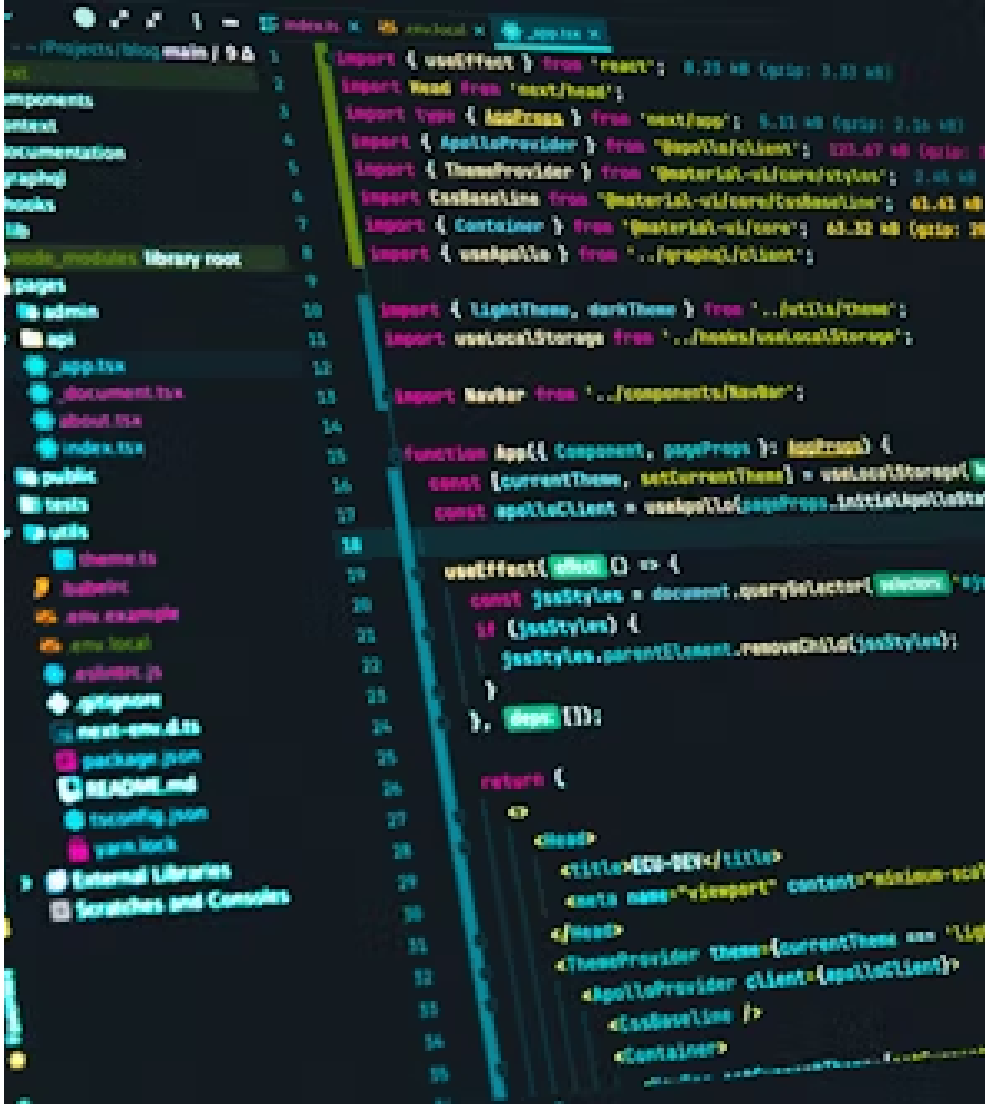
Méthodes

Une fonction se référant directement à un objet

Par exemple `.push()` et `.pop()`

```
var maListe = [1, 2, 3];
var nouvelleListe = maListe.push(4);
```

```
var autreListe = [6, 7, 8];
var autreNouvelleListe = autreListe.pop();
```



