TP3 : Écriture de sommes finies en **Python**

▶ Dans votre dossier `Info_2a`, créer le dossier `TP_3`.

I. Avant propos

On considère dans ce TP les suites $(u_n)_{n\in\mathbb{N}}$ et (v_n) suivantes :

$$\forall n \in \mathbb{N},\ u_n = \frac{n^2}{3^n} \qquad \text{et} \qquad \begin{cases} v_0 = 1 \\ \forall n \in \mathbb{N},\ v_{n+1} = \dfrac{2\,v_n}{\mathrm{e}^{v_n} + \mathrm{e}^{-v_n}} \end{cases}$$

Objectif du TP : il s'agit d'explorer les différentes méthodes permettant le calcul des sommes partielles d'ordre n : $S_n = \sum_{k=0}^{n} u_k$ et $T_n = \sum_{k=0}^{n} v_k$.

II. Calcul des sommes partielles d'ordre n

II.1. Calcul de S_n

Il s'agit ici d'illustrer le cas où la suite (u_n) est donnée sous forme explicite.

II.1.a) Méthode itérative

▶ Écrire une fonction `calculSn` qui :
- × prend en paramètre un entier `n`,
- × renvoie une variable `S`,
- × à l'aide d'une structure itérative, calcule la valeur de $S_{\texttt{n}}$ et stocke le résultat dans `S`.

▶ Que vaut S_0 ? S_5 ? S_{10} ? S_{100} ? S_{1000} ?

II.1.b) Utilisation des fonctionnalités Python

- Écrire une fonction `premSuiteU` qui :
  - × prend en paramètre un entier `n`,
  - × renvoie une variable `U`, vecteur contenant initialement `n` + 1 zéros,
  - × à l'aide d'une structure itérative, calcule les `n` + 1 premières valeurs de la suite (u_n) et stocke le résultat dans `U`.

- Que réalisent les appels `V = [k for k in range(1,6)]`, puis `sum(V)` ?
  Détailler le rôle de la fonction `sum`.

- En déduire un appel permettant de calculer S_{10}.

- Une fonctionnalité de **Python** est de permettre la création de liste en compréhension. La syntaxe pour créer une liste en compréhension (ou parle aussi de compréhension de liste) est assez proche de celle utilisée en mathématique. Il s'agit de spécifier la propriété que doivent vérifier les éléments que l'on souhaite faire apparaître dans la liste. Typiquement, on peut demander de créer la liste contenant toutes les valeurs entières comprises entre `0` et `10` (exclu) ou encore la liste des cinq premiers carrés d'entiers. Pour ce faire, on procède généralement en demandant l'itération (à l'aide d'un `for`) des valeurs d'un intervalle (défini par un `range`).

```
Out [1] :   [k for k in range(10)]
In  [1] :   [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Out [2] :   [k**2 for k in range(1, 6)]
In  [2] :   [1, 4, 9, 16, 25]
```

  Il est aussi possible d'ajouter des conditions à l'aide de `if`. Il est par exemple possible de créer la liste de tous les carrés multiples de 6 d'entiers plus petits que 20.

```
Out [3] :   [k**2 for k in range(20) if k**2 % 6 == 0]
In  [3] :   [0, 36, 144, 324]
```

- Comment peut-on, à l'aide d'une compréhension de liste, créer la viste `U` des 101 premiers éléments de la suite (u_n) ?

► Comment obtient-on alors S_{100} à l'aide de `U` ? Et comment récupérer S_5 ?

II.2. Calcul de T_n

Il s'agit ici d'illustrer le cas où la suite (u_n) est donnée sous forme récurrente.

II.2.a) Méthode itérative

► Écrire une fonction `calculTn` qui :
- × prend en paramètre un entier `n`,
- × renvoie une variable `T`,
- × à l'aide d'une structure itérative, calcule la valeur de $S_{\texttt{n}}$ et stocke le résultat dans `T`.

On pourra utiliser une variable auxiliaire `v` afin de calculer les différents termes de (v_n).

► Que vaut T_0 ? T_{100} ? T_{10000} ?

II.2.b) Utilisation des fonctionnalités Python

► Écrire une fonction `premSuiteV` qui :
- × prend en paramètre un entier `n`,
- × renvoie une variable `V`, vecteur contenant initialement `n` + 1 zéros,
- × à l'aide d'une structure itérative, calcule les `n` + 1 premières valeurs de la suite (v_n) et stocke le résultat dans `V`.

► En déduire un appel permettant de calculer T_{10}.

III. Calcul des n premières sommes partielles

III.1. Calcul des n premières sommes partielles de $\sum u_n$

► Soit $n \in \mathbb{N}$. Quel lien y a-t-il entre S_n et S_{n+1} ?

► En tirant profit de l'égalité précédente, écrire une fonction `premSn` qui :

× prend en paramètre une variable `n`,

× renvoie une variable `tabS`, vecteur contenant initialement `n` + 1 zéros,

× à l'aide d'une structure itérative, stocke la valeur de S_i dans la $i^{\text{ème}}$ case de `tabS`.

On ne devra pas effectuer d'appel à `calculSn` mais on pourra s'inspirer de son code.

► Comme précédemment, on aurait aussi pu tirer parti des fonctionnalités de **Python**.
Que réalise les appels `V = [k for k in range(1,6)]` puis `np.cumsum(V)` ? Détailler le rôle de la fonction `cumsum`.

