FORMATION DEV WEB WEB MOBILE
CCP 2 - DÉVELOPPER LA PARTIE BACK-END D'UNE APPLICATION WEB

JAVASCRIPT

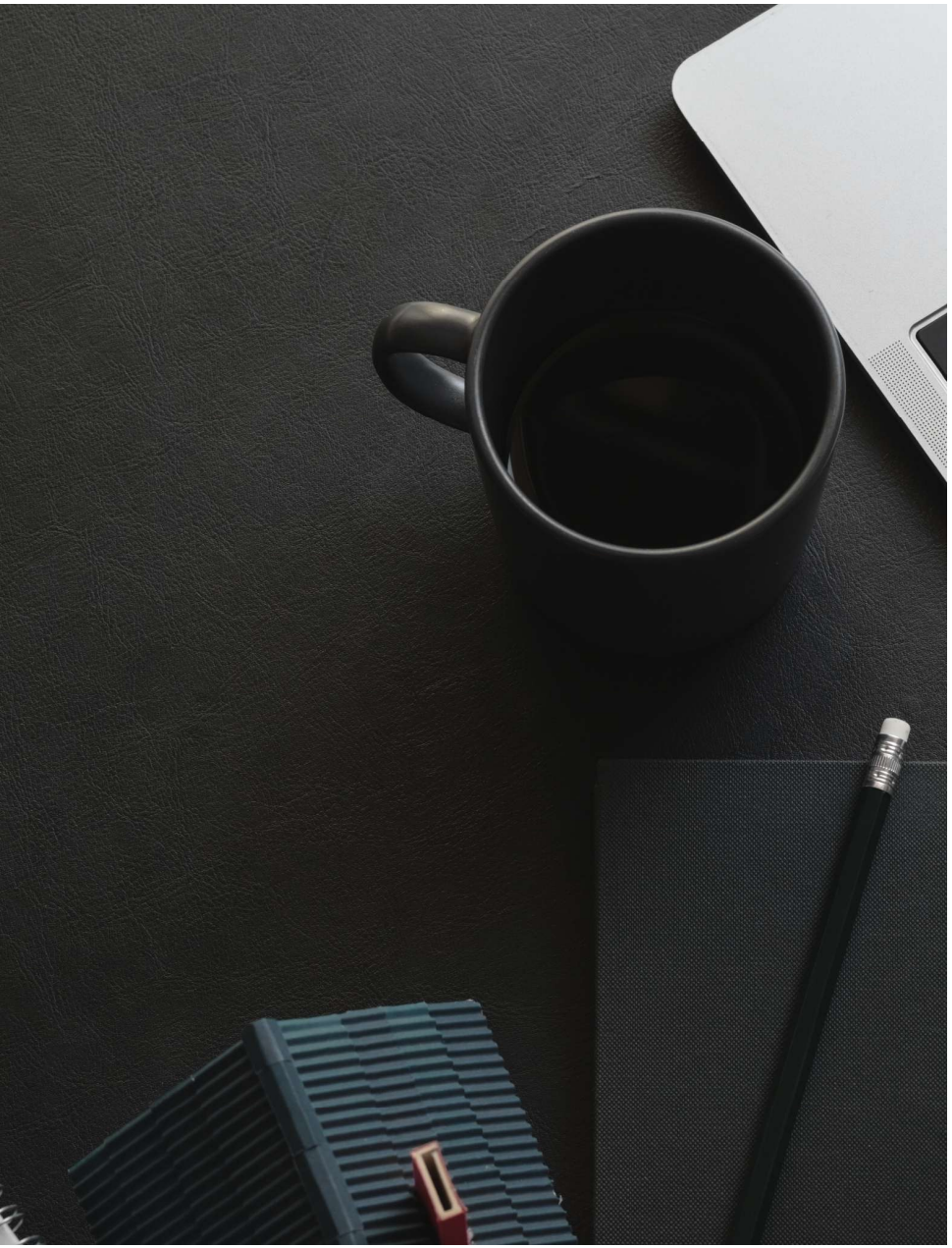
JOUR 1 - LES TYPES ET LES FONCTIONS

PRESENTE PAR NATACHA DESSE

AFEC

DEROULE MODULE

- Les types et les fonctions
- Manipuler le DOM et l'asynchrone



LES TYPES ET FONCTIONS

LES OBJECTIFS

Consolider les fondamentaux du langage JavaScript.

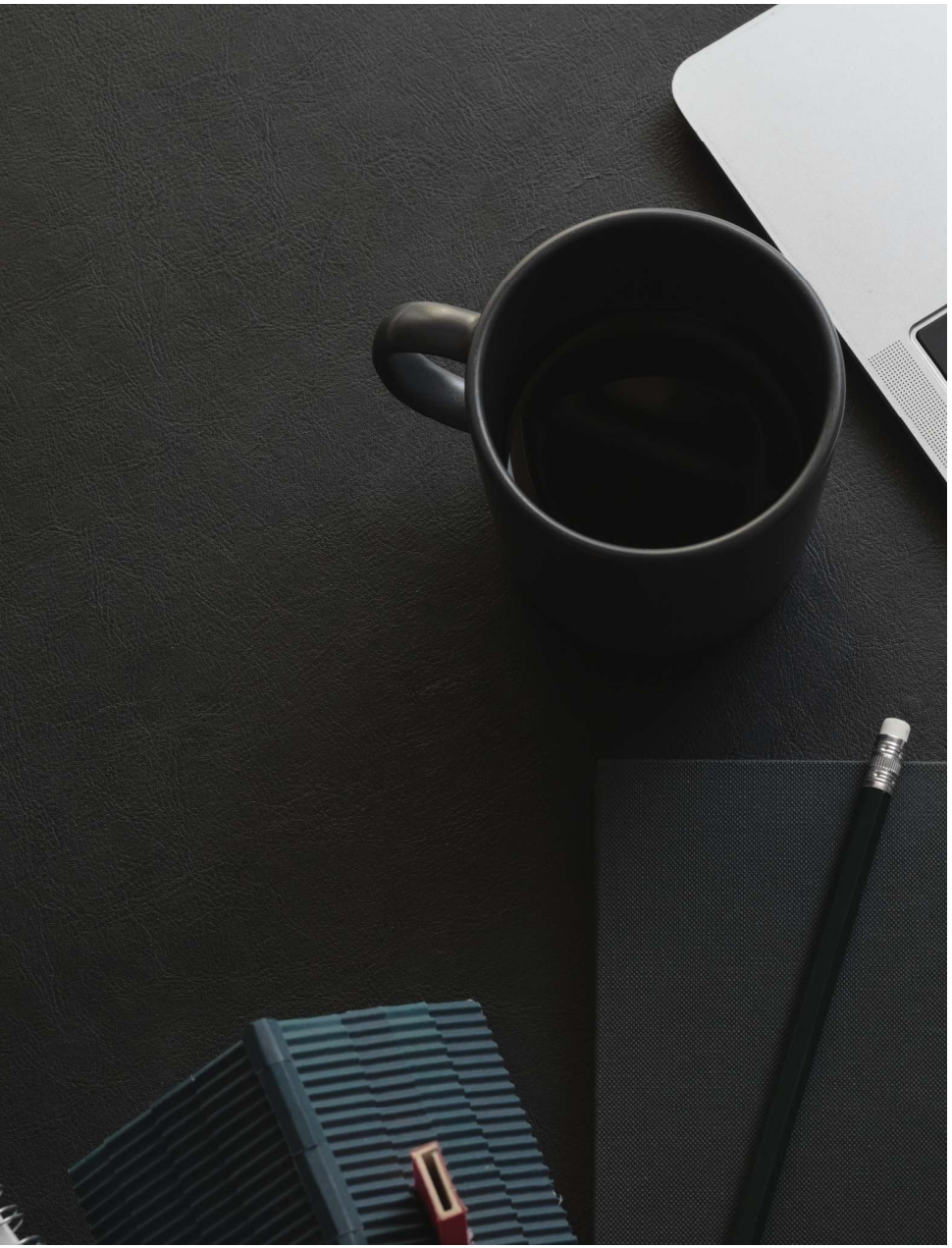
Approfondir la compréhension des fonctions, objets, tableaux

