1: Les bases

Introduction à la Programmation et à JavaScript

Langage de programmation

JavaScript est un langage qui permet de donner des ordres (appelés *instructions*) à une machine.

La machine en question peut être:

- un navigateur web
- un ordinateur
- un serveur sur internet
- un objet intelligent (ex: Arduino, Raspberry Pi...)
- voire même une carte à puce ! (ex: carte SIM, carte bleue...)

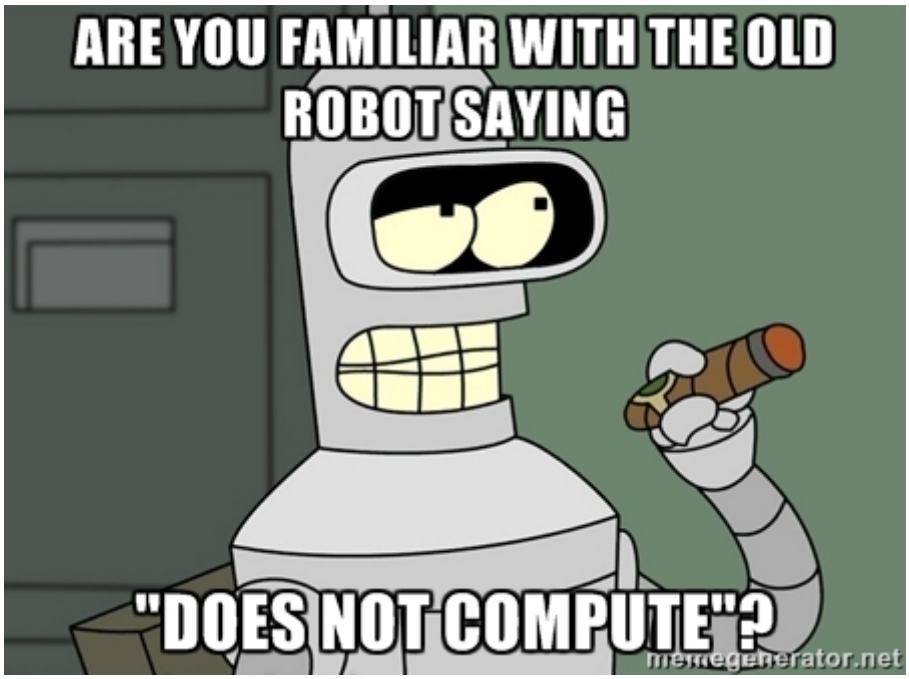
On va donc apprendre des commandes et structures pour expliquer ce qu'on attend de cette machine.

Qu'est-ce qu'un programme ?

Le code source d'un programme est un peu comme une recette de cuisine: on y définit une liste d'actions (les *instructions*) qui permettent à quiconque de reproduire le plat, autant de fois qu'il le veut.

Comme dans une recette, si votre code source est trop vague ou contient des erreurs, le résultat risque d'être décevant, voire complètement à côté de la plaque.

Contrairement à une recette de cuisine, il faut garder en tête que ce n'est pas un humain qui va interpréter votre code source, mais une machine. Sachant que, contrairement à un humain, une machine n'est pas capable de raisonner intuitivement, il faut être extrêmement rigoureux dans le respect des règles de grammaire et d'orthographe du langage.



Si vous écrivez votre code de manière approximative, deux choses peuvent se passer:

- si vous avez de la chance, la machine vous avertira qu'elle n'a pas compris une de vos instructions => elle affichera un message d'erreur pour vous aider à la corriger;
- soit, dans certains cas, il ne se passera rien de visible. Dans ce cas, ce sera à vous de relire votre code, et de vous mettre à la place de la machine pour essayer de comprendre comment elle l'interprète.

Exécution de code JavaScript

Il y a deux façons d'exécuter nos instructions JavaScript:

