

**JS**

**Apprendre  
à coder  
avec JS**

# **Module 1**

**Stocker  
différents types  
de données**

# Introduction

Javascript est un langage de programmation qui va nous permettre de rendre dynamique nos pages Web.

Il existe de nombreux langages de programmation: PHP, .Net, Java, Python, C++, C, C#... Bien souvent orientés serveur (BACK) mais pour ce qui est côté utilisateur (FRONT), le JS est à priori le langage à privilégier.

Il a été inventé tout particulièrement pour le WEB donc se plie parfaitement à tout ce dont on peut avoir besoin sur la partie FRONT d'un projet.

# Types de données

En programmation, toutes les données ont un type. Ici on parlera de **primitives** (des types de données principales en JS) :

- **Nombre** (num) : Utilisé pour représenter des valeurs numériques, comme 42, 3.14, ou -7.
- **Chaîne de caractère** (string) : Utilisée pour représenter du texte, comme "Bonjour" ou 'JavaScript'.
- **Boolean** (bool) : Représente une valeur vraie (true) ou fausse (false), souvent utilisée pour prendre des décisions dans le code.
- **Undefined** : Indique une valeur qui n'a pas été définie.
- **Null** : Indique l'absence intentionnelle de valeur.

# Les Variables

Une variable en programmation c'est simplement un conteneur qui stocke une ou plusieurs valeurs de n'importe quel type.

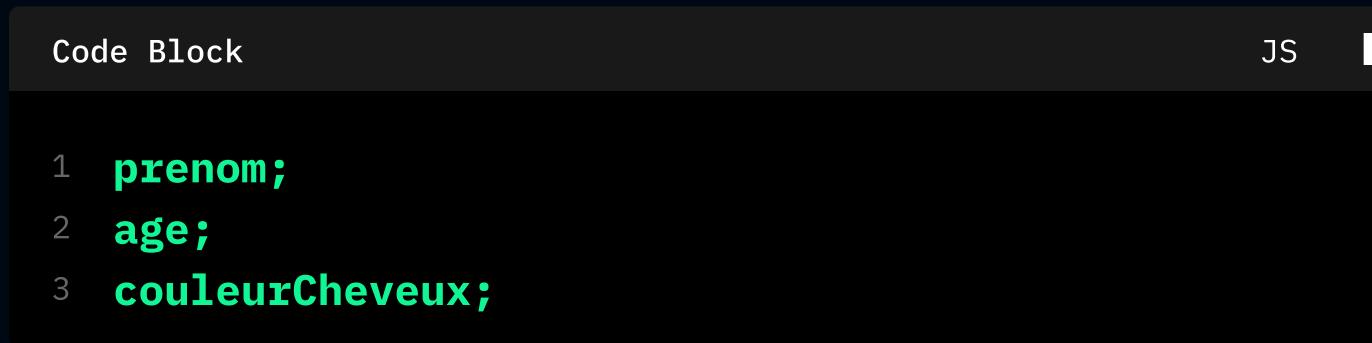
On déclare toujours une variable avant de pouvoir l'utiliser.

Pour ça on va utiliser le mot clé **let** et je n'oublie JAMAIS **le point virgule!!** (Comme en CSS)



```
Code Block JS JS
1 let prenom = "Jeanette";
2 let age = 95;
3 let couleurCheveux = "rose";
```

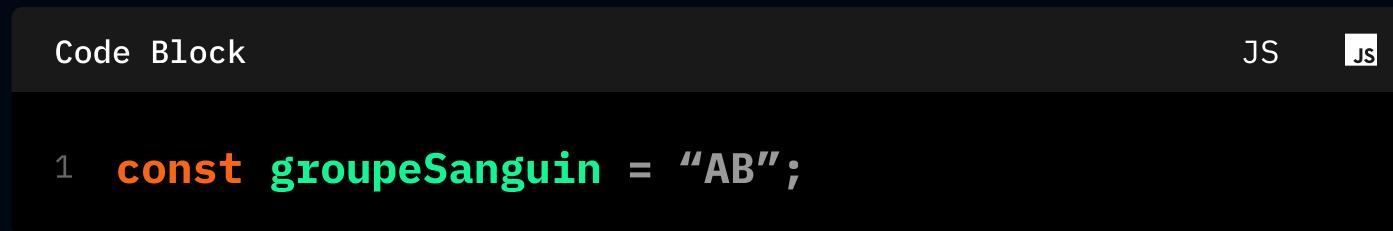
Une fois ma variable déclarée, je peux l'appeler n'importe où dans mon code pour l'utiliser en enlevant simplement le mot-clé **let**.



```
Code Block JS JS
1 prenom;
2 age;
3 couleurCheveux;
```

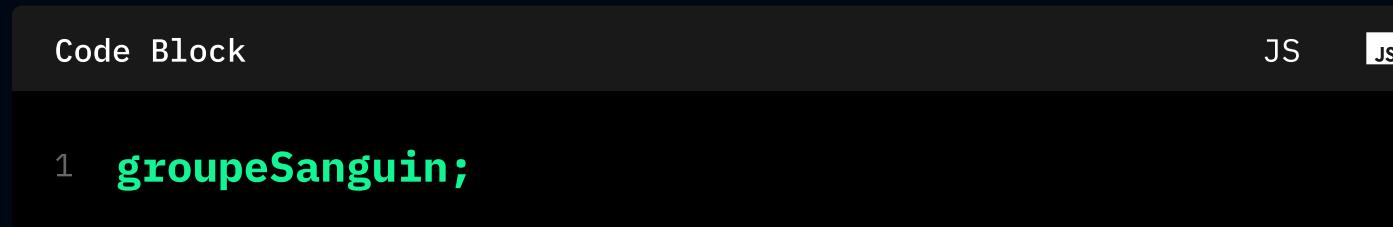
# Les Constantes

Pour déclarer une valeur immuable (qui ne changera jamais), on n'utilisera pas une variable mais une constante. La seule différence dans la déclaration de celle-ci est le mot-clé utilisé: **const**.



A screenshot of a code editor interface. At the top, it says "Code Block" and has tabs for "JS" and "JS". Below the tabs, there is a single line of code: "1 const groupeSanguin = "AB";". The "const" keyword is highlighted in orange, and the variable name "groupeSanguin" is highlighted in green.

Comme pour la variable si je veux faire appel à ma constante pour utiliser/lire la valeur stockée, j'enlève simplement le mot-clé.



A screenshot of a code editor interface, identical to the one above, but with the "const" keyword removed from the first line of code. Now it only shows "1 groupeSanguin;" where the "const" keyword is no longer present.

# Exercice

Créer un fichier Index.html, un fichier script.js et relier les deux en ajoutant une **balise script** dans le **body** de l'Index.html

Dans le fichier script.js, on voudra stocker plusieurs valeurs:

- Le nom de l'animal (**petName**)
- L'espèce de l'animal (**species**)
- L'âge de l'animal (**age**)
- Si l'animal est stérilisé ou non (**neutered**).

Indiquer si chacune de ces valeurs sera stockée dans une **variable** ou une **constante** et les déclarer dans le fichier.

# La Console

## Où la trouver?

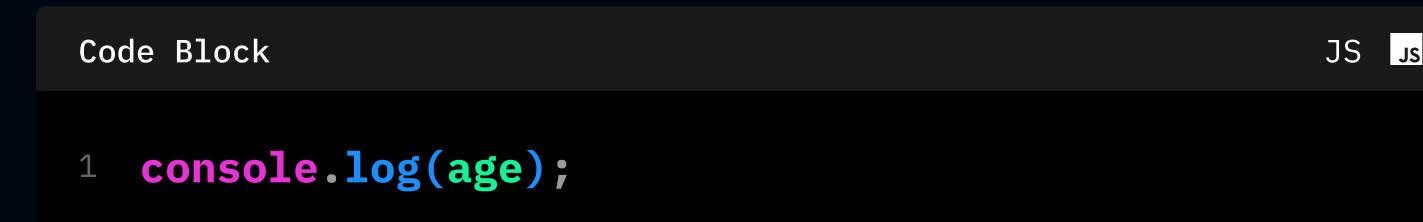
- Dans la plupart des navigateurs web, ouvrez l'inspecteur (clic droit > Inspecter) et allez dans l'onglet **Console**.

## À quoi ça sert?

- Permet d'afficher des messages, des erreurs et de déboguer du code.

## L'utilisation

- `console.log()` va nous permettre d'afficher des choses dans la console (valeurs de variables, types...)  
**Ex:** `console.log(age)`; affichera la valeur de la variable `age`.
- On peut exécuter tout type de ligne de code dans la console (même sans `console.log()`).



A screenshot of a browser's developer tools console. The title bar says "Code Block". Below it, there are two tabs: "JS" and "JS" (repeated). The main area contains a single line of code: "1 console.log(age);".

