🤔 RP - 323 - Programmation fonctionnelle

> [!TIP] **Référence Javascript:** https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference
> **Tester du code JS** : https://runjs.app/play
> **Convertir en PDF** : https://marketplace.visualstudio.com/items?itemName=manuth.markdown-converter

Table des matières

Introduction

Ce module m'a permis de découvrir la programmation fonctionnelle en JavaScript à travers des cas concrets, notamment l'analyse d'un gros jeu de données de notes scolaires (`jsonData`).
Les objectifs principaux pour moi :

- Comprendre la différence entre approche impérative et fonctionnelle.
- Manipuler les tableaux avec `map`, `filter`, `reduce`, etc.
- Limiter les effets de bord en privilégiant des fonctions pures.
- Travailler avec des données réelles (notes, élèves, branches) et en extraire des infos utiles.
- Gagner en lisibilité, maintenabilité et réutilisabilité dans mon code.

L'idée générale : prendre des données comme celles de `jsonData.evaluations` et, grâce aux outils Javascript modernes, construire rapidement des statistiques, tris, regroupements et filtrages sans tout recoder à la main avec des boucles partout.

Opérateurs javascript super-cooool 😎

opérateur `?:`

Expression conditionnelle courte.

```
const note = 5.2;
const resultat = note >= 4 ? 'réussi' : 'échec'; // 'réussi'
```

opérateur `??`

Renvoie la valeur de droite seulement si celle de gauche vaut `null` ou `undefined`.

```
const branche = null ?? 'Inconnue'; // 'Inconnue'
const note = 0 ?? 4;                // 0 (0 est accepté)
```

⚠️ Contrairement à `||`, les valeurs `0`, `''` ou `false` ne sont pas remplacées.

```
const a = 0 || 4;  // 4  (dangereux si 0 est valide)
const b = 0 ?? 4;  // 0  (correct ici)
```

opérateur ??=

Assigne une valeur uniquement si la variable est `null` ou `undefined`.

```
const config = { seuilReussite: null };

config.seuilReussite ??= 4; // devient 4
config.mode ??= 'standard'; // devient 'standard'
```

opérateur de décomposition 'spread' ...

Copier, fusionner et étendre facilement des tableaux / objets.

```
const base = ['Français', 'Maths'];
const options = ['Physique', 'Chimie'];

const toutesBranches = [...base, ...options];

const eleve = { nom: "TERNET", prenom: "Alain" };
const details = { classe: "EMF", annee: "2024-2025" };

const eleveComplet = { ...eleve, ...details };
```

Déstructuration

Extraire rapidement certaines valeurs et récupérer le reste.

```
const [premiereEval, ...autres] = jsonData.evaluations;
const { etablissement, annee_scolaire } = jsonData;
```

Date et Heure

Lien : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Date

Obtenir la date et/ou heure actuelle

```
const maintenant = new Date();

console.log(maintenant.toLocaleDateString());
console.log(maintenant.toLocaleTimeString());

const jour = maintenant.getDate();
const mois = maintenant.getMonth() + 1; // janvier = 0
const annee = maintenant.getFullYear();

console.log(`${jour}.${mois}.${annee}`);
console.log(maintenant.toISOString());
```

Math

Lien : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Math

`Math.PI` - la constante π

Accès direct à π, utile pour les calculs géométriques.

```
console.log(Math.PI); // 3.141592653589793
```

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

Retire le signe, utile pour les écarts.

```
const ecart = Math.abs(4 - 5.7); // 1.7
```

`Math.pow()` - élever à une puissance

`Math.pow(base, exposant)` ou `base ** exposant`.

```
console.log(Math.pow(2, 3)); // 8
console.log(2 ** 3);         // 8
```

`Math.min()` - plus petite valeur

```
console.log(Math.min(3, 7, -2, 0)); // -2
```

`Math.max()` - plus grande valeur

```
console.log(Math.max(3, 7, -2, 0)); // 7
```

