Indentations en python

Complément - niveau basique

Comme on l'a vu dans la vidéo, en python on utilise les indentations pour structurer le code. Ce complément vise à approfondir cette notion.

Imbrications

Nous l'avons signalé également dans la vidéo, la pratique la plus courante est d'utiliser systématiquement une marge de 4 espaces

```
if 'g' in 'egg':
    print 'OUI'
else:
    print 'NON'
```

Voyons tout de suite comment on pourrait écrire plusieurs tests imbriqués

```
entree = 'spam'
if 'a' in entree:
    if 'b' in entree:
        cas11 = True
        print 'a et b'
    else:
        cas12 = True
        print 'a mais pas b'
else:
    if 'b' in entree:
        cas21 = True
        print 'b mais pas a'
    else:
        cas22 = True
        print 'ni a ni b'
```

Dans cette construction assez simple, remarquez bien **les deux points ':'** à chaque début de bloc, c'est-à-dire à chaque fin de ligne `if` ou `else`

Cette façon d'organiser le code peut paraître très étrange, notamment aux gens habitués à un autre langage de programmation, puisqu'en général les syntaxes des langages sont conçues de manière à être insensibles aux espaces et à la présentation.

Comme vous le constaterez à l'usage cependant, une fois qu'on s'y est habitué cette pratique est très agréable, une fois qu'on a écrit la dernière ligne du code, on n'a pas à réfléchir à

refermer le bon nombre d'accolades ou de 'END'.

Par ailleurs, comme pour tous les langages, votre éditeur favori connaît cette syntaxe et va vous aider à respecter la règle des 4 caractères. Nous ne pouvons pas publier ici une liste des commandes disponibles par éditeur, nous vous invitons le cas échéant à échanger entre vous sur le forum pour partager les recettes que vous utilisez avec votre éditeur / environnement de programmation favori.

Complément - niveau intermédiaire

Espaces *vs* tabulations

Il nous faut par contre donner quelques détails sur un problème que l'on rencontre fréquemment sur du code partagé entre plusieurs personnes quand celles-ci utilisent des environnement différents.

Pour faire court, ce problème est **susceptible d'apparaître dès qu'on utilise des tabulations**, plutôt que des espaces, pour implémenter les indentations. Aussi, le message à retenir ici est **de ne jamais utiliser de tabulations dans votre code python**

En version longue, il existe un code ASCII pour un caractère qui s'appelle *Tabulation* (alias Control-i, qu'on note aussi `^I`); l'interprétation de ce caractère n'étant pas clairement spécifiée, il arrive qu'on se retrouve dans une situation comme la suivante.

Bernard utilise l'éditeur `vim`; sous cet éditeur il lui est possible de mettre des tabulations dans son code, et de choisir la valeur de ces tabulations. Aussi il va dans les préférences de `vim`, choisit Tabulation=4, et écrit un programme qu'il voit comme ceci

```
if 'a' in entree:
    if 'b' in entree:
        cas11 = True
        print 'a et b'
    else:
        cas12 = True
        print 'a mais pas b'
```

Sauf qu'en fait, il a mis un mélange de tabulations et d'espaces, et en fait le fichier contient (avec ^I pour tabulation)

```
if 'a' in entree:
^Iif 'b' in entree:
^I^Icas11 = True
    ^Iprint 'a et b'
^Ielse:
^I^Icas12 = True
    ^Iprint 'a mais pas b'
```

Bernard envoie son code à Alice qui utilise `emacs`. Dans son environnement, `emacs` affiche une tabulation comme 8 caractères. Du coup Alice "voit" le code suivant

```
if 'a' in entree:
        if 'b' in entree:
                cas11 = True
            print 'a et b'
    else:
        cas12 = True
            print 'a mais pas b'
```

Bref, c'est la confusion la plus totale. Aussi répétons-le, **n'utilisez jamais de tabulations dans votre code python**

Complément - niveau avancé

Vous pouvez trouver du code qui ne respecte pas la convention des 4 caractères.

En version courte: **Utilisez toujours des indentations de 4 espaces**.

En version longue, et pour les curieux: python **n'impose pas** que les indentations soient de 4 caractères. Aussi vous pouvez rencontrer un code qui ne respecte pas cette convention, et il nous faut, pour être tout à fait précis sur ce que python accepte ou non, préciser ce qui est réellement requis par python.

La règle utilisée pour analyser votre code, c'est que toutes les instructions dans un même bloc sont présentées avec le même niveau d'indentation. Si deux lignes successives - modulo les blocs imbriqués - ont la même indentation, elles sont dans le même bloc.

Voyons quelques exemples. Tout d'abord le code suivant est **légal**, quoique, redisons-le pour la dernière fois, **pas du tout recommandé**

```
# code accepté mais pas du tout recommandé
if 'a' in 'pas du tout recommande':
  succes = True
  print 'OUI'
else:
      print 'NON'
```

En effet les deux blocs (après `if` et après `else`) sont des blocs distincts, ils sont libres d'utiliser deux indentations différentes (ici 2 et 6)

Par contre la construction ci-dessous n'est pas légale

```
# ceci n'est pas correct et rejeté par python
if 'a' in entree:
    if 'b' in entree:
        cas11 = True
        print 'a et b'
      else:
        cas12 = True
        print 'a mais pas b'
```

En effet les deux lignes 'if' et 'else' font logiquement partie du même bloc, elles **doivent** avoir la même indentation. Avec cette présentation le lecteur python émet une erreur et ne peut pas interpréter le code.