Récursivité (Correction)

1. Exercices divers

1.1. Algorithme d'Euclide

Q.1. Ecrire une fonction récursive `pgcd(a,b)` qui calcule le plus grand commun diviseur de deux entiers en utilisant l'algorithme d'Euclide.

```
def pgcd(a,b):
    if b==0:
        return a
    return pgcd(b,a%b)
```

Q.2. En déduire une fonction `bezout` qui étant donnés deux entiers a et b calcule le triplet (d,u,v) comme expliqué ci-dessus.

```
def bezout(a,b):
    if b==0:
        return a,1,0
    (d,u,v)=bezout(b,a%b)
    u,v=v,u-(a//b)*v
    return d,u,v
```

1.2. PGCD Rapide

Q.1. Ecrire la fonction qui calcule le pgcd rapide.

```
def PGCDrapide(a,b):
    if a==0:
        return b
    if b==0:
        return a
    else:
        if (a%2,b%2)==(0,0):
            return 2*PGCDrapide(a//2,b//2)
        if (a%2,b%2)==(1,0):
            return PGCDrapide(a,b//2)
        if (a%2,b%2)==(0,1):
            return PGCDrapide(a//2,b)
        else:
            m=min(a,b)
            M=max(a,b)
            return PGCDrapide(M-m,m)
```

1.3. Déterminant

Q.2. Ecrire 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).

```
def zeros(n):
    return [[0 for j in range(n)] for i in range(n)]

def extraire(A,lig,col):
    n=len(A)
    M=zeros(n-1)
    p=-1
    for i in range (n):
        if i!=lig:
            p+=1
            q=-1
            for j in range (n):
                if j!=col:
                    q+=1
                    M[p][q]=A[i][j]

    return M
```

Q.3. Ecrire une fonction récursive `determinant(A)` retournant le déterminant de *A*.

```
def determinant(A):
    n=len(A)
    if n==1:
        return A[0][0]
    else:
        C=0 #On initialise C
        for i in range(n):
            C+=((-1)**i) * A[i][0] * determinant(extraire(A,i,0))
        return C
```

Q.4. 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*.

```
#Confrontation du résultat avec la bibliothèque de numpy.linalg
from numpy import linalg as npl
A=[[1, 2, 2, -1], [4, 8, 2, -4], [1, -2, 2, -1], [4, -4, 2, 1]]
print("\n")
print("Soit la matrice : {}".format(A))
print("\n")
print("Resultat avec notre programme recursif : ")
print("det = {}".format(determinant(A)))
print("\n")
print("Resultat avec la bibliotheque linalg : ")
print("det = {}".format(npl.det(A)))
```

Ce qui donne :

```
Soit la matrice : [[1, 2, 2, -1], [4, 8, 2, -4], [1, -2, 2, -1], [4, -4, 2, 1]]

Resultat avec notre programme recursif :
det = -120

Resultat avec la bibliotheque linalg :
det = -119.99999999999997
```

1.4. Récursivité mutuelle

Q.5. Implémenter les suites définies par : $u_{n+1} = u_n/v_n$; et : $v_{n+1} = v_n + u_n$, avec $u_0 = 1$ et $v_0 = 1$.

```
def u(n):
    if n==0:
        return 1
    return u(n-1)/v(n-1)

def v(n):
    if n==0:
        return 1
    return v(n-1)+u(n-1)
```

2. Tours de Hanoï

Q.6. En partant d'un raisonnement sur deux disques, trouver l'algorithme de résolution des tours de Hanoï sur un nombre *N* disques.

- Il faut penser récursivement. Pour résoudre le problème des tours de Hanoï avec deux disques, on a déplacé de A vers B, puis de A vers C pour finalement déplacer de B vers C (une seule et unique opération est à chaque fois possible). Donc (2-1) disque de A vers B, puis 1 disques (le plus grand) de A vers C et à la fin (2-1) disque de B vers C. La logique est donc la suivante : "isoler le plus grand disque de A puis le placer à la base de la tour de destination C puis appliquer le même algorithme de B vers C". En résumé, l'algorithme de résolution des tours de Hanoï sur un nombre n de disques est le suivant :

- Déplacer n disques de A vers C en passant par B :
  - Déplacer (n-1) disques de A vers B en passant par C;
  - Déplacer 1 disque de A vers C ;
  - Déplacer (n-1) disques de B vers C en passant par A.

Q.7. Ecrire sous Python une telle fonction **`hanoi`** et la tester.

```
# Nombre de palets
N = ["_","__","___"]

# On veut déplacer N palets de A vers C en passant par B
def hanoi(A, B, C, N):
    if len(N)>0:
        # On déplace N-1 palets de A vers B en passant par C
        hanoi(A, C, B, N[:-1])
        # On déplace le dernier palet de A vers C
        print("{} de {} vers {}".format(N[-1],A, C))
        # On déplace n-1 palets de B vers C en passant par A
        hanoi(B, A, C, N[:-1])

# Les tours sont représentées par les chaînes "A", "B", "C"
hanoi("A", "B", "C", N)
```

Ce qui retourne :

```
_ de A vers C
__ de A vers B
_ de C vers B
___ de A vers C
_ de B vers A
__ de B vers C
_ de A vers C
```