INF136 – NORMES DE PROGRAMMATION

par

IANNICK GAGNON

MAÎTRE D'ENSEIGNEMENT

SERVICE DES ENSEIGNEMENTS GÉNÉRAUX

MONTRÉAL, LE 24 FÉVRIER 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

La qualité de ce document est importante pour nous. Veuillez communiquer toute erreur à iannick.gagnon@etsmtl.ca.

TABLE DES MATIÈRES

INTRODUCTION

Ce document constitue un référentiel centralisé des normes de programmation imposées dans le cadre de ce cours. Les normes de programmation sont un ensemble de règles qui régissent l'écriture du code. Elles permettent d'écrire du code qui est plus **clair**, plus **lisible** et de **meilleure qualité** au sens large. **Vous devez appliquer ces règles en tout temps.**

Notez que ce document s'inspire du guide de style PEP 8 pour « Python Enhancement Proposal » qui est la référence officielle en la matière. Nous vous invitons à consulter la version originale à l'adresse qui suit : https://www.python.org/dev/peps/pep-0008/.

RÉSUMÉ

Les normes de programmation sont les règles qui dictent comment le code doit être écrit, documenté et commenté.

Les programmes doivent débuter par un en-tête qui décrit le programme, ces entrées et les valeurs retournées en utilisant le style proposé par Google présenté à la Section 1. La description doit inclure le type et expliquer la nature de son contenu. Les en-têtes sont une forme de commentaires, mais se distinguent des commentaires qui sont dans le code par le fait qu'ils s'adressent aux utilisateurs de vos programmes plutôt qu'aux programmeurs.

La fameuse question « Qu'y a-t-il dans un nom? » perd tout son sens en programmation, car la réponse est que tout y est. Les noms des variables/fonctions/procédures doivent être significatifs et écrits en minuscules avec les mots séparés de barres de soulignement. Les constantes sont soumises aux mêmes règles à la différence qu'elles s'écrivent uniquement en majuscules. On utilise ces dernières pour remplacer ce qu'on appelle les nombres magiques qui sont des valeurs numériques écrites à même le code.

Finalement, l'indentation joue un rôle important dans Python et si les règles d'usage ne sont pas respectées, l'interpréteur lancera une erreur de type `IndentationError` qui aura pour effet d'interrompre l'exécution du programme. Une unité d'indentation correspond à une tabulation ou 4 espaces. Il est important de ne pas mélanger les tabulations et les espaces et d'être consistant-es. Toutes les lignes de code à l'intérieur d'une fonction/procédure, d'une instruction de répétition et d'une structure de sélection/choix doivent être indentées. Le nombre de caractères d'une ligne de code/commentaire est limité à 120.

1. EN-TÊTE DU PROGRAMME

Chaque déclaration de sous-programme doit être accompagnée d'un commentaire d'en-tête qui contient les informations suivantes :

1. Une description de la tâche accomplie.
2. Une description de chacun des arguments d'entrée.
3. Une description de chacune des valeurs de retour.

Les en-têtes s'adressent à des utilisateurs plutôt qu'à des programmeurs. Ils décrivent donc le **quoi** et non pas le **comment**. À l'inverse, les commentaires que l'on retrouve dans le code décrivent le comment.

Dans ce cours, nous utilisons le modèle d'en-tête de Google dont voici un exemple générique :

```
"""
Ceci est un exemple d'en-tête qui suit le style de Google.

Arguments :
    param1 (type): Description du premier paramètre.
    param2 (type): Description du deuxième paramètre.

Retourne :
    (type): Description de la/les valeurs de retour.
"""
```

À titre d'exemple, voici l'en-tête d'une fonction qui retourne le n^e terme de la suite de Fibonacci :

```
"""
Retourne la nième valeur de la suite de Fibonacci. Cette
dernière correspond à : 1 1 2 3 5 8 13 21 (…).

Les deux premiers termes valent 1 et les termes suivants
sont définis comme étant égaux à la somme des deux termes
précédents. Le prochain terme de la suite décrite ci-dessus
est donc égal à la somme de 13 et 21, soit 34.

Arguments :
    n (int): La position du terme à retourner.

Retourne :
    (int): Le nième terme de la suite de Fibonacci.
"""
```

