

---

# Cours de Javascript II

The letters 'JS' in a bold, black, sans-serif font, centered within a bright yellow square. This square is positioned on the right side of the slide, overlapping a larger, semi-transparent orange rectangle that serves as a background element.

**JS**

---

# Partie II

---

1. Les variables et les types de valeurs
  - Présentation des variables
  - Les types de données
  - Les opérateurs arithmétiques
  - La concaténation
  - Les constantes

## 1.1 Présentation des variables

---

- Qu'est-ce qu'une variable ?

Une variable est un conteneur servant à stocker des informations de manière temporaire, comme une chaîne de caractères (un texte) ou un nombre par exemple.

Le propre d'une variable est de pouvoir varier, c'est-à-dire de pouvoir stocker différentes valeurs au fil du temps et c'est cette particularité qui les rend si utiles.

Notez bien déjà qu'une variable en soi et la valeur qu'elle va stocker sont deux éléments différents et qui ne sont pas égaux. Encore une fois, une variable n'est qu'un conteneur. Vous pouvez imaginer une variable comme une boîte dans laquelle on va pouvoir placer différentes choses au cours du temps.

## 1.1 Présentation des variables

---

- Les règles de déclaration des variables en JavaScript

Lorsqu'on stocke une valeur dans une variable, on dit également qu'on assigne une valeur à une variable.

Pour pouvoir utiliser les variables et illustrer leur intérêt, il va déjà falloir les créer. Lorsqu'on crée une variable en JavaScript, on dit également qu'on « déclare » une variable.

Pour déclarer une variable en JavaScript, nous allons devoir utiliser le mot clef **var** ou le mot clef **let** (explication de la différence entre les deux par la suite) suivi du nom qu'on souhaite donner à notre variable.

Concernant le nom de nos variables, nous avons une grande liberté dans le nommage de celles-ci mais il y a quand même quelques règles à respecter :

## 1.1 Présentation des variables

---

- Le nom d'une variable doit obligatoirement commencer par une lettre ou un **underscore** (**\_**) et ne doit pas commencer par un chiffre;
- Le nom d'une variable ne doit contenir que des lettres, des chiffres et des underscores mais pas de caractères spéciaux;
- Le nom d'une variable ne doit pas contenir d'espace.

De plus, le nom des variables est sensible à la casse en JavaScript. C'est à dire que l'usage de majuscules ou de minuscules avec un même nom va permettre de définir des variables différentes. Par exemple, les noms **texte**, **TEXTE** et **tEXTe** vont pouvoir définir des variables différentes.

Enfin, sachez qu'il existe des noms « réservés » en JavaScript. Vous ne pouvez pas utiliser ces noms comme noms pour vos variables, tout simplement car le langage JavaScript les utilise déjà pour désigner différents éléments intégrés au langage.

## 1.1 Présentation des variables

---

Vous pouvez également noter qu'on utilise généralement la convention **lower camel case** pour définir les noms de variable en JavaScript. Elle stipule simplement que lorsqu'un nom de variable est composé de plusieurs mots, on colle les mots ensemble en utilisant une majuscule pour chaque mot sauf le premier. Par exemple, si on décide de nommer une variable « monage » on écrira en JavaScript **let monAge** ou **var monAge**.

```
js > JS script.js > ...  
1  let prenom;  
2  let monAge;
```

Nos deux premières variables sont désormais créées. Cependant, elles ne stockent aucune valeur pour le moment.

## 1.1 Présentation des variables

---

- Initialiser une variable

Lorsqu'on assigne une valeur pour la première fois à une variable, c'est-à-dire lorsqu'on stocke une valeur dans une variable qui n'en stockait pas encore, on dit également qu'on initialise une variable.

On va pouvoir initialiser une variable après l'avoir déclarée ou au moment de sa déclaration. Les deux façons de faire sont équivalentes en termes de résultat mais il est plus rapide (en termes d'écriture de code) d'initialiser une variable lors de sa déclaration puisque cela nous va nous éviter d'avoir à réécrire le nom de la variable.

