Exercice 1 : Vrai/Faux

1) Si on exécute une fonction sans suivre sa spécification, une erreur se produit ?
2) Un commentaire n'est pas lu par un interpréteur python ?
3) Trop de commentaires ralentissent l'exécution du programme ?
4) Le mot *assert* suivi d'une condition booléenne *True* stoppe l'exécution de la fonction ?
5) L'exécution de l'instruction $if\ x > 0\ and\ if\ x < 5$ provoque-t-elle une erreur ?
6) $assert\ x = 0$ est-il correct ?

1) Non, pas obligatoirement (cf exemple du cours sur la somme de deux éléments)
2) Oui, l'exécution se fait sans tenir compte des commentaires
3) Non, car les commentaires ne sont pas pris en compte
4) Non, c'est False qui stoppe l'exécution
5) Oui, l'instruction and doit être entourée de 2 booléens, or les 2 if ne sont pas des booléens
6) Non, il faut une condition $assert\ x == 0$

Exercice 2 :

Prévoir si l'on observe des effets de bords avec les codes ci-dessous :

```
m=[1,2]
def f():
    liste=m[:]
    liste[-1]=3
    return liste
f()
print(m)

m=[[1,1],[2,2]]
def f2():
    liste=m[:]
    liste[-1][-1]=3
    return liste
f2()
print(m)
```

Pas d'effet de bord avec la fonction f (car copie superficielle : l'adresse de chaque élément de la liste est référencée dans liste)

En revanche, l'appel de f2 conduit à un effet de bord car on modifie une sous-liste qui a été copiée de manière superficielle et donc chaque valeur de m est copiée par référence. Avec une copie en profondeur, par d'effet de bord.

```
m=[[1,1],[2,2]]
def f3():
    liste=copy.deepcopy(m)
    liste[-1][-1]=3
    return liste
f3()
print(m)
```

Exercice 3 :

La fonction décrite ci-dessous calcule le n-ième nombre de Fibonacci.
Ces nombres notés u_n sont définis par $u_0 = 0, u_1 = 1$ et $u_n = u_{n-1} + u_{n-2}$

```
liste1=[0,1]
def fibo(n):
    for i in range(2,n+1):
        liste1.append(liste1[i-1]+liste1[i-2])
    return liste1[n]
```

1) Faire l'appel fibo(3) puis fibo(5) et commenter
2) Proposer un programme qui évite le problème soulevé

```
corrigé
liste1=[0,1]
def fibo1(n):
    for i in range(2,n+1):
        liste1.append(liste1[i-1]+liste1[i-2])
    return liste1[n]

print(liste1)#[0,1] la liste n'est pas encore modifiée
print(fibo1(3))#la valeur retournée est correct
print(liste1)#la liste est modifié
print(fibo1(5))#on rajoute des éléments à la mauvaise place
en partant de la liste modifiée
print(liste1)#donc c'est faux au 2e appel


liste2=[0,1]
def fibo2(n):
    copie=liste2[:]
    for i in range(2,n+1):
        copie.append(copie[-1]+copie[-2])
    return copie[n]

print(liste2)#[0,1] la liste n'est pas et ne sera pas
modifiée
print(fibo2(3))#la valeur retournée est correct
print(liste2)#la liste n'est pas modifiée et on repart bien
avec une bonne initialisation
print(fibo2(5))#les éléments sont placés au bon endroit
print(liste2)#OK
```

Exercice 4 :

On donne ci-dessous une fonction qui doit prendre en argument deux entiers naturels m et n non nuls et qui renvoie le quotient q et le reste r tels que : $m = nq + r$. Sans modifier les instructions de cette fonction :

