Les fondamentaux : Variables, Opérateurs et Types

En javascript, comme tout les langages de programmation, il existe 3 notions fondamentales : Types, Opérateurs et variables

Les Types

Un « type », c'est comme une grosse famille contenant certaines données. Parmi ces familles et existe 2 branches :

les types primitifs

Voici les différentes primitif :

1. `number` (représente n'importe quelle nombre)
2. `boolean` (représente une données vrai ou fausse)
3. `string` (représente n'importe quel texte)
4. `undefined` (représente une valeur non définie)
5. `null` (représente une valeur définie, mais qui ne contient rien)

Les types complexe

1. `array` (ce sont des tableaux)
2. `objets` (ce sont des groupement de données)
3. `function` (ce sont des boites de codes réutilisable)

Les numbers

Représente l'intégralité des nombres imaginable :

```
125 // Nombre 125
125.36
65.8 - 85 - 85.63
```

> **Pour faire un nombre à virgule il faut utiliser "."**

> **Pour faire un nombre négatif il est obligatoire de « coller » le signe moins au nombre**

Il existe 2 `number` un peu particulier :

1. L'infinie : `Infinite` (`-Inifinite`)
2. Un opération qui échoue : `NaN` (`Not A Number`), c'est le nombre retourné lorsque l'on fait une division par `0`.

Les opérateurs sur les nombres :

On peut réaliser des opérations sur les nombres :

```
52 + 12 // Addition
12 - 5 // soustraction
12 * 2 // Multiplication
53 / 2 // Division
53 % 3 // Reste d'une division euclidienne
10 ** 2 // Puissance
10++ // Incrémentation avec retour initiale (Ajoute 1 à 10, 10 + 1)
++10 // Incrémentation avec retour finale (Ajoute 1 à 10)
10-- // Décrémentation avec retour initiale (Enlève 1 à 10, 10 - 1)
--10 // Décrémentation avec retour finale (Enlève 1 à 10)
```

Les booleans

Les « booleans », sont inventé par un mathématicien Anglais : « James Bool ». C'est une algèbre basé sur 2 valeurs : Le vrai et le faux. C'est les fondations de l'informatique !

En informatique, certaines données peuvent « vrai » ou « fausse » :

```
true // vrai (1)
false // faux (0)
```

Les opérateurs logique

Les opérateur logique nous permette de « combiner » des valeurs vrai ou fausse :

```
true && false // ET (false)
true && true // ET (vrai)
false && true // ET (false)

true || false // OU (vrai)
true || true // OU (vrai)
false || true // OU (vrai)
false || false // OU (faux)

!true // NON VRAI (false)
!false // NON FAUX (vrai)
```

> **Le non (`!`) doit être collé au boolean pour fonctionner !**

Les opérateurs de comparaisons

Les opérateurs de comparaisons permettent de comparées des données entre elle :

```
10 > 12 // Supérieur (false)
10 < 12 // Inférieur (true)
10 >= 10 // Supérieur ou égale (true)
10 <= 12 // Inférieur ou égale (true)
10 == 10 // Égale à (true)
10 === 10 // Identique à (true)
10 != 10 // N"est pas égale (false)
10 !== 10 // N'est pas identique à (false)
```

Les strings

Les string (chaînes de caractères) sont un type de données permettant de représenter du texte. Pour représenter du texte, il faut utiliser les caractères suivants :

- `""` les guillemets double
- `''` les guillemets simple
- ``` `` Les backtick (alt gr - 7)

Exemple

```
'Coucou les amis'
"J'ai 19 ans"`Je suis aussi du texte`
```

> Attention, les guillemets (double ou simple) ne fonctionne que sur 1 seule ligne !

```
"coucou
les
amis" // Erreur impossible de faire plusieurs ligne !
```

```
`Coucou
les
amis` // Fonctionne très bien !
```

Égalité vs Identité

Attention, nous avons vue 2 opérateurs de comparaison : l'égalité (`==`) et l'identité (`===`) :

```
10 == '10' // true (égalité)
10 === '10' // false (identité, égalité + même type)
```

Les opérateurs sur les string

Il est possible « d'assembler » plusieurs texte ensemble, de les réunir. On appel ce phénomène : `concaténation`

```
'Coucou' + ' les amis' // 'Coucou les amis'
'Coucou' + 'les amis' // "Coucou les amis"
'Coucou' + ' ' + 'les amis' // 'Coucou les amis'
```

Le `null` et `undefined`

Le `null` et `undefined` sont généralement utilisé par javascript pour symboliser des valeur vide ou non définie. On les utilises très rarement, cependant nous verrons que nous pouvons souvent les rencontrer.

Elles s'écrivent comme le nom l'indique :

