Les différentes écritures d'un nombre (e, 0x, 0o, 0b), les fonctions utiles sur les nombres (number) : parseInt, parseFloat, toString, toFixed, les arrondis, isNaN, isFinite, random, min, max, ...

Les différentes écritures d'un nombre

Les nombres en JavaScript sont implémentés comme des nombres sur 64 bits à précision double selon le format IEEE-754 (norme sur l'arithmétique) qui est un standard pour la représentation des nombres flottants et qui permet d'avoir jusqu'à 16 chiffres significatifs.
Le type numérique possède également trois valeurs spéciales symboliques : +Infinity, -Infinity et NaN (pour désigner une valeur qui n'est pas un nombre).

Le format IEEE-754 permet de représenter des valeurs entre ±2^−1022 et ±2^+1023, ce qui correspond à des valeurs entre ±10^−308 et ±10^+308 avec une précision sur 53 bits. Les nombres entiers compris sur l'intervalle ±2^53 − 1 peuvent être représentés exactement.

Le type BigInt est une addition récente à JavaScript qui permet de représenter de grands entiers. Toutefois, il n'est pas possible de mélanger les BigInt et les nombres (Number) dans les mêmes opérations et on ne peut pas utiliser l'objet Math avec les valeurs BigInt.

Il est possible d'utiliser quatre types de littéraux numériques : décimal, binaire, octal et hexadécimal.

1. Les nombres décimaux :

On voit ici que les littéraux numériques qui commencent par un zéro (0) et contiennent un chiffre strictement inférieur à 8 après ce 0 sont analysés comme étant exprimés en base octale.

```
1234567890;
42;

// Attention à l'utilisation des zéros en début de nombre

0888; // 888 analysé en base décimal
0777; // en mode non-strict, analysé en base octale, ce qui correspondd à 511 en base décimale
```

2. Les nombres binaires :

Pour utiliser des nombres binaires, on utilise un littéral qui commence par un 0 suivi d'un b minuscule ou majuscule (0b ou 0B). Si les chiffres qui suivent ce préfixe ne sont pas des 0 ou des 1, une exception SyntaxError sera levée.

```
var FLT_BITSIGNE = 0b10000000000000000000000000000000; // 2147483648
var FLT_EXPOSANT = 0b01111111100000000000000000000000; // 2139095040
var FLT_MANTISSE = 0B00000000000111111111111111111111; // 8388607
```

Valeur en binaire	Valeur en décimal	Valeur en hexadécimal
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
00000101	5	5
00000110	6	6
00000111	7	7
00001000	8	8
00001001	9	9
00001010	10	A

3. Les nombres octaux :

Pour utiliser des nombres en base octale, on utilisera un préfixe avec un 0. Si les nombres qui suivent ce 0 ne sont pas compris entre 0 et 7 (au sens strict), le nombre sera interprété comme un nombre décimal.

```
var n = 0755; // 493 en base 10
var m = 0644; // 420 en base 10
```

En mode strict, ECMAScript 5 interdit cette syntaxe octale. Cette syntaxe ne fait pas partie d'ECMAScript 5 mais est supportée par la majorité des navigateurs. Avec ECMAScript 2015 (ES6), il est possible de représenter un nombre en notation octale grâce au préfixe "0o" :

```
var a = 0o10; // Notation octale pour ES2015
```

4. Les nombres hexadécimaux :

Pour utiliser des nombres exprimés en base hexadécimale, on utilisera un préfixe avec un zéro suivi d'un x majuscule ou minuscule (0x ou 0X). Si les chiffres qui suivent ce préfixe ne sont pas 0123456789ABCDEF, cela provoquera une exception SyntaxError.

```
0xFFFFFFFFFFFFFFFFF;   // 295147905179352830000
0x123456789ABCDEF;     // 81985529216486900
0XA                    // 10
```

5. Notation scientifique :

```
1E3     // 100
2e6     // 2000000
0.1e2   // 10
```

L'objet number :

L'objet Number possède certaines propriétés représentant les constantes numériques telles que : la valeur maximale représentable en JavaScript, une valeur spéciale pour dire que la valeur numérique n'est pas un nombre et l'infini. Ces valeurs ne peuvent pas être modifiées, on pourra les utiliser de la façon suivante :

```
var plusGrandNombre = Number.MAX_VALUE;
var plusPetitNombre = Number.MIN_VALUE;
var infini = Number.POSITIVE_INFINITY;
var infiniNégatif = Number.NEGATIVE_INFINITY;
var pasUnNombre = Number.NaN;
```

On utilisera toujours ces valeurs directement avec l'objet natif Number (et non pas avec les propriétés d'une instance d'un objet Number qu'on aurait créé).