Pratiquer avec des exercices concrets (mini-projets, petits défis de code).

NUAGE DE MOTS

Que connaissez vous de Javascript ?

https://www.menti.com/aldpp4773piz



INTRODUCTION

QU'EST CE QUE LE JAVASCRIPT ?

Le JavaScript est un langage de programmation créé en 1995. Le JavaScript est aujourd'hui l'un des langages de programmation les plus populaires et il fait partie des langages **web** dits « standards » avec le HTML et le CSS.

Son évolution est gérée par le groupe **ECMA International** qui se charge de publier les standards de ce langage.

Pour définir ce qu'est le JavaScript et le situer par rapport aux autres langages, et donc pour comprendre les intérêts et usages du JavaScript il faut savoir que :

- Le JavaScript est un langage **dynamique**
- Le JavaScript est un langage (principalement) côté **client**
- Le JavaScript est un langage **interprété**
- Le JavaScript est un langage **prototypé**

LIBRAIRIE ET FRAMEWORK

Librairie

Une **librairie** ou « bibliothèque » JavaScript est un **ensemble** de **fichiers** de code JavaScript **homogènes** (= qui se concentrent sur un aspect particulier du langage) qu'on va devoir **télécharger** pour les utiliser.

Ces fichiers de code contiennent des **structures** de **code** prêtes à l'emploi qu'on va pouvoir utiliser immédiatement pour **gagner** du temps en développement.

Parmi les librairies les plus célèbres, on peut notamment citer jQuery.

Framework

Finalement, un framework est relativement similaire dans son but à une « super librairie ».

Les framework vont également nous fournir un ensemble de codes tout prêts pour nous faire gagner du temps en développement.

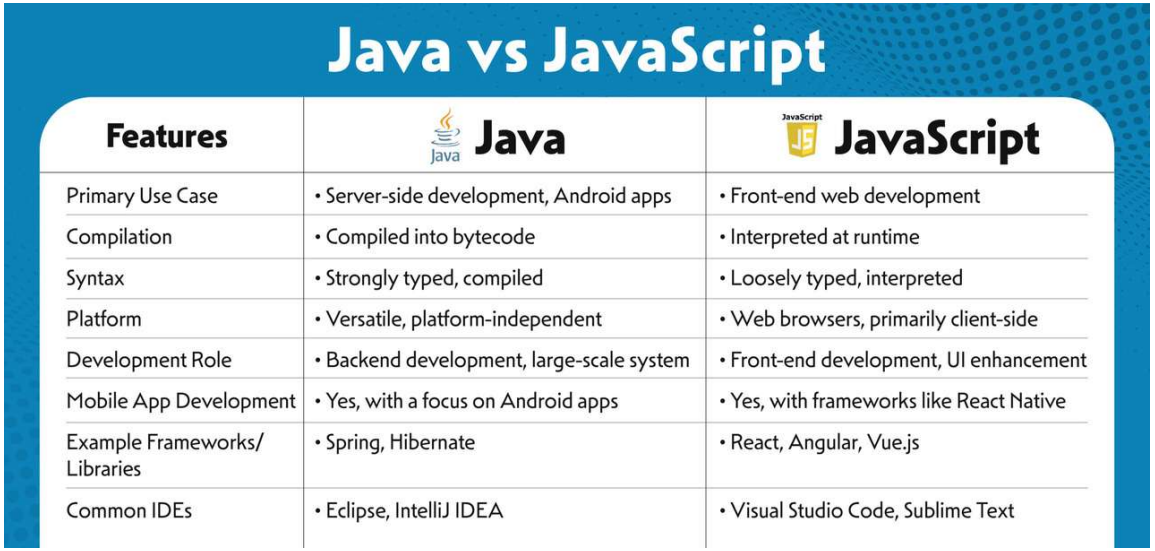
La grande différence entre un framework et une librairie réside dans l'**inversion du contrôle** : lorsqu'on télécharge une librairie, on peut l'utiliser comme on le souhaite en intégrant ses éléments à nos scripts tandis que pour utiliser un framework il faut **respecter son cadre** (ses règles). Les framework JavaScript les plus connus aujourd'hui sont Angular.js et React.js.

