🤔 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

> Votre introduction avec notamment les objectifs opérationnels du module.

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` représente la valeur de la constante mathématique π (environ 3.14159).

```
    // Calculer l'aire d'un cercle de rayon r
    const r = 5
    const aire = Math.PI * r ** 2
    console.log("air du cercle = 78,53981634")
```

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

La valeur absolue d'un nombre est sa distance par rapport à zéro sur la droite numérique, sans tenir compte de son signe.

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

`Math.pow()` - élever à une puissance

La méthode `Math.pow()` permet d'élever un nombre à une puissance donnée.

```
console.log(Math.pow(2, 3)); // 8
console.log(Math.pow(5, 2)); // 25
```

`Math.min()` - plus petite valeur

La méthode `Math.min()` permet de trouver la plus petite valeur parmi une liste de nombres.

```
console.log(Math.min(1, 2, 3)); // 1
console.log(Math.min(5, 10, 2)); // 2
```

`Math.max()` - plus grande valeur

La méthode `Math.max()` permet de trouver la plus grande valeur parmi une liste de nombres.

```
console.log(Math.max(1, 2, 3)); // 3
console.log(Math.max(5, 10, 2)); // 10
```

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

La méthode `Math.ceil()` permet d'arrondir un nombre à la valeur entière supérieure la plus proche.

```
console.log(Math.ceil(5.1)); // 6
console.log(Math.ceil(5.9)); // 6
```

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

La méthode `Math.floor()` permet d'arrondir un nombre à la valeur entière inférieure la plus proche.

```
console.log(Math.floor(5.1)); // 5
console.log(Math.floor(5.9)); // 5
```

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

La méthode `Math.round()` permet d'arrondir un nombre à la valeur entière la plus proche.

```
console.log(Math.round(5.1)); // 5
console.log(Math.round(5.9)); // 6
```

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

La méthode `Math.trunc()` permet de supprimer la partie décimale d'un nombre et de ne conserver que la partie entière.

```
console.log(Math.trunc(5.1)); // 5
console.log(Math.trunc(5.9)); // 5
```

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

La méthode `Math.sqrt()` permet de calculer la racine carrée d'un nombre.

```
console.log(Math.sqrt(4)); // 2
console.log(Math.sqrt(9)); // 3
```

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

La méthode `Math.random()` permet de générer un nombre aléatoire entre 0.0 (compris) et 1.0 (non compris).

```
console.log(Math.random()); // ex: 0.123456789
console.log(Math.random() * 10); // ex: 5.678901234
console.log(Math.floor(Math.random() * 10)); // ex: 5 (nombre entier entre
0 et 9)
console.log(Math.floor(Math.random() * 10) + 1); // ex: 6 (nombre entier
entre 1 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

stringify() permet de convertir un objet JavaScript en une chaîne JSON.

```

```

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

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

parse() permet de convertir une chaîne JSON en un objet JavaScript.

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

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

La méthode `split()` permet de diviser une chaîne de caractères en un tableau de sous-chaînes, en utilisant un séparateur spécifié.

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

split peut aussi prendre un second argument qui limite le nombre d'éléments dans le tableau résultant.

```
const phrase = "Bonjour tout le monde";
const mots = phrase.split(" ", 2);
console.log(mots); // ["Bonjour", "tout"]
```

ou sélectionner dans son tableau la chaine de caractères que l'on veut.

```
const phrase = "Bonjour tout le monde";
const mots = phrase.split(" ")[2];
```

```
console.log(mots); // "le"
```

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

La méthode `trim()` permet de supprimer les espaces en trop au début et à la fin d'une chaîne de caractères.

```
const phrase = "   Bonjour tout le monde   ";
console.log(phrase.trim()); // "Bonjour tout le monde"
```

La méthode `trimStart()` permet de supprimer les espaces en trop au début d'une chaîne de caractères.

```
const phrase = "   Bonjour tout le monde   ";
console.log(phrase.trimStart()); // "Bonjour tout le monde   "
```

La méthode `trimEnd()` permet de supprimer les espaces en trop à la fin d'une chaîne de caractères.

```
const phrase = "   Bonjour tout le monde   ";
console.log(phrase.trimEnd()); // "   Bonjour tout le monde"
```

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

La méthode `padStart()` permet de compléter une chaîne au début jusqu'à atteindre une longueur donnée, en ajoutant des caractères spécifiés.
La méthode `padEnd()` fait la même chose mais à la fin de la chaîne.

```
const code = "42";
console.log(code.padStart(5, "0")); // "00042"
console.log(code.padEnd(5, "_"));   // "42___"
```

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)

Permettent d'afficher des messages d'information, d'avertissement ou d'erreur dans la console, chacun avec un style visuel distinct.

```
console.info('Info : tout va bien');
console.warn('Attention : quelque chose cloche');
console.error('Erreur : une erreur est survenue');
```

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

Affiche un tableau ou un objet sous forme de tableau dans la console, pratique pour visualiser des données structurées.