Le tableau qui suit liste certaines des propriétés de Number.

Propriétés de Number :

Propriété	Description
`Number.MAX_VALUE`	Le plus grand nombre qu'on peut représenter en JavaScript (`±1.7976931348623157e+308`).
`Number.MIN_VALUE`	Le plus petit nombre qu'on peut représenter en JavaScript (`±5e-324`).
`Number.NaN`	Une valeur spéciale signifiant que la valeur n'est pas un nombre.
`Number.NEGATIVE_INFINITY`	L'infini négatif, renvoyé lorsqu'on dépasse la valeur minimale.
`Number.POSITIVE_INFINITY`	L'infini positif, renvoyé lorsqu'on dépasse la valeur maximale.
`Number.EPSILON`	La différence entre 1 et la première valeur supérieure à 1 qui puisse être représentée comme `Number` . (`2.220446049250313e-16`)
`Number.MIN_SAFE_INTEGER`	Le plus petit entier qu'on puisse représenter en JavaScript. (−2^53 + 1 ou `-9007199254740991`)
`Number.MAX_SAFE_INTEGER`	L'entier le plus grand qu'on puisse représenter en JavaScript (+2^53 − 1 ou `+9007199254740991`)

Méthode	Description
`Number.parseFloat()`	Analyse un argument qui est une chaîne de caractères et renvoie un nombre décimal. Cette méthode est équivalente à la fonction `parseFloat()` .
`Number.parseInt()`	Analyse un argument qui est une chaîne de caractères et renvoie un entier exprimé dans une base donnée. Cette méthode est équivalente à la fonction `parseInt()` .
`Number.isFinite()`	Détermine si la valeur passée en argument est un nombre fini.
`Number.isInteger()`	Détermine si la valeur passée en argument est un nombre entier.
`Number.isNaN()`	Détermine si la valeur passée en argument est `NaN` . Cette version est plus robuste que la fonction globale `isNaN()` .
`Number.isSafeInteger()`	Détermine si la valeur fournie est un nombre qu'il est possible de représenter comme un entier sans perdre d'information.

Le prototype de Number fournit certaines méthodes pour exprimer les valeurs représentées par les objets Number dans différents formats. Le tableau suivant liste certaines de ces méthodes de Number.prototype.

L'objet Math :

L'objet natif Math possède des propriétés et des méthodes statiques pour représenter des constantes et des fonctions mathématiques. Ainsi, la propriété PI de l'objet Math représente la valeur de la constante \pi (3.141...), on peut l'utiliser dans les applications avec :

```
Math.PI
```

De la même façon, les fonctions mathématiques usuelles sont des méthodes de Math. On retrouve par exemple les fonctions trigonométriques, logarithmiques et exponentielles ainsi que d'autres fonctions. Si on souhaite utiliser la fonction sinus, on pourra écrire :

```
Math.sin(1.56)
```

Note : Les méthodes trigonométriques de Math prennent des arguments exprimés en radians.
Le tableau suivant liste les méthodes de l'objet Math.

À la différence des autres objets, on ne crée pas d'objet de type Math. Ses propriétés sont statiques, on les appelle donc toujours depuis l'objet Math.