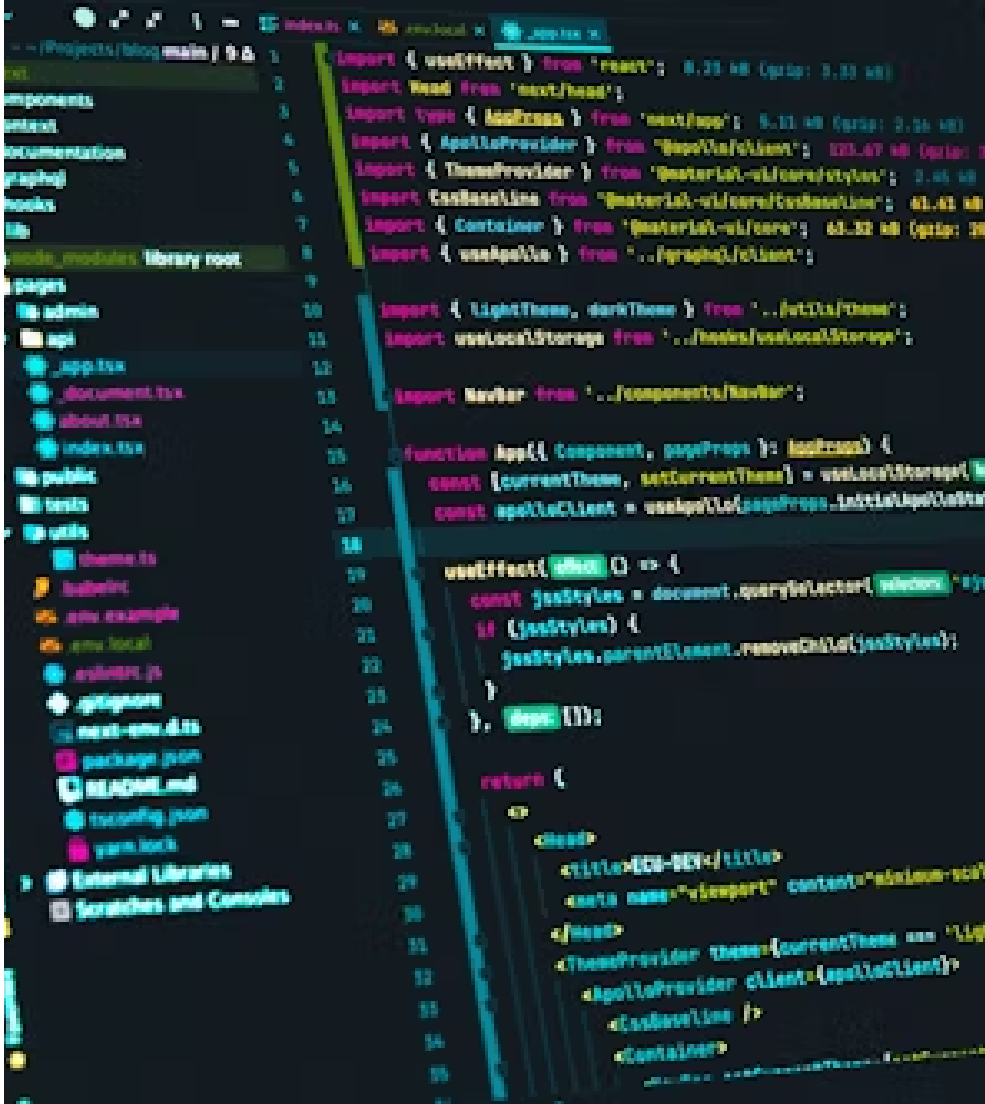
Fonctions

JavaScript est un langage de programmation *fonctionnel*

- définir une fonction
- exécuter une fonction
- une fonction peut avoir des paramètres
- les paramètre peuvent correspondre à des valeurs (arguments)

Par exemple :

```
function direBonjour() {
  alert('Bonjour!');
}
direBonjour();
```



