

# JavaScript

Dynamiser un site internet

*Côté client*

→ Qu'est-ce que c'est ?

→ Pourquoi ?

→ Comment l'utiliser ?

→ Les *variables*

→ Les *commentaires*

→ Les *fonctions*

→ Les *conditions*

→ Les *tableaux*

→ Les *objets*

→ Les *boucles*

# Javascript – Les bases

---

# Qu'est-ce que c'est ?

---

JavaScript est un langage de programmation qui permet d'implémenter des mécanismes complexes sur une page web. C'est la **troisième couche** des technologies standards du web.



# Pourquoi l'utiliser ?

---

Si votre objectif est de rendre vos pages interactives alors vous n'avez pas trop le choix, le JavaScript est un passage obligé car c'est le seul langage de script compris par les navigateurs.

Il sera très utile pour **créer des interactions** sur une page web.

# Des exemples d'utilisation

---

Afficher un décompte

Drag & Drop

Créer une calculatrice

...

# Comment l'utiliser ? (1/2)

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mon premier HTML lié à Javascript</title>
</head>
<body>
  <script src="mon-script.js"></script>
</body>
</html>
```

*index.html*

# Comment l'utiliser ? (2/2)

```
alert('Hello world!');
```

mon-script.js

# Exercice

- (1) Créer une nouvelle page HTML nommée « *index.html* »
- (2) Faire le lien entre l'HTML et un fichier JavaScript nommé « mon-script.js »
- (3) Dans ce javascript ajouter le code suivant :  
*alert('Bonjour');*
- (4) Dites moi ce que vous voyez à l'écran ?



# Algorithme

L'algorithme est un ensemble de règles décrivant de manière précise une séquence d'opérations.

Afin de pouvoir être exécuté de manière systématique, un algorithme doit absolument éviter l'ambiguïté. Or les langues naturelles, comme le français, sont intrinsèquement ambiguës. Il est donc indispensable d'écrire les algorithmes dans un langage formel, dont la sémantique est précisément définie.

# Les variables (le typage)

Il existe deux familles de types :

- Les types simples
- Les types avancés

# Les variables : les types simples

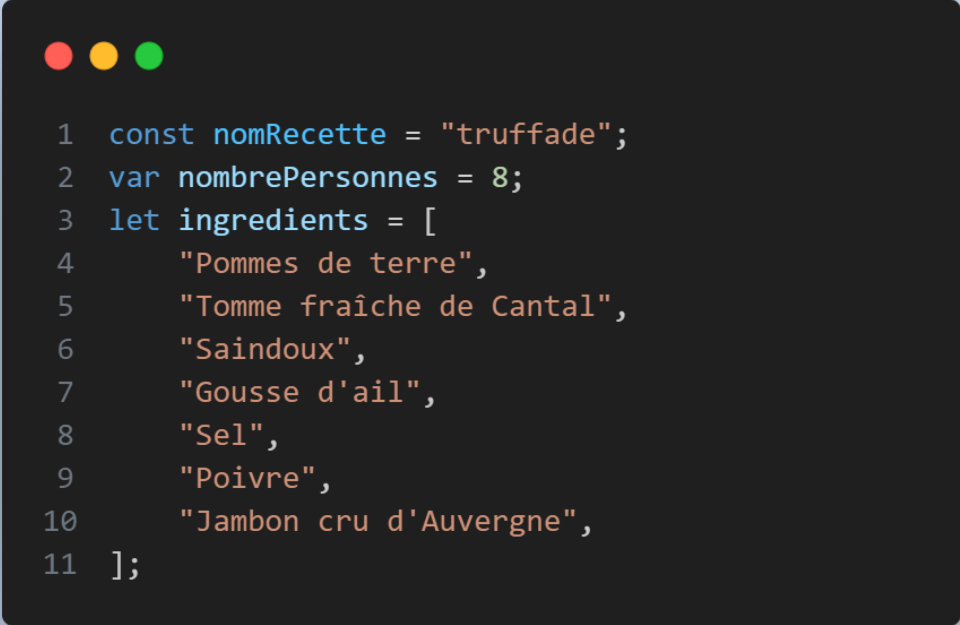
- nombre
- booléen: `true`, `false`
- chaîne de caractères / string: `'coucou'`
- `null`

# Les variables : les types avancés

- `undefined`
- objet: `{ prop: 'valeur' }`
- tableau / array: `[ 1, 2, 3 ]`
- fonction: `function(){ /* ... */ }`

# Les variables : exemple

Pour déclarer une variable en Javascript, nous utilisons les trois syntaxes suivantes :




