.

NOM	CORRIGE.....
PRENOM	DU PROF.....

Exercices

Calculs de Sommations

L'objectif de cette séance est de comprendre le lien étroit entre l'instruction `for` et l'usage de Σ.

1. A partir de l'exemple de l'aide-mémoire, organiser le calcul de $S = \sum_{k=1}^{2024} k$:

```
S=0
for k in range(1,2025):
        S=S+K
print(S)
```

donner le résultat affiché :

```
|
-> 2 049 300
```

puis vérifier le résultat avec la formule adéquate du cours :

$$\text{formule : } \sum_{k=0}^{n} k \ = \ \frac{n(n+1)}{2}$$

Valeur de vérification calculée à la console :

```
>>> n = 2024
>>> n*(n+1)/2
|
-> 2 049 300.0
```

2. Reproduire la méthode pour obtenir un résultat de la somme $T = \sum_{k=1}^{161} k^2$:

Script itératif proposé :

```
T=0
for k in range(1,162):
        T=T+k**2
print(T)
```

Résultat affiché :

```
|
-> 1 404 081
```

puis vérifier le résultat avec une formule adéquate (généreusement fournie) :

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

Valeur de vérification calculée à la console :

```
>>> n = 161
>>> n*(n+1)*(2*n+1)/6
|
-> 1 404 081.0
```

3. *En autonomie !* Traiter le calcul de $V = \sum_{k=4}^{1601}(2k-1)^2$ de façon analogue :

Script :

```
V=0
for k in range(4,1602):
        V=V+(2*k-1)**2
print(V)
```

Calcul théorique (formules) :

On calcule $\sum_{k=1}^{3}(2k-1)^2 = (2-1)^2 + (4-1)^2 + (6-1)^2 = 35$ et ainsi :

$$\begin{aligned}
\sum_{k=4}^{1601}(2k-1)^2 &= \sum_{k=1}^{1601}(4k^2-4k+1) - 35 = 4\sum_{k=1}^{1601}k^2 - 4\sum_{k=1}^{1601}k + 1601 - 35 \\
&= 4\frac{n(n+1)(2n+1)}{6} - 4\frac{n(n+1)}{2} + 1566 \quad \text{avec } n = 1601 \\
&= 2\frac{n(n+1)(2n+1)}{3} - 2n(n+1) + 1566 \quad \text{avec } n = 1601
\end{aligned}$$

Valeur de vérification :

```
>>> n = 1601
>>> A = 2*n*(n+1)*(2*n+1)/3
>>> A - 2*n*(n+1)+1566
-> 5 471 579 165.0
```

4. *Et en choisissant n ?* On considère la somme $G_n = \sum_{k=0}^{n}\frac{1}{3^k}$.

On va améliorer notre script pour qu'il puisse calculer directement G_n à l'appel de l'entier n.
On peut commencer par vérifier qu'une valeur entrée est bien un entier naturel (vous rappelez-vous) ?

```
def sumG(n):
        if n==floor(n) & n>=0 :
                G=0
                for k in range(n+1):
                        G=G+1/(3**k)
                        return(G)
        else :
                print("entree n non entiere")
```

Et sa traditionnelle gamme de tests (on pensera à vérifier le comportelent de `sumG` lorsque l'entrée n'est pas un entier naturel)

$n=$	0	2	10	25	-2	3.56	150
`sumG`(n)	1.000	1.444	1.499	1.499	erreur	erreur	1.500

Calculs théoriques (formules : celle-là vous êtes censés la (re)connaître) :

$$\begin{aligned}\sum_{k=0}^{n}\frac{1}{3^k} &= \frac{1-\left(\frac{1}{3}\right)^{n+1}}{1-\frac{1}{3}} = \frac{1}{2/3}\left(1-\frac{1}{3^{n+1}}\right)\\ &= \frac{3}{2}-\frac{1}{2\cdot 3^n}\end{aligned}$$

5. *Et maintenant à la chaîne*

Pour chacune des sommes suivantes, programmer une fonction Python utilisant une boucle `for` permettant le calcul, n étant donné en entrée, de :

$$1)\ B_n=\sum_{k=0}^{n}\left(2(-1)^k+1\right)\quad 2)\ H_n=\sum_{k=1}^{n}\frac{1}{k}\quad 3)\ A_n=\sum_{k=0}^{n}|15-k|$$

Script pour B_n :

```
def sumB(n):
        if n==floor(n) & n>=0 :
                B=0
                for k in range(n+1):
                        B=B+(2*(-1)**k + 1)
                        return(B)
        else :
                print("entree n non entiere")
```

Script pour H_n :

```
def sumH(n):
        if n==floor(n) & n>=0 :
                H=1
                for k in range(1,n+1):
                        H=H+1/k                 #attention à la division par 0
                return(H)
        else :
                print("entree n non entiere")
```

Script pour A_n

```
def sumA(n):
        if n==floor(n) & n>=0 :
                A=0
                for k in range(n+1):
                        A=A+abs(15-k)
                return(A)
        else :
                print("entree n non entiere")
```

puis tester vos fonctions sur chacune de ces sommes pour $n = 21$ et indiquer les résultats :

```
>>> B(21)
-> 24.0
>>> H(21)
-> 3.645358
>>> A(21)
-> 141.0
```

6. *Attardons-nous sur une somme particulière :* Générer la liste des valeurs $[B_0\ ;\ B_1\ ;\ \ldots\ ;\ B_{20}]$ (instructions, pas les valeurs)

```
L=np.zeros([21])
for k in range(21):
        L[k]=B(k)
print(L)
```

Quelle conjecture peut-on effectuer sur le comportement de la suite $(B_n)_{n\in\mathbb{N}}$?

```
sumB(n) renvoie n+1 ou n+3 alternativement
```

Vous pourriez même réussir à démontrer formellement votre conjecture ! ! !

$$\begin{aligned} B_n = \sum_{k=0}^{n} \left(2(-1)^k + 1\right) &= 2\sum_{k=0}^{n}(-1)^k \quad + (n+1) = 2 \times \frac{1-(-1)^{n+1}}{1-(-1)} + (n+1) \\ &= n + 1 + 1 + (-1)^n = n + 2 + (-1)^n \end{aligned}$$

ce qui donne bien, pour n pair que $B_n = n + 3$ et pour n impair que $B_n = n + 1$.

7. Sommations par commande directe : Au moyen de la commande `import numpy as np`, on peut utiliser directement ces instructions Python pour réaliser des calculs de sommes :
   — On peut utiliser `np.sum({ndarray})` pour le calcul de la somme des éléments listés dans un objet de type `ndarray` par exemple, taper les commandes suivantes puis comparer avec $1 + 3 + 5 + 7 + 9$:

```
>>>L=np.array([1,3,5,7,9])
>>>S=np.sum(L)
-> 25
```