DS 4 - Informatique

Exercice 1. 1. Ecrire une fonction `f` Python qui prend en argument un flottant x et retourne la valeur de la fonction f définie par

$$f(x) = \sqrt{x+1}$$

La fonction Python `f` retournera une erreur si x n'est pas dans l'ensemble de définition de f.

2. Ecrire une fonction `suite` Python qui prend en argument un entier n et une valeur de u_0 et retourne la valeur de u_n où la suite $(u_n)_{n\in\mathbb{N}}$ est définie par récurrence par

$$u_{n+1} = \sqrt{u_n + 1}$$

Correction 1.

1.
```python
def f(x):
    if x<-1:
        return 'Apprends tes ensembles de définition'
    else:
        return (x+1)**(0.5)
```

2.
```python
def suite_u(n,u_0):
    u=u_0
    for i in range(n):
        u=f(u)
    return u
```

Exercice 2 (Coefficients binomiaux)**.**

Soit $(k,n) \in \mathbb{N}^2$. L'objectif de l'exercice est de calculer de deux manières le coefficient binomial $\binom{n}{k}$. On rappelle que par convention $0! = 1$ et

$$\binom{n}{k} = \begin{cases} \dfrac{n!}{k!\,(n-k)!} & \text{si } k \leqslant n, \\ 0 & \text{sinon.} \end{cases}$$

On prendra bien garde à distinguer le cas $k \leqslant n$ et $k > n$

1. Une première méthode utilisant la formule usuelle.
   (a) Écrire une fonction `factoriel(n)` qui prend un entier `n` et qui renvoie la valeur de $n!$.
   (b) Utiliser la fonction précédente pour écrire une fonction `binome1(n,k)` permettant de calculer $\binom{n}{k}$, en prenant (k,n) en argument.
2. Une deuxième méthode utilisant la formule du triangle de Pascal.
   On rappelle que l'on a la formule suivante, dite de Pascal, $\forall (k,n) \in (\mathbb{N}^*)^2, k \leqslant n$,

$$\binom{n}{k} + \binom{n}{k-1} = \binom{n+1}{k}$$

   (a) Compléter (sur votre copie) la fonction `Pascal(L)` qui prend en argument une liste d'entiers `L` correspondant à une ligne du triangle de Pascal et qui renvoie la liste correspondant à la ligne suivante du triangle de Pascal, obtenue après utilisation de la formule de Pascal.
   Par exemple, `Pascal([1,4,6,4,1])` doit renvoyer `[1,5,10,10,5,1]`.

```python
def Pascal(L):
    n=len(L)
    L2=[1 for i in range(....)] #initialisation de la liste renvoyée
    for k in .......:
        L2[k] = L[k] +L[k-1] #utilisation de la formule de Pascal
    return(L2)
```

(b) Utiliser la fonction précédente pour écrire une fonction `binome2(n,k)` renvoyant la valeur de $\binom{n}{k}$.

Correction 2.

1. (a)
```
def factoriel(n):
    P=1
    for i in range(1,n+1):
        P=P*i
    return P
```
(b)
```
def binom1(n,k):
    if k>n:
        return 0
    else:
        return factoriel(n)/(factoriel(k) * factoriel(n-k))
```
2. (a)
```
def Pascal(L):
    n=len(L)
    L2=[1 for i in range(n+1)] #initialisation de la liste renvoyée
    for k in range(1,n+1):
        L2[k] = L[k] +L[k-1] #utilisation de la formule de Pascal
    return(L2)
```
(b)
```
def binom2(n,k):
    if k>n:
        return 0
    else:
        L=[1]
        for i in range(n):
            L=Pascal(L)
        return L[k]
```

Exercice 3. Soit $p \in \mathbb{N}$. Dans cet exercice, on représente des équations linéaires à p inconnues par des listes de taille $p+1$ contenant les coefficients de l'équation dans l'ordre, en finissant par le coefficient du second membre. Ainsi :

$$[a_1, a_2, \ldots, a_p, b] \text{ représente l'équation } a_1x_1 + a_2x_2 + \cdots + a_px_p = b.$$

Par exemple, l'équation $3x - 2y - z = 5$ est représentée par la liste $[3, -2, -1, 5]$.
De même, une solution éventuelle $(x_1, \ldots, x_p) \in \mathbb{R}^p$ est représentée par une liste de longueur p.