```
1  const nomRecette = "truffade";
2  var nombrePersonnes = 8;
3  let ingredients = [
4      "Pommes de terre",
5      "Tomme fraîche de Cantal",
6      "Saindoux",
7      "Gousse d'ail",
8      "Sel",
9      "Poivre",
10     "Jambon cru d'Auvergne",
11 ];
```

# Les variables

---

Pour déclarer une variable en Javascript, nous utilisons les trois syntaxes suivantes :

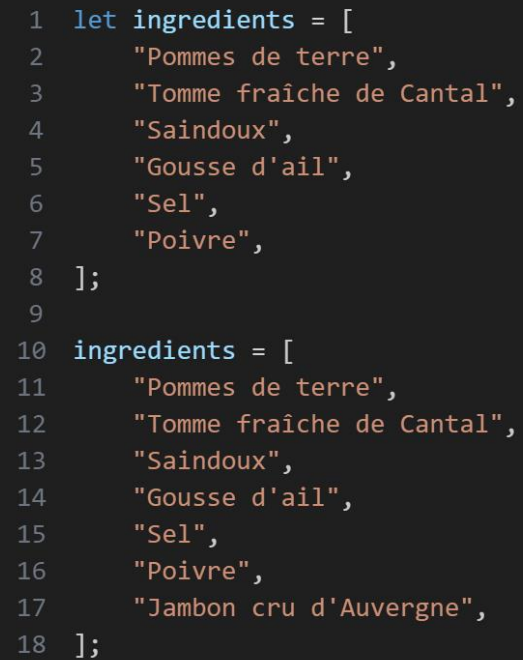


```
1  const nomRecette = "truffade";
2  var nombrePersonnes = 8;
3  let ingredients = [
4      "Pommes de terre",
5      "Tomme fraîche de Cantal",
6      "Saindoux",
7      "Gousse d'ail",
8      "Sel",
9      "Poivre",
10     "Jambon cru d'Auvergne",
11 ];
```

# Les variables

---

Le but d'une variable est qu'elle est **évolutive** au cours de l'algorithme

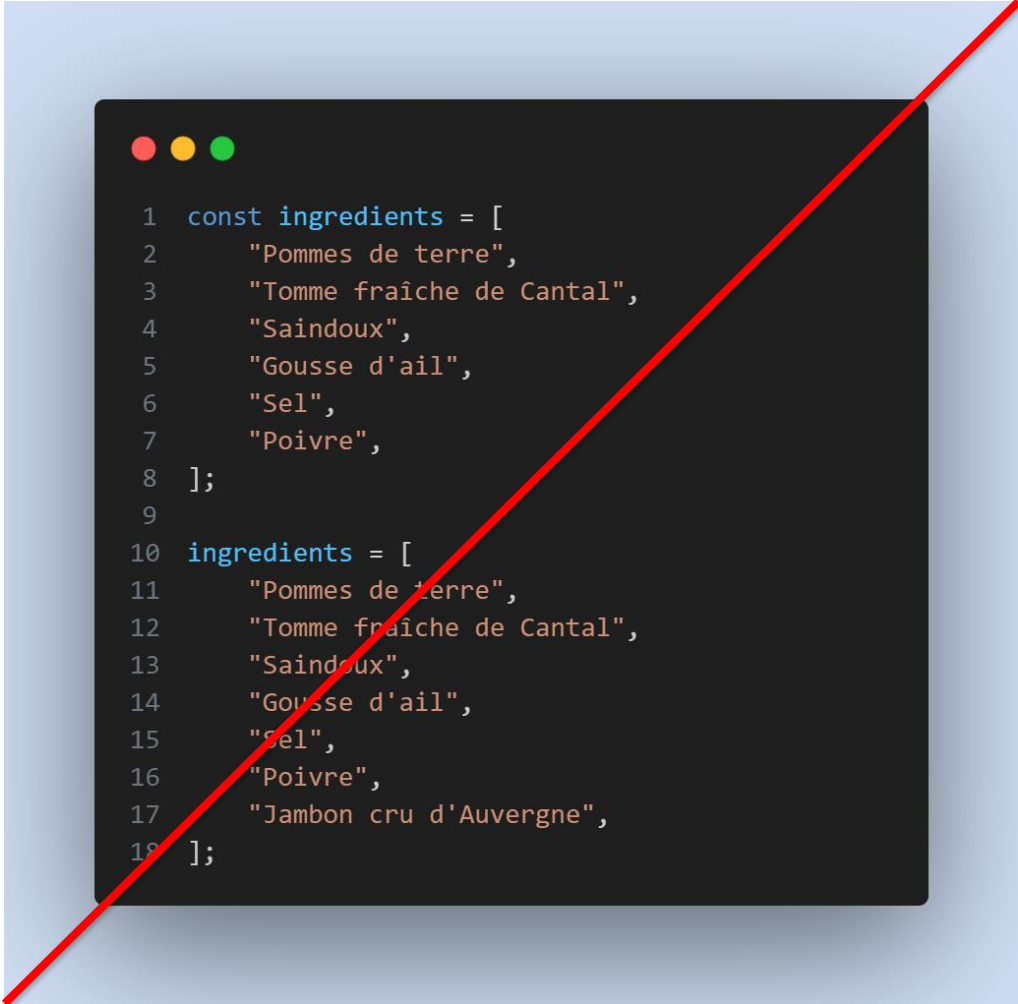


