😳 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

Dans ce module, je vais apprendre la programmation fonctionnelle et comprendre ses différences avec la programmation impérative.

Pendant 5 semaines (1,5 jour par semaine), je vais :

- Découvrir les paradigmes de programmation et l'intérêt de la programmation fonctionnelle
- Utiliser les fonctions de base comme `map`, `filter` et `reduce`
- Appliquer des concepts comme fonctions pures, immuabilité, closures, currying et récursion
- Mettre en pratique les bonnes pratiques et patterns

Opérateurs javascript super-cooool 😎

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

En Javascript, *Math.PI* est une constante contenant la valeur de π.

```
// Exemple : calculer la circonférence d'un cercle
let rayon = 5;
let circonference = 2 * Math.PI * rayon;
console.log("La circonférence du cercle est : " + circonference);
```

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

En JavaScript, *Math.abs()* est une fonction qui renvoie la valeur absolue d'un nombre — c'est-à-dire le nombre sans son signe.

```
console.log(Math.abs(-5)); // 5
console.log(Math.abs(5));  // 5
console.log(Math.abs(0));  // 0
```

`Math.pow()` - élever à une puissance

Math.pow(x, y) calcule x élevé à la puissance y, c'est-à-dire x^y

```
console.log(Math.pow(5, 3));
// Expected output: 125

console.log(Math.pow(4, 0.5));
// Expected output: 2

console.log(Math.pow(7, -2));
// Expected output: 0.02040816326530612
//                  (1/49)

console.log(Math.pow(-7, 0.5));
// Expected output: NaN
```

`Math.min()` - plus petite valeur

La fonction *Math.min()* renvoie le plus petit nombre d'une série de 0 ou plusieurs nombres ou bien NaN si au moins un des arguments fourni n'est pas un nombre ou ne peut pas être converti en nombre.

```
console.log(Math.min(2, 3, 1));
// Expected output: 1

console.log(Math.min(-2, -3, -1));
// Expected output: -3
```

`Math.max()` - plus grande valeur

La méthode statique Math.max() renvoie le plus grand nombre parmi ceux passés en paramètres, ou -Infinity si aucun paramètre n'est fourni.

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

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

La fonction Math.ceil() retourne le plus petit entier plus grand ou égal au nombre donné.

```
console.log(Math.ceil(0.95));
// Expected output: 1

console.log(Math.ceil(4));
// Expected output: 4

console.log(Math.ceil(7.004));
// Expected output: 8

console.log(Math.ceil(-7.004));
// Expected output: -7
```

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

Renvoie le plus grand entier inférieur ou égal au nombre donné.

```
console.log(Math.floor(4.7)); // 4
console.log(Math.floor(9.2)); // 9
console.log(Math.floor(-3.8)); // -4
```

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

Renvoie l'entier le plus proche du nombre fourni.

```
console.log(Math.round(4.3)); // 4
console.log(Math.round(4.7)); // 5
console.log(Math.round(-3.5)); // -3
```

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

Supprime la partie décimale d’un nombre, sans arrondir.

```
console.log(Math.trunc(4.9));  // 4
console.log(Math.trunc(-3.7)); // -3
```

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

Renvoie la racine carrée positive d’un nombre.

```
console.log(Math.sqrt(9));  // 3
console.log(Math.sqrt(16)); // 4
```

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

Renvoie un nombre flottant ≥ 0 et < 1.

```
console.log(Math.random()); // Exemple : 0.435672
console.log(Math.random() * 10); // Exemple : 7.12 (entre 0 et 10)
```

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 un objet JavaScript en chaîne JSON.

```
const personne = { nom: "Alice", age: 25 };
const json = JSON.stringify(personne);
console.log(json); // '{"nom":"Alice","age":25}'
```

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

Convertit une chaîne JSON en objet JavaScript utilisable.

```
const json = '{"nom":"Alice","age":25}';
const personne = JSON.parse(json);
console.log(personne.nom); // Alice
```

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

Divise une chaîne en tableau selon un séparateur.

```
const texte = "pomme,banane,cerise";
const fruits = texte.split(",");
console.log(fruits); // ["pomme", "banane", "cerise"]
```

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

- trim() → supprime espaces début et fin
- trimStart() → supprime espaces début
- trimEnd() → supprime espaces fin

```
const texte = "   hello world   ";
console.log(texte.trim());      // "hello world"
console.log(texte.trimStart()); // "hello world   "
console.log(texte.trimEnd());   // "   hello world"
```

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

Ajoute des caractères au début ou à la fin pour atteindre une longueur donnée.

```
const num = "5";
console.log(num.padStart(3, "0")); // "005"
console.log(num.padEnd(3, "-"));   // "5--"
```

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

- `console.info()` → message informatif
- `console.warn()` → avertissement
- `console.error()` → erreur

```
console.info("Info : opération réussie");
console.warn("Attention : valeur inattendue");
console.error("Erreur : impossible de charger le fichier");
```

