# TP N°5

## Développement d'applications web

## JavaScript : Fonctions et Tableaux

Préparée par : RGUIBI Farah

# 1 Vue d'ensemble

Ce document présente l'analyse de **5 exercices pratiques** démontrant la maîtrise des concepts fondamentaux de **JavaScript**, incluant les **fonctions classiques**, les **fonctions fléchées**, la **manipulation de tableaux** et les **méthodes de parcours modernes**.

# 2 Exercice 1 : Fonction Factorielle

## 2.1 Objectif

Créer une fonction calculant la factorielle d'un nombre entier positif avec gestion d'erreurs.

## 2.2 Technologies utilisées

- **Fonctions classiques JavaScript**
- **Structures conditionnelles** (`if`)
- **Boucles** (`for`)
- **Gestion d'erreurs** (validation d'entrée)

## 2.3 Code implémenté - Version Console

```
function factorielle(x) {
    if (x < 0) {
        console.log("Erreur : la factorielle n'existe pas pour les nombres negatifs
    ");
        return;
    }
    let s = 1;
    for (let i = 1; i <= x; i++) {
        s = s * i;
    }
    return s;
}
console.log(factorielle(5)); // Resultat : 120
```

## 2.4 Code implémenté - Version Alert

```
function factorielle(x) {
    if (x < 0) {
        window.alert("Erreur : la factorielle n'existe pas pour les nombres
    negatifs");
        return;
    }
    let s = 1;
    for (let i = 1; i <= x; i++) {
        s = s * i;
    }
    return s;
}
window.alert(factorielle(5)); // Affiche : 120
```

## 2.5 Résultat

- **Entrée :** 5
- **Sortie :** 120 (car $5! = 5 \times 4 \times 3 \times 2 \times 1$)
- **Gestion d'erreur :** Message approprié pour les nombres négatifs

# 3 Exercice 2 : Fonctions Fléchées (Arrow Functions)

## 3.1 Objectif

Démontrer l'utilisation de la syntaxe moderne des fonctions fléchées en JavaScript (ES6+).

## 3.2 Technologies utilisées

- **Arrow Functions** (ES6+)
- **Paramètres de fonction**
- **Prompt** et interaction utilisateur

## 3.3 Implémentations

### 3.3.1 Fonction à un paramètre

```
const fonction1 = x => x + 5;
console.log(fonction1(3)); // Resultat : 8
```

### 3.3.2 Fonction à deux paramètres

```
const fonction2 = (x, y) => x + y;
console.log(fonction2(3, 6)); // Resultat : 9
```

### 3.3.3 Fonction interactive avec prompt

```
const fonction = () => {
    let num = Number(prompt("Entrez un nombre :"));
    return num + 2;
}
console.log(fonction()); // Resultat : nombre saisi + 2
```

## 3.4 Avantages des fonctions fléchées

- Syntaxe concise et moderne
- Pas de binding du `this`
- Idéales pour les callbacks et fonctions simples
- Return implicite pour les expressions simples

# 4 Exercice 3 : Manipulation de Tableaux

## 4.1 Objectif

Implémenter des fonctions utilitaires pour manipuler des tableaux de nombres.

## 4.2 Tableau de test

```
const numbers = [3, 8, 1, 12, 7, 5];
```

## 4.3 Fonctions implémentées

### 4.3.1 Trouver le maximum

```
function findMax(numbers) {
    let max = numbers[0];
    for (let n of numbers) {
        if (n > max) {
            max = n;
        }
    }
    return max;
}
console.log(findMax(numbers)); // Resultat : 12
```

### 4.3.2 Calculer la somme

```
function sumArray(numbers) {
    let s = 0;
    for (let n of numbers) {
        s += n;
    }
    return s;
}
console.log(sumArray(numbers)); // Resultat : 36
```

### 4.3.3 Compter les occurrences

```
function countOccurrences(arr, value) {
    let s = 0;
    for (let elem of arr) {
        if (elem == value) {
            s++;
        }
    }
    return s;
}
let occu = countOccurrences(numbers, 2);
console.log("les occurences sont", occu); // Resultat : 0
```

### 4.3.4 Supprimer les doublons

```
function removeDuplicates(arr) {
    return [...new Set(arr)];
}
let n = removeDuplicates(numbers);
console.log(removeDuplicates(n)); // Tableau sans doublons
```

**Technique utilisée :** Utilisation de l'objet `Set` qui ne conserve que les valeurs uniques, puis conversion en tableau avec l'opérateur spread (`...`).

### 4.3.5 Trouver l'index d'une valeur

```
function findIndex(arr, value) {
    for (let i = 0; i <= arr.length; i++) {
        if (arr[i] == value) {
            return i;
        }
    }
    return -1;
}
console.log(findIndex(numbers, 1));  // Resultat : 2
console.log(findIndex(numbers, 8));  // Resultat : 1
```

## 4.4 Résultats des tests

- **Maximum du tableau :** 12
- **Somme totale :** 36
- **Index de la valeur 1 :** 2
- **Index de la valeur 8 :** 1
- **Occurrences de 2 :** 0

# 5 Exercice 4 : Méthode Map()

## 5.1 Objectif

Utiliser la méthode `map()` pour transformer un tableau de nombres.

## 5.2 Code implémenté

```
const numbers = [1, 2, 3, 4, 5];
const fonction = () => numbers.map(n => n * 2);
console.log(fonction());
```

## 5.3 Résultat

```
[2, 4, 6, 8, 10]
```

## 5.4 Principe

La méthode `map()` crée un **nouveau tableau** en appliquant une fonction à chaque élément du tableau d'origine. Dans cet exemple, chaque nombre est multiplié par 2.

## 5.5 Avantages de map()

- Approche fonctionnelle et déclarative
- Ne modifie pas le tableau original (immutabilité)
- Code plus concis et lisible
- Facilite le chaînage de méthodes

# 6 Exercice 5 : Manipulation d'Objets avec Map()

## 6.1 Objectif

Transformer un tableau d'objets pour concaténer des propriétés.

## 6.2 Données de test

```
const personnes = [
    { prenom: 'Ahmed', nom: 'ALAMI' },
    { prenom: 'Fatima', nom: 'ALAOUI' },
    { prenom: 'Karim', nom: 'BAGHDADI' }
];
```

## 6.3 Code implémenté

```
const nomsComplets = (tab) => tab.map(p => `${p.prenom} ${p.nom}`);
const resultat = nomsComplets(personnes);
console.log(resultat);
```

## 6.4 Résultat

```
[
    "Ahmed ALAMI",
    "Fatima ALAOUI",
    "Karim BAGHDADI"
]
```