Rappel : Structure d'une fonction

Modèle

```
def nom_fonction(arg1, arg2…):
    """ Information générale
       param arg1 : explication et type
       param arg2 : explication et type
       return variable :explication
                        et type
       précondition : explication
       postcondition : explication
    """
    Instruction 1
    Instruction 2
    ---
    return variable
```

Docstring

Corps de la fonction

Les docstrings sont destinées aux utilisateurs

Les éléments essentiels pour les fonctions sont :

- ce que fait la fonction ou la méthode,
- ce qu'elle prend en argument,
- ce qu'elle renvoie.

On peut ajouter des informations générales sur leur fonctionnement.

Règles

- Sur un ligne ou plusieurs selon les cas
- Ecrire des docstrings avec des triples doubles guillemets :
  - `"""Ceci est une docstring recommandée."""`

mais pas

  - `'''Ceci n'est pas une docstring recommandée.'''.`

Exemple

```
def produit_de_nombres(nombre1, nombre2):
    """
     Multiplication de deux nombres
     entiers.
    Cette fonction ne sert pas à grand
     chose.

    Paramètres
    ----------
    nombre1 : int Le premier nombre
    nombre2 : int  Le second nombre

    Retour
    -------
    int : Le produit des deux nombres.
    """

    return nombre1 * nombre2
```

Structure des programmes

Organisation du code

Il est fondamental de toujours structurer et organiser son code de la même manière. Ainsi, on sait tout de suite où trouver l'information et un autre programmeur pourra s'y retrouver.

Principe généraux

- Découper le programmes en de nombreuses fonctions
  - Qui doivent être bien nommée et bien faire une seule chose
- Ne pas utiliser de variables globales
- Au plus 40 lignes par fonction en comptant les commentaires
- Au plus 4 niveaux d'indentation
- Éviter d'utiliser `break` et `continue`
- Éviter les codes cryptique (code génial?)
- Ne pas réinventer la roue : utiliser les bibliothèques python

Éviter les variables globales

- Ne pas utiliser de variables globales, c'est à dire de variables définies sans niveau d'indentation
- Mais vous pouvez utiliser **des constantes globales**

Exemples :

Ne pas faire cela

```
x = 10
y = 10
print('x + y = ', x + y)
```

Faire cela à la place

```
def main() :
    """la fonction principale"""
    x = 10
    y = 10
    print('x + y = ', x + y)
```

LES CONSTANTES

Utilité et utilisation

- Évitez d'utiliser des valeurs constantes codées en dur au milieu du code
- Définissez les au début du programme
- Utilisez des MAJUSCULES (avec des underscore pour séparer les mots)
  - Plus facilement maintenable et plus lisible

Elles sont définies au niveau global, c'est-à-dire qu'elle est disponible pour toutes les fonctions du programme sans être passé en argument.

C'est correct car elle sont traitée comme des valeurs constantes et ne devraient être modifiées par aucune fonction.

Exemple 1

Ne pas faire cela

```
While erreur > 0.00001 :
            ….
```

Mais plutôt faire

```
TOLERANCE = 0.00001
---
While erreur > TOLERANCE:
    ---
```

Exemple 2

Ne pas faire cela

```
if taille > 200
            ---
```

Mais plutôt faire

```
MAX_TAILLE = 200
---
if taille > MAX_TAILLE:
    ---
```

Structure d'un programme

Docstring du programme

Cette docstring décrit globalement le script et donne quelques informations sur la version du script, les auteurs...

Importation des modules

Un module par ligne.

Définition des constantes

Le nom des constantes est en majuscule.

Définition d'une fonction 1

La docstring indique ce que retourne la fonction, et décrit les paramètres, le retour, les préconditions et les postconditions

...

Définition d'une fonction n

Le programme principal

C'est celui qui est lancé lorsqu'on exécute le script. Il est composé d'appels des fonctions définies précédemment.

Appel du programme principal

Exemple

```
"""
Ce programme calcule le périmètre d'un cercle dont
le rayon a été demandé au clavier à l'utilisateur.
le nombre de chiffres après la virgule est de 5 par
défaut
Auteur : Turing21
"""
import math

NOMBRE_CHIFFRES = 5

def perimetre_cercle(un_rayon):
    """Calculer le périmètre d'un cercle à partir
         de son rayon.
     :param un_rayon: le rayon du cercle (positif)
     :return le périmètre d'un cercle de rayon
      un_rayon
    """
    diametre = 2 * un_rayon
    return round(math.pi*diametre,NOMBRE_CHIFFRES)

def main():
    """Le programme principal."""

    saisie = input("Rayon du cercle : ")
    le_rayon = float(saisie)
    perimetre = perimetre_cercle(le_rayon)

    print("Le périmètre d'un cercle de rayon",
            le_rayon, "est", perimetre)

#Appel du programme principal
main()
```

PYLINT

Pylint vérifie ce qui suit (et plus)

- Les lignes doivent comporter moins de 80 caractères.
- Les fonctions doivent avoir des docstrings.
- Les noms de variables et les noms de fonctions doivent être de la forme suivante:
  - généralement 3 caractères minuscules ou plus
  - en utilisant des caractères de soulignement si nécessaire.
  - par exemple :
    - `degre_de_courbure = 10`
    - `def ecrit_nom(nom)`
- Le code doit être espacé correctement par exemple:
  - espace autour des opérateurs d'affectation et de comparaison,
  - après les virgules, etc.
- Au plus 40 lignes par fonction.
- Au plus 4 paramètres à une fonction
- Complexité limitée dans une fonction
  - chaque for, while, if, else, elif etc, compte comme un seul point de branchement;
  - jusqu'à 6 points de branchement de ce type sont autorisés dans chaque fonction.
- L'indentation doit être de 4 espaces.
- Au plus deux return dans une fonction.
- Les modules et les programmes ne doivent utiliser aucune variable globale.
- Pylint vérifie que toutes les variables déclarées dans la "portée externe", c'est-à-dire en dehors de toute fonction, sont écrites EN MAJUSCULE.
  - Par convention, cela indique qu'elles sont des constantes et non des variables.
  - Cela signifie également que toute tentative d'utilisation de variables globales sera rejetée.

Pylint et l'interdiction des variables globales

Exemple

Un programme simple comme celui ci-dessous échouera à passer le test Pylint.

```
taille = float(input('Quelle est votre
                     taille en cm? '))
    if taille > 200:
    print("Désolé, vous êtes trop grand
          pour monter dans le vaisseau
          spatial:(")
```

En effet,

- `'taille'` est maintenant une variable globale (car elle est en dehors de toutes les fonctions et visible à tous) .
- Il échouera également car il n'y a pas de docstring pour l'ensemble du programme.

Cela signifie que le seul code que vous pouvez exécuter en dehors d'une fonction est le code qui appelle une fonction. Cela peut sembler étrange mais cela rend votre code beaucoup plus portable.

On peut changer le programme précédent:

```
"""Programme qui satisfait le test
    Pylint.
   Auteur:  Ada Lovelace
   Date:  annee 1842.
    """

MAX_TAILLE = 200

def main():
    """Programme principal"""
    taille = float(input('Quelle est
     votre taille en cm? '))
    if taille > MAX_TAILLE:
        print("Désolé, vous êtes trop
         grand pour monter dans
         le vaisseau spatial:(")

# Execute le programme
main()
```

Notez que la variable MAX_TAILLE est défini au niveau global, c'est-à-dire qu'elle est disponible pour toutes les fonctions du programme sans être passé en argument. C'est correct car elle est traitée comme une valeur constante et ne devrait être modifié par aucune fonction.