```
1 let ingredients = [  
2   "Pommes de terre",  
3   "Tomme fraîche de Cantal",  
4   "Saindoux",  
5   "Gousse d'ail",  
6   "Sel",  
7   "Poivre",  
8 ];  
9  
10 ingredients = [  
11   "Pommes de terre",  
12   "Tomme fraîche de Cantal",  
13   "Saindoux",  
14   "Gousse d'ail",  
15   "Sel",  
16   "Poivre",  
17   "Jambon cru d'Auvergne",  
18 ];
```

# Les variables

---

Attention, une variable constante n'est pas évolutive, d'où son nom.



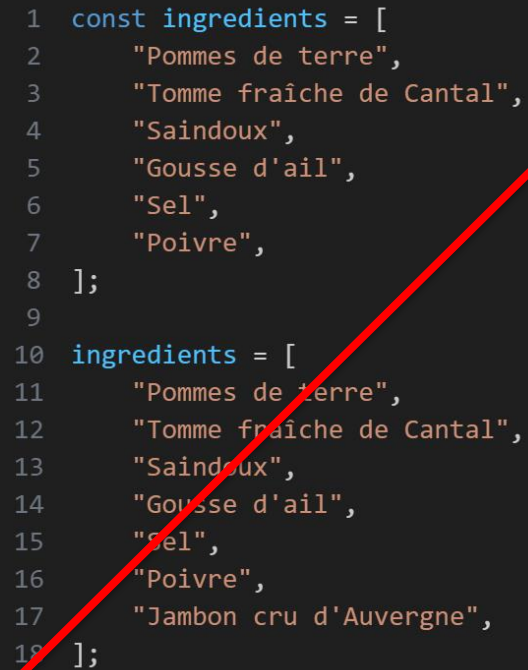
```
1  const ingredients = [  
2    "Pommes de terre",  
3    "Tomme fraîche de Cantal",  
4    "Saindoux",  
5    "Gousse d'ail",  
6    "Sel",  
7    "Poivre",  
8  ];  
9  
10 ingredients = [  
11   "Pommes de terre",  
12   "Tomme fraîche de Cantal",  
13   "Saindoux",  
14   "Gousse d'ail",  
15   "Sel",  
16   "Poivre",  
17   "Jambon cru d'Auvergne",  
18 ];
```



# Les variables

---

Attention, une variable constante n'est pas évolutive, d'où son nom.

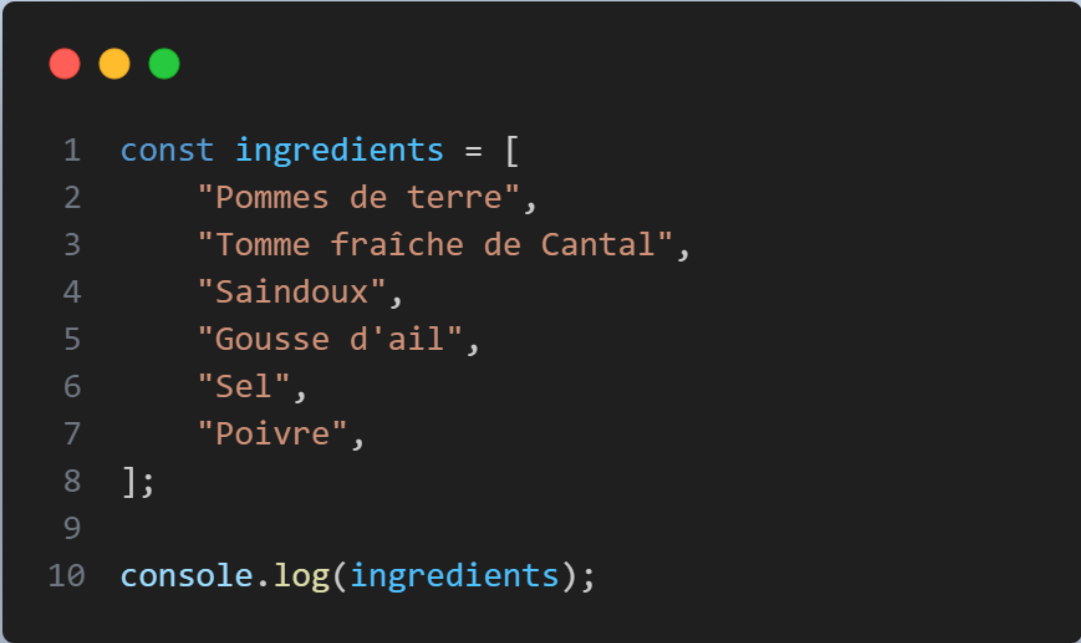


```
1  const ingredients = [  
2    "Pommes de terre",  
3    "Tomme fraîche de Cantal",  
4    "Saindoux",  
5    "Gousse d'ail",  
6    "Sel",  
7    "Poivre",  
8  ];  
9  
10 ingredients = [  
11   "Pommes de terre",  
12   "Tomme fraîche de Cantal",  
13   "Saindoux",  
14   "Gousse d'ail",  
15   "Sel",  
16   "Poivre",  
17   "Jambon cru d'Auvergne",  
18 ];
```

# Les variables : afficher dans la console

---

Il peut être très pratique d'afficher en console une variable.