```
const fruits = ["pomme", "banane", "orange"];
console.table(fruits);

const obj = { nom: "Alice", age: 25 };
console.table(obj);
```

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

Permettent de mesurer le temps d'exécution d'un bloc de code.

- `console.time(label)` démarre le chronomètre avec un nom.
- `console.timeLog(label)` affiche le temps écoulé depuis le début.
- `console.timeEnd(label)` arrête le chronomètre et affiche le temps total.

```
console.time("traitement");
for (let i = 0; i < 1000000; i++) { /* ... */ }
console.timeLog("traitement"); // Affiche le temps intermédiaire
console.timeEnd("traitement"); // Affiche le 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

La méthode `forEach()` permet d'exécuter une fonction pour chaque élément d'un tableau, sans retourner de nouveau tableau.

```
const fruits = ["pomme", "banane", "orange"];
fruits.forEach(fruit => {
  console.log(fruit);
});
// Affiche chaque fruit dans la console
```

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

La méthode `entries()` retourne un nouvel objet itérable qui contient des paires [index, valeur] pour chaque élément du tableau. Pratique pour parcourir à la fois l'index et la valeur.

```
const fruits = ["pomme", "banane", "orange"];
for (const [index, valeur] of fruits.entries()) {
  console.log(index, valeur);
}
// Affiche :
// 0 "pomme"
// 1 "banane"
// 2 "orange"
```

`values()` - parcourir les valeurs d'un tableau

La méthode `values()` retourne un objet itérable contenant les valeurs du tableau, utile pour parcourir uniquement les valeurs.

```
const fruits = ["pomme", "banane", "orange"];
for (const valeur of fruits.values()) {
  console.log(valeur);
}
// Affiche :
// "pomme"
// "banane"
// "orange"
```

`in` - parcourir les clés d'un tableau

La boucle `for...in` permet de parcourir les clés (indices) d'un tableau ou les propriétés d'un objet.

```
const fruits = ["pomme", "banane", "orange"];
for (const index in fruits) {
  console.log(index); // Affiche : 0, 1, 2
}
```

`of` - parcourir les valeurs d'un tableau

La boucle `for...of` permet de parcourir directement les valeurs d'un tableau, de façon simple et lisible.

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

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

La méthode `find()` permet de retourner le premier élément d'un tableau qui satisfait une condition donnée (fonction de test). Si aucun élément ne correspond, elle retourne `undefined`.

```
const nombres = [1, 4, 7, 10];
const premierGrand = nombres.find(n => n > 5);
console.log(premierGrand); // 7
```

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

La méthode `findIndex()` retourne l'index du premier élément d'un tableau qui satisfait une fonction de test. Retourne -1 si aucun élément ne correspond.

```
const nombres = [1, 4, 7, 10];
const index = nombres.findIndex(n => n > 5);
console.log(index); // 2
```

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

`indexOf()` retourne l'index de la première occurrence d'une valeur dans un tableau, ou -1 si elle n'est pas trouvée.
`lastIndexOf()` fait la même chose mais pour la dernière occurrence.

```
const fruits = ["pomme", "banane", "orange", "banane"];
console.log(fruits.indexOf("banane")); // 1
console.log(fruits.lastIndexOf("banane")); // 3
```

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

- `push()` ajoute un ou plusieurs éléments à la fin du tableau.
- `pop()` supprime et retourne le dernier élément du tableau.
- `shift()` supprime et retourne le premier élément du tableau.
- `unshift()` ajoute un ou plusieurs éléments au début du tableau.

```
const fruits = ["pomme", "banane"];
fruits.push("orange");      // ["pomme", "banane", "orange"]
fruits.pop();               // ["pomme", "banane"]
fruits.shift();             // ["banane"]
fruits.unshift("kiwi");     // ["kiwi", "banane"]
```

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

Permet de créer un nouveau tableau contenant une portion du tableau d'origine, sans le modifier. On précise l'index de début et éventuellement l'index de fin (non inclus).

```
const fruits = ["pomme", "banane", "orange", "kiwi"];
const sousTableau = fruits.slice(1, 3); // ["banane", "orange"]
```

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

Permet de modifier le contenu d'un tableau en supprimant, ajoutant ou remplaçant des éléments à partir d'un index donné.

```
const fruits = ["pomme", "banane", "orange"];
fruits.splice(1, 1); // Supprime "banane" à l'index 1
// ["pomme", "orange"]

fruits.splice(1, 0, "kiwi"); // Insère "kiwi" à l'index 1
// ["pomme", "kiwi", "orange"]

fruits.splice(0, 1, "ananas"); // Remplace "pomme" par "ananas"
console.log(fruits); // ["ananas", "kiwi", "orange"]
```

`concat()` - joindre deux tableaux

Permet de fusionner deux ou plusieurs tableaux en un nouveau tableau sans modifier les originaux.

```
const a = [1, 2];
const b = [3, 4];
```

```
const c = a.concat(b); // [1, 2, 3, 4]
```

`join()` - joindre des chaînes de caractères

La méthode `join()` permet de fusionner tous les éléments d'un tableau en une seule chaîne de caractères, en utilisant un séparateur choisi.

