Cours NSI 1ère:

Mise au point de programmes

1 Introduction

Quand on développe des programmes, la majeure partie du temps est passée à lire ou relire du code (et non pas à l'écrire), qu'il s'agisse de comprendre comment une fonctionnalité marche ou pourquoi celle-ci ne marche pas.
On peut facilement en conclure qu'il faut certes qu'un code fonctionne, (et on pourra s'interesser à des méthodes pour rendre son code plus fiable) mais il faut surtout qu'il soit lisible, le plus facilement possible. Nous allons nous intéresser à 2 points principaux pour rendre le code plus utilisable et réutilisable.

2 Capacités

- Prototyper une fonction.
- Décrire les préconditions sur les arguments.
- Décrire des postconditions sur les résultats.
- Utiliser des jeux de tests.

3 Les commentaires

Les commentaires doivent être, autant que possible, des phrases complètes. L'objectif des commentaires n'est pas de décrire ligne par ligne le fonctionnement d'un programme mais d'indiquer les grandes étapes de ce que fait un programme/une fonction. Ou de détailler le fonctionnement d'une partie qui est particulièrement compliquée à comprendre.

Un exemple avec le calcul du pgcd:

```
#Récupération des 2 valeurs
a=eval(input("Valeur de a ?"))
b=eval(input("Valeur de b ?"))
# Utilisation de l'algorithme d'Euclide pour trouver le pgcd
while a!=b:
    d=abs(b-a)
    b=a
    a=d
#Résultats
print("pgcd=",d)
if d==1:
    print("Les deux entiers sont premiers entre eux.")
```

4 Nommage des variables et fonctions

La norme python voudrait que toutes les variables et fonctions soient nommées avec des caractères en minuscule avec le caractère _ séparant chaque mot.
Par exemple: `ma_variable_de_test`
La même chose est proposée pour les constantes mais avec les caractères qui sont tous en majuscule. Une constante est une variable à laquelle on donne une valeur qui ne changera pas une fois le programme lancé.
Par exemple: `PI` ou `HAUTEUR_ECRAN`
Néanmoins il éxiste d'autres "méthodes" de nommage de variable:

- maVariableDeTest (tout attaché avec des majuscules pour séparer les mots).
- MaVariableDeTest (pareil mais une majuscule au premier mot en plus).

Bien qu'il ne vous sera pas demandé de respecter les méthodes de nommage demandée par python, il peut être interessant d'en retenir au moins une et de s'en servir, ainsi tout le code sera écrit de la même manière.
Un exemple de ce qu'il faudrait éviter en terme de nommage:

```
#Initialisation
MonTableau = [2,3,6,90]
VALEUR_MIN = MonTableau[0]
valeurMax = MonTableau[0]
#Parcours et recherche des valeurs min et max dans le tableau
for mon_nombre in MonTableau:
    if mon_nombre < VALEUR_MIN:
        VALEUR_MIN = mon_nombre
    if mon_nombre > valeurMax:
        valeurMax = mon_nombre
```

5 Jeux de test

Il arrive que le programme que l'on vient d'écrire ne fonctionne pas correctement, pas du tout ou bien qu'un problème ne survienne que dans certains cas précis.
Dans ces cas il faut identifier la source du problème avant de pouvoir le régler.

Afin de réduire l'apparition de ce type de problème quand on écrit un programme, on peut y joindre des jeux de tests. Ces tests servent à vérifier que le programme ou la fonction renvoie bien les valeurs attendues. De plus si l'on constate une erreur dans un cas précis, on peut l'ajouter aux jeux de tests une fois qu'on a trouvé sa cause et qu'on a corrigé notre code.

L'intérêt de ces jeux de tests est double:

- Permettre de vérifier que notre code fait ce que l'on veut une fois qu'on a fini de l'écrire.
- Permettre de vérifier que, si on modifie notre code, celui-ci fonctionne toujours de la même manière qu'avant.

Afin d'écrire les tests on peut utiliser le mot clé `assert` suivi par un test, si le résultat n'est pas True, le programme affichera une erreur.
Quand on écrit un jeu de tests, on veut en général tester un cas spécifique où tout est sensé bien se passer mais aussi les cas aux limites, car ce sont ceux qui ont le plus de chance de ne pas fonctionner :

- Pour une fonction qui utilise les tableaux, tester un tableau vide, un tableau ou tous les éléments sont identiques, ...
- Pour une fonction qui utilise des chaînes de caractères, tester une chaîne vide, une avec un seul caractère, ...
- Pour une fonction qui fait des opérations calculatoires, tester avec 0, des nombres négatifs, ...

```
1  def est_pair(chiffre):
2      return chiffre % 2 == 0
3
4  assert est_pair(4) == True
5  assert est_pair(1) == False
6  assert est_pair(9234) == True
7
8  def au_carre(chiffre):
9      return chiffre**2
10
11 assert au_carre(5) == 25
12 assert au_carre(-5) == 25
13 assert au_carre(1) == 1
14 assert au_carre(0) == 0
```

L'utilisation de tests ne garanti pas que le code n'a pas d'erreur. Néanmoins, cela réduira les chances qu'une erreur survienne, d'autant plus si vous modifiez votre fonction.
Prenons l'exemple suivant:

```
def division(num,den):
    return num/den


assert division(0,2) == 0
assert division(-3,1) == -3
assert division(9,3) == 3
assert division (1,0.5) == 2
```

On risque d'avoir une erreur si on mets le dénominateur à 0 malgré nos tests. On peut gérer l'erreur de la manière suivante.

```
def division(num,den):
    if den != 0:
        return num/den
    else:
        return "On ne peut pas diviser par 0"

assert division(5,0) == "On ne peut pas diviser par 0"
assert division(0,2) == 0
assert division(-3,1) == -3
assert division(9,3) == 3
assert division (1,0.5) == 2
```

Comme vous pouvez le voir, suite à la gestion de l'erreur rajoutée, on a aussi mis en place un nouveau test.

