JavaScript offre plusieurs manières de définir des fonctions :

1. **Les fonctions régulières (function declaration & function expression)**
2. **Les fonctions fléchées (Arrow Functions)**

1. Les Fonctions Régulières en JavaScript

Les fonctions régulières peuvent être définies de deux manières principales :

- **Déclaration de fonction (Function Declaration)**
- **Expression de fonction (Function Expression)**

a. Déclaration de Fonction (Function Declaration)

Une déclaration de fonction est définie avec le mot-clé function. Elle peut être appelée avant sa définition grâce au **hoisting** (remontée de la fonction).

Exemple :

```
function saluer(nom) {
  return `Bonjour, ${nom} !`;
}

console.log(saluer("Alice")); // Bonjour, Alice !
```

b. Expression de Fonction (Function Expression)

Une expression de fonction affecte une fonction à une variable. Contrairement aux déclarations de fonction, **elles ne sont pas "hoistées"**, ce qui signifie qu'elles doivent être définies avant d'être appelées.

Exemple :

```
const direBonjour = function(nom) {
  return `Bonjour, ${nom} !`;
};

console.log(direBonjour("Bob")); // Bonjour, Bob !
```

c. Les Fonctions Fléchées (Arrow Functions)

Les fonctions fléchées sont une syntaxe plus concise introduite avec ES6. Elles sont souvent utilisées dans les **callbacks** et les **fonctions anonymes**.

i. Syntaxe de base

```
const addition = (a, b) => a + b;

console.log(addition(4, 5)); // 9
```

ii. Cas avec un seul paramètre

Si une fonction prend **un seul paramètre**, les parenthèses peuvent être omises.

Exemple :

```
const double = x => x * 2;

console.log(double(5)); // 10
```

iii. Cas avec plusieurs instructions

Si la fonction contient **plusieurs instructions**, il faut utiliser {} et return.

Exemple :

```
const soustraction = (a, b) => {
  console.log(`Soustraction de ${a} - ${b}`);
  return a - b;
};

console.log(soustraction(10, 3)); // 7
```

iv. Différence entre une fonction régulière et une fonction fléchée

Les fonctions fléchées **ne possèdent pas leur propre this**. Elles héritent de this du contexte parent.

Exemple avec une fonction régulière :

```
  const objet = {
   nom: "Alice",
   salut: function() {
      console.log(`Bonjour, je suis ${this.nom}`);
   }
};

objet.salut(); // Bonjour, je suis Alice
```

Ici, this fait référence à objet.

Exemple avec une fonction fléchée :

```
 const objet = {
   nom: "Bob",
   salut: () => {
      console.log(`Bonjour, je suis ${this.nom}`);
   }
};

objet.salut(); // Bonjour, je suis undefined
```

La fonction fléchée ne lie pas this à l'objet objet, donc this.nom est undefined.

Exercices corriges

Exercice 1 :

Créer un objet calculatrice avec les méthodes addition, soustraction, et multiplication.

Correction :

```
 const calculatrice = {
   addition: (a, b) => a + b,
   soustraction: (a, b) => a - b,
   multiplication: (a, b) => a * b
};

console.log(calculatrice.addition(4, 5)); // 9
console.log(calculatrice.soustraction(10, 3)); // 7
console.log(calculatrice.multiplication(3, 3)); // 9
```

Exercice 2 :

a. Créer une fonction sommeTableau qui prend un tableau de nombres et retourne la somme de ses éléments **sans utiliser reduce()**.

b. Refaire l'exercice en utilisant reduce

Exercice 3 :

a. Créer une fonction trouverMax qui prend un tableau de nombres et retourne le plus grand nombre **sans utiliser Math.max()**.

b. Refaire l'exercice en utilisant Math.max()

Exercice 4 :

a. Créer une fonction **trouverIndex** qui prend un tableau et une valeur, et retourne l'index de cette valeur dans le tableau **sans utiliser indexOf()**.

b. Refaire l'exercice en utilisant indexOf()

Exercice 5 :

Créer une fonction compterOccurrences qui prend un tableau et une valeur, et retourne **le nombre de fois** que cette valeur apparaît dans le tableau.

