Introduction à *JavaScript*

Qu'est-ce que le JavaScript ?

- JavaScript est un langage de programmation web ou d'interprétation
- JavaScript est utilisé pour ajouter de l'interactivité et de la dynamique aux sites web.
- JavaScript peut également être utilisé côté serveur grâce à des environnements comme Node.js pour créer des applications web complètes.

Attention ! Ne pas confondre avec le langage Java

Ou coder en JavaScript ?

- la balise `<script>` délimite code JavaScript
- peut être insérée soit dans l'entête (`<head>`) soit dans le corps (`<body>`) de la page HTML
- les scripts sont exécutés dans leur ordre d'apparition dans la page

Placer les scripts dans l'en tête pour qu'ils soient chargés et exécutés avant la construction et l'affichage de la page

Par défaut, les scripts placés dans le body sont exécutés au fur et à mesure que celui-ci est chargé et que le DOM est construit.
→ met en attente le moteur d'analyse HTML/CSS

Souvent les scripts sont placés à la fin du corps de la page (juste avant </body>) → améliorer la vitesse de chargement de la page.

Images et css en provenance de différents sites peuvent être téléchargées simultanément (en parallèle) mais une fois que le navigateur a rencontré une balise script ce n'est (par défaut) plus possible → les scripts bloquent les téléchargements parallèles

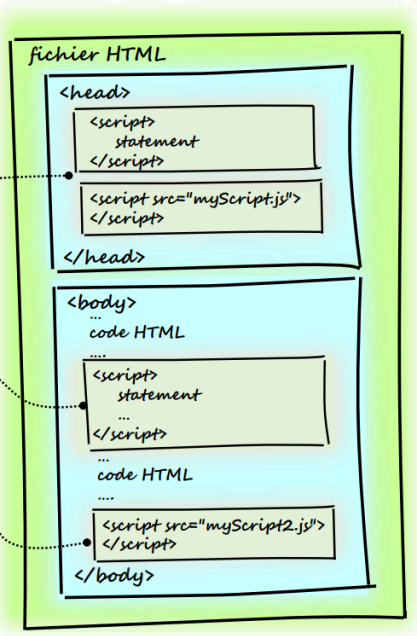


figure d'après "Head First HTML 5 programming"
Eric Freeman, Elisabeth Robson- Ed. O'Reilly, 2011

Fichier *JavaScript* Externe

Le JavaScript est généralement écrit dans des fichiers distincts avec l'extension **`.js`**.

Tout d'abord, créez un fichier séparé avec l'extension **`.js`**, par exemple **script.js**

Dans votre fichier HTML, utilisez la balise **<script>** dans la section **<head>** pour lier votre fichier JS externe :

```
<head>
    <meta charset="UTF-8">
    <title>Site de Voyage</title>
    <link rel="stylesheet" href="styles.css">
    <script src="script.js"></script>
</head>
```

Les commentaires

Tout comme en HTML et en CSS, dans le code JavaScript, vous pouvez écrire des commentaires qui ne seront pas interprétés par le navigateur. Ces commentaires sont destinés à fournir des explications aux autres développeurs sur le fonctionnement du code (et à vous-même, si vous revenez sur votre code après un certain temps et avez du mal à vous souvenir de ce que vous avez fait). Ils sont extrêmement utiles et devraient être utilisés régulièrement, surtout pour des applications de grande envergure. Il existe deux types de commentaires :

1. Commentaire sur une ligne

```
// Ceci est un commentaire
```

2. Commentaires sur plusieurs lignes

```
/*
  Ceci est un commentaire
  sur deux lignes
*/
```

Syntaxe du *JavaScript*

```
// Fonction: créer un nouveau paragraphe et l'ajouter en bas du HTML

function createParagraph() {
    let para = document.createElement("p");
    para.textContent = "Cliquez ICI !";
    document.body.appendChild(para);
  }

  /*
    1. Regrouper les coordonnées de tous les boutons de la page et les organiser en tableau
    2. Faire une boucle dans ce tableau et ajouter un "click event listener" à chaque bouton

    Quand le bouton est cliqué, la fonction "createParagraph()" sera exécutée
  */

  let buttons = document.querySelectorAll("button");

  for (let i = 0; i < buttons.length; i++) {
    buttons[i].addEventListener("click", createParagraph);
  }
```

Dans cet exemple, ce code JavaScript crée une fonction nommée **`createParagraph( )`** et associe cette fonction à tous les boutons présents dans la page HTML.

Les variables

Une variable est un espace alloué qui nous permet de stocker des informations de différents types, de manière temporaire.

Comment Déclarer une variable en Javascript?

Pour déclarer une variable en JavaScript on commence par le mot clef **var/let/const** suivi du **nom** de la variable. Une variable est constituée de deux parties : son **nom** et sa **valeur**.

```
const nom = 'jean'
let prenom = 'Herard'
var age = 12
```

*Il n'est pas nécessaire de préciser les fins de lignes en JavaScript c'est-à-dire de mettre des points virgules **(;)** à la fin de chaque ligne, mais il est recommandé de le faire malgré que les fins de lignes soient automatiquement détectées par l'interpréteur de JavaScript.*

Mot clef

Les variables sont des conteneurs pour stocker des données et peuvent être déclarées de 3 manières :

- En utilisant **var**
- En utilisant **let**
- En utilisant **const**

Quand utiliser var, let ou const ?

