



Tester l'existence et le contenu de variables

<http://jsfiddle.net/ZZbh5>

Les boucles

Quand on parle de répéter un album en boucle, ça vous évoque quoi ? Le fait de répéter plusieurs fois, automatiquement, les chansons de l'album. C'est le même principe en programmation. Les boucles permettent de répéter une action plusieurs fois tant qu'une condition est respectée, ou jusqu'à ce qu'une condition soit respectée.

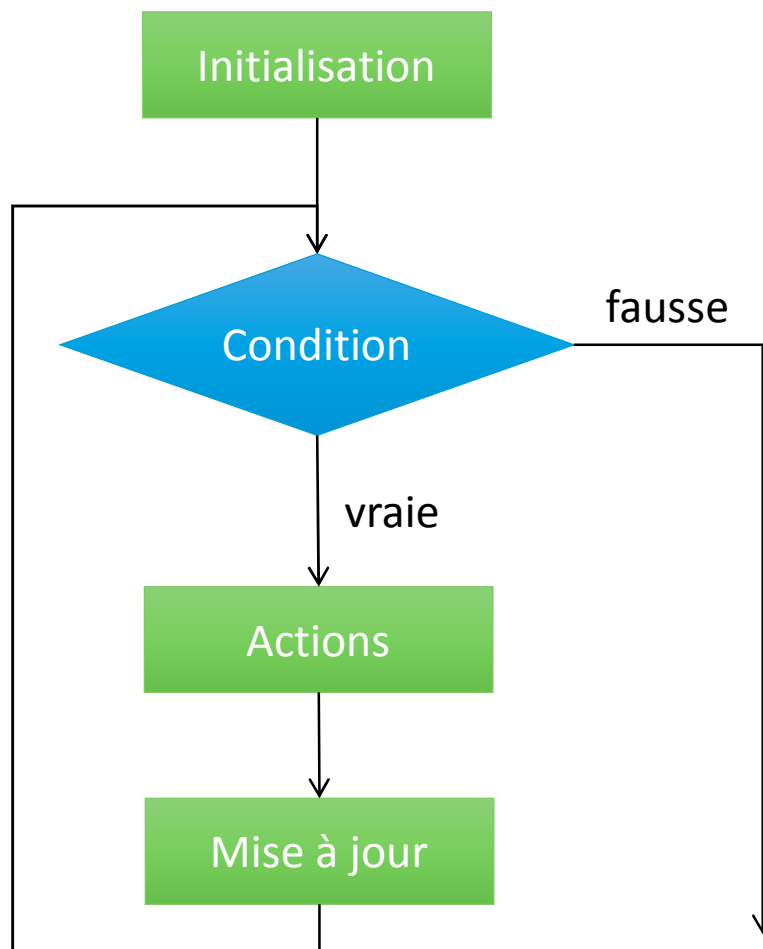
Et oui, nous allons, comme pour les tests, utiliser les conditions dans les boucles. Il faut, bien entendu, qu'un moment la condition devienne fausse, sinon on génère une boucle infinie... vous vous rappelez des sites qui ouvraient milles fenêtres popup ? ☺ C'est un beau cas d'utilisation d'une boucle infinie, une boucle qui ne s'arrête jamais ! Haaaaaaa !

Une boucle peut être exprimée de cette façon « **Tant que** la condition est vraie, exécuter ces actions » ou bien de cette façon « **Pour** toutes les données que voici, exécuter ces actions ».

Une boucle fonctionne de cette manière :

1. On aura généralement besoin d'une variable de boucle, qu'on va initialiser avant la boucle avec une valeur qui **ne va pas rencontrer la condition** de la boucle (sinon la boucle ne s'enclencherait pas !).
2. A chaque passage de boucle ou **itération**, on vérifie si la condition est toujours vraie, si c'est le cas on va rentrer dans la boucle, c'est-à-dire exécuter les actions dans la boucle. Si la condition est fausse, on sort de la boucle et on continue à exécuter le code suivant.
3. On doit, au sein de la boucle, à la fin de toutes les actions, mettre à jour la variable de boucle pour que, à un moment, sa valeur ne corresponde plus à la condition et que la boucle s'arrête.
4. On retourne au point 2

Ce flux est illustré dans le schéma suivant :



L'incrémentation

Avant de plonger dans les types de boucles, arrêtons-nous un instant car c'est l'endroit idéal pour parler de l'incrémentation. Qu'est-ce que l'incrémentation ? C'est le fait d'ajouter une unité à un nombre au moyen d'une syntaxe courte. À l'inverse, la décrémentation permet de soustraire une unité.