Exercice 6 :

Créer une fonction estTrie qui prend un tableau et retourne true si les éléments sont triés en ordre croissant, sinon retourne false.

Exercice 7 :

Créer une fonction inverserTableau qui prend un tableau et retourne un **nouveau tableau inversé**, **sans utiliser reverse()**.

Exercice 8 :

a. Créer une fonction fusionnerTableaux qui prend deux tableaux et retourne un **nouveau tableau contenant tous les éléments des deux tableaux**, sans utiliser .concat().

b. Refaire l'exercice en utilisant .contact()

Exercice 9 :

a. Créer une fonction supprimerDoublons qui prend un tableau et retourne un **nouveau tableau sans doublons**, **sans utiliser Set**.

b. Refaire l'exercice en utilisant Set

Les fonctions de manipulation des tableaux en Javascript

Les **tableaux (arrays)** en JavaScript sont des structures de données permettant de stocker plusieurs valeurs sous une seule variable. JavaScript propose plusieurs **méthodes intégrées** pour manipuler les tableaux, notamment pour **ajouter, supprimer, filtrer, trier et transformer** les éléments.

a. Création d'un tableau en JavaScript

Un tableau peut être créé de plusieurs manières :

```
 // Création d'un tableau vide
let tableau1 = [];

// Tableau avec des valeurs
let fruits = ["Pomme", "Banane", "Orange"];

// Utilisation du constructeur Array
let nombres = new Array(10, 20, 30);

console.log(fruits[0]); // "Pomme"
console.log(nombres[2]); // 30
```

b. Ajouter et supprimer des éléments

i. Ajout d'éléments

Méthode	Description
push()	Ajoute un élément à la fin du tableau
unshift()	Ajoute un élément au début du tableau

Exemple :

```
 let fruits = ["Pomme", "Banane"];
fruits.push("Orange");  // ["Pomme", "Banane", "Orange"]
fruits.unshift("Mangue"); // ["Mangue", "Pomme", "Banane", "Orange"]
console.log(fruits);
```

ii. Supprimer des éléments

Méthode	Description
pop()	Supprime et retourne le dernier élément
shift()	Supprime et retourne le premier élément
splice()	Supprime un ou plusieurs éléments à une position donnée

Exemple :

```
let fruits = ["Pomme", "Banane", "Orange"];

fruits.pop();  // ["Pomme", "Banane"]

fruits.shift(); // ["Banane"]

fruits.splice(0, 1); // Supprime "Banane", tableau vide []

console.log(fruits);
```

c. Parcourir un tableau

i. Boucles classiques

```
  let nombres = [10, 20, 30];
// Avec for
for (let i = 0; i < nombres.length; i++) {
   console.log(nombres[i]);
}
// Avec for...of (ES6)
for (let nombre of nombres) {
   console.log(nombre);
}
```

ii. forEach()

```
 let fruits = ["Pomme", "Banane", "Orange"];
fruits.forEach((fruit, index) => {
   console.log(`${index} : ${fruit}`);
});
```

d. Transformation des tableaux

Méthode	**Description**
map()	Transforme chaque élément et retourne un **nouveau tableau**
filter()	Retourne un **nouveau tableau** avec les éléments qui respectent une condition
reduce()	Calcule une valeur unique à partir du tableau

i. map() : Transformation

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

ii. filter() : Filtrage

```
 let ages = [18, 25, 30, 15, 10];
let adultes = ages.filter(age => age >= 18);

console.log(adultes); // [18, 25, 30]
```

iii. reduce() : Accumulateur

```
let nombres = [10, 20, 30];
let somme = nombres.reduce((acc, val) => acc + val, 0);

console.log(somme); // 60
```

e. Rechercher des éléments

Méthode	**Description**
indexOf()	Retourne l'index du premier élément trouvé, sinon -1
includes()	Retourne true si l'élément existe, sinon false
find()	Retourne le **premier élément** qui respecte une condition
findIndex()	Retourne l'index du premier élément qui respecte une condition

```
let nombres = [5, 10, 15, 20];
console.log(nombres.indexOf(10)); // 1
console.log(nombres.includes(15)); // true
console.log(nombres.find(num => num > 10)); // 15
console.log(nombres.findIndex(num => num > 10)); // 2
```