1. Déclarez toujours les variables

2. Toujours utiliser **const** si la valeur ne doit pas être modifiée

3. Toujours utiliser **const** si le type ne doit pas être modifié (Tableaux et Objets)

4. N'utilisez que **let** si vous ne pouvez pas l'utiliser **const**

5. À utiliser uniquement **var** si vous DEVEZ prendre en charge les anciens navigateurs.

Type de variable en JavaScript

On peut stocker différents types d'informations dans une variable comme:

- Les **nombres** comme par exemple : **42**
- Les **chaînes de caractères** comme par exemple : **'Une nouvelle chaîne'**
- Les **booléens** comme par exemple : **true**
- Les **tableaux** comme par exemple : **[' Paris ', ' Lille ']**
- Les **objets** comme par exemple : **{ age : 43 }**
- Il existe aussi le type **undefined**, un peu particulier, il permet de déterminer si une variable est définie ou non.
- On peut utiliser l'opérateur **typeof** pour savoir le type de données

Les nombres

```
let nb = -34
let nb1= 3.14
let nb2c = 65
```

- Des entiers **(int)** / **65 || -34**
- Des nombre réels **(float)** / **3.14**

Lorsqu' on manipule des réels, pour marquer la partie décimale on utilise un point.

Les chaînes de caractère

On utilise les chaînes de caractères pour stocker des mots ou des phrases. Pour indiquer à l'interpréteur qu'on est en train de travailler avec une chaîne de caractères, on entoure le texte par des guillemets simples **‘ ‘** ou des guillemets doubles **” “**.

```
var nom = "Denzel Washington"
let phrase = 'bonjour comment ca va?'
```

Si votre chaîne de caractère contient des **‘ ou “** , il faut les écharper.

```
let chaine1 = "C'est bon"
let chaine2 = 'C\'est bon aussi'
```

Les booléens

Un booléen est un type de variable à deux états qui nous permet de stocker une information qui peut être vraie **(true en anglais)** ou fausse **(false en anglais)**

```
let vrai = true
let faux = false
```

Les Tableaux [‘ ’]

Les tableaux permettent de stocker une liste d'informations.

```
var employe = ['Jacques', 'Herard', 'Pierre']
var test= [true, 34, 'Melo']
```

Les Objets { }

```
let eleve = {
  nom: 'Jacques',
  age: 22,
  notes: [12, 34, 20]
}
```

Un objet est une collection de données et/ou de fonctionnalités associées. Il contient habituellement différentes parties comme des variables et des actions (qu'on appelle propriétés et méthodes quand elles sont à l'intérieur de ces objets).

Les Opérateurs

Les opérateurs sont des symboles spéciaux utilisés pour effectuer des opérations sur des variables et des valeurs. En JavaScript, ils sont largement utilisés et comprennent plusieurs catégories.

Opérateurs Arithmétiques

Ces opérateurs sont utilisés pour effectuer des opérations mathématiques courantes.

- Addition (+)
- Soustraction (-)

Opérateur	Description	Exemple	Résultat
+	Addition	`5 + 2`	7
-	Soustraction	`5 - 2`	3
*	Multiplication	`5 * 2`	10
/	Division	`5 / 2`	2.5
%	Modulo (reste de division)	`5 % 2`	1
++	Incrémentation	`let a = 5; a++`	6
--	Décrémentation	`let b = 2; b--`	1

Opérateurs d'Affectation

Ils sont utilisés pour attribuer des valeurs à des variables JavaScript.

Opérateur	Description	Exemple	Résultat
=	Affectation	`let a = 10`	10
+=	Addition puis affectation	`a += 5`	15
-=	Soustraction puis affectation	`a -= 2`	13

Opérateurs de Comparaison

Ces opérateurs comparent deux valeurs et renvoient un booléen.

Opérateur	Description	Exemple	Résultat
==	Égalité	`5 == "5"`	true
===	Égalité stricte	`5 === "5"`	false
!=	Inégalité	`5 != "5"`	false
!==	Inégalité stricte	`5 !== "5"`	true
>	Supérieur	`5 > 2`	true
<	Inférieur	`5 < 2`	false

Opérateurs Logiques

Utilisés pour déterminer la logique entre des variables ou des valeurs.

Opérateur	Description	Exemple	Résultat
&&	ET logique	`(5 > 0 && 2 < 10)`	true
||	OU logique	`(5 > 10 || 2 < 10)`	true
!	NON logique	`!(5 > 10)`	true

Exercices Pratiques :

Exercice 1 : Manipulation des variables

Consigne :

- Déclarez une variable **age** et initialisez la avec votre âge.
- Ensuite, créez une autre variable **ageFutur** qui stockera votre âge dans 10 ans et affichez les deux variables dans la console.

```
let age = 25; // Remplacez par votre âge
let ageFutur = age + 10;

console.log("Âge actuel : " + age);
console.log("Âge dans 10 ans : " + ageFutur);
```

Exercice 2 : Calcul de l'âge en jours

Consigne :

- Déclarez une variable **age** avec votre âge actuel. -
- Ensuite, calculez et stockez dans une variable **ageEnJours** votre âge en jours (en supposant une année de 365 jours) et affichez-le dans la console.