- Proposer une spécification
- Une assertion permettant de tester les arguments proposés (entier naturels non nuls)
- Une assertion permettant de vérifier la propriété $m = nq + r$

```
def divise(m,n):
    q,r=0,m
    while r>=n:
        r=r-n
        q=q+1
    return q,r

print(divise(5,2))
```

corrigé

```
def divise2(m,n):
    """cette fonction renvoie m//n (quotient) et m%n (le
reste)
    c-à-d le quotient q et le reste r de la division de m/n
    tels que m=q*n+r à chaque itération
    il faut surtout  non nul"""
    assert n!=0
    q,r=0,m
    while r>=n:
        r=r-n
        q=q+1
        assert m==q*n+r
    return q,r

print(divise2(5.5,2.2))
```

Exercice 5 :

Ecrire une fonction *ajoute* qui prend en paramètre deux listes de nombres de même longueur et renvoie une nouvelle liste en effectuant la somme terme à terme des éléments des deux listes. Utiliser l'instruction *assert* pour vérifier les longueurs des listes.

corrigé

```
def ajout(L1,L2):
    """ si les listes L1 et L2 sont
    de même longueur, cette fonction renvoie
    une liste donnant dont chaque élément est la somme terme
    à terme des éléments de L1 et L2"""
    assert len(L1)==len(L2)
    n=len(L1)
    L=[]
    for i in range(n):
        L.append(L1[i]+L2[i])
    return L
```

Exercice 6 :

La fonction moyenne prend en paramètre une liste de nombres qui est non vide. Aidez-vous des deux commentaires pour corriger le code de cette fonction.

```
def moyenne(liste):
    n=len(liste)
    #initialisation d'une liste avec la 1e valeur de liste
    lisse=liste[0]
    for i in range(n-1):
        #moyenne de deux éléments consécutifs
        m=(liste[i]+liste[i+1]/2)
        lisse.append(m)
    return lisse
```

corrigé

```
def moyenne2(liste):
    """fonction qui renvoie une nouvelle liste lisse
    chaque élt étant la moyenne de deux valeurs consécutives
de liste
    sauf lisse[0]=liste[0]"""
    n=len(liste)
    #initialisation d'une liste avec la 1e valeur de liste
    lisse=[liste[0]]#pb si on initiale avec un int
    for i in range(n-1):
        #moyenne de deux éléments consécutifs
        m=(liste[i]+liste[i+1])/2#pb de () si on veut la moy
        lisse.append(m)
    return lisse
```

Exercice 7 :

La fonction *lissage* prend en paramètre une liste de nombres qui est non vide. Utiliser le commentaire pour corriger le code de cette fonction

```
def lissage (liste):
    n=len(liste)
    lisse=[liste[0]]
    for i in range(1,n-1):
        #ajoute à la liste lisse la moyenne des 3 élts
consécutifs
        lisse.append((sum(liste[i-1:i+1]))/3)
    lisse.append(liste[n-1])
    return lisse

corrigé
def lissage2 (liste):
    n=len(liste)
    lisse=[liste[0]]
    for i in range(1,n-1):
        #ajoute à la liste lisse la moyenne des 3 élts
consécutifs
        lisse.append((sum(liste[i-1:i+2]))/3)#problème de
slicing
    lisse.append(liste[n-1])
    return lisse
```

Exercice 8 :

La fonction qui suit prend en paramètre un entier *n* et doit renvoyer *True* si *n* est un nombre pair et *False* sinon.

```
def pair(n):
    """n est un entier
    renvoie True si n est pair, sinon renvoie False"""
    if n%2:
        return True
    else:
        return False
```

En utilisant cette fonction, on obtient exactement le résultat contraire à celui qui est prévu. Expliquer pourquoi et écrire une fonction qui satisfait la spécification.

Corrigé :

```
def pair(n):
    """n est un entier
    renvoie True si n est pair, sinon renvoie False"""
    if not n%2:
        return True
    else:
        return False
```

Exercice 9 :

La fonction f prend en paramètre un nombre (entier ou flottant)

```
def f(x):
    return 2*x,3*x,5*x
```

1) Quel est le type du résultat renvoyé par la fonction f ?
2) Après l'instruction $a, b, c = f(2)$, quelle est la valeur de c ?

Corrigé :

Cette fonction renvoie une séquence de données de type tuple (ici (4,6,10) et $c = 10$