JAVASCRIPT != JAVA

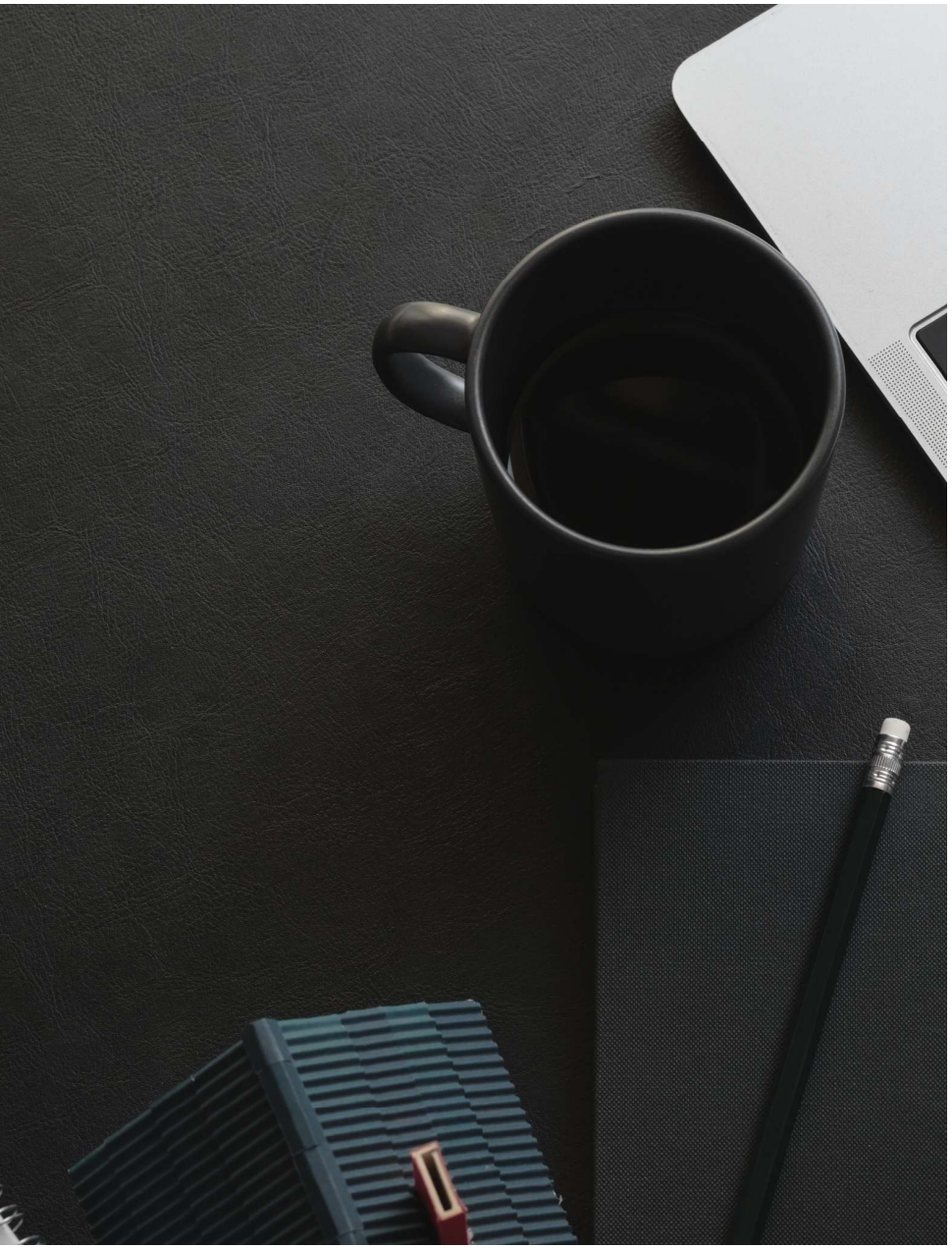
Ces deux langages, bien que syntaxiquement assez proches à la base, reposent sur des **concepts fondamentaux** complètement **différents** et servent à effectuer des tâches totalement différentes.

Pourquoi des noms aussi proches ? Java est une technologie créée originellement par **Sun Microsystems** tandis que JavaScript est un langage créé par la société **Netscape**.

Avant sa sortie officielle, le nom original du JavaScript était « **LiveScript** ». Quelques jours avant la sortie du LiveScript, le langage est renommé JavaScript.



A l'époque, Sun et Netscape étaient partenaires et le Java était de plus en plus populaire. Il est donc communément admis que le nom « JavaScript » a été choisi pour des **raisons marketing** et pour créer une **association** dans la tête des gens avec le Java afin que les deux langages se **servent mutuellement**.



VARIABLES ET TYPES

LES VARIABLES

Définition

Les variables sont des contenants pour stocker des valeurs de données.
En JavaScript, elles sont typées dynamiquement, ce qui signifie qu'elles peuvent stocker différents types de données.

Déclaration

Il existe trois façons de déclarer une variable JavaScript:

- var
- let
- const

Chaque méthode a des caractéristiques uniques en termes de portée, de hoisting et de mutabilité.

Portée

Variables Globales
Déclarée en dehors de toute fonction, et accessible de partout dans le code.

Variables Locales
Déclarée à l'intérieur d'une fonction et accessible uniquement à l'intérieur de cette fonction.

var

- Portée de fonction.
- Peut être redéclarée et actualisée.

let

- Portée du bloc.
- Peut être mise à jour mais pas redéclarée.

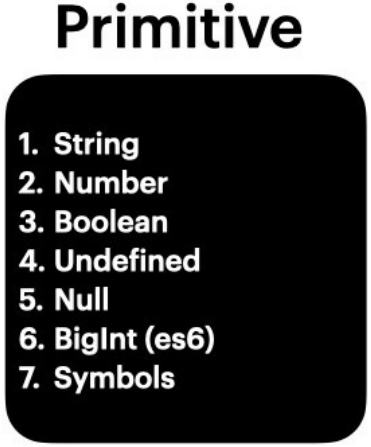
const

- Portée du bloc.
- Impossible à mettre à jour ou redéclarer.

VALEURS PRIMITIVES

Types primitifs

- number
- string
- boolean
- null
- undefined
- symbol
- bigint



typeof pour tester les types.

STRUCTURE DE DONNEES : OBJECTS & ARRAYS

Objets - Clés chaînes, valeurs de tous types

```
1 const user = {
2   name: "Alice",
3   age: 30
4 };
5
```

- Les clés sont des chaînes de caractères (ou des Symbol)
- Les valeurs peuvent être de n'importe quel type
- Accès par user.name ou user["name"]
- Pas d'ordre

Array - Tableau indexé numériquement

```
1 const fruits = ["apple",
  "banana", "cherry"];
2
```

- Ordonné
- Indice numérique (fruits[0])
- Taille dynamique
- Méthodes puissantes : map, filter, reduce, forEach, etc.

