

# TYPES ET VARIABLES

# Objectifs

- Travailler avec des types primitifs en JS
- Comprendre la différence entre `let` et `const`
- Utiliser des template strings
- Travailler avec des opérateurs et des méthodes communes

## Write Your Review



Click on a star to change your rating 1 - 5, where 5 = great! and 1 = really bad

### Your Review:

Your Reivew

999 Characters remaining

### Your Info:

#### Name:

Name

#### Email:

Email

☐ I Agree to the Terms blah blah blah

Submit

# Types primitifs

- **Number**
- **Boolean**
- **String**
- **Null**
- **Undefined**

\* Le type “object” n’est pas considéré comme “primitif” car il sert à stocker des collections de données.

50

7

3.865

-34

-743.23

**JavaScript n'a qu'un seul type pour les nombres.**

- Positifs!
- Négatifs!
- Entiers!
- Décimales!\*

\* Attention à la précision

```
// Addition
50 + 50 //100

//Soustraction
90 - 1 //89

//Multiplication
111 * 7 //777

//Division
25 / 10 //2.5

//Modulo (reste)
27 % 2 //1

//Exponentiation
2 ** 4 //16
```

**On peut faire des  
opérations mathématiques  
sur ces nombres.**

// CRÉÉ UN COMMENTAIRE

```
// Infinity
```

```
1 / 0 //Infinity
```

```
//Not a number
```

```
0 / 0 //NaN
```

```
1 + NaN //NaN
```

```
"sympa" / 2 //NaN
```

**Il existe des valeurs  
numériques spéciales.**

```
Math.PI  
// 3.141592653589793
```

```
Math.round(0.9)  
// 1
```

```
Math.random()  
// 0.39280721703248656
```

**L'objet Math nous donne accès à différentes valeurs et méthodes utiles pour faire des calculs (arrondir, trigonométrie).**

**Une des méthodes que vous utiliserez le plus est**

**`Math.random()`**



let quelqueChose = valeur;

NOM DE LA VARIABLE

MOT CLÉ

OPÉRATEUR D'AFFECTATION

```
let let = valeur;  
//réutilisation d'un mot clé
```

```
let mon-nom = valeur;  
// lettres, chiffres,$ et _
```

```
let 5ympa = valeur;  
// Le premier caractère ne  
doit pas être un chiffre.
```

```
let monnom = valeur;  
// On favorise le camelCase  
pour enchaîner les mots
```

```
let x = valeur;  
// On essaie de donner un  
nom pertinent
```

**Il y a certaines  
conventions à respecter  
pour la façon dont vous  
nommez vos variables.**

```
let score = 1;  
score = -score; // -1
```

```
score++; // 0  
score--; // -1
```

```
score += 10; // 9  
score *= 3; // 27
```

**Il existe des opérateurs unaires (qui fonctionnent avec un seul opérande) et des opérateurs d'assignation courts.**

\*NOUS VERRONS LE “+” UNAIRE QUAND NOUS PARLERONS DE CONVERSION DE TYPES.

const LUMIERE = 299792458;

NOM DE LA CONSTANCE

MOT CLÉ

OPÉRATEUR D'AFFECTATION

```
const PI = 3.1415926;  
const seed = Math.random()
```

**On utilise les majuscules pour nommer une constante uniquement si sa valeur est “hardcoded”.**

**Si la valeur est calculée, on utilise les mêmes conventions que pour une variable.**

**On utilise systématiquement une constante pour stocker des tableaux et des objets.**

var

"use strict"

# Types primitifs

- ~~Number~~
- **Boolean**
- String
- Null
- Undefined

\* Le type “object” n’est pas considéré comme “primitif” car il sert à stocker des collections de données.



**true**

**false**

```
let isRunning = true;
```

```
isRunning = 666;
```

```
// Est-ce que ceci est permis?
```

```
let isRunning = true;  
isRunning = 666;  
// oui!
```



**En JavaScript, les  
variables peuvent  
changer de type!**



**En JavaScript, les  
variables peuvent  
changer de type!**

# Types primitifs

- ~~Number~~
- ~~Boolean~~
- **String**
- BigInt
- null
- undefined

\* Le type “object” n’est pas considéré comme “primitif” car il sert à stocker des collections de données.

**Les guillemets simples et doubles fonctionnent de la même façon. Choisissez un style et tenez-vous y.**

```
let titre = "Major"; // " ... "  
let message = 'Il dit "Floating in my tin can."'; // ' ... '  
let prenom = "Tom"; // Erreur!
```

Si l'opérateur binaire **+** est appliqué à des Strings, il les fusionne (Concaténation).

```
let titre = "Major";  
let prenom = "Tom";  
let nomComplet = titre + prenom;  
// Major Tom
```

```
let x = 1;  
let y = "12";  
console.log(x + y);  
// Qu'est-ce qui va s'afficher  
// dans la console?
```



```
// Opérateur binaire +  
10 + 12 // 22  
"Major" + "Tom" //"Major Tom"
```

```
// Opérateur unaire +  
let x = 1;  
+x // 1
```

```
let y = "10"  
+y // 10
```

```
x + +y // 11
```

**L'unaire `+` appliqué à une seule valeur, ne fait rien avec les nombres, mais si l'opérande n'est pas un nombre, alors il est converti en nombre.**

Vous pouvez également convertir des strings en nombres avec les fonctions `parseInt()` et `parseFloat()`.

