5 minutes

Les types en JavaScript

Qu'est-ce qu'un type ?

Un type est un ensemble de structures de données natives à un langage.

Comme nous l'avons vu, en `JavaScript` les types sont **faibles et dynamiques** ce qui signifie qu'il n'est pas nécessaire de déclarer le type et qu'il peut changer au cours du temps.

En `JavaScript` il y a deux ensembles de types : les **primitifs** et les **objets**.

Nous verrons que les deux grandes différences entre ces ensembles sont :

Premièrement, que les primitifs sont passés par valeur alors que les objets sont passés par référence.

Deuxièmement, que les primitifs sont comparés par valeur alors que les objets sont comparés par référence.

Nous étudierons en détails ces différences dans une leçon spécifique.

Les types primitifs

Les types primitifs sont au nombre de 7 : les booléens, les nombres, les chaînes de caractère, les symboles, `null`, `undefined` et `BigInt`.

Les booléens (`boolean`)

Les booléens ont seulement deux valeurs `false` ou `true`.

Un booléen représente une assertion logique. Ils permettent notamment de raisonner facilement pour les conditions et les boucles.

Grâce à eux, on peut écrire facilement, si c'est vrai exécute un bloc sinon un autre bloc :

```
if (maVar === true) {
  // Exécute ce bloc
} else {
  // Sinon exécute ce bloc
}
```

Copier

Nous verrons bien sûr en détails les blocs et les conditions.

Une instruction `JavaScript` permet de connaître le type d'un élément ou d'une variable :

```
typeof true; // "boolean"
const booleen = false;
typeof booleen; // "boolean"
```

Copier

Les nombres (`number`)

En `JavaScript` il y a deux types pour les nombres : les `number` et les `BigInt`.

Le format par défaut des nombres est `number` en `JavaScript`, ainsi :

```
typeof 1; // "number"
typeof 0.1; // "number"
```

Copier

Il permet d'écrire des nombres décimaux et des entiers.

Le format `number` permet d'écrire sans perte de précision des nombres jusqu'à `2^53` comme nous le verrons en détails dans le chapitre sur les nombres.

Pour les nombres décimaux nous avons également une perte de précision par arrondi, comme nous l'expliquerons.

Les `BigInt` (`bigint`)

Les `BigInt` sont une nouveauté en `JavaScript` et sont dans les spécifications du langage `ECMAScript 2020`.

Il permet de manipuler les nombres supérieurs à `2^53`.

Les `BigInt` ne peuvent pas être utilisés avec des `number` :

```
BigInt(123456789012345678)+1; // TypeError: Cannot mix BigInt and other types, use explicit conversions
```

Copier

Pour déclarer un `BigInt` on peut soit faire :

```
BigInt(123456789012345678);
```

Copier

Soit terminer par un `n` le nombre :

```
123456789012345678n
```

Copier

Les `BigInt` ne produise pas de nombre décimaux dans leurs opérations, la partie décimale sera toujours tronquée :

```
5n/2n // 2n
1n/3n // 0n
```

Copier

Avec `BigInt` vous pouvez additionner, soustraire et multiplier des entiers sans perte de précision :

```
9007199254740991n + 2n // 9007199254740993n
```

Copier

Cela règle totalement les problèmes pour les larges entiers en `JavaScript`.

Il reste le problème des décimaux, selon `Google` cela sera réglé dans les années à venir en introduisant un nouveau type `BigDecimal` grâce aux fondements de `BigInt`.

Les chaînes de caractère (`string`)

Ce type permet de manier les données textuelles en `JavaScript`.

Il se déclare en utilisant les simples ou les doubles guillemets ou encore l'accent grave :

```
'test';
"test";
`test`;
typeof `test`; // "string"
```

Copier

Nous verrons que si l'utilisation de guillemets simples ou doubles ne fait aucune différence, l'utilisation de l'accent grave a un effet sur la chaîne de caractères.

Le type `null`

Ce type a une seule valeur : `null` qui représente l'absence, le fait de ne pas avoir de valeur.

```
const a = null;
console.log(a); // null
```

Copier

Signifie que la constance `a` n'a pas de valeur, nous le spécifions explicitement.

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

Copier

Pour le coup, il s'agit bien d'un bug bien connu du langage `JavaScript`. Le type devrait être `null` et non `object`.

Ce bug n'est pas réparable car il a été introduit lors de la création du langage et tout le `Web` a donc ce problème. Les travaux entrepris pour y remédier par le `TC39` ont été abandonnés définitivement car cela pose de trop grands problèmes de compatibilité.

Cela n'a aucune incidence pour les développeurs, mais il fallait une explication à cette bizarrerie !

Le type `undefined`

Ce type a une seule valeur : `undefined` qui représente le fait qu'une valeur n'est pas définie. C'est la valeur par défaut en `JavaScript` :

```
let a;
console.log(a); // undefined
typeof undefined; "undefined"
```

Copier

La différence entre `null` et `undefined` est que `null` est une valeur ne pouvant être assignée qu'explicitement alors qu'`undefined` est une valeur par défaut attribuée par l'interpréteur pour indiquer l'absence de valeur.

Autrement dit, si vous pouvez dans votre programme assigner comme valeur au choix `null` ou `undefined`, l'interpréteur `JavaScript` ne pourra assigner "tout seul" que `undefined`.

Il est ainsi recommandée d'assigner toujours `null` quand vous voulez indiquer l'absence de valeur car c'est vous le faites explicitement.

Ne vous inquiétez pas, nous verrons de nombreux exemples dans la formation.

Les symboles

Un symbole est le type primitif représentant une **donnée unique et inchangeable**.

```
const symbol = Symbol();
typeof symbol; // "symbol"
```

Copier

Ils sont très peu utilisés en `JavaScript`, ils servent à éviter les collisions de noms dans les clés des propriétés d'objets.

Les objets

Un objet est une collection de propriétés.

Il y a de nombreux objets natifs en `JavaScript`.

Voici une liste non exhaustive : les objets littéraux, les tableaux, les tableaux typés (11 types différents comme `Int16Array`), les `Map`, les `Set`, les `WeakMaps`, les `WeakSets`, les fonctions, les dates, les données structurées (le format `JSON` par exemple), les tampons de données binaires (`ArrayBuffer` par exemple), les proxies (`Proxy` et `Reflect`), les objets de contrôle d'abstraction (`Promise`, `Generator`, `AsyncFunction` principalement).

Vous n'avez probablement jamais entendu parlé de la plupart de ces objets et c'est normal : `JavaScript` étant réputé comme un langage de "débutant" la plupart des développeurs ne font pas l'effort de l'apprendre et il est en fait méconnu.

Or c'est une très grande erreur d'appréciation : `JavaScript` est aujourd'hui un langage mature, à la fois riche et complexe qui demande du temps pour l'apprendre convenablement. Cet apprentissage est nécessaire pour devenir un expert du langage.

Nous vous enseignerons tous ces objets au fur et à mesure de la formation.