STRUCTURE DE DONNEES : MAP & SET

Map - Clés de n'importe quel type

```
const map = new Map();
map.set("name", "Alice");
map.set(42, "the answer");

```

- Les clés peuvent être des objets, fonctions, ou primitives
- Ordre d'insertion préservé
- Méthodes : set, get, has, delete, clear
- Meilleur pour gérer des paires clé/valeur que Object si clés ≠ chaînes

Set - Ensemble de valeurs uniques

```
const set = new Set([1, 2, 3, 3]);
console.log(set); // Set(3) {1, 2, 3}
```

- Pas de doublons
- Ordre d'insertion préservé
- Méthodes : add, has, delete, clear

STRUCTURE DE DONNEES : LEQUEL CHOISIR

Structure	Clé	Ordonné	Doublons autorisés ?	Remarques
Object	string/symbol	❌	✅	Base pour les données
Array	index	✅	✅	Pour les listes
Map	tout type	✅	✅ (clés uniques)	Pour les paires clés complexes
Set	n/a	✅	❌	Pour les ensembles uniques

LES OPERATEURS ARITHMETIQUES

Les opérateurs arithmétiques vont nous permettre d'effectuer toutes sortes d'opérations mathématiques entre les valeurs de type Number contenues dans différentes variables.

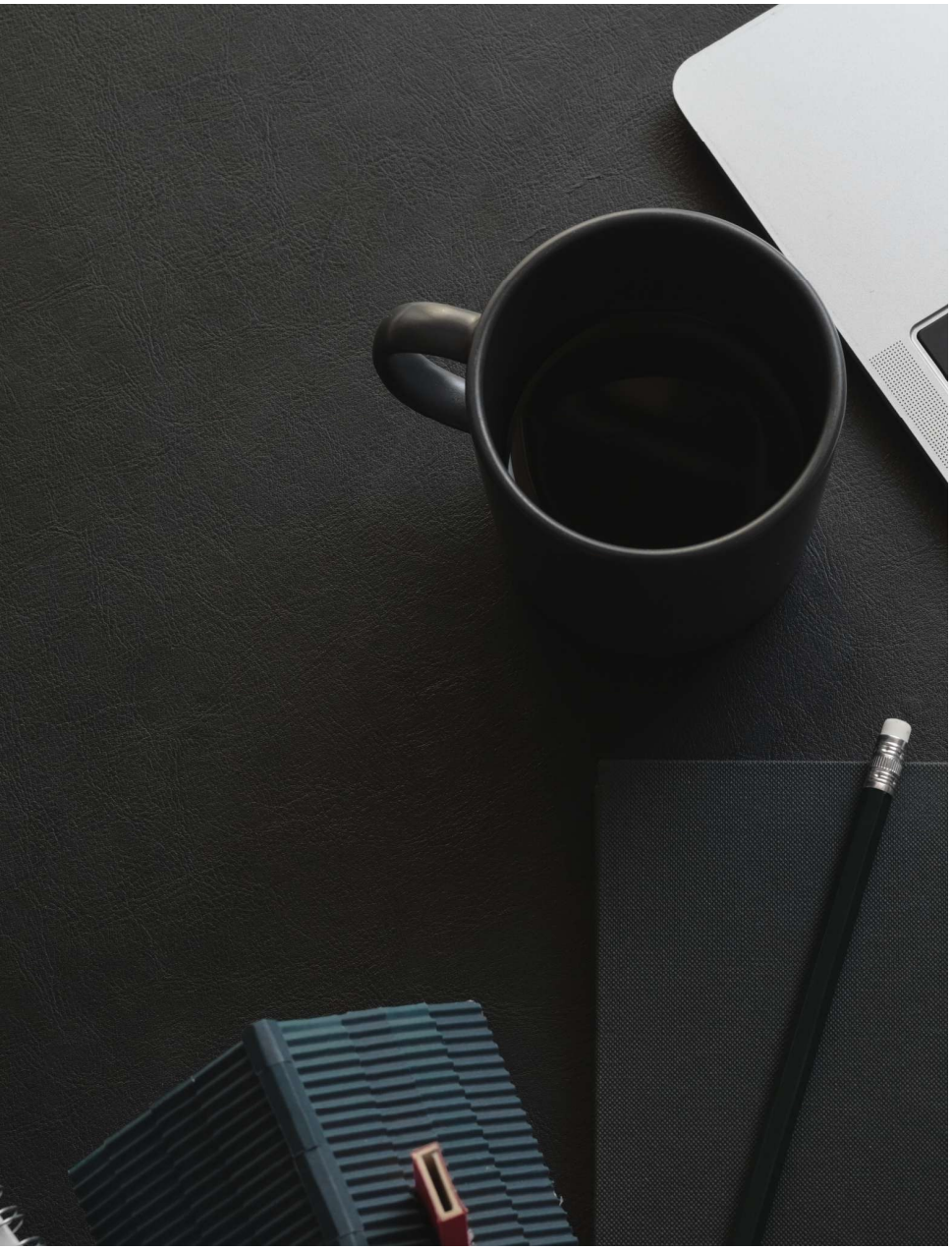
Opérateur	**Nom de l'opération associée**
+	Addition
–	Soustraction
*	Multiplication
/	Division
%	Modulo (reste d'une division euclidienne)
**	Exponentielle (élévation à la puissance d'un nombre par un autre)

LES OPERATEURS D'AFFECTATION

Les opérateurs d'affectation vont nous permettre, comme leur nom l'indique, d'affecter une certaine valeur à une variable.

Nous connaissons déjà bien l'opérateur d'affectation le plus utilisé qui est le signe =. Cependant, vous devez également savoir qu'il existe également des opérateurs « combinés » qui vont effectuer une opération d'un certain type (comme une opération arithmétique par exemple) et affecter en même temps.