Pour initialiser une variable, on utilise l'opérateur = qui est dans ce cas non pas un opérateur d'égalité mais un opérateur d'assignation ou d'affectation comme ceci :

## 1.1 Présentation des variables

---

```
// On déclare et on initialise la variable en même temps
let prenom = 'André';

// On déclare la variable puis on l'initialise
let monAge;
monAge = 45;
```

Le mot clef **let** (ou **var**) n'est utilisé et ne doit être utilisé que pour déclarer une variable. Lorsqu'on manipule une variable ensuite, on se contente d'utiliser son nom.

Vous remarquerez qu'on utilise des apostrophes droites ou guillemets simples pour entourer la valeur « André » mais pas pour la valeur « 45 ». Cela est dû au fait que « Pierre » est une valeur textuelle tandis que « 45 » est un chiffre et ces valeurs ne vont pas pouvoir être manipulées de la même façon en JavaScript. (Détails par la suite)



## 1.1 Présentation des variables

---

- Modifier la valeur stockée dans une variable

Pour affecter une nouvelle valeur dans une variable déjà initialisée, on va se contenter d'utiliser à nouveau l'opérateur d'affectation `=`.

En faisant cela, la nouvelle valeur va venir écraser l'ancienne valeur stockée qui sera alors supprimée.

```
// On déclare et on initialise la variable en même temps
let prenom = 'André';

// On déclare la variable puis on l'initialise
let monAge;
monAge = 45;

/* On modifie la valeur stockée dans prenom.
La variable stocke désormais la valeur "Laura" */
prenom = 'Laura';
```

## 1.1 Présentation des variables

---

- La différence entre les mots clefs `let` et `var`

Lorsque le JavaScript a été créé et jusqu'à il y a quelques années, nous n'avions accès qu'au mot clef `var` qu'on devait utiliser pour déclarer nos variables.

Finalement, les créateurs du JavaScript ont fini par penser que le mot clef `var` pouvait porter à confusion et ont créé un nouveau mot clef pour déclarer les variables: le mot clef `let`.

En même temps qu'un nouveau mot clef a été créé, ils en ont profité pour résoudre quelques problèmes liés à la déclaration de variables en utilisant `var`, ce qui fait que `let` ne va pas nous permettre de créer des variables de la même façon que `var`.

Il existe 3 grandes différences de comportement entre les variables déclarées avec `var` et avec `let` :

## 1.1 Présentation des variables

---

### La remontée ou « hoisting » des variables

Lorsqu'on utilise la syntaxe avec **var**, on n'est pas obligé de déclarer la variable avant de la manipuler dans le code, on peut très bien effectuer des manipulations en haut du code et la déclarer en fin de code.

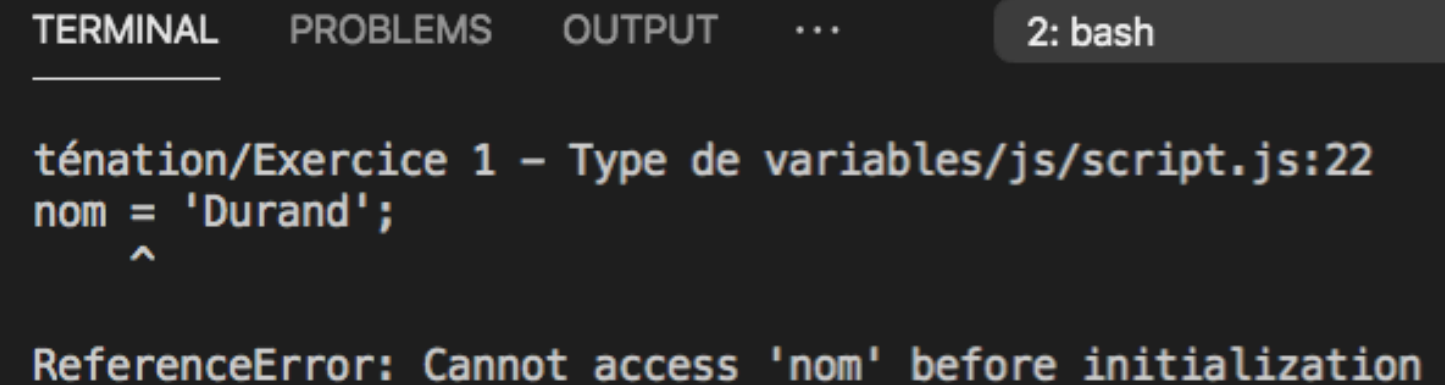
Cela est possible car le JavaScript va traiter les déclarations de variables effectuées avec **var** avant le reste du code JavaScript. Ce comportement est appelé remontée ou hoisting.