► Quel appel, tirant parti des fonctionnalités **Python**, permet d'obtenir les 101 premiers éléments de la suite (S_n) ? On utilisera la fonction `cumsum`.

III.2. Calcul des n premières sommes partielles de $\sum v_n$

▶ Soit $n \in \mathbb{N}$. Quel lien y a-t-il entre T_n et T_{n+1} ?

▶ En tirant profit de l'égalité précédente, écrire une fonction `premTn` qui :

- × prend en paramètre une variable `n`,
- × renvoie une variable `tabT`, vecteur contenant initialement `n` + 1 zéros,
- × à l'aide d'une structure itérative, stocke la valeur de T_i dans la $i^{\text{ème}}$ case de `tabT`.

On ne devra pas effectuer d'appel à `calculTn` mais on pourra s'inspirer de son code. On pourra notamment utiliser une variable `v` calculant les valeurs successives des termes de la suite (v_n).

À retenir : on a étudié deux méthodes en **Python** pour obtenir les éléments de (S_n).

1) La suite des sommes partielles étant une suite (grande nouvelle), on peut se servir des procédés vus en TP1.

2) On peut aussi préférer créer la liste des premiers éléments de la suite (u_n) et utiliser les instructions `sum` ou `cumsum` suivant ce que l'on cherche à obtenir. Encore une fois, il faut se reporter au TP1.

IV. Tracé des sommes partielles de $\sum u_n$

▶ Compléter le programme suivant afin qu'il permette d'effectuer le tracé des 51 premiers éléments de (S_n). On exécutera ce programme.

```
1  import matplotlib.pyplot as plt
2  n = 100
3  N = [k for k in range(n+1)]
4  U = [k**2 / 3**k for k in range(n+1)]
5  tabS = np.cumsum(U)
6  plt.plot(N, tabS, 'rx')
```

▶ Quelle conjecture peut-on émettre sur la nature de la série $\sum u_n$?

V. Suite des sommes partielles aux concours

V.1. EML 2015

On considère l'application $f : \mathbb{R} \to \mathbb{R}$, $x \mapsto f(x) = x^3\, e^x$
et la suite réelle $(u_n)_{n\in\mathbb{N}}$ *définie par :* $u_0 = 1$ *et* $\forall n \in \mathbb{N}, u_{n+1} = f(u_n)$.
Il était demandé de démontrer que la série $\displaystyle\sum_{n\geqslant 1} \frac{1}{f(n)}$ converge (de somme S) et que :

$$\forall n \in \mathbb{N}^*,\ \left| S - \sum_{k=1}^{n} \frac{1}{f(k)} \right| \leqslant \frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n}$$

► *En déduire une fonction* ***Python*** *qui calcule une valeur approchée de* S *à* 10^{-4} *près.*
(on pourra se reporter au TP2)

V.2. ECRICOME 2018

Pour tout entier naturel n non nul, on pose : $u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.

- *Écrire une fonction d'en-tête* `def u(n) :` *qui prend en argument un entier naturel n non nul et qui renvoie la valeur de u_n.*

Commentaire

V.3. ESSEC-II 2019

On souhaite écrire une fonction en **Python** pour calculer l'entropie d'une variable aléatoire X dont le support de la loi est de la forme $A = \{0, 1, \ldots, n\}$ où n est un entier naturel. On suppose que la liste `P` de **Python** est telle que pour tout k de A, `P[k+1]` $= \mathbb{P}([X = k])$. Compléter la fonction ci-dessous d'argument `P` qui renvoie l'entropie de X, c'est-à-dire $-\sum_{k=0}^{n} \mathbb{P}([X = k]) \log_2\big(\mathbb{P}([X = k])\big)$, où :

$$\begin{array}{rccc} \log_2 & : & \mathbb{R}_+^* & \to & \mathbb{R} \\ & & x & \mapsto & \dfrac{\ln(x)}{\ln(2)} \end{array}$$

```
1  def Entropie(P) :
2      ...
3      return h
```

Si nécessaire, on pourra utiliser l'instruction `len(P)` qui donne le nombre d'éléments de `P`.

Remarque : Ici, la liste `P` doit contenir les valeurs :

$$\Big[\ \mathbb{P}([X = 0]),\ \mathbb{P}([X = 1]),\ \ldots,\ \mathbb{P}([X = n])\ \Big]$$

Ainsi :

× `P[0]` (premier élément de la liste `P`) contient $\mathbb{P}([X = 0])$.

× `P[1]` (deuxième élément de la liste `P`) contient $\mathbb{P}([X = 1])$.

× ...

× `P[n]` ($(n+1)^{\text{ème}}$ élément de la liste `P`) contient $\mathbb{P}([X = n])$.

Autrement dit, pour tout `k` de $[\![0, n]\!]$, `P[k]` contient $\mathbb{P}([X = k])$.

V.4. ESSEC-I 2021

L'énoncé définissait la suite (z_n) suivante :

$$\begin{cases} z_0 = 1 \\ \forall n \in \mathbb{N},\ z_{n+1} = \dfrac{n+2}{n+1} \displaystyle\sum_{k=0}^{\min(n,d)} \alpha_k\ z_{n-k} \end{cases}$$

On note Δ la liste $[\alpha_0,\ \cdots,\ \alpha_d]$.
Écrire une fonction **Python** d'entête `def z(Delta,n)` qui calcule z_n si `Delta` représente la liste Δ et stocke le résultat dans une variable `r`.