# Exercice

Reprenons l'exercice précédent:

1. Dans le script.js, ajouter un **console.log** pour afficher le nom de l'animal
2. Lancer l'Index.html en live server.
3. Ouvrir la console du Navigateur.
4. Afficher la valeur de la variable **age**.
5. Changer la valeur de la variable **age**.
6. Vérifier que la modification s'est effectuée

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Crée une variable contenant le nom de la boutique.
2. Crée une variable contenant la ville où se situe la boutique.
3. Crée une variable booléenne indiquant si la boutique est ouverte ou fermée.
4. Crée une variable contenant le nombre de produits disponibles.
5. Crée une variable contenant un slogan (une phrase courte).
6. Affiche dans la console :
  - un message contenant le nom de la boutique et la ville
  - le slogan
  - un message indiquant que la boutique est ouverte

**Dépot Github**

# **Module 2**

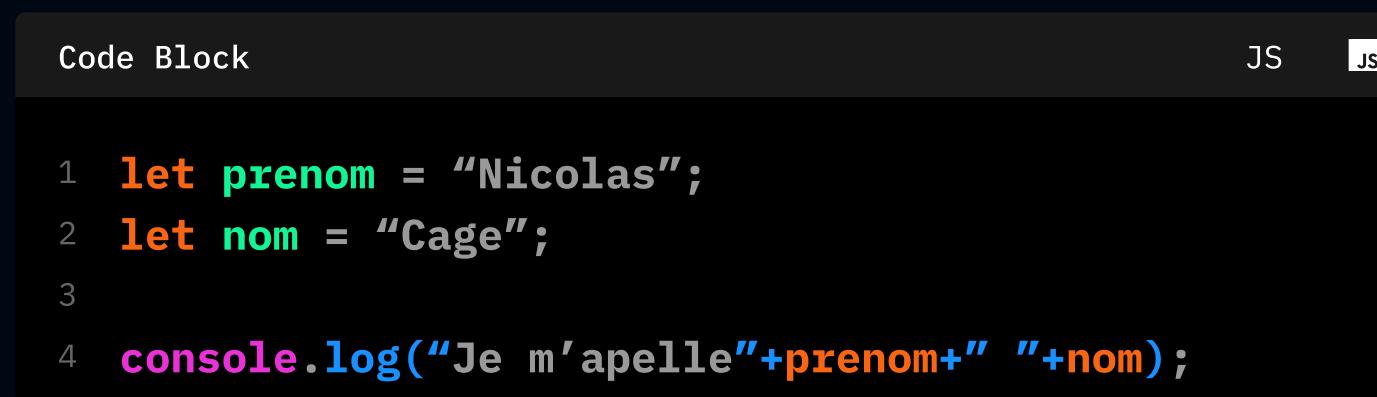
**Traiter les  
données  
textuelles**

# Les chaînes de caractères 1/3

## La Concaténation

On peut combiner plusieurs chaînes de caractères en une seule grâce à l'opérateur +

Ex:



A screenshot of a code editor window titled "Code Block". The tab bar at the top shows "JS" twice. The code area contains the following JavaScript code:

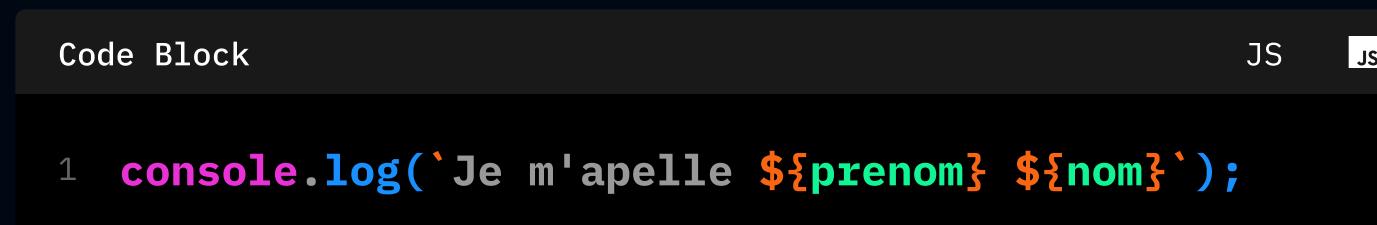
```
1 let prenom = "Nicolas";
2 let nom = "Cage";
3
4 console.log("Je m'appelle"+prenom+" "+nom);
```

Attention si on n'ajoute pas d'espace entre mes variables, on aura une phrase qui ressemble à ça : “Je m'appelle JohnDoe”.

## L'Interpolation ou template string

L'intérêt ici est similaire à la concaténation mais l'écriture va différer

Ex:



A screenshot of a code editor window titled "Code Block". The tab bar at the top shows "JS" twice. The code area contains the following JavaScript code:

```
1 console.log(`Je m'appelle ${prenom} ${nom}`);
```

# Exercice

1. Crée une variable **dishName** avec le nom d'un plat.
2. Crée deux variables **recipeMessage1** et **recipeMessage2** qui indiqueront “Le plat du jour est ...” :
  - une fois avec concaténation
  - une fois avec template string (ou interpolation)
3. Affiche les deux dans la console.

# Les chaînes de caractères 2/3

## La Longueur

La propriété **.length** est utilisée pour savoir de combien de caractère est composée une chaîne.

```
Code Block JS JS

1 let phrase = "Je suis une tartine";
2
3 console.log(phrase.length);
```

## Conversion de casse

On peut passer une phrase de minuscule à majuscule ou inversement avec **toUpperCase()** et **toLowerCase()**

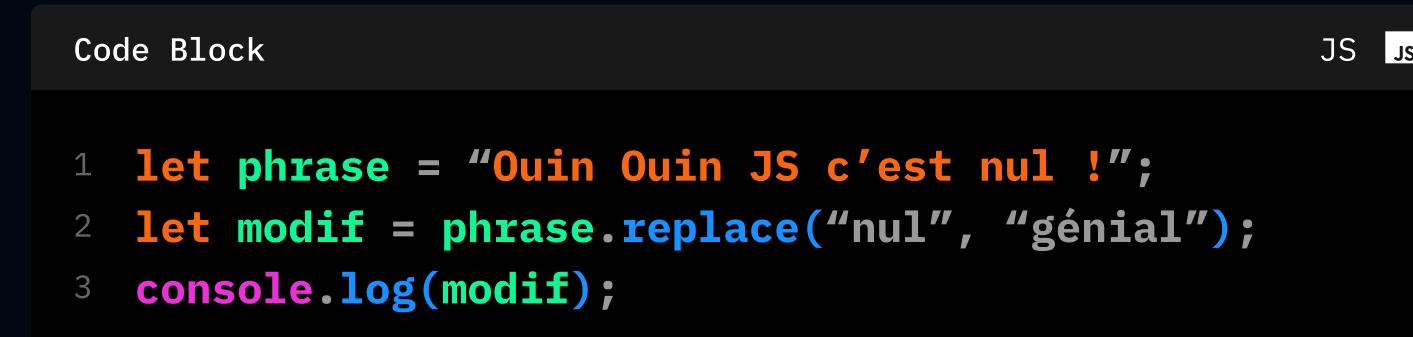
```
Code Block JS JS

1 let nom = "Dudon";
2 let nomMaj = nom.toUpperCase();
3 console.log(nomMaj);
```

# Les chaînes de caractères 3/3

## Remplacer du texte

On peut utiliser **replace()** pour remplacer des parties d'une chaîne de caractère par une autre.



A screenshot of a code editor window titled "Code Block". The tab bar at the top shows "JS" and "js". The code block contains the following JavaScript code:

```
1 let phrase = "Ouin Ouin JS c'est nul !";
2 let modif = phrase.replace("nul", "génial");
3 console.log(modif);
```

Il existe encore plein de propriétés et méthodes qui permettront d'utiliser et modifier les données de type string....

