2: Conditions

Comparaisons de valeurs et conditions

Un programme est sensé pouvoir prendre des décisions de manière autonome, sur la base de conditions.

Pour exprimer ces conditions, on s'appuie généralement sur des comparaisons entre valeurs. Nous allons commencer par la comparaison la plus simple: **l'égalité de valeurs**.

Comparaison d'égalité: `==` et `===`

En JavaScript il existe deux opérateurs de comparaison d'égalité:

- `==` vérifie l'égalité de deux valeurs de manière laxiste;
- `===` vérifie l'égalité de deux valeurs de manière stricte.

Une **égalité laxiste** consiste à dire que deux valeurs sont vues comme équivalentes, mais pas exactement égales.

> Exemple: `1` (nombre) et `"1"` (chaîne de caractères) représentent tous les deux le chiffre `1`, mais sont de types différents => ils sont égaux seulement selon l'opérateur d'égalité laxiste.

L'**égalité stricte** vérifie en plus que le type des deux valeurs comparées est le même.

Quelques exemples:

```
1 == '1'; // => true
1 === `1`; // => false (car types différents)

0 == false; // => true
0 === false; // => false

null == undefined; // => true
null === undefined; // => false
```

La comparaison entre **valeurs de types avancés** fonctionne différemment. Ce ne sont pas les valeurs à proprement parler qui sont comparées, mais la référence vers cette valeur.

Explication:

```
[1,2] == [1,2]; // => false, car deux tableaux ont été créés

var monTab = [1,2];
monTab == monTab; // => true, car la variable monTab référence un seul et même tableau

monTab == [1,2]; // => false, car ce sont deux références de tableaux différents

var monTab2 = monTab;
monTab == monTab2; // => true, car monTab2 fait référence au même tableau que monTab
```

ATTENTION !

Notez que les opérateurs de comparaison d'égalité `==` et `===` ne doivent pas être confondus avec l'opérateur d'affectation `=`.

Vous verrez plus tard que se tromper d'opérateur peut causer des erreurs silencieuses qui peuvent mettre des heures à être détectées et corrigées dans votre code !

Donc soyez attentifs à bien les différencier quand vous écrivez votre code.

Opérateurs d'inégalité

Quand on sait que deux valeurs ne sont pas égales, notre programme peut s'adapter à différents cas de figure.

Nous avons vu que les opérateurs `===` et `==` permettaient d'évaluer l'égalité (stricte ou laxiste), en retournant une valeur `true` quand c'était le cas. Le langage JavaScript fournit aussi leurs **opérateurs contraires**: `!==` et `!=` (strict, et laxiste, respectivement). Ceux-ci retournent une valeur `true` quand les valeurs comparées ne sont pas égales.

Exemples:

```
1 == 1; // => true
1 === 1; // => true
1 == '1'; // => true
1 === '1'; // => false

1 != 1; // => false
1 !== 1; // => false
1 != '1'; // => false
1 !== '1'; // => true
```

Le langage JavaScript fournit aussi les opérateurs de comparaison suivants:

- strictement inférieur: `<`,
- strictement supérieur: `>`,
- inférieur ou égal: `<=`,
- supérieur ou égal: `>=`.

Exemples:

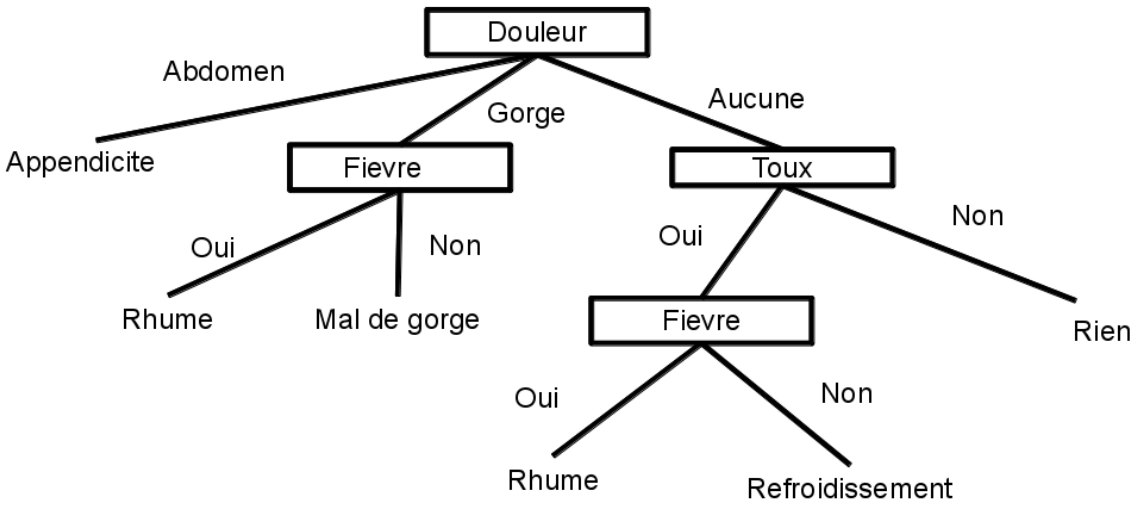
```
1 < 1; // => false
1 > 1; // => false
1 <= 1; // => true
1 >= 1; // => true

1 < 2; // => true
1 > 2; // => false
1 <= 2; // => true
1 >= 2; // => false
```