Exercice 3 : Concaténation de chaînes de caractères

Consigne :

- Déclarez deux variables **prenom** et nom avec vos prénoms et nom.
- Ensuite, combinez-les dans une variable **nomComplet** pour former votre nom complet et affichez-le dans la console.

Exercice 4 : Utilisation de variables pour stocker des booléens

Consigne :

- Déclarez une variable **estFavori** et attribuez-lui la valeur booléenne **true** si vous aimez le chocolat, sinon attribuez-lui **false**.
- Ensuite, affichez dans la console un message disant si le chocolat est votre favori ou non.

(L'utilisation du if/else est nécessaire)

Les conditions en JavaScript

Contexte : Dans le développement web, la logique conditionnelle est un pilier fondamental. En JavaScript, les structures conditionnelles permettent à nos programmes de prendre des décisions basées sur des critères spécifiques.Que ce soit pour valider des données utilisateur, gérer des flux de navigation, ou exécuter des calculs complexes, les conditions sont omniprésentes et essentielles.

Les structures conditionnelles permettent à votre programme de prendre des décisions en fonction de certaines conditions. Voici les principales structures conditionnelles en JavaScript :

Instruction { if }

L'instruction **if** est la plus simple des structures conditionnelles. Elle exécute un bloc de code si une condition spécifiée est vraie.

```
if (condition) {
    // bloc de code à exécuter si la condition est vraie
}
```

Exemple :

Supposons que l'on souhaite afficher la catégorie d'une personne à un utilisateur en fonction de son âge, en s'assurant que la variable **"age"** est déclarée.

```
let age = 18
if (age == 18) {
    "Il est Majeur!"
}
```

Instruction { else }

```
if (condition) {
    // bloc de code si la condition est vraie
} else {
    // bloc de code si la condition est fausse
}
```

L'instruction **else** est utilisée avec **if** pour exécuter un bloc de code lorsque la condition if est fausse.

Exemple :

Maintenant, on souhaite afficher un autre message à l'utilisateur dans le cas où l'âge de la personne n'est pas **égal à 18**, c'est-à-dire lorsque l'expression est fausse.

```
let age = 15

if (age == 18 ) {
    "Il est Majeur!"
} else {
    "Il est Mineur"
}
```

Instruction { else if }

L'instruction **else if** permet de tester plusieurs conditions.

```
if (condition1) {
    // bloc de code si condition1 est vraie
} else if (condition2) {
    // bloc de code si condition2 est vraie
} else {
    // bloc de code si aucune des conditions précédentes n'est vraie
}
```

Exemple :

En reprenant le même exemple, si l'on souhaite afficher la catégorie d'une personne pour chaque tranche d'âge.

```
let age = 18

if (age == 18) {
    "Il est Majeur!"
} else if (age > 18) {
    "Il est Adulte!"
} else {
   "Il est mineur!"
}
```

Opérateur de comparaison

Lorsqu'on souhaite créer des conditions en JavaScript ou tester la valeur d'une variable, on utilise ce qu'on appelle les opérateurs de comparaison.

Exemple :

Parfois, on souhaite combiner des conditions pour obtenir un résultat spécifique, par exemple : autoriser uniquement les filles de 20 ans dans un restaurant. Cela requiert l'utilisation des opérateurs booléens.

```
let age = 18
var sexe = 'F'

if (age == 18 && sexe == 'F') {
    "Vous avez accès!"
} else if (age > 18) {
    "Accès refuse!"
}
```

```
a == b // a égale à b
a === b // a == b et a est de même "type" que b
a >= b // a supérieur OU égal à b
a > b   // a strictement supérieur à b
a <= b // a inférieur OU égal à b
a < b   // a strictement inférieur à b
```

Opérateur Ternaire

L'opérateur ternaire est une forme abrégée de l'instruction **if-else**.

```
// condition ? <instruction a executer si vrai> : <instruction a executer si faux>

sexe= 'M'
"Je suis " + (sexe == 'M'? "Homme" : "Femme")
```

Instruction { switch }

Le **switch case** permet d'exécuter une opération en fonction de la valeur de l'expression en paramètre. Il peut parfois remplacer une série de **if** et **else**. On y a recours lorsque de nombreux cas doivent être gérés.

L'instruction **switch** est donc utilisée pour effectuer différentes actions en fonction de différentes conditions.

```
switch (expression) {
    case valeur1:
        // Instruction à exécuter si le résultat de l'expression correspond à la valeur1
        break;
    case valeur2:
        // Instruction à exécuter si le résultat de l'expression correspond à la valeur2
        break;
    default:
        // Instruction par défaut si les valeurs ne correspondent à aucune des valeurs attendues
        break;
}
```

Exemple :

Pour un petit calendrier qui afficherait un mois si la valeur d'une case correspond à la valeur de la variable **"mois"** passée en paramètre.

```
let mois = 2

        switch (mois) {
            case 1:
                "Janvier"
                break
            case 2:
                "Fevrier"
                break
            default:
                "Ce Mois n'existe pas"
                break
        }
```

Conclusion : Les structures conditionnelles sont essentielles pour contrôler le flux d'exécution de votre programme en JavaScript. Elles permettent à votre code de réagir différemment selon les données et les situations, rendant vos applications plus dynamiques et interactives.