Leur comportement diffère du `+` unaire. Vous pouvez référer au tableau ci-contre.

x	parseInt(x)	parseFloat(x)	Number(x)	+x
"123"	123	123	123	123
" +123"	123	123	123	123
" -123"	-123	-123	-123	-123
"123.45"	123	123.45	123.45	123.45
" -123.45"	-123	-123.45	-123.45	-123.45
"12e5"	12	1200000	1200000	1200000
"12e-5"	12	0.00012	0.00012	0.00012
"0123"	123	123	123	123
"0000123"	123	123	123	123
"0b111"	0	0	7	7
"0o10"	0	0	8	8
"0xBABE"	47806	0	47806	47806
"4294967295"	4294967295	4294967295	4294967295	4294967295
"123456789012345678"	123456789012345680	123456789012345680	123456789012345680	123456789012345680
"12e999"	12	Infinity	Infinity	Infinity
" "	NaN	NaN	0	0
"123foo"	123	123	NaN	NaN
"123.45foo"	123	123.45	NaN	NaN
" 123 "	123	123	123	123
"foo"	NaN	NaN	NaN	NaN
"12e"	12	12	NaN	NaN
"0b567"	0	0	NaN	NaN
"0o999"	0	0	NaN	NaN
"0xFUZZ"	15	0	NaN	NaN
" +0"	0	0	0	0
" -0"	0	0	0	0
"Infinity"	NaN	Infinity	Infinity	Infinity
" +Infinity"	NaN	Infinity	Infinity	Infinity
" -Infinity"	NaN	-Infinity	-Infinity	-Infinity
BigInt(1)	1	1	1	Error
null	NaN	NaN	0	0
undefined	NaN	NaN	NaN	NaN
true	NaN	NaN	1	1
false	NaN	NaN	0	0
Infinity	NaN	Infinity	Infinity	Infinity
NaN	NaN	NaN	NaN	NaN

P	O	U	L	E	T
0	1	2	3	4	5

```
// format: string.method()
```

```
let msg = "Say hello to my little  
friend.";
```

```
msg.toUpperCase();  
// "SAY HELLO TO MY LITTLE FRIEND"  
// Rappel: msg est immuable
```

```
let couleur = " jaune ";  
couleur.trim();  
// "jaune"
```

```
// On peut enchaîner les méthodes  
couleur.trim().toUpperCase();  
// "JAUNE"
```

**Bien que les Strings soient immutables, nous avons accès à des méthodes pour travailler avec.**

**Liste: MDN**

```
// format: string.method(args)
```

```
let msg = "baseball";
```

```
msg.indexOf("ball");
```

```
// 4
```

```
msg.slice(4);
```

```
// "ball"
```

```
msg.slice(2,5);
```

```
// "seb"
```


```
msg.replace("base", "basket");
```

```
// "basketball"
```

**Certaines méthodes  
prennent des arguments.**

**Liste: MDN**

## Caractères spéciaux

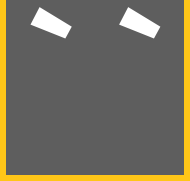
 est un caractère d'échappement.  
C'est donc un caractère qui déclenche  
une interprétation alternative du ou des  
caractères qui le suivent.


<code>\n</code>	Nouvelle ligne
<code>\'</code>	Guillemet simple, apostrophe
<code>\"</code>	Guillemet double
<code>\t</code>	Tab
<code>\uXXXX</code>	Symbole Unicode

```
let nom = "Ada";
```

```
// une variable encapsulée  
`Hello, ${nom}!`;  
// Hello, Ada!
```

```
// une expression encapsulée  
`Le résultat ${1 + 2}`;  
// le résultat est 3
```

Les backticks  sont des guillemets à fonctionnalité étendue. Ceux-ci créent des “template strings”.

Ils nous permettent d’intégrer des expressions ou des variables dans une chaîne de caractère en encapsulant celles-ci dans 

# Types primitifs

- ~~Number~~
- ~~Boolean~~
- ~~String~~
- **Null**
- **Undefined**

\* Le type “object” n’est pas considéré comme “primitif” car il sert à stocker des collections de données.



```
let vide = null;
```

```
let quelqueChose;  
// undefined
```

```
let moche = undefined;
```

**Le type “null” se réfère à une absence intentionnelle de valeur et doit être assignée.**

**Les variables qui n’ont pas de valeur assignée sont de type “undefined”.**

**N’affectez jamais manuellement “undefined” à une variable.**

```
typeof "PUP"; // "string"
```

```
typeof 2; // "number"
```

```
typeof NaN; // "number"
```

```
let x;
```

```
typeof x; // "undefined"
```

```
typeof null; // "object"
```