Python 1
Exercices sur les suites et sommes

Pour les graphiques la bibliothèque à importer est **matplotlib.pyplot** que l'on importera ainsi :

```
import matplotlib.pyplot as plt
```

Pour représenter une liste **y** en fonction d'une liste **x** :
x = liste des abscisses (ou tableau)
y = liste des ordonnées (ou tableau)
plt.plot(x,y)
plt.show()

☞ Rappel...
Par défaut, les points ainsi représentés sont reliés. Pour obtenir des points isolés, on ajoute un style en option, exemples :
- `plt.plot(x,y,'+')` : marque les points avec des +
- `plt.plot(x,y,'o')` : marque les points avec des •

Dans cette fiche, nous allons reprendre des questions classiques sur des suites et sommes : calculs de termes, représentations graphiques, calculs de sommes, création de listes de termes...

I Petit mélange...

On considère les suites $(s_n)_{n\in\mathbb{N}}$, $(t_n)_{n\in\mathbb{N}}$, $(u_n)_{n\in\mathbb{N}}$, $(v_n)_{n\in\mathbb{N}}$ et $(w_n)_{n\in\mathbb{N}}$ définies par :

$$\forall n\in\mathbb{N},\ s_n=\frac{n^2}{2^n}\ ;\ \begin{cases} t_0=1 \\ \forall n\in\mathbb{N},\ t_{n+1}=\mathrm{e}^{-t_n}\end{cases}\ ;\ \begin{cases} u_1=1 \\ \forall n\in\mathbb{N}^*,\ u_{n+1}=u_n+\dfrac{1}{n^2u_n}\end{cases}\ ;\ \begin{cases} v_0=v_1=1 \\ \forall n\in\mathbb{N},\ v_{n+2}=v_{n+1}+3v_n\end{cases}\ ;\ \forall n\in\mathbb{N}^*,\ w_n=\frac{(2n)!}{n^n}$$

1. 1.a. Écrire une fonction **liste_suite_s** prenant en argument d'entrée un entier naturel n et renvoyant la liste composée des valeurs s_0 à s_n.
   1.b. Représenter alors les termes s_0 à s_{10} sur un graphique.

```python
import matplotlib.pyplot as plt

def liste_suite_s(n):
    L=[k**2/(2**k) for k in range(0,n+1)]
    return L

plt.plot(range(0,11),liste_suite_s(10),"+")
plt.show()
```

2. 2.a. Écrire une fonction **suite_t** prenant en argument d'entrée un entier naturel n et renvoyant la valeur de t_n.
   2.b. En utilisant la fonction **suite_t**, écrire une fonction **somme_t** prenant en argument d'entrée un entier naturel n et renvoyant la valeur de $\sum_{k=0}^{n} t_k$. Quel est l'inconvénient de cette méthode ? En proposer une autre.
   2.c. Sans utiliser la fonction **suite_t**, écrire une fonction **liste_suite_t** prenant en argument d'entrée un entier naturel n et renvoyant la liste composée des valeurs t_0 à t_n.

```python
import numpy as np

def suite_t(n):
    t=1
    for k in range(1,n+1):
        t=np.exp(-t)
    return t

def somme_t(n):
    L=[suite_t(k) for k in range(0,n+1)]
    return sum(L)

def somme_t_bis(n):
    t,S=1,1
    for k in range(1,n+1):
        t=np.exp(-t)
        S=S+t
```

```
    return S

def liste_suite_t(n):
    t,T=1,[1]
    for k in range(1,n+1):
        t=np.exp(-t)
        T.append(t)
    return T

def liste_suite_t_bis(n):
    T=[1]
    for k in range(1,n+1):
        t=np.exp(-T[k-1])
        T.append(t)
    return T
```

3. Écrire une fonction **`suite_u`** prenant en argument d'entrée un entier naturel non nul n et renvoyant la valeur de u_n.

```
def suite_u(n):
    u=1
    for k in range(2,n+1):
        u=u+1/((k-1)**2*u) #attention : k = rg du terme calculé
    return u
```

4. Écrire une fonction **`suite_v`** prenant en argument d'entrée un entier naturel n et renvoyant la valeur de v_n. *Donner deux versions, dont une récursive.*

```
def suite_v(n):
    if n==0 or n==1:
        return 1
    else:
        u,v=1,1
        for k in range(2,n+1):
            u,v=v,v+3*u
        return v

def suite_v_bis(n):
    if n==0 or n==1:
        return 1
    else :
        return suite_v_bis(n-1)+3*suite_v_bis(n-2)
```

5. A l'aide d'une liste définie en compréhension, écrire une fonction **`suite_w`** prenant en argument d'entrée un entier naturel non nul n et renvoyant la valeur de w_n.

Aide

La commande **`np.prod(L)`** renvoie le produit des nombres de la liste **`L`**.

```
import numpy as np

def suite_w(n):
    L=[k for k in range(1,2*n+1)]
    N=np.prod(L)
    return N/(n**n)
```

6. Représenter la fonction $x \longmapsto e^{-x}$, la première bissectrice ainsi que les termes de la suite $(t_n)_{n\in\mathbb{N}}$ sur un même graphique, en faisant apparaître les termes de $(t_n)_{n\in\mathbb{N}}$ en une spirale.

```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(0,1,100)
plt.plot(x,x)
plt.plot(x,np.exp(-x))

t=0
for k in range(1,11):
    plt.plot([t,t],[t,np.exp(-t)],'r')
    plt.plot([t,np.exp(-t)],[np.exp(-t),np.exp(-t)],'r')
    t=np.exp(-t)

plt.show()
```

II Séries harmoniques

7. Recopier et exécuter le programme suivant :

```
import matplotlib.pyplot as plt
import numpy as np

L=[1/k for k in range(1,101)]
S=np.cumsum(L)
x=np.linspace(1,100,10000)
plt.plot(range(1,101),S,'r+')
plt.plot(x,np.log(x))
plt.show()
```

Que permet la commande `np.cumsum(L)` ? Que met en évidence le graphique obtenu ?

- La commande `np.cumsum(L)` prend en argument d'entrée une liste `L` et renvoie en sortie une liste `M` telle que, pour tout $k \in [0, len(L)]$, l'élément k de `M` est la somme des éléments 0 à k de `L`. C'est donc une commande très pratique pour créer une liste de termes d'une suite définie par une somme.
- Ce graphique met en évidence l'équivalent classique : $\sum_{k=1}^{n} \frac{1}{k} \underset{n\to+\infty}{\sim} \ln(n)$.

8. Adapter le programme précédent afin qu'il affiche les 100 premiers termes de la suite des sommes partielles de la série $\sum_{n\geq 1} \frac{(-1)^n}{n}$. Que peut-on conjecturer ? Comment pourrait-on démontrer cette conjecture ?

```
import matplotlib.pyplot as plt

L=[(-1)**k/k for k in range(1,101)]
S=np.cumsum(L)
plt.plot(range(1,101),S,'r+')
plt.show()
```