

# Visualisation de données

## 04 - Données

1 mars 2024

Noemi Romano  
noemi.romano@heig-vd.ch



# Cours précédent

# Cours précédent

Syntaxe Chaînage de méthodes (fonctions)

# Cours précédent

**Syntaxe** Chaînage de méthodes (fonctions)

**Manipulation du DOM** `select()`, `.attr()`, `.append()`, `.on("event")`

# Cours précédent

**Syntaxe** Chaînage de méthodes (fonctions)

**Manipulation du DOM** `select()`, `.attr()`, `.append()`, `.on("event")`

**Données** `data(données).join(enter, update, exit)`

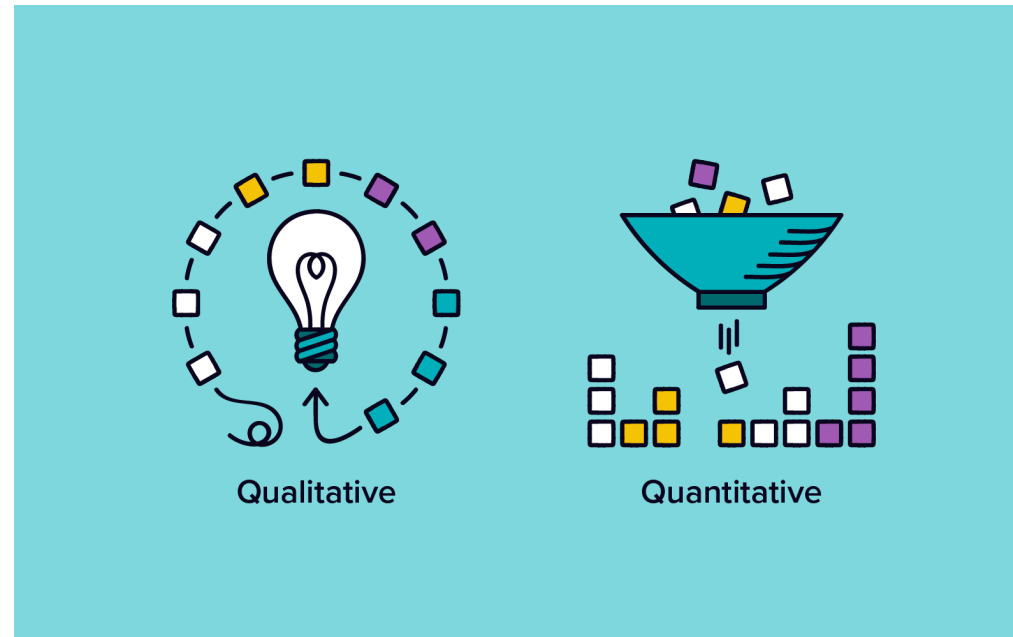
# Types de données

## Qualitatives

- Nominales
- Ordinales

## Quantitatives

- Discrètes
- Continues







# d3-fetch

## Installation

```
npm install d3-fetch
```

# CSV

## (Comma-Separated Values)

```
prenom,anne_naissance
```

```
Alphonse,1932
```

```
Béatrice,1964
```

```
Charlotte,1988
```

# D3

**csv**(input[, init][, row])

# Vite JS

Plugin @rollup/plugin-dsv pour Vite

JS index.js

```
import { csv } from "d3-fetch";

csv("chemin/du/fichier.csv")
  .then(function(data) {
    // Dessiner ici
  })
  .catch(function(error){
    // Gérer les erreurs ici
  })
```



```
$ npm install -D @rollup/plugin-dsv
```

JS vite.config.js

```
import { defineConfig } from 'vite'

import dsv from '@rollup/plugin-dsv'

export default defineConfig({
  plugins: [
    dsv(),
  ],
})
```

JS index.js

```
import data from "chemin/du/fichier.csv"
```

↳ d3-fetch | csv

↳ Vite JS | Rollup Plugin DSV



# JSON

## (Javascript Object Notation)

```
[{  
  "prenom" : "Alphonse",  
  "annee_naissance" : 1932  
},  
{  
  "prenom" : "Béatrice",  
  "annee_naissance" : 1964  
},  
{  
  "prenom" : "Charlotte",  
  "annee_naissance" : 1988  
}]
```

# D3

**json**(input[, init][, row])

```
JS index.js

import { json } from "d3-fetch";

json("chemin/du/fichier.json/ou/url/api")
  .then(function(data) {
    // Dessiner ici
  })
  .catch(function(error){
    // Gérer les erreurs ici
  })
```

# Vanilla JS

```
JS index.js

// Basic GET request
fetch(url)
  .then((response) => {
    // Parser la réponse en json
    return response.json();
  })
  .then((data) => {
    // Dessiner ici
    console.log('Données reçues:', data);
  })
  .catch((error) => {
    // Handle errors
    console.error('Erreur:', error);
  });
```

