Correction - TP 1

Révisions : Listes et programmation

Exercice 1

Voici le programme demandé :

```
a = float(input('Donner une valeur non nulle pour a : '))
b = float(input('Donner une valeur pour b : '))
c = float(input('Donner une valeur pour c : '))
x = b**2-4*a*c
if x>0 :
    print('P admet deux racines.')
elif x==0 :
    print('P admet une racine.')
else :
    print('P n admet pas de racine.')
```

Exercice 2

1. Voici une proposition de programme :

```
def f(x) :
    if x > 0 :
        return 1/x
    else :
        return 0
```

2. (a) Ce programme définit la fonction suivante :

$$g(x) = \begin{cases} -2 & \text{si } x < -1, \\ 2x & \text{si } -1 \leq x \leq 1, \\ 2 & \text{si } x > 1. \end{cases}$$

(b) Voici une proposition :

```
def g(x) :
    if x < -1 :
        y = -2
    elif x<= 1 :
        y = 2*x
    else :
        y = 1
    return y
```

Exercice 3

1. (a) Voici une proposition pour le calcul de S_n :

```
n = int(input('Donner n : '))
S = 0
for i in range(1,n+1) :
    for j in range(1,n+1) :
        S = S + i/2**j
print(S)
```

(b) Voici une proposition pour le calcul de T_n :

```
n = int(input('Donner n : '))
T = 0
for i in range(1,n+1) :
    for j in range(i,n+1) :
        T = T + i**3/(j*(j+1))
print(T)
```

2. Pour S_n, on a :

$$\begin{aligned} S_n &= \left(\sum_{i=1}^{n} i\right) \times \left(\sum_{j=1}^{n} \left(\frac{1}{2}\right)^j\right) = \left(\sum_{i=1}^{n} i\right) \times \left(\sum_{j=0}^{n} \left(\frac{1}{2}\right)^j - 1\right) = \frac{n(n+1)}{2} \times \left(\frac{1-\left(\frac{1}{2}\right)^{n+1}}{1-\frac{1}{2}} - 1\right) \\ &= \frac{n(n+1)}{2} \times \left(2 - 2\left(\frac{1}{2}\right)^{n+1} - 1\right) = \frac{n(n+1)}{2} \times \left(1 - \left(\frac{1}{2}\right)^{n}\right) \end{aligned}$$

Pour T_n, on a :

$$\begin{aligned} T_n &= \sum_{j=1}^{n} \left(\sum_{i=1}^{j} \frac{i^3}{j(j+1)}\right) = \sum_{j=1}^{n} \frac{1}{j(j+1)} \left(\sum_{i=1}^{j} i^3\right) = \sum_{j=1}^{n} \frac{1}{j(j+1)} \times \frac{j^2(j+1)^2}{4} = \sum_{j=1}^{n} \frac{j(j+1)}{4} \\ &= \frac{1}{4}\sum_{j=1}^{n} j^2 + \frac{1}{4}\sum_{j=1}^{n} j = \frac{n(n+1)(2n+1)}{24} + \frac{n(n+1)}{8}. \end{aligned}$$

Exercice 4

1. Voici un programme pour calculer u_n :

```
def suiteu(n) :
    u = 1
    for k in range(n) :
        u = u**2 + 1
    return u
```

2. Voici un programme pour obtenir le rang du premier dépassement d'un réel a par la suite (u_n) :

```
def rang(a) :
    u = 1
    n = 0
    while u < a :
        u = u**2 + 1
        n = n+1
    return n
```

Exercice 5

Voici le programme demandé :

```python
def valabs(x) :
    if x < 0 :
        return -x
    else :
        return x

def rang() :
    u = 1/2
    n = 0
    while valabs(u-1) > 10**(-3) :
        u = (2*u) / (u+1)
        n = n+1
    return n
```

Exercice 6

1. Voici la procédure pour calculer u_n :

```python
def suiteu(n) :
    u = 1
    for k in range(n) :
        u = 2*k*u + 3
    return u
```

