Ce TP illustre l'utilisation de **Python** pour calculer les termes de suites récurrentes. On verra aussi des calculs de sommes ou de produits.

I Suites récurrentes :

Exemple : Dans l'éditeur, ouvrez une nouvelle page, puis créez la fonction suivante :

```
1 def suite_arithmetique(n,u0,r) :
2     """ Cette fonction calcule et renvoie le n-ième terme de la suite arithmétique (u_n) de premier terme u_0 = u0 et de raison r """
3     u=u0 # Initialisation
4     for k in range(1,n+1) :
5         u=u+r # ou u+=r
6     return u
```

Enregistrez cette fonction dans un fichier, puis retournez dans la console, exécutez le fichier et tapez les instructions suivantes :

☞ $suite\_arithmetique(12,3,-2)$ ☞ $3-2\times 12$

Exercice 1: Compléter la fonction suivante :

```
1 def suite_geometrique(n,u0,q) :
2     """ Cette fonction calcule et renvoie le n-ième terme de la suite géométrique (u_n) de premier terme u_0 = u0 et de raison q """
3     u=u0 # Initialisation
4     for k in range(1,n+1) :
5         ...
6     return u
```

Exercice 2: Ecrire une fonction $suite\_aritmetico\_geometrique(n,u0,a,b)$ qui calcule et renvoie le $n-$ième terme de la suite arithmético-géométrique (u_n) de premier terme $u_0=u0$ et définie par : $\forall n\in\mathbb{N},\ u_{n+1}=au_n+b$

Exercice 3: Ecrire une fonction recudouble$(n,a,b,alpha,beta)$ qui donne la valeur de u_n pour un n donné avec :

$$u_{n+2}=\alpha u_{n+1}+\beta u_n,\quad u_0=a,u_1=b,\alpha=alpha,\beta=beta$$

Tester la fonction avec $\alpha=\beta=a=b=1$, et déterminer une valeur approchée de $\dfrac{u_{n+1}}{u_n}$ pour les grandes valeurs de n. Comparer avec le nombre d'or.

Exercice 4: Modifier la fonction précédente pour renvoyer la liste $[u_0,u_1,\cdots,u_n]$ au lieu de simplement u_n.

II Sommes et produits

```
1 def somme_carre(n) :
2     """ Cette fonction calcule et renvoie la somme des n premiers carrés, 1**2+2**2+...+n**2 """
3     s=0 # Initialisation
4     for k in range(n+1) :
5         s=s+k**2 # ou s+=k**2
6     return s
```

Enregistrez cette fonction dans un fichier, puis retournez dans la console, exécutez le fichier et tapez les instructions suivantes :

☞ $s=somme\_carre(7)$ ☞ s ☞ $t=7*8*15/6$ ☞ t

Exercice 5: Ecrire une fonction sommeimpair(n) qui étant un entier n donné, renvoie la valeur de $\displaystyle\sum_{k=0}^{n}(2k+1)$.
Tester cette fonction avec quelques valeurs de n, puis conjecturer le résultat général. Vérifier votre hypothèse.

Exercice 6: Ecrire une fonction sommegeo(n,x) qui étant donnés un entier n et un réel x, renvoie la valeur de $\displaystyle\sum_{k=0}^{n}x^k$.
Tester cette fonction avec des grandes valeurs de n et différentes valeurs de x. Que peut-on conjecturer sur les limites éventuelles de cette somme quand $[n\to+\infty]$.

Exercice 7: Ecrire une fonction factorielle(n) qui étant un entier n donné, renvoie la valeur de $n!$.
Tester cette fonction avec quelques valeurs de n.

Exercice 8: Ecrire une fonction $c =$ coeffbinome(n, p) qui étant donnés deux entiers n et p, renvoie la valeur de $\binom{n}{p}$.
Que renvoie la fonction avec $n = 10000$ et $p = 1$? Puis avec $n = 50000$ et $p = 1$?
On peut mesurer le temps d'exécution d'un programme avec les instructions suivantes :
from time import perf_counter
Dans la console :
☞ $t1 = perf\_counter(); c =$ coeffbinome$(100000, 1); t2 = perf_counter(); print(t2 - t1)$
Etes-vous capable de faire ce calcul de tête ? Améliorer alors la fonction en constatant que : $\binom{n}{p} = \dfrac{n!}{p!(n-p)!} = \dfrac{n \times (n-1) \times \ldots \times (n-p+1)}{p \times (p-1) \times \ldots \times 1}$
$\binom{n}{p} = \dfrac{n-p+1}{1} \times \dfrac{n-p+2}{2} \times \ldots \times \dfrac{n-p+p}{p}$

Exercice 9: En utilisant la fonction donnant $\binom{n}{k}$ écrite précédemment, écrire une fonction
sommebinome(n, a, b) qui étant donnés un entier n et deux réels a et b, renvoie la valeur de $\sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$. Comparer le résultat obtenu avec la valeur de $(a+b)^n$.
Ecrire une fonction sommebinomepair(n) qui étant donnés un entier n, renvoie la valeur de $\sum_{0 \leq 2k \leq n} \binom{n}{2k}$. Conjecturer le résultat général.

Exercice 10: Ecrire une fonction doubleproduit(n) qui étant un entier n donné, renvoie la valeur de $\prod_{1 \leq i,j \leq n} ij$. Tester cette fonction avec quelques valeurs de n, puis conjecturer le résultat général. Vérifier votre hypothèse.

III La suite de Syracuse

Exercice 11: L'algorithme de Syracuse consiste à itérer l'opération suivante : à un nombre entier n, on associe $n/2$ si n est pair et $3n+1$ si n est impair. On conjecture (on ne sait toujours pas si c'est vrai) que quelque soit l'entier considéré initialement dans cet algorithme, on arrive toujours à 1 après un certain nombre d'itérations. C'est en tout cas vrai pour tous les entiers avec lesquels l'algorithme a été testé.
Notons qu'alors la suite devient périodique : $1, 4, 2, 1, 4, 2, 1, 4, 2, \cdots$.

a Ecrire une fonction recu(n, a) qui calcule et renvoie le $n-$ième terme de la suite de Syracuse avec $u_0 = a \in \mathbb{N}$.

b Ecrire une fonction reculiste(n, a) qui calcule et renvoie la liste des n premiers termes de la suite de Syracuse avec $u_0 = a \in \mathbb{N}$.

c Ecrire une fonction syracuse(a) qui calcule et renvoie le premier entier n tel que $u_n = 1$.

d Ecrire une fonction maxisyracuse(a) qui calcule et renvoie la valeur maximale atteinte par la suite de Syracuse initialisée par $u_0 = a \in \mathbb{N}$.

e Ecrire une fonction longueursyracuse(n) qui calcule et renvoie l'entier $a \leq n$ pour lequel la suite de Syracuse compte le plus grand nombre de termes avant 1.