TP1 : Calcul du $m^{\text{ème}}$ élément, des m premiers éléments d'une suite

(Révisions sur la structure itérative `for`*)*

- Dans le dossier `Info_2a` (resp. `Info_3a`), créer le dossier `TP_1`.

I. Calcul du $10^{\text{ème}}$ élément d'une suite

On considère la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par : $\begin{cases} \forall n \in \mathbb{N},\ u_{n+1} = 2\,u_n + n + 1 \\ u_0 = 1 \end{cases}$

D'aute part, on considère le programme **Python** suivant :

```
1  u = 1
2  u = 2 * u + 0 + 1
3  u = 2 * u + 1 + 1
4  u = 2 * u + 2 + 1
5  u = 2 * u + 3 + 1
6  u = 2 * u + 4 + 1
```

- Que réalise ce programme ?

  - À la fin de l'exécution de la première ligne, la variable `u` contient la valeur de u_1.
  - La valeur de u_1 est donnée par la formule : $u_1 = 2\,u_0 + 0 + 1$.
    La deuxième ligne consiste à mettre à jour la variable `u` en lui assignant la valeur calculée par `2 * u + 0 + 1`. Or en début de ligne 1, la variable `u` contient u_0.
    Ainsi, en fin de ligne, une fois l'affectation exécutée, la variable `u` contient la valeur de u_1.
  - Ainsi, `u` contient les valeurs successives de la suite (u_n).

- Comment obtenir la valeur de u_{10} ? De u_{20} ? De u_{250} ?

  - Pour calculer u_{10} et u_{20}, on peut envisager d'ajouter des lignes au programme précédent.
  - Cette méthode ne convient évidemment pas pour le calcul de u_{250}. On fait alors appel à une structure itérative. Plus précisément, étant donné que l'on connaît le nombre d'itérations, on utilise une boucle `for`.

```
1  u = 1
2  for i in range(250) :
3    u = 2 * u + i + 1
```

II. Calcul du $m^{\text{ème}}$ élément d'une suite

- Dans l'éditeur de texte, écrire un programme qui :
  - × demande initialement à l'utilisateur d'entrer au clavier la valeur d'un entier `m`,
  - × initialise une variable `u` à la valeur 1,
  - × met à jour `u` dans une structure itérative de sorte à ce que `u` contienne la valeur du $m^{\text{ème}}$ élément de la suite en fin de boucle.
  - × affiche la valeur de `u`.

  Sauvegarder ce programme sous le nom `emeSuiteU.py`.

```
m = int(input("Prière d''entrer un entier m : "))
u = 1
for i in range(m) :
    u = 2 * u + i + 1
print(u)
```

- Calculer u_{12} et u_{20} à l'aide du programme précédent.

On obtient $u_{12} = 12274$ et $u_{20} = 3145706$.

- Dans un nouvel onglet de l'éditeur de texte, copier-coller le programme précédent.
  Modifier ce programme afin d'obtenir une fonction `emeSuiteU` qui :
  - × prend en paramètre une variable `m`,
  - × calcule en sortie une variable `u` contenant le $\texttt{m}^{\text{ème}}$ élément de la suite (u_n).

```
def emeSuiteU(m) :
    u = 1
    for i in range(m) :
        u = 2 * u + i + 1
    return u
```

- Calculer u_7 et u_{15} à l'aide de la fonction précédente.

On obtient $u_7 = 375$ et $u_{15} = 98287$.

- Selon vous, quels sont les avantages de la représentation sous forme de programme avec dialogue utilisateur ? Sous forme de fonction ?

- D'un point de vue utilisateur, la version avec dialogue utilisateur, plus ludique, peut être plus appréciée.
- D'un point de vue algorithmique, il faut privilégier la version sous forme de fonction. L'avantage est que le calcul réalisé par `emeSuiteU` peut facilement être utilisé ailleurs (notamment dans une autre fonction) : il suffit pour ce faire d'écrire l'appel `emeSuiteU(m)` (avec `m` choisi correctement).

III. Calcul des m premiers éléments d'une suite

- Dans l'éditeur de texte, écrire un programme qui :
  - × demande initialement à l'utilisateur d'entrer au clavier la valeur d'un entier `m`,
  - × crée une liste `U` contenant initialement l'unique élément u_0,
  - × met à jour la liste `U` dans une structure itérative de sorte à ce que `U` contienne les valeurs des m premiers éléments de la suite (u_n) en fin de boucle.