Ce comportement a été jugé comme inadapté dans les versions récentes de JavaScript et a donc été corrigé dans la déclaration de variables avec **let**: les variables utilisant la syntaxe **let** doivent obligatoirement être déclarées avant de pouvoir être utilisées.

## 1.1 Présentation des variables

```
// Ceci fonctionne
prenom = 'André';
var prenom;

// Ceci ne fonctionne pas et renvoie une erreur
nom = 'Durand';
let nom;
console.log(nom);
```



Terminal window showing a JavaScript error. The terminal title bar includes '2: bash'. The command prompt shows the file path 'ténation/Exercice 1 - Type de variables/js/script.js:22'. The code being executed is 'nom = 'Durand';' with a caret under the 'n' in 'nom'. The error message displayed is 'ReferenceError: Cannot access 'nom' before initialization'.

### La redéclaration de variables

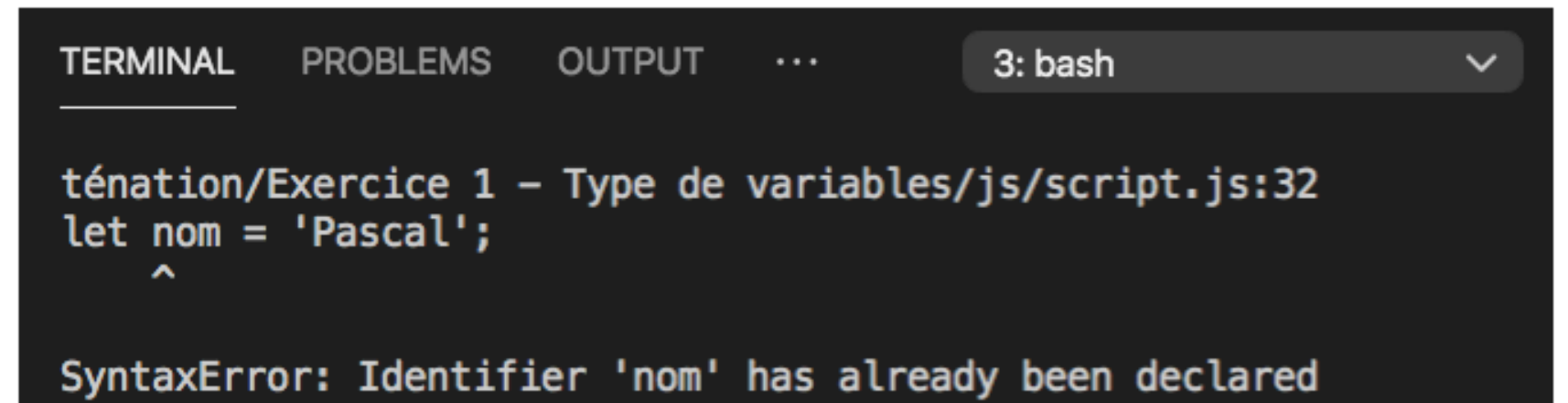
Avec l'ancienne syntaxe **var**, on avait le droit de déclarer plusieurs fois une même variable en utilisant à chaque fois **var** (ce qui avait pour effet de modifier la valeur stockée).

La nouvelle syntaxe avec **let** n'autorise pas cela. Pour modifier la valeur stockée dans une variable avec la nouvelle syntaxe, il suffit d'utiliser le nom de la variable et de lui affecter une autre valeur.

