Les éléments du langage

Tout langage de programmation se compose de ces différents éléments: les variables, les tests, les boucles. Nous allons plonger dans la base de la programmation !

Variables, types et valeurs

Déclaration et initialisation d'une variable

Vous vous souvenez de votre cours de math ? On y parlait déjà de variables, par exemple dans le calcul suivant, on trouve 3 variables: x, a et b.

x = (a + b)/4

On **assigne** des valeurs à a et b afin de trouver la valeur de x.

C'est le même principe en informatique, une variable permet de stocker une valeur, une information, qu'elle soit numérique, sous forme de texte, ou autre.

Afin d'utiliser une variable il faut d'abord la **déclarer**. Il s'agit tout simplement de lui réserver un espace de stockage en mémoire, rien de plus. Une fois la variable déclarée, vous pouvez commencer à y stocker des données sans problème.

Pour déclarer une variable, il suffit d'écrire par exemple:

```js
var myVariable;
```

Notez la présence d'un point-virgule. Après chaque instruction, il faut mettre un point-virgule. C'est comme ça ☺

le JavaScript est sensible à la casse ou *case-sensitive*, ce qui veut dire que les majuscules et minuscules comptent.

```
var myVariable;
var MyVariable;
```

sont 2 variables différentes !

On peut nommer une variable comme on le souhaite, cependant il existe certains noms réservés qu'on ne peut pas utiliser, sinon le code ne marchera pas. La liste complète des mots réservés

Une fois une variable déclarée on peut **l'initialiser**, c'est-à-dire lui assigner une valeur, par exemple

```js
var myVariable;
myVariable = 2;
```

Généralement, les variables sont déclarées au tout début du bloc de code. Cela permet de traduire le comportement par défaut du JavaScript qui est de placer toutes les déclarations de variables tout en haut du bloc de code où elles se trouvent. Cela veut dire que si on initialise une variable avant de la déclarer, ça va quand même marcher ! On appelle ça le « *hoisting* ».

Contrairement à de nombreux langages, le JavaScript est un langage typé *dynamiquement*. Cela veut dire, généralement, que toute déclaration de variable se fait avec le mot-clé `var` sans distinction du contenu, tandis que dans d'autres langages, comme le Java, il est nécessaire de préciser quel type de contenu la variable va devoir contenir (un nombre, du texte, etc...)

Cela veut aussi dire que l'on peut y mettre du texte en premier lieu puis l'effacer et y mettre un nombre quel qu'il soit, et ce, sans contraintes. Il faut donc être très attentif !

Les types en JavaScript peuvent être divisés en 2 catégories : les types *primitifs* (nombres, textes, booléens, null et undefined) et les types *objets* (toute autre valeur qui n'est pas de type primitif)

Types primitifs

*Numérique (*Number*)*

Ce type représente tout nombre, qu'il soit entier, négatif, décimal,…

Pour assigner un type numérique à une variable, il vous suffit juste d'écrire le nombre seul :

```
var number = 5;
```

Attention, dans le cas d'un nombre décimal, il faut utiliser le . et non la virgule, par exemple:

```
var number = 5.5;
```

Quelle est l'utilité ? De pouvoir faire des calculs par exemple ! Et à cette fin, vous pouvez utiliser les opérateurs arithmétiques que vous connaissez tous comme +, -, /, *.
Mais également, moins connu, l'opérateur %, qui permet de calculer le modulo (*le reste de la division entière*). A quoi sert le modulo ? A vérifier si un nombre est pair par exemple, ou multiple d'un chiffre.

*Chaîne de caractère (*String*)*

Ce type représente n'importe quel texte. On peut l'assigner de deux façons différentes :

```
var text1 = "Mon premier texte"; // Avec des guillemets
var text2 = 'Mon deuxième texte'; // Avec des apostrophes
```

Il est important de préciser que si vous écrivez `var myVariable = '2';` alors le type de cette variable est une chaîne de caractères et non pas un type numérique.