2. Voici la procédure pour calculer v_n :

```python
def suitev(n) :
    vavant = 1
    vapres = -2
    for k in range(n) :
        aux = vapres
        vapres = 2*vapres - vavant
        vavant = aux
    return vavant
```

3. Voici la procédure pour calculer a_n et b_n :

```python
def suitesab(n) :
    a = 1
    b = 2
    for k in range(n) :
        aaux = a
        baux = b
        a = aaux**2 / (aaux + baux)
        b = baux**2 / (aaux + baux)
    return a,b
```

Exercice 7

1. Montrons que $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ sont adjacentes :

   - $(u_n)_{n\in\mathbb{N}^*}$ est croissante :

$$u_{n+1} - u_n = \sum_{k=1}^{n+1} \frac{1}{k^2} - \sum_{k=1}^{n} \frac{1}{k^2} = \sum_{k=1}^{n} \frac{1}{k^2} + \frac{1}{(n+1)^2} - \sum_{k=1}^{n} \frac{1}{k^2} = \frac{1}{(n+1)^2} \geq 0.$$

- $(v_n)_{n\in\mathbb{N}^*}$ est décroissante :

$$\begin{aligned} v_{n+1}-v_n &= \left(u_{n+1}+\frac{1}{n+1}\right)-\left(u_n+\frac{1}{n}\right)=\frac{1}{(n+1)^2}+\frac{1}{n+1}-\frac{1}{n} \\ &= \frac{n+n(n+1)-(n+1)^2}{n(n+1)^2}=\frac{-1}{n(n+1)^2}\leq 0. \end{aligned}$$

- $v_n-u_n=\left(u_n+\frac{1}{n}\right)-u_n=\frac{1}{n}\underset{n\to+\infty}{\longrightarrow}0.$

On a ainsi démontré que $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ sont adjacentes et, d'après le théorème des suites adjacentes, elles convergent vers une même limite ℓ. De plus on a pour tout $n\in\mathbb{N}^*$:

$$u_n\leq\ell\leq v_n.$$

2. On cherche le premier $n\in\mathbb{N}$ tel que $v_n-u_n\leq\varepsilon$. On utilise donc une boucle `while` :

```
def approx(eps) :
    n = 1
    u = 1
    v = 2
    while v-u > eps :
        n = n+1
        u = u+1/(n**2)
        v = u+1/n
    return(u)
```

Exercice 8

1. Par exemple, `r = list(range(0,53,5))`.
2. L'opérateur `range` ne tolère que des pas entiers donc il n'est pas possible de l'utiliser ici.
   On peut utiliser une construction en compréhension en remarquant que $s=()_{i\in[\![0,10]\!]}$:

```
s = [i/10 for i in range(11)]
```

3. L'énumérateur commence à 5, donc $a=5$. Les termes décroissent de 1 en 1, donc le pas $c=-1$. Puisque l'énumérateur s'arrête avant b, on prend $b=-1$.
4. On remarque que $u=(2^i)_{i\in[\![0,5]\!]}$ et $v=\left(\frac{1}{i^2}\right)_{i\in[\![1,7]\!]}$ donc :

```
u = [2**i for i in range(6)]
v = [1/i**2 for i in range(1,7)]
```

5. (a) Pour x, on effectue une déclaration exhaustive, à la main élément par élément.
   (b) Pour y, on effectue une construction en compréhension :

```
y = [10 if x[i] == 0 else x[i] for i in range(len(x))]
```

Exercice 9

1. Voici le programme demandé :

```
def retirer(L,x) :
    M = []
    for k in L :
        if k != x :
            M.append(k)
    return M
```

2. Voici le programme demandé :

```
def dernierevaleurnonnulle(L) :
    k = len(L) - 1
    while k >= 0 :
        if L[k] != 0 :
            return L[k]
        else :
            k = k-1
    return 0
```

On part de la fin de la liste et on remonte jusqu'à temps d'obtenir une valeur non nulle. Si c'est le cas, le programme s'arrête en retournant cette valeur. Sinon, la boucle `while` s'arrête lorsque tous les éléments ont été parcourus et on retourne 0.

