Javascript ES 6 > Variables

> Pour tester rapidement du JS dans votre navigateur : touche F12 > onglet console.

Les types de variables

JavaScript utilise 5 types de données :

- Booléen : les mots `true` pour vrai et `false` pour faux
- Objet :
- `null` : mot réservé qui ne représente pas de valeur. La valeur null est affectée volontairement à une variable.
- `undefined` : mot réservé qui est renvoyé par une variable référencée qui n'a pas encore été définie (la variable existe mais n'a reçu aucune affectation de valeur).

Type	Définition	Exemple
Chaîne	Toute suite de caractères comprise entre guillemets ou quotes telle que "suite de caractères" ou 'suite de caractères'.	`var lastname = "LOPER"; var phrase = "A vaincre sans péril, on triomphe sans gloire.";`
Nombre (entier, décimal etc.)	Tout nombre entier ou avec une virgule/un point tout entier en octal ou hexadécimal.	Entier : `var age = 42` Décimal : `3.1416` Hexadécimal : `0xFFA8`
Booléen	Les mots `true` pour vrai et `false` pour faux.	`true`, `false`
Tableau	Un tableau est une variable qui contient une liminte d'éléments individuels (chaînes ou nombre) ou d'autres tableaux (on parle alors de sous-tableaux). Il existe plusieurs syntaxe pour déclarer un tableau, que nous aborderons dans le cours sur les tableaux.	`var afruits = ["Pomme", "Poire", "Fraise"]; var ages = [12, 18, 24, 42];`
Objet	Toute utilisation de variable par référence vers tout objet natif JavaScript (Array, Date, String etc.) ou tout objet du DOM.	`var o = new Person(); o.lastname = "LOPER"; o.firstname = "Dave" o.age = 42;`

type primitifs / objet

Le typage

Une variable peut changer de type après une affectation. On peut vérifier le type en cours d'une variable avec l'instruction `typeof` :

```
console.log(typeof 42); // Affiche "number"
console.log(typeof "LOPER"); // Affiche "string"
console.log(typeof true); // Affiche "boolean"
```

Déclaration de variable avec `const`

`const` permet de déclarer une variable à assignation unique : **la variable ne peut être assignée qu'une seule fois, lors de sa déclaration.**

Exemple

```
const iAge = 42;

console.log(iAge); // Affiche 42
```

`const` requiert une assignation de valeur lors de la déclaration. Si vous écrivez :

```
const iAge;
iAge = 42;
console.log(iAge);
```

vous obtiendrez le message d'erreur *Uncaught SyntaxError: Missing initializer in const declaration*.

Si vous tentiez de réassigner/modifier la valeur assignée :

```
const iAge = 42;
iAge++;
console.log(iAge);
```

vous aurez le message suivant : *Uncaught TypeError: Assignment to constant variable.*

Déclaration avec `let`

Avec `let`, une déclaration peut se faire indépendamment de l'assignation et la valeur peut être modifiée/réassignée :

Exemple

```
let iAge; // Declaration

iAge = 42; // Assignation independante de la déclaration

console.log(iAge); // Affiche 42

iAge++; // Modification de la valeur (on ajoute 1)

console.log(iAge); // Affiche 43
```

Hoisting

Le hoisting ne se comporte pas de la même manière selon que l'on déclare une variable avec `var`, `let` ou `const`.

Hoisting avec `var`

Le moteur d'interprétation de Javascript procède à une première lecture du code avant son exécution, et à ce stade remonte - hisse - les déclarations de variables et de fonctions au début de script : c'est le *hoisting*, ou hissage en français.

Exemple

Quand vous écrivez ce code :

```
var sPrenom = "Dave";

console.log("Bonjour "+sPrenom);

var sNom = "Loper";

bonjour(sNom);

function bonjour(sNom)
{
   console.log("Ca farte "+sNom+" ?");
}

var iAge = prompt("Quel est ton âge ?");
```

Javascript remonte les déclarations en haut du script : **seules les déclarations sont hissées, PAS les assignations de valeurs**.

```
/* Déclarations de variable et de fonctions remontées
* en début de script par le moteur d'interprétation
*/
var sPrenom;
var sNom;
var iAge;

function bonjour(sNom)
{
   console.log("Ca farte "+sNom+" ?");
}

/* Exécution du code */
console.log("Bonjour "+sPrenom);

bonjour(sNom);

var iAge = prompt("Quel est ton âge ?");
```

Hoisting avec `let` et `const`

C'est très simple : avec `const` et `let`, il n'y a pas de hoisting !

