Spécifications et Tests

1 Introduction

Exemple :

On donne la fonction suivante ; pouvez-vous indiquer ce qu'elle fait ?

```
def fonction(t):
    m = 0
    for i in range(len(t)):
        if t[i] > t[m]:
            m = i
    return m
```

Sans aucune explication, avec un nom de fonction et de variable trop neutres, il peut être difficile de comprendre le code d'un programmeur !

On ne peut se contenter d'écrire du code, il faut **expliquer** ce que l'on fait, et s'assurer que le programme se **comporte correctement**. Il faut adopter de bonnes pratiques.

2 Documenter une fonction

On commence par les **spécifications** de la fonction, que l'on écrira suite à son entête (dans sa *docstring*), entre triples guillemets.

```
def fonction(t):
    """On ecrit ici les specifications"""
    m = 0
    for i in range(len(t)):
        if t[i] > t[m]:
            m = i
    return m
```

L'utilisateur aura alors accès à ces informations par la fonction `help`.

```
>>> help(fonction)
Help on function fonction in module __main__:

fonction(t)
    On écrit ici les spécifications
```

Spécifications d'une fonction

- On indique **la tâche effectuée** par la fonction ;
- Puis on précise les **paramètres attendus en entrée**, ainsi que **leur type** (entier, chaine de caractères, liste, ...) et on indique le **type de la valeur obtenue** en retour. C'est ce qu'on appelle **prototyper** une fonction ;
- On précise les **contraintes éventuelles** imposées sur les paramètres entrés : on parle de **préconditions** ;
- On indique les **propriétés vérifiées** par la valeur obtenue en retour : on parle de **postconditions**.

Exemple :

cc BY NC SA

Retour sur la fonction donnée en exemple ; on complète sa documentation et on choisit des noms de variable et de fonctions plus explicites.

```
def indice_max_tableau(tab):
    """Cette fonction donne l'indice du
     maximum de la liste tab ;
     on suppose la liste tab non vide."""
    indice_max = 0
    for i in range(len(tab)):
        if tab[i] > tab[indice_max]:
            indice_max = i
    return indice_max
```

Maintenant l'on sait ce que fait cette fonction, et qu'il ne faut pas entrer de liste vide pour paramètre ; mais que se passe-t-il si l'utilisateur le fait quand même ?

```
>>> indice_max_tableau([])
0
```

On obtient une réponse incohérente ! La liste étant vide, elle n'admet pas de maximum, et il n'y a pas d'élément d'indice 0 !

3 Tester la fonction

Exemple :

1. Pour traiter le problème précédent on peut **intégrer un test à la fonction**, qui vérifiera si la liste entrée est vide, et dans ce cas on interrompra le programme, par exemple avec une **assertion**. C'est ce qu'on appelle de la **programmation défensive**.

```
def indice_max_tableau(tab):
    """Cette fonction donne l'indice du
     maximum de la liste tab ;
     on suppose la liste tab non vide."""
    assert len(tab) > 0,"La liste est vide !"
    indice_max = 0
    for i in range(len(tab)):
        if tab[i] > tab[indice_max]:
            indice_max = i
    return indice_max
```

Le message écrit après le test s'affiche si le test est Faux (on a entré une liste vide).

```
>>> indice_max_tableau([])
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "C:\Users\natha\Documents\0. Informatique\1NSI\7.  Spécification et Te
sts\foncMax4.py", line 5, in indice_max_tableau
    assert len(tab) > 0,"La liste est vide !"
AssertionError: La liste est vide !
```

2. On peut aussi tester si le tableau est vide et **retourner une valeur**, ce qui interrompra la fonction, mais laissera la suite du programme se poursuivre. En général, on retourne la valeur **None**.

```python
def indice_max_tableau(tab):
    """Cette fonction donne l'indice du
     maximum de la liste tab ;
     on suppose la liste tab non vide."""
    if len(tab) == 0:
        return None
    else:
        indice_max = 0
        for i in range(len(tab)):
            if tab[i] > tab[indice_max]:
                indice_max = i
        return indice_max
```

Si on entre une liste vide, le programme s'arrête, sans message d'erreur.