# 1.1 Présentation des variables

```
// Ceci fonctionne
var prenom = 'André';
var prenom = 'Laura';

// Ceci ne fonctionne pas et renvoie une erreur
let nom = 'Durand';
let nom = 'Pascal';
console.log(prenom, nom);
```

A screenshot of a code editor's terminal window. The window has tabs for 'TERMINAL', 'PROBLEMS', and 'OUTPUT'. The 'TERMINAL' tab is active, showing a command prompt '3: bash'. Below the prompt, the file path 'ténation/Exercice 1 - Type de variables/js/script.js:32' is displayed. The code 'let nom = 'Pascal';' is shown with a caret under the 'n' in 'nom'. Below the code, the error message 'SyntaxError: Identifier 'nom' has already been declared' is displayed.

```
TERMINAL PROBLEMS OUTPUT ... 3: bash
ténation/Exercice 1 - Type de variables/js/script.js:32
let nom = 'Pascal';
  ^
SyntaxError: Identifier 'nom' has already been declared
```

## La portée des variables

La « portée » d'une variable désigne l'endroit où cette variable va pouvoir être utilisée dans un script. Il est un peu tôt pour vous expliquer ce concept puisque pour bien le comprendre il faut déjà savoir ce qu'est une fonction.

Nous reparlerons donc de cette portée des variables lorsque nous aborderons les fonctions en JavaScript.

## 1.1 Présentation des variables

---

Vous pouvez pour le moment retenir que les variables déclarées avec **var** et celles avec **let** au sein d'une fonction ne vont pas avoir la même portée, c'est-à-dire qu'on ne va pas pouvoir les utiliser aux mêmes endroits.

### Le choix de la déclaration des variables : plutôt avec let ou plutôt avec var

La syntaxe de déclaration des variables avec **let** correspond à la nouvelle syntaxe. La syntaxe avec **var** est l'ancienne syntaxe qui est vouée à disparaître.

Vous devriez donc aujourd'hui toujours utiliser le mot clef **let** pour déclarer vos variables.

### Quelle utilité pour les variables en pratique ?

Les variables vont être à la base de la plupart de nos scripts JavaScript. En effet, il va être très pratique de stocker différents types d'informations dans les variables

## 1.1 Présentation des variables

---

pour ensuite manipuler simplement ces informations notamment lorsqu'on n'a pas accès à ces informations lorsqu'on crée le script.

Par exemple, on va pouvoir demander à des utilisateurs de nous envoyer des données grâce à la fonction `prompt()`. Lorsqu'on écrit notre script avec notre fonction `prompt()`, on ne sait pas encore ce que les utilisateurs vont nous envoyer comme données. Dans ce cas, notre script va être créé de manière à ce que les données envoyées soient stockées lors de leur envoi dans des variables qu'on définit. Cela nous permet déjà de pouvoir manipuler les dites variables et par extension les données qu'elles vont stocker.

À ce propos, il existe de nombreux moyens d'afficher le contenu d'une variable en JavaScript, que ce soit via la console JavaScript du navigateur, en utilisant une fonction `alert()` ou encore en insérant le contenu de notre variable au sein du contenu HTML de notre page.



## 1.2 Les types de données

---

En JavaScript, il existe 7 types de valeurs différents. Chaque valeur qu'on va pouvoir créer et manipuler en JavaScript va obligatoirement appartenir à l'un de ces types. Ces types sont les suivants:

- **String** ou « chaîne de caractères » en français;
- **Number** ou « nombre » ;
- **Boolean** ou « booléen » ;
- **Null** ou « nul / vide » ;
- **Undefined** ou « indéfini » ;
- **Symbol** ou « symbole » ;
- **Object** ou « objet » ;



## 1.2 Les types de données

---

- Le type chaîne de caractères ou String

Le premier type de données qu'une variable va pouvoir stocker est le type String ou chaîne de caractères. Une chaîne de caractères est une séquence de caractères, ou ce qu'on appelle communément un texte.

Notez que toute valeur stockée dans une variable en utilisant des guillemets ou des apostrophes sera considérée comme une chaîne de caractères, et ceci même dans le cas où nos caractères sont à priori des chiffres:

```
let prenom = "Je m'appelle Julia";  
let age = 36;  
let age2 = '36';
```

L'utilisation de guillemets ou d'apostrophe fait qu'une valeur est immédiatement considérée comme une chaîne de caractères, quelle que soit cette valeur.