Quelques opérateurs booléens

```
// || OU

true || true // true
true || false // true
false || true // true
false || false // false
```

```
// && ET

true && true // true
true && false // false
false && true // false
false && false // false
```

Exercices sur les Structures Conditionnelles en JavaScript

Exercice 1: Vérification de l'Âge

Écrivez un programme JavaScript qui demande l'âge de l'utilisateur.

- Si l'utilisateur a 18 ans ou plus, affichez "Accès autorisé"
- sinon affichez "Accès refusé".

(L'utilisation de la fonction prompt() est nécessaire)

Exercice 2: Classification des Notes

Écrivez un programme qui classe une note en "Excellent", "Bien", "Suffisant", ou "Insuffisant" selon les critères suivants:

- 0-50 Insuffisant
- 51-70 Suffisant
- 71-90 Bien
- 91-100 Excellent.

(L'utilisation du if/else - else if est nécessaire)

Exercice 3: Jour de la Semaine

Écrivez un programme qui, en fonction d'un numéro (1-7), affiche le jour de la semaine correspondant (1 pour Lundi, 7 pour Dimanche).

Exercice 4: Calculatrice Simple

Créez une calculatrice simple qui prend deux nombres et une opération (addition, soustraction, multiplication, division) et affiche le résultat.

Exercice 5: Gestion d'un Menu de Restaurant

Vous développez une application pour un restaurant qui propose différents plats. Le menu change en fonction du jour de la semaine. De plus, le restaurant offre une réduction spéciale pour les étudiants. Votre tâche est de créer un programme qui, en fonction du jour de la semaine et du statut de l'utilisateur (étudiant ou non), affiche le plat du jour et le prix après réduction si applicable.

Instructions

- Utilisez une structure **switch** pour sélectionner le plat du jour en fonction du jour de la semaine.
- Utilisez une structure **if** pour appliquer une réduction de 20% sur le prix du plat si l'utilisateur est un étudiant.
- Affichez le plat du jour et le prix final.

Données :
Jours de la semaine: Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche.

Plats du jour:

Lundi: Pizza - 10€
Mardi: Pasta - 8€
Mercredi: Burger - 9€
Jeudi: Salade - 7€
Vendredi: Poisson - 12€
Samedi et Dimanche: Spécial Week-end - 15€

Réduction étudiante: 20%

Tâche pour l'Utilisateur :

Modifiez les variables **jour** et **estEtudiant** pour tester différents scénarios. Vérifiez si le programme affiche correctement le plat du jour et le prix après réduction pour les étudiants.

```
let jour = "Mercredi"; // Jour actuel
let estEtudiant = true; // Statut de l'utilisateur
let plat, prix;
```

Aller plus loin en utilisant la fonction **prompt()** pour le jour.

Les Boucles

Il existe de nombreux chemins pour effectuer une boucle mais certaines s'utilisent plus souvent que d'autres. Parmi les
instructions permettant d'effectuer des itérations, on retrouves :

- **for** parcourt un bloc de code plusieurs fois
- **for/in** parcourt les propriétés d'un objet
- **for/of** parcourt les valeurs d'un objet itérable
- **while** parcourt un bloc de code alors qu'une condition spécifiée est vraie
- **do/while** parcourt également un bloc de code lorsqu'une condition spécifiée est vraie

Les boucles (Structure répétitive) permettent de répéter plusieurs fois une ou plusieurs instructions en fonction d'une condition précise. Il existe deux types de boucles en général: la boucle **while** et la boucle **for**.

Boucle { while }

La boucle **while** exécute un bloc de code tant que la condition spécifiée est vraie et vérifiée, alors on rentre dans la boucle pour l'exécution de l'instruction, sinon on sort de la boucle.

```
while (condition) {
    // Code à exécuter tant que la condition est vraie
}
```

Elle est souvent utilisée lorsque le nombre d'itérations n'est pas connu à l'avance.

Exemple :

Pour éviter que cette boucle ne tourne indéfiniment, voici comment l'utiliser : on déclare une variable **"compteur"** (" i " dans l'exemple) pour initialiser la boucle et l'incrémenter à chaque itération*.

```
let i = 0

while (i < 3) {
    console.log("Je compte " + i);
    i = i + 1; // i++
}
```

***Itération** : Une itération est un tour complet à l'intérieur d'une boucle. Chaque fois que le code à l'intérieur de la boucle s'exécute, c'est une itération.*

On peut aussi écrire **i++**, ce qui augmente la valeur de i à chaque passage dans la boucle. Pour arrêter la boucle de force, on peut utiliser le mot clé "break".

```
let i = 0; // Initialisation de la variable i à 0

while (i < 10) { // Début de la boucle while avec la condition i < 10
    console.log("Je compte " + i); // Affichage du message dans la console

    if (i == 5) { // Vérification si i est égal à 5
        break; // Sortie de la boucle si i est égal à 5
    }

    i++; // Incrémentation de i à chaque itération
}
```

Boucle { for }

La boucle for est l'une des boucles les plus courantes. Elle est utilisée lorsque le nombre de répétitions est connu à l'avance.

```
for (initialisation; condition; incrémentation) {
    // Code à exécuter pour chaque itération
}
```

Initialisation : Cette partie est exécutée une seule fois, au début de la boucle. Elle initialise généralement une variable de contrôle utilisée pour suivre le nombre d'itérations. Par exemple,`let i = 0` ; initialise une variable ` i ` à 0.