# Exercice

1. Crée une variable **animalDescription** (ex : "Petit chat tigré très joueur").
2. Affiche dans la console:
  - la longueur de la description
  - la version en majuscules
  - une version où tu remplaces "joueur" par "peureux"

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Crée une variable `welcomeMessage` contenant un message construit avec une concaténation.
2. Crée une variable `welcomeMessage2` contenant le même message, mais cette fois avec une template string.
3. Stocke la longueur du slogan dans une variable.
4. Crée une variable contenant une version en majuscules du slogan.
5. Modifie le slogan en changeant un mot grâce à `.replace()`.
6. Affiche dans la console :
  - `welcomeMessage`
  - `welcomeMessage2`
  - la longueur du slogan
  - la version en majuscules
  - la version modifiée du slogan

[Dépot Github](#)

# **Module 3**

**Traiter les  
données  
numériques**

# Les nombres 1/2

## Opérations mathématiques simples

On va se servir des opérateurs **+ - / \*** qui ont les même fonctions qu'en mathématiques

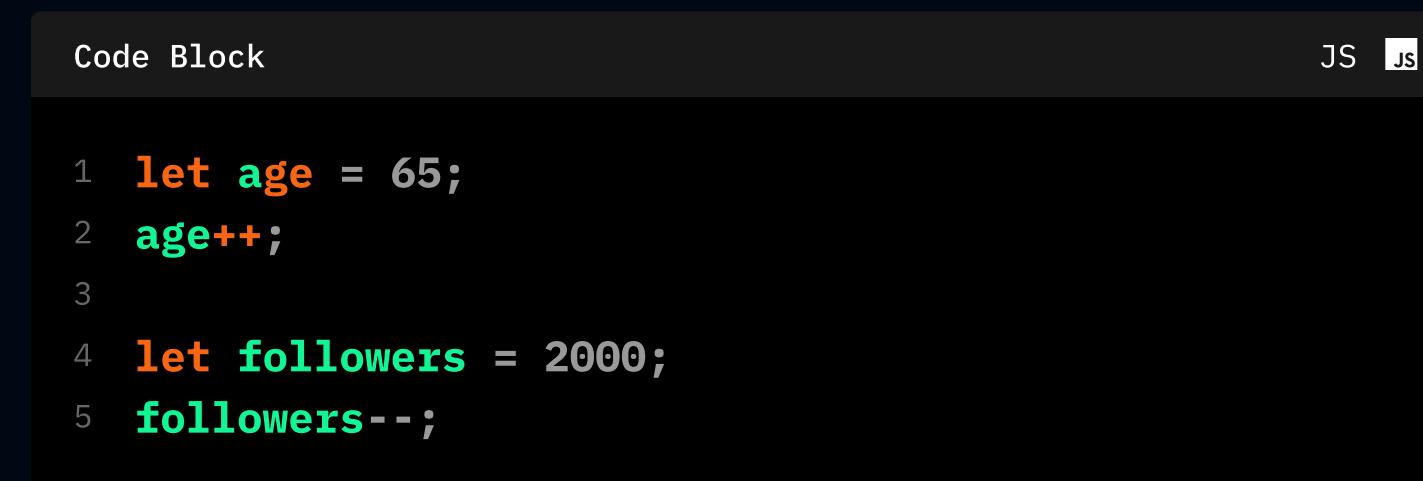
Code Block JS js

```
1 let nombre1 = 40;
2 let nombre2 = 2;
3
4 let addition = nomnbre1 + nombre2;
5 let soustraction = 66 - 35;
6 let multiplication = nombre1 * nombre2;
7 let division = 100 / 4;
```

# Les nombres 2/2

## Incrémation et décrémation

Si on souhaite augmenter (**incrémenter**) ou diminuer (**décrémenter**) une valeur numérique de 1, on va utiliser les opérateurs suivants: **++** ou **--**



The image shows a screenshot of a code editor window titled "Code Block". At the top right, there are tabs for "JS" and "JS". The code block contains the following five lines of JavaScript:

```
1 let age = 65;
2 age++;
3
4 let followers = 2000;
5 followers--;
```

Il existe encore plein de propriétés et méthodes qui permettront d'utiliser et modifier les données de type number....

# Exercice

1. Créer une variable **foodPerDay** qui contiendra la portion en gramme de croquettes pour un chien par jour.
2. Créer une autre variable **foodPerWeek** qui servira à calculer combien je dois acheter de grammes de croquettes pour 1 semaine.
3. Afficher le résultat dans la console avec un message.

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Crée une variable contenant un prix HT d'exemple (un nombre).
2. Crée une variable TVA contenant le taux de TVA (ex : 0.2).
3. Calcule le prix TTC et stocke-le dans une nouvelle variable.
4. Affiche dans la console :
  - le prix HT
  - le prix TTC
5. Crée une variable salesCount initialisée à 0, puis :
  - incrémente-la à l'aide de ++
  - affiche sa nouvelle valeur dans la console

[Dépot Github](#)

# **Module 4**

**Début de  
l'automatisation  
des projets**

# Les fonctions 1/3

Un peu comme en CSS avec les classes, les fonctions sont simplement des bout de code réutilisables.

## Déclaration

On commence toujours par déclarer une fonction grâce au mot clé **function** puis du **nom de la fonction** (c'est nous qui décidons), de **parenthèses** puis d'**accolades** (pour écrire nos lignes de code à l'intérieur).

A screenshot of a dark-themed code editor window titled "Code Block". The tab bar at the top shows "JS" twice. The code area contains three lines of JavaScript:

```
1 function bonjour() {  
2     console.log("Bonjour");  
3 }
```

## Appel

Une fois notre fonction déclarée, peut l'appeler n'importe où dans notre code (pour l'utiliser). Comme avec les variables, on va juste écrire le **nom de la fonction**(sans oublier les **parenthèses** ni le point virgule)

A screenshot of a dark-themed code editor window titled "Code Block". The tab bar at the top shows "JS" twice. The code area contains one line of JavaScript:

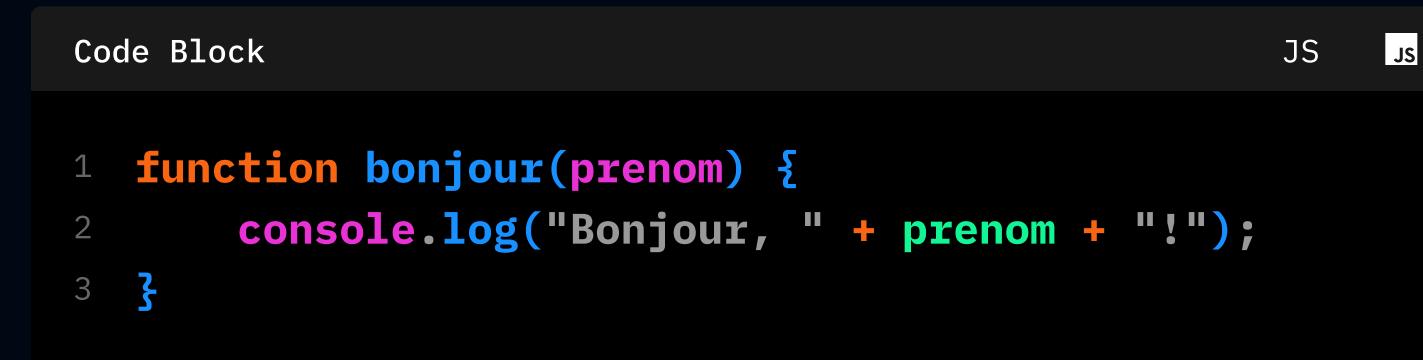
```
1 bonjour();
```

# Les fonctions 2/3

## Paramètres

### Au moment de la déclaration

Entre les parenthèses d'une fonction que l'on **déclare** se trouvent les paramètres. Il s'agit d'une valeur attendue qui va nous servir dans le code de la fonction.



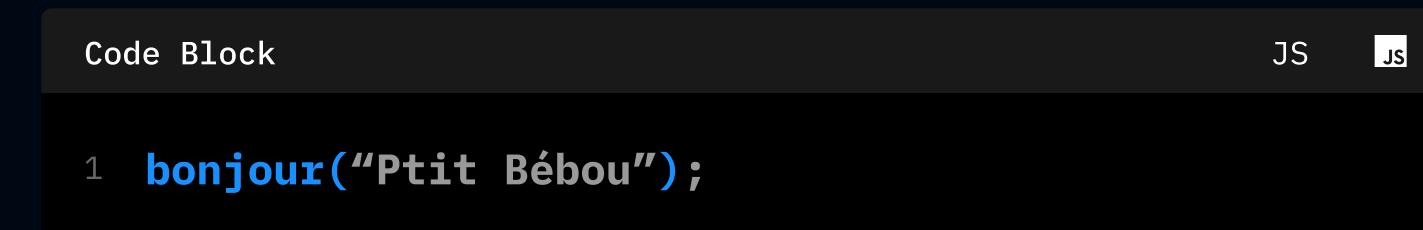
A screenshot of a code editor window titled "Code Block". The tab bar at the top shows "JS" and "JS". The code block contains the following JavaScript code:

```
function bonjour(prenom) {  
    console.log("Bonjour, " + prenom + "!");  
}
```

## Arguments

### Au moment de l'Appel

Entre les parenthèses d'une fonction que l'on **appelle** se trouvent les arguments. Il s'agit de la valeur réelle donnée pour le paramètre. Elle s'appliquera partout où le paramètre a été inscrit dans la fonction.