## 1.2 Les types de données

Pour s'en convaincre, on peut utiliser la fonction `typeof` qui nous permet de vérifier le type d'une valeur.

```
let prenom = "Je m'appelle Julia";
let age = 36;
let age2 = '36';

document.getElementById('p1').innerHTML = 'Type de prenom : ' + typeof prenom;
document.getElementById('p2').innerHTML = 'Type de age : ' + typeof age;
document.getElementById('p3').innerHTML = 'Type de age2 : ' + typeof age2;

console.log(typeof prenom, typeof age, typeof age2);
```

```
<> index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <script type="text/javascript" src="js/script.js"></script>
8      <title>Exercices - Les variables</title>
9  </head>
10
11 <body>
12     <h1>Titre principal</h1>
13     <p>Un paragraphe</p>
14     <p id='p1'></p>
15     <p id='p2'></p>
16     <p id='p3'></p>
17 </body>
18
19 </html>
```

Une note: vous pouvez retenir que `document.getElementById(id)` nous permet d'accéder à l'élément HTML qui possède l'`id` précisé. Ensuite, `innerHTML` nous permet d'injecter du texte dans l'élément. Dans le cas présent, on injecte « Type de ... » suivi du type de la variable qui est renvoyé par `typeof`.

## 1.2 Les types de données

---

### Un point sur l'utilisation des guillemets et apostrophes droits et sur l'échappement

Si on avait utilisé des apostrophes pour délimiter la valeur « je m'appelle Julia » stockée dans **let prenom**, JavaScript aurait considéré que l'apostrophe dans « m'appelle » correspond à la fin de la chaîne de caractère.

Pour éviter cela, on va «**échapper**» la chaîne, c'est-à-dire neutraliser sa signification spéciale:

```
// Délimiteurs non trouvés dans la chaîne = rien à échapper
let a = "Je m'appelle Julia";

// Délimiteurs non trouvés dans la chaîne (apostrophe non droit) = rien à échapper
let b = 'Je m'appelle Julia';

// Délimiteurs trouvés dans la chaîne = on échappe le caractère en question
let c = 'Je m\'appelle Julia';

// Délimiteurs non trouvés dans la chaîne = rien à échapper
let d = "Je m'appelle «Julia»";

// Délimiteurs non trouvés dans la chaîne = rien à échapper
let e = "Je m'appelle \"Julia\"";
```

## 1.2 Les types de données

---

- Le type nombre ou Number

Les variables en JavaScript vont également pouvoir stocker des nombres. En JavaScript, il n'existe qu'un type prédéfini qui va regrouper tous les nombres qu'ils soient positifs, négatifs, entiers ou décimaux (à virgule) et qui est le type **Number**.

```
let x = 10; // x stocke un entier positif
let y = -5; // y stocke un entier négatif
let z = 2.17; // z stocke un nombre décimal positif
```

Un booléen, en algèbre, est une valeur binaire (soit 0, soit 1). En informatique, le type booléen est un type qui ne contient que deux valeurs : les valeurs **true** (vrai) et **false** (faux).

Il est particulièrement utile en valeur de test pour vérifier si le test est validé ou non.

## 1.2 Les types de données

---

- Les types de valeurs Null et Undefined

La valeur **null** correspond à l'absence de valeur ou du moins à l'absence de valeur connue. Pour qu'une variable contienne **null**, il va falloir stocker cette valeur qui représente donc l'absence de valeur de manière explicite.

La valeur **null** va être utile dans certains cas où on souhaite explicitement indiquer une absence de valeur connue.

La valeur **undefined** correspond à une variable «non définie», c'est-à-dire une variable à laquelle on n'a pas affecté de valeur.

Si on déclare une variable sans lui attribuer de valeur, alors son type sera **Undefined**. Si on déclare une variable et qu'on lui passe **null**, alors son type sera **Object**.

## 1.3 Présentation des opérateurs arithmétiques

---

- Qu'est-ce qu'un opérateur ?

Un opérateur est un symbole qui va être utilisé pour effectuer certaines actions notamment sur les variables et leurs valeurs.