  Sauvegarder ce programme sous le nom `premSuiteU.sce`.

```
m = int(input("Prière d''entrer un entier m : "))
U = [1]
for i in range(m) :
    U.append(2 * U[i] + i + 1)
print(U)
```

- Calculer les 5 premiers éléments de la suite à l'aide du programme précédent.

On obtient : $[1, 3, 8, 19, 42]$.

- Dans un nouvel onglet de l'éditeur de texte, copier-coller le programme précédent.
  Modifier ce programme afin d'obtenir une fonction `premSuiteU` qui :
  - × prend en paramètre une variable `m`,
  - × calcule en sortie une variable `U` contenant les `m` premiers éléments de la suite (u_n).

```
def premSuiteU(m) :
    U = [1]
    U(1) = 1
    for i in range(m) :
        U.append(2 * U[i] + i + 1)
    return U
```

- Effectuer le tracé des 30 premières valeurs de la suite (u_n). *(on pourra importer les librairies* `matplotlib.pyplot` *et* `numpy`*)*
  Quelle conjecture peut-on émettre sur la limite de la suite (u_n) ?

```
m = 30
absc = np.linspace(0, m, m+1)
U = premSuiteU(m)
plt.plot(absc, U)
```

Au vu du graphique obtenu, on peut conjecturer que la suite (u_n) tend vers $+\infty$.
(il est par exemple simple de démontrer que, pour tout $n \in \mathbb{N}$, $u_n \geqslant 2^{n-1}$)

IV. Suites $u_{n+1} = f(u_n)$ aux concours

IV.1. EML - 2018

On pose : $u_0 = 4$ et $\forall n \in \mathbb{N},\ u_{n+1} = \ln(u_n) + 2$.

▸ Écrire une fonction **Python** d'en-tête `def suite(n) :` qui, prenant en argument un entier n de $\mathbb{N}$, renvoie la valeur de u_n.

```
1  import numpy as np
2  def suite(n) :
3      u = 4
4      for k in range(n) :
5          u = np.log(u) + 2
6      return u
```

La variable `u` est créée pour contenir successivement les valeurs u_0, u_1, …, u_n.

- On initialise donc cette variable à $u_0 = 4$ avec la ligne 2 :

```
3      u = 4
```

- On met ensuite à jour `u` de manière itérative avec la ligne 4 :

```
5          u = np.log(u) + 2
```

Remarque
Si on avait souhaité afficher tous les n premiers termes de la suite (u_n), on aurait modifié le script précédent de la façon suivante :

```
1  import numpy as np
2  def suite(n) :
3      u = [4]
4      for k in range(1,n) :
5          u.append(np.log(u[k-1]) + 2)
6      return u
```

IV.2. EML - 2019

On introduit la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par :

$$u_1 = 1 \quad \text{et} \quad \forall n \in \mathbb{N}^*,\ u_{n+1} = u_n + \frac{1}{n^2\, u_n} = \frac{1}{n}\, f(n\, u_n) \qquad \text{où } f : t \mapsto t + \frac{1}{t}$$

▸ Recopier et compléter les lignes 3 et 4 de la fonction **Python** suivante afin que, prenant en argument un entier n de $\mathbb{N}^*$, elle renvoie la valeur de u_n.

```
1  def suite(n) :
2      u = 1
3      for k in ..................
4          u = ..................
5      return u
```

```
1  def suite(n) :
2      u = 1
3      for k in range(1,n) :
4          u = (1/n) * (n*u + 1/(n*u))
5      return u
```

Détaillons l'obtention de ce programme.
La variable `u` est crée pour contenir successivement les valeurs u_1, …, u_n.

- On initialise donc cette variable à $u_1 = 1$ avec la ligne 2.

```
2      u = 1
```

- On met ensuite à jour `u` à l'aide d'une structure itérative (boucle for) avec les lignes 3 à 4.

```
3      for k in range(1,n) :
4          u = (1/n) * (n*u + 1/(n*u))
```

Remarque
On pouvait également coder la fonction f dans un script à part. On aurait alors obtenu les deux programmes suivants :

```
1  def f(t) :
2      y = t + 1/t
3      return y
```

```
1  def suite(n) :
2      u = 1
3      for k in range(1,n) :
4          u = (1/n) * f(n*u)
5      return u
```

IV.3. EDHEC - 2019

On considère la suite (u_n) définie par : $\forall n \in \mathbb{N}$, $u_n = \dfrac{4^n\,(n!)^2}{(2n+1)!}$.