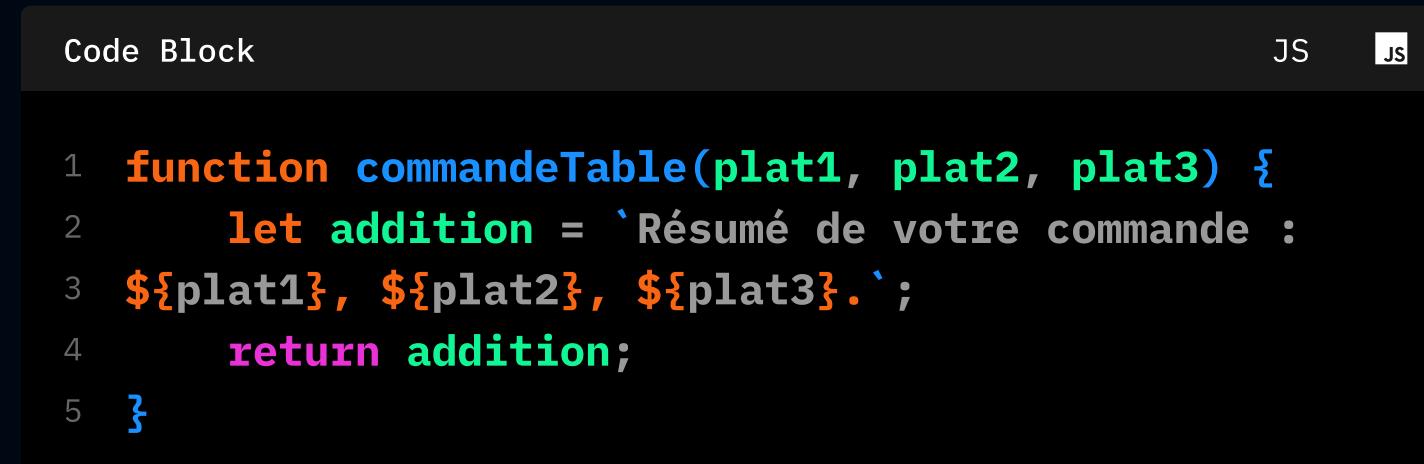
A screenshot of a code editor window titled "Code Block". The tab bar at the top shows "JS" and "JS". The code block contains the following JavaScript code:

```
bonjour("Ptit Bébou");
```

# Les fonctions 3/3

## Return

Si on veut **récupérer une valeur en sortie de fonction** pour pouvoir la réutiliser par la suite (et pas juste l'afficher), on doit utiliser dans la fonction le mot-clé **return**. Grâce à lui, on pourra stocker le résultat de notre fonction dans une variable pour le réutiliser.



```
Code Block JS
1 function commandeTable(plat1, plat2, plat3) {
2     let addition = `Résumé de votre commande :
3 ${plat1}, ${plat2}, ${plat3}.`;
4     return addition;
5 }
```

**Attention!! Return termine automatiquement la fonction! Si du code suit le mot-clé return il ne sera pas exécuté!**

# Exercice

1. Crée une fonction nommée selfCare qui prend 3 paramètres :

- mentalHealth → la santé mentale actuelle (un nombre)
- bonus → le nombre de points gagnés grâce à l'activité (un nombre)
- activity → le nom de l'activité réalisée (une chaîne de caractères)

2. Dans la fonction :

- Affiche un message dans la console, par exemple :  
"Tu fais l'activité : méditation"
- La fonction doit retourner la nouvelle valeur de santé mentale  
(mentalHealth + bonus)

3. Dans ton script :

- Crée une variable (ex : let mh = 50;)
- Appelle ta fonction au moins deux fois avec des activités différentes
- Affiche la santé mentale à la fin

# Exercice

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Crée une fonction `calculatePriceTTC(priceHT)` qui :
  - reçoit un prix HT en paramètre
  - calcule le prix TTC
  - renvoie ce prix TTC
2. Crée une fonction `formatPrice(price)` qui :
  - reçoit un nombre
  - arrondit la valeur à 2 décimales
  - renvoie une chaîne de caractères du type : "12,99 €"
3. Teste les deux fonctions en affichant le résultat formaté dans la console pour plusieurs prix HT de ton choix.

**Dépot Github**

# **Module 5**

## **Manipulation du DOM**

# Le DOM

## Définition

Le **DOM** (Document Object Model) est un **objet JS** qui **représente la page HTML**. Chaque partie de la page est noeud qui peut être lu, modifié, supprimé...

## Sélectionner un élément dans la page

### **getElémentById()**

On doit trouver un élément précis donc... on cherche par son **id** avec **getElemetById("id")**

Code Block      JS

```
1 const title = document.getElementById("main-
2 title");
3
```

### **querySelector()**

On doit modifier plusieurs élément donc... on cherche par la sélecteur CSS (nom d'attribut HTML, classe...) avec **querySelector("selecteur")**

Code Block      JS

```
1 const info = document.querySelector(".info");
2
```

# Le DOM

## Modifier un élément

Une fois un élément sélectionné, on peut :

### changer son texte

Code Block JS

```
1 element.textContent = "Nouveau texte";  
2  
3
```

### changer son HTML

Code Block JS

```
1 element.innerHTML = "<strong>Hello</strong>";  
2  
3
```

### changer son style

Code Block JS

```
1 element.style.color = "red";  
2  
3
```

# Le DOM

## Créer un nouvel élément

On peut aussi créer du HTML directement depuis le JS:

### Créer l'élément

Code Block      JS

```
1 const title = document.createElement("h3");  
2
```

### Renseigner son contenu

Code Block      JS

```
1 title.textContent = "T-shirt Code & Chill";  
2
```

### Ajouter l'élément à la page

Code Block      JS

```
1 const card = document.querySelector(".card");  
2  
3 card.appendChild(title);  
4
```

# Exercice

1. Créer une section avec l'**id cardContainer** dans le HTML.
2. Sélectionner la section depuis le JS.
3. Créer trois variables **nom**, **prenom** et **age** avec des valeurs attribuées
4. Créer depuis le JS:
  - a. un h2 qui contiendra le **nom** et **prénom**
  - b. un p qui contiendra l'**âge**
5. Attribuer les éléments tout juste créés à la section **CardContainer**.

# Exercice

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Crée quatre variables représentant le produit vedette :
  - `featuredProductName` : le nom du produit
  - `featuredProductPriceHT` : le prix HT
  - `featuredProductDescription` : la description du produit
  - `featuredProductImage` : l'URL de son image
2. Récupère dans le DOM l'élément qui accueillera le produit.

[Dépot Github](#)

# Exercice

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

3. Crée une fonction `createFeaturedProductCard()` qui devra :

- créer un élément `<article>`
- créer un `<img>` et définir sa `src` et son `alt`
- créer un `<h3>` contenant le nom du produit
- calculer le prix TTC du produit à partir du prix HT
- formater ce prix TTC avec `formatPrice()`
- créer un `<p>` contenant le prix TTC
- créer un `<p>` contenant la description du produit
- ajouter tous ces éléments dans l'article
- retourner l'article

4. Affiche ton produit vedette dans la page :

- vide le contenu de `#product-list`
- appelle la fonction `createFeaturedProductCard()`
- ajoute la carte renournée dans `productList` avec `appendChild()`