Il existe différents types d'opérateurs qui vont nous servir à réaliser des opérations de types différents. Les plus fréquemment utilisés sont :

- Les opérateurs arithmétiques ;
- Les opérateurs d'affectation / d'assignation ;
- Les opérateurs de comparaison ;
- Les opérateurs d'incrémentation et décrémentation ;
- Les opérateurs logiques ;
- L'opérateur de concaténation ;
- L'opérateur ternaire ;
- l'opérateur virgule.

## 1.3 Présentation des opérateurs arithmétiques

---

- Les opérateurs arithmétiques

On peut utiliser 5 opérateurs arithmétiques :

- l'addition (+) ;
- la soustraction (-) ;
- la multiplication (\*) ;
- la division (/) ;
- le modulo (%). Le modulo est le reste d'une division.

ou :

```
var number1 = 3,  
    number2 = 2,  
    result;  
result = number1 * number2;  
alert(result);  
//Affiche : « 6 »
```

```
var number = 3;  
number = number + 5;  
alert(number);  
//Affiche : « 8 »
```

qui équivaut à :

```
var number = 3;  
number += 5;  
alert(number);  
//Affiche : « 8 »
```



## 1.3 Présentation des opérateurs arithmétiques

---

- Les opérateurs d'affectation

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

L'opérateur d'affectation le plus utilisé qui est le signe **=**. Cependant, 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	Définition
<b>+=</b>	Additionne puis affecte le résultat
<b>-=</b>	Soustrait puis affecte le résultat
<b>*=</b>	Multiplie puis affecte le résultat
<b>/=</b>	Divise puis affecte le résultat
<b>%=</b>	Calcule le modulo puis affecte le résultat



## 1.3 Présentation des opérateurs arithmétiques

---

```
let x = 4; //x stocke 2
let y = 10; //y stocke 10

/*On ajoute 3 à la valeur stockée précédemment par x (4) puis on
*affecte le résultat à x. x stocke désormais 4 + 3 = 7*/
x += 3;

/*On multiplie la valeur de y (10) par celle de z (7) puis on affecte
*le résultat à y. y stocke désormais 10 * 7 = 70*/
y *= x;

alert('x stocke : ' + x + ' et y stocke : ' + y);
```

- Les opérateurs d'affectation

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

## 1.4 La concaténation

---

- La concaténation

Concaténer consiste à ajouter une chaîne de caractères à la fin d'une autre, les mettre «mettre bout à bout» en utilisant l'opérateur de concaténation qui est le signe **+**.

```
var hi = 'Bonjour ',  
    name = 'toi qui lit ceci !',  
    result;  
result = hi + name;  
alert(result);  
// Affiche : « Bonjour toi qui lit ceci !»
```

ou :

```
var text = 'Bonjour ';  
text += 'toi qui lit ceci !';  
alert(text);  
// Affiche « Bonjour toi qui lit ceci !»
```

## 1.5 Les constantes

---

- Définition et utilité des constantes en JavaScript

Une constante est similaire à une variable au sens où c'est également un conteneur pour une valeur. Cependant, à la différence des variables, on ne va pas pouvoir modifier la valeur d'une constante.

En effet, une fois qu'une valeur est attribuée à une constante, celle-ci est attribuée de façon définitive et ne va pas pouvoir être modifiée. C'est d'ailleurs de là que les constantes portent leur nom : car leur valeur est constante.

- Déclarer une constante

Pour créer ou déclarer une constante en JavaScript, nous allons utiliser le mot clef **const**.

## 1.5 Les constantes

---

On va pouvoir déclarer une constante exactement de la même façon qu'une variable à la différence qu'on va utiliser **const** à la place de **let**.

Il faut obligatoirement initialiser une constante lors de sa déclaration, c'est-à-dire lui passer une valeur immédiatement sinon une erreur sera retournée.

```
const prenom = 'Julia';  
const age = 36;  
console.log(prenom, age);
```

## Liens externes pour aller plus loin

---

- [https://developer.mozilla.org/fr/docs/Learn/JavaScript/First\\_steps/Variables](https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/Variables)
- [https://www.w3schools.com/js/js\\_variables.asp](https://www.w3schools.com/js/js_variables.asp) (en anglais)