SUITES RÉCURRENTES - ALGORITHMES À CONNAÎTRE

Question : donner le terme u_n de la suite définie par une relation de récurrence.

```
# suite récurrente d'ordre 1
# exemple simple u(n+1) = u(n)**2 + 1
def recc(u0,n):
    u = u0
    for i in range(n):
        u = u**2 + 1
    return u

# suite récurrente d'ordre 2
# exemple simple u(n+2) = 2.u(n+1)-u(n)+3
def recc2(u0,u1,n):
    (u,v) = (u0,u1)
    for i in range(n):
        (u,v) = (v,2*v-u+3)
    return u
```

type de boucle : for

- On sait à l'avance combien de calculs on doit réaliser : n si on commence à u_0 (de u_1 à u_n).
  Si on commence à u_1, cela fait $n-1$ calculs…
- si on utilise **i in range(n)**, avec la donnée u_0, cela veut dire que $i = 0$ lors du calcul de u_1. Attention à ne pas décaler les indices.

Nombre de termes en mémoire :

- Pour une suite récurrente d'ordre 1, on n'a besoin que d'un seul terme en mémoire.
- pour une récurrence d'ordre 2, il en faut 2 : u_n et u_{n+1} pour calculer u_{n+2}.
  → affectation par **couples** : u prend l'ancienne valeur de v et v est calculée par la relation de récurrence.
  L'affectation est simultanée !
- pour une récurrence d'ordre 3, il en faut 3… et ainsi de suite.
  → affectation par triplets…

Erreurs fréquentes :

- Calculer le mauvais nombre de termes : u_{n-1} au lieu de u_n.
  → tester à la main pour de petites valeurs de n.
- Décaler les indices lors du calcul de u_n.
  → tester à la main pour les premiers pas de la boucle.

Dans la suite, on travaille avec la récurrence d'ordre 1.

À vous de généraliser à une récurrence d'ordre supérieure.

Question : donner le premier rang n tel que $u_n \geq M$

```
# avec relation ordre 1
def rang(u0,M):
    u = u0; n = 0
    while u < M:
        u = u**2 + 1
        n += 1
    return n
```

Vérification préalable :

- S'assurer qu'un tel rang existe ! c'est-à-dire que M n'est pas un majorant de la suite.

Type de boucle : while

- Car on ne sait pas à l'avance combien de calculs vont être réalisés.
  → On effectue les calculs **tant que** le contraire de la relation est vérifié : $u_n < M$.

Erreurs fréquentes :

- Se tromper d'un indice sur n.
  → tester à la main en supposant que l'on dépasse M dès les premières valeurs.
- Oublier d'initialiser et d'incrémenter l'indice.

Question : donner le premier terme u_n tel que $u_n \geq M$

```
def termeSup(u0,M):
    u = u0
    while u < M:
        u = u**2 + 1
    return u
```

Idem, mais on renvoie le terme de la suite au lieu de son indice.

Ici, l'indice ne sert pas dans le calcul de la suite, on n'en a donc pas besoin, mais il pourrait intervenir s'il apparaissait dans la relation de récurrence.

Question : donner le dernier terme u_n tel que $u_n < M$

```
# on suppose M > u0, sinon renvoie une erreur
def termeInf(u0,M):
    u = u0
    while u < M:
        (u,v) = (u**2 + 1,u)
    return v
```

Vérification préalable :

- S'assurer qu'un tel rang existe ! c'est-à-dire que M n'est pas un majorant de la suite.

Type de boucle : while

- On ne sait pas à l'avance combien de calculs vont être réalisés.

L'effet mémoire : On ne sait pas à l'avance quand on va dépasser M : lorsque l'on s'en rend compte, il est déjà *trop tard* et on est déjà au terme suivant.

Pour ne pas avoir à recalculer le terme précédent, on le garde en mémoire d'une fois sur l'autre.

→ même méthode que la récurrence d'ordre 2 avec affectation par couple.

Question : Conjecturer la monotonie de la suite.

```
def croissante(u0,n):
    u = u0
    for i in range(n):
        (u,v) = (u**2 + 1, u)
        if u < v:
            return False
    return True
```

Vérification préalable :

- On peut déjà calculer quelques termes pour se faire une idée.

Type de boucle : for

- On va vérifier notre conjecture sur un nombre de termes prédéfinis.
  On sait à l'avance jusqu'où on teste (sinon, on s'expose au risque d'une boucle infinie).

Optimisation :

- Dès que l'on rencontre un terme qui contredit notre conjecture, on s'arrête avec **return**.
  Rien ne sert de faire la boucle jusqu'au bout, si on sait que la conjecture est fausse.
- On calcule la suite en même temps que l'on vérifie la relation de récurrence.
  Il ne faut pas utiliser la fonction définie précédemment **recc**, sinon, pour chaque calcul de u_n, on repartirait de u_0 alors que l'on vient de calculer u_{n-1}.

Question : Calculer la somme des termes de la suite

```
def somme(u0,n):
    u = u0 ; s = u0
    for i in range(n):
        u = u**2 + 1
        s += u
    return s
```

Type de boucle : for

- On sait au départ le nombre de termes à sommer.

Optimisation :

- Comme pour la conjecture sur la monotonie, il faut utiliser la relation de récurrence pour limiter les calculs.

Erreurs fréquentes :

- sommer le mauvais nombre de termes.
  → vérifier à la main pour de petites valeurs de n.