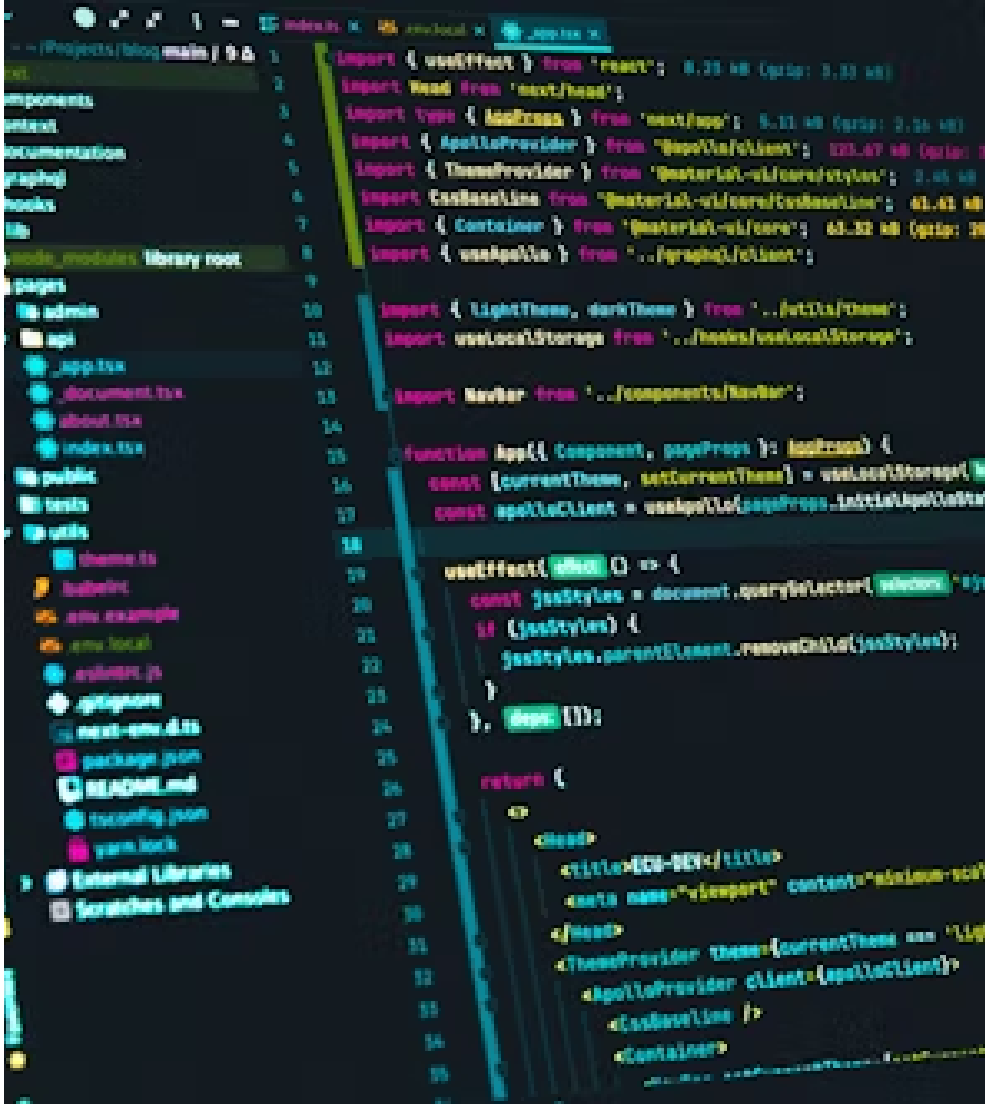
Un deuxième exemple

```
// les fonctions peuvent prendre des *arguments*
// essayons avec votre nom!
function direBonjour(nom) {
  // il faudra faire une concaténation de la chaîne de caractères
  alert('Bonjour ' + nom + '!');
}

// essayons avec un nom de personne!
direBonjour('Nom');
```

Les arguments

Exemple :

```
function direBonjour(nom) {
  alert('Bonjour ' + nom + '!');
}

direBonjour('Alice');
```



Atelier JavaScript

Un jeu devinette en JS – réalisé par Louis-Olivier Brassard

Boucles

Une séquence d’instructions ou de codes répétée jusqu’à l’obtention d’un résultat final.

Exemple :

```
console.log(`Comptons jusqu’à 3… ou 2?`);

for (let i = 0; i < 3; i++) {
console.log(i);
}
```

While : parmi les trois options proposées, laquelle permet d’obtenir le même résultat ?