Exemple

```
const sNom = "Loper";

let sPrenom = "Dave";

bonjour(sPrenom+" "+sNom+ "!");

function bonjour(sNom)
{
   console.log("Ca farte "+sNom+" ?");
}

let iAge = prompt("Quel est ton âge ?");
```

L'interprétation par le moteur Javascript sera la suivante : seule la fonction `bonjour()` est hissée en tête du code :

```
function bonjour(sNom)
{
   console.log("Ca farte "+sNom+" ?");
}

const sNom = "Loper";

let sPrenom = "Dave";

bonjour(sPrenom+" "+sNom+ "!");

let iAge = prompt("Quel est ton âge ?");
```

La portée des variables

Selon que l'on utilise `var`, `const` ou `let`, la portée de la variable ne sera pas la même. Nous aborderons ce point en détail dans le cours suivant sur les fonctions.

Les erreurs `undefined`, `not defined` etc.

`undefined`

L'erreur `undefined` se produit lorsqu'une variable a bien été déclarée mais n'a pas reçu d'assignation de valeur.

Exemple

```
var age;
console.log(age); // Affiche 'undefined'
```

Il aurait fallu écrire :

```
var age = 42;
console.log(age); // Affiche '42'
```

`not defined`

L'erreur `not defined` se produit lorsqu'une variable n'a jamais été déclarée, ou ne se trouve pas dans le scope (portée) du bloc de code exécuté (la notion de scope sera abordée dans le cours suivant sur les fonctions).

Exemple

```
console.log(lastname); /* Affiche 'not defined' car la variable 'lastname' n'a jamais été déclarée dans le code
```

`null`

`null` est une valeur affectée par le développeur de façon volontaire lorsqu'il souhaite qu'une variable n'ait plus de valeur.

Exemple

```
var age = 42;
console.log(age); // Affiche '42'

/* On ne veut plus que la variable 'age' ait une valeur,
* on lui assigne la valeur 'null'
*/
age = null;
console.log(age); // Affiche 'null'
```

Conclusion : que faut-il utiliser pour déclarer une variable ?

Comment choisir quelle déclaration de variable utiliser ?

Les bonnes pratiques constatées dans le monde du Javascript sont les suivantes :

1. Ne plus utiliser `var` (R.I.P.). Cependant, vous risquez de continuer à croiser `var` dans des projets existants, il est donc important d'en connaître le fonctionnement (hoisting etc.).
2. Ecrivez toute déclaration de variable avec `const`.
3. Si votre programme requiert de changer/réassigner une valeur, remplacez alors `const` par `let`.

> Le respect des points 2 et 3 vous permettra de distinguer au premier coup d'oeil les variables dont la valeur est modifiée.

Exercices

Exercice 1

A partir du code suivant (peut importe ce qu'il fait), écrivez ce que voit l'interpréteur Javascript, sachant que les déclarations de fonctions sont elles aussi hissées.

```
const iLongueurRectangle = parseInt(prompt("Longueur du rectangle ?"));

function AireRectangle(iLongueur, iLargeur)
{
   let iAire = iLongueur * iLargeur;
   console.log("Surface du rectangle : "+iAire);
}

let iLargeurRectangle = parseInt(prompt("Largeur du rectangle ?"));

AireRectangle(iLongueurRectangle, iLargeurRectangle);

var iRayonCercle = parseInt(prompt("Rayon du cercle ?"));

AireCercle(iRayonCercle);

function AireCercle(iRayon)
{
   var iSurface = Math.PI * Math.pow(iRayon, 2);
   iSurface++;
   console.log("Surface du cercle : "+iSurface);
}
```

Exercice 2

A partir du code d'origine de l'exercice 1, réécrire les déclarations de variables en tenant compte des principes vus dans ce cours.

Corrigés des exercices

Corrigés

Sources

- bases
- https://putaindecode.io/articles/es6-es2015-la-declaration-de-variables-avec-const-let-et-var