TD 2

Exercices d'application

Savoirs et compétences :

- *Alg – C15 : Récursivité : avantages et inconvénients.*

Exercice 1 – Fonction mystère

D'après ressources de C. Lambert. On donne la fonction suivante :

■ Python
```python
def mystere(L):
    """
    Ceci est la fonction mystère, saurez-vous trouver
    son but ?
    Entrée :
        * L(list) : liste de nombres entiers ou réels
    Sortie :
        * ???
    """
    n = len(L)
    if n==0 :
        return (None)
    elif n==1 :
        return (L[0])
    else :
        x = mystere(L[0:n-1])
        if x<=L[-1] :
            return (x)
        else :
            return (L[-1])
```

Question 1 *Sans coder la fonction, déterminer le résultat de l'instruction* ***print(mystere([14, 20, 3, 16]))*** *? Vous pourrez représenter de façon graphique l'empilement et le dépilement de la pile d'exécution.*

Question 2 *D'après vous quel est le but de cette fonction ?*

Question 3 *Programmer la fonction et tester l'instruction précédente. Sur plusieurs exemples, vérifiez la conjecture faite à la question précédente.*

Question 4 *Question subsidiaire. – Montrer que la propriété suivante est une propriété d'invariance :*
$\mathscr{P}$: l'algorithme retourne le plus petit élément de la liste de taille k, s'il existe.

R

- Il faudra montrer que l'algorithme se termine au moyen d'un variant de boucle.
- Il faudra montrer que $\mathscr{P}$ est une propriété d'invariance.

Correction n est un variant de boucle car :

- si $n=1$ ou $n=0$, l'algorithme se termine ;
- si $n>1$ chaque appel récursif est réalisé avec l'argument $n-1$. n décrit donc une suite strictement décroissante, jusqu'à ce que $n=1$ (terminaison de l'algorithme).

Soit la propriété suivante : soit une liste de taille k. L'appel à la fonction mystère retourne le plus petit élément à chaque itération.

Soit une liste de 2 éléments. x reçoit le premier élément de la liste. x est comparé au second élément. Si x est inférieur au second élément, c'est donc le plus petit élément. x est bien retourné. Sinon c'est bien le second élément qui est retourné.

Soit une liste de taille $k+1$. x reçoit le résultat de `mystere(L[0:k])`. D'après la propriété, x contient donc le plus petit élément de la liste `[0:k]`. x est comparé à l'élément $k+1$. Si x est inférieur à cet élément, c'est donc le plus petit élément. x est bien retourné. Sinon c'est que l'élément $k+1$ est le plus petit de la liste. C'est bien celui qui est retourné.

La propriété énoncée est donc un invariant de boucle.

Exercice 2 – Palindrome...

D'après ressources de C. Lambert. On souhaite réaliser une fonction *miroir* dont le but est de retourner le «miroir» d'une chaîne de caractères. Par exemple le résultat de *miroir("miroir")* serait *"riorim"*.

Question 1 *Programmer la fonction miroir_it permettant de répondre au problème de manière itérative.*

Question 2 *Programmer la fonction miroir_rec permettant de répondre au problème de manière récursive.*

Question 3 *Que renvoie la fonction si la chaîne de caractère est "Eh ! ça va la vache" ?*

Question 4 *Évaluer la complexité algorithmique de chacune des deux fonctions.*

Exercice 3 – Calcul de $n!$

On rappelle la définition de $n!$:

$$\forall n \in \mathbb{N} : \begin{cases} n! = \prod_{k=1}^{n} k & \text{si } n \geq 1 \\ n! = 1 & \text{si } n = 0 \end{cases}$$

R Pour vérifier vos résultats, vous pouvez utiliser la fonction disponible dans la bibliothèque math :

```
>>> from math import factorial
>>> print(factorial(4))
    24
```

Question 1 *Définir la fonction fact_it permettant de calculer $n!$ de façon itérative.*