Opérateur
+=
-=
*=
/=



STRUCTURE DE CONTROLE

IF / ELSE / ELSE IF

```
const age = 18;

if (age < 18) {
  console.log("Mineur");
} else if (age === 18) {
  console.log("Tout juste majeur !");
} else {
  console.log("Majeur");
}
```

Opérateurs de comparaisons

'==	Permet de tester l'égalité sur les valeurs
'===	Permet de tester l'égalité en termes de valeurs et de types
!=	Permet de tester la différence en valeurs
<>	Permet également de tester la différence en valeurs
!==	Permet de tester la différence en valeurs ou en types
<	Permet de tester si une valeur est strictement inférieure à une autre
>	Permet de tester si une valeur est strictement supérieure à une autre
<=	Permet de tester si une valeur est inférieure ou égale à une autre
>=	Permet de tester si une valeur est supérieure ou égale à une autre

Opérateurs logiques

Opérateur	**Nom**	**Fonction**
&&	ET logique	Retourne vrai si **les deux opérandes** sont vrais
		Retourne la **première valeur fausse**, ou la dernière
||	OU logique	Retourne vrai si **au moins un opérande** est vrai
		Retourne la **première valeur vraie**
!	NON logique	Inverse la valeur logique
		Peut convertir une valeur en booléen

OPERATEUR TERNAIRE

Opérateur ternaire

condition ? valSiVrai : valSiFaux

```
condition ? valSiVrai : valSiFaux
```

```
const age = 20;
const status = age >= 18 ? "majeur" :
"mineur";
```

SWITCH

Pour comparer plusieurs valeurs

```
1 const fruit = "banane";
2
3 switch (fruit) {
4   case "pomme":
5     console.log("C'est une pomme");
6     break;
7   case "banane":
8     console.log("C'est une banane");
9     break;
10  default:
11    console.log("Fruit inconnu");
12 }
```

BOUCLES

Boucle for

```
for (let i = 0; i < 3; i++) {
  console.log(i); // 0, 1, 2
}
```

Boucle while

```
let i = 0;
while (i < 3) {
  console.log(i);
  i++;
}
```

Boucle do...while

Exécutée au moins une fois

```
let i = 0;
do {
  console.log(i);
  i++;
} while (i < 3);
```

Boucle for ... of

pour les objets itérables (tableaux, chaînes, etc

```
const fruits = ["pomme", "banane",
"cerise"];
for (const fruit of fruits) {
  console.log(fruit);
}
```

Boucle for ... in

pour les objets et non pas les tableaux. Pour itérer sur les clés d'un objet

```
const person = { name: "Alice", age: 30 };
for (const key in person) {
  console.log(`${key}: ${person[key]}`);
}
```

CONTROLE DE BOUCLE

Break

Interrompt la boucle

```
1 for (let i = 0; i < 10; i++) {
2   if (i === 3) break;
3   console.log(i); // 0, 1, 2
4 }
```

Continue

Passe à l'itération suivante

```
1 for (let i = 0; i < 5; i++) {
2   if (i === 2) continue;
3   console.log(i); // 0, 1, 3, 4
4 }
```

TRY ... CATCH

Permet d'intercepter les erreurs

```
try {
  throw new Error("Oups !");
} catch (err) {
  console.error("Erreur attrapée :", err.message);
} finally {
  console.log("Bloc finally exécuté");
}
```

RECUPITULATIF

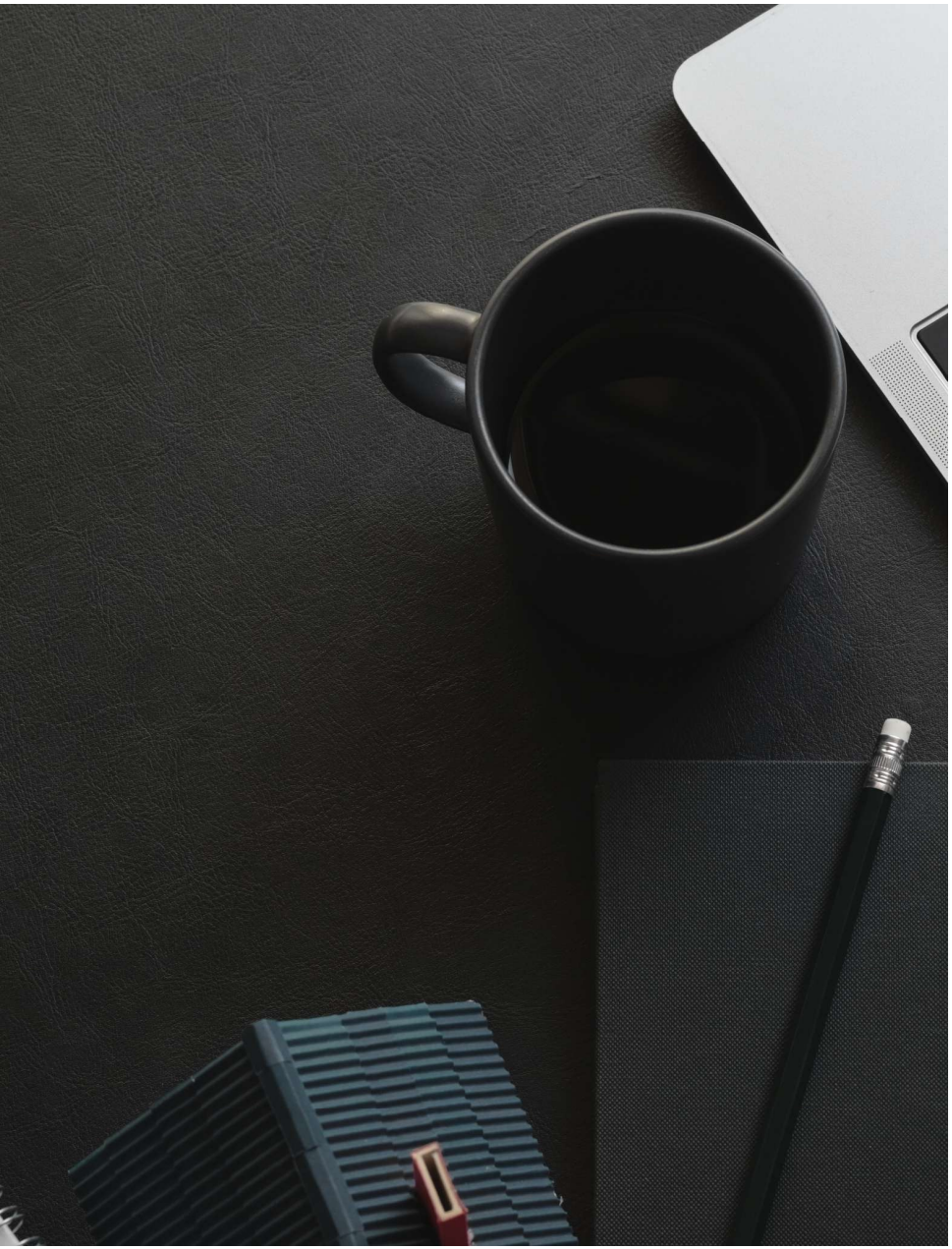
Structure	Utilité
if / else	Test conditionnel simple ou multiple
switch	Choix parmi plusieurs cas
for	Boucle avec compteur
while	Boucle tant que condition vraie
for...of	Itération sur tableau ou chaîne
for...in	Itération sur propriétés d'objet
break	Interrompt une boucle
continue	Saute à l'itération suivante
try...catch	Gérer les exceptions

EXERCICES

Variables conditions boucles

ENONCES DES EXERCICES

exercices_variables_conditions_boucles.js



LES FONCTIONS

LES FONCTIONS A QUOI CA SERT ?