↳ d3-fetch | json, ↳ MDN | Fetch API



# d3 - array

## Installation

```
npm install d3-array
```



# Statistiques

**max**(iterable[, accessor])

**min**(iterable[, accessor])

**sum**(iterable[, accessor])

**extent**(iterable[, accessor])

**mean**(iterable[, accessor])

↳ d3-array | Summarizing data

```
JS index.js

import { max, min, sum, extent, mean } from "d3-array";

const data = [5, 10, 4, 25];

const maxValue = max(data); // Expected output: 25
const minValue = min(data); // Expected output: 5
const sumValues = sum(data); // Expected output : 44
const extentValues = extent(data); // Expected output : [4, 25]
const meanValues = mean(data); // Expected output : 11
```

# Transformer

## Array.prototype.map(callback)

- **map** : méthode qui crée un nouveau tableau en appliquant une fonction de rappel à chaque élément du tableau d'origine
- **callback** : la fonction à appliquer à chaque élément, prenant en paramètre l'élément actuel et renvoyant la valeur transformée

```
JS index.js

// Exemple de tableau de nombres
const nombres = [1, 2, 3, 4, 5];

// Utilisation de la méthode map pour doubler chaque nombre
const doubles = nombres.map(function(nombre) {
  return nombre * 2;
});

// Affichage du tableau d'origine et du nouveau tableau
console.log("Tableau avec les nombres doublés :", doubles);
//Expected output [2, 4, 6, 8, 10]
```

[↳ MDN Web Docs | Array.Prototype.map\(.\)](#)

# Filtrer

## `Array.prototype.filter(callback)`

- **filter** : méthode qui crée un nouveau tableau en filtrant les éléments du tableau d'origine en fonction d'une condition définie dans la fonction de rappel
- **callback** : la fonction à appliquer à chaque élément, prenant en paramètre l'élément actuel et renvoyant un booléen indiquant s'il doit être inclus dans le nouveau tableau filtré

```
JS index.js

// Exemple de tableau de nombres
const nombres = [1, 2, 3, 4, 5, 6];

// Méthode filter pour filtrer les nombres pairs
const nombresPairs = nombres.filter(function(nombre) {
  return nombre % 2 === 0;
});

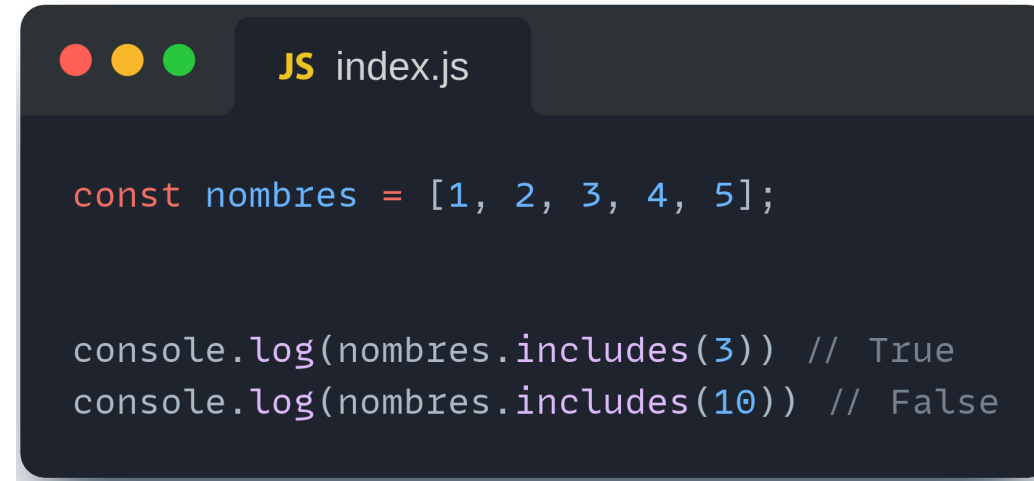
// Affichage du tableau d'origine et du nouveau tableau
console.log("Tableau avec les nombres pairs :", nombresPairs);
//Expected output [2, 4, 6]
```

[↳ MDN Web Docs | Array.prototype.filter\(\)](#)

