RP - 323 - Programmation fonctionnelle

Auteur : Lefort Julie

Table des matières

Introduction

Objectifs opérationnels

Dans ce module, nous allons apprendre à développer de manière fonctionnelle en Javascript.
Nous allons apprendre les principes et les règles, ainsi que nous exercer au traitement de données grace à des fonctions spécialement utilisées en PF.
Ce RP permettra donc de recenser les choses vues durant ce module, en expliquant les méthodes et en donnant des exemples pour leur utilisation.

Voici les objectifs du module, repris du PR: Introduction à la programmation fonctionnelle :

- Paradigmes de programmation
- Fonctions fléchées

- Définition et utilité de la programmation fonctionnelle Fonctions fondamentales
- Fonction map
- Fonction filter
- Fonction reduce Concepts de programmation fonctionnelle
- Fist class citizen
- Fonctions lambda
- Immuabilité
- Fonctions pures
- Composition de fonctions :
    - Fonctions unaires
    - Currying
    - Closure
    - Fonction pipe
- Récursion
- Builder pattern
- Refactorisation

Opérateurs javascript

opérateur `?:`

> L'expression `question?valeur1:valeur2` retournera `valeur1` si `question` vaut `true` sinon elle retournera `valeur2`.

```
const age = 15;
const resultat = age >= 18 ? 'majeur' : 'mineur'; // 'mineur'
```

opérateur `??`

Cet opérateur logique se nomme l'opérateur de "coalescence des nuls".

> Renvoie son opérande de droite lorsque son opérande de gauche vaut `null` ou `undefined` et qui renvoie son opérande de gauche sinon.

```
const foo1 = null ?? 'default'; // "default"
const foo2 = 0 ?? 42; // 0
```

> [!CAUTION] Contrairement à l'opérateur logique OU (`||`), l'opérande de gauche sera également renvoyé s'il s'agit d'une valeur équivalente à `false` et pas seulement `null` et `undefined`.
>
> ⚠ En d'autres termes **ATTENTION** ‼ lors de l'utilisation de `||` pour fournir une valeur par défaut à une variable, car on peut rencontrer des comportements inattendus lorsqu'on considère certaines valeurs comme correctes et utilisables (par exemple une chaine vide `''` ou `0`) ‼

```
const foo3 = 0 || 42; // 42 => ATTENTION !
const foo4 = 1 || 42; // 1
const foo5 = null || 'salut !'; // 'salut !'
const foo6 = '' || 'salut !'; // 'salut !' => ATTENTION !
```

opérateur ??=

Cet opérateur logique se nomme l'opérateur d'affectation de "coalescence des nuls", également connu sous le nom d'opérateur affectation logique nulle.

> Évalue l'opérande de droite et l'attribue à gauche **UNIQUEMENT si l'opérande de gauche est nulle** (`null` ou `undefined`).

```
const a = { duration: 50 };
a.duration ??= 10; // pas fait
a.speed ??= 25; // fait => { duration: 50, speed: 25 }
```

opérateur de décomposition 'spread' ...

L'opérateur de décomposition spread `...` permet de décomposer un itérable (comme un tableau) en en ses éléments distincts. Cela permet de rapidement copier tout ou une partie d'un tableau existant dans un autre tableau ou d'en extraire facilement des parties.

```
// Combiner des valeurs existantes dans un nouveau tableau
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];

// Extraire uniquement ce qui est utile d'un tableau
const numbers = [1, 2, 3, 4, 5, 6];
const [one, two, ...rest] = numbers;

// Mariage d'objets avec mise à jour :-)
const myVehicle = {
    brand: 'Ford',
    model: 'Mustang',
    color: 'red',
};
const updateMyVehicle = {
    type: 'car',
    year: 2021,
    color: 'yellow',
};
const myUpdatedVehicle = { ...myVehicle, ...updateMyVehicle };
```

Déstructuration

L'opérateur de décomposition spread `...` sert aussi à isoler certains éléments afin de les utiliser ensuite, et de **mettre le reste** d'un coup ailleurs.

```
const valeurs = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const [a, b, ...c] = valeurs;
console.log(a); // 1
console.log(b); // 2
console.log(c); // [3, 4, 5, 6, 7, 8, 9, 10]
```