Une **fonction** correspond à un bloc de code **nommé** et **réutilisable** et dont le but est d'**effectuer** une **tâche précise**. En JavaScript, comme dans la plupart des langages les supportant, nous allons très souvent utiliser des fonctions car celles-ci possèdent de nombreux atouts que l'on va énumérer par la suite.

Le langage JavaScript dispose de **nombreuses fonctions** que nous pouvons utiliser pour effectuer différentes tâches. Les fonctions définies dans le langage sont appelées **fonctions prédéfinies** ou fonctions prêtes à l'emploi car il nous suffit de les appeler pour nous en servir

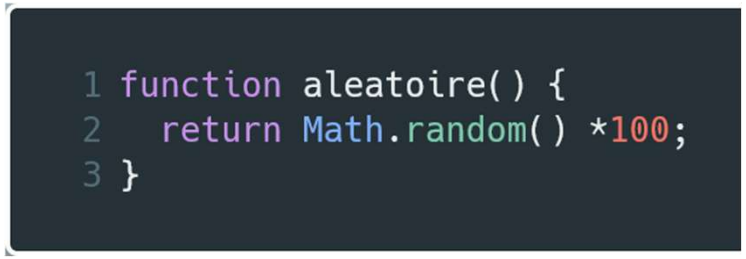
Par exemple, le JavaScript dispose d'une fonction nommée *random*() (qui appartient à l'objet *Math*) et qui va générer un nombre décimal aléatoire entre 0 et 1 ou encore d'une fonction *replace*() (qui appartient cette fois-ci à l'objet *String*) qui va nous permettre de chercher et de remplacer une expression par une autres dans une chaine de caractères.

```
1 let prez = 'Bonjour, je suis Pierre';
2
3 const rnd = Math.random();
4
5 let prez2 = prez.replace('Pierre',
  'Mathilde');
```

CREER NOS PROPRES FONCTIONS ?

En plus des nombreuses fonctions JavaScript prédéfinies et immédiatement utilisables, nous allons pouvoir créer nos propres fonctions en JavaScript lorsque nous voudrons effectuer une tâche très précise.

Pour pouvoir utiliser une fonction personnalisée, en pratique, il faut déjà la définir. Pour définir une fonction, on va utiliser le mot clef function suivi du nom que l'on souhaite donner à notre fonction puis d'un couple de parenthèses dans lesquelles on peut éventuellement définir des paramètres (je reviendrai là-dessus plus tard) et finalement d'un couple d'accolades dans lesquelles on va placer le code de notre fonction.



LES DIFFERENTS TYPES

Traditionnelle

```
1 function bonjour(nom) {
2   return "Bonjour " + nom;
3 }
4 console.log(bonjour("Alice")); // Bonjour
  Alice
```

Avantage : peut être appelée avant sa définition grâce au hoisting.

Anonyme

```
1 const saluer = function(nom) {
2   return "Salut " + nom;
3 };
4 console.log(saluer("Bob")); // Salut Bob
```

Elle est affectée à une variable.
Pas “hoistée” => ne peut pas être appelé avant sa définition

Fléchée

```
1 const direHello = (nom) => {
2   return "Hello " + nom;
3 };
4 console.log(direHello("Claire")); // Hello
  Claire
```

Syntaxe raccourcie quand il n’y a qu’un seul paramètre et une seule instruction :

```
1 const rapide = nom => "Yo " + nom;
2 console.log(rapide("Dan")); // Yo Dan
```

LE HOISTING ET LA PORTEE DES VARIABLES

Var - portée fonctionnelle

```
function test() {
  if (true) {
    var x = 5;
  }
  console.log(x); // 5 ← car var est visible dans toute la fonction
}
```

🚫 Évite var aujourd'hui : peut causer des bugs liés à sa portée large.

let - portée bloc

```
if (true) {
  let y = 10;
  console.log(y); // 10
}
console.log(y); // ❌ ReferenceError : y is not defined
```

utilisation de let pour des variables changeantes

const - portée bloc

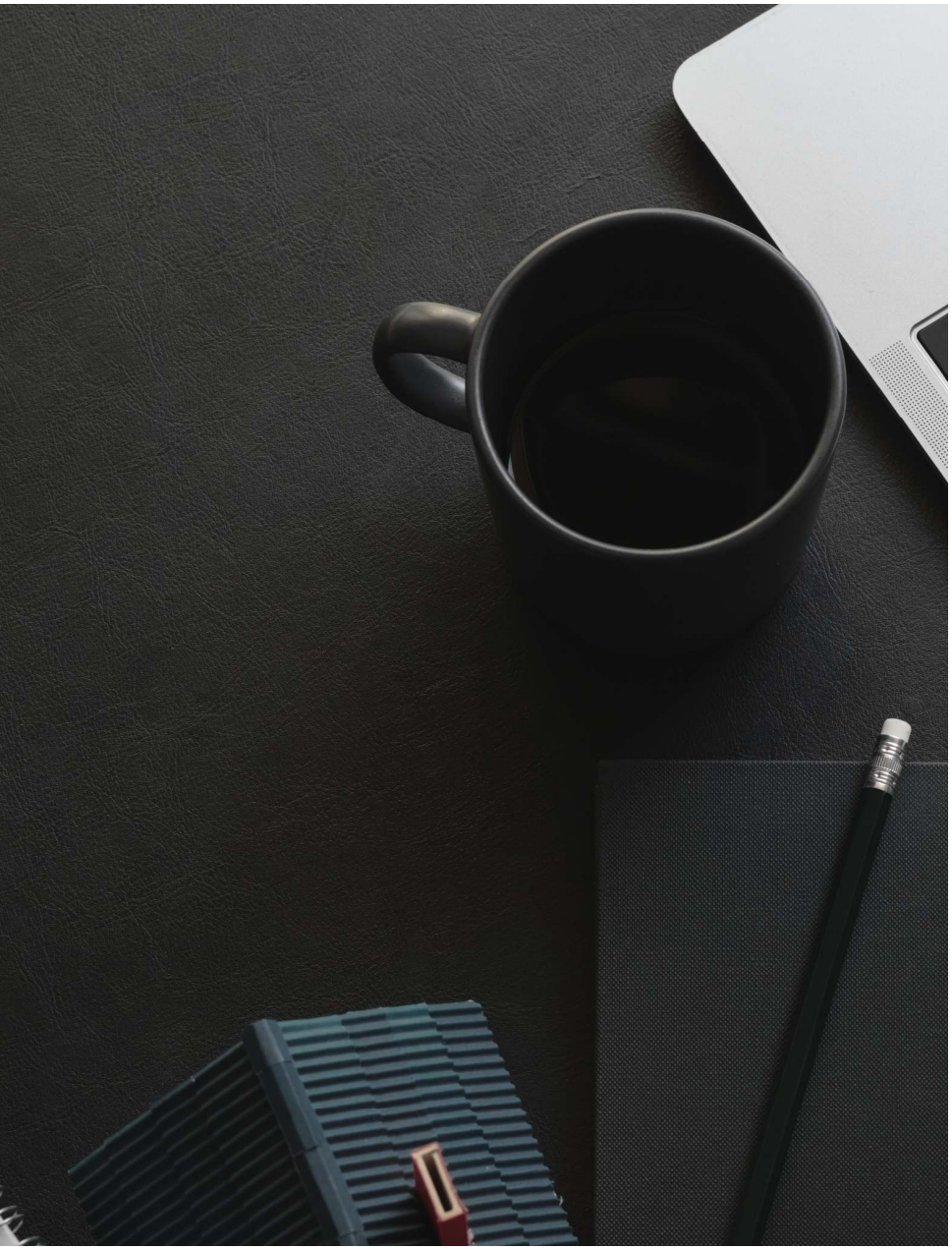
```
const nom = "Alice";
nom = "Bob"; // ❌ TypeError : assignment to constant variable
```

