😳 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
- Opérateurs javascript super-cooool 😎
  - opérateur `?:`
  - opérateur `??`
  - opérateur `??=`
  - opérateur de décomposition 'spread' `...`
  - Déstructuration
- Date et Heure
  - Obtenir la date et/ou heure actuelle
- Math
  - `Math.PI` - la constante π
  - `Math.abs()` - la |valeur absolue| d'un nombre
  - `Math.pow()` - élever à une puissance
  - `Math.min()` - plus petite valeur
  - `Math.max()` - plus grande valeur
  - `Math.ceil()` - arrondir à la prochaine valeur entière la plus proche
  - `Math.floor()` - arrondir à la précédente valeur entière la plus proche
  - `Math.round()` - arrondir à la valeur entière la plus proche
  - `Math.trunc()` - supprime la virgule et retourne la partie entière d'un nombre
  - `Math.sqrt()` - la raçine carrée d'un nombre
  - `Math.random()` - générer un nombre aléatoire entre 0.0 (compris) et 1.0 (non compris)
- JSON
  - `JSON.stringify()` - transformer un objet Javascript en JSON
  - `JSON.parse()` - transformer du JSON en objet Javascript
- Chaînes de caractères
  - `split()` - un ciseau qui coupe une chaîne là où un caractère apparaît et produit un tableau
  - `trim()`, `trimStart()` et `trimEnd()` - épuration des espaces en trop dans une chaîne (trimming)
  - `padStart()` et `padEnd()` - aligner le contenu dans une chaîne de caractères
- Console
  - `console.log()` - Afficher un message sur la console
  - `console.info()`, `warn()` et `error()` - Afficher un message sur la console (filtrables)
  - `console.table()` - Afficher tout un tableau ou un objet sur la console
  - `console.time()`, `timeLog()` et `timeEnd()` - Chronométrer une durée d'exécution
- Tableaux
  - `forEach` - parcourir les éléments d'un tableau

    - `entries()` - parcourir les couples index/valeurs d'un tableau
    - `in` - parcourir les clés d'un tableau
    - `of` - parcourir les valeurs d'un tableau
    - `find()` - premier élément qui satisfait une condition
    - `findIndex()` - premier index qui satisfait une condition
    - `indexOf()` et `lastIndexOf()` - premier/dernier élément qui correspond
    - `push()`, `pop()`, `shift()` et `unshift()` - ajouter/supprime au début/fin dans un tableau
    - `slice()` - ne conserver que certaines lignes d'un tableau
    - `splice()` - supprimer/insérer/remplacer des valeurs dans un tableau
    - `concat()` - joindre deux tableaux
    - `join()` - joindre des chaînes de caractères
    - `keys()` et `values()` - les clés/valeurs d'un objet
    - `includes()` - vérifier si une valeur est présente dans un tableau
    - `every()` et `some()` - vérifier si plusieurs valeurs sont toutes/quelques présentes dans un tableau
    - `fill()` - remplir un tableau avec des valeurs
    - `flat()` - aplatir un tableau
    - `sort()` - pour trier un tableau
    - `map()` - tableau avec les résultats d'une fonction
    - `filter()` - tableau avec les éléments passant un test
    - `groupBy()` - regroupe les éléments d'un tableau selon un règle
    - `flatMap()` - chaînage de map() et flat()
    - `reduce()` et `reduceRight()` - réduire un tableau à une seule valeur
    - `reverse()` - inverser l'ordre du tableau
- Techniques
    - ``(backticks) - pour des expressions intelligentes
    - `new Set()` - pour supprimer les doublons
- Fonctions
    - Déclaration de fonction
    - Fonctions immédiatement invoquées (IIFE)
- Conclusion

Introduction

> Dans ce module, nous allons apprendre comment programmer de manière fonctionnelle avec les différents opérateurs, les différentes chaînes de caractères, etc...
> Voici les objectifs du module :
>
> - Analyser et décrire les exigences en vue de la réalisation d'une programmation fonctionnelle.
> - Implémenter de manière efficiente des algorithmes et des problèmes d'applications selon le paradigne de programmmation fonctionnelle et les exigences données.
> - Améliorer et optimiser le code impératif implémenté en utilisant la programmation fonctionnelle (refactorisation).
> - Vérifier l'exactitude et la qualité de l'implémentation.