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

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

`Object.keys()` retourne un tableau contenant toutes les clés (propriétés) d'un objet.
`Object.values()` retourne un tableau contenant toutes les valeurs des propriétés de cet objet.

```
const obj = { nom: "Alice", age: 25 };
console.log(Object.keys(obj));   // ["nom", "age"]
console.log(Object.values(obj)); // ["Alice", 25]
```

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

La méthode `includes()` permet de savoir si un tableau contient une valeur spécifique. Retourne `true` si la valeur est trouvée, sinon `false`.

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

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

- `every()` teste si **tous** les éléments du tableau satisfont une condition. Retourne `true` si c'est le cas, sinon `false`.
- `some()` teste si **au moins un** élément du tableau satisfait une condition. Retourne `true` si c'est le cas, sinon `false`.

```
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

La méthode `fill()` permet de remplir tout ou partie d'un tableau avec une valeur donnée.

```
const tableau = [1, 2, 3, 4];
tableau.fill(0); // [0, 0, 0, 0]
tableau.fill(5, 1, 3); // [0, 5, 5, 0]
```

`flat()` - aplatir un tableau

La méthode `flat()` permet de "aplatir" un tableau contenant des sous-tableaux en un seul niveau.

```
const tableau = [1, [2, [3, 4]], 5];
console.log(tableau.flat()); // [1, 2, [3, 4], 5]
console.log(tableau.flat(2)); // [1, 2, 3, 4, 5]
## `sort()` - pour trier un tableau

La méthode `sort()` permet de trier les éléments d'un tableau selon
l'ordre alphabétique ou selon une fonction de comparaison personnalisée.

```javascript
const nombres = [3, 1, 4, 2];
nombres.sort(); // [1, 2, 3, 4] (pour les nombres, attention à la
comparaison)
const mots = ["banane", "pomme", "kiwi"];
mots.sort(); // ["banane", "kiwi", "pomme"]
```

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

La méthode `map()` crée un nouveau tableau contenant les résultats de l'application d'une fonction à chaque élément du tableau d'origine.

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

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

La méthode `filter()` crée un nouveau tableau contenant uniquement les éléments qui passent un test (fonction de filtrage).

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

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

La méthode `groupBy()` permet de regrouper les éléments d'un tableau selon une fonction de classement (clé).

```
const personnes = [
  { nom: "Alice", age: 25 },
  { nom: "Bob", age: 30 },
  { nom: "Charlie", age: 25 }
];
const groupes = personnes.groupBy(p => p.age);
// { 25: [{nom: "Alice", ...}, {nom: "Charlie", ...}], 30: [{nom: "Bob",
...}] }
```

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

La méthode `flatMap()` applique une fonction à chaque élément du tableau puis "aplatit" le résultat d'un niveau. Pratique pour transformer et fusionner des sous-tableaux en une seule étape.

```
const nombres = [1, 2, 3];
const resultat = nombres.flatMap(n => [n, n * 2]); // [1, 2, 2, 4, 3, 6]
```

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

La méthode `reduce()` parcourt le tableau et combine ses éléments pour produire une seule valeur (somme, produit, objet, etc.).
`reduceRight()` fait la même chose mais en partant de la fin du tableau.

```
const nombres = [1, 2, 3, 4];
const somme = nombres.reduce((acc, val) => acc + val, 0); // 10
const concatInverse = nombres.reduceRight((acc, val) => acc + val, ""); //
"4321"
```

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

La méthode `reverse()` inverse l'ordre des éléments d'un tableau directement (modifie le tableau original).

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

Techniques

`` (backticks) - pour des expressions intelligentes

Les backticks (`) permettent de créer des chaînes de caractères multi-lignes et d'insérer des variables ou expressions avec `${...}`.

```
const nom = "Alice";
console.log(`Bonjour ${nom} !`); // Bonjour Alice !
```

`new Set()` - pour supprimer les doublons

`Set` est une structure qui ne garde que les valeurs uniques. Pratique pour retirer les doublons d'un tableau.

```
const nombres = [1, 2, 2, 3, 3, 3];
const uniques = [...new Set(nombres)]; // [1, 2, 3]
```

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 module m'a permis de découvrir et de manipuler les principaux opérateurs, méthodes et techniques de la programmation fonctionnelle. on dispose désormais d'un pense-bête pratique pour écrire du code plus lisible, efficace et moderne.

J'ai bien aimé le contenu du module, même si c'était un peu compliqué par rapport aux autres cours. On verra bien pour l'évaluation...