Une autre précision importante, si vous utilisez les apostrophes pour « encadrer » votre texte et que vous souhaitez utiliser des apostrophes dans ce même texte, il vous faudra alors « échapper » vos apostrophes de cette façon :

```
var text = 'Ça c\'est quelque chose !';
```

Pourquoi ? Car si vous n'échappez pas votre apostrophe, le JavaScript croira que votre texte s'arrête à l'apostrophe contenue dans le mot « c'est ». Du coup, une erreur sera générée et la suite de votre script ne s'exécutera pas.

À noter que ce problème est identique pour les guillemets. Une autre possibilité est d'encadrer le texte avec des guillemets quand vous utilisez des apostrophes dedans:

```
var text = "Ça c'est quelque chose !";
```

Une opération intéressante avec le texte s'appelle la **concaténation**, elle permet de « coller » 2 bouts de texte ensemble pour créer un nouveau texte. Il suffit d'utiliser l'opérateur + entre 2 textes :

```
var text = "Bonjour" + " " + "Madame";
//text vaudra "Bonjour Madame"
```

*Booléens (*Boolean*)*

Ce type permet simplement de stocker "vrai" ou "faux" :

```
var isTrue = true;
var isFalse = false;
```

Quelle est l'utilité ? De pouvoir vérifier des conditions par exemple.

Sachez qu'il est également possible de "convertir" une variable d'un type à l'autre, par exemple de transformer une string qui contient "42" en une variable numérique qui contient le nombre 42 et inversement.

null et undefined

Il arrive qu'on doive dire qu'une variable est déclarée mais **ne contient rien**. Dans ce cas, on utilisera *null*.

Si une variable **n'a pas été initialisée** ou simplement pas déclarée, elle sera *undefined.* Evidemment, si vous exécuter une action avec une variable undefined, le code va planter.

Exemples de *undefined* et de *null.*
Ce test appelle *console.log()*, pensez) activer Firebug ;-)

http://jsfiddle.net/xa4zs/

On peut bien sûr tester si une variable est *null* ou *undefined*

Plus sur les types dits primitifs :
https://msdn.microsoft.com/en-us/library/ie/7wkd9z69%28v=vs.94%29.aspx

Types objets

JavaScript possède d'autres types natifs que ceux vus précédemment. Dans le cas des types primitifs, des nombres, textes ou booléens littéraux sont généralement utilisés – comme dans les exemples vu précédemment.

Dans le cas des types objets, nous devons passer par l'initialisation d'un objet. Un objet est un type de données qui permet de stocker des propriétés (des variables propres à l'objet) qu'on pourrait imaginer comme étant des **caractéristiques** – par exemple, une télévision a plusieurs caractéristiques : la taille de l'écran, la résolution, la couleur,…
Nous le verrons plus tard, un objet peut aussi posséder des **comportements**, appelées méthodes; dans le cas d'une télévision : allumer, éteindre, changer de chaine, augmenter le volume,…

Un objet s'initialise ou *s'instancie* à l'aide de ce qu'on appelle un **constructeur** et via le mot-clé *new* suivi du « type » de l'objet.

Le type Date

Le constructeur Date() permet de créer des objets qui représentent des dates et des heures. Dans l'exemple ci-dessous on génère une date d'expiration :

```
//quand on instancie une nouvelle Date, elle sera à la date
courante
var date = new Date();
//on lui rajoute le nombre de jours en millisecondes
date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
```

Important à savoir, JavaScript stocke les dates en millisecondes. Mais heureusement pour nous, le constructeur *Date* peut prendre en charge plusieurs paramètres pour nous permettre de construire une date, mais retenez surtout ces deux-cis :

```
new Date();
new Date(année, mois, jour [, heure, minutes, secondes,
millisecondes ]);
```

- La première ligne instancie un objet *Date* dont la date est fixée à celle de l'instant même de l'instanciation.
- La deuxième ligne permet de spécifier manuellement chaque composant qui constitue une date, nous retrouvons donc en paramètres obligatoires : l'année, le mois et le jour. Les quatre derniers paramètres sont facultatifs (c'est pour cela que vous voyez des crochets, ils signifient que les paramètres sont facultatifs). Ils seront initialisés à 0 s'ils ne sont pas spécifiés, nous y retrouvons : les heures, les minutes, les secondes et les millisecondes.