Condition : Avant chaque itération, cette condition est évaluée. Si elle est vraie, le code à l'intérieur de la boucle est exécuté. Si elle est fausse, la boucle se termine. Par exemple, `i < 5;` vérifie si la variable ` i ` est inférieure à 5.

Incrémentation : À la fin de chaque itération de la boucle, cette partie est exécutée. Elle modifie généralement la variable de contrôle pour se rapprocher de la condition d'arrêt. Par exemple, ` i++; ` incrémente la variable ` i ` de 1 à chaque itération.

La boucle **for** permet d'exécuter le code un certain nombre de fois. On l'utilise lorsque le nombre de répétitions est connu à l'avance.

```
for (let i = 0; i < 3; i++) {
    console.log("Je compte " + i);
}
```

On utilise également la boucle for lorsqu'on travaille avec les tableaux. Elle est essentielle pour manipuler efficacement les tableaux.

```
let eleves = ['Gregory', 'Reine', 'Mohammed', 'Ashiri'];

for (let i = 0; i < eleves.length; i++) {
    console.log(eleves[i]); // Affiche chaque élément du tableau dans la console
}
```

Les Tableaux

Les tableaux en JavaScript sont des structures de données utilisées pour stocker des collections d'éléments. Ils sont flexibles et peuvent contenir des éléments de différents types, y compris d'autres tableaux. Les tableaux sont essentiels pour gérer des listes d'éléments dans le développement web.

Constructeur de Tableau :

```
let eleves = new Array("Karim", "Leopold", "Loutfy", "Wemby");
```

Il est également possible de créer un tableau de cette manière **“ Syntaxe Littérale “** :

```
let eleves = ['Gregory', 'Reine', 'Mohammed', 'Ashiri'];
```

Accès aux Éléments d'un Tableau

Les éléments d'un tableau sont accessibles via leur index. En JavaScript, les indices de tableau commencent à **0**.

```
let eleve1 = eleves[0]; // "Gregory"
let eleve2 = eleves[1]; // "Reine"
let eleve3 = eleves[2]; // "Mohammed"
```

Modification d'un Tableau

Les tableaux en JavaScript sont modifiables, ce qui signifie que vous pouvez changer, ajouter ou supprimer des éléments.

```
eleves.push("Reda"); // Ajoute un élément dans le tableau
eleves.pop("Liam"); // Supprime le dernier élément
```

- **push() :** Ajoute un ou plusieurs éléments à la fin du tableau.
- **pop() :** Supprime le dernier élément du tableau.

Parcourir un Tableau

Les tableaux en JavaScript disposent de plusieurs fonctions intégrées qui facilitent la manipulation et la gestion des données.
Ces fonctions permettent de réaliser des opérations courantes telles que la recherche, le tri, la filtration, la transformation, et bien plus. Voici quelques-unes des fonctions les plus utilisées liées aux tableaux en JavaScript :

- **forEach() :** Permet de répéter une action pour chacune des valeurs du tableau
- **sort() :** Trie les éléments d'un tableau et renvoie le tableau.
- **splice() :** Change le contenu d'un tableau en ajoutant, supprimant et/ou remplaçant des éléments.
- **unshift() :** Ajoute un ou plusieurs éléments au début du tableau.

Ces fonctions rendent les tableaux en JavaScript extrêmement puissants et polyvalents pour la manipulation de données. Comprendre et savoir utiliser ces fonctions est crucial pour tout développeur JavaScript.

Exercices sur les Structures répétitives

Exercice 1: Calcul de la moyenne

Créez un tableau de notes et utilisez une boucle **for** pour calculer la moyenne de ces notes.

Exercice 2: Génération de calendrier

Générez un calendrier pour un mois spécifique en utilisant une boucle **for**.

(L'utilisation du if/else - else if est nécessaire)

Exercice 3: Recherche dans un dictionnaire

Imaginez un dictionnaire de mots. Utilisez une boucle for pour chercher un mot spécifique dans le dictionnaire et affichez-le s'il est trouvé.

Exercice 4: Inversion d'un Tableau

Utilisez une boucle **for** pour inverser l'ordre des éléments d'un tableau.

Par exemple, inversez le tableau [1, 2, 3, 4, 5].

TD 1 : Système de Notation des Étudiants

Contexte :

Vous êtes chargé de développer un système de notation pour une classe d'étudiants. Chaque étudiant a une série de notes et vous devez calculer la moyenne de chaque étudiant. Ensuite, en fonction de cette moyenne, vous attribuerez une note finale selon une échelle spécifique.

Données :
Chaque étudiant a un ensemble de notes représentées dans un tableau de tableaux.