Conditions

Dans un programme (codé en langage JavaScript ou pas), les conditions sont une des instructions les plus incontournables.

C'est grâce aux conditions que votre programme peut prendre des **décisions** et donc d'effectuer des actions de manière autonome (ou automatique), en fonction des données qui lui sont fournies.

On peut représenter ces décisions sous forme d'un arbre:



Cet exemple d'arbre illustre un diagnostic médical, en fonction des symptômes.

Cet arbre pourrait être traduit en pseudo-code de la manière suivante:

- `si` *douleur à abdomen*, `alors` **appendicite**
- `sinon, si` *douleur à la gorge* `et`:
  - `si` *fièvre*, `alors` **rhume**
  - `sinon`, **mal de gorge**

    -
- `sinon, si`...

Puis être implémenté en langage JavaScript de la manière suivante:

```
// supposons que les variables douleur et fievre soient fournies
var diag;
if (douleur === 'abdomen') {
  diag = 'appendicite';
} else if (douleur === 'gorge') {
  if (fievre === true) {
    diag = 'rhume';
  } else {
    diag = 'mal de gorge';
  }
}
```

En JavaScript, les conditions s'expriment à l'aide des mots-clés `if` et `else`. On les emploie de la manière suivante:

```
if (expression) {
  // si expression == true, alors les instructions entre ces accolades vont
s'exécuter
} else {
  // sinon, ce sont les instructions entre ces accolades là qui vont s'exécuter
}
```

Le mot-clé `else` (facultatif) permet d'exécuter une séquence d'instructions seulement si `expression` n'est pas vraie.

Vous pouvez utiliser une comparaison en guise d'`expression`.

Exemple:

```
var monNombre = 1, resultat;
if (monNombre === 1) {
  resultat = 'monNombre vaut 1';
} else {
  resultat = 'monNombre ne vaut pas 1';
}
resultat;
// => cette liste d'instructions va afficher 'monNombre vaut 1'
```

> A retenir: dans un bloc `if-else`, soit les instructions entre la première paire d'accolades sera exécuté, soit celles de la deuxième paire d'accolades.
>
> Par ailleurs, observez bien la manière d'agencer les accolades et les espaces.

Condition avec multiples alternatives

Dans le cas où vous aimeriez définir plus de deux comportements alternatifs, vous pouvez employer ajouter des cas `else if()` entre votre bloc `if{}` et votre bloc `else{}`.

Exemple:

```
if (monNombre === 1) {
  resultat = 'monNombre vaut 1';
} else if (monNombre > 1) {
  resultat = 'monNombre est supérieur à 1';
} else {
  resultat = 'monNombre n\'est ni égal à 1, ni supérieur à 1';
}
resultat;
```

Comme pour les blocs `if-else`, seules les instructions d'**une** paire d'accolades seront exécutées.

Par contre, si vous écrivez plusieurs blocs `if-else` à la suite les uns des autres, ceux-ci seront complètement indépendants.

Instructions pour interagir avec l'utilisateur: `prompt` et `alert`

Maintenant qu'on a vu comment faire en sorte qu'un programme prenne des décisions, il faut qu'on soit capable d'interagir avec lui.

A ce stade, nous allons employer deux instructions pour cela:

- `alert` permet d'afficher un message à l'utilisateur;
- et `prompt` permet de lui demander de saisir une chaîne de caractères.

Par exemple, voici comment **afficher** `Bonjour !`:

```
alert('Bonjour !');
```

> Comme vous le voyez, il faut fournir le message à afficher entre parenthèses. Et, comme il s'agit ici d'une chaîne de caractères littérale, il ne faut pas oublier de mettre le texte entre apostrophes.

