PYTHON - PEP 8

Dans ce chapitre nous aborderons les notions suivantes :

Code pythonique

PEP 8

Pylint

Utiliser l'extension Pylint avec un éditeur de texte

Python Extension Proposal : PEP

Le langage Python a été créé par Guido van Rossum (*dictateur bienveillant à vie*) mais grandit et évolue grâce à sa communauté. Chaque proposition d'amélioration est publique et publiée sur **python.org** . Elles sont connues sous le nom de *Python Enhancement Proposal* (proposition d'amélioration de Python) et portent chacunes un numéro. Certaines sont acceptées, d'autres reportées mais toutes sont ouvertes au débat. Vous pouvez en consulter la liste sur la page : **https://www.python.org/dev/peps/.**

La PEP 8

La PEP 8 a pour objectif de définir des règles de développement communes entre développeurs. En effet, vous avez pu remarquer que le langage est assez flexible. Par exemple, il est possible d'utiliser 2 espaces ou 4 pour l'indentation. Vous pouvez également ne pas mettre d'espace entre les opérateurs. Le code fonctionnera toujours ! Mais ce n'est pas parce que vous **pouvez** que vous **devez** le faire !

Que se passe-t-il le jour où une autre personne reprend votre projet ? Elle essaiera de le comprendre avant tout. Si le code est complexe, dense, et qu'il ne suit pas des règles communes, votre interlocuteur aura besoin de plus de temps et d'énergie pour se l'approprier. Ce n'est pas ce que vous souhaitez !

Guido van Rossum, d'ailleurs, insiste sur le fait qu'un développeur passe **plus de temps à lire du code qu'à l'écrire.** C'est pourquoi votre code doit avant tout être facile à comprendre. Autrement dit, **lisible**.

Contenu de la PEP 8

Nous verrons quelques points clés qui sont essentiels mais nous ne détaillerons pas la PEP 8 dans son ensemble. Pourquoi ? Avant tout car elle est très bien expliquée sur le site ! Mais également car j'aimerais attirer votre attention sur quelques points clés, essentiels à connaître, avant de vous laisser la parcourir par vous-même.

Mise en page

Votre code suit une certaine syntaxe et une mise en page. Vous le savez déjà : l'indentation a une importance capitale dans ce langage ! Voici quelques règles importantes :

> **Une ligne doit contenir 80 caractères maximum.**
>
> ***L'indentation doit être de 4 espaces.***
>
> **Ajoutez deux lignes vides entre deux éléments de haut niveau, des classes par exemple, pour des questions d'ergonomie.**
>
> **Séparez chaque fonction par une ligne vide.**
>
> **Les noms (variable, fonction, classe, ...) ne doivent pas contenir d'accent. Que des lettres ou des chiffres !**

```python
#Je suis une ligne composée de quatre-vingts caractères maximum, ni plus, ni moins !
print("-- I am a line made of eighty characters maximum, no more, no less! --")

if n < m:
    print('indent: 4 spaces')


class FirstClass:
    def __init__(self, luggages):
        self.luggages = luggages

    def travel(self):
        print('Luxe, calme et volupté')


# Séparez chaque élément de haut niveau par deux lignes
class SecondClass(FirstClass):
    def __init__(self):
        self.luggages = 0

    # Séparez chaque élément de bas niveau par une ligne
    def travel(self):
        print('Down the road, I go ... ')
```

Documentation strings **(Docstrings) obligatoires**

Une doctring est un ensemble de mots qui documente un bout de code. Elle commence par trois guillemets ouvrants, le commentaire que vous souhaitez apporter puis trois guillements fermants.

Exemple :

```python
""" Ceci est un document. Je suis ici pour expliquer ce que le code suivant fera.
Oh, moi aussi je suis multiligne !
"""
```

Selon la PEP 8, chaque partie de votre code devrait contenir une Doctring.

- **tous les modules publics**
- **toutes les fonctions**
- **toutes les classes**
- **toutes les méthodes de ces classes**

Imports

L'import d'une librairie doit être rapide à déceler. Il est également important de bien différencier la source des librairies : standard, externe ou locale. Cela permet de savoir ce qu'il faut installer.

- Les imports sont à placer au début d'un script.
- Ils précèdent les Docstrings.
- Une ligne par librairie. Exemple : **import os**
- Une ligne peut néanmoins inclure plusieurs composantes. Exemple : **from subprocess import Popen, PIPE**
- L'import doit suivre l'ordre suivant : Bibliothèques standard, Bibliothèques tierces et imports locaux. Sautez une ligne entre chacun de ces blocs.

Espaces dans les instructions

Les espaces suivent la syntaxe anglosaxone et non française. De manière plus générale, elle s'axe sur la lisibilité tout en supprimant les espaces surperflus.