Par exemple : [[75, 80, 85], [90, 95, 100], [60, 65, 70]] où chaque sous-tableau représente les notes d'un étudiant.

L'échelle de notation est la suivante :

90 et plus : A | 80 - 89 : B | 70 - 79 : C | 60 - 69 : D | Moins de 60 : E

Objectif :
Écrivez un programme qui calcule la moyenne des notes pour chaque étudiant, puis attribuez une note finale basée sur l'échelle ci-dessus. Afficher la moyenne et la note finale pour chaque étudiant.

TD 2 : Système de Surveillance des Températures

Contexte :
Vous êtes chargé de développer un système de surveillance pour un réseau de capteurs de température. Ces capteurs enregistrent la température à différents endroits d'une installation industrielle. Votre tâche est de vérifier les lectures de température et d'identifier les situations où des températures anormales sont enregistrées.

Objectif :
Le système doit parcourir les lectures de température de chaque capteur et déterminer si elles sont dans une plage normale. Si une température est en dehors de cette plage, le système doit alerter l'utilisateur.

Données :
Les lectures de température sont stockées dans un tableau de tableaux. Chaque sous-tableau représente les lectures d'un capteur différent.
La plage normale de température est de 18°C à 25°C.

```
let lecturesTemperature = [
    [21, 23, 25, 22, 20, 19], // Capteur 1
    [26, 28, 22, 21, 20, 19], // Capteur 2
    [18, 17, 23, 22, 21, 20]  // Capteur 3
];
```

Tâche :

Écrivez un programme JavaScript qui utilise des boucles et des conditions pour vérifier chaque lecture de température. Si une température est en dehors de la plage normale, affichez un message d'alerte indiquant le numéro du capteur et la température anormale.

L'utilisateur du programme doit pouvoir :

- Modifiez le tableau **lecturesTemperature** pour tester le programme avec différentes lectures de température.
- Essayez d'ajouter un nouveau capteur avec ses propres lectures pour voir comment le programme gère les données supplémentaires.
- Ajustez la plage de température normale et observez les changements dans les alertes générées.

(L'utilisation de la propriété .length est nécessaire)

Les Fonctions

Une fonction est comme un « sous-programme » qui exécute des opérations répétitives. Plutôt que de réécrire tout le code à chaque fois, on crée une fonction qu'on peut appeler à tout moment. Cela permet non seulement d'alléger le code, mais aussi de le rendre plus lisible.

Définition d'une Fonction :

- Une fonction est définie avec le mot-clé function
- suivi d'un nom, d'une liste de paramètres entre parenthèses ()
- et d'un bloc de code entre accolades {}.

```
function nomDeLaFonction(parametre1, parametre2) {
    // Code à exécuter
}
```

Appel d'une Fonction

Pour exécuter le code d'une fonction, vous devez l'appeler par son nom suivi de parenthèses.

```
function saluer(nom) {
    console.log("Bonjour " + nom + " !");
}

saluer("Alice"); // Affiche "Bonjour Alice !"
```

Paramètres et Arguments

Les fonctions peuvent prendre des paramètres, qui agissent comme des variables au sein de la fonction. Lorsque vous appelez une fonction, vous fournissez des arguments, qui sont les valeurs réelles pour ces paramètres.

```
function additionner(a, b) {
    console.log(a + b);
}

additionner(5, 7); // Affiche 12
```

Fonctions avec Retour de Valeur

Les fonctions peuvent retourner des valeurs à l'aide du mot-clé return. Une fois qu'une fonction atteint une instruction return, elle arrête son exécution et renvoie la valeur spécifiée

```
function multiplier(a, b) {
    return a * b;
}

let resultat = multiplier(3, 4); // resultat vaut 12
```

Vous pouvez rendre un paramètre optionnel dans une fonction en lui donnant une valeur par défaut. Pour cela, il suffit d'ajouter une petite règle dans votre code.