```
1  const ingredients = [  
2      "Pommes de terre",  
3      "Tomme fraîche de Cantal",  
4      "Saindoux",  
5      "Gousse d'ail",  
6      "Sel",  
7      "Poivre",  
8  ];  
9  
10 console.log(ingredients);
```

# Les variables : afficher dans la console

Voici le rendu qu'on pourra avoir en affichant notre console : *en appuyant sur F12 depuis votre navigateur préféré*



# Exercice

(1) Reprendre le script « mon-script.js » et déclarer deux variables de type « entier » :

$a = 12$   
 $b = 58$

(2) Créer une troisième variable et faire l'addition de  $a$  et  $b$ .

(3) Afficher en console le résultat de cette addition

# Les commentaires

---

On retrouve deux types de commentaires :

- Le commentaire sur une seule ligne
- Le commentaire sur plusieurs lignes

# Les commentaires : exemples



```
1 let ilPleut = true; // un commentaire sur une seule ligne
2
3 /* Ceci est un commentaire
4 sur plusieurs lignes */
```

# Exercice

(1) En reprenant le code précédent (sur l'addition), ajoutez des commentaires pour que votre code soit compris pour un futur développeur qui reprendra votre travail.

# Les fonctions

---

Les fonctions font partie des briques fondamentales de JavaScript.

Une fonction est une procédure JavaScript, un ensemble d'instructions effectuant une tâche ou calculant une valeur. Afin d'utiliser une fonction, il est nécessaire de l'avoir auparavant définie au sein de la portée dans laquelle on souhaite l'appeler.

Le gros avantage d'utiliser les fonctions est que l'on pourra utiliser à plusieurs reprises dans la suite de notre code.



# Les fonctions

Une fonction est nommée

Une fonction peut avoir 0..n paramètres



Une fonction peut retourner une valeur

# Les fonctions : exemple pour une addition

Une fonction est nommée

=> addition

Une fonction peut avoir 0..n paramètres

=> a et b



Une fonction peut retourner une valeur

=> retourne  $a + b$

# Les fonctions : exemple de code



```
1 function addition(a, b) {  
2     return a + b;  
3 }  
4  
5 let somme = addition(10, 12);  
6 console.log(somme);
```

# Les fonctions fournies par Javascript

L'avantage est que Javascript est un langage nous donnant accès à bon nombre de fonctions, exemple :

(1) `alert()` : Affiche la sortie sur une boîte de dialogue du navigateur

(2) `toLowerCase()` : Convertit une chaîne de caractères en minuscules

(3) `Console.log()` : Affiche la sortie sur la console du navigateur, utile pour le débogage

Voici une liste des différentes fonctions que l'on peut utiliser :

[https://waytolearnx.com/2019/09/liste-des-fonctions-javascript.html?utm\\_content=cmp-true](https://waytolearnx.com/2019/09/liste-des-fonctions-javascript.html?utm_content=cmp-true)

# Exercice #1

- (1) Reprenez l'exemple précédent et définissez votre propre fonction que l'on nommera « addition »
- (2) *addition* est une fonction qui prendra deux paramètres : **a** et **b** – et qui retournera la **somme** de ces deux nombres

# Exercice #2

- (1) Demandez à l'utilisateur de saisir son prénom (il existe une fonction appelée déjà une fonction `prompt` fournie par Javascript)
- (2) Définir une fonction nommée « hello » permettant d'afficher en console :  
=> Bonjour *\*Le prénom saisi par la personne\** !

# Les conditions

---

Une condition est très utile dans un algorithme. Elle nous permet de tester si une valeur est bien celle attendue. La notion de « condition » nous permet d'ajouter de l'intelligence au sein de notre programme.

# Les conditions

---

Voici la syntaxe attendue en Javascript pour écrire une condition.




```
1  if (<booléen>) {  
2      // Code si VRAI  
3  }
```



# Les conditions

---

Dans l'exemple ci-dessous, nous demandons à l'utilisateur de saisir sa note. Si sa note est supérieur ou égale à 10, alors nous affichons en console « Bravo tu as la moyenne ! ».




```
1 let note = prompt('Saisie ta note');  
2  
3 if (note >= 10) {  
4     console.log('Bravo tu as la moyenne !');  
5 }
```

# Les conditions

---

Nous pouvons aller encore plus loin dans le cas où la condition n'est pas remplie.

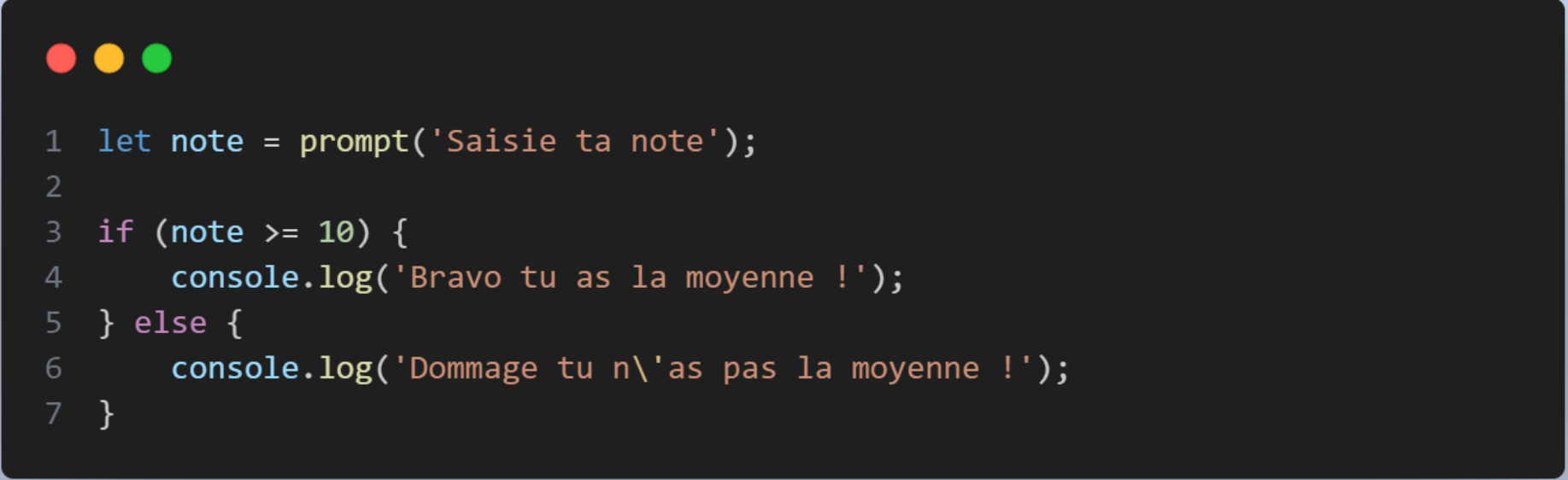