Exercice

Proposez un jeu de test pour la fonction suivante:

```
def valeur_minimum(tableau):
    min = tableau[0]
    for valeur in tableau:
        if min > valeur:
            min = valeur
    return min
```

6 Documentation/Spécification

Une dernière étape consiste à écrire la documentation d'une fonction. Il est important de l'écrire, car celle-ci permet d'identifier le fonctionnement de la fonction sans avoir à en lire tout le code.
Pour documenter une fonction en python, on utilise ce que l'on appelle un `docstring`. Il s'agit d'une chaîne te texte que l'on écrit au début d'une fonction, celle-ci commence et termine par 3 caractère `"`.
La docstring doit indiquer les types attendus en entrée et en sortie de fonction.

On peut même écrire ces commentaires avant de commencer à développer la fonction, afin de bien fixer son fonctionnement.

```
def division(num,den):
    """ Renvoie un nombre si on divise par autre chose que 0, et
    renvoie "On ne peut pas diviser par 0" sinon.

    paramètres:
    num - un nombre
    den - un nombre"""
    if den != 0:
        return num/den
    else:
        return "On ne peut pas diviser par 0"
```

On peut accéder à la docstring d'une fonction de la manière suivante:
`print(division.__doc__)`
On peut utiliser cette manière d'afficher la documentation non seulement sur les fonctions que l'on à écrit mais aussi sur les fonctions de python ou des bibliothèques que l'on utilise.
par exemple:
`print(print.__doc__)` Affichera:

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.
    stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

7 Exercices

Exercice 1

Soit les fonctions suivantes, prenez le temps de comprendre ce qu'elles font:

```
def fonction_1(a,b,c,d):
    return(a+b+c+d)

def Fonction2(A,B,C,D):
    return(fonction_1(A,B,C,D)/4)
```

Ce code est fonctionnel, néanmoins il pourrait être amélioré.
Corrigez les fonctions pour avoir une documentation et des noms corrects, proposez un jeu de test pour chaque fonction.

Exercice 2

Prenez le temps d'étudier le code suivant et de comprendre ce qu'il fait.

```
def fonction1(a,b):
  coefficient=10
  c=coefficient*coefficient*a*b
  print("L'aire d'un rectangle de mesure",a,"et",b,"qui subit un
    agrandissement de coefficient",coefficient,"est",c)

#volume d'un parallélépipède de coté a,b,c qui subit un
    agrandissement de coefficient d
def fonction2(a,b,c,d):
  return(a*b*c*d*d*d)
```

Que dois-je faire pour avoir l'aire d'un rectangle de coté 5 et 6, agrandit 10 fois. Est-ce que je peut avoir, avec les fonctions ci-dessus l'aire d'un rectangle de coté 5 et 6, mais agrandit 5 fois ? .....
Même question pour un pavé de cotés 5, 2 et 3: .....

.....
Ré-écrivez ce code pour qu'il contienne une documentation ainsi que des jeux de tests.

Exercice 3

Soit la fonction `tri_de_tableau(tableau)`.
Le docstring de la fonction est accessible en écrivant:
`print(tri_par_insertion.__doc__)`

```
tri_par_insertion(tableau)
tableau est un tableau de valeurs entières ou flottantes.

Modifie le tableau passé en entrée afin qu'il soit trié.
Le tri marche sur des nombres (entiers et/ou flottants).
Ne renvoie rien si le tableau ne contient que des nombres.
Renvoie le texte "Erreur" si le tableau contient d'autres types de
    données (texte, booléens, ...).
```

Après avoir lu le docstring, répondez aux questions suivantes:
Comment peut-on tester que la fonction fait bien ce qui est écrit ci-dessus ?
Que faut-il tester ?
Proposez un jeu de test.

Exercice 4

Un cas concret:
On à développé une application d'envoi de messages. Afin de ne pas surcharger nos serveurs nous avons créé une fonction permettant de vérifier que l'envoi des pièces-jointes corresponds aux critères suivants:

- On peut envoyer au maximum 10 pièces jointes dans un message.
- Le poids total des pièces jointes ne doit pas depasser 30 Mo.
- Le poids maximal d'une seule pièce jointe est de 25 Mo.

Notre fonction est la suivante: `verif_avant_envoi(tableau)`.
La fonction renvoie True si les conditions sont respectées, False sinon.
Pour simplifier l'exercice, considéront que le tableau contient le poids de chaque pièce jointe en Mo (par exemple [4,0.5,12] représente 3 pièces jointes de 4, 0.5 et 12 Mo respectivement).
Quels sont les tests que vous devriez faire pour pouvoir vérifier :

- Que l'on peut envoyer un message qui respecte toutes ces conditions.
- Que si une des conditions n'est pas respectée, le message ne peut pas être envoyé.