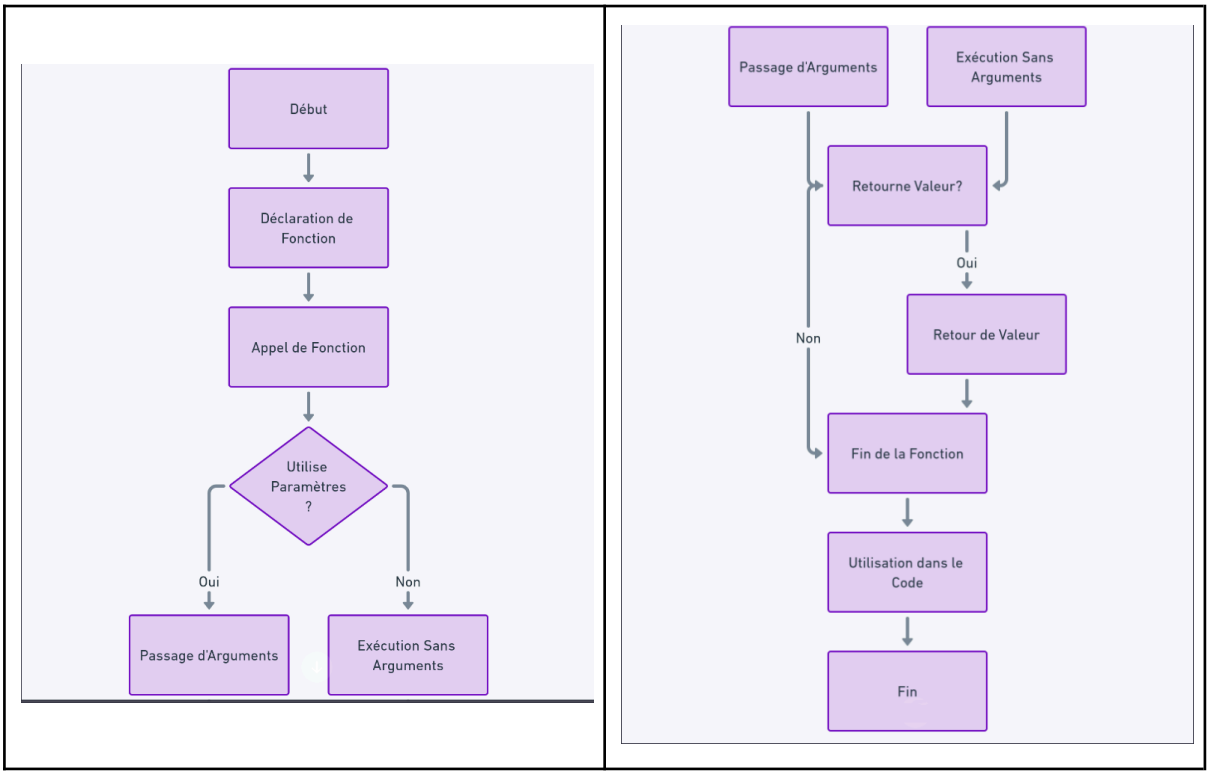
```
let multiplier = function (nombre){
    if (nombre === undefined) {
    nombre = 1
    }
    return nombre * 5
}

multiplier(5) //cela donne 25
multiplier() //1
```

Fonctions Fléchées (Arrow Functions)

Introduites dans ES6, les fonctions fléchées offrent une syntaxe plus concise pour écrire des expressions de fonction. ***(Utile pour les fonctions courtes)***

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

console.log(additionner(2, 3)); // Affiche 5
```

Flux de contrôle d'une fonction



Portée des Variables dans les Fonctions

Les variables définies à l'intérieur d'une fonction ne sont pas accessibles de l'extérieur de celle-ci. Elles sont dites "locales" à la fonction.

```
function exemplePortee() {
    let variableLocale = "Je suis locale";
    console.log(variableLocale);
}

exemplePortee(); // Affiche "Je suis locale"
console.log(variableLocale); // Erreur : variableLocale n'est pas définie
```

Les Méthodes (Fonction en JavaScript)

```
let objet = {
    propriete: valeur,
    methode: function() {
      // actions
    }
  };
```

En JavaScript, une méthode est une fonction associée à un objet. Elle est définie comme une propriété d'un objet et peut être appelée pour effectuer une action liée à cet objet.

Les Objets en JavaScript

En JavaScript, un objet est une collection de propriétés, et une propriété est une association entre un nom (ou clé) et une valeur. Parfois, cette valeur peut être une fonction. Si c'est le cas, on appelle cette propriété une "méthode" de l'objet.

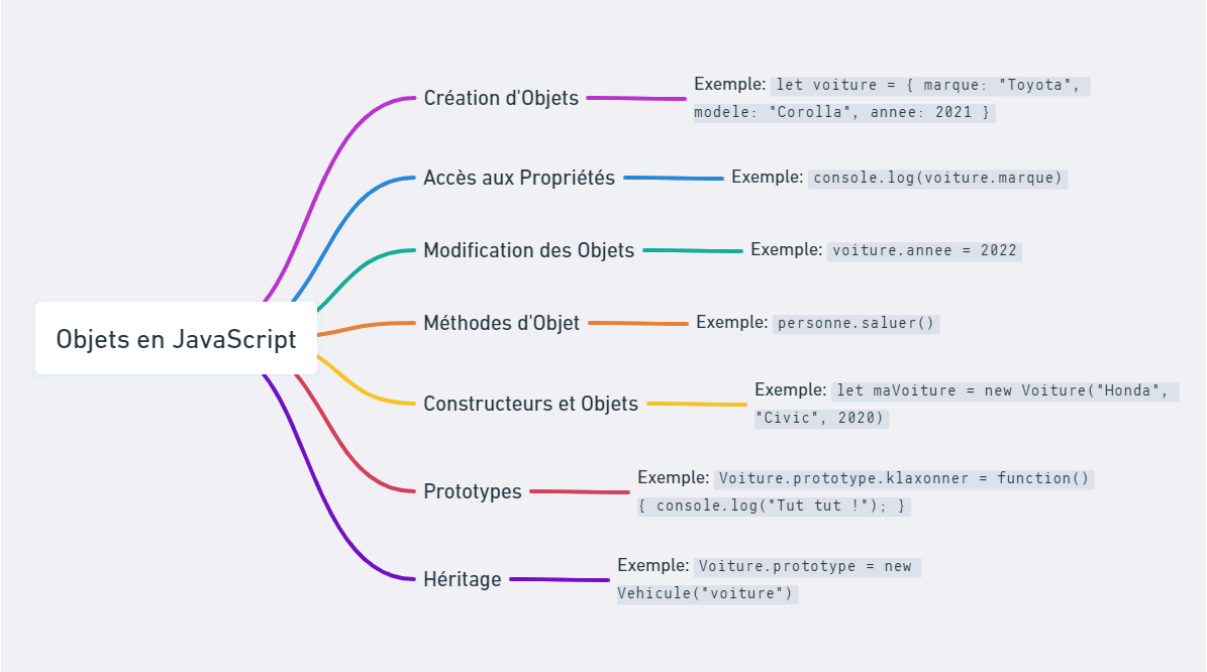
Dans l'exemple suivant, `**voiture**` est un objet avec trois propriétés: `**marque**`,`**modele**` et `**annee**`**.** On peut accéder aux propriétés via le point `**.**` ou les crochets `**[ ]**`**.**