Le hoisting

```
console.log(a); // undefined
var a = 5;

console.log(b); // ❌ ReferenceError
let b = 10;
```

➡️ var est déclaré en haut (hoisting), mais pas initialisé.
➡️ let et const ne sont ni accessibles ni initialisables avant la déclaration.



DEBUGGER

LES METHODES CONSOLE

console.log(valeur)

```
1 const x = 5;
2 console.log("La valeur de x est :", x);
```

console.table(objetOuTableau)

Affiche un tableau ou un objet sous forme de tableau bien lisible

```
1 const fruits = ["pomme", "banane", "cerise"];
2 console.table(fruits);
3
4 const utilisateurs = [
5   { nom: "Alice", age: 25 },
6   { nom: "Bob", age: 30 },
7 ];
8 console.table(utilisateurs);
9
```

console.error(“une erreur est survenue)

```
1 console.error("Une erreur s’est produite !");
```

debugger

Le navigateur s’arrête à la ligne debugger comme à un point d’arrêt

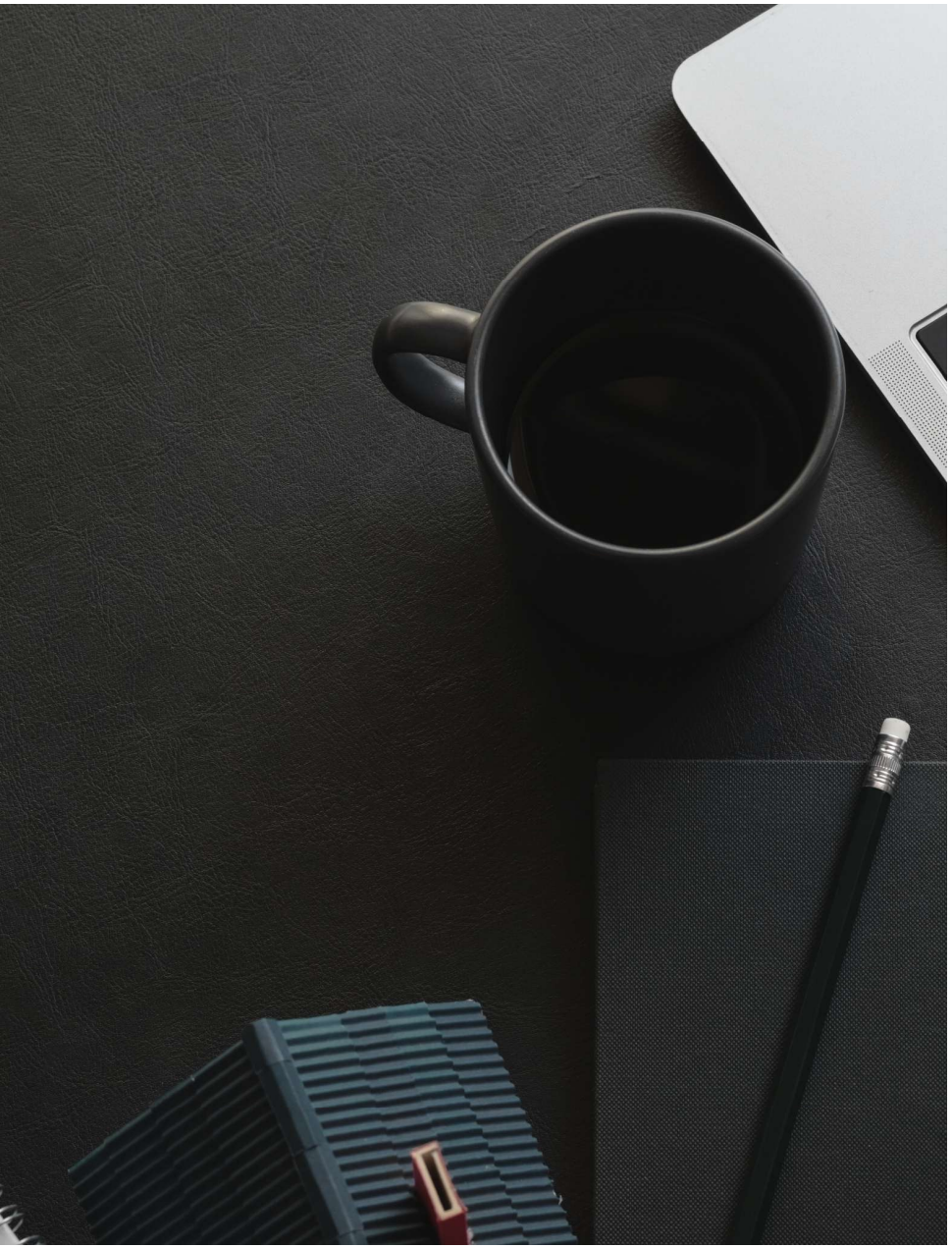
```
1 function somme(a, b) {
2   debugger;
3   return a + b;
4 }
5 somme(2, 3);
```