Méthode	Description
abs()	Valeur absolue
sin(), cos(), tan()	Fonctions trigonométriques standards (les arguments sont exprimés en radians)
asin(), acos(), atan(), atan2()	Fonctions trigonométriques inverses (les valeurs renvoyées sont exprimées en radians)
sinh(), cosh(), tanh()	Fonctions trigonométriques hyperboliques (les arguments sont exprimés en radians)
asinh(), acosh(), atanh()	Fonctions trigonométriques hyperboliques inverses (les valeurs renvoyées sont exprimées en radians).
pow(), exp(), expm1(), log10(), log1p(), log2()	Fonctions exponentielles et logarithmiques
floor(), ceil()	Renvoie le plus petit/grand entier inférieur/supérieur ou égal à l'argument donné
min(), max()	Renvoie le plus petit (resp. grand) nombre d'une liste de nombres séparés par des virgules
random()	Renvoie un nombre aléatoire compris entre 0 et 1
round(), fround(), trunc(),	Fonctions d'arrondis et de troncature
sqrt(), cbrt(), hypot()	Racine carrée, cubique et racine carrée de la somme des carrés des arguments
sign()	Renvoie le signe d'un nombre et indique si la valeur est négative, positive ou nulle
clz32(), imul()	Le nombre de zéros qui commencent un nombre sur 32 bits en représentation binaire. La résultat de la multiplication de deux arguments sur 32 bits effectuée comme en C.

La fonction parseInt() analyse la chaîne de caractère fournie et le converti en nombre entier.

```
let string = "10";
let parse = parseInt(string);                    //10
console.log(typeof(parse));                      //number
```

La fonction parseFloat() transforme la chaîne de caractères fournis en nombre flottant (avec virgule).

```
let string = "10.56";
let parse = parseFloat(string);                   //10.56
console.log(typeof(parse));                      //number
```

La méthode toString() renvoie une chaîne de caractères représentant le tableau ou l'objet.

```
let array = [1, 2, 'a', '1a'];
console.log(array);                        // (4) [1, 2, 'a', '1a']
console.log(array.toString());             // 1,2,a,1a
```

La méthode **toFixed()** converti un « nombre » en « string » en arrondissant après la virgule, au nombre de chiffre qui lui est indiqué :

```
let nbr = 123.456;

nbr.toFixed();                             //123
nbr.toFixed(1);                            //123.5 (arrondi)
nbr.toFixed(3);                            //123.456
nbr.toFixed(5);                            //123.45600 (des 0 sont ajoutés)
console.log(typeof(nbr.toFixed()));        //string
```

Il existe une différence importante entre **Math.round()** et **toFixed()** lorsqu'on veut arrondir un nombre. La fonction **toFixed()**, contrairement à **Math.round,** ne se contente pas seulement d'arrondir un chiffre mais le convertis également en chaîne de caractère.

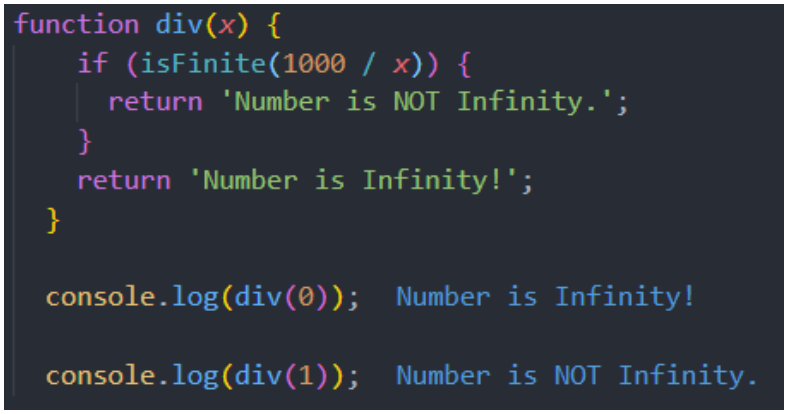
La méthode **isNaN**() permet de définir si la valeur est un nombre

```
function milliseconds(x) {
    if (isNaN(x)) {
      return 'Not a Number!';
    }
    return x * 1000;
}

console.log(milliseconds('100F'));  Not a Number!

console.log(milliseconds('0.0314E+2'));  3140
```

La méthode **isFinite**() détermine si la valeur passée en argument est un nombre fini. Si nécessaire, le paramètre est d'abord converti en nombre.



```
function div(x) {
    if (isFinite(1000 / x)) {
      return 'Number is NOT Infinity.';
    }
    return 'Number is Infinity!';
  }

  console.log(div(0));  Number is Infinity!

  console.log(div(1));  Number is NOT Infinity.
```