**Dépot Github**

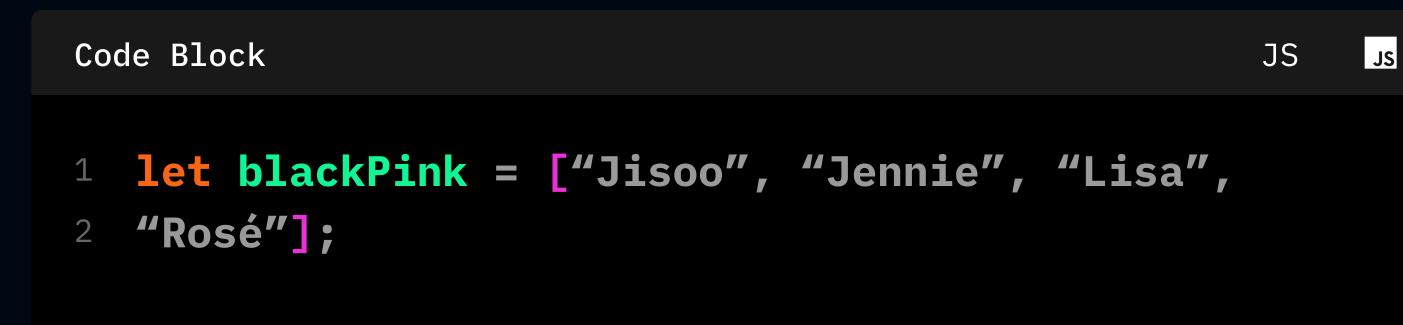
# **Module 6**

## **Itération et stockage de multiples données**

# Les tableaux 1/2

## Déclaration

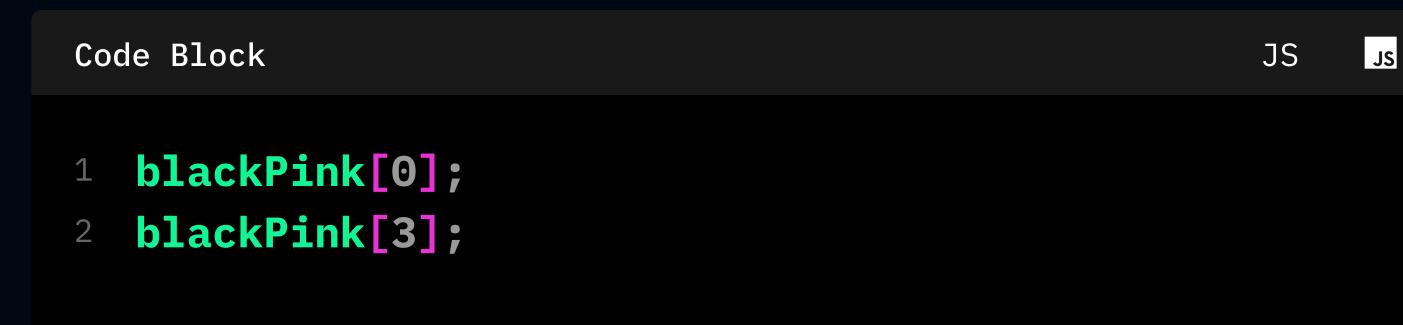
Les tableaux ou **Array** vont servir à stocker plusieurs données à la fois. Pour déclarer un tableau, il suffit d'écrire nos valeurs **entre crochets []**



```
Code Block JS
1 let blackPink = ["Jisoo", "Jennie", "Lisa",
2 "Rosé"];
```

## Index

Pour accéder aux valeurs stockées dans un tableau, on va utiliser un chiffre qui correspond à la **position** de la valeur dans le tableau, **l'Index**. Il commence toujours à **0** et augmente au fur et à mesure qu'on ajoute des éléments au tableau et on l'utilise entre les crochets pour indiquer de chercher dans le tableau la position indiquée.



```
Code Block JS
1 blackPink[0];
2 blackPink[3];
```

# Les tableaux 2/2

## Modifier

Pour modifier un élément d'un tableau, on utilisera son index et y appliquera une nouvelle valeur

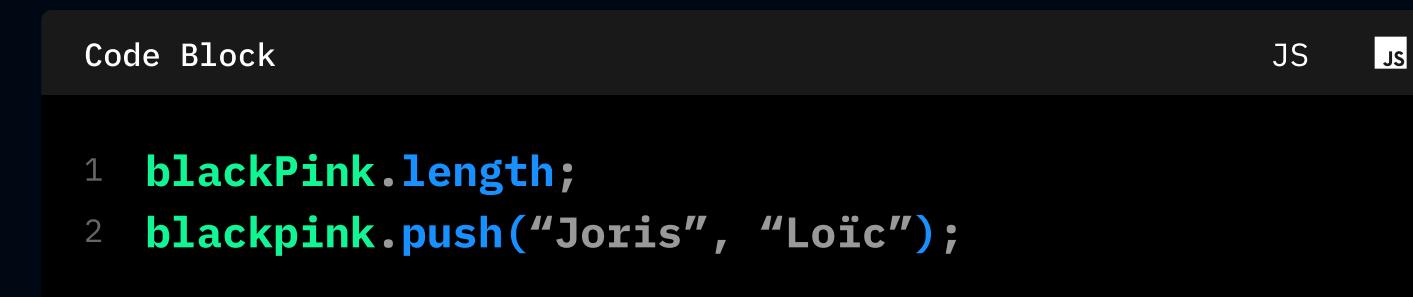


```
Code Block JS JS  
1 blackPink[0] = "Jessica";
```

## Propriétés des tableaux

Il existe des propriétés très utiles pour gérer les tableaux. On les ajoute après le nom du tableau sans oublier les parenthèses.

- **.length** : Obtenir le nombre d'éléments dans un tableau.
- **.push()** : Ajouter un élément à la fin d'un tableau.
- **.pop()** : Supprimer le dernier élément d'un tableau.
- **.indexOf()** : Trouver l'index d'un élément spécifique dans un tableau.



```
Code Block JS JS  
1 blackPink.length;  
2 blackpink.push("Joris", "Loïc");
```

# Exercice

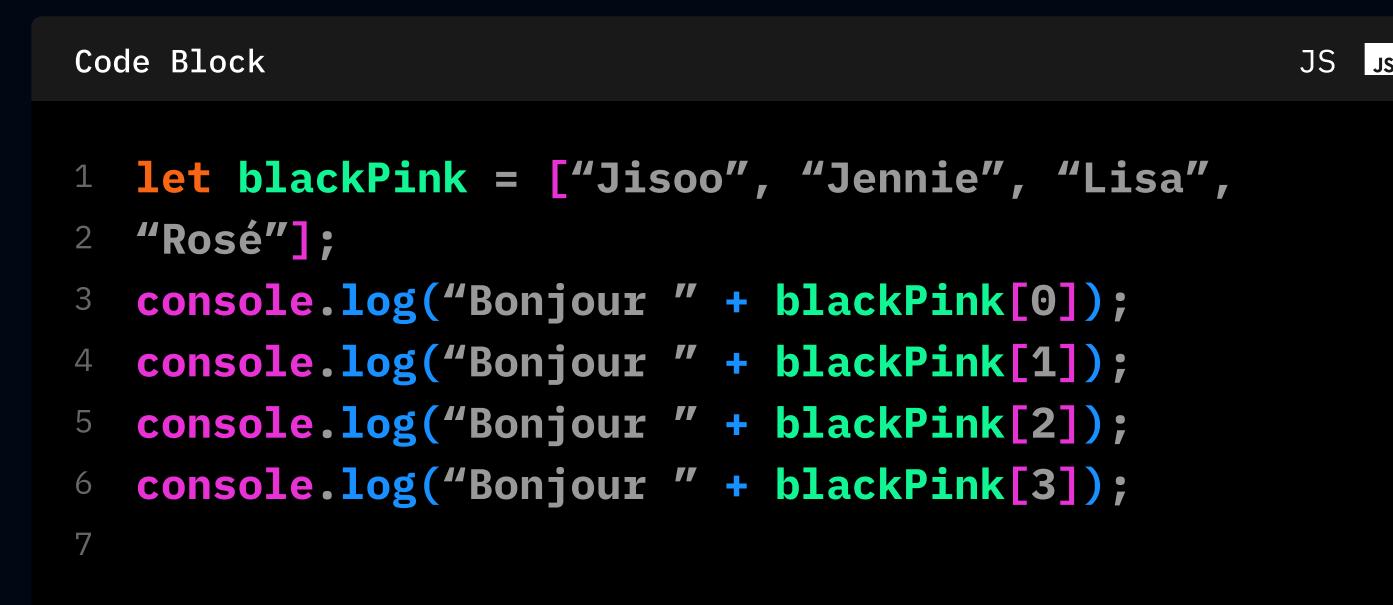
1. Créer un tableau dishes qui contiendra 3 plats.
2. Calculer la longueur du tableau.
3. Modifier le deuxième élément.
4. Supprimer le dernier élément.
5. Recalculer la longueur du tableau.

# Les Itérations 1/2

Une itération c'est juste une répétition en algorithmie.

## Tableau

Récupérer toutes les valeurs d'un tableau pour le moment peut sembler répétitif



```
Code Block JS JS

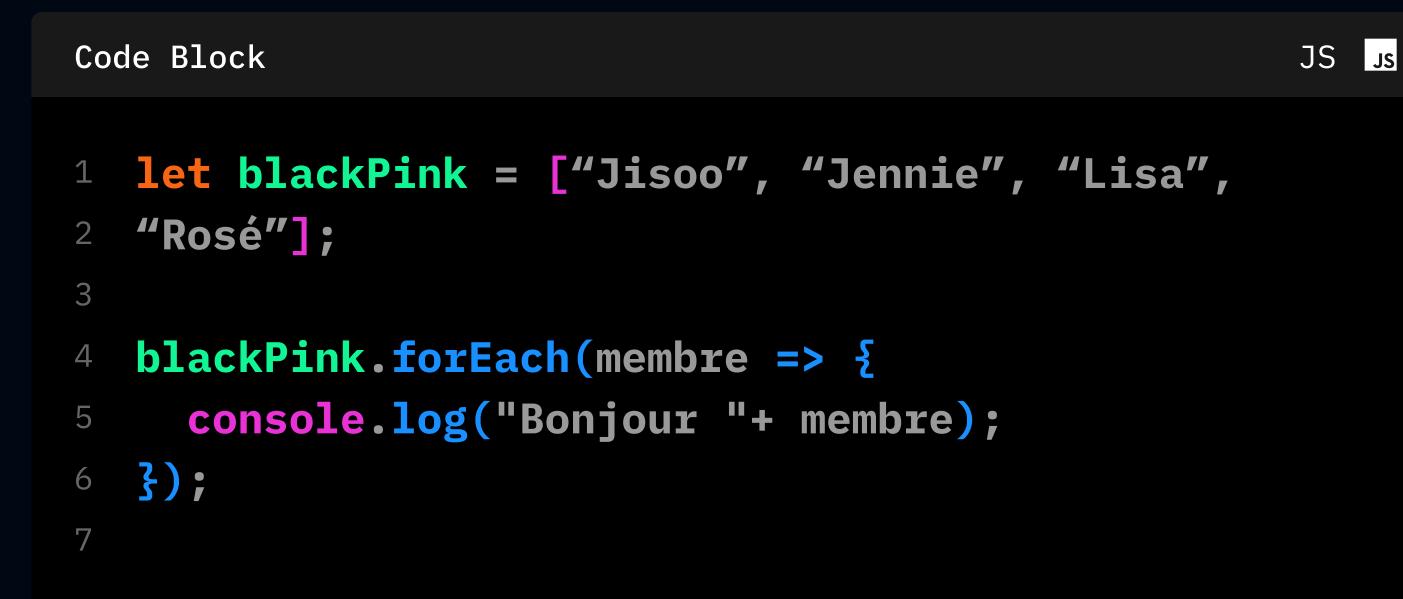
1 let blackPink = ["Jisoo", "Jennie", "Lisa",
2 "Rosé"];
3 console.log("Bonjour " + blackPink[0]);
4 console.log("Bonjour " + blackPink[1]);
5 console.log("Bonjour " + blackPink[2]);
6 console.log("Bonjour " + blackPink[3]);
7
```

