IA.3 – Les LISTES… et les tests IF

Soit (I_n) la suite définie par : $\begin{cases} I_0 = 0{,}6 \\ I_{n+1} = (n+1)I_n - 0{,}4 \end{cases}$

Que renvoie mystere(100) ?

```
def mystere(n) :
    I=0.6
    L=[I]
    for i in range(n) :
        I=(1+i)*I-0.4
        L.append(I)
    return L
```

0.6 → I ⇒ L=[0.6]

0 → i : (1+0)*0,6-0.4=0.2 → I ⇒ L=[0.6 , 0.2]

1 → i : (1+1)*0,2-0.4=0 → I ⇒ L=[0.6 , 0.2 , 0]

Donc les termes successifs de la suite sont rangés dans la liste.
Pour n=100, on a rentré I_0, la boucle calcule et range les 100 termes suivants : de u_1 jusqu'à u_{100}
Donc l'algorithme renvoie la liste des 101 premiers termes de la suite, de I_0 à I_{100}

Soit (u_n) la suite définie par : $\begin{cases} u_0 = 5 \\ u_{n+1} = 2 + \ln(u_n^2 - 3) \end{cases}$

```
def suite(k) :
    L=[]
    u=5
    for i in range(k+1) :
        L.append(u)
        u=2+log(u**2-3)
    return(L)
```

1. Compléter l'algorithme ci-contre pour que suite(k) qui prend en paramètre un entier k, renvoie la liste des k premières valeurs de la suite (u_n).

Remarque : On précise que, pour tout réel strictement positif a, log(a) renvoie la valeur du logarithme népérien de a.

On commence par rentrer u_0 pour le 1er tour de la boucle
Donc pour rentrer les k premiers termes, il faut avoir le terme u_{k-1} : mais attention, au k-ième tour de boucle, u_{k-1} est calculée mais n'est pas mise dans la liste…
C'est pourquoi on doit aller jusqu'à k+1

2. On a exécuté suite(9) ci-dessous. Emettre deux conjectures : l'une sur le sens de variation de la suite (u_n) et l'autre sur son éventuelle convergence.

```
>>> suite(9)
[ 5, 5.091042453358316, 5.131953749864703,
5.150037910978289, 5.157974010229213, 5.1614456706362954,
5.162962248594583, 5.163624356938671, 5.163913344065642]
```

Semble être **croissante** et peut-être converger vers une valeur entre 5, 16 et 5,17… mais pas sûr

3. On a créé à la suite la fonction mystere(n) ci-contre et exécuté mystere(1000), ce qui a renvoyé 1.

```
>>> mystere(10000)
1
```

Cet affichage contredit-il la conjecture émise sur le sens de variation de la suite (u_n) ? Justifier.

```
def mystere(n) :
    L=suite(n)
    c=1
    for i in range(n-1) :
        if L[i]>L[i+1] :
            c=0
    return c
```

Dans L on met donc les n premiers termes de u_n, de u_0 à u_{n-1}
On met un compteur/témoin à 1
La boucle permet de tester successivement chacune des n valeurs de la liste : Si pour une valeur, elle se retrouve plus petite que la suivante, le compteur/témoins passe à 0.
Comme la valeur de c renvoyée est toujours 1, ça veut dire que les 1000 premiers termes de la suite vérifient la croissance. Donc non, ça ne contredit pas

Soit (p_n) la suite définie par : $\begin{cases} p_1 = 1 \\ p_{n+1} = 0{,}2p_n + 0{,}7 \end{cases}$

On souhaite disposer de la liste des premiers termes de la suite (p_n) pour $n \geq 1$.
Pour cela, on utilise une fonction appelée repas programmée en langage Python dont on propose trois versions, indiquées ci-dessous.

Programme 1

```
1  def repas(n):
2    p=1
3    L=[p]
4    for k in range(1,n):
5      p = 0.2*p+0.7
6      L. append(p)
7     return(L)
```

Programme 2

```
1  def repas(n):
2    p=1
3    L=[p]
4    for k in range(1,n+1):
5      p = 0.2*p+0.7
6      L. append(p)
7    return(L)
```

Programme 3

```
def repas(n):
  p=1
  L=[p]
  for k in range(1,n):
    p = 0.2*p+0.7
    L.append(p+1)
  return(L)
```

1. Lequel de ces programmes permet d'afficher les n premiers termes de la suite (p_n) ?
Les 3 programme commencent par rentrer p_1 dans la liste. Pour avoir les n premiers termes, il faut aller jusqu'au terme p_n : c'est-à-dire les n-1 termes suivants.
Il faut donc que la boucle tourne **n-1 fois** : ça élimine l'algo 2 (tourne n fois)
Dans l'algo 3, la boucle fait rentrer les termes $p_i + 1$ allez savoir pourquoi.
Donc c'est l'algo 1

2. Avec ce programme, donner le résultat renvoyé pour $n = 5$.
A la calculatrice, on calcule les 5 premiers termes de p_n
$\Rightarrow$ $[\mathbf{1}, \mathbf{0.9}, \mathbf{0.88}, \mathbf{0.876}, \mathbf{0{,}8752}]$

Une donnée binaire est une donnée qui ne peut prendre que deux valeurs : 0 ou 1.
Une donnée de ce type est transmise successivement d'une machine à une autre.
Chaque machine transmet la donnée reçue soit de manière fidèle, c'est-à-dire en transmettant l'information telle qu'elle l'a reçue (1 devient 1 et 0 devient 0), soit de façon contraire (1 devient 0 et 0 et devient 1).
La transmission est fidèle dans 90% des cas, et donc contraire dans 10% des cas. La première machine reçoit toujours la valeur 1.

```
1  def simulation(n) :
2          donnee=1
3          liste=[donnee]
4          for k in range(n) :
5                  if random()<0.1 :
6                          donnee=1-donnee
7                  liste.append(donnee)
8          return liste
```

Cette transmission est modélisée par la fonction simulation qui prend en paramètre le nombre n de transmissions réalisées d'une machine à une autre, et qui renvoie la liste des valeurs successives de la donnée binaire.

On rappelle que l'instruction random() renvoie un nombre aléatoire de l'intervalle $[0; 1[$.
Par exemple, simulation(3) peut renvoyer [1, 0, 0, 1].

En gros l'algorithme simule la transmission des données dans n machines : à chaque machine, soit la donnée est transmise à l'identique, soit elle est changée en son contraire. Le changement est aléatoire, paramétré par la fonction random() : 1 chance sur 10 de changer

Déterminer le rôle des instructions des 5e et 6e lignes.

5e ligne : if random()<0.1 : *si le nombre aléatoire est dans l'intervalle $[0; 0{,}1[$. Ce qui correspond à 1 chance sur 10, les 10% de transmission contraire*

6e ligne : donnee=1-donnee : *dans le cas où on a une transmission contraire, le 1 devient 0 (or 1-1=0) et le zéro devient 1 (or 1-0=1), donc ça change la donnée binaire en son contraire.*