Correction

```python
def fact_it(n):
    """
    Calcul de n! de manière itérative
    Entrée :
     * n(int) : nombre entier naturel
    Sortie
     * res (int): résultat, nombre entier naturel
    """
    # Vérifie que n est bien positif ou nul
    # On pourrait aussi vérifier que n est bien un
    # integer
    assert (n >= 0),"Nombre négatif"
    if n==0:
        return 1
    else :
        res = 1
        k=n
```

```
    while k>0:
        res=res * k
        k=k-1
    return res
```

Question 2 *Donner alors la complexité algorithmique de votre algorithme.*

Correction Il y a n opérations dans le pire des cas. En conséquence, $C(n) = \mathcal{O}(n)$.

Question 3 *Définir la fonction fact_rec permettant de calculer $n!$ de façon récursive.*

Correction
```
def fact_rec(n):
    """
    Calcul de n! de manière récursive
    Entrée :
     * n(int) : nombre entier naturel
    Sortie
     * (int): résultat, nombre entier naturel
    """
    if n<2:
        return 1
    else:
        return n*fact_rec(n-1)
```

Question 4 *Évaluer* 1010! *dans les trois cas (itératif, récursif et fonction native de Python). Expliquer ce qu'il se passe ?*

Correction Dans le cas de l'appel récursif, on réalise que Python limite à 1000 le nombre d'appels.

Exercice 4 – Suite de Fibonacci

D'après ressources de C. Lambert.

On définit la suite de Fibonacci de la façon suivante :

$$\forall n \in \mathbb{N}, \begin{cases} u_0 = 0, u_1 = 1 \\ u_{n+2} = u_n + u_{n+1} \end{cases}$$

Question 1 *Définir la fonction fibonacci_it permettant de calculer u_n par une méthode itérative. Évaluer la complexité algorithmique de l'algorithme.*

Question 2 *Définir la fonction fibonacci_rec permettant de calculer u_n par une méthode récursive « intuitive». Évaluer la complexité algorithmique de l'algorithme.*

Question 3 *Observer comment passer du couple (u_n, u_{n+1}) au couple (u_{n+1}, u_{n+2}). En déduire une autre méthode récursive pour calculer le n^e terme de la suite de Fibonacci. Évaluer la complexité algorithmique de l'algorithme.*

Exercice 5 – Calcul de déterminant

D'après ressources de David Prévost – UPSTI

On souhaite calculer le déterminant d'une matrice (carrée) A par la formule de développement sur la première ligne (ou sur la première colonne) mais pas une autre (par exemple si on le faisait sur la troisième, la fonction ne conviendrait pas pour un déterminant 2x2).

Question 1 *Écrire une fonction extraire(A,lig,col) qui permet de fabriquer la matrice extraite de A en rayant la ligne lig et la colonne col (dans l'optique d'obtenir son déterminant).*

Correction

```
def zeros(n):
    """
    Crée une matrice de zéros de taille nxn
    Entrée :
```

```
     * n(int) : nombre entier positif
    Sortie :
     * m(list) : liste de taille nxn de zéros
    """
    # En utilisant une liste de compréhension,
    # on pourrait procéder ainsi :
    # return [[0 for j in range(n)] for i in range(n)]
    m = []
    for i in range(n):
        mm = []
        for k in range(n):
            mm.append(0)
        m.append(mm)
    return m
```

Correction

```
def extraire(M,l,c):
    """
    Supprime la ligne l et la colonne c de
    la matrice M
    Entrées :
     * M(list(list)) : matrice de taille nxn
     * l(int) : numéro de ligne àsupprimer
     * c(int) : numéro de colonne àsupprimer
    Sortie :
     * MM(list(list)) : matrice de taille n-1 x n-1
     en ayant supprimer la ligne l et la colonne c
    """
    n = len(M)
    MM = zeros(n-1)
    p=-1
    for i in range(n):
        if i!= l:
            p=p+1
            q=-1
            for j in range(n):
                if j!= c:
                    q=q+1
                    MM[p][q]=M[i][j]

    return MM
```