Il est aussi possible d'afficher la valeur d'une variable:

```
var monMessage = 'Hello !';
alert(monMessage); // usage d'une variable => pas d'apostrophes
```

Maintenant, voici comment **inviter l'utilisateur à saisir** une chaîne de caractères, puis l'afficher:

```
var sonPrenom = prompt('Quel est ton prénom ?');
alert('Bonjour, ' + sonPrenom + ' ! :-)');
```

> Êtes vous capable d'interpréter ce que signifie le code à l'intérieur des parenthèses du `alert` ? On appelle ça "la concaténation". On verra ça plus en détail dans les cours à venir.

Combinaison d'expressions conditionnelles `&&` et `||`

Dans l'exemple précédent, on a vu que chaque alternative dépendait du résultat d'une seule expression de comparaison de valeurs.

```
if (maPremiereValeur === maDeuxiemeValeur) { /* ... */
} else if (maPremiereValeur > maDeuxiemeValeur) { /* ... */ }
```

Dans certains cas, une alternative est définie par la combinaison de plusieurs expressions.

Par exemple, imaginons un programme qui propose comment s'habiller en fonction du temps qu'il fait, en suivant les critères suivants:

- S'il fait beau et chaud, suggérer de porter un short,
- S'il pleut ET qu'il fait chaud, suggérer de prendre un parapluie,
- S'il pleut ET qu'il fait froid, suggérer de prendre un manteau à capuche.

Sachant qu'il est possible d'imbriquer des conditions, on pourrait l'implémenter de la manière suivante:

```
if (temps === 'beau') {
  suggestion = 'short';
} else if (temps === 'pluie') {
  if (temperature >= 20) {
    suggestion = 'parapluie';
  } else if (temperature < 20) {
    suggestion = 'manteau à capuche';
  }
}
```

Mais il est aussi possible d'exprimer nos critères de manière plus linéaire, en combinant les conditions à l'aide de l'opérateur `&&`:

```
if (temps === 'beau') {
  suggestion = 'short';
} else if (temps === 'pluie' && temperature >= 20) {
  suggestion = 'parapluie';
} else if (temps === 'pluie' && temperature < 20) {
  suggestion = 'manteau à capuche';
}
```

Dans cette implémentation, nous avons combiné les critères de `temps` et de `temperature` dans une même expression conditionnelle.

Lorsqu'une alternative `if/else` est définie par plusieurs expressions liées par l'opérateur `&&` (appelé **et**), il faut que *toutes* ces expressions soient vraies afin que les instructions associées à cette alternative soient exécutées.

Il existe aussi un opérateur `||` (appelé **ou**) qui permet de définir des alternatives qui seront exécutées si *au moins une* des expressions est vraie.

Exemple:

```
if (rdvPrevu === true || envieDallerAuxToilettes === true) {
  message = 'excusez-moi, je vais devoir vous laisser';
} else {
  message = 'nous pouvons en discuter tout de suite, si vous voulez !';
}
```

... ce qui pourrait aussi s'écrire ainsi, sous sa forme développée:

```
if (rdvPrevu === true) {
  message = 'excusez-moi, je vais devoir vous laisser';
} else if (envieDallerAuxToilettes === true) {
  message = 'excusez-moi, je vais devoir vous laisser';
} else {
  message = 'nous pouvons en discuter tout de suite, si vous voulez !';
}
```

Dans ce cas, nous voyons que la combinaison d'expression conditionnelle avec `||` permet de réduire la redondance de notre code, en ne définissant qu'une seule fois une même liste d'instructions correspondante à deux cas alternatifs.

Indentation et autres conventions à respecter

Dans le cadre de ce cours, nous allons respecter un extrait des conventions du guide de style d'Airbnb. (c'est illustré de nombreux exemples très clairs qui aident à comprendre les bonnes et mauvaises manières d'écrire son code)

Extrait des règles qui seront à respecter:

- mettre les chaînes de caractères entre apostrophes, (cf 6.1)
- usage et placement des accolades (*braces*, en anglais), (cf 16.1)
- et indentation de 2 espaces. (cf 18.1)

Par ailleurs, pensez à inclure systématiquement `'use strict';` en première ligne de tous vos programmes.

Pour vérifier et corriger l'indentation de votre code JavaScript, vous pouvez utiliser un outil en ligne comme jsbeautifier ou la fonction "prettier" de repl.it.

Si vous préférez utiliser votre éditeur de code préféré (ex: sublime text) pour vous aider à corriger l'indentation et/ou les conventions ci-dessus, configurez-le de manière à ce qu'il respecte le guide de style d'Airbnb.