EXERCICES

Variables conditions boucles

ENONCES DES EXERCICES

exercices_fonctions.js



TABLEAUX ET OBJETS

LES TABLEAUX

Un tableau est une liste indexée (chaque élément a un indice numérique).

```
const animaux = ["chat", "chien"];
animaux.push("lapin");        // ["chat", "chien", "lapin"]
animaux.pop();                // ["chat", "chien"]
animaux.unshift("oiseau");    // ["oiseau", "chat" "chien"]
animaux.shift();              // ["chat", "chien"]
```

splice

```
const arr = ["a", "b", "c", "d"];
arr.splice(1, 2); // Supprime 2 éléments à partir de l'indice 1
console.log(arr); // ["a", "d"]
```

slice

```
const part = arr.slice(0, 2); // Copie de "a" et "d"
console.log(part); // ["a", "d"]
```

Méthode	**Effet**
push(el)	Ajoute à la fin
pop()	Retire le dernier élément
unshift(el)	Ajoute au début
shift()	Retire le premier élément
splice(i,n)	Supprime/modifie à partir de l'indice i
slice(i,j)	Retourne une **copie** de la portion [i, j)

LES METHODES AVANCEES

map - transforme chaque élément

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

filter - garde certains éléments

```
1 const pairs = nombres.filter(n => n % 2 === 0);
2 console.log(pairs); // [2]
```

reduce - cumule une valeur finale

```
1 const somme = nombres.reduce((acc, val) => acc + val, 0);
2 console.log(somme); // 6
```

LES OBJETS

Un objet est une collection de paires clé/valeur.

```
const utilisateur = {
  nom: "Alice",
  age: 30
};
```

Acces aux propriétés

```
console.log(utilisateur.nom);       // "Alice"
console.log(utilisateur["age"]);    // 30
```

La notation [ ] permet d'utiliser une clé dynamique (variable)

Boucle for... in

```
for (let cle in utilisateur) {
  console.log(cle + " : " + utilisateur[cle]);
}
// nom : Alice
// age : 30
```

Déstructuration

Depuis un objet :

```
const { nom, age } = utilisateur;
console.log(nom); // "Alice"
console.log(age); // 30
```

Depuis un tableau ::

```
const couleurs = ["rouge", "vert", "bleu"];
const [c1, c2] = couleurs;
console.log(c1); // rouge
console.log(c2); // vert
```

EXERCICES

Tableaux - Objets

ENONCES DES EXERCICES

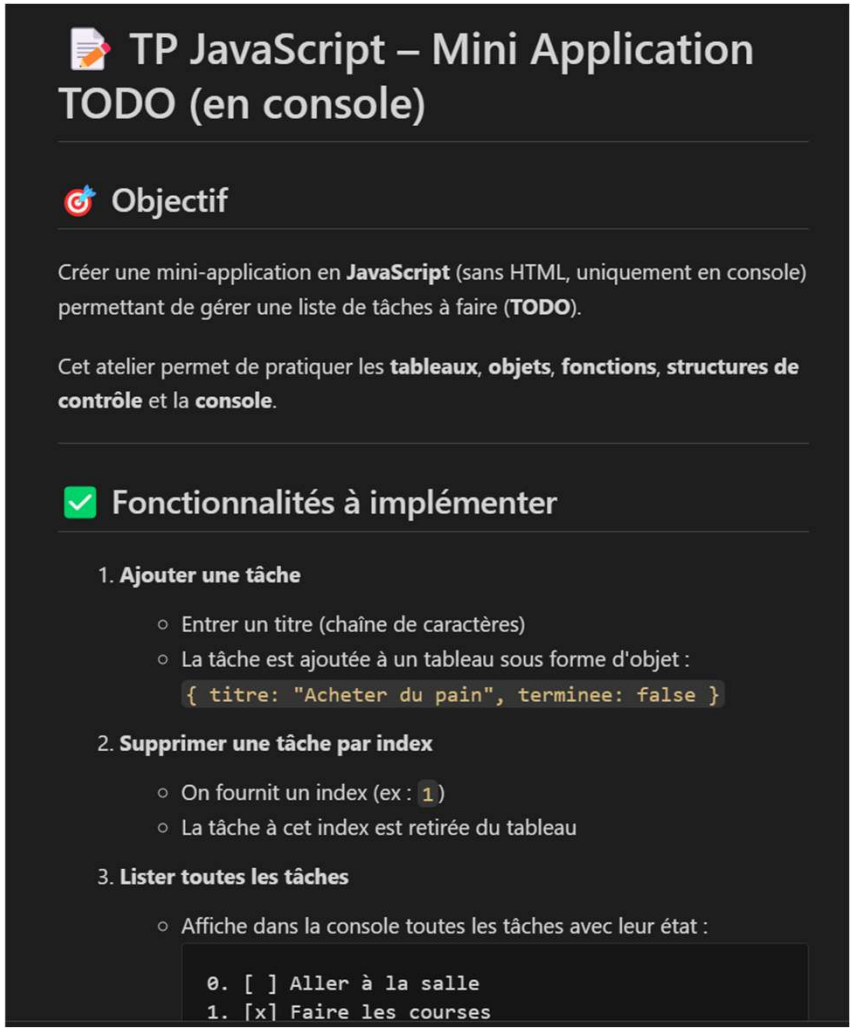
exercices_fonctions.js

ATELIER

Réalisation d'une TODO console

CONSIGNES

Nous allons réaliser une TODO console.
Suivez le guide



📝 TP JavaScript – Mini Application TODO (en console)

🎯 Objectif

Créer une mini-application en **JavaScript** (sans HTML, uniquement en console) permettant de gérer une liste de tâches à faire (**TODO**).

Cet atelier permet de pratiquer les **tableaux**, **objets**, **fonctions**, **structures de contrôle** et la **console**.

✅ Fonctionnalités à implémenter

1. **Ajouter une tâche**
   - Entrer un titre (chaîne de caractères)
   - La tâche est ajoutée à un tableau sous forme d'objet : `{ titre: "Acheter du pain", terminee: false }`
2. **Supprimer une tâche par index**
   - On fournit un index (ex : `1`)
   - La tâche à cet index est retirée du tableau
3. **Lister toutes les tâches**
   - Affiche dans la console toutes les tâches avec leur état :

```
0. [ ] Aller à la salle
1. [x] Faire les courses
```

QUIZZ