`Math.ceil()` - arrondir à l'entier supérieur

```
console.log(Math.ceil(4.2)); // 5
```

`Math.floor()` - arrondir à l'entier inférieur

```
console.log(Math.floor(4.9)); // 4
```

`Math.round()` - arrondir à l'entier le plus proche

```
console.log(Math.round(4.4)); // 4
console.log(Math.round(4.6)); // 5
```

`Math.trunc()` - partie entière

```
console.log(Math.trunc(4.9));  // 4
console.log(Math.trunc(-4.9)); // -4
```

`Math.sqrt()` - racine carrée

```
console.log(Math.sqrt(9)); // 3
```

`Math.random()` - nombre aléatoire `[0, 1)`

```
const tirage = Math.random(); // ex: 0.37
const noteRandom = Math.round(1 + Math.random() * 5); // 1 à 6
```

JSON

Lien : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/JSON

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

Pour stocker ou transmettre des données.

```
const extrait = {
  etablissement: jsonData.etablissement,
  nbEvaluations: jsonData.evaluations.length
};

console.log(JSON.stringify(extrait));
```

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

Pour relire une chaîne JSON.

```
const texteJson = '{"nom":"TARISTE","note":5.9}';
const evalObj = JSON.parse(texteJson);

console.log(evalObj.nom);
console.log(evalObj.note);
```

Chaînes de caractères

Lien : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/String

`split()` - découper en tableau

```
const date = "19.08.2024";
const [jour, mois, annee] = date.split('.');
```

`trim()`, `trimStart()` et `trimEnd()`

```
const nomSale = "  VOYANTE  ";
nomSale.trim();      // "VOYANTE"
nomSale.trimStart(); // "VOYANTE  "
nomSale.trimEnd();   // "  VOYANTE"
```

`padStart()` et `padEnd()`

```
const num = '5';
num.padStart(3, '0'); // '005'
num.padEnd(4, '-');   // '5---'
```

Console

Lien : https://developer.mozilla.org/fr/docs/Web/API/console

console.log()

```
console.log('Début de l’analyse des évaluations');
```

console.info(), warn(), error()

```
console.info('Chargement terminé.');
console.warn('Aucune note pour cette branche.');
console.error('Erreur critique.');
```

console.table()

```
console.table(jsonData.evaluations.slice(0, 5));
```

console.time(), timeLog(), timeEnd()

```
console.time('calcul');

const moyenne = jsonData.evaluations
  .filter(e => e.branche === 'Maths')
  .reduce((sum, e) => sum + e.note, 0) / jsonData.evaluations.filter(e => e.branche ===
'Maths').length;

console.timeLog('calcul');
console.timeEnd('calcul');
```

Tableaux

Lien : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array

```
const evaluations = jsonData.evaluations;
```

forEach

```
evaluations.slice(0, 3).forEach(e => {
  console.log(`${e.nom} ${e.prenom} : ${e.note}`);
});
```

entries()

```
for (const [index, evalObj] of evaluations.slice(0, 3).entries()) {
  console.log(index, evalObj.nom, evalObj.note);
}
```

in

Parcourt les index d'un tableau.

```
for (const i in evaluations.slice(0, 3)) {
  console.log(i); // 0, 1, 2
}
```

of

Parcourt directement les valeurs.

```
for (const evalObj of evaluations.slice(0, 3)) {
  console.log(evalObj.nom, evalObj.note);
}
```

find()

```
const premiereSous4 = evaluations.find(e => e.note < 4);
```

findIndex()

```
const indexVoyante = evaluations.findIndex(e => e.nom === 'VOYANTE');
```

indexOf() / lastIndexOf()