f. Trier et inverser un tableau

Méthode	**Description**
sort()	Trie un tableau en **ordre alphabétique** (modifie l'original)
reverse()	Inverse l'ordre des éléments
localeCompare()	Compare des chaînes pour un tri avancé

i. Exemple de tri

```
 let nombres = [30, 2, 15, 8];
// Attention : `sort()` trie par défaut comme des chaînes !
nombres.sort((a, b) => a - b);
console.log(nombres); // [2, 8, 15, 30]
```

ii. Exemple d'inversion

```
let lettres = ["a", "b", "c"];
lettres.reverse();
console.log(lettres); // ["c", "b", "a"]
```

g. Concaténer et joindre un tableau

Méthode	**Description**
concat()	Fusionne plusieurs tableaux
join()	Convertit un tableau en **chaîne de caractères**

Exemples

```
 let tab1 = [1, 2, 3];
let tab2 = [4, 5, 6];
let fusion = tab1.concat(tab2);
console.log(fusion); // [1, 2, 3, 4, 5, 6]
let mots = ["Bonjour", "tout", "le", "monde"];
console.log(mots.join(" ")); // "Bonjour tout le monde"
```

Exercices pratiques :

Exercice 1 : Doubler chaque élément d'un tableau

```
 let nombres = [2, 5, 10];

// Résultat attendu : [4, 10, 20]
```

Exercice 2 : Trouver les nombres pairs

```
let nombres = [1, 2, 3, 4, 5, 6];

// Résultat attendu : [2, 4, 6]
```

Exercice 3 : Calculer la somme des éléments

```
let nombres = [5, 10, 15];

// Résultat attendu : 30
```

Exercice 4 : Vérifier si "Banane" est dans la liste

```
let fruits = ["Pomme", "Orange", "Raisin"];

// Résultat attendu : false
```

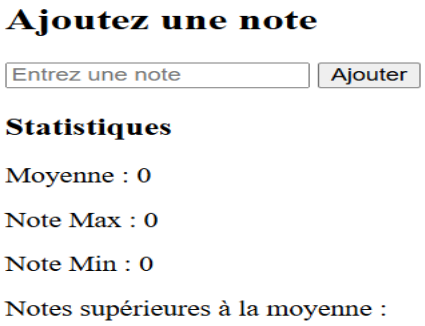
Exercice 5 : Trier et inverser un tableau

```
let nombres = [20, 5, 10, 2];

// Résultat attendu après tri croissant : [2, 5, 10, 20]

// Résultat après inversion : [20, 10, 5, 2]
```

Exercice 6 : Simulateur de gestion de notes scolaires

Créer un **formulaire** où l'utilisateur entre des notes, et le simulateur :

1. **Ajoute la note** à un tableau.
2. **Calcule la moyenne générale**.
3. **Affiche la note maximale et minimale**.
4. **Affiche les notes supérieures à la moyenne**.



Exercice 7 : Simulateur de gestion de stock

Créer une interface où l'on peut **ajouter, supprimer et rechercher des produits** avec les fonctionnalités suivantes :

1. **Ajouter un produit** avec nom et prix.
2. **Afficher la liste des produits**.
3. **Supprimer un produit** par son nom.
4. **Rechercher les produits à partir d'un prix minimum**.

Exercice 3 : Simulateur de calcul d'intérêts bancaires

Créer un **simulateur de placement** où l'utilisateur :

1. **Entre un capital initial**.
2. **Spécifie un taux d'intérêt annuel**.
3. **Définit la durée en années**.
4. **Affiche le capital final** avec un détail par année.

Simulateur de Placement

12000 | 20 | 5 | Calculer

Résultats

Capital Final : 29859.84€

- Année 1 : 14400.00€
- Année 2 : 17280.00€
- Année 3 : 20736.00€
- Année 4 : 24883.20€
- Année 5 : 29859.84€