Évaluation des tests

Complément - niveau basique

On a vu dans la vidéo que l'instruction conditionnelle `if` permet d'implémenter simplement des branchements à plusieurs choix, comme dans cet exemple:

```
entree = 'bag'
if 'a' not in entree:
    print 'sans a'
elif 'b' not in entree:
    print 'sans b'
else:
    print 'avec a et avec b'
```

Comme on s'en doute, les expressions conditionnelles **sont évaluées jusqu'à obtenir un résultat vrai** - ou considéré comme vrai -, et le bloc correspondant est alors exécuté. Le point important ici est qu'une fois qu'on a obtenu un résultat vrai, on sort de l'expression conditionnelle **sans évaluer les autres conditions**.

Dans l'exemple ci-dessus, les conditions elles-mêmes ne sont pas ***actives***. On veut dire par là: la valeur de `entree` reste ***identique***, que l'on ***évalue ou non*** les deux expressions:

- `'a' not in entree`
- `'b' not in entree`

Mais nous allons voir ci-dessous qu'il est relativement facile d'écrire des conditions qui **modifient** les objets mutables sur lesquelles elles opèrent. On dit dans ce cas qu'elles font un **effet de bord**; et c'est pourquoi il est important de bien assimiler la règle des évaluations des expressions dans un `if`.

Complément - niveau intermédiaire

Rappel sur la méthode `pop`

Pour illustrer la notions d'**effet de bord**, nous revenons sur la méthode de liste `pop()` qui, on le rappelle, renvoie un élément de liste après l'avoir effacé de la liste.

```
# on se donne une liste
liste = ['premier', 'deuxieme', 'troisieme']
print "liste", liste
```

```
# pop(0) renvoie le premier element de la liste, et raccourcit la liste
element = liste.pop(0)
print "après pop(0):", "élément=", element, "et liste=", liste


# et ainsi de suite
element = liste.pop(0)
print "après pop(0):", "élément=", element, "et liste=", liste
```

Conditions avec effet de bord

Une fois ce rappel fait, voyons maintenant l'exemple suivant:

```
liste = range(5)
print 'liste en entree:', liste, 'de taille', len(liste)
if liste.pop(0) <= 0:
    print 'cas 1'
elif liste.pop(0) <= 1:
    print 'cas 2'
elif liste.pop(0) <= 2:
    print 'cas 3'
else:
    print 'cas 4'
print 'liste en sortie de taille', len(liste)
```

Avec cette entrée, le premier test est positif (car `pop(0)` renvoie 0), aussi on n'exécute en tout `pop()` qu'**une seule fois**, et donc à la sortie la liste n'a été raccourcie que d'un élément.

Exécutons à présent le même code avec une entrée différente (vivement les fonctions !)

```
liste = range(5, 10)
print 'liste en entree:', liste, 'de taille', len(liste)
if liste.pop(0) <= 0:
    print 'cas 1'
elif liste.pop(0) <= 1:
    print 'cas 2'
elif liste.pop(0) <= 2:
    print 'cas 3'
else:
    print 'cas 4'
print 'liste en sortie de taille', len(liste)
```

On observe que cette fois la liste a été raccourcie 3 fois, car les trois tests se sont révélés négatifs.

Cet exemple vous montre qu'il faut être attentif avec des conditions qui font des effets de bord. Bien entendu, ce type de pratique est de manière générale **vivement déconseillé**.

Short-circuit

La règle d'évaluation des expressions dans un `if` peut être mise en rapport avec ce qui se passe en général dans l'évaluation d'une expression booléenne, mais en dehors du contexte d'un `if`

Voyons un exemple très voisin de ce qui précède

```
liste = range(5)
print 'liste en entree de taille', len(liste)

print liste.pop(0) <= 0 or liste.pop(0) <= 1 or liste.pop(0) <= 2
print 'liste en sortie de taille', len(liste)
```

On voit ici que pour évaluer l'expression

```
liste.pop(0) <= 0 or liste.pop(0) <= 1 or liste.pop(0) <= 2
```

on a commencé par évaluer le premier terme, qui s'est avéré vrai, il n'était pas nécessaire d'évaluer les autres termes, on ne l'a donc pas fait du tout, c'est pourquoi la liste en sortie n'a été raccourcie que d'un élément.

Ce comportement est connu sous le nom de ***short-circuit***; seules les évaluations strictement nécessaires à l'obtention du résultat sont réalisées. En python, les instructions `and` et `or` font du ***short-circuit***. Regardons un exemple avec `and` maintenant

```
liste = range(1)
print 'liste en entree de taille', len(liste)

print False and liste.pop(0) <= 1 and liste.pop(0) <= 2
print 'liste en sortie de taille', len(liste)
```

Comme un seul `False` est suffisant pour que toute l'expression soit fausse, les `list.pop(0)` ne sont pas évalués.