On admet que, si `L` est une liste de flottants, la commande `np.prod(L)` renvoie le produit des éléments de `L`.

Compléter le script **Python** suivant afin qu'il permette de calculer et d'afficher la valeur de u_n pour une valeur de n entrée par l'utilisateur.

```
1  n = int(input(‘‘entrez une valeur pour n : ‘‘)
2  x = [k for k in range(1,n+1)]
3  m = 2 * n + 1
4  y = [k for k in range(1,m+1)]
5  v = ..........
6  w = ..........
7  u = .......... * (v**2) / w
8  print(u)
```

- Commentons tout d'abord le début du programme proposé.
  - × On commence par demander à l'utilisateur d'entrer une valeur pour l'entier `n`.

```
1  n = int(input(‘‘entrez une valeur pour n : ‘‘)
```

  - × On stocke ensuite dans la variable `x` la liste contenant les entiers de 1 à `n`.

```
2  x = [k for k in range(1,n+1)]
```

  - × On stocke de plus dans la variable `m` la quantité $2\mathtt{n}+1$.

```
3  m = 2 * n + 1
```

  - × On stocke enfin dans la variable `y` la liste contenant les entiers de 1 à `m`, c'est-à-dire les entiers de 1 à $2\mathtt{n}+1$.

```
4  y = [k for k in range(1,m+1)]
```

- On cherche maintenant à stocker dans la variable `u` la valeur de $u_{\texttt{n}}$.
  D'après la question ***5.b)***, pour tout $n \in \mathbb{N}$:

$$u_n = \frac{4^n\,(n!)^2}{(2n+1)!}$$

  × L'énoncé propose de compléter la commande suivante :

```
7  u = .......... * (v**2) / w
```

  Par analogie avec la formule de la question ***5.b)***, on souhaite donc stocker dans la variable `v` la valeur `n!` et dans la variable `w` la valeur `(2n+1)!`.

  × On cherche alors une commande permettant d'obtenir $\texttt{n!} = 1 \times \cdots \times \texttt{n}$.
  D'après l'énoncé, la commande `np.prod(L)` renvoie le produit des éléments de `L`.
  La variable `x` contient les entiers de 1 à `n`. Ainsi, pour obtenir `n!`, on peut utiliser la commande : `np.prod(x)`. On obtient donc :

```
5  v = np.prod(x)
```

  × De même, comme la variable `y` contient les entiers de 1 à `2n+1`, pour obtenir `(2n+1)!`, on peut utiliser la commande : `np.prod(y)`. On obtient :

```
6  w = np.prod(y)
```

  × Pour finir, on complète la ligne 7 de la façon suivante (toujours d'après la formule de ***5.b)***) :

```
7  u = (4**n) * (v**2) / w
```

Commentaire

- On pouvait aussi stocker la valeur `n!` dans la variable `v` à l'aide d'une boucle for :

```
1  v = 1
2  for i in range(1,n+1) :
3    v = v * i
```

- Comme dit précédemment, compléter correctement le programme **Python** démontre la bonne compréhension de la mécanismes en jeu et est suffisant pour obtenir les points alloués à cette question.

IV.4. ESSEC-I 2020

Après avoir établi la formule $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ lorsque $k \in [\![1, n]\!]$, écrire une fonction **Python** qui calcule les coefficients binomiaux.

Soit $k \in [\![1, n]\!]$.

- Tout d'abord :
$$k\binom{n}{k} = k\,\frac{n!}{k!\,(n-k)!} = \frac{n!}{(k-1)!\,(n-k)!}$$

- Par ailleurs :
$$n\binom{n-1}{k-1} = n\,\frac{(n-1)!}{(k-1)!\,((n-1)-(k-1))!} = \frac{n!}{(k-1)!\,(n-k)!}$$

Ainsi : $k\binom{n}{k} = n\binom{n-1}{k-1}$.

D'où : $\forall k \in [\![1, n]\!],\ \binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$.

Commentaire

La relation sur les coefficients binomiaux peut aussi se faire par dénombrement.
Pour ce faire, on considère un ensemble E à n éléments.
(on peut penser à une pièce qui contient n individus)
On souhaite alors construire une partie P à k éléments de cet ensemble contenant un élément distingué *(on peut penser à choisir dans la pièce un groupe de k individus dans lequel figure un représentant de ces individus).*
Pour ce faire, on peut procéder de deux manières :

1) On choisit d'abord la partie à k éléments de E : $\binom{n}{k}$ possibilités.

