W2-S6-C1-conditions-1

December 14, 2014

1 Conditions & Expressions Booléennes

1.1 Complément : niveau basique

Nous présentons rapidement dans ce notebook comment construire la condition qui contrôle l'exécution d'un `if`

1.1.1 Tests considérés comme vrai

Lorsqu'on écrit une instuction comme

```
if <expression>:
   <do_something>
```

le résultat de l'expression peut **ne pas être un booléen**. Pour la plupart des types, il existe des valeurs particulières qui **sont** considérées comme fausses.

Par exemple, pour n'importe quel type numérique, la valeur 0 est considérée comme fausse. Cela signifie que

```
In []: if 3 - 3:
           print "ne passera pas par là"
```

De même, une chaîne vide, une liste vide, un tuple vide, sont considérés comme faux. Bref, vous voyez l'idée générale.

```
In []: if "":
           print "ne passera pas par là"
       if []:
           print "ne passera pas par là"
       if ():
           print "ne passera pas par là"
```

Enfin le singleton `None`, que nous verrons bientôt, est lui aussi considéré comme faux.

1.1.2 Égalité

Les tests les plus simples se font à l'aide des opérateurs d'égalité, qui fonctionnent sur presque tous les objets (nous verrons un contre-exemple dans la section sur les références partagées). Comme nous l'avons déjà vu, l'opérateur `==` vérifie si deux objets ont la même valeur

```
In []: bas = 12
       haut = 25.82
       # égalité
       if bas == haut:
           print '=='
```

```
        # non égalité
        if bas != haut:
            print '!='
```

En genéral, deux objets de types différents ne peuvent pas être égaux.

```
In []: if [1, 2] != (1, 2):
            print '!='
```

Par contre, des `float`, des `int` et des `long` peuvent être égaux entre eux.

```
In []: bas_reel = 12.
       if bas == bas_reel:
            print '=='
```

Signalons à titre un peu anecdotique une syntaxe ancienne: historiquement on pouvait aussi noter <> le test de non égalité. On trouve ceci dans du code ancien mais il faut éviter de l'utiliser

```
In []: # l'ancienne forme de !=
       if bas <> haut:
            print '<> est obsolete'
```

1.1.3 Les opérateurs de comparaison

Sans grande surprise on peut aussi écrire

```
In []: if bas <= haut:
            print '<='
       if bas < haut:
            print '<'
       if bas >= haut:
            print '>='
       if bas > haut:
            print '>'
```

On peut là aussi les utiliser sur une palette assez large de types, comme par exemple avec les listes

```
In []: [1, 2] <= [2, 3]
```

Il est parfois utile de vérifier le sens qui est donné à ces opérateurs selon le type; ainsi par exemple ils désignent l'inclusion sur les ensembles - que nous verrons bientôt.

Il faut aussi se méfier avec les types numériques, si un complexe est impliqué, comme par exemple:

```
>>> 3 <= 3j
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: no ordering relation is defined for complex numbers
```

1.1.4 Connecteurs logiques et / ou / non

On peut bien sûr combiner facilement plusieurs expressions entre elles, grâce aux opérateurs `and`, `or` et `not`

```
In []: # il ne faut pas faire ceci, mettez des parenthèses
       if 12 <= 25. or [1, 2] <= [2, 3] and not 12 <= 32 :
            print "OK mais pourrait être mieux"
```

En termes de priorités: le plus simple si vous avez une expression compliquée reste de mettre les parenthèses qui rendent son évaluation claire et lisible pour tous. Aussi on préfèrera de beaucoup la formulation équivalente

```
In []: # c'est mieux
       if 12 <= 25. or ([1, 2] <= [2, 3] and not 12 <= 32) :
           print "OK, c'est équivalent et plus clair"

       # c'est bien le parenthésage ci-dessus, puisque:
       if (12 <= 25. or [1, 2] <= [2, 3]) and not 12 <= 32 :
           print "ce n'est pas équivalent, ne passera pas par là"
```

1.1.5 Pour en savoir plus

Reportez-vous à la section sur les opérateurs booléens dans la documentation python