```
1  if (<booléen>) {  
2      // Code si VRAI  
3  } else {  
4      // Code si FAUX  
5  }
```

# Les conditions

---

Reprenons notre exemple de toute à l'heure et affichons « Dommage tu n'as pas la moyenne » lorsque l'utilisateur possède une note strictement inférieure à 10.

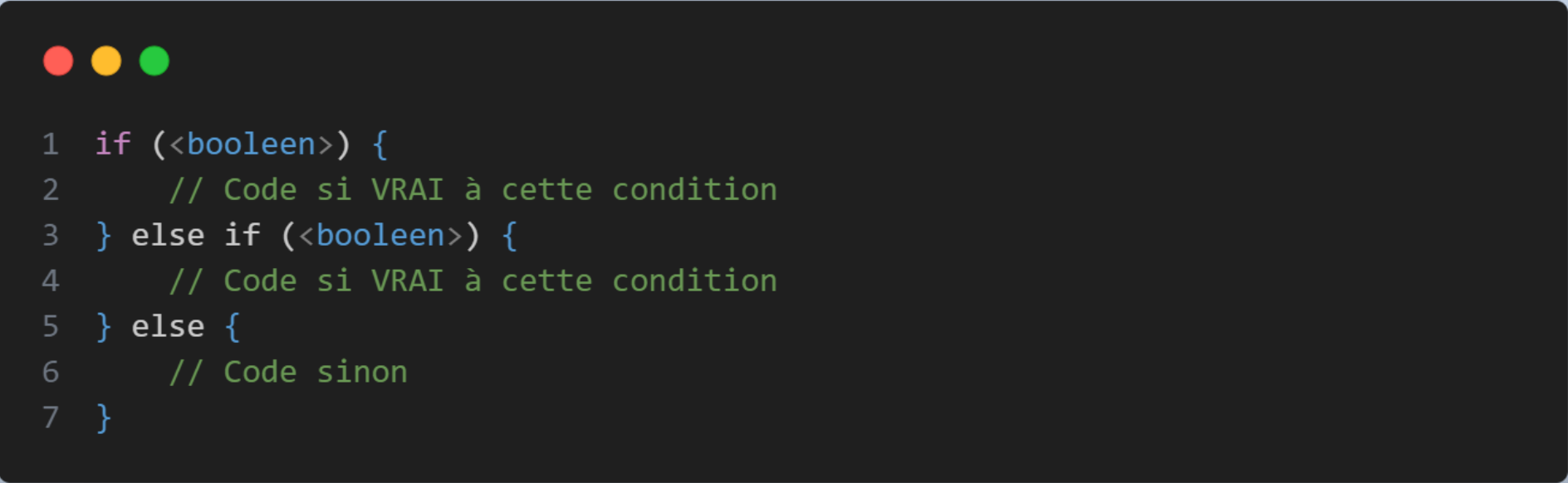