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

Affiche un tableau ou un objet sous forme de tableau lisible.

```
const personnes = [
  { nom: "Alice", age: 25 },
  { nom: "Bob", age: 30 }
];
console.table(personnes);
```

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

Mesure le temps d'exécution d'un code.

```
console.time("test");
for(let i=0;i<100000;i++){}  // code à mesurer
console.timeLog("test");      // temps intermédiaire
console.timeEnd("test");      // temps total
-
```

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 pour chaque élément du tableau.

```
const fruits = ["pomme", "banane", "cerise"];
fruits.forEach(fruit => console.log(fruit));
```

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

Renvoie un itérateur donnant [index, valeur].

```
const fruits = ["pomme", "banane"];
for (const [i, fruit] of fruits.entries()) {
  console.log(i, fruit);
}
```

`in` - parcourir les clés d'un tableau

Parcourt les indices du tableau.

```
const fruits = ["pomme", "banane"];
for (const i in fruits) {
  console.log(i); // 0, 1
}
```

`of` - parcourir les valeurs d'un tableau

Parcourt directement les valeurs du tableau.

```
const fruits = ["pomme", "banane"];
for (const fruit of fruits) {
  console.log(fruit); // "pomme", "banane"
}
```

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

Renvoie le premier élément pour lequel la fonction renvoie true.

```
const nombres = [5, 12, 8, 130];
const premierGrand10 = nombres.find(n => n > 10);
console.log(premierGrand10); // 12
```

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

Renvoie l'index du premier élément qui vérifie la condition, ou -1 si aucun ne correspond.

```
const nombres = [5, 12, 8, 130];
const index = nombres.findIndex(n => n > 10);
console.log(index); // 1
```

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

Cherchent la position d'un élément dans un tableau.

- `indexOf()` → première occurrence
- `lastIndexOf()` → dernière occurrence

```
const nombres = [1, 2, 3, 2, 4];
console.log(nombres.indexOf(2));      // 1
console.log(nombres.lastIndexOf(2));  // 3
```

`push()`, `pop()`, `shift()` et `unshift()` - ajouter/supprimer au début/fin d'un tableau

Permettent d'ajouter ou retirer des éléments d'un tableau :

- `push()` → ajoute à la fin
- `pop()` → supprime à la fin
- `unshift()` → ajoute au début
- `shift()` → supprime au début

```
const fruits = ["pomme", "banane"];

// Fin
fruits.push("cerise");  // ["pomme", "banane", "cerise"]
fruits.pop();           // ["pomme", "banane"]

// Début
fruits.unshift("kiwi"); // ["kiwi", "pomme", "banane"]
```

```
fruits.shift();         // ["pomme", "banane"]
```

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

Crée une copie d'une portion du tableau, sans le modifier.

```
const fruits = ["pomme", "banane", "cerise", "kiwi"];
const partie = fruits.slice(1, 3);
console.log(partie); // ["banane", "cerise"]
```

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

Modifie le tableau d'origine : peut supprimer, insérer ou remplacer des éléments.

```
const fruits = ["pomme", "banane", "cerise"];

// Supprimer 1 élément à partir de l'index 1
fruits.splice(1, 1);
console.log(fruits); // ["pomme", "cerise"]

// Insérer sans supprimer
fruits.splice(1, 0, "kiwi");
console.log(fruits); // ["pomme", "kiwi", "cerise"]

// Remplacer un élément
fruits.splice(1, 1, "mangue");
console.log(fruits); // ["pomme", "mangue", "cerise"]
```

`concat()` - joindre deux tableaux

Crée un nouveau tableau en combinant plusieurs tableaux ou valeurs.

```
const fruits = ["pomme", "banane"];
const legumes = ["carotte", "tomate"];
const aliments = fruits.concat(legumes);
console.log(aliments); // ["pomme", "banane", "carotte", "tomate"]
```

`join()` - joindre des chaînes de caractères

Transforme un tableau en une seule chaîne, en séparant les éléments avec un délimiteur.

```
const fruits = ["pomme", "banane", "cerise"];
const texte = fruits.join(", ");
console.log(texte); // "pomme, banane, cerise"
```

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

Permettent d'obtenir les clés (`keys()`) ou les valeurs (`values()`) d'un objet sous forme de tableau.

```
const personne = { nom: "Alice", age: 25, ville: "Paris" };

console.log(Object.keys(personne));   // ["nom", "age", "ville"]
console.log(Object.values(personne)); // ["Alice", 25, "Paris"]
```

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

Renvoie `true` si le tableau contient la valeur recherchée, sinon `false`.

```
const fruits = ["pomme", "banane", "cerise"];
console.log(fruits.includes("banane")); // true
console.log(fruits.includes("kiwi"));   // false
```

`every()` et `some()` - vérifier si toutes/quelques valeurs respectent une condition

- `every()` → renvoie `true` si tous les éléments respectent la condition.
- `some()` → renvoie `true` si au moins un élément la respecte.

```
const nombres = [2, 4, 6, 8];

console.log(nombres.every(n => n % 2 === 0)); // true (tous pairs)
console.log(nombres.some(n => n > 5));        // true (au moins un > 5)
```

