Bonnes pratiques python

Bonnes pratiques de bases

Voici une liste de bonnes pratiques pythons. **Suivez-les au maximum !** Elles ne sont pas coûteuses et faciliteront la vie à vos chargés de TP tout en vous permettant acquérir de bon reflexes de programmation utiles partout !

1. **Nom de variable et de fonction signifiant** : vos variables représentent quelque chose, alors donnez leur comme nom ce qu'elles représentent. Qu'une personne qui découvrent votre code puisse en une lecture comprendre vos variables. Par exemple :

```python
liste_a_trier = [2,3,4,5,6,8]
nom = "Alice"

def inversion_valeur(valeur_gauche, valeur_droite):
    """
    Echange la position des valeurs en entrée et retourne un set des deux valeurs
    """
    valeur_tampon = valeur_droite
    valeur_droite = valeur_gauche
    valeur_gauche = valeur_tampon
    return  (valeur_gauche,valeur_droite)
```

   En plus cela vous aide souvent à écrire un code sans erreur

2. **Nom de variable en snake case** : vos variables sont toutes en minuscule avec les mots séparés par des undescore (_). Exception des variables constantes écrites en majuscule avec les mots séparé par des undescore (_)
3. **Un seul return par fonction** : même si ce n'est pas fondamentalement faux d'avoir plusieurs return dans une fonction, privilégiez un return unique à la fin de la fonction. Cela évite les erreurs avec des returns au milieu de la fonction (*cela ne concerne pas les fonction récurrente*)
4. **N'utilisez pas la console python** : il semblerait que certains d'entre vous aient le reflexe d'utiliser la console python pour exécuter les codes python. **C'EST UNE TRES MAUVAISE PRATIQUE**. Préférez toujours exécuter votre script en entier. Cela vous permet de vous assurer que votre code fonctionne dans son intégralité et qu'il n'y a pas besoin d'une manipulation de votre part pour le faire fonctionner.
5. **Ne vérifiez pas vos comparaisons** : n'écrivez pas

```python
if (variable_a_tester > seuil) == True :
    pass
```

   C'est complètement inutile ! `variable_a_tester > seuil` retourne déjà `True` si la condition est remplie et `False` sinon. Faire ce test en plus est une perte de temps. Ecrivez directement

```python
if (variable_a_tester > seuil) :
    pass
```

Cela vaut pour toutes les expressions qui retournent un booléens et dans tout les langages (donc R également)

6. **Documentez votre code** : ajouter des commentaires, et des docstrings de coûte pas beaucoup sur le moment mais rend le code plus lisible par la suite. Au minimum expliquez toujours ce que fait une fonction, ajouter des commentaires pour expliquer des zones de code difficiles. N'oubliez jamais que votre code doit être lisible par d'autres, ainsi que par vous dans une semaine.
7. **Limitez les imbrications** : évitez d'avoir trop de boucles et conditions imbriquées. Voici quelques conseils :
    - Pas plus de 3 niveaux d'imbrications. Si cela vous arrive découpez votre code en fonction plus petites avec des noms signifiants
    - Assurez vous de toujours voir sur votre écran le début et la fin d'une boucle. Si cela vous arrive découpez votre code en fonction plus petites avec des noms signifiants.
8. **Utilisez la bonne collection** : python propose différente collection, chacune avec des avantages et défauts. La liste n'est pas toujours le meilleurs choix.

Astuces et bonnes pratiques avancées

- Joindre des strings : préférez

```python
"".join(string1, string2)
```

à

```python
string1 + string2
```

- Comparer une variable à None avec `is` ou `is not`

```python
if foo is None:
```

- Préférez la structure `is not` à `not ... is`

```python
# Correct
if foo is not None:

# Faux
if not foo is None:
```

- Préférez `''.startswith()` and `''.endswith()` au découpage de string pour tester un suffixe ou préfixe.

```python
# Correct:
if foo.startswith('bar'):
```

- Pour tester si une séquence est vide utilisez le fait qu'une séquence vide est égale à `False` en python

```
# Correct:
if not seq: # teste si non vide
if seq: # teste si vide
```