```
let voiture = {
  marque: "Toyota",
  modele: "Corolla",
  annee: 2021
};

  console.log(voiture.marque); // Affiche "Toyota"
  console.log(voiture["modele"]); // Affiche "Corolla"

  voiture.annee = 2022; // Modifie l'année
  voiture.couleur = "rouge"; // Ajoute une couleur "rouge"
```

Il est également possible de modifier ou d'ajouter des propriétés, comme ci-dessus.

- Modification de la propriété existante `**annee**`.
- Ajout d'une nouvelle propriété `**couleur**`.

Voici un diagramme illustrant la structure et le fonctionnement des objets en JavaScript :



Ce diagramme montre les différents aspects des objets en JavaScript, y compris la création d'objets, l'accès et la modification de leurs propriétés, l'utilisation de méthodes, la création d'objets via des constructeurs, l'utilisation de prototypes pour ajouter des fonctionnalités, et l'héritage pour étendre les objets.

Les Méthodes (Cas d'Utilisation)

Une fonction utilisée sur un objet est appelée `**méthode**`. Les méthodes sont employées sur les types d'objets intégrés à JavaScript, comme les chaînes de caractères ou les nombres.

Dans le cadre de la validation des données d'un formulaire d'inscription, on peut :

- Utiliser une méthode appelée `**valider**` dans un objet `**utilisateur**` pour vérifier les données saisies.
- Puis on appelle la méthode `**valider**` lorsque l'utilisateur soumet le formulaire.

```
let utilisateur = {
    nom: "",
    email: "",
    valider: function() {
      if (this.nom !== "" && this.email.includes("@")) {
        console.log("Validation réussie");
      } else {
        console.log("Erreur de validation");
      }
    }
  };

  // Supposons que ceci est appelé lors de la soumission du formulaire
utilisateur.valider();
```

`this` dans les Méthodes

Le mot-clé **`this`** dans une méthode fait référence à l'objet auquel la méthode appartient.

```
let personne = {
    nom: "Alice",
    saluer: function() {
      console.log("Bonjour, je suis " + this.nom); // `this` fait référence à `personne`
    }
  };
```

Avec ES6, vous pouvez utiliser des fonctions fléchées pour définir des méthodes, mais attention au comportement de **`this`** qui ne sera pas lié à l'objet.

```
let personne = {
  nom: "Alice",
  saluer: () => {
    console.log("Bonjour, je suis " + this.nom); // `this` ne fait pas référence à `personne` ici
  }
};
```

Exercices sur les Fonctions

TD 1 : Calculatrice Simple

Tâche :

- Créer 4 fonctions : addition, soustraction, multiplication et division.
- Chaque fonction doit prendre deux nombres en paramètre et retourner le résultat de l'opération.
- Assurez-vous que les fonctions retournent **false** si une erreur survient, par exemple, si une chaîne de caractères est fournie en entrée.

Données supplémentaires :

- Assurez-vous que les fonctions vérifient le type des entrées pour éviter des erreurs d'exécution. **(L'utilisation de typeof est nécessaire)**
- La division par zéro est également prise en compte pour éviter une autre source d'erreur.

Objectif :

- Appelez chacune des fonctions avec différentes valeurs en entrée pour vérifier leur bon fonctionnement.

Introduction à JavaScript

TD 2 : Bibliothèque

Tâche :

1. Initialiser une Bibliothèque

2. Créer 3 fonctions

    - **ajouterLivre (titre, description)**: Ajoute un nouveau livre à la liste.
    - Créer un objet **nouveauLivre{}** à l'intérieur contenant id, titre, descritption.

    (L'utilisation de la fonction Date est nécessaire pour l'id)

    - **supprimerLivre(id)**: Supprime un livre avec l'identifiant spécifié de la bibliothèque

    (L'utilisation de la fonction Filter est nécessaire pour l'id)

    - **afficherBibliothèque()**: Affiche la liste des livres.

    (L'utilisation de la fonction ForEach est nécessaire)

Objectif :

- Affichez la liste des livres dans la bibliothèque
- Pouvoir ajouter un livre à la liste
- Pouvoir supprimer un livre de la liste