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

Math.PI est une constante qui représente la valeur de π (≈ 3.14159) et sert pour les calculs en lien avec les cercles ou les ang

```
const aireCercle = (rayon) => {
  return Math.round((Math.PI * rayon * rayon) * 100) / 100;
};
```

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

Math.abs() retourne la valeur absolue d'un nombre, c'est-à-dire le nombre sans son signe (transforme un négatif en positif).

```
console.log(Math.abs(-7));
```

`Math.pow()` - élever à une puissance

Math.pow() permet de calculer une puissance, en élevant un nombre à la puissance d'une autre. Voici l'exemple ci-dessous :

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

`Math.min()` - plus petite valeur

Math.min() a comme but de renvoyer le plus petit nombre d'une liste de valeurs donnnées. Voici l'exemple ci-dessous :

```
console.log(Math.min(4, 9, -2, 7));
```

`Math.max()` - plus grande valeur

Math.max() a comme but de renvoyer la plus grande valeur d'une liste de valeurs de données. Voici l'exemple ci-dessous :

```
console.log(Math.max(4, 9, -2, 7));
```

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

Math.ceil() permet d'arrondir un nombre à l'entier supérieur, même si la décimale est faible :

```
console.log(Math.ceil(4.1));
```

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

Math.floor() permet d'arrondir un nombre à l'entier inférieur, en supprimant la partie décimale :

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

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

Méthode utilisée pour arrondir des nombres, dans l'exemple ci-dessous nous avons arrondi au demi-point :

```

```

```
const note = (pointsMax) => (points) => {
  return  Math.round(((points / pointsMax) * 5 + 1)* 2 ) /2;
};
```

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

Math.trunc() permet de supprimer la partie décimale d'un nombre et renvoie uniquement sa partie entière, sans arrondir :

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

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

Math.sqrt() renvoie la racine carrée d'un nombre :

```
console.log(Math.sqrt(25));
```

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

Math.random est une méthode qui renvoie une valeur entre 0 et 0.99999999999 et ensuite vous la multipliez pour avoir quelque chose entre un min et un max Voici l'exemple ci-dessous :

```
public static void main(String[] args) {
  int nombre = (int) (Math.random() * (max - min + 1)) + min;
  System.out.println("b");
};
```

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

JSON.stringify() sert à convertir un objet JavaScript en chaîne de texte JSON. Voici l'exemple ci-dessous :

```
const user = { name: "Emma", age: 20};
const jsonText = JSON.stringify(user);
console.log(jsonText);
```

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

JSON.parse() sert à convertir une chaîne JSON en objet JavaScript, c'est l'inverse de JSON.stringify. Voici l'exemple ci-dessous :

```
const jsonText = '{"name: "Emma", "age": 20}';
const user = JSON.parse(jsonText);
console.log(user.name);
```

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

split() sert à couper une chaîne de caractères en un tableau, selon un séparateur choisi. Voici l'exemple ci-dessous :

```
const phrase = "salut, au revoir, bonjour";
const separer = phrase.split(",");
console.log(separer);
```

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

trim(), trimStart(), trimEnd() servent à supprimer les espaces inutiles dans une chaîne de caractères. Voici l'exemple ci-dessous :

```
const text = "   Salut ça va?   ";
console.log(text.trim()); // enlève les espaces au début et à la fin
console.log(text.trimStart()); // enlève seulement au début
console.log(text.trimEnd()) // enlève seulement à la fin
```

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

padStart() et padEnd() servent à compléter une chaîne avec des caractères (souvent des zéros ou espaces) jusqu'à une longueur donnée. Voici l'exemple ci-dessous :

```
const num = "7";
console.log(num.padStart(3, "0")); // ajoute des 0 au début (007)
console.log(num.padEnd(4, ".")) // ajoute des points à la fin (7...)
```