# Vérifier l'inclusion d'un élément

**Array.prototype.includes(value)**

- **includes** : méthode qui vérifie si un tableau inclut une certaine valeur parmi ses éléments
- **value** : la valeur à rechercher dans le tableau



```
JS index.js

const nombres = [1, 2, 3, 4, 5];

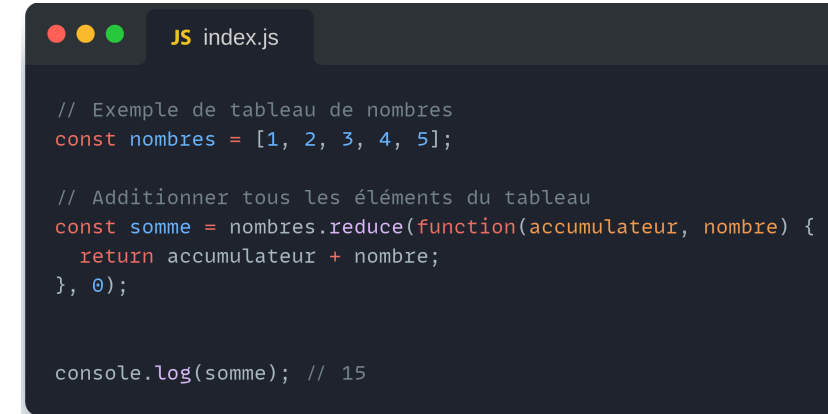
console.log(nombres.includes(3)) // True
console.log(nombres.includes(10)) // False
```

[↳ MDN Web Docs | Array.Prototype.includes\(\)](#)

# Réduire (aggréger)

**Array.prototype.reduce(callback[, initialValue])**

- **reduce** : méthode qui applique une fonction de rappel à chaque élément du tableau, produisant ainsi une seule valeur résultante
- **callback** : la fonction à appliquer, prenant en paramètre un accumulateur et l'élément actuel
- **initialValue** : une valeur optionnelle servant d'accumulateur initial



```
JS index.js

// Exemple de tableau de nombres
const nombres = [1, 2, 3, 4, 5];

// Additionner tous les éléments du tableau
const somme = nombres.reduce(function(accumulateur, nombre) {
  return accumulateur + nombre;
}, 0);

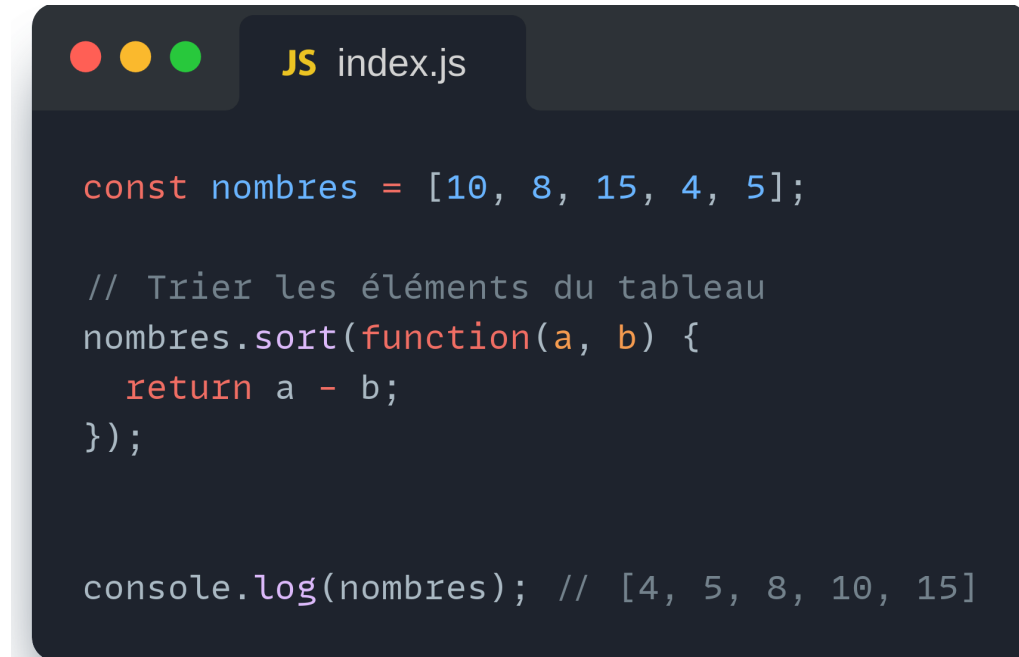
console.log(somme); // 15
```

[↳ MDN Web Docs | Array.Prototype.reduce\(.\)](#)

# Trier

**Array.prototype.sort([, compareFunction])**

- **sort** : méthode qui trie les éléments d'un tableau selon un critère spécifié par une fonction de comparaison
- **compareFunction** : une fonction de comparaison optionnelle déterminant l'ordre de tri

A screenshot of a code editor window titled "JS index.js". The code defines a constant array "nombres" with values [10, 8, 15, 4, 5]. It then uses the "sort" method with a comparison function that returns "a - b". Finally, it logs the sorted array to the console. The code is as follows:

```
const nombres = [10, 8, 15, 4, 5];

// Trier les éléments du tableau
nombres.sort(function(a, b) {
  return a - b;
});

console.log(nombres); // [4, 5, 8, 10, 15]
```

[↳ MDN Web Docs | Array.Prototype.sort\(.\)](#)

# Examples

↳ Observable: d3-fetch, d3-array.