Date et Heure

Lien vers la documentation officielle :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Date

Obtenir la date et/ou heure actuelle

```
const maintenant = new Date(); // Obtenir l'un comme l'autre

console.log(maintenant.toLocaleDateString()); // ex: "06.06.2025"
console.log(maintenant.toLocaleTimeString()); // ex: "15:23:42"

const jour = maintenant.getDate();
const mois = maintenant.getMonth() + 1; // Attention : janvier = 0
const annee = maintenant.getFullYear();
const heure = maintenant.getHours();
const minute = maintenant.getMinutes();
const seconde = maintenant.getSeconds();
console.log(`${jour}/${mois}/${annee} - ${heure}h${minute}`);

// Au format ISO (standard international)
console.log(maintenant.toISOString()); // ex: "2025-06-06T13:23:42.123Z"
```

Math

Lien vers la documentation officielle :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Math

`Math.PI` - la constante π

Représente la constante Pi (environ 3.14159)

```
console.log(Math.PI);

// Resultat
3.141592653589793
```

`Math.abs()` - la |valeur absolue| d'un nombre

Retourne la valeur absolue d'un nombre (pas de valeur négative)

```
function difference(a, b) {
  return Math.abs(a - b);
}

console.log(difference(3, 5));

// Resultat
2
```

`Math.pow()` - élever à une puissance

Permet d'élever un nombre à une puissance. Le premier nombre est l'indice et le 2eme est l'exposant

```
console.log(Math.pow(7, 3));

// Resultat
343
```

`Math.min()` - plus petite valeur

Renvoie la plus petite valeur d'une collection

```
console.log(Math.min(2, 3, 1));

// Resultat
1
```

`Math.max()` - plus grande valeur

Renvoie la plus grande valeur d'une collection

```

```

```
console.log(Math.max(2, 3, 1));

// Resultat
3
```

`Math.ceil()` - arrondir à la prochaine valeur entière la plus proche

Arrondit à la valeur entière la plus proche, mais seulement vers le haut

```
console.log(Math.ceil(7.004));

// Resultat
8
```

`Math.floor()` - arrondir à la précédente valeur entière la plus proche

Arrondit à la valeur entière la plus proche, mais seulement vers le bas

```
console.log(Math.floor(7.95));

// Resultat
7
```

`Math.round()` - arrondir à la valeur entière la plus proche

Permet d'arrondir un nombre à la valeur entière la plus proche.
Exemple pour calculer une note d'école arrondie:

```
Math.round((48 / 52) * 5 + 1)

// Resultat
6
```

`Math.trunc()` - supprime la virgule et retourne la partie entière d'un nombre

```
Math.trunc(13.37)

// Résultat
13
```

`Math.sqrt()` - la raçine carrée d'un nombre

Fais la racine carrée d'un nombre

```
Math.sqrt(4)

// Resultat
2
```

`Math.random()` - générer un nombre aléatoire entre 0.0 (compris) et 1.0 (non compris)

Génère une valeur aléatoire entre 0.0 et 1, peut ensuite être adaptée pour renvoyer d'autres valeurs

```
function getRandomInt(max) {
  return Math.floor(Math.random() * max);
}

console.log(getRandomInt(3));
// Resultat
1, 2 ou 3
```

JSON

Lien vers la documentation officielle :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/JSON

`JSON.stringify()` - transformer un objet Javascript en JSON

Convertit une valeur JavaScript en une chaîne JSON, en remplaçant éventuellement les valeurs si une fonction de remplacement est spécifiée ou en incluant éventuellement uniquement les propriétés spécifiées si un tableau de remplacement est spécifié.

```
console.log(JSON.stringify({ x: 5, y: 6 }));

// Résultat
{"x":5,"y":6}
```

`JSON.parse()` - transformer du JSON en objet Javascript

Analyse une chaîne JSON et construit la valeur ou l'objet JavaScript correspondant. Une fonction de restauration optionnelle peut être fournie pour effectuer une transformation sur l'objet résultant avant son renvoi.

```
const json = '{"result":true, "count":42}';
const obj = JSON.parse(json);

console.log(obj.count);

console.log(obj.result);

// Résultat
42
true
```

Chaînes de caractères

Lien vers la documentation officielle :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/String