Question 2 *Écrire une fonction récursive determinant(A) retournant le déterminant de A.*

Correction

```
def determinant(M):
    """
    Calcule le déterminant de la matrice M.
    Entrée :
     * M(list) : matrice de taille nxn
    Sortie :
     * d(flt) : déterminant de la matrice
    """
    n = len(M)
    if len(M)==1:
        return M[0][0]
    else :
        d=0
        for i in range(n):
            d = d+(-1)**i * M[i][0]*determinant(extraire(M,i,0))
    return d
```

Question 3 *Tester l'algorithme sur une matrice (de taille au moins 3,3) et comparer le résultat à ce que retourne la fonction det du module linalg de la bibliothèque numpy.*

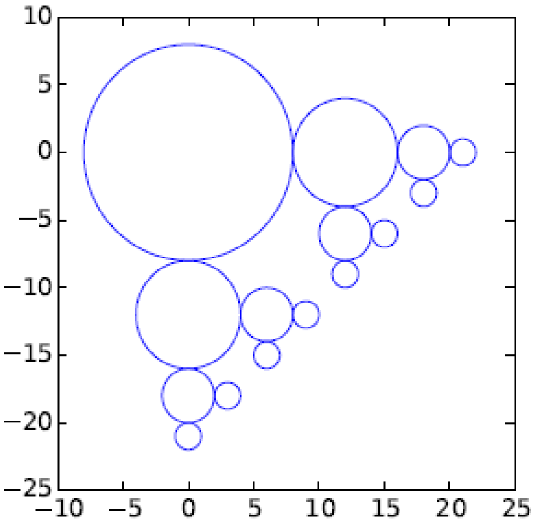
Correction

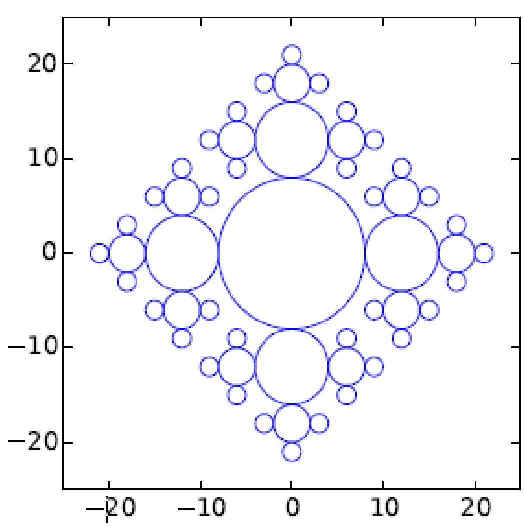
```
from numpy import linalg
print(linalg.det(MM))
```

Exercice 6 – Bubble bobble

D'après ressources de Jean-Pierre Becirspahic `http://info-llg.fr/`.
On suppose disposer d'une fonction *cercle(x, y, r)* qui trace à l'écran un cercle de centre $(x; y)$ de rayon r.

Question *Définir deux fonctions récursives permettant de tracer les dessins présentés figure suivante (chaque cercle est de rayon moitié moindre qu'à la génération précédente).*





Correction

```
import matplotlib.pyplot as plt
import numpy as np

def cercle(coord, r):
    x,y=[],[]
    for t in range (101):
        x.append(coord[0]+r*np.cos(t*np.pi/50))
        y.append(coord[1]+r*np.sin(t*np.pi/50))
    plt.plot(x,y)


def bulle1(n,x=0,y=0,r=8):
    cercle([x,y],r)
    if n>1 :
        bulle1(n-1,x+3*r/2,y,r/2)
        bulle1(n-1,x,y-3*r/2,r/2)

bulle1(6)
plt.axis('equal')
plt.show()
```