Bonnes pratiques de présentation de code

Complément - niveau basique

La PEP-008

On trouve dans la PEP-008 (en anglais) (http://legacy.python.org/dev/peps/pep-0008/) les conventions de codage qui s'appliquent à toute la librairie standard, et qui sont certainement un bon point de départ pour vous aider à trouver le style de présentation qui vous convient.

Nous vous recommandons en particulier les sections sur

- l'indentation (http://legacy.python.org/dev/peps/pep-0008/#code-lay-out)
- les espaces (http://legacy.python.org/dev/peps/pep-0008/#whitespace-in- expressions-and-statements)
- les commentaires (http://legacy.python.org/dev/peps/pep-0008/#comments)

Un peu de lecture : le module `pprint`

Voici par exemple le code du module `pprint` (comme PrettyPrint) de la librairie standard, que nous avons déjà rencontré et qui permet d'imprimer des données.

La fonction du module - le pretty printing - est évidemment accessoire ici, mais vous pouvez y voir illustré

- le *docstring* pour le module, jusqu'en ligne 35
- les indentations, comme nous l'avons déjà mentionné sont à 4 espaces, et sans tabulation
- l'utilisation des espaces, notamment autour des affectations et opérateurs, des définitions de fonction, des appels de fonctions...
- les lignes qui restent dans une largeur "raisonnable" (79 caractères); vous pouvez voir notamment
- la façon de couper les lignes pour respecter cette limite en largeur

```
from modtools import show_module_html
import pprint
show_module_html(pprint, lineno_width=3)
```

Coupures de ligne

Nous allons zoomer dans ce module pour voir quelques exemples de coupure de ligne.

Coupure de ligne sans *backslash* (\)

```
show_module_html(pprint,
                 beg="def pprint",
                 end="def pformat",
                 lineno_width=3)
```

La fonction `pprint` (ligne ~55) est une commodité (qui crée une instance de `PrettyPrinter`, sur lequel on envoie la méthode `pprint`).

Vous voyez ici qu'il n'est pas nécessaire d'insérer un backslash (\) à la fin de la ligne 57 car il y a une parenthèse ouvrante qui n'est pas fermée

---

Coupure de ligne avec *backslash* (\)

```
show_module_html(pprint,
                 beg="components), readable, recursive",
                 end="elif _len(object) ",
                 lineno_width=3)
```

Dans ce fragment au contraire, vous voyez en ligne 290 qu'il a fallu cette fois insérer un *backslash* \ comme caractère de continuation pour que l'instruction puisse se poursuivre en ligne 291.

Complément - niveau intermédiaire

Les deux-points ':'

Vous pouvez [vous reporter à ce lien](https://docs.python.org/2/faq/design.html (https://docs.python.org/2/faq/design.html)

why-are-colons-required-for-the-if-while-def-class-statements) si vous êtes

intéressé par la question de savoir pourquoi on a choisi un délimiteur (le caractère deux-points :) pour terminer les instructions comme `if`, `for` et `def`.

Complément - niveau avancé

Signalons enfin, pour ceux qui sont intéressés par les conventions de codage, que tout ceci ne s'applique bien entendu qu'à python. Pour quelques exemples qui s'appliqueraient au langage C, vous pouvez par curiosité voir le CodingStyle pour le noyau linux (https://www.kernel.org/doc/Documentation/CodingStyle) , qui lui-même fait référence - et pas en bien -

aux Gnu Coding Standards (http://www.gnu.org/prep/standards/standards.html) .

Même si le langage est différent, le but des conventions de codage est toujours le même, il s'agit d'améliorer la lisibilité, et il peut être intéressant de voir ce que Linus Torvalds a à dire, dans le style qui est le sien, sur les indentations (le tout premier chapitre).