Pas d'espace avant : mais un après. Exemple :

`{oeufs: 2}`

Opérateurs : un espace avant et un après. Exemple :

```
i = 1 + 1
```

Aucun espace avant et après un signe `=` lorsque vous assignez la valeur par défaut du paramètre d'une fonction. Exemple : `def elephant(trompe=True, pattes=4)`

Une instruction par ligne.

Commentaires

Même si les développeurs ne sont pas tous d'accord sur ce point, les commentaires jouent un rôle essentiel dans la compréhension d'un code. Voici quelques bonnes pratiques :

Ecrivez des phrases complètes, ponctuées et compréhensibles.

Le commentaire doit être cohérent avec le code.

Il doit suivre la même indentation que le code qu'il commente.

Evitez d'enfoncer des portes ouvertes : ne décrivez pas le code, expliquez plutôt à quoi il sert.

Il doit être **en anglais**.

Guido van Rossum est très clair sur ce point. Il l'explique d'ailleurs ainsi : "Aux développeurs Python de pays non anglophones : écrivez vos commentaires en anglais, s'il vous plaît, sauf si vous êtes sûr à 120% que ce code ne sera jamais lu par des personnes qui ne parlent pas votre langage.

Conventions de nommage

Il s'agit d'un point essentiel que nous avons couvert tout au long de ces cours sur Python. En voici les grandes lignes :

Modules : nom court, tout en minuscules, tiret du bas si nécessaire. `great_module.py`

paquets : nom court, tout en minuscules, tirets du bas très déconseillés. `paquet`

classes : lettres majuscules en début de mot. `MonObjetPython`

exceptions : similaire aux classes mais avec un Error à la fin. `MonExceptionError`

fonctions : minuscules et tiret du bas : `my_function()`

méthodes : minuscules, tiret du bas et `self` en premier paramètre : `my_method(self)`

arguments des méthodes et fonctions : identique aux fonctions. `my_function(param=False)`

variables : identique aux fonctions.

constantes : tout en majuscules avec des tirets si nécessaire. `I_WILL_NEVER_CHANGE`

privé : précédé de deux tirets du bas : `__i_am_private`

protégé : précédé d'un tiret du bas : `_i_am_protected`

Comparaisons

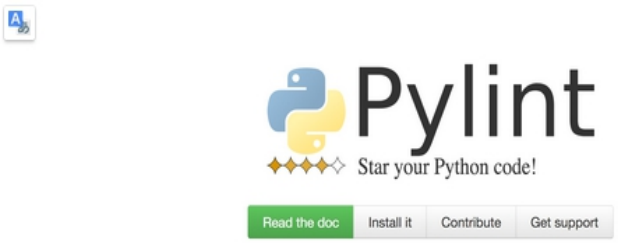
Par convention, il est mieux d'utiliser `is` ou `is not` lors d'une comparaison avec `None`. Par exemple :

```
if datafile is None:
    print("Je suis Pythonique")
```

La PEP 8 en action dans le projet

Les conventions de la PEP 8 s'appliquent au fur et à mesure du développement d'un projet. Néanmoins, il est intéressant d'effectuer une recherche de temps en temps (avant un commit par exemple) pour déceler les quelques incartades qui pourraient se glisser. Plus vous coderez, plus vous la connaîtrez par coeur et donc moins vous aurez besoin de la regarder !

Bien sûr, vous pouvez le faire à la main : reprenez la liste plus haut et appliquez-la à votre code. Mais c'est fastidieux et chronophage. Plusieurs solutions d'analyse de code existent pour vous faciliter le travail ! Laissez-moi vous en présenter une : Pylint.



Pylint est un logiciel en ligne de commande qui analyse le code en fonction des standards de la PEP 8. Il détecte également les erreurs et le code dupliqué.

Utiliser Pylint

Commencez par installer Pylint :

```text
pip install pylint
```

Quand Pylint est installé, lancez l'analyse en tapant `pylint [votrescript.py]` .

```text
pylint parite.py
```

Il existe de nombreuses extensions qui permettent d'utiliser Pylint directement dans un éditeur de texte. Linter-pylint pour Atom ou SublimeLinter-pylint pour Sublime Text par exemple. Dans les deux cas, vous devez lancer l'éditeur en ayant activé l'environnement virtuel ! Sinon votre éditeur ne connaîtra pas le chemin vers l'exécutable et ne pourra pas lancer Pylint.

> **Installez l'extension Pylint pour l'éditeur de votre choix.**
>
> **Fermez ce dernier.**
>
> **Ouvrez votre console et déplacez-vous dans le répertoire de travail souhaité.**
>
> **Activez l'environnement virtuel :** `source env/bin/activate`
>
> **Ouvrez les fichiers avec l'éditeur de texte :** `atom .` **pour Atom ou** `subl .` **pour Sublime Text.**

La PEP 8 est un ensemble de règles très importantes mais qui peuvent être transgressées dans certains contextes spécifiques. Retenez simplement qu'elle garantit un code lisible et de qualité qui améliore la collaboration entre développeurs.