`split()` - un ciseau qui coupe une chaîne là où un caractère apparaît et produit un tableau

Permet de séparer un string json en fonction du charatère spécifié et renvoie un nouveau tableau.

```
const str = "The quick brown fox jumps over the lazy dog.";

const words = str.split(" ");
console.log(words[3]);

// Résultat
fox
```

`trim()`, `trimStart()` et `trimEnd()` - épuration des espaces en trop dans une chaîne (trimming)

Supprime les espaces avant et après un String et renvoie le résultat.
trimEnd supprime les espace seulement à la fin et trimStart seulement au début.

```
const greeting = "   Hello world!   ";

console.log(greeting.trim());
console.log(greeting.trimStart());
console.log(greeting.trimEnd());

// Résultat
"Hello World"
"Hello World   "
"   Hello World"
```

`padStart()` et `padEnd()` - aligner le contenu dans une chaîne de caractères

Permet de rallonger les String avec un charactère donné pour que le String atteigne une longueur précise. padStart() rallonge le début et padEnd() rallonge la fin.

```
const str1 = "Breaded Mushrooms";

console.log(str1.padEnd(25, "."));
console.log(str1.padStart(25, "."));

// Résultat
"Breaded Mushrooms........"
"........Breaded Mushrooms"
```

Console

Lien vers la documentation officielle : https://developer.mozilla.org/fr/docs/Web/API/console

`console.log()` - Afficher un message sur la console

```
console.log('Coucou !'); // Coucou !
```

`console.info()`, `warn()` et `error()` - Afficher un message sur la console (filtrables)

Permet d'afficher des messages précis sur la console.

- info(): message d'information
- warn(): avertissement

- error(): erreurs

```
console.error("Erreur");
console.warn("Avertissement");
console.info("Info");
```

`console.table()` - Afficher tout un tableau ou un objet sur la console

Affiche les données tabulaires sous forme de tableau.

```
console.table(data)
console.table(data, columns)
```

`console.time()`, `timeLog()` et `timeEnd()` - Chronométrer une durée d'exécution

Permet de calculer combien de temps est nécéssaire pour exécuter une opération. time(): démarre le timer timeLog(): affiche le temps écoulé timeEnd(): arrête le timers

```
console.time('boucle');
for (let i = 0; i < 1_000_000; i++) {
    Math.sqrt(i);
    if (i === 500_000) console.timeLog('boucle');
}

console.timeEnd('boucle');

// Exemple de résultat
boucle: 12.34ms
boucle: 25.67ms
```

Tableaux

Lien vers la documentation officielle :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array

`forEach` - parcourir les éléments d'un tableau

Exécute une fonction définie dans le code sur chaque élément d'un tableau.

```
const array = ["a", "b", "c"];

array.forEach((element) => console.log(element));

// Résultat
"a"
"b"
"c"
```

`entries()` - parcourir les couples index/valeurs d'un tableau

Permet de parcourir les couples index/valeurs d'un tableau en créant un nouvel itérateur qui contient les clé valeurs.

```
const array = ["a", "b", "c"];

const iterator = array.entries();

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

// Résultat
Array [0, "a"]
Array [1, "b"]
```

`in` - parcourir les clés d'un tableau

Permet de parcourir les clés d'un tableau afin de vérifier si un index existe.

```
const arr = ["a", "b", "c"];

console.log(1 in arr);
console.log(3 in arr);
console.log("a" in arr);

// Résultat
true
false
false
```

`of` - parcourir les valeurs d'un tableau

Permet de parcourir les valeurs du tableau dans une boucle for ... of.

```
const arr = ["a", "b", "c"];

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

// Résultat
a
b
c
```

`find()` - premier élément qui satisfait une condition

Parcours les éléments du tableau etourne le premier élément qui satisfait la condition établie.