```
null // null
undefined // undefined
```

La coercion

Il éxiste un petit opérateur : `?` permettant de choisir la valeur non null ou undefined d'un couple :

```
null ?? 'coucou' // "coucou"
undefined ?? 'salut' // "salut"
'coucou' ?? null // "coucou"
'salut' ?? undefined // "salut"
```

Les priorités

Il est possible de définir des opérations plus ou moins prioritaire sur d'autre, pour cela, comme en mathématique, on utilise les `()` :

```
1 + 3 - 4 // 0
1 + (3 - 4) // 0
2 * 3 + 5 // 11
true || (false && true) // true
```

Les array

Les array (tableaux) représente une liste de valeur. Il est possible de regroupé des données dans une liste numéroté, ce sont les tableaux (array).

Pour délimiter un tableaux on utilise `[]`, ce tableaux peut contenir les valeurs de notre choix, ces dernières doivent simplement être séparé par une `,` :

```
[10, 12, 9, 20]
['coucou', true, 123, 'les amis']
['salut', true, 125454, 'les amis']
```

Les array (tableaux) sont des liste numéroté. C'est à dire que chaque élément de la liste possède une position dans la liste que l'on appel un `index`. Cependant, en informatique on commence à compté à partir de `0`

Nous pouvons demander un éléments à une position données d'un tableaux :

```
[10, 12, 9, 20][2] // 9
['coucou', true, 123, 'les amis'][3] // Les amis
['coucou', true, 123, 'les amis'][10] // undefined
```

Nous pouvons aussi connaître la taille d'une liste :

```
[10, 11, 9, 20].length // 4
```

> Le point et length doivent être attaché ! C'est une propriété.

Les objets

Les objets, ce sont des dictionnaires à valeur. On peut y ranger des mots (des clefs `keys`) auquel on attache des définitions (valeurs `value`). Ces objets sont délimité par `{}`

Chaque clef et valeurs (`keys: value`) sont séparé par le caractère `:`.

```
{
  nom: 'Dupont',
  prenom: 'Jean',
  age: 25,
  notes: [12, 8, 10, 8]
}
```

Pour accéder à une définissions (une valeur), on utilise le `.` et le nom de sa clefs :

```
{
  nom: 'Dupont',
  prenom: 'Jean',
  age: 25,
  notes: [12, 8, 10, 8]
}.age // 25
```

On peut aussi utiliser une syntaxe alternative, similaire à celle des tableaux :

```
{
  nom: 'Dupont',
  prenom: 'Jean',
  age: 25,
  notes: [12, 8, 10, 8]
}['age'] // 25
```

Attention, il existe des règles bien spécifique pour les clefs de nos objets. Nous ne pouvons pas mettre de caractères spéciaux :

```
{
  prénom: 'Jean' // NON, Erreur le é est un caractère spéciale !
}
```

On ne met que des chiffres ou des lettres !

Les clefs dynamique

Il est possible de créer des clefs avec des chaînes de caractères. Dans ce cas, nous pouvons mettre des caractères spéciaux :

```
{
  "prénom": 'Jean' // Valide !
}['prénom']
```

Les variables

Les variables se sont des petits identifiants contenant des valeurs que notre ordinateur enregistre en mémoire.

Le phénomène d'enregistrer cet identifiant et cette valeur est nommé la **déclaration**.

Une fois la déclaration effectué, nous pouvons utiliser la valeur contenue dans cette variable en utilisant son identifiant. L'ordinateur se souviendra de ce qu'il contient :

Le déclaration

Javascript est une langage qui a beaucoup évolué, nous avons 3 façons différentes de déclarer des variables :

- En utilisant le mot clefs `var` : Permet de déclarer une variable, mais hélas souffre de gros problèmes de performance. Il est aujourd'hui presque « bannie » de votre code.
- En utilisant le mot clefs `let` : Permet de déclarer une variable mais cette fois bien plus optimisé !
- En utilisant le mot clefs `const` : Permet de déclarer une variable qui ne peut être changé !

Exemple

```
var nom = 'Dupont' // Non, ici on utilise var
let prenom = 'Jean' // Préféré
const age = 23 // Le plus souvent

age = 56 // Erreur ! J'ai une constante elle ne peux pas changer
prenom = 'Gerome' // Ça passe, je n'ai pas de constante

// Plein de ligne de codes ....

nom + ' ' + prenom // "Dupont Jean"

14 + age // 37

const notes = [12, 9, 17, 20]