Nous verrons au cours 3 qu'il est possible de donner des **valeurs par défaut** à des paramètres ce qui les rendent **optionnels** du point de vue de l'appel puisqu'ils ont déjà une valeur d'assignée si aucune n'est fournie. Il faut identifier le paramètre avec le mot-clé *optionnel* avec la déclaration de son type et indiquer sa valeur par défaut dans la description :

```
"""
Approximation de la racine carrée par la méthode de Babylone.

Arguments :
    x (float) : Valeur dont on veut calculer la racine carrée.
    max_iter (int, optionnel) : Nombre maximal d'itérations
        fixé à 100 par défaut.

Retourne :
    Rien.
"""
```

Remarquez aussi que lorsqu'une description de paramètre prend plus d'une ligne, **ces lignes subséquentes sont indentées**.

Il se peut qu'un sous-programme ne possède **aucuns paramètres** et/ou **ne retourne aucune valeur**. Dans ce cas, on utilise les formulations suivantes :

```
"""
Affiche le menu principal.

Arguments :
    Aucuns.

Retourne :
    Rien.
"""
```

2. LIBRAIRIES

Lorsque vous importez des librairies, vous devez respecter les règles suivantes :

- Placez les instructions d'importation (`import`) au début du fichier et jamais dans un sous-programme :

```
from math import sin      ✓
from math import cos      ✓

def programme_principal():

  from math import tan    x

  theta = float(input('Saisissez un angle en radians : '))

  print(f'sin({theta}) = {sin(theta)}\n'
        f'cos({theta}) = {cos(theta)}\n'
        f'tan({theta}) = {tan(theta)}\n')
```

- Il ne doit y avoir qu'**une seule importation par ligne** :

```
from math import sin          ✓
from math import cos          ✓

from math import asin, acos   x


…
```

- Ne jamais importer le contenu entier d'une librairie avec l'instruction suivante :

```
from math import *


…
```

3. NOMS DES FONCTIONS ET DES VARIABLES

Les noms des fonctions s'écrivent en utilisant le **snake_case** tel que suggéré dans PEP 8. Ce dernier consiste de **minuscules séparées par des barres de soulignement**. Le nom **doit commencer par une lettre**.

Voici quelques exemples :

- `calculer_somme`
- `inverser_matrice`
- `afficher_message`

Voici des exemples de signatures de ces fonctions :

- `def calculer_somme(nombre1, nombre2):`
- `def inverser_matrice(matrice):`
- `def afficher_message(msg):`

La même règle s'applique aux noms de variables :

- `nb_occurences`
- `valeur_max`
- `taille_liste`

À cette syntaxe s'ajoute le besoin d'utiliser des noms significatifs. Autrement dit, le nom d'un programme ou d'une variable doit être descriptif. Voici des contre-exemples pour les fonctions ci-dessus :

- `def calculer(nombre1, nombre2):`
- `def inverser(matrice):`
- `def afficher(msg):`

Notez qu'il est possible que le nom d'une variable commence par une barre de soulignement, mais cet usage doit être justifié et vous sera montré si et quand nécessaire.

4. CONSTANTES

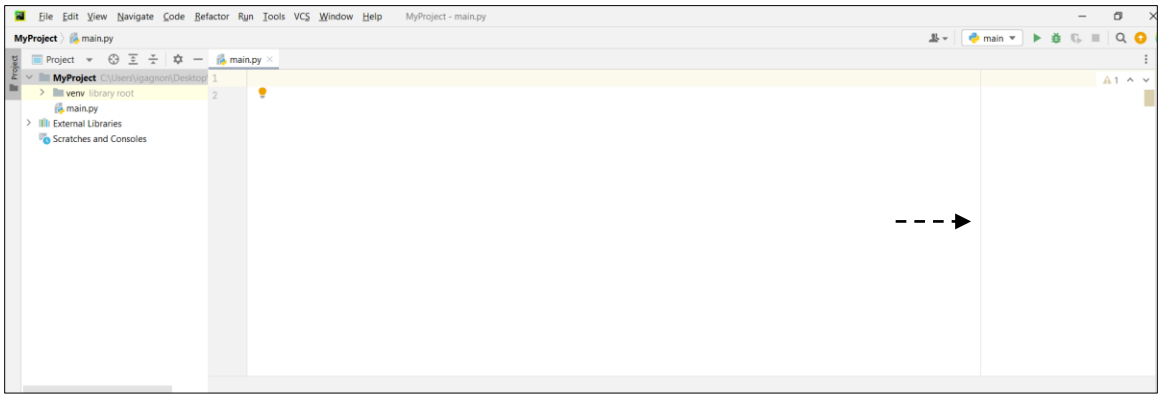
Les constantes sont, par définition, des valeurs qui ne varient pas. Celles-ci s'écrivent en majuscules séparées par des barres de soulignement. Par exemple :

