**TP : Découverte de JavaScript Moderne (ES6+)**

Achraf Içame G2.

**Objectif général**

Apprendre les bases pratiques du langage JavaScript moderne à travers des exercices guidés :  
*variables, fonctions, objets, tableaux, callbacks, promesses, async/await, etc.*

**Installation et préparation de l’environnement**

**1. Installer Node.js**

* Va sur <https://nodejs.org>
* Télécharge la version **LTS** et installe-la.
* Vérifie :
  + node -v
  + npm -v

**2. Créer ton projet**

mkdir tp-js-base

cd tp-js-base

npm init -y

**3. Créer un fichier principal**

Crée un fichier index.js :

touch index.js

**Variables et portée**

**Exercice 1**

1. Déclare trois variables :

var x = 5

let y = 10

const z = 15

1. Essaie de réassigner chacune d’elles.
2. Observe les erreurs et explique pourquoi.

REPONSE :

Observation :

Type d’erreur : Uncaught TypeError TypeError: Assignment to constant variable.

Donc les constantes peuvent pas être réassigner.

**Exercice 2**

Teste la portée :

function testScope() {

if (true) {

var a = "var visible partout";

let b = "let visible ici seulement";

}

console.log(a);

console.log(b); // Pourquoi erreur ?

}

testScope();

REPONSE :

Observation :

Type d’erreur : Uncaught ReferenceError ReferenceError: b is not defined.

La variable b est visible seulement au niveau du bloc :

if (true) {

    var a = "var visible partout";

    let b = "let visible ici seulement";}

**Fonctions classiques et fléchées**

**Exercice 3**

Crée deux fonctions qui saluent un utilisateur :

function sayHello(name) {

return `Bonjour ${name}`;

}

const sayHelloArrow = (name) => `Bonjour ${name}`;

Compare leur comportement et explique la différence.

REPONSE :

Observation :

Les deux fonctions ont le même objectif , mais la fonction fléché n’a pas besoin du mot clef return.

**Exercice 4**

Teste la portée de this :

const person = {

name: "Sara",

sayHello: function () {

console.log("Bonjour " + this.name);

},

sayHelloArrow: () => {

console.log("Bonjour " + this.name);

},

};

person.sayHello();

person.sayHelloArrow();

Observation :

{name: 'Sara', sayHello: ƒ, sayHelloArrow: ƒ}

Bonjour Sara

{}

Bonjour undefined

Dans la fonction fléchée le mot clef this fait référence à l’objet globale , ce qui donne la sortie undefined dans la console.

**Import / Export de modules (ES6)**

Apprendre à **séparer le code en plusieurs fichiers** pour mieux l’organiser et le réutiliser.

**Étape 1 : Créer un module**

Crée un fichier mathUtils.js :

// mathUtils.js

export const PI = 3.14;

export function carre(x) {

return x \* x;

}

// Export par défaut

export default function message() {

console.log("📘 Module mathUtils chargé !");

}

**Étape 2 : L’utiliser dans ton script principal**

Dans ton index.js :

// index.js

import message, { PI, carre } from "./mathUtils.js";

message(); // 📘 Module mathUtils chargé !

console.log("PI =", PI);

console.log("Carré de 5 =", carre(5));

**Astuce :**  
Ajoute dans ton package.json :

{ "type": "module" }

Pour que Node.js reconnaisse la syntaxe import/export.

📘 Module mathUtils chargé !

mathUtils.js:8

PI = 3.14

index.js:4

Carré de 5 = 25

Observation :

On peut importer des fonctions d’un fichier à un autre , et les utiliser à condition d’avoir le mot clef export. Remarque que l’exportation nécessite le fait de préciser que le type des exportations est des < modules >.

**Tableaux et méthodes modernes**

**Exercice 5 – Manipulation de base**

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

fruits.push("kiwi");

fruits.pop();

console.log(fruits);

Observation :

L’affichage des éléments du tableau « fruits» aussi que l’utilisation des méthodes push et pop qui sert à manipuler ce dernier en ajoutant et supprimant des éléments avec la même approche d’une pile.

**Exercice 6 – map, filter, reduce**

const nums = [1, 2, 3, 4, 5];

// Multiplie chaque nombre par 2

console.log(nums.map(n => n \* 2));

// Garde seulement les nombres pairs

console.log(nums.filter(n => n % 2 === 0));

// Calcule la somme totale

console.log(nums.reduce((sum, n) => sum + n, 0));

Observation :

L’utilisation des méthodes map , filter et reduce qui sert à faire des traitements sans apporter de changements au tableau original comme map qui applique n\*2 sur chaque element du tableau, filter qui sert a filtrer que les nombres paires du tableau, reduce qui prend le tableau et le rend en scalaire , dans notre cas en faisant la somme.