On distingue ensuite un élément de cet ensemble P : $\binom{k}{1} = k$ possibilités.
(on choisit d'abord les k individus et on élit ensuite un représentant de ces individus)

Ainsi, il y a $k\binom{n}{k}$ manières de construire P.

2) On choisit d'abord, dans E, l'élément à distinguer : $\binom{n}{1} = n$ possibilités.

On choisit ensuite $k-1$ éléments dans E qui, pour former P, en y ajoutant l'élément précédent : $\binom{n-1}{k-1}$ possibilités.
(on choisit d'abord le représentant puis on lui adjoint un groupe de $k-1$ individus)

Ainsi, il y a $n\binom{n-1}{k-1}$ manières de construire P.

On retrouve ainsi le résultat.

- En itérant la formule précédente, on obtient :
$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1} = \frac{n}{k}\,\frac{n-1}{k-1}\binom{n-2}{k-2} = \cdots = \frac{n\,(n-1)\cdots(n-k+1)}{k\,(k-1)\cdots 1}$$

(cette formule se démontre rigoureusement par récurrence)

- On propose alors la fonction **Python** suivante.

```
def CoeffBin(k, n) :
    c = 1
    for i in range(1,k+1) :
        c = c * (n - i + 1) / i
    return c
```

ou encore :

```
def CoeffBin(k, n) :
    if k==0 :
        c = 1
    else :
        c = (n / k) * CoeffBin(n - 1, k - 1)
    return c
```

IV.5. ECRICOME 2022

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par son premier terme $u_0 > 0$ et la relation de récurrence :

$$\forall n \in \mathbb{N}, \quad u_{n+1} = g(u_n) \qquad \text{où} \qquad \begin{array}{rccc} g & : & \mathbb{R}_+^* & \to & \mathbb{R} \\ & & x & \mapsto & \exp\left(\left(2 - \dfrac{1}{x}\right)\ln(x)\right) \end{array}$$

Écrire une fonction **Python** qui prend en argument un réel `u0` et un entier `n` et renvoie sous forme de liste les $n+1$ premières valeurs de la suite $(u_n)_{n\in\mathbb{N}}$ de premier terme $u_0 =$ `u0`.

On commence par coder la fonction g.

```
import numpy as np
def g(x) :
    y = np.exp( (2 - 1/x) * np.log(x) )
    return y
```

On propose ensuite la fonction **Python** suivante.

```
def Prem_Suite_u(u0, n) :
    U = [u0]
    for i in range(n) :
        U.append( g(U[i]) )
    return U
```

Détaillons les éléments de ce $2^{\text{ème}}$ script.

- **Début de la fonction**
  On commence par préciser la structure de la fonction :
  - × cette fonction se nomme `Prem_Suite_u`,
  - × elle prend en paramètre les variables `u0` et `n`,
  - × elle renvoie la valeur stockée dans la variable `U`.

```
1  def Prem_Suite_u(u0, n) :
```

```
5      return U
```

  On initialise ensuite la variable `U` à la liste contenant uniquement l'élément `u0`. C'est cette variable qui contiendra les `n` + 1 premiers termes de la suite (u_n).

```
2      U = [u0]
```

- **Structure itérative**
  Les lignes 3 à 4 consistent à mettre à jour la variable `U` pour que ses coordonnées contiennent les termes successifs de la suite (u_n). Pour cela on met en place une structure itérative (boucle for). À chaque itération, on concatène à la liste `U` le terme suivant de la suite (u_n) à l'aide de la relation de récurrence définissant cette suite.

```
3      for i in range(n) :
4          U.append( g(U[i]) )
```

- **Fin de la fonction**
  À l'issue de cette boucle, la variable `U` contient les `n` + 1 premiers termes de la suite (u_n).

Commentaire

- On décrit ici de manière précise les instructions afin d'aider le lecteur un peu moins habile en **Python**. Cependant, l'écriture du script démontre la compréhension de toutes les commandes en question et permet sans doute d'obtenir la totalité des points alloués à cette question.
- Si on avait souhaité coder une fonction qui renvoie seulement le $n^{\text{ème}}$ terme de la suite (u_n), on aurait modifié le script précédent de la façon suivante :

```
1  def Suite_u(u0, n) :
2      u = [u0]
3      for i in range(n) :
4          u = g(u)
5      return u
```