Remarque : On va également faire des tests afin de garantir à l'utilisateur d'une fonction qu'elle **réalise bien la tâche prévue** ; on fera les tests en **cours de conception**.

Tests

Là encore, on peut utiliser des **assertions**, ce qu'on appelle des *tests unitaires*. La plupart du temps, ces test sont **écrits avant la fonction** elle-même. On sait ce qu'on attend de notre fonction...
Lors de l'écriture de la fonction, si la fonction ne passe pas l'un des tests prévu, on effectue une **correction**, et on relance la série de tests. En effet, en corrigeant une erreur on peut en introduire une nouvelle !
On ne peut tout tester, il faut trouver **un "bon" ensemble** de tests.

Exemple :

```python
def indice_max_tableau(tab):
    """Cette fonction donne l'indice du
     maximum de la liste tab ;
     on suppose la liste tab non vide."""
    if len(tab) == 0:
        return None
    else:
        indice_max = 0
        for i in range(len(tab)):
            if tab[i] > tab[indice_max]:
                indice_max = i
        return indice_max

assert indice_max_tableau([]) == None
assert indice_max_tableau([-5]) == 0
assert indice_max_tableau([10,20,-30]) == 1
assert indice_max_tableau([7,-3,5,-2,7]) == 0
```

Si tous les tests se déroulent correctement, rien ne se passe ; sinon, un message d'erreur s'affiche indiquant quel test a échoué. Une fois que tous les tests sont passés avec succès, on peut retirer ces tests du code.

On peut remarquer ici un problème qui n'a pas été envisagé, il peut y avoir plusieurs réponses possibles si le maximum est présent plusieurs fois. Ici, la fonction retourne l'indice de la première occurrence du maximum.

CC BY NC SA

Remarques :

• Pour corriger les erreurs, il faut les **localiser** et les **identifier**.

Pour cela, on peut inclure de manière temporaire des **instructions d'affichage** et suivre l'**exécution pas à pas** du code...

• Il existe d'**autres méthodes** pour tester son code, notamment l'utilisation d'un module de Python, appelé **unittest**, mais nous ne les aborderons pas pour le moment.

4 Exercices

Exercice 1 :

1. Vérifier que la fonction `alpha` ci-dessous n'est pas correcte en réalisant les tests mentionnés et corriger le script.
2. Enlever le commentaire devant la dernière assertion et tester à nouveau la fonction. Trouver une solution au problème.

```
def alpha(n):
    """ Cette fonction renvoie la nieme lettre de l'alphabet,
    n etant un entier compris entre 1 et 26"""
    a = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    return a[n]

# Debut des tests
assert(alpha(1) == "A")
assert(alpha(9) == "I")
assert(alpha(26) == "Z")
#assert(alpha(30) == "")
```

Exercice 2 :

Pour la fonction suivante, lui donner un meilleur nom, une chaîne de documentation (docstring), et des tests.

```
def f(t):
    s = 0
    for i in range(len(t)):
        s += t[i]
    return s
```

Exercice 3 :

On veut écrire une fonction qui prend en paramètre un mot et renvoie True si le mot commence et se termine par la même lettre, et False sinon.

Recopier le code ci-dessous et le compléter :

CC BY NC SA

```python
def deb_mot_egal_fin(mot):
    pass

assert deb_mot_egal_fin("rouler") == True
assert deb_mot_egal_fin("chien") == False
assert deb_mot_egal_fin("") == None
assert deb_mot_egal_fin("Ana") == True
assert deb_mot_egal_fin(" ici") == False
```

Exercice 4 :

On donne la fonction ci-dessous. Effectuer les tests nécessaires pour s'assurer qu'elle fait ce qui est indiqué, en expliquant le choix de vos tests.

```python
def division(a,b):
    """La fonction doit renvoyer la quotient et le reste de
    la division euclidienne de a par b ;
    a et b sont des entiers positifs, avec b non nul"""
    r = a
    q = 0
    while r >= b:
        q +=1
        r = r - b
    return (q,r)
```


cc BY NC SA