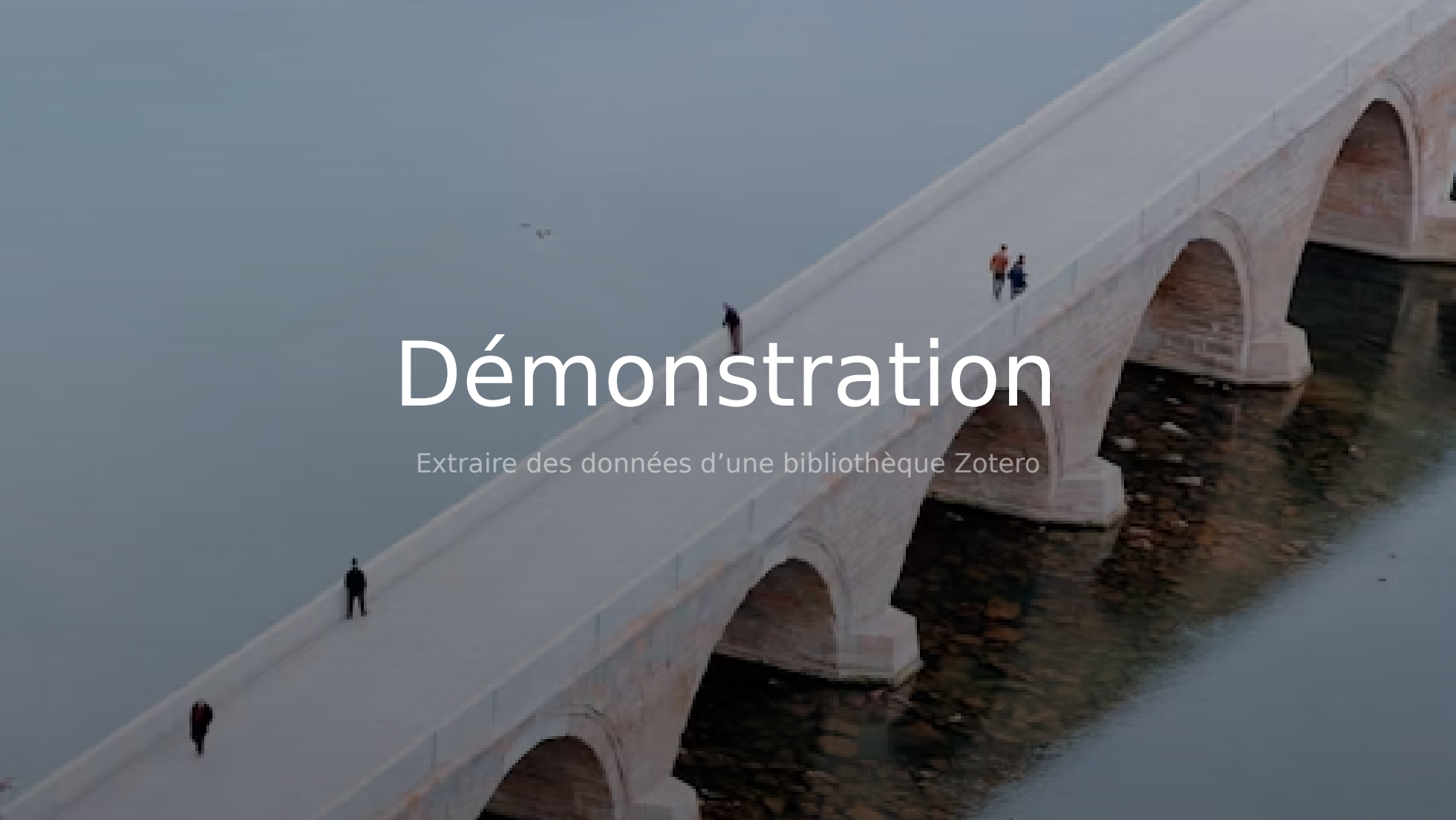
```
let i = 0;
while(i < 3); {
i++;
};
console.log(i)
```

```
let z = 0;
while (z < 3); {
  console.log(z);
  z++;
};
```

```
let y = 0;
while (y < 3) {
  console.log(y)
};
```

if... else

```
if (x < y) {
      reponse = prompt(`Trop petit!`);
    } else {
      reponse = prompt(`Trop grand!`);
    }
```



Démonstration

Extraire des données d’une bibliothèque Zotero

Déclaration des variables

```
const apiUrl = `https://api.zotero.org/groups/4764734/items?key=${taCleZotero}
&format=bib&style=https://raw.githubusercontent.com/citation-style-language/styles/master/apa-5th-edition.csl`;
```

Nous fetcher à l’API Zotero

Et convertir les données obtenues en format texte

```
fetch(apiUrl)
  .then(response => response.text())
```

Afficher notre réponse en correspondence d’un élément HTML

```
  .then(bibliography => {
    const bibliographieIci = document.getElementById('bibliographieIci');
    bibliographieIci.innerHTML = bibliography;
  })
```

Capturer les erreurs éventuelles et les afficher dans la console

```
  .catch(error => console.error('Error:', error));
```

MVC

La plupart des frameworks JavaScript suivent ce schéma.

- **Model** : Le composant central du modèle. Il s’agit de la structure de données dynamique de l’application
- **View** : Toute représentation de l’information
- **Controller** : Accepte les input et les convertit en commandes pour le modèle ou la vue.

Merci !

giulia.ferretti@umontreal.ca