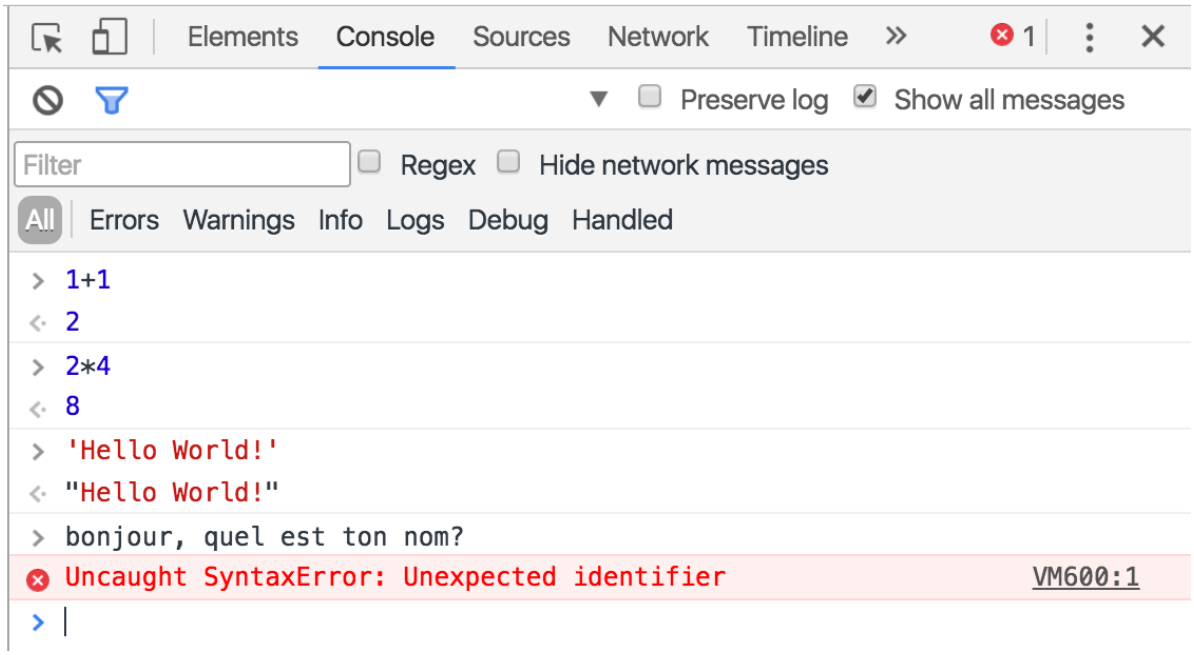
1. de manière interactive: via une console
2. de manière programmative: en rédigeant un code source

Dans le premier cas, chaque instruction sera exécutée immédiatement puis retournera un résultat, comme sur une calculatrice.

Dans le deuxième cas, on va écrire une liste d'instructions dans un fichier (appelé "code source"), pour que notre machine puisse exécuter ce fichier en une seule fois.

Pour tester JavaScript en mode interactif:

- Ouvrez une fenêtre de navigation privée (mode incognito) depuis Google Chrome,
- ouvrez la console JavaScript en utilisant le raccourci clavier `Cmd` + `Alt` + `J` (sur Mac) ou `Ctrl` + `Shift` + `J` (sur PC/Window)
- tapez `1+1` et validez. => La console devrait afficher le résultat de cette opération. Ensuite tapez `alert('bonjour');` et observez ce qui se passe.



> Il existe de nombreuses consoles JavaScript plus ou moins évoluées: repl.it, glot.io, jsbin, jsfiddle, et codepen permettent d'éditer son code de manière un peu plus confortable mais ne conservent pas d'historique de vos instructions, contrairement à la console de Google Chrome.

Manipulation de valeurs

Types de valeurs

En langue Française, il existe plusieurs types de mots: les noms, les verbes, les adjectifs, les prénoms, etc...

De la même façon, en maths, on peut manipuler plusieurs types de valeurs: nombres entiers, décimaux, rationnels (fractions), complexes, etc...

En JavaScript, c'est pareil. Il est possible d'exprimer et de manipuler des valeurs de différents types.

Types simples

- booléen (*boolean*): `true`, `false`
- nombre (*integer/float number*): `999`, `0.12`, `-9.99`
- chaîne de caractères (*string*): `'coucou'`
- aucune valeur: `null`

Types avancés

- non défini: `undefined`
- objet (*object*): `{ prop: 'valeur' }`
- tableau (*array*): `[ 1, 2, 3 ]`
- fonction (*function*): `function(){ /* ... */ }`

Variables et opérateur d'affectation

Variables et valeurs littérales

Dans l'exercice ci-dessus, nous avons manipulé des valeurs de manière **littérale**. C'est à dire qu'elles étaient explicitement affichées dans le code.

Dans un véritable programme, les valeurs dépendent souvent d'une saisie de l'utilisateur, ou d'autre chose. Du coup, elles sont rarement exprimées de manière littérale. On utilise pour ça une représentation symbolique: **les variables**.

En maths, on représente habituellement une variable sous forme d'une lettre minuscule. Par exemple "*soit x=4*" veut dire qu'on définit une variable appelée `x` représente actuellement la valeur `4`. (un nombre entier, en l'occurrence)

En JavaScript, une variable représentée par une suite de lettres (minuscules et/ou majuscules) pouvant contenir aussi des chiffres et le symbole *underscore* (`_`).