On cherche toujours un moyen d'automatiser au maximum en algorithmie donc pour l'itération, on ne peut pas se permettre de tout faire manuellement et c'est là qu'entrent en jeu les Boucles!

# Les Itérations 2/3

## Boucle ForEach

La méthode **ForEach()** s'applique exclusivement aux tableaux. Elle prendra en argument une fonction à appliquer. Cette fonction aura un **paramètre** qui équivaudra à **chaque valeur de notre tableau.**



The image shows a screenshot of a code editor window titled "Code Block". At the top right, there are tabs for "JS" and "JS". The code block contains the following JavaScript code:

```
1 let blackPink = ["Jisoo", "Jennie", "Lisa",
2 "Rosé"];
3
4 blackPink.forEach(membre => {
5   console.log("Bonjour " + membre);
6 });
7
```

# Les Itérations 3/3

## Boucle For

La boucle **For()** permet également de répéter du code plusieurs fois. Elle prend en paramètres la déclaration d'un index, la condition pour que la boucle continue et l'incrémentation de l'index.

Code Block JS

```
1 for(let i = 0; i < 5; i++) {  
2     console.log("Tentative de mot de passe: " + (i  
3 + 1));  
4 }
```

# Exercice

1. Créer un tableau games qui contiendra jeux.
2. Utiliser une boucle adaptée pour afficher dans la console un message disant : “Le jeu suivant a été ajouté à votre bibliothèque Steam: ...”

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Crée un tableau `productNames` contenant les noms de plusieurs produits (au moins 3).
2. Crée un tableau `productPricesHT` contenant leurs prix HT, dans le même ordre.
3. Affiche dans la console la longueur du tableau `productNames`.
4. Crée une fonction `displayProductsInConsole()` qui :
  - parcourt les deux tableaux avec une boucle `for`
  - récupère le nom du produit avec `productNames[i]`
  - récupère le prix HT avec `productPricesHT[i]`
  - calcule le prix TTC grâce à `calculatePriceTTC()`
  - formate le prix TTC grâce à `formatPrice()`
  - affiche une ligne du type :  
"Produit 1 : Mug JS – 14,99 € TTC"
5. Appelle la fonction.

**Dépot Github**

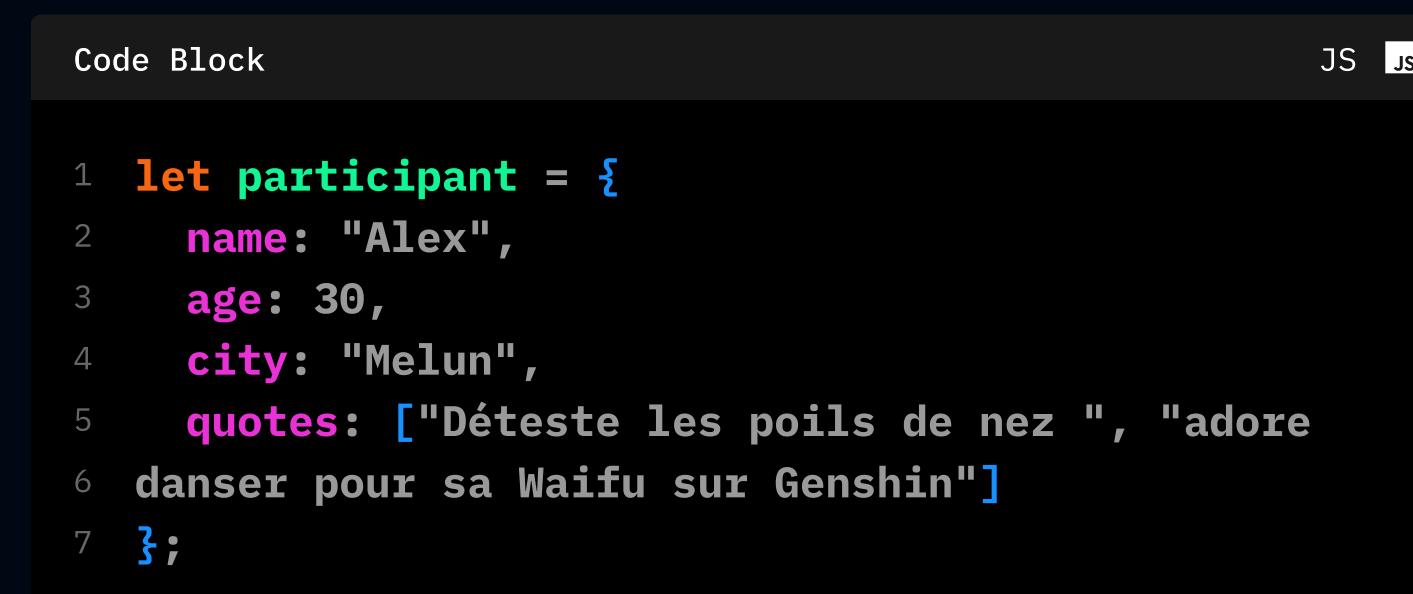
# **Module 7**

## **Objets et Tableaux d'objets**

# Les Objets

## Déclaration

Un objet est un **type de donnée** qui va regrouper plusieurs informations qui concernent la même chose avec un système de **propriété/valeur**.

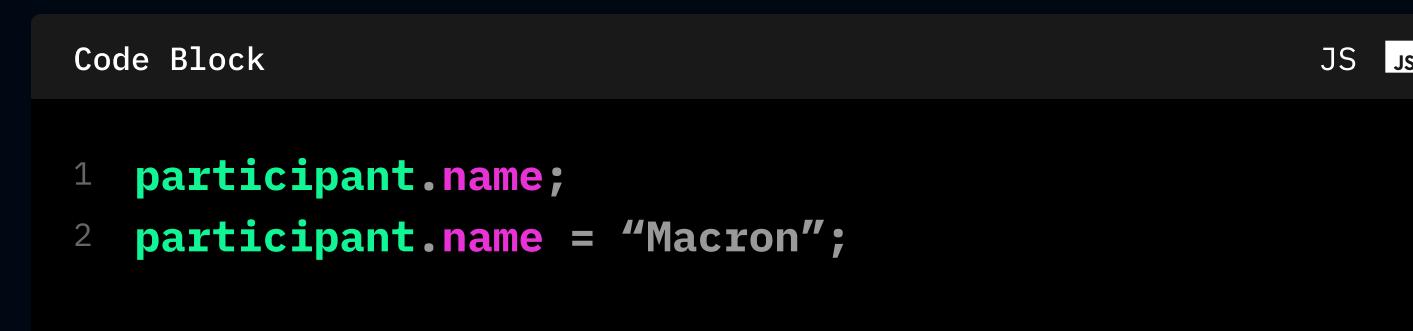


```
Code Block JS JS

1 let participant = {
2   name: "Alex",
3   age: 30,
4   city: "Melun",
5   quotes: ["Déteste les poils de nez ", "adore
6 danser pour sa Waifu sur Genshin"]
7 };
```

## Accéder à une propriété

Pour utiliser ou modifier les valeurs d'une propriété, je dois l'appeler.



```
Code Block JS JS

1 participant.name;
2 participant.name = "Macron";
```

# Exercice

1. Crée un objet recipe avec :
  - a. Un titre (title)
  - b. Portions (servings)
  - c. Vegetarien ou non (isVegetarian)
2. Affiche une phrase dans la console : "Titre : ... - Pour ... personnes".