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

const found = array.find((element) => element > 10);

console.log(found);

// Résultat
12
```

`findIndex()` - premier index qui satisfait une condition

Retourne le premier index qui satisfait la condition établie en parcourant le tableau.

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

const isLargeNumber = (element) => element > 13;

console.log(array.findIndex(isLargeNumber));

// Résultat
3
```

`indexOf()` et `lastIndexOf()` - premier/dernier élément qui correspond

Renvoie le premier ou le dernier index où se trouvent l'élément recherché en fonction de l'index de départ spécifié.
Si il n'est pas trouvé, il renvoie -1.

```
const beasts = ["ant", "bison", "camel", "duck", "bison"];
```

```
console.log(beasts.indexOf("bison"));
// Expected output: 1

// Cherche depuis l'index 2
console.log(beasts.indexOf("bison", 2));
// Expected output: 4

console.log(beasts.indexOf("giraffe"));

console.log(beasts.lastIndexOf("bison"));
// Expected output: -1

// Résultat
1
2
4
-1
4
```

`push()`, `pop()`, `shift()` et `unshift()` - ajouter/supprime au début/fin dans un tableau

- push() : ajoute un élément à la fin d'un Array et renvoie la nouvelle longueur de l'Array
- pop() : supprime le dernier élément d'un tableau et renvoie cet élément.
- shift() : supprime le premier élément d'un tableau et renvoie cet élément.
- unshift() : ajoute l'élément spécifié au début d'un Array et renvoie la nouvelle longueur

```
// PUSH
  const animals = ["pigs", "goats", "sheep"];

  const count = animals.push("cows");
  console.log(count);

  console.log(animals);

  // Résultat
  Array ["pigs", "goats", "sheep", "cows"]
// POP
  const plants = ["broccoli", "cauliflower", "cabbage", "kale", "tomato"];

  console.log(plants.pop());

  console.log(plants);

  // Résultat
  "tomato"
  Array ["broccoli", "cauliflower", "cabbage", "kale"]

// SHIFT
  const array = [1, 2, 3];
```

```
  const firstElement = array.shift();

  console.log(array);

  console.log(firstElement);

  // Résultat
  Array [2, 3]
  1

// UNSHIFT
  const array = [1, 2, 3];

  console.log(array.unshift(4, 5));

  console.log(array);

  // Résultat
  5
  Array [4, 5, 1, 2, 3]
```

`slice()` - ne conserver que certaines lignes d'un tableau

Retourne une copie de seulement une partie du tableau dans un nouveau tableau sélectionné du début à la fin. Le tableau original n'est pas modifié

```
const animals = ["ant", "bison", "camel", "duck", "elephant"];

console.log(animals.slice(2));

console.log(animals.slice(2, 4));

// Résultat
Array ["camel", "duck", "elephant"]
Array ["camel", "duck"]
```

`splice()` - supprimer/insérer/remplacer des valeurs dans un tableau

Modifie le contenu d'un tableau en supprimant ou en remplaçant les éléments existants et/ou en ajoutant de nouveaux éléments à leur place

```
const months = ["Jan", "March", "April", "June"];
months.splice(1, 0, "Feb");
// Insère à l'index 1

console.log(months);

//Résultat
```

```
Array ["Jan", "Feb", "March", "April", "June"]
```

`concat()` - joindre deux tableaux

Permet de fusionner plusieurs tableaux ensemble. Cette méthode ne modifie pas les tableaux de base mais en retourne un nouveau

```
const array1 = ["a", "b", "c"];
const array2 = ["d", "e", "f"];
const array3 = array1.concat(array2);

console.log(array3);

// Résultat
Array ["a", "b", "c", "d", "e", "f"]
```

`join()` - joindre des chaînes de caractères

Permet de transformer un tableau en une seule chaîne, en insérant un séparateur entre chaque élément.

```
const elements = ["feu", "eau", "vent"];

console.log(elements.join(", "));
console.log(elements.join(" / "));
console.log(elements.join(""));

// Résultat
"feu, eau, vent"
"feu / eau / vent"
"feueauvent"
```

`keys()` et `values()` - les clés/valeurs d'un objet

- keys() retourne toutes les clés d'un objet.
- values() retourne toutes les valeurs d'un objet.

```
const obj = { nom: "Julie", age: 21 };

console.log(Object.keys(obj));
console.log(Object.values(obj));

// Résultat
["nom", "age"]
["Julie", 21]
```

`includes()` - vérifier si une valeur est présente dans un tableau

Vérifie si une valeur est présente dans un tableau et renvoie true ou false en fonction du résultat.

```
const array = [1, 2, 3];
const pets = ["cat", "dog", "bat"];

console.log(array.includes(2));

console.log(pets.includes("cat"));

console.log(pets.includes("at"));

// Résultat
true
true
false
```