Généralement, on emploie une notation appelée **camel case** pour nommer les variables en JavaScript. Cette notation consiste à coller plusieurs mots, en mettant en majuscule seulement la première lettre de chaque mot, sauf celle du premier mot.

Exemple: `nombreSaisiParUtilisateur` est un bon nom pour une variable JavaScript.

> *Camel* veut dire "chameau", en Français. Avez-vous compris pourquoi ?

Opérateur d'affectation

Comme en maths, l'opérateur d'affectation `=` permet d'affecter une valeur (à droite) à une variable (à gauche).

> Par exemple, si on tape `monNombre = 1` dans une console JavaScript, puis qu'on y tape `monNombre`, la console nous répondra `1` car c'est la valeur actuelle de la variable `monNombre`.

Chaque usage de l'opérateur d'affectation sur une variable **changera sa valeur actuelle**, quitte à remplacer la valeur qu'elle portait précédemment.

> Par exemple, si on tape `monNombre = 1`, `monNombre = 2` puis `monNombre`, la console JavaScript nous répondra `2` car c'est la dernière valeur qui a été affectée à la variable `monNombre`.

La valeur affectée (à droite du `=`) peut être une valeur **littérale**, ou celle d'une **autre variable**.

> Par exemple, si on tape `autreNombre = monNombre`, puis `autreNombre`, la console JavaScript nous répondra `2` car nous avons affecté la valeur de la variable `monNombre` (`2`, tel qu'affecté dans l'exemple précédent) à notre nouvelle variable `autreNombre`.

À noter que l'opérateur d'affectation se comporte différemment selon que la valeur affectée est de **type simple** (nombre, chaîne de caractères...) ou **avancé** (objet, tableau, fonction...):

- l'affectation d'une valeur de type simple **dupliquera** cette valeur dans la variable affectée;
- alors qu'en affectant une valeur de type avancé, notre variable affectée sera en fait une **référence** à cette valeur.

Exemple d'affectation de valeur de type simple:

```
maValeurSimple = 1;
autreValeurSimple = maValeurSimple;
// à ce stade, nos deux variables valent 1
maValeurSimple = 2;
// => autreValeurSimple vaut toujours 1, alors que maValeurSimple vaut 2
```

Exemple d'affectation de valeur de type avancé:

```
monTableau = [ 1, 2, 3 ];
autreVariable = monTableau;
// à ce stade, nos deux variables valent [ 1, 2, 3 ]
monTableau = [ 4, 5, 6 ];
// cette instruction a affecté un autre tableau à monTableau
// => autreValeurSimple vaut toujours [ 1, 2, 3 ], car il référence toujours le
tableau qui lui avait été affecté
```

Exemple d'affectation de valeur de type avancé, **avec modification**:

```
monTableau = [ 9, 8, 7 ];
autreVariable = monTableau;
// à ce stade, nos deux variables valent [ 9, 8, 7 ]
monTableau.sort();
// cette instruction a rangé notre tableau dans l'ordre croissant
// => monTableau vaut désormais [ 7, 8, 9 ]
// => ... et autreVariable vaut aussi [ 7, 8, 9 ], car la tableau qu'il référencé
a été modifié.
```

Création de variable

Dans nos exemples, les variables ont été créées automatiquement par la console JavaScript lors de leur première affectation de valeur.

Mais nous allons désormais créer nos variables de manière explicite, à l'aide du **mot-clé** `var`.

En effet, l'usage de `var` permet de s'assurer qu'on a pas déjà créé une variable du même nom.

Il est possible **d'affecter une valeur** à notre variable lors de sa création avec `var`.

Il restera évidemment possible d'affecter d'autres valeurs à notre variable, à l'aide de l'opérateur d'affectation, mais sans utiliser `var` ensuite.

Exemple:

```
// on va crééer une variable de type chaîne de caractères:
var maVariable = 'mon texte';
// => maVariable vaut 'mon texte'
maVariable = `mon nouveau texte`;
// => maVariable vaut désormais 'mon nouveau texte'
```

Vous remarquerez que:

- j'ai ajouté des **espaces** autour de l'opérateur d'affectation pour améliorer la lisibilité de mon code;
- et j'ai terminé mes instructions par un **point-virgule** (`;`).

Afin d'améliorer la lisibilité de votre code, et d'éviter tout ambiguïté pouvant occasionner des erreurs, je compte sur vous pour appliquer ces règles.