TP # 10

Quelques révisions et compléments

Exercice 1

1.(a) Proposer une fonction qui calcule $n!$:

```
def factorielle(n) :
```

2. Pour $0 \leqslant p \leqslant n$, rappeler la définition avec des factorielles :

$$\binom{n}{p} = \text{.....}$$

Proposer alors une fonction `binome` qui calcule $\binom{n}{p}$ (avec $0 \leqslant p \leqslant n$) :

```
def binome(n,p) :
```

3. Pour $0 \leqslant p \leqslant n$, écrire $\binom{n}{p}$ comme un produit de seulement p termes :

$$\binom{n}{p} = \text{.....}$$

(Remarque : cette formule a l'avantage de marcher également lorsque $p > n$...)
En déduire un autre moyen de calculer ce coefficient en Python :

```
def binome2(n,p) :
```

4. Comparer les temps de calcul de $\binom{99\,999}{3}$ avec les fonctions `binome` et `binome2`.
Ceci explique pourquoi on essaye en général de se passer des factorielles.

Exercice 2

On définit pour tout $n \in \mathbb{N}$ et $x \in \mathbb{R}$, $\quad S_n(x) = \sum_{k=0}^{n} \frac{x^k}{k!}$.

1. Proposer une première façon de calculer la valeur de la somme $S_n(x)$, en utilisant la fonction `factorielle` précédente

```
def somme(n,x) :
```

2. En remarquant que $S_n(x) = \sum_{k=0}^{n} a_k$ avec $a_0 = \text{.....}$ et $a_{k+1} = a_k \times \text{.....}$,
proposer une autre façon de calculer cette somme :

```
def somme2(n,x) :

  S = ... ; a = ...

  for k in range( .......... ) :
    S = S + a
    a = a * ..........................

  return S
```

Vérification (analyse du programme)

3. Comparer les temps de calcul de $S_{3000}(2)$ avec `somme` et `somme2`.

La fonction `somme` est très couteuse en calcul, car pour chaque nouveau terme à additionner dans la somme, on calcule $k!$ "à partir du début". La fonction `somme2` exploite une relation de récurrence entre les termes de la somme pour économiser les calculs.

Bonus : Penser à une approche similaire pour calculer $S_n = \sum_{k=1}^{n} \frac{(-1)^k}{(2k-1)!}$.

Retour sur les calculs de sommes/produits : En fait, si la liste des valeurs à sommer ou à multiplier est facile à construire, on dispose des instructions suivantes :

`np.sum` et `np.prod` : somme/produit des termes d'une liste !

Après avoir importé la bibilothèque numpy, si `L = [` $a_0, a_1, \dots, a_n$ `]` est une liste de valeur, les instructions

`np.sum(L)` et `np.prod(L)`

renvoient respectivement $\sum_{k=0}^{n} a_k$ et $\prod_{k=0}^{n} a_k$

Exemple : `np.sum( [ 1/i for i in range(1,6)] )` renvoie la valeur de $\sum_{i=1}^{5} \frac{1}{i}$.

Exercice 3

1. Proposer une fonction qui prend en entrée n et calcule $\sum_{k=1}^{n} \frac{(-1)^k}{k^2}$:

```
def somme(n) :
```

2. Proposer une nouvelle façon concise de calculer $n!$ et $\binom{n}{p}$.
(il vaut mieux voir $\binom{n}{p}$ comme un produit de p termes) :

```
def factorielle(n) :
```

```
def binome(n,p) :
```

Exercice 4

Proposer une fonction qui prend en entrée un réel $a > 0$ et renvoie le plus petit indice $n \geqslant 2$ tel que $\frac{\ln(n)}{\sqrt{n}} \leqslant a$. (Pourquoi un tel indice existe bien ?)

Exercice 5

On pose $a_0 = 1$, $b_0 = 2$ et $\forall n \in \mathbb{N}$, $a_{n+1} = \sqrt{a_n b_n}$ et $b_{n+1} = \frac{a_n + b_n}{2}$.
On admet que (a_n) est croissante, (b_n) décroissante et $\lim_{n\to+\infty} (a_n - b_n) = 0$.
Ces deux suites sont ainsi adjacentes, donc convergent vers un même réel ℓ.

1. Pour tout $n \in \mathbb{N}^*$, quel encadrement de ℓ peut-on donner ?

..... $\leqslant \ell \leqslant$ .....

2. Compléter la fonction suivante, qui prend en entrée un réel $\varepsilon > 0$, pour qu'elle renvoie un encadrement de la valeur de ℓ à ε près.

```
import numpy as np

def approx(eps) :

  a = .... ; b = ....

  while ...................... :









  return( ..... , ..... )
```

3. Et si l'on veut juste une approximation de ℓ à ε près par valeur inférieure ?