# Les Tableaux d'objets

## Stocker dans des tableaux

Dans de nombreux cas (quasi toujours), on a besoin de **stocker** nos objets quelque part. On utilisera un **tableau** pour ça.

Code Block JS

```
1 const recipes = [
2   {
3     title: "Curry de légumes",
4     servings: 4,
5     vegetarian: true
6   },
7   {
8     title: "Pâtes carbonara",
9     servings: 2,
10    vegetarian: false
11  }
12];
```

## Parcourir un tableau d'objets

Pour récupérer, modifier, afficher les infos d'objets se trouvant dans des tableaux, on va devoir **boucler** avec un **for... of**

Code Block JS

```
1 for (let recipe of recipes) {
2   console.log(recipe.title);
3 }
```

# Exercice

1. Créer un tableau de recettes (en gardant le modèle d'objet de l'exercice précédent)
2. Boucler sur le tableau et:
  - a. Pour chaque recette, afficher dans la console son titre.
  - b. Pour chaque recette, créer un paragraphe avec comme contenu le titre de la recette en modifiant le DOM

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Crée un tableau products contenant plusieurs objets.

Chaque objet doit avoir au minimum :

- id
- name
- priceHT
- description
- image (URL)

2. Récupère dans le DOM l'élément <section id="product-list">.

3. Assure-toi que ta fonction createProductCard(product) fonctionne correctement avec les objets du tableau.

4. Crée une fonction displayProductsInPage() qui :

- vide le conteneur
- parcourt le tableau products
- crée une carte produit pour chaque élément
- ajoute chaque carte dans la page

5. Appelle displayProductsInPage() pour afficher tous les produits.

**Dépot Github**

# **Module 8**

## **Conditions et DOM dynamique**

# Les opérateurs de comparaison 1/3

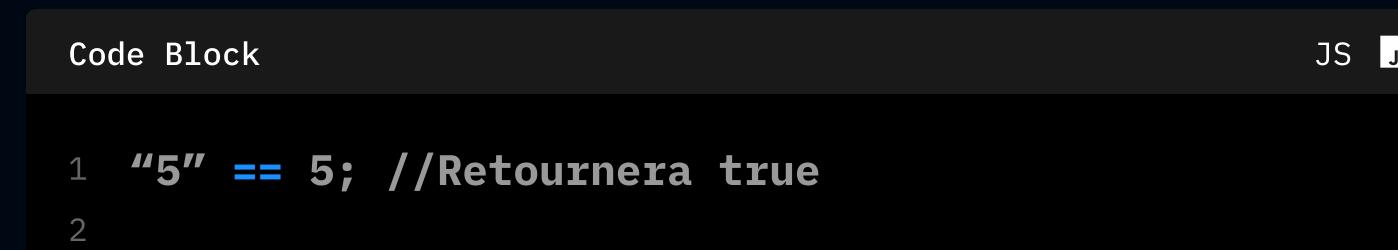
Un opérateur de comparaison est un signe qui nous permet de comparer deux valeurs.  
La comparaison retournera soit true soit false.

## Egalités et Inégalités

### Egalité

Pour vérifier si 2 valeurs sont égales on utilise l'opérateur **==**

Attention!! Il ne prend pas en compte le type de donnée (type number ou string)

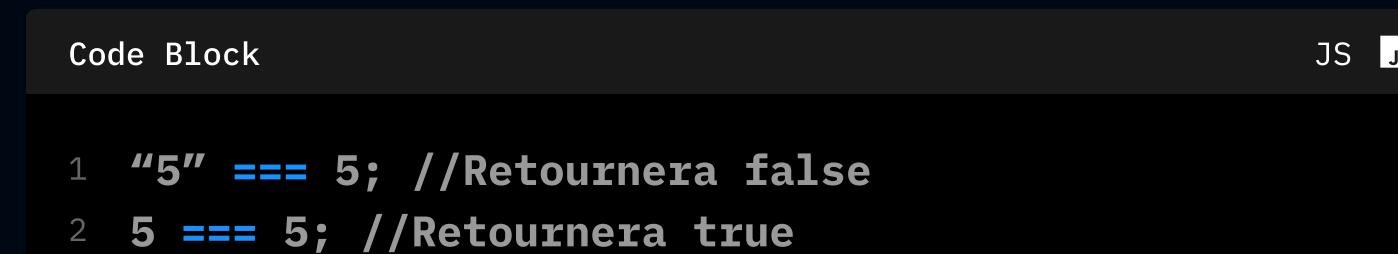


```
Code Block JS JS
1 "5" == 5; //Retournera true
2
```

### Egalité stricte

Pour vérifier si 2 valeurs sont strictement égales on utilise l'opérateur **====**

Attention!! Avec **====** on prend en compte le type de donnée.



```
Code Block JS JS
1 "5" === 5; //Retournera false
2 5 === 5; //Retournera true
```

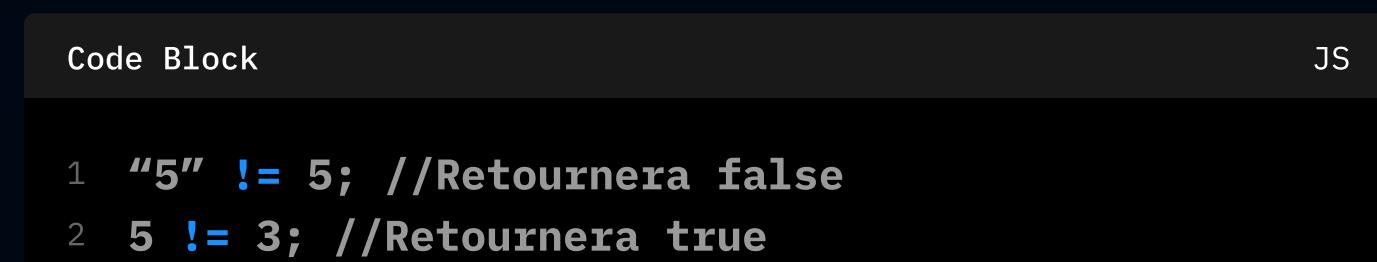
# Les opérateurs de comparaison 2/3

## Egalités et Inégalités

### Inégalité

Pour vérifier si 2 valeurs sont inégales on utilise l'opérateur **!=**

Attention!! Il ne prend pas en compte le type de donnée.



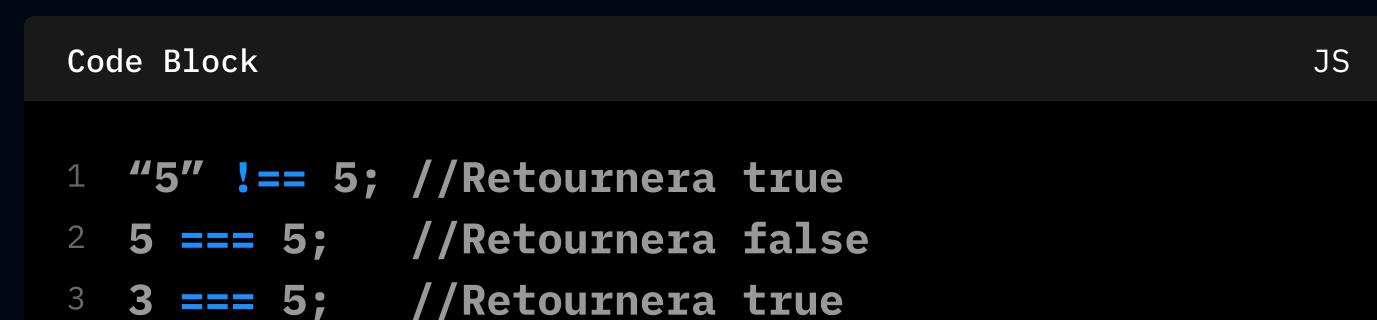
Code Block JS

```
1 "5" != 5; //Retournera false
2 5 != 3; //Retournera true
```

### Inégalité stricte

Pour vérifier si 2 valeurs sont strictement inégales on utilise l'opérateur **!==**

Attention!! Avec **!==** on prend en compte le type de donnée.



Code Block JS

```
1 "5" !== 5; //Retournera true
2 5 === 5; //Retournera false
3 3 === 5; //Retournera true
```

# Les opérateurs de comparaison 3/3

## Infériorité ou supériorité

### Infériorité

Avec l'opérateur **<** je vérifie si la valeur de gauche est **inférieure** à celle de droite.

Avec l'opérateur **<=** je vérifie si la valeur de gauche est **inférieure ou égale** à celle de droite.

Code Block JS

```
1 5 < 6; //Retournera true
2 5 <= 5; //Retournera true
```

### Supériorité

Avec l'opérateur **>** je vérifie si la valeur de gauche est **supérieure** à celle de droite.

Avec l'opérateur **>=** je vérifie si la valeur de gauche est **supérieure ou égale** à celle de droite.

