.

| | |
|---|---|
| NOM | ..... |
| PRENOM | ..... |

# Exercices

### Calculs de Sommations

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

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

```
S=.............
for k in range(1,2025):
        ..........................
print(.......................)
```

donner le résultat affiché :

```
...
...
```

puis vérifier le résultat avec la formule adéquate du cours :

```
forumle :
```

Valeur de vérification calculée à la console :

```
>>>
>>>
>>>
...
```

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

Script itératif proposé :

```
T=.............
for ........................
        ..........................
print(.......................)
```

Résultat affiché :

```
...
...
```

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 :

```
>>>
>>>
>>>
...
```

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

Script :

```
...
...
...
...
...
...
...
...
...
```

Calcul théorique (formules) :

```
...
...
...
...
...
...
...
```

Valeur de vérification :

```
>>>
>>>
>>>
...
```

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 ..................... :
                ...
                ...
                ...
                ...
                ...
                ...
        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)

```
...
...
...
...
```

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

```
...
...
...
...
...
...
```

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 $B_n$ :

```
...
...
...
...
...
...
...
```

Script pour $H_n$ :

```
...
...
...
...
...
...
...
```

Script pour $A_n$

```
...
...
...
...
...
...
...
```

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

```
>>>
...
>>>
...
>>>
...
```

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

```
...
...
...
...
...
...
```

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

```
...
...
```

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

```
...
...
...
...
...
...
...
...
...
...
```

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)
....
```