Une fois un objet Date instancié, vous pouvez utiliser ces méthodes :

- *getFullYear()* : renvoie l'année sur 4 chiffres ;
- *getMonth()* : renvoie le mois (0 à 11) ;
- *getDate()* : renvoie le jour du mois (1 à 31) ;
- *getDay() :* renvoie le jour de la semaine (0 à 6, la semaine commence le dimanche) ;
- *getHours()* : renvoie l'heure (0 à 23) ;
- *getMinutes()* : renvoie les minutes (0 à 59) ;
- *getSeconds()* : renvoie les secondes (0 à 59) ;
- *getMilliseconds()* : renvoie les millisecondes (0 à 999).

Exemples de calcul de dates, avec jQuery et *datepicker*

http://jsfiddle.net/fkKMv/

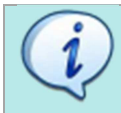
Moment.js, une librairie JavaScript incontournable pour travailler avec les dates : http://momentjs.com/

Le type Array

Pour créer des tableaux (liste de valeurs) : Voir ce chapitre

Le type RegExp

Pour travailler avec des expressions régulières afin d'effectuer des recherches ou remplacements performants dans du texte. Les expressions régulières ne sont pas abordées dans ce cours.



Plus sur les expressions régulières avec *RegExp* :
http://www.w3schools.com/js/js_obj_regexp.asp

Le type Math

Contrairement aux autres types objets, *Math* n'est pas un constructeur mais un objet global à partir duquel on peut effectuer des opérations mathématiques comme : arrondir à la valeur supérieure ou inférieure, calculer des exposants, générer des nombres aléatoires, obtenir le nombre Pi, etc…

Plus sur *Math* :
http://www.w3schools.com/js/js_obj_math.asp

Transformations de types

Imaginez le cas où un utilisateur renseigne un montant dans un champ texte et vous voulez calculer le montant TVA comprise, vous serez bien obligé de transformer le texte rentré dans le champ par l'utilisateur en valeur numérique.

Il est bien sûr possible de transformer une valeur d'un certain type en un autre type, par exemple de convertir un texte qui représente un nombre en nombre. Ou bien l'inverse.

Dans ce cas, vous pouvez effectuer une conversion explicite en utilisant les fonctions *Number()*, *String()* ou *Boolean()*

```
Number("3"); // => 3
String(false) ; // => "false"
Boolean(0); // => false
```

Pour convertir des nombres en texte, il est également possible d'utiliser plusieurs méthodes de Number() : *.toFixed()* pour préciser un nombre de chiffres après la virgule, *.toPrecision()* pour préciser le nombre de chiffres au total. Mais aussi *.toString()* et *.toExponential()*.

```
var n = 3.7514615;
console.log(n.toFixed(2)); // 3.75
console.log(n.toPrecision(5)); // 3.7515
```

Plus sur les méthodes de conversion de *Number()* :
http://www.w3schools.com/jsref/jsref_obj_number.asp

Pour convertir du texte en valeur numérique, il est également possible d'utiliser les méthodes globales *parseInt()* pour transformer vers un entier ou *parseFloat()* pour entier ou décimal. Ces méthodes sont plus flexibles que *Number()* mais sont donc aussi moins fiables si vous chercher une conversion « stricte »

```
parseInt("10 burgers"); // => 10
parseFloat("34.12"); // => 34.12
```

Article super intéressant sur la conversion de texte en nombre en JavaScript : http://www.js-attitude.fr/2012/12/26/convertir-un-nombre-en-texte-en-javascript/

En dehors de ces conversions explicites, il existe des conversions implicites. En effet, JavaScript est très flexible sur ce point : si JavaScript veut du texte, il va convertir ce que vous lui donnez en texte ; s'il veut un nombre, il va convertir ce que vous voulez en nombre. Cela peut aller très très loin, voici juste quelques exemples :

```
10 + " burgers"; // => "10 burgers"
"7" * "4";// => 28
+"10" ; // => 10
```

Un « style » de programmation complètement cryptique qui se base sur les conversions implicites en JS : http://www.jsfuck.com/