- `MAX_ITER`
- `NB_JOURS`
- `TAILLE_MIN`

Notez que comme c'est le cas pour les variables, les noms des constantes doivent être **pertinents** et **descriptifs**. La constante `MAX_ITER` correspond, par exemple, au nombre maximal d'itérations permises par un algorithme.

5. LONGUEUR MAXIMALE D'UNE LIGNE DE CODE

La **longueur maximale permise d'une ligne de code/commentaire** est de **120 caractères**. Dans PyCharm et Google Colab, cela correspond à la barre verticale à droite de l'éditeur :



6. COMMENTAIRES DE CODE

Comme nous l'avons expliqué à la Section 0, **les en-têtes servent à documenter le code pour les utilisateurs**. Cela consiste à expliquer le ***quoi***. **Les commentaires servent à expliquer le *comment* et s'adressent aux programmeurs.** Ces derniers s'intéressent à la logique derrière les manipulations de données.

Voic un exemple de code avec des commentaires :

```
def nombre_occurences(une_liste, valeur_recherchee):
  """
  Calcule le nombre d'occurences d'une valeur recherchée.

  Arguments :
      une_liste (list): Liste qui contient des nombres à virgule.
      valeur_recherchee (float) : Une valeur recherchée.

  Retourne  :
      (int): Le nombre d'occurences de la valeur recherchée.
  """

  # Initialiser le compteur à zéro
  nb_occurences = 0

  # Parcourir  les éléments de la liste
  for element in une_liste:

    # Incrémenter le compteur d'occurences
    if element == valeur_recherchee:
      nb_occurences += 1

  # Retourner le nombre d'occurences
  return nb_occurences
```

L'exemple précédent contient une abondance de commentaires et en général, mieux vaut plus que pas assez. Il est néanmoins important **d'éviter d'écrire des commentaires qui décrivent la syntaxe**, car les programmeurs n'en ont généralement pas besoin. **Il est préférable de ne conserver les commentaires qui décrivent la logique**. Voici l'exemple repris :

```
def nombre_occurences(une_liste, valeur_recherchee):
  """
  Calcule le nombre d'occurences d'une valeur recherchée.

  Arguments :
      une_liste (list): Liste qui contient des nombres à virgule.
      valeur_recherchee (float) : Une valeur recherchée.

  Retourne  :
      (int): Le nombre d'occurences de la valeur recherchée.
  """

  nb_occurences = 0

  for element in une_liste:

    Incrémenter le compteur d'occurrences
    if element == valeur_recherchee:
      nb_occurences += 1

  return nb_occurences
```

Pas besoin, par exemple, de spécifier que "`nb_occurences = 0`" sert à initialiser le nombre d'occurrences à zéro, car cela est évident. Il est aussi entendu, sans besoin de le mentionner, que la ligne "`for element in une_liste:`" parcourt des éléments d'une liste.

7. INDENTATION

L'indentation joue un rôle fondamental dans le langage Python, car elle délimite les blocs de codes. **Un bloc de code est un ensemble d'instructions qui sont exécutées séquentiellement.** Il s'agit de normes qui sont aussi des exigences syntactiques dont voici les règles à retenir :

- La première ligne ne peut être indentée.
- L'unité d'indentation de base est une tabulation ou 4 espaces blancs.
- Il ne faut pas mélanger les deux types d'indentation. Soit on utilise la tabulation, soit on utilise les espaces blancs, mais pas les deux.
- Toutes les lignes de code à l'intérieur d'une fonction sont décalées.
- Toutes les lignes de code à l'intérieur d'une instruction de répétition (i.e., une boucle) sont décalées.
- Toutes les lignes de code à l'instruction d'une instruction conditionnelle (i.e., un if) sont décalées.