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

Remplit un tableau existant avec une valeur fixe, éventuellement sur une portion définie.

```
//tableau.fill(valeur, debut, fin)
const tab = [1, 2, 3, 4];
tab.fill(0);          // [0, 0, 0, 0]
```

```
tab.fill(5, 1, 3);    // [0, 5, 5, 0] → remplace indices 1 et 2
```

`flat()` - aplatir un tableau

Transforme un tableau imbriqué en un tableau à une seule dimension (ou plusieurs niveaux).

```
const tab = [1, [2, 3], [4, [5, 6]]];
console.log(tab.flat());      // [1, 2, 3, 4, [5, 6]]
console.log(tab.flat(2));     // [1, 2, 3, 4, 5, 6]
```

`sort()` - pour trier un tableau

Trie les éléments du tableau. Par défaut, les éléments sont triés en ordre lexicographique.
Pour un tri numérique, il faut fournir une fonction de comparaison.

```
const nombres = [10, 5, 20, 1];
nombres.sort();
console.log(nombres); // [1, 10, 20, 5] → ordre lexicographique

nombres.sort((a, b) => a - b);
console.log(nombres); // [1, 5, 10, 20] → ordre croissant numérique
```

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

Applique une fonction à chaque élément d'un tableau et renvoie un nouveau tableau avec les résultats.

```
const nombres = [1, 2, 3, 4];
const doubles = nombres.map(n => n * 2);
console.log(doubles); // [2, 4, 6, 8]
```

```
//Renvoie un tableau ne contenant que les marques de moto
  const listeMotos = dataMotos.map(moto =>
        `${moto.marque}`
    );
```

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

Crée un nouveau tableau ne contenant que les éléments qui satisfont une condition spécifique.

```
const nombres = [1, 2, 3, 4, 5];
const pairs = nombres.filter(n => n % 2 === 0);
console.log(pairs); // [2, 4]
```

```
//Filtrer les motos avec une marque commençant par "H"
    const resultat = dataMotos.filter(moto =>
        moto.marque.startsWith("H")
    );
```

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

Crée un objet où les clés correspondent au résultat de la fonction appliquée à chaque élément, et les valeurs sont des tableaux d'éléments correspondants.

```
const nombres = [6.1, 4.2, 6.3];
const groupes = nombres.groupBy(Math.floor);
console.log(groupes);
// { '4': [4.2], '6': [6.1, 6.3] }
```

```
//Grouper les empereurs par dynastie
const result = groupBy(empereurs, (empereur) => empereur.dynasty);
```

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

Applique une fonction à chaque élément du tableau comme `map()`, puis aplatit le résultat d'un niveau.

```
const phrases = ["Bonjour le monde", "Salut tout le monde"];
const mots = phrases.flatMap(phrase => phrase.split(" "));
console.log(mots);
// ["Bonjour", "le", "monde", "Salut", "tout", "le", "monde"]
```

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

Applique une fonction cumulatrice sur chaque élément du tableau pour produire une seule valeur finale.

- `reduce()` → de gauche à droite
- `reduceRight()` → de droite à gauche

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

```
// Somme de tous les éléments
const somme = nombres.reduce((acc, val) => acc + val, 0);
console.log(somme); // 10

// Concaténation de droite à gauche
const texte = ["a", "b", "c"].reduceRight((acc, val) => acc + val, "");
console.log(texte); // "cba"
```

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

Inverse directement l'ordre des éléments dans le tableau.

```
const nombres = [1, 2, 3, 4];
nombres.reverse();
console.log(nombres); // [4, 3, 2, 1]
```

Techniques

`` (backticks) - pour écrire des chaînes plus pratiques

Les backticks (`) permettent de créer des chaînes de caractères faciles à lire, où l'on peut :

- Mettre directement des variables ou des calculs dedans
- Écrire du texte sur plusieurs lignes

```
const nom = "Alice";
const age = 25;

// Insérer des variables directement dans le texte
const texte = `Bonjour, je m'appelle ${nom} et j'ai ${age} ans.`;
console.log(texte);
// Affiche : Bonjour, je m'appelle Alice et j'ai 25 ans.

// Écrire sur plusieurs lignes facilement
const multiLignes = `Ligne 1
Ligne 2
Ligne 3`;
console.log(multiLignes);
// Affiche :
// Ligne 1
// Ligne 2
// Ligne 3
```

`new Set()` - pour supprimer les doublons

Crée un ensemble contenant uniquement des valeurs uniques. Très pratique pour enlever les doublons d'un tableau.

```
const nombres = [1, 2, 2, 3, 4, 4, 5];
const unique = [...new Set(nombres)];
console.log(unique); // [1, 2, 3, 4, 5]
```

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

Durant ce module, j'ai adoré travailler et découvrir une nouvelle manière de coder en JavaScript, même s'il m'a fallu un certain temps pour comprendre certaines fonctions comme reduce. Je pense avoir appris un tas de choses super utiles pour la suite de ma formation en tant que développeur web.