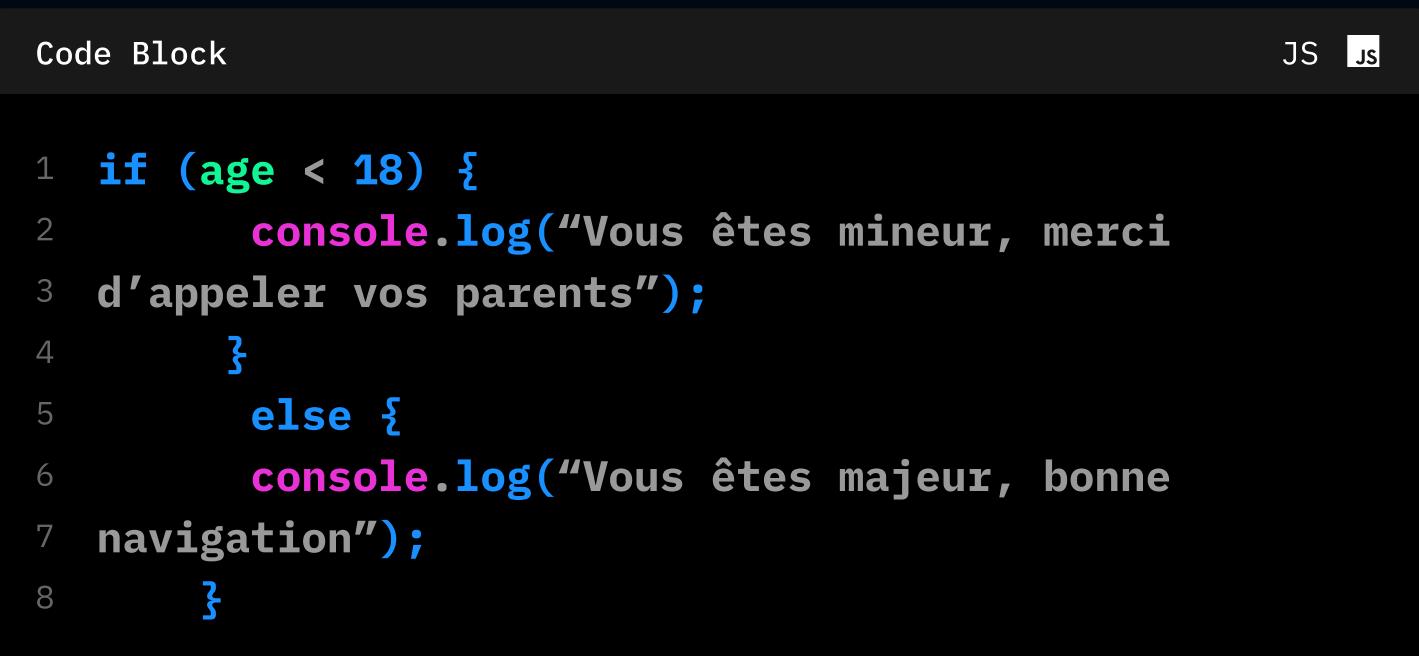
Code Block JS

```
1 5 > 3; //Retournera true
2 5 >= 5; //Retournera true
```

# Les conditions

Pour écrire une condition on doit préciser les mots-clés **if** (suivi de **parenthèses et en paramètres les valeurs à vérifier**) et **else** chacun suivit d'accolades qui contiendront le code à exécuter.

Par exemple, je veux un message différent si la personne connectée est majeure ou mineure. Il me faudra préciser **SI** l'âge de l'utilisateur est inférieur à 18 le message indiquera "Vous êtes mineur, merci d'appeler vos parents", **SINON** il indiquera "Vous êtes majeur, bonne navigation".



The image shows a screenshot of a code editor window titled "Code Block". The tab bar at the top indicates "JS" is selected. The code block contains the following JavaScript code:

```
1 if (age < 18) {  
2     console.log("Vous êtes mineur, merci  
3 d'appeler vos parents");  
4 }  
5 else {  
6     console.log("Vous êtes majeur, bonne  
7 navigation");  
8 }
```

# Exercice

1. Créer un paragraphe dans l'index.html avec l'**id** “status”.
2. Créer une variable **score** avec une valeur à 45.
3. Créer une variable **message** vide.
4. **Si** le score est inférieur à 50, le message prendra en valeur : “Continue a t’entraîner”  
**sinon** le message prendra en valeur : “Bravo”
5. Récupérer le paragraphe par son **identifiant** et lui appliquer le message.

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Crée deux variables globales pour représenter l'état du panier :
  - cartItemCount (nombre d'articles dans le panier)
  - cartTotal (total du panier en TTC)
2. Crée une fonction generateCartMessage(total) qui renvoie :
  - "Votre panier est vide." si total vaut 0
  - un message d'incitation si total est strictement inférieur à 50
  - un message "Livraison offerte 🎉" (ou équivalent) si total est supérieur ou égal à 50
3. Récupère les éléments liés au panier dans le DOM, par exemple :
  - cart-count → affichage du nombre d'articles
  - cart-total → affichage du total formaté
  - cart-message → affichage du message généré

**Dépot Github**

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

4. Crée une fonction `updateCartDisplay()` qui :
  - met à jour l'affichage du nombre d'articles
  - met à jour l'affichage du total (tu peux utiliser `formatPrice(cartTotal)`)
  - met à jour le message du panier avec `generateCartMessage(cartTotal)`
5. Modifie manuellement dans ton code les valeurs de `cartItemCount` et `cartTotal` pour tester :
  - un panier vide
  - un panier avec `total < 50`
  - un panier avec `total ≥ 50`

À chaque fois, appelle `updateCartDisplay()` et observe le rendu dans la page.

[Dépot Github](#)

# **Module 9**

## **Evènements et interactions**

# Les évènements

## Quand?

Un **événement** est quelque chose qui se produit quand:

- l'utilisateur clique
- il passe la souris
- il appuie sur une touche
- un élément est chargé
- un formulaire est soumis

## AddEventListener

En JS, on gère les événements avec la méthode AddEventListener. Elle peut écouter plusieurs types d'évenements. On doit lui indiquer que faire en cas d'actiovation.

Code Block

JS

```
1 element.addEventListener("click", function() {
2   // code exécuté quand l'événement se produit
3});
```

# Exercice

1. Créer un paragraphe dans l'index.html avec l'**id** “eggCount” et un bouton avec l'**id** “add”.
2. Créer une variable **eggs** avec une valeur à 0.
3. Au clic sur le bouton add:
  - a. Incrémenter la variable **eggs** de 1.
  - b. Insérer un contenu textuel au paragraphe **eggCount** : “Oeufs : ...”.

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

1. Vérifie que tu disposes bien :
  - du tableau products (tableau d'objets produit)
  - des fonctions calculatePriceTTC(priceHT) et formatPrice(price)
  - des variables globales cartItemCount et cartTotal
  - de la fonction generateCartMessage(total)
  - de la fonction updateCartDisplay() qui met à jour le DOM du panier
2. Modifie la fonction createProductCard(product) pour :
  - créer un bouton (par ex. <button>Ajouter au panier</button>)
  - ajouter une classe CSS si besoin (ex: "btn-add")
  - ajouter un écouteur d'événement :
    - sur le clic ("click")
    - qui appelle la fonction addToCart(product)

**Dépot Github**

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

4. Crée une fonction `addToCart(product)` qui :
  - reçoit un objet `product` en paramètre
  - calcule le prix TTC du produit avec `calculatePriceTTC(product.priceHT)`
  - ajoute ce montant à `cartTotal`
  - incrémente `cartItemCount` de 1
  - appelle `updateCartDisplay()` pour mettre à jour le DOM (compteur, total, message)
5. Vérifie que ta fonction `displayProductsInPage()` :
  - vide bien le conteneur de produits
  - parcourt le tableau `products`
  - crée une carte pour chaque produit (avec le bouton inclus)
  - ajoute toutes les cartes dans la page

[Dépot Github](#)

# Projet Fil Rouge

**Vous Venez D'être Recruté Par Une Petite Marque Qui Souhaite Ouvrir Sa Boutique En Ligne.**  
**Pour L'instant, Il N'existe Rien : Pas De Pages, Pas D'interface, Pas De Panier.**  
**Votre Mission Est De Construire, Étape Après Étape, La Toute Première Version Du Site.**

6. Au chargement de la page :

- appelle `displayProductsInPage()` pour afficher les produits
- appelle `updateCartDisplay()` une première fois pour initialiser l'affichage du panier (0 article, total 0, message de base)

7. Teste le comportement :

- clique plusieurs fois sur “Ajouter au panier” pour différents produits
- observe comment évoluent :
  - le nombre d'articles
  - le total
  - le message du panier

[Dépot Github](#)

**JS**

**Apprendre  
à coder  
avec JS**