Par exemple, pour incrémenter une variable numérique on peut le faire de cette manière

JS

```
var incrementVariable = 0;  
incrementVariable = incrementVariable + 1;
```

Mais on peut l'écrire de manière plus courte avec l'opérateur ++ (ou -- pour une décrémentation)

JS

```
var incrementVariable = 0;  
incrementVariable++;
```

Vous allez comprendre l'intérêt de l'incrémentation dans les exemples des boucles qui utilisent des *Array*.

Le type array

Bon, les boucles utilisent très régulièrement un autre type de données qu'on appelle *array*, ou tableaux en Français. Pour ne pas confondre avec les tables HTML, on va garder le terme *array*!

Là où les variables de type numérique ou string ne contiennent qu'une seule valeur, un *array* en anglais, est une variable qui contient plusieurs valeurs, appelées **items**. Chaque item est accessible au moyen d'un **indice** (*index* en anglais) et dont la numérotation commence **à partir de 0**. Voici un schéma représentant un tableau, qui stocke cinq items :

Index	0	1	2	3	4
Item	« HTML 5 »	« CSS 3 »	« Photoshop »	« JavaScript »	« jQuery »

Un *array* contenant 5 éléments aura donc un *index* maximal de 4, c'est-à-dire « nombre d'éléments – 1 ».

En JavaScript, on écrirait l'*array* ci-dessus de cette manière.

JS

```
var skills = ['HTML 5', 'CSS 3', 'Photoshop', 'JavaScript',  
             'jQuery'];
```

Et pour récupérer et modifier des valeurs, il faut préciser le numéro de l'*index* en crochets :

JS

```
skills[1] = 'PHP'; //Modifier  
console.log(skills[1]); //Accéder
```

Types de boucles

while

C'est celle qui colle le plus au flux décrit ci-dessus. Voici sa syntaxe :

```
while (condition) {  
    action1;  
    action2;  
}
```

Exemple :



Tant qu'on n'atteint pas un *div* défini d'une collection de *div*, on rajoute du code HTML aux *div* parcourus

<http://jsfiddle.net/g8XJx/>

Une existe une variante de la boucle *while*, la boucle *do while*. Elle ressemble très fortement à la boucle *while*, sauf que dans ce cas la boucle est toujours exécutée au moins une fois. Dans le cas d'une boucle *while*, si la condition n'est pas valide, la boucle n'est pas exécutée. Avec *do while*, la boucle est exécutée une première fois, puis la condition est testée pour savoir si la boucle doit continuer.

Cependant vous allez rarement l'utiliser, mais il faut savoir la reconnaître. Sa syntaxe :

```
do {  
    action;  
} while (condition);
```

for

La boucle que vous allez utiliser le plus souvent est la boucle *for*. Elle fonctionne de la même manière qu'un *while* mais l'initialisation, la condition et l'incréméntation se déclarent sur une seule ligne, comme ceci :

```
for (initialisation; condition; incréméntation) {  
    action;  
}
```

1. Dans le premier bloc, *initialisation*, on initialise une variable, généralement à 0.
2. On définit dans *condition*, la condition d'arrêt de la boucle, afin de ne pas créer la fameuse boucle infinie. Généralement on dit que la boucle continue tant que la variable initialisée est plus petite qu'une valeur définie.

3. Enfin, dans le bloc d'*incrémentation*, on indique que la variable sera **incrémentée** à chaque passage de la boucle, afin de la faire arriver à la valeur d'arrêt

Voici un exemple qui ne sert à rien mais qui aide à comprendre comment ça s'écrit

```
JS for (var iter = 0; iter < 5; iter++) {  
    alert('Itération n°' + iter);  
}
```

On peut aussi partir « à l'envers » en décrémentant jusque 0.

Priorité d'exécution


Les trois blocs qui constituent la boucle `for` ne sont pas exécutés en même temps :


- *Initialisation* : juste avant que la boucle ne démarre. C'est comme si les instructions d'initialisation avaient été écrites juste avant la boucle
- *Condition* : avant chaque passage de boucle
- *Incrémentation* : après chaque passage de boucle. Cela veut dire que, si vous faites un *break* dans une boucle *for*, le passage dans la boucle lors du *break* ne sera pas comptabilisé.

Un *break* permet de « casser » l'exécution d'une boucle et donc d'en sortir directement. L'instruction *continue*, quant à elle permet de mettre fin à une itération, mais attention, elle ne provoque pas la fin de la boucle : l'itération en cours est stoppée, et la boucle passe à l'itération suivante. Mais vous l'utiliserez rarement !

Le fonctionnement même du JavaScript fait que la boucle *for* est nécessaire dans la majorité des cas comme la manipulation des *array*. Nous verrons aussi un peu plus tard une variante de la boucle *for*, appelée *for in*.

Exemples :

 Adapter la hauteur de 3 *div* en fonction du plus haut des *div*
<http://jsfiddle.net/WZ6ac/>

 Créer des *span* sous forme de « tags » à partir d'un *array*
<http://jsfiddle.net/EvBmX/>