`every()` et `some()` - vérifier si plusieurs valeurs sont toutes/quelques présentes dans un tableau

Vérifier la présence de valeurs dans un tableau. every() renvoie true si toutes les cases remplissent la condition, alors que some() renvoie true si au moins 1 case rempli la condition.

```
const isBelowThreshold = (currentValue) => currentValue < 40;

const array1 = [1, 30, 67, 29, 10, 13];

console.log(array1.every(isBelowThreshold));

console.log(array1.some(isBelowThreshold));
// Résultat
false
true
```

`fill()` - remplir un tableau avec des valeurs

Permet de remplir un tableau avec des valeurs en spécifiant l'index de départ et l'index d'arrêt.

```
const array = [1, 2, 3, 4];

// Remplir avec des 0 de l'index 2 à 4
console.log(array.fill(0, 2, 4));

// Remplir de 5 depuis l'index 1
console.log(array.fill(5, 1));
```

```
// Remplir de 6
console.log(array.fill(6));

// Résultat
Array [1, 2, 0, 0]
Array [1, 5, 5, 5]
Array [6, 6, 6, 6]
```

`flat()` - aplatir un tableau

Crée un nouveau tableau dans lequel tous les éléments des sous-tableaux sont concaténés récursivement jusqu'à la profondeur spécifiée.

```
const arr1 = [0, 1, 2, [3, 4]];

console.log(arr1.flat());

// Résultat
Array [0, 1, 2, 3, 4]
```

`sort()` - pour trier un tableau

Permet de trier les objets du tableau par odre alphabétique ou ordre croissant pour les chiffres.

```
// Code
const months = ["March", "Jan", "Feb", "Dec"];
months.sort();
console.log(months);
// OU months.sort((a, b) => a.month.localeCompare(b.month));

const array = [1, 30, 4, 21, 100000];
array.sort();
console.log(array);

// Résultat
Array ["Dec", "Feb", "Jan", "March"]
Array [1, 100000, 21, 30, 4]
```

`map()` - tableau avec les résultats d'une fonction

Permet de gérer l'affichage de données (tableaux, JSON, etc.)

```
// Données
const dataMotos = [
    {
```

```
        marque: "Honda",
        modeles: [
            { nom: "CB750 Four", annee: 1969, prix: 12000, puissance: 67, couple:
60 },
            { nom: "Africa Twin CRF1100L", annee: 2020, prix: 15000, puissance:
102, couple: 105 },
            { nom: "CBR1000RR Fireblade", annee: 2024, prix: 23000, puissance:
218, couple: 113 },
            { nom: "Gold Wing", annee: 2024, prix: 33000, puissance: 126, couple:
170 },
        ]
    }, ... ]

//Code
function actionM1() {
    const resultat = dataMotos.map((moto) => moto.marque);

    afficherObjet(resultat);
}

//Résultat
[
   "Honda",
   "Yamaha",
   "Ducati",
   "BMW",
   "Harley-Davidson"
]
```

`filter()` - tableau avec les éléments passant un test

Permet de filtrer des données afin d'obtenir seulement celles dont on a besoin

```
// Données
const dataVilles = [
    // FRIBOURG (FR)
    { ville: "Fribourg", canton: "FR", habitants: 39000 },
    { ville: "Bulle", canton: "FR", habitants: 24400 },
    { ville: "Villars-sur-Glâne", canton: "FR", habitants: 12500 },
    ... ]
// Code
function actionF2() {

    const resultat = dataVilles.filter(ville => ville.habitants > 10000 &&
ville.habitants < 25000 && ville.canton === "FR");

    afficherObjet(resultat);
}

// Résultat
[
```

```
    {
        "ville": "Bulle",
        "canton": "FR",
        "habitants": 24400
    },
    {
        "ville": "Villars-sur-Glâne",
        "canton": "FR",
        "habitants": 12500
    }
]
```