// Plus tard dans le code ...
notes.length // 4
notes[2] // 17
notes[3] // 20
notes[notes.length - 1] // 20

const eleve = {
  nom: 'Dupont',
  prenom: 'Jean',
  age: 32,
  notes: [12, 8, 16, 14],
  profPrincipal: {
```

```
    nom: 'Dupont',
    prenom: 'Jeanne',
    age: 52,
  },
}

// Plus tard dans le code

eleve.nom + ' ' + eleve.prenom // 'Dupont Jean'
eleve.notes[3] // 14
eleve.notes[eleve.notes.length - 1] // 14

eleve.profPrincipal.age // 52
```

Interpolation

Lorsqu'on utilise des chaînes de caractères, il est possible d'utiliser une autre technique que la `concatenation` pour les assembler, c'est : `L'interpolation` :

Pour faire une interpolation, il faut tout d'abord utiliser les backtick : `` `et placer les variables entre ${} ` ``

```
const nom = 'Dupont'
const prenom = 'Jean'
const age = 25

const nomComplet = nom + ' ' + prenom // concatenation
const nomComplet2 = `${nom} ${prenom}` // interpolation !

'Bonjour ' + nom + ' ' + prenom + ', vous avez ' + age + ' ans'

`Bonjour ${nom} ${prenom}, vous aves ${age} ans` // interpolation
```

Destructuration et Restructuration

Lorsque l'on travail avec des objets, ou des tableaux, nous pouvons utiliser une technique très puissante pour extraire son contenue : **les destructuration**

```
const notes = [12, 8, 9, 5]

const premierNote = notes[0] // Valide, 12

const [premierNote2] = notes // Même principe, mais en destructurant

premierNote2 // 12

const [firstNote, secondNote, ...restDesNotes] = notes

firstNote // 12
secondNote // 8
restDesNotes // [9, 5]
```

Ça marche aussi avec les objets

```
const eleve = {
  nom: 'Dupont',
  prenom: 'Jean',
  age: 25,
}

const { nom, prenom } = eleve

nom // "Dupont"
prenom // "Jean"
```

Il est possible de « fusionner » des tableaux et des objets en utilisant la **restructuration** :

```
const notes = [12, 14, 5]
const notes2 = [8, 18, 16]

// J'utilise la restructuration pour assembler les 2 tableaux
const notes3 = [...notes, ...notes2] // [12, 14, 5, 8, 18, 16]
const notes3 = [notes, notes2] // [[12, 14, 5], [8, 18, 16]]

const inscrit = {
  nom: 'Dupont',
  prenom: 'Jean',
}

const eleve = {
  notes: [12, 15, 16],
  classe: '3eme 6',
}

const eleveFinal = { ...inscrit, ...eleve } // {nom: ..., prenom: ..., notes:
...n, classe: ...}
```

Quelques petits bonus et règles à suivre

Le nommage : camel case

Lorsque l'on doit nommer des variables, des clefs d'objets, on utilise le camel case : Ça consiste à écrire plusieurs mot séparé par des majuscules :

```
const monPrenom = 'Jean'
const maPremierNote = 15
```

Attention, nous ne mettons pas de majuscule au début !

```
const eleve = {
  sonNom: 'Dupont',
  sonPrenom: 'Jean',
}
```

L'indentation

Il est très important pour la lisibilité de respecter ce qu'on appel l'indentation. C'est à dire, d'utiliser une tabulation pour symboliser un « niveaux » :

```
// Sans indentation (mauvaise pratique)
const eleve = {
sonNom: 'Dupont',
          sonPrenom: 'Jean'
}

// Avec indentation
const eleve = {
  sonNom: 'Dupont',
  sonPrenom: 'Jean'
}
```

Les indentations utilisent la touche tabulation, attention à bien respecter le bon niveau !

En javascript, généralement les tabulation ou indentation correspondent à 2 espaces (certains langage en utilise plutôt 4)

Les commentaires

Il est possible de « commenter » son code, se sont des lignes qui sont ignoré par le compilateur javascript qui servent simplement à documenter / expliquer votre code :

```
// Voici un commentaire javascript sur une seule ligne

/*
Voici
un commentaire javascript
sur
plusieurs ligne
*/

/**
 * Voici une documentation
 * Ce petit commentaire peut être
 * « extrait » pour créer des sites internet
 * documentant automatiquement votre code.
 *
 * Attention, ils doivent se mettre au dessus d'une
 * « déclaration »
 *
 * Ici l'objectif est de fournir de la documentation
 * sur la constante age
 */
const age = 56
```