Sur un tableau de valeurs simples.

```
const branches = ['Maths', 'Physique', 'Maths', 'Français'];
branches.indexOf('Maths');     // 0
branches.lastIndexOf('Maths'); // 2
```

push(), pop(), shift(), unshift()

```
const notes = [4, 5];
notes.push(5.5);
notes.pop();
notes.unshift(3);
notes.shift();
```

slice()

```
const top5 = evaluations.slice(0, 5);
```

splice()

```
const copie = [...evaluations];
copie.splice(0, 1); // supprime le premier
```

concat()

```
const a = evaluations.slice(0, 2);
const b = evaluations.slice(2, 4);
const fusion = a.concat(b);
```

join()

```
const noms = ['VOYANTE', 'TERNET', 'TARISTE'];
noms.join(', '); // "VOYANTE, TERNET, TARISTE"
```

keys() et values() (objets)

```
const exemple = evaluations[0];
Object.keys(exemple);   // ['date','nom','prenom','branche','note']
Object.values(exemple); // [...]
```

includes()

```
const toutesBranches = evaluations.map(e => e.branche);
toutesBranches.includes('Maths');
```

every() et some()

```
evaluations.every(e => e.note >= 4); // tous réussis ?
evaluations.some(e => e.note < 3);   // au moins un gros échec ?
```

fill()

```
new Array(3).fill('en attente');
```

flat()

```
[ [1, 2], [3, 4] ].flat(); // [1, 2, 3, 4]
```

sort()

```
const notesTriees = [...evaluations]
  .map(e => e.note)
  .sort((a, b) => a - b);
```

map()

```
const nomsComplets = evaluations.map(e => `${e.nom} ${e.prenom}`);
```

filter()

```
const maths = evaluations.filter(e => e.branche === 'Maths');
const echec = evaluations.filter(e => e.note < 4);
```

groupBy()

```
const parBranche = Object.groupBy(evaluations, e => e.branche);
// parBranche['Maths'] → toutes les évaluations de Maths
```

flatMap()

```
const elevesUniques = [...new Set(
  evaluations.flatMap(e => `${e.nom} ${e.prenom}`)
)];
```

reduce() / reduceRight()

```
const sommeNotes = evaluations.reduce((sum, e) => sum + e.note, 0);
const moyenneGlobale = sommeNotes / evaluations.length;
```

reverse()

```
const ordreInverse = [...evaluations].reverse();
```

Techniques

`` (backticks) - templates

```
const e = evaluations[0];
`Le ${e.date}, ${e.prenom} ${e.nom} a obtenu ${e.note} en ${e.branche}.`;
```

`new Set()` - supprimer les doublons

```
const eleves = [...new Set(evaluations.map(e => `${e.nom} ${e.prenom}`))].sort();
```

Fonctions

Déclaration de fonction

```
function moyenneBranche(branche) {
  const notes = evaluations.filter(e => e.branche === branche);
  return notes.reduce((sum, e) => sum + e.note, 0) / notes.length;
}

const moyenneEleve = function (nom, prenom) {
  const notes = evaluations.filter(e => e.nom === nom && e.prenom === prenom);
  return notes.reduce((sum, e) => sum + e.note, 0) / notes.length;
};

const estReussi = (note) => note >= 4;

const labelNote = (note) => note >= 4 ? 'OK' : 'NOK';
```

Fonctions immédiatement invoquées (IIFE)

```
(() => {
  const nbEvaluations = evaluations.length;
  console.log(`Nombre total d'évaluations : ${nbEvaluations}`);
})();
```

Conclusion

Ce module m'a obligé à structurer ma manière de coder.
Travailler sur un gros jeu de données comme `jsonData` m'a montré l'intérêt concret de la programmation fonctionnelle :

- parcourir, filtrer et transformer des tableaux sans multiplier les boucles imbriquées ;
- écrire des fonctions courtes, réutilisables et prévisibles ;
- utiliser des méthodes comme `map`, `filter`, `reduce`, `groupBy`, combinées aux opérateurs modernes (`...`, `??`, backticks, `new Set`) pour obtenir rapidement les infos utiles.

Ces outils rendent le code plus clair, plus compact et plus simple à maintenir, surtout lorsqu'il s'agit de manipuler des données réelles comme des résultats scolaires.