Exercice 10

1. Voici le programme demandé :

```
def chercher(L,x) :
    for k in L :
        if k == x :
            return True
    return False
```

2. Voici le programme demandé :

```
def chercherindice(L,x) :
    for k in range(len(L)) :
        if L[k] == x :
            return k
    return -1
```

Exercice 11

1. Voici le programme demandé :

```
def monmax(L) :
    m = L[0]
    for k in L :
        if k > m :
            m = k
    return m
```

2. (a) Voici la première version demandée :

```
def max2(L) :
    m = monmax(L)
    M = [k for k in L if k != m]
    return monmax(M)
```

(b) Voici la deuxième version (où on ne parcourt qu'une seule fois la liste) :

```
def max2(L) :
    m1 = L[0] #m1 représentera le premier max
    m2 = L[1] #m2 représentera le deuxième max
    if m1 < m2 :
        m1, m2 = m2, m1 #on ordonne m1 et m2 pour que m1 > m2
    for k in range(2,len(L)) :
         if L[k] > m1 :
             m1, m2 = L[k], m1
#L[k] est le nouveau max et m1 est le nouveau deuxième max
         elif L[k] > m2 :
             m2 = L[k]
#m1 est toujours le max et L[k] est le nouveau deuxième max
    return m2
```

Exercice 12

Voici le programme (on a utilisé la fonction `valabs` de l'exercice 5) :

```
def plusproches(L) :
    x = L[0]
    y = L[1]
    m = valabs(x-y)
    for i in range(len(L)) :
        for j in range(i+1,len(L)) :
            if valabs(L[i]-L[j]) > m :
                x = L[i]
                y = L[j]
                m = valabs(x-y)
    return x,y
```

Exercice 13

1. On rappelle que `int(x)` donne la partie entière de `x` s'il est positif. Donc :

```
def milieu(a,b) :
    return int((a+b)/2)
```

2. On suit le programme décrit dans l'énoncé en le traduisant en `Python` :

```
1  def recherchedicho(L,x) :
2      n = len(L)
3      deb = 0
4      fin = n-1
5      while n != 1 :
6          m = milieu(deb, fin)
7          if (-1)**n == 1 :
8              if (L[m] + L[m+1])/2 > x
9                  fin = m
10             else :
11                 deb = m+1
12         else :
13             if L[m] == x :
14                 deb = m
15                 fin = m
16             elif L[m] > x :
17                 fin = m-1
18             else :
19                 deb = m+1
20         n = fin - deb + 1
21     if L[deb] == x :
22         return True
23     else :
24         return False
```

3. Décrivons ligne par ligne ce programme :

L2 `n` contient la longueur de la portion de liste obtenue à chaque étape.

L3-4 `deb` et `fin` contiennent les indices de début et de fin de la portion de liste obtenue à chaque étape (la liste initiale est `L`).

L5 On s'arrête dès que la portion de liste obtenue est réduite à un élément.

L6 Si la liste n'est pas réduite à un élément, alors on commence par identifier l'indice qui permettra de partager la liste étudiée en deux.

L7 Si la liste contient un nombre pair d'éléments.

L8-9 Cas où la moitié de la liste contenant éventuellement x est la première.

L10-11 Cas où la moitié de la liste contenant éventuellement x est la deuxième.

L12 Si la liste contient un nombre impair d'éléments.

L13-15 Cas où l'élément central vaut x.

L16-17 Cas où la moitié de la liste contenant éventuellement x est la première.

L18-19 Cas où la moitié de la liste contenant éventuellement x est la deuxième.

L20 Dans tous les cas, la partie dans laquelle se trouve éventuellement x a pour indice de début `deb` et pour indice de fin `fin`. Sa longueur est donc `n = fin - deb + 1`.

L21-24 A la sortie de la boucle while, on a réduit l'étude à une liste contenant un seul élément qu'il faut tester.