**Exercice 7 – find, some, every**

console.log(nums.find(n => n > 3));

console.log(nums.some(n => n < 0));

console.log(nums.every(n => n > 0));

Observation :

Les méthodes find , some , every sont des fonctions qui sert a appliquer des conditions sur les éléments du tableau , find retourne le premier element du tableau qui respecte la condition , some retourne true si au moins un element respecte la condition, alors que every retourne true si tous les éléments respectent cette dernière.

**Objets et déstructuration**

**Exercice 8**

const user = { id: 1, name: "Ali", city: "Rabat" };

// Déstructuration

const { name, city } = user;

console.log(`${name} habite à ${city}`);

// Renommage

const { name: fullName, ...rest } = user;

console.log(fullName);

console.log(rest);

Observation :

Ce code illustre comment extraire, renommer et regrouper des propriétés d’un objet de manière concise grâce à la **déstructuration** et à l’**opérateur de reste**

**Asynchronisme et Promesses**

**Exercice 9 – Promise simple**

const p = new Promise((resolve) => {

setTimeout(() => resolve("Opération terminée !"), 2000);

});

p.then(result => console.log(result));

Observation :

Ce code montre comment exécuter une tâche asynchrone (ici, un délai de 2 secondes) avec une promesse, puis récupérer le résultat une fois l’opération terminée.

**Exercice 10 – async/await**

async function getUsers() {

try {

const res = await fetch("https://jsonplaceholder.typicode.com/users");

const data = await res.json();

console.log(data);

} catch (e) {

console.error("Erreur :", e.message);

}

}

getUsers();

Observation :

Ce code montre comment utiliser async/await pour effectuer une requête API de manière asynchrone, récupérer des données en JSON, et gérer les erreurs avec un bloctry**/**catch.

**Fonctions avancées et opérateurs modernes**

**Exercice 11 – Template literals**

const name = "Nadia";

const hour = new Date().getHours();

console.log(`Bonjour ${name}, il est ${hour}h`);

Observation :

affiche un message personnalisé avec le nom de la personne et l’heureactuelle, en utilisant les templateliterals et l’objet Date

**Exercice 12 – Spread / Rest**

const arr1 = [1, 2];

const arr2 = [...arr1, 3, 4];

console.log(arr2);

function sum(...numbers) {

return numbers.reduce((a, b) => a + b, 0);

}

console.log(sum(1, 2, 3, 4));

Observation :

Ce code montre l’usage de ... pour copier et étendre un tableau, puis pour regrouper des arguments dans une fonction.  
Il affiche [1, 2, 3, 4] et la somme des nombres passée, soit 10.

**Exercice 13 – Optional chaining et Nullish coalescing**

const settings = { theme: null };

console.log(settings.theme ?? "light"); // light

const user2 = { profile: { email: "x@y.com" } };

console.log(user2.profile?.email); // x@y.com

console.log(user2.address?.city); // undefined

Observation :

?? (**nullish coalescing**) : renvoie "light" si la valeur est null ou undefined.

?. (**optional chaining**) : permet d’accéder à une propriété sans erreur, même si l’objet intermédiaire

**Gestion des produits**

**Objectif**

Créer un script qui :

1. Définit une liste de produits (nom, prix, date d’expiration).
2. Filtre les produits non expirés.
3. Calcule la somme totale.
4. Affiche le résultat formaté.

**Exemple**

const produits = [

{ nom: "Lait", prix: 10, expireLe: "2025-12-01" },

{ nom: "Yaourt", prix: 5, expireLe: "2024-01-01" },

{ nom: "Jus", prix: 8, expireLe: "2026-02-15" },

];

const aujourdHui = new Date();

const valides = produits.filter(p => new Date(p.expireLe) > aujourdHui);

const total = valides.reduce((s, p) => s + p.prix, 0);

console.log("Produits valides :", valides);

console.log("Total :", total, "DH");

Code :

const produits = [

  { nom: "Huile d'olive", prix: 70, expireLe: "2026-05-10" },

  { nom: "Miel de Taznakht", prix: 120, expireLe: "2027-03-20" },

  { nom: "Dattes de Erfoud", prix: 90, expireLe: "2025-11-15" },

];

// 2. Filtrer les produits non expirés

const aujourdHui = new Date();

const produitsValides = produits.filter(p => new Date(p.expireLe) > aujourdHui);

// 3. Calculer la somme totale

const total = produitsValides.reduce((somme, p) => somme + p.prix, 0);

// 4. Afficher le résultat formaté

console.log("Produits valides :");

produitsValides.forEach(p => console.log(`- ${p.nom} (${p.prix} MAD)`));

console.log(`Total : ${total} MAD`);

**Remise du TP**

1. Crée un dépôt **GitHub ou GitLab**.
2. Mets ton dossier tp-js-base à jour.
3. Copie-colle ton URL dans ton Google Sheet de rendu.