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)

console.info(), console.warn(), console.error() servent à afficher des messages dans la console avec des niveaux d'importance différentes. Voici l'exemple ci-dessous :

```
console.info("Info : chargement terminé.") // informatif, en bleu
console.warn("Attention : données incomplètes.") // avertissement, en jaune
console.error(" Erreur : échec de la requête !") // erreur, en rouge
```

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

console.table() affiche les données sous forme de tableau dans la console, ce qui rend la lecture plus claire. Voici l'exemple ci-dessous :

```
const users = [
  {name: "Emma", age: 20},
  {name: "Julie", age: 17}
];
console.table(users);
```

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

Les méthodes console.time(), console.timeLog() et consoletimeEnd() servent à mesurer le temps d'exécution d'un morceau de code. Voici l'exemple ci-dessous :

```
console.time("test"); // démarre le chronomètre

for(let i = 0; i < 1e6; i++) {} // code à mésurer

console.timeLog("test") // affiche le temps écoulé jusqu'ici
console.timeEnd("test") // affiche le temps total et stop le chrono
```

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

forEach() permet d'exécuter une fonction pour chaque élément d'un tableau, sans renvoyer de nouveau tableau. Voici l'exemple ci-dessous :

```
const eleves = ["emma", "julie", "cem"];

eleves.forEach(eleve => {
  console.log(eleve);
});
```

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

entries() renvoie un itérateur contenant les paires [index, valeur] d'un tableau ou les paires [clé, valeur] d'un objet Map. Voici l'exemple ci-dessous :

```
const eleves = ["emma", "julie", "cem"];

for(const [index, eleve] of eleves.entries()) {
  console.log(index, eleve);
}
```

`in` - parcourir les clés d'un tableau

in sert à vérifier si une propriété existe dasn un objet ou un index dans un tableau. Voici l'exemple ci-dessous :

```
const user = {name: "Emma", age: 20};

console.log("name" in user); // true
console.log("email" in user); //false
```

`of` - parcourir les valeurs d'un tableau

of est utilisé dans un boucle for...of pour parcourir directement les valeurs d'un tableau, d'une chaîne ou d'un autre objet itérable. Voici l'exemple ci-dessous :

```
const eleves = ["emma", "cem", "julie"];

for(const eleve of eleves) {
  console.log(eleve);
}
```

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

find() permet de retourner le premier élément d'un tableau qui satisfait une condition donnée (fonction de test).

```
const numbers = [5, 12, 8, 130, 44];
const found = numbers.find(num => num > 10);
console.log(found) // il va donc retourner le premier nombre supérieur à 10
```

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

findIndex() renvoie l'index du premier élément d'un tableau qui satisfait une condition donnée. Voici l'exemple ci-dessous :

```
const numbers = [5, 12, 8, 130, 44];
const index = numbers.findIndex(num => num > 10);
console.log(index);
```

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

indexOf() et lastIndexOf() servent à trouver la position (index) d'un élément dans un tableau ou une chaîne. Voici l'exemple ci-dessous :

```
const eleves = ["julie", "emma", "cem"];

console.log(eleves.indexOf("emma")); // résultat : 1
console.log(eleves.lastIndexOf("cem")); // résultat : 2
```

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

push(), pop(), shift() et unshift() servent à ajouter ou rétirer des éléments d'un tableau. Voici l'exemple ci-dessous :

```
const eleves = ["emma", "julie"];

eleves.push("cem") // le tableau devient ["emma", "julie", "cem"]
eleves.pop(); // rétire le dernier donc ça revient comme avant
eleves.shift(); // rétire le premier donc il reste que ["julie"]
eleves.unshift("emma") // ajoute au début donc ça devient ["emma", "julie"]
```

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

slice() sert à copier une partie d'un tableau ou d'une chaîne sans le modifier. Voici l'exemple ci-dessous :

```
const eleves = ["emma", "julie", "cem", "noé", "diogo"];
const partie = eleves.slice(1,3); // le résultat sera julie et noé
```

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

splice() sert à modifier un tableau directement : on peut supprimer, ajouter ou remplacer des éléments. Voici l'exemple ci-dessous :

```
const eleves = ["cem", "julie", "emma"];
eleves.splice(1,1); // ça supprime 1 élément à partir de l'index 1 donc le
résultat est ["cem", "emma"]
```

