NSI programmation

Spécifier programmes

qkzk

Spécification

De manière générale la **spécification** est un ensemble de d'exigences à satisfaire par un produit ou un service.

En programmation, **spécifier** un programme revient à décrire explicitement ce qu'il doit faire et dans quelles conditions.

Spécifier une fonction

Considérons la fonction suivante :

```
def f(n):
  x = 1
  y = 1
  l = [x]
  k = 0
  while k < n:
    x, y = y, x + y
    l.append(x)
    k += 1
  return l
```

Il est difficile de savoir ce qu'elle fait sans lire ou exécuter le code.

Voici la même fonction mais avec des spécifications convenables :

```
def fibonacci(n):
  '''
  Liste des termes de la suite de Fibonacci
  jusqu'à l'indice n inclus

  @param n: (int) l'indice maximal voulu
  @return: (list) la liste des termes
  '''
  x = 1
  y = 1
  suite_fibonacci = [x]
  indice = 0
  while indice < n:
    x, y = y, x + y
    suite_fibonacci.append(x)
    indice += 1
  return suite_fibonacci
```

Cette fois on dispose d'éléments pour comprendre le code.

1. Sa documentation en haut
2. Des variables explicites

Spécifications attendues dans une fonction

Documentation

La **documentation** (docstring) d'une fonction en Python est constituée d'une chaîne de caractères sur plusieurs lignes :

```
"""
Chaîne sur plusieurs
lignes
"""
```

Documentation

On y précise :

- Ce qui fait la fonction de manière succinte
- Les paramètres d'entrées et leur type : `@param n: (int) l'indice...`
- La sortie et son type.

  Pas de sortie ? `@return: (None)`
- Les conditions d'utilisation et effets de bord : `@CU : La table`
- Eventuellement des tests

Comment accéder à la documentation ?

Une fois qu'une fonction est en mémoire, on peut afficher sa documentation avec `help(nom_fonction)`

```
>>> import math
>>> help(math.cos)

Help on built-in function cos in module math:

cos(x, /)
    Return the cosine of x (measured in radians).
```

Intérêt

- programmer : documenter AVANT d'écire le code donne un objectif clair
- relire : les programmes complexes sont difficiles à comprendre. La documentation simplifie cette étape
- collaborer : travailler à plusieurs demande de l'organisation et une documentation claire est indispensable

Documenter : un attendu

La documentation fait partie des éléments attendus et qui seront toujours évalués.

Si vous ne documentez pas vos fonctions, **vous n'obtiendrez jamais le maximum des points.**

Variables explicites

Afin de rendre le code *lisible* par un être humain, il faut nommer convenablement les objets qu'on emploie.

- On sépare les mots avec des soulignés : `ma_fonction` On peut rencontrer aussi des majuscules entre les mots : `maFonction`
- Les noms de fonctions doivent décrire ce qu'elles font.
  - `def f(n): ...` est mauvais,
  - `def fibonacci(n) ...` est bon.
- Les noms de variables doivent décrire les objets vers lesquels elles pointent :
  - `t = 180` est mauvais
  - `taille = 180` est bon

Spécification et attendus

Nous allons distinguer plusieurs situations :

Lire du code

Vous devez être capable de dire si un code **correspond à sa spécification**.

```
def presenter(liste):
    '''
    Transforme une liste en une chaîne de caractères affichable à l'écran.
    Chaque élément occupe une ligne

    @param liste : (list) la liste d'entrée
    @return: (str) la chaîne affichable à l'écran
    '''
    return '\n'.join(liste)
```

Spécifier du code

Vous devez être capable **d'écrire** la spécification d'une fonction

```
def presenter(liste):
    '''
    documentation à écrire
    '''
    return '\n'.join(liste)
```

Programmer

Vous devez être capable **de programmer** une fonction à partir de sa spécification.

```
def presenter(liste):
    '''
    Transforme une liste en une chaîne de caractères affichable à l'écran.
    Chaque élément occupe une ligne

    @param liste : (list) la liste d'entrée
    @return: (str) la chaîne affichable à l'écran
    '''
    # votre code ici
```

Spécification d'un script

readme.md

Un script (fichier `.py` indépendant) doit aussi être documenté. S'il est hébergé en ligne sur un git, on peut intégrer un fichier `readme.md` qui contient les informations :

README.md

- auteur,
- objectif,
- librairies nécessaires,
- contexte ou énoncé,
- cahier des charges,
- Avancée du projet etc.

Docstring d'un script

- Ce qui fait le script :

```
'''
titre : éléments de la suite de Fibonacci
auteur : qkzk
objectif : Affiche la suite de Fibonacci
etc.
'''

# votre code ici
```

- S'il n'y a pas de fichier README.md, vous pouvez intégrer les consignes et quelques informations

Conventions d'écriture : PEP8

Les conventions d'écritures en Python font partie du projet Python lui même et sont indiquées dans PEP8 (Python Enhancement Proposal 8 : proposition d'amélioration de Python n° 8).

Un résumé de la PEP8 de Python.