Bonnes pratiques en Python

The Zen of Python

```
>>> import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

En français :

Préfère :

> la beauté à la laideur,
> l'explicite à l'implicite,
> le simple au complexe
> et le complexe au compliqué,
> le déroulé à l'imbriqué,
> l'aéré au compact.

Prends en compte la lisibilité.
Les cas particuliers ne le sont jamais assez pour violer les règles.
Mais, à la pureté, privilégie l'aspect pratique.
Ne passe pas les erreurs sous silence,
... ou bâillonne-les explicitement.
Face à l'ambiguïté, à deviner ne te laisse pas aller.
Sache qu'il ne devrait [y] avoir qu'une et une seule façon de procéder,

même si, de prime abord, elle n'est pas évidente, à moins d'être Néerlandais.
Mieux vaut maintenant que jamais.
Cependant jamais est souvent mieux qu'immédiatement.
Si l'implémentation s'explique difficilement, c'est une mauvaise idée.
Si l'implémentation s'explique aisément, c'est peut-être une bonne idée.
Les espaces de nommage ! Sacrée bonne idée ! Faisons plus de trucs comme ça.

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.
- Une instruction par ligne.
- Séparez chaque fonction par une ligne vide.
- Les noms (variable, fonction, classe, ...) ne doivent pas contenir d'accent. Que des lettres ou des chiffres !

Conventions de nommage

Noms à éviter

N'utilisez jamais les caractères suivants de manière isolée comme noms de variables : `l` (L minuscule), `O` (o majuscule) et `I` (i majuscule). L'affichage de ces caractères dans certaines polices fait qu'ils peuvent être aisément confondus avec les chiffres `0` ou `1`.

Noms de variables, fonctions et méthodes

La même convention est utilisée pour les noms de variables, de fonctions ou de méthodes : le nom est entièrement écrit en minuscules et les mots sont séparés par des signes soulignés (`_`).

```
def nom_de_fonction():
    pass
```

Constantes

Les constantes doivent être écrites entièrement en majuscules, les mots étant séparés par un signe souligné (`_`).

```
NOM_DE_MA_CONSTANTE = 12
```

Les 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.

Pour les opérateurs : un espace avant et un après.

```
# il faut faire ceci :

variable = 'valeur'

ceci == cela

1 + 2

# il ne faut pas faire cela :

variable='valeur'

ceci==cela

1+2
```

Une exception, on regroupe les opérateurs mathématiques ayant la priorité la plus haute pour distinguer les groupes :

```
a = x*2 - 1

b = x*x + y*y

c = (a+b) * (a-b)
```

Aucun espace avant et après un signe `=` lorsque vous assignez la valeur par défaut du paramètre d'une fonction.

```
def fonction(argument='valeur'):            # ça c'est ok
    pass

resultat = fonction(argument='valeur')      # ça aussi
```

On ne met pas d'espace à l'intérieur des parenthèses, crochets ou accolades.

```
# il faut faire ceci :

2 * (3 + 4)

def fonction(arg='valeur'):

{str(x): x for x in range(10)}

val = dico['key']

# il ne faut pas faire cela :

2 * ( 3 + 4 )

def fonction( arg='valeur' ):

{ str(x): x for x in range(10) }

val = dico[ 'key' ]
```

On ne met pas d'espace avant les deux points et les virgules, mais après oui.

```
# il faut faire ceci :

def fonction(arg1='valeur', arg2=None):

dico = {'a': 1}

# il ne faut pas faire cela :

def fonction(arg='valeur' , arg2=None) :

dico = {'a' : 1}
```

Les imports de librairie

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.

```
# il faut faire ceci :
import sys
import os
from minibelt import (dmerge, get, iget, normalize,
                      chunks, window, skip_duplicates, flatten)

# il ne faut pas faire cela :
import sys, os
```

```
s = ("Les chaînes Python sont automatiquement "
     "concaténées si elles sont "
     "uniquement séparées par des espaces "
     "ou sauts de lignes.")
```

Les 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 débutants avec une majuscule, 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 théoriquement être **en anglais**

```
# Les commentaires débutent avec le dièse (hashtag)
```

Docstrings obligatoires

Une doctring (chaîne de documentation) 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.

```
"""Ceci est une docstring qui va expliquer le fonctionnement pour une fonction ou
un morceau de code.
"""
```

Chaque partie de votre code devrait contenir une Doctring.

- tous les modules publics
- toutes les fonctions
- toutes les classes et méthodes de ces classes (les classes seront vues en terminale)

Remarque : La documentation d'une fonction (ou méthode) doit décrire son comportement et documenter ses arguments, sa valeur de retour, ses effets de bord, les exceptions qu'elle peut lever et les restrictions concernant son appel (quand ou dans quelles conditions appeler cette fonction). Les paramètres optionnels doivent également être documentés.

Pour aller plus loin

http://nguyen.univ-tln.fr/share/Python/pep8.pdf