`concat()` - joindre deux tableaux

concat() sert à fusionner plusieurs tableaux ou chaines en un nouveau tableau, sans modifier les originaux. Voici l'exemple ci-dessous :

```
const vraisEleves = ["emma", "julie", "cem"];
const fauxEleves = ["marion", "jeremy"];
```

```
const eleves = vraisEleves.concat(fauxEleves); // ["emma", "julie", "cem",
"marion", "jeremy"]
```

`join()` - joindre des chaînes de caractères

join() sert à transformer un tableau en chaîne de caractères, en mettant un séparateur entre les éléments. Voici un exemple ci-dessous :

```
const eleves = ["emma", "cem", "julie"];
console.log(fruits.join(" - ")); //emma - cem - julie
```

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

keys() et values() servent a récupérer les clés (index) ou les valeurs d'un tableeau ou d'un objet Map. Voici l'exemple ci-dessous :

```
const eleves = ["cem", "emma", "julie"];

for (const index of eleves.keys()) {
  console.log(index); // le résultat sera 0, 1, 2

for (const valeur of eleves.values()) {
  console.log(valeur); // le résultat sera "cem", "emma", "julie"
  }
}
```

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

includes() sert à vérifier si un tableau ou une chaîne contient une valeur donnée, et renvoie true ou false.

```
A FAIRE PAR VOS SOINS...
const eleves = ["emma", "julie", "cem"];

console.log(eleves.includes("emma")); // true
console.log(eleves.includes("diogo")); // false
```

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

every() et some() servent à tester des conditions sur les éléments d'un tableau. every() renvoie true si tous les éléments respectent la condition. some() renvoie true si au moins un élément respecte la condition. Voici l'exemple ci-dessous :

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

console.log(nombres.every(n => n % 2 === 0)); // true car tous sont pairs
console.log(nombres.some(n => n > 6)); // true car il y a au moins un > 6
```

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

fill() sert à remplir un tableau avec une même valeur, sur tout ou une partie de celui-ci. La syntaxe est "array.fill(valeur, début, fin) donc il remplit de début (inclus) à fin (exclu). Voici l'exemple ci-dessous :

```
const nombres = [1, 2, 3, 4, 5];
nombres.fill(0); // le résultat est [0, 0, 0, 0, 0]

const autres = [1, 2, 3, 4, 5];
autres.fill(9,1,4);  // le résultat est [1, 9, 9, 9, 5]
```

`flat()` - aplatir un tableau

flat() sert à aplatir un tableau de tableaux en retirant les niveaux imbriqués. Voici l'exemple ci-dessous :

```
const arr = [1, [2, 3], [4, [5, 6]]];
console.log(arr.flat()); // [1, 2, 3, 4, [5, 6]] il est aplati d'un niveau
console.log(arr.flat(2)); // [1, 2, 3, 4, 5, 6] il est aplati de deux niveaux
```

`sort()` - pour trier un tableau

sort() sert à trier les éléments d'un tableau, par ordre alphabétique par défaut, ou selon une fonction de comparaison. Voici l'exemple ci-dessous :

```
const eleves = ["emma", "julie", "cem"];
eleves.sort(); // ["cem", "emma", "julie"]

// pour un tri numérique
const nombres = [10, 2, 30],
nombres.sort((a, b) => a - b); // [2, 10, 30]
```

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

map() sert à créer un nouveau tableau en transformant chaque élément selon une fonction donnée. Voici l'exemple ci-dessous :

```
const resultat = dataVilles.map(ville => ({ Ville: ville.ville, Canton:
ville.canton }));
/* le résultat est :
 {
      "Ville": "Fribourg",
      "Canton": "FR"
   }, */
```

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

filter() sert à créer un nouveau tableau contenant uniquement les éléments qui respectent une condition, il ne modifie pas le tableau original. Il renvoie un nouveau tableau avec les éléments correspondants. Voici l'exemple ci-dessous :