```
1 let note = prompt('Saisie ta note');
2
3 if (note >= 10) {
4     console.log('Bravo tu as la moyenne !');
5 } else {
6     console.log('Dommage tu n\'as pas la moyenne !');
7 }
```

# Les conditions

---

Nous pouvons également utiliser des conditions multiples

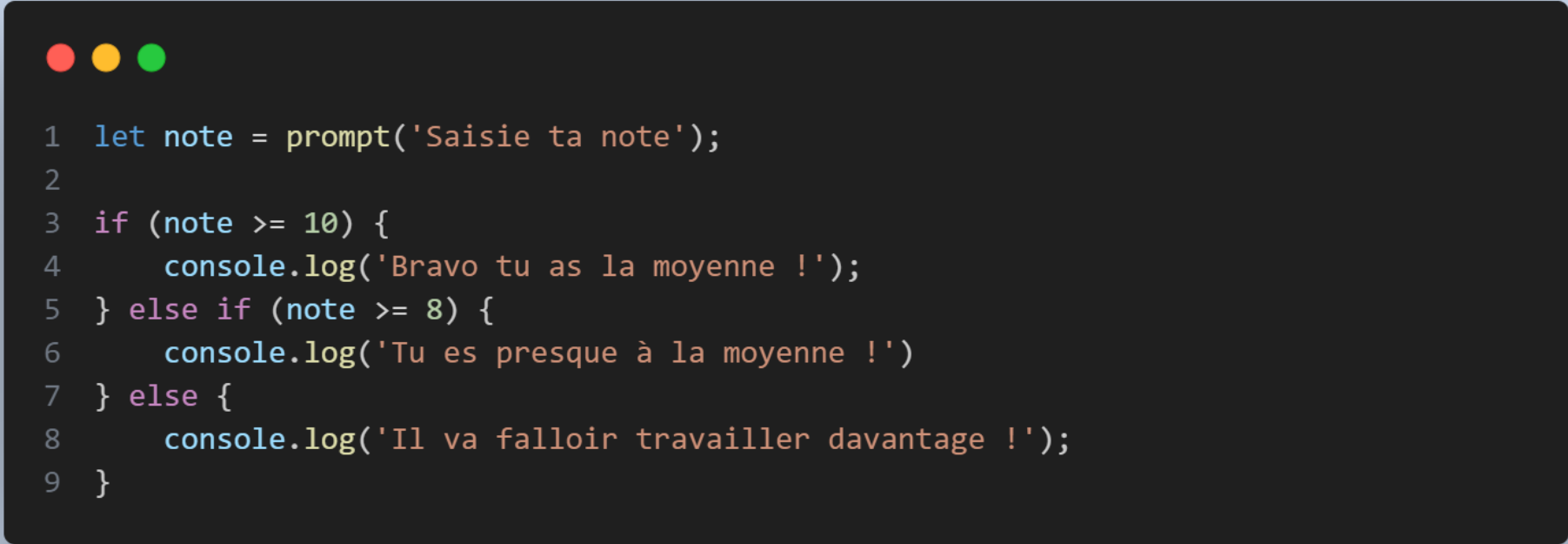


```
1 if (<booleen>) {  
2     // Code si VRAI à cette condition  
3 } else if (<booleen>) {  
4     // Code si VRAI à cette condition  
5 } else {  
6     // Code sinon  
7 }
```

# Les conditions


---

Reprenons notre dernier exemple en ajoutant une distinction au niveau des notes situées entre 8 et 10.



```
1 let note = prompt('Saisie ta note');
2
3 if (note >= 10) {
4     console.log('Bravo tu as la moyenne !');
5 } else if (note >= 8) {
6     console.log('Tu es presque à la moyenne !')
7 } else {
8     console.log('Il va falloir travailler davantage !');
9 }
```

# Les conditions : opérateurs de comparaison



```
1 a == b // a égale à b
2 a === b // a == b et a est de même "type" que b
3 a >= b // a supérieur OU égal à b
4 a > b // a strictement supérieur à b
5 a <= b // a inférieur OU égal à b
6 a < b // a strictement inférieur à b
7 a != b // a est différent de b
8 a !== b // a est strictement différent de b
```

# Exercice

Reprenez l'exemple précédent. Grâce aux conditions affichez en console une indication supplémentaire sur le résultat de la fonction "addition" :

- Le résultat de l'addition est inférieur à 0
- Le résultat de l'addition est compris entre 0 et 10
- Le résultat de l'addition est supérieur à 10

# Les tableaux

---

Les tableaux servent à conserver un ensemble d'informations dans une structure organisée, un peu comme une boîte de rangement où chaque élément a sa place. Cette collection peut accueillir divers types de données, allant des chiffres et des textes à des objets plus complexes. Imaginez un tiroir dans lequel vous pourriez non seulement mettre des chaussettes mais aussi d'autres petits tiroirs; de la même manière, un tableau est capable d'englober un autre tableau, permettant ainsi une flexibilité et une profondeur de stockage remarquables.



# Les tableaux : un exemple

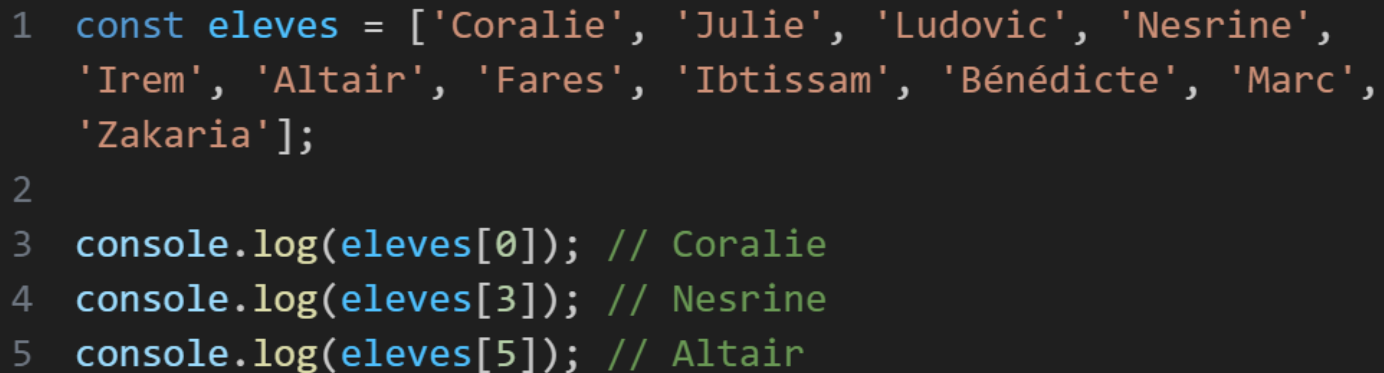


