Tests et conditions

Les tests permettent d'exécuter une action (ou de ne pas l'exécuter !) en fonction de certains critères, ou *conditions*. Les tests s'appliquent toujours au type booléen, c'est-à-dire à "vrai" ou "faux". Un test est toujours du type "**Si** la condition est vraie, **alors** j'exécute ceci, **sinon** j'exécute cela"

Par exemple, « si l'utilisateur est belge, alors afficher un message à l'écran » ou « si l'utilisateur arrive en bas de la page, afficher un message »

Exemple :

Tester si l'utilisateur vient pour la première fois sur le site
http://www.ravelrumba.com/blog/firstimpression-js-library-detecting-new-visitors/

Détecter le pays de l'utilisateur et afficher un message approprié

http://jsfiddle.net/9hNp4/

De quoi sont constituées les conditions ? De **valeurs à tester** et de deux types **d'opérateurs** : de logique et de comparaison. Pour être plus clair: une condition sera vraie ou fausse en fonction du résultat d'une comparaison et/ou d'un test logique.

Les opérateurs de comparaison

Ces opérateurs vont permettre de comparer diverses valeurs entre elles. En tout, ils sont au nombre de huit, les voici :

Opérateur	Signification
==	égal à
!=	différent de
===	contenu **et** type égal à *(à privilégier)*
!==	contenu ou type différent de *(à privilégier)*
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

Le résultat d'une comparaison renvoie **vrai** ou **faux**, en bref, une valeur de type **booléen** ;-)

Exemple:

```
var hour = getHour();
if(hour === 12){
  alert('le dîner est prêt !');
}
```

Les opérateurs logiques

Pourquoi ces opérateurs sont-ils nommés comme étant « logiques » ? Car ils fonctionnent sur le même principe qu'une table de vérité en électronique. Avant de décrire leur fonctionnement, il nous faut d'abord les lister, ils sont au nombre de trois :

Opérateur	Type de logique	Utilisation
&&	ET	valeur1 && valeur2
||	OU	valeur1 || valeur2
!	NON	!valeur

Lorsque qu'on teste plusieurs valeurs, il faut avoir en tête les résultats possibles, voici la table de vérité des opérateurs "ET" et "OU". Les résultats d'un test logique est aussi un **booléen**

valeur1	Opérateur	valeur2	Résultat
Faux	**ET**	Faux	Faux
Faux	**ET**	Vrai	Faux
Vrai	**ET**	Faux	Faux
Vrai	**ET**	Vrai	Vrai
Faux	**OU**	Faux	Faux
Faux	**OU**	Vrai	Vrai
Vrai	**OU**	Faux	Vrai
Vrai	**OU**	Vrai	Vrai

L'opérateur **!** quant à lui renvoie l'inverse d'une valeur booléenne. Si une variable est « vraie », son inverse sera « faux ».

Du coup, il est possible de combiner au sein d'un même test, des opérateurs de comparaison et des opérateurs logiques, comme dans l'exemple ci-dessous

```
var hour = getHour();
var name = getUsername();
if(hour === 12 && name === 'Michel'){
  alert('le dîner est prêt pour Michel !');
}
```

Types de tests

if-else

Nous l'avons déjà vue dans les exemples précédents, retenez-le bien car c'est le type de test le plus rencontré. Il permet d'exécuter une action si une condition est vraie, et une autre action si elle est fausse.

Sa syntaxe se présente sous cette forme

```
if (test) {
        action;
}
else {
        autreAction;
}
```

Il est possible d'enchaîner les tests if-else, par exemple pour dire, "**si** le visiteur est Belge, **alors** afficher un message vert, **sinon si** le visiteur est Français, **alors** afficher un message bleu, **sinon** afficher un message jaune"

La syntaxe sera telle que:

```
if(test) {
        action;
}
else if (test2) {
        autreAction;
}
else {
        actionParDéfaut;
}
```

switch

C'est une variante du if-else. Si on doit tester plusieurs pays avec des if-else, ça risque de vite devenir ennuyeux. Dans le cas où il faut faire beaucoup de tests sur une même donnée (ici, le code pays du visiteur), alors il est préférable d'utiliser un *switch* car il permet de mieux s'y retrouver. Cette instruction effectue une comparaison par rapport à la valeur **et au type**.