```
const resultat = dataEvaluations.filter(evalu => evalu.branche === 'Maths' &&
evalu.note == '6.0');
/* le résultat est :
  {
      "date": "02.12.2024",
      "nom": "NISSENS",
      "prenom": "Remy",
      "branche": "Maths",
      "note": 6
   }, */
```

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

groupBy() sert à regrouper les éléments d'un tableau selon une clé ou une condition, et renvoie un objet dont chaque propriété représente un groupe. Voici l'exemple ci-dessous :

```
const users = [
  {name: "Emma", age: 10},
  {name: "Julie", age: 17},
  {name: "Cem", age: 25},
  {name: "Diogo", age: 17}
];

const  group = Object.groupBy(users, user => user.age);
/* le réultat sera :
  18: [{ name: "Cem", age: 25 }],
  22: [{ name: "Julie", age: 17 }, { name: "Diogo", age: 17 }],
  16: [{ name: "Emma", age: 10 }] */
```

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

flatMap() combine map() et flat(). Il transforme chaque élément d'un tableau, puis aplatit le résultat. Voici l'exemple ci-dessous :

```
const commandes = [
  { id: 1, produits: ["tasse", "assiette"] },
  { id: 2, produits: ["vase"] },
  { id: 3, produits: ["bol", "plat"] }
];

// il extrait  tous les produits dans un seul tableau
const produits = commandes.flatMap(c => c.produits); // le résultat est  ["tasse",
"assiette", "vase", "bol", "plat"]
```

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

reduce() et reduceRight() servent à accumuler (réduire) les valeurs d'un tableau en une seule valeur finale, en appliquant une fonction à chaque élément. La différence est que reduceRight() parcourt le tableau de droite à gauche. Le reduce() est utilisé pour les sommes, totaux, regroupements. En revanche le reduceRight() est utilisé pour les concaténations inversées ou les calculs rétroactifs. Voici l'exemple ci-dessous :

```
// reduce()
const panier = [
  { produit: "tasse", prix: 12},
  { produit: "vase", prix: 30},
  { produit: "assiette", prix: 18}
];

const total = panier.reduce((acc, produitPanier) => acc + produitPanier.prix, 0);
// le résultat est 60

// reduceRight()
const lettres = ["a", "b", "c"];
const mot = lettres.reduceRight((acc, 1) => acc + 1); // le résultat est "cba"
```

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

reverse() sert à inverser l'ordre des éléments d'un tableau directement (en le modifiant), il change le tableau original. Voici l'exemple ci-dessous :

```
const nombres = [1, 2, 3, 4];
nombres.reverse(); // le résultat est [4, 3, 2, 1]
```

Techniques

``(backticks) - pour des expressions intelligentes

Les backticks (accent grave ``) servent à créer des chaînes de caractères dynamiques en JavaScript. Voici l'exemple ci-dessous :

```
const name = "Emma";
const age = 18;

console.log(`Bonjour, je m'appelle ${name} et j'ai ${age} ans.`);
// Bonjour, je m'appelle Emma et j'ai 20 ans.

const texte = `Ligne 1
Ligne 2
Ligne 3`;
console.log(texte);
console.log(`Dans 5 ans, j'aurai ${age + 5} ans.`);
// Dans 5 ans, j'aurai 23 ans.
```

`new Set()` - pour supprimer les doublons

new Set() crée un ensemble (Set), càd une collection de valeurs uniques donc aucun doublon n'est autorisé. Voici l'exemple ci-dessous :

```
const nombres = [1, 2, 2, 3, 4, 4];
const uniques = [...new Set(nombres)]; // le résulat est [1, 2, 3, 4]
```

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

Je trouve que durant ce module, nous avons appris beaucoup de chose même si parfois c'était pas très facie. Maintenant je sais comment développer de manière fonctionnelle.

Mes points forts

Je trouve que j'ai assez bien compris l'utilisation des filter(), map(), les pipes ou les builder pattern. J'ai bien réussi à les implémenter durant les exercices.

Mes points faibles

Mon principal point faible durant ce module était la méthode reduce(). J'ai vraiment eu de la peine à la comprendre, surtout qu'elle devenait de plus en plus complexe. Cependant, au fil du temps, j'ai commencé à mieux la comprendre, même si j'ai encore un peu de difficulté.