```
1 const eleves = ['Coralie', 'Julie', 'Ludovic', 'Nesrine',  
  'Irem', 'Altair', 'Fares', 'Ibtissam', 'Bénédicte', 'Marc',  
  'Zakaria'];
```

```
2
```

# Les tableaux : un exemple

Pour afficher la valeur d'une entrée dans un tableau, nous pouvons faire :



```
1 const eleves = ['Coralie', 'Julie', 'Ludovic', 'Nesrine',  
  'Irem', 'Altair', 'Fares', 'Ibtissam', 'Bénédicte', 'Marc',  
  'Zakaria'];  
2  
3 console.log(eleves[0]); // Coralie  
4 console.log(eleves[3]); // Nesrine  
5 console.log(eleves[5]); // Altair
```

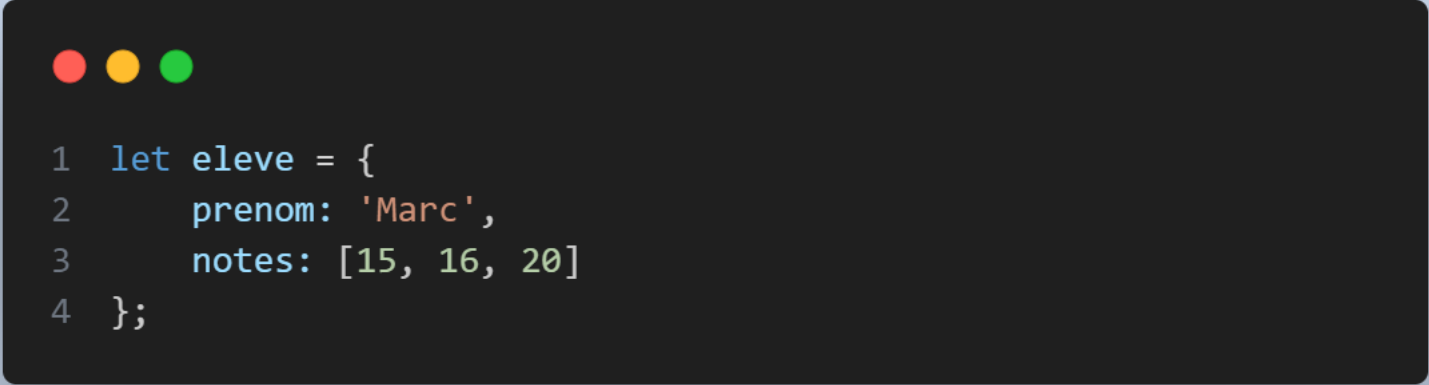
# Exercice

(1) Construisez un tableau d'entiers avec les valeurs : 10, 16, 80, 47, 49, 55, 68, 99

(2) Afficher l'ensemble de ces variables en console

# Les objets

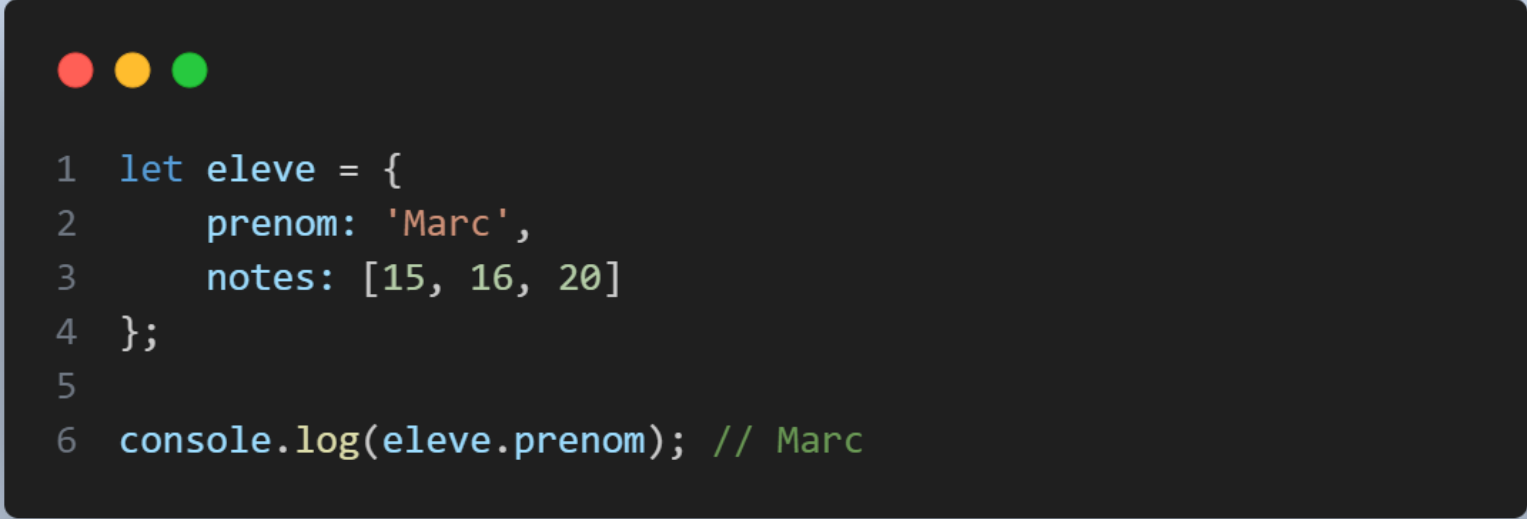
Les objets sont utilisés pour conserver des données plus élaborées qu'un simple tableau. Nous allons utiliser des index (ou clés) pour identifier nos différentes données stockées :