Sa syntaxe :

```
switch(variable) {
        case valeurPossible1 :
                action1;
                break ;
        case valeurPossible2 :
                action2;
                break ;
        case valeurPossible3 :
                action3;
                break ;
        default :
                actionParDéfaut;
               break ;
}
```

L'instruction *break* permet de « casser » le switch et d'en sortir dès que l'action adéquate a été exécutée. Si on ne mentionne pas le *break*, les autres cas vont également s'exécuter !

default permet lui de définir une action par défaut si aucune des valeurs des *case* ne correspond.

Détecter le pays de l'utilisateur avec un switch et afficher un message approprié

http://jsfiddle.net/zrUtK

ternaire

Une autre variante du if-else. Si vous voulez épargner vos doigts lors de l'écriture de if-else, vous pouvez passer par une condition ternaire. C'est la même chose en plus court, mais en parfois moins lisible... Préférez le if-else, mais sachez reconnaître celui-ci.

Syntaxe:

(*test*) **?** *action si test est vrai* **:** *action si test est faux*

ou

résultat **=** (*test*) **?** *valeur si test est vrai* **:** *valeur si test est faux*

Détecter le pays de l'utilisateur avec l'opérateur ternaire et afficher un message approprié

http://jsfiddle.net/Xns2Z/

Tester *null* et *undefined*

Pour vérifier l'existence d'une variable (qu'elle ne soit pas *undefined*) ou son contenu (qu'elle ne soit pas *null* ou vide), on peut le faire de la manière classique ci-dessous :

```
var notSet = null;
if(notSet === null)
{
      alert("la variable n'a pas de valeur")
}
```

```
var notInit;
if(notInit === undefined)
{
      alert("variable pas initialisée")
}
```

Du coup, pour tester *null* et *undefined* mais aussi par exemple une string vide, il faudrait combiner plusieurs conditions. Il existe une notation beaucoup plus concise !

Vous savez que les variables peuvent être de plusieurs types : les nombres, les chaînes de caractères, etc. En fait le type d'une variable (quel qu'il soit) peut être converti en booléen même si à la base on possède un nombre ou une chaîne de caractères.

Quels contenus seront « faux » ? *False* évidemment mais aussi un nombre qui vaut zéro ou qui vaut *NaN*, une string vide, *null* ou *undefined*. Tous les autres cas seront « vrais ». Du coup, on peut écrire quelque chose comme ça, qui est nettement plus rapide :

```
var notSet = null;
if(notSet)
{
      alert("la variable n'a pas de valeur")
}
```

Quand est-on dans le vrai en JavaScript ? http://www.js-attitude.fr/2012/09/10/truthy-ou-falsy-en-javascript/

Truth, Equality and JavaScript: https://javascriptweblog.wordpress.com/2011/02/07/truth-equality-and-javascript/

Une autre petite subtilité du JavaScript nous est offerte par l'opérateur **OU**. Celui-ci, en plus de sa fonction principale, permet de renvoyer la première variable possédant une valeur évaluée à *true*.

```
var conditionTest1 = '', conditionTest2 = 'Une chaîne de
caractères';

alert(conditionTest1 || conditionTest2);
//affichera 'Une chaîne de caractères'
```

Concrètement, à quoi cela peut-il servir ?
Par exemple, certains merveilleux navigateurs comme Internet Explorer ont des fonctionnalités qui sont les mêmes que dans le standard ECMA mais appelées différemment ! Chouette hein ? Hé bien cette notation permettra au navigateur de choisir la fonctionnalité qui existe pour lui.

Si vous êtes déjà curieux, voici un petit exemple : on souhaite récupérer le contenu texte d'un DIV grâce au DOM (nous verrons ceci en détail dans le chapitre ad hoc). Avec IE on doit utiliser « *innerText* » et pour les autres navigateurs avec le standard ECMA, « *textContent* ». Une manière concise de l'écrire serait :

```
var txt = div.textContent || div.innerText || '';
```