`groupBy()` - regroupe les éléments d'un tableau selon un règle

Permet de grouper les éléments d'un tableau en fonction d'une colonne.

```
// Code
const inventory = [
  { name: "asparagus", type: "vegetables", quantity: 9 },
  { name: "bananas", type: "fruit", quantity: 5 },
  { name: "goat", type: "meat", quantity: 23 },
  { name: "cherries", type: "fruit", quantity: 12 },
  { name: "fish", type: "meat", quantity: 5 },
];

const result = Object.groupBy(inventory, ({ quantity }) =>
  quantity < 6 ? "restock" : "sufficient",
);

// Resultat
Array [Object { name: "bananas", type: "fruit", quantity: 5 }, Object { name:
"fish", type: "meat", quantity: 5 }]
```

`flatMap()` - chaînage de map() et flat()

Permet de créer un nouveau tableau à partir du résultat d'un traitement d'élément.

```
// Code
const resultat = dataMotos.flatMap((moto) => moto.modeles.map((modele) =>
`${moto.marque} / ${modele.nom}`)).sort();

// Résultat
[
    "BMW / K1600GTL",
    "BMW / R nineT",
    "BMW / R1250GS",
    "BMW / S1000RR",
    ...
]
```

`reduce()` et `reduceRight()` - réduire un tableau à une seule valeur

Exécute une fonction décrite par l'utilisateur sur chaque élément du tableau et retourne une seule valeur à la fin.
`reduceRight()` permet de faire la même chose, mais de droite à gauche (`reduce` le fait de gauche à droite).

```
// Code
const resultat = dataEvaluations.reduce((petit, note) => (note.date < petit.date ?
{ ...note } : petit), dataEvaluations[0]);

    afficherObjet(resultat.date);

// Resultat
```

`reverse()` - inverser l'ordre du tableau

Inverse l'ordre des éléments directement dans le tableau original.

```
const arr = [1, 2, 3];

console.log(arr.reverse());

// Résultat
[3, 2, 1]
```

Techniques

``(backticks) - pour des expressions intelligentes

Permet de créer des chaînes dynamiques, avec des expressions ${...} et du texte sur plusieurs lignes.

```
const nom = "Julie";
const age = 21;

const message = `Je m'appelle ${nom} et j'ai ${age} ans.`;

console.log(message);
```

```
// Résultat
"Je m'appelle Julie et j'ai 21 ans."
```

`new Set()` - pour supprimer les doublons

Crée un nouveau tableau sans doublon.

```
// Code
const resultat = [...new Set(dataEvaluations.map((e) => e.nom + ' ' +
e.prenom))].sort();
    afficherObjet(resultat);

// Resultat
[
   "D'ŒUF John",
   "FICE Eddy",
   "HARONI Mac",
   "NISSENS Remy",
   "RIQUE Théo",
   "TARISTE Guy",
   "TERNET Alain",
   "TERRIEUR Alain",
   "TERRIEUR Alex",
   "VOYANTE Claire"
]
```

Fonctions

Déclaration de fonction

Standard

```
function doStuff(a, b, c) {
    return a + b + c;
}
```

Sous forme d'expression de fonction

```
const doStuff = function (a, b, c) {
    return a + b + c;
};
```

Sous forme d'expression de fonction anonyme

```
const doStuff = (a, b, c) => {
    return a + b + c;
};
```

Sous forme raccourcie

S'il n'y a qu'un seul argument et que son corps n'a qu'une seule expression, on peut omettre le return et le corps de la fonction :

```
const doStuff = (a) => `Salut ${a} !`;
```

Fonctions immédiatement invoquées (IIFE)

IIFE = Immediately Invoked Function Expressions.

Ces fonctions sont définies et **exécutées immédiatement**. Elles sont souvent utilisées pour créer un **contexte isolé** ou encapsuler du code sans polluer l'espace global.

```
(function(){ ... })()
```

ou

```
(() => { ... })()
```

Conclusion

Ce que j'ai appris

Dans ce module, nous avons appris à utiliser des fonctions de programmation fonctionnelle, ce qui est très nouveau dans notre façon de coder. Nous avons aussi pu revoir des notions javascript que nous avions déjà apprises lors d'autres modules.

Mes points forts

Je pense avoir bien compris la plupart des notions vues en classe comme le pipe, les buildeer pattern, map(), filter() etc. J'ai trouvé que j'arrivai bien à implémenter ce qui était demandé.

Mes points faibles

J'ai encore un peu de peine à utiliser les reduce(), surtout la logique de déconstruction des données.