DS3 Partie informatique - Corrigé

Exercice 1.

Dans toute la correction, les parties en italiques sont des commentaires explicatifs.

1. *On initialise une somme à 1, et on ajoute à chaque étape le terme suivant de la somme à l'aide d'une boucle bornée.*

```
def somme(n):
    res = 0
    for k in range(1, n+1):    # k va de 1 à n
        res = res + (-1)**k / k**2
    return res
```

2. *On doit importer la fonction exponentielle du module* `math`*, par exemple. Le reste est simplement une structure conditionnelle !*

```
from math import exp
def f(x):
    if x == 0:
        return 1
    else:        # cette ligne est en fait inutile
        return (exp(x)-1)/x
```

Exercice 2.

1. *On doit garder en mémoire l'indice du maximum au lieu de sa valeur. On obtient le code suivant :*

```
def ind_maxi(L):
    indice = 0
    n = len(L)
    for i in range(1,n):
        if L[i] > L[indice]:
            indice = i
    return indice
```

2. *Le plus simple est ici d'utiliser du « slicing » de liste :*

```
def enleve_element(L, i):
    return L[0:i] + L[i+1:len(L)]
```

3. L'opération `+` permet de **concaténer** deux listes, c'est-à-dire que cette opération renvoie une liste qui contient d'abord tous les éléments de la première liste, puis tous les éléments de la seconde.
   Par exemple, l'instruction `[4] + [11, 30]` renverra la liste `[4, 11, 30]`.
4. *À chaque étape, on regarde l'indice du plus grand élément puis on le retire de la liste. Entre ces deux opérations, on doit rajouter le plus grand élément au début de la liste* `Ltrie`*. Et on répète cette opération jusqu'à avoir fini, c'est-à-dire jusqu'à ce qu'on ait retiré tous les éléments de la liste d'origine !*

```
def tri(L):
    Ltrie = []
    n = len(L)
    while L != []:
        i = ind_maxi(L)
        Ltrie = [L[i]] + Ltrie
        L = enleve_element(L, i)
    return(Ltrie)
```

Exercice 3. Ensemble de Mandelbrot

1. Pour $c = 0$, on peut montrer par récurrence que pour tout $n \in \mathbb{N}$, on a $z_n = 0$.
   En effet, cette propriété est vraie au rang 0 puisque $z_0 = 0$.
   Soit $n \in \mathbb{N}$ tel que $z_n = 0$. Alors on a $z_{n+1} = z_n^2 + c = 0^2 + 0 = 0$. La propriété est donc héréditaire.
   Ainsi, pour $c = 0$, on obtient la suite constante égale à 0 : elle est bien bornée.
   On en déduit que $c = 0$ appartient bien à l'ensemble de Mandelbrot.
2. Pour $c = 1 + i$, on a :
   - $u_0 = 0$
   - $u_1 = u_0^2 + c = 0^2 + 1 + i = 1 + i$
   - $u_2 = u_1^2 + c = (1+i)^2 + 1 + i = 1 + 2i + i^2 + 1 + i = 2 + 3i - 1 = 1 + 3i$
   - $u_3 = (1+3i)^2 + 1 + i = 1 + 6i + 9i^2 + 1 + i = 2 + 7i - 9 = -7 + 7i$

   On a $|u_0| = 0$, $|u_1| = \sqrt{2}$, $|u_2| = \sqrt{10}$ et $|u_3| = \sqrt{98} = 7\sqrt{2}$.
   La suite des modules ne semble pas bornée (je peux même conjecturer qu'elle diverge vers $+\infty$) : je ne pense pas que $c = 1 + i$ appartienne à l'ensemble de Mandelbrot.
3. *Il s'agit simplement d'appliquer la formule du module.*

```
from math import sqrt
def module(z):
    mod = sqrt(z.real**2 + z.imag**2)
    return mod
```

4. *On doit ici calculer successivement tous les termes de la suite, puisqu'elle est définie par récurrence.*

```
def suite_z(n, c):
    z_k = 0
    for _ in range(n):
        z_k = z_k**2 + c
    return z_k
```

5. *On vérifie que chacun des termes est bien de module inférieur à 2, à l'aide des fonctions codées précédemment. Dès que j'obtiens un module trop grand, je peux renvoyer* ***`False`****. Si j'arrive au bout en ayant réussi chaque test, je peux renvoyer* ***`True`****.*

```
def verif(c):
    for n in range(101):
        if module(suite_z(n, c)) >= 2:
            return False
    return True
```

   Une version plus optimale, qui permet de ne pas recalculer tous les termes de la suite à chaque vérification :

```
def verif_opti(c):
    z = 0
    for n in range(101):
        if module(z) >= 2:
            return False
        else:
            z = z**2 + c
    return True
```