1. Écrire une fonction Python `est_solution(E, X)`, qui prend en paramètre une liste E représentant une équation et une liste X représentant une solution éventuelle, et qui renvoie :
   — `True` si X est bien solution de l'équation E,
   — `False` sinon.

   Par exemple : `est_solution([3, -2, -1, 5], [1, 0, -2])` devra renvoyer `True` puisque $3 \times 1 - 2 \times 0 - 1 \times (-2) = 5$.
2. Écrire une fonction Python `compte_zéros(E)` qui prend en argument une liste E, et qui compte le nombre de zéros consécutifs au début de la liste. Par exemple :
   — `compte_zéros([0, 0, 1, 0, 3, 0])` devra renvoyer 2
   — `compte_zéros([1, 0, 0, -2])` devra renvoyer 0
3. On représente un système linéaire de n équations à p inconnues par une liste S dont les éléments sont les listes représentant les équations du système. Il s'agit donc d'une liste de listes. L'instruction `len(S)` renvoie alors le nombre d'équations du système.

   On supposera dans la suite que tous les systèmes représentés en Python ont bien un sens mathématique et ont au moins une équation.

(a) Donner la liste (c'est une liste dont les éléments sont des listes correspondant aux équations) Python qui représente le système suivant :

$$\begin{cases} x + 2y = 0 \\ 2x - y = 3 \end{cases}$$

(b) Écrire une fonction `nb_inconnues(S)` qui prend en argument un système S et qui renvoie le nombre d'inconnues du système.

(c) Écrire une fonction `est_solution_système(S, X)` qui prend en argument un système S représenté de cette manière, et une liste X correspondant à une solution éventuelle, et qui renvoie un booléen indiquant si X est une solution du système S.

(d) Compléter (sur votre copie) la fonction suivante afin qu'elle permette de déterminer si un système S est échelonné ou non.

```
def est_echelonne(S):
    n = len(S)
    p = nb_inconnues(S)
    for i in range(n-1):
        if compte_zeros(S[i]) < p:  # si la ligne i n'est pas  triviale
            if compte_zeros(S[i+1]) <= compte_zeros(S[i]):
                #si la ligne suivante compte moins de zero
                return ........
        else:
            if compte_zeros(S[i+1]) <p :
                #si la ligne suivante n'est pas triviale
                return ......
    return ......
```

4. Écrire une fonction `rang(S)`, qui prend en argument un système S supposé échelonné et qui renvoie son rang. En déduire une fonction `nombre_solutions(S)` qui prend en argument un système S échelonné, et qui renvoie son nombre de solutions.

Correction 3.

1.
```
def est_solution{E,X}:
    val=0
    for i in range(len(X)):
        val+=E[i]*X[i]
    if val==E[-1]:
        return True
    else:
        return False
```

2.
```
def compte_zeros(E):
    c=0
    for e in E:
        if e==0:
            c+=1
        else:
            return c
    return c
```

autre solution avec boucle while :

```
def compte_zeros(E):
    c=0
    while E[c]==0 and c<len(E):
```

```
            c+=1
        return c
```

3. (a)

```
[[1,2,0],[2,-1,3]]
```

(b)

```
def nb_inconnues(S):
    return( len(S[0])-1)
```

(c)

```
def est_solution_systeme(S,X):
    for E in S:
        if est_solution(E,X) == False:
            return False
    return True
```

(d)

```
def est_echelonne(S):
    n = len(S)
    p = nb_inconnues(S)
    for i in range(n-1):
        if compte_zeros(S[i]) < p:  # si la ligne i n'est pas  triviale
            if compte_zeros(S[i+1]) <= compte_zeros(S[i]):
                #si la ligne suivante compte moins de zero
                return False
        else:
            if compte_zeros(S[i+1]) <p :
                #si la ligne suivante n'est pas triviale
                return False
    return True
```

(e)

```
def rang(S):
    p=nb_inconnues(S)

    r=0
    for E in S:
        if compte_zeros(E)<p:
            r+=1
    return(r)
```

```
def nb_solutions(S):
    p=nb_inconnues(S)
    for E in S:
        if compte_zeros(E)>=p:
            if E[-1] ! = 0:
                return 'pas de solution'

    r=rang(S)
    if r==p:
        return '1 solution'
    else:
        return 'Infinité de solutions'
```