```
1 let eleve = {  
2   prenom: 'Marc',  
3   notes: [15, 16, 20]  
4 };
```

# Les objets

Pour pouvoir afficher le prénom de notre élève en console, on peut utiliser la syntaxe suivante :



```
1 let eleve = {  
2     prenom: 'Marc',  
3     notes: [15, 16, 20]  
4 };  
5  
6 console.log(eleve.prenom); // Marc
```

# Exercice

(1) Définir une variable d'objet reprenant ce schéma :

```
let eleve = {  
  prenom: 'Votre prénom',  
  notes: [15, 16, 20]  
};
```

(2) Afficher en console les différentes notes saisies : 15, 16, 20

# Les boucles

---

Les boucles permettent de répéter une certaine logique suivant une condition précise.


Par exemple :

- Afficher tous les entiers entre 0 et 10
- Pour tous les éléments d'un tableau, afficher les éléments
- ...

# Les boucles

---

Voici le code permettant d'afficher tous les entiers entre 1 et 10 :




```
1 let i = 1;
2
3 while(i <= 10) {
4     console.log(i);
5
6     i = i+1; // équivalent à i++
7 }
```



# Les boucles

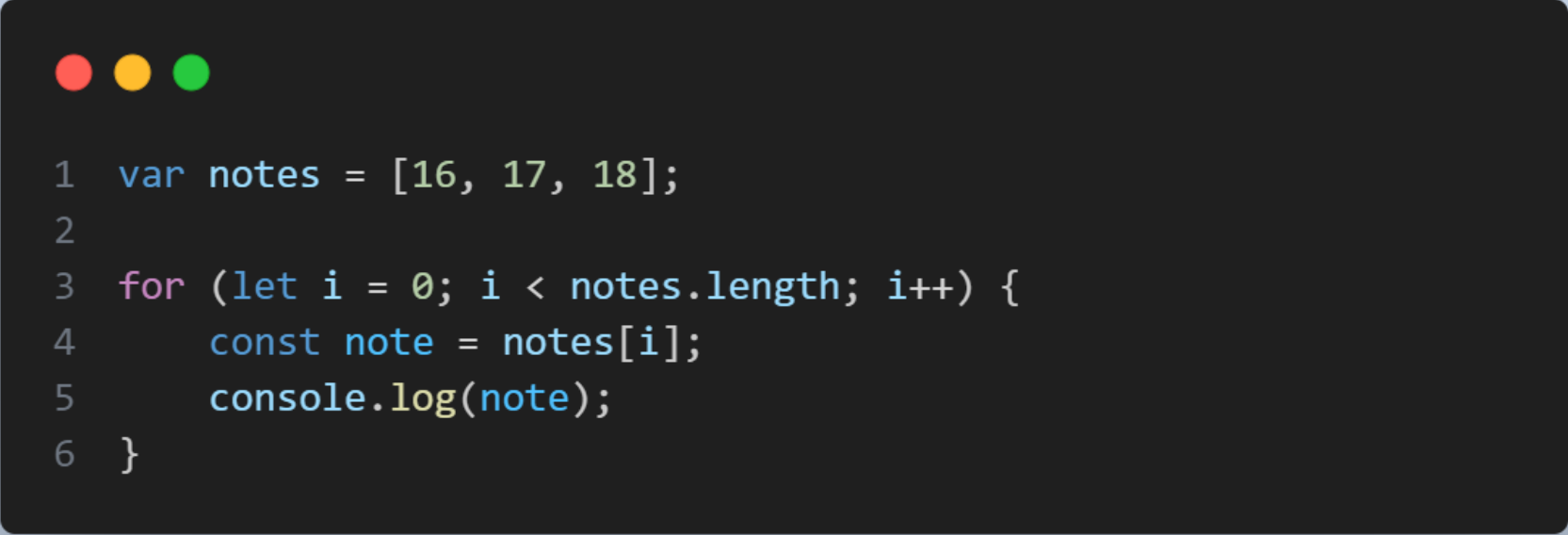
Il existe une alternative à while qu'on appelle « for ». Ceci affiche le même résultat que la diapo précédente.



```
1  for (let i = 1; i <= 10; i++) {  
2      console.log(i);  
3  }
```

# Les boucles

Il est possible de boucler sur des tableaux, ce qui est très pratique pour rendre un algorithme dynamique :



```
1 var notes = [16, 17, 18];  
2  
3 for (let i = 0; i < notes.length; i++) {  
4     const note = notes[i];  
5     console.log(note);  
6 }
```

# Exercice

(1) Reprenez le schéma précédent :

```
let eleve = {  
  prenom: 'Votre prénom',  
  notes: [15, 16, 20]  
};
```

(2) A l'aide d'une boucle, afficher toutes les notes.