NSI Terminale	TD Messages d'erreurs et gestion des erreurs	Séquence 4 Programmation

Exercice 1 : QCM

1) Quel message d'exception s'affiche si on tente d'exécuter le code suivant ?

```
a=1
for i in range (3):
    print("i={},a={}".format(i,a)
    a=2*a
```

a. NameError b. SyntaxError **c.** IndexError **d.** IndentationError

2) À quel moment l'exception précédente est-elle levée ?

a. avant l'exécution **b.** pendant l'exécution

3) Quel message d'exception s'affiche si on tente d'exécuter le code suivant ?

```
v=1
while v<100:
    if v%7==0:
        print(v,"est un multiple de 7")
         else:
        print(v,"n'est pas un multiple de 7")
    v=v+1
```

a . NameError **b.** SyntaxError **c.** IndexError d. IndentationError

4) A quel moment l'exception précédente est-elle levée ?

a. avant l'exécution **b.** pendant l'exécution

5) On considère le code suivant qui, étant donné un terme u_n de la suite de Syracuse, renvoie le terme u_{n+1}.

```
def syracuse(un:int)->int:
    if un%2==0:
        return un//2
    else:
        return 3 * un + 1
```

Le test suivant est proposé :
assert syracuse(32)==16.

Parmi ces autres tests, lequel semble le plus urgent à ajouter ?

a. ***assert*** syracuse(16)==8 **b.** ***assert*** syracuse(0)==0 **c.** ***assert*** syracuse(3)==10

Exercice 2 : Tests couvrants

On dispose de la fonction *tous_differents(lst)* qui indique si les éléments de la liste *lst* sont tous différents et dont la première ligne est la suivante : ***def*** tous_differents(lst :list)->bool :
À l'aide d'assertions, écrire des tests couvrants pour cette fonction.

assert type(lst)==list
assert type(tous_differents([1,2]))==bool

Exercice 3 :

On souhaite associer une valeur numérique à une chaîne de caractères. La valeur associée à la chaîne est la somme des valeurs associées à chacun des caractères qui la composent. Aux lettres non accentuées de A à Z, qu'elles soient en minuscules ou en majuscules, on associe leur numéro dans d'ordre dans l'alphabet, de 1 à 26. À tous les autres symboles (lettres accentuées, chiffres, espaces, ponctuations), on associe la valeur 0.
On rappelle que *ord(x)* est le code numérique associé à la lettre passée en paramètre.
La chaîne *string.ascii_uppercase* est définie dans le module string et vaut "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

1) Pour chacune des propositions suivantes, proposer un test qui montre que la fonction ne répond pas au cahier des charges de l'énoncé.

```
#proposition1
def valeur1(chaine):
    s=0
    for c in chaine:
        if c>="A" and c<="Z":
            s = s +ord(c) -ord('A') +1
    return s
```

assert valeur1("a")==1

```
# proposition2
def valeur2(chaine):
    s=0
    for c in chaine.upper():
        if c>="A" and c<"Z":
            s = s +ord(c) -ord('A') +1
    return s
```

assert valeur2("Z")==26

```
#proposition3
import string
def valeur3(chaine):
    s=0
    for c in chaine.upper():
        if c in string.ascii_uppercase:
            s=s+string.ascii_uppercase.index(c)
    return s
```

assert valeur3("a")==1

2) Proposer un ensemble complet de tests pour la fonction que l'on veut créer (on nomme cette fonction *valeur*).

```
assert valeur1("a")==1
assert valeur2("Z")==26
assert valeur3("a")==1
```

3) Écrire le code de cette fonction et vérifier qu'elle passe tous les tests proposés en 2).

```
import string

def valeur3(chaine):
    s=0
    for c in chaine.upper():
        if c in string.ascii_uppercase:
            s=s + string.ascii_uppercase.index(c) + 1
    return s
```

Exercice 4 :

Le produit scalaire de deux vecteurs de l'espace $\vec{u}$ et $\vec{v}$ de coordonnées respectives $(u1; u2; u3)$ *et* $(v1; v2; v3)$ est le nombre réel $u1v1 + u2v2 + u3v3$.
Le code suivant permet de calculer des produits scalaires. Il est testé sur des vecteurs de l'espace choisis au hasard :

```
import random
def scalaire (v1:tuple,v2:tuple)->float:
    """ calcule le produit scalaire des deux vecteurs v1 et v2"""
    s=0
    for k in range(len(v1)):
        s = s + v1[k] * v2[k]
    return s
def random_vect(n:int)->tuple:
    """ Génère un vecteur de taille n contenant des entiers non nuls entre -10 et 10"""
    v = []
    for i in range(n):
        val = random.randint(-10,10)
        if val != 0:
            v.append(val)
    return tuple(v)
def main():
    for i in range(20):
        print( scalaire(random_vect(3),random_vect(3)))
main()
```

Prises séparément les deux premières fonctions semblent opérationnelles, comme le montrent les appels suivants effectués dans la console :

```
>>> scalaire((1,2,1),(-1,3,-2))
3
>>> random_vect(3)
(-10, -10, -9)
>>>
```

Pourtant lorsqu'on exécute la fonction main(), le programme n'affiche que quelques produits scalaires (ici -12, -56 et 29) et plante :

```
*** Console de processus distant Réinitialisée ***
-12
-56
29
Traceback (most recent call last):
  File "C:\Users\Laure Guillaud\Dropbox\0.Spé NSI\Terminale-NSI\4-Langage et programmation\S4-Gestion_des_bugs\ex 4.py", line 19, in <module>
    main()
  File "C:\Users\Laure Guillaud\Dropbox\0.Spé NSI\Terminale-NSI\4-Langage et programmation\S4-Gestion_des_bugs\ex 4.py", line 18, in main
    print( scalaire(random_vect(3),random_vect(3)))
  File "C:\Users\Laure Guillaud\Dropbox\0.Spé NSI\Terminale-NSI\4-Langage et programmation\S4-Gestion_des_bugs\ex 4.py", line 6, in scalaire
    s = s + v1[k] * v2[k]
IndexError: tuple index out of range
```

1) En analysant le Traceback donné ci-dessus, indiquer quel est le type d'exception qui a été levée et sur quelle ligne de code.
C'est une indexError à la ligne 6
2) Sur la ligne en question, qu'est-ce qui pourrait avoir provoqué l'erreur ?
3) Proposer d'ajouter un affichage print(…) à un endroit du programme pour essayer de mettre en évidence la nature du problème, puis réexécuter le code (on ne veut pas résoudre le pb mais seulement mettre en évidence sa nature).
4) Expliquer l'erreur. Pourquoi ne se produit-elle pas toujours au même moment ?
5) Proposer un correctif à ce bug.

Exercice 5 :
On donne le code d'un programme, non commenté où les fonctions ne sont pas spécifiées et comportant un grand nombre d'erreurs (9 en tout).

Ouvrir le fichier *Ex 5.py* et trouver les 9 erreurs commises. Vous les corrigerez en mettant un commentaire à l'endroit de l'erreur et en indiquant le type de l'erreur commise.