Algorithmique et programmation – Terminale spécialité

Chapitre 2 – Calculer les termes d'une suite

Pr. Elakili · Algorithme · Python · TI-82 · Octobre 2026

Site web : missionmaths360.com

Méthode

Pour une suite définie par u_0 et $u_{n+1} = f(u_n)$, on part de u_0 et on applique n **fois** la relation de récurrence : on sait combien de tours faire, donc on utilise une boucle **Pour**. Une seule variable u suffit : à chaque tour, la nouvelle valeur **remplace** l'ancienne.

Pack 1 – calcul de u_n $u_0 = 2$ et $u_{n+1} = 0{,}5\,u_n + 3$

Algorithme

Saisir n
$u \leftarrow 2$
Pour k allant de 1 à n
  $u \leftarrow 0{,}5\,u + 3$
Fin Pour
Afficher u

Python

```
def terme(n):
    u = 2
    for k in range(n):
        u = 0.5*u + 3
    return u
```

TI-82

```
:Prompt N
:2→U
:Pour(K,1,N)
:0.5U+3→U
:Fin
:Disp U
```

Tableau d'exécution pour $n = 3$:

tour	avant	1	2	3
u	2	4	5	5,5

Le programme renvoie `5.5` : c'est u_3.
À la main : $u_1 = 4$, $u_2 = 5$, $u_3 = 5{,}5$.
Et `terme(10)` renvoie `5.99609375`.

Pack 2 – la relation dépend de n $u_0 = 1$ et $u_{n+1} = u_n + 2n + 1$

Algorithme

Saisir n
$u \leftarrow 1$
Pour k allant de 0 à $n-1$
  $u \leftarrow u + 2k + 1$
Fin Pour
Afficher u

Python

```
def terme2(n):
    u = 1
    for k in range(n):
        u = u + 2*k + 1
    return u
```

TI-82

```
:Prompt N
:1→U
:Pour(K,0,N-1)
:U+2K+1→U
:Fin
:Disp U
```

Au tour où le compteur vaut k, la variable contient u_k et on calcule $u_{k+1} = u_k + 2k + 1$: **le compteur joue le rôle de** n, il doit donc aller de 0 à $n-1$. Termes obtenus : 1, 2, 5, 10, 17, 26 $\Rightarrow$ on conjecture $u_n = n^2 + 1$ (à démontrer par récurrence).

Sur la TI-82 sans programme : le mode Suite et la table

1. `mode` : choisir **SUITE** (au lieu de FONCTION), `entrer`.
2. `f(x)` : remplir `nMin=0`, `u(n)=0.5u(n-1)+3`, `u(nMin)=2`.
   La lettre `u` : `2nde` `7` ; la lettre `n` : `X,T,θ,n`.
3. `2nde` `fenêtre` (déf table) : `DébTable=0`, `PasTable=1`.
4. `2nde` `graphe` (table) : on lit $u_0, u_1, u_2, \ldots$ en faisant défiler.

n	u
0	2
1	4
2	5
3	5,5
4	5,75

☞ Piège de la TI : elle calcule $u(n)$ à partir de $u(n-1)$

La relation $u_{n+1} = 0{,}5\,u_n + 3$ se tape `u(n)=0.5u(n-1)+3`. Quand la relation contient n, on remplace n par $n-1$: $u_{n+1} = u_n + 2n + 1$ se tape `u(n)=u(n-1)+2(n-1)+1`.

Les listes en Python : trois façons de les créer (exigées par le programme)

- **En extension :** `L = [1, 4, 9, 16]`.
- **Par ajouts successifs :** `L = []`, puis `L.append(x)` dans une boucle (pack 3).
- **En compréhension :** `[3*2**k for k in range(6)]` donne `[3, 6, 12, 24, 48, 96]` : idéal pour une suite **explicite** ($v_k = 3 \times 2^k$). Pour un seul terme d'une suite explicite, pas de boucle : `return 3*2**n` (TI : `3*2^N`).

Pack 3 – la liste des premiers termes $[u_0, u_1, \ldots, u_n]$ (toujours $u_0 = 2$, $u_{n+1} = 0{,}5\,u_n + 3$)

Algorithme

Saisir n
$L \leftarrow [2]$
Pour k allant de 1 à n
  ajouter à L le nombre
  $0{,}5 \times$ (dernier élément) $+ 3$
Fin Pour
Afficher L

Python

```
def liste_termes(n):
    L = [2]
    for k in range(n):
        L.append(0.5*L[-1] + 3)
    return L
```

TI-82

```
:Prompt N
:{2}→L1
:Pour(K,1,N)
:0.5L1(K)+3→L1(K+1)
:Fin
:Disp L1
```

L1 : | 2nde | 1 |

`liste_termes(4)` renvoie `[2, 4.0, 5.0, 5.5, 5.75]` : $n+1$ **termes**, de u_0 à u_n.

À retenir. En Python, `L[0]` est le **premier** élément (donc u_0), `L[-1]` le **dernier**, `len(L)` le nombre d'éléments, `L.append(x)` ajoute `x` à la fin. Sur la TI, les listes commencent à l'indice 1 : `L1(1)` contient u_0.

Pack 4 – deux suites qui dépendent l'une de l'autre

On pose $u_0 = 1$, $v_0 = 12$ et, pour tout n, $u_{n+1} = \dfrac{u_n + 2v_n}{3}$ et $v_{n+1} = \dfrac{u_n + 3v_n}{4}$.

Algorithme

Saisir n
$u \leftarrow 1$; $v \leftarrow 12$
Pour k allant de 1 à n
  $w \leftarrow u$ (on garde u_k)
  $u \leftarrow (u + 2v)/3$
  $v \leftarrow (w + 3v)/4$
Fin Pour
Afficher u et v

Python

```
def deux_suites(n):
    u, v = 1, 12
    for k in range(n):
        u, v = ((u + 2*v)/3,
                (u + 3*v)/4)
    return u, v
```

TI-82

```
:Prompt N
:1→U
:12→V
:Pour(K,1,N)
:U→W
:(U+2V)/3→U
:(W+3V)/4→V
:Fin
:Disp U,V
```

☞ Le piège de l'écrasement

Si l'on écrit `u = (u+2*v)/3` **puis** `v = (u+3*v)/4`, le calcul de v utilise le **nouveau** u : on obtient $v_1 \approx 11{,}08$ au lieu de $v_1 = \frac{37}{4} = 9{,}25$. Il faut sauvegarder u_k dans une variable auxiliaire w – ou, en Python, utiliser l'**affectation simultanée** `u, v = …, …`

Résultats :

n	0	1	2	3
u_n	1	8,333	8,944	8,995
v_n	12	9,25	9,021	9,002

les deux suites semblent converger vers la même limite 9.

À vous de jouer (algorithme + Python + TI-82 à chaque fois)

1. Soit $u_0 = 5$ et $u_{n+1} = 3u_n - 4$. Écrire un programme qui renvoie u_n, puis vérifier avec le mode Suite de la TI.
2. Soit $w_0 = 3$ et $w_{n+1} = 2w_n - n$. Écrire la relation à taper dans le mode Suite de la TI, puis un programme renvoyant la liste $[w_0, \ldots, w_n]$.
3. Écrire en une ligne la liste des 11 premiers termes de la suite $t_n = n^2 + 1$.
4. Soit $a_0 = 1$ et $a_{n+1} = \dfrac{a_n}{n+1}$. Écrire le programme (algorithme, Python, TI) qui renvoie a_n. Calculer a_5 ; que reconnaît-on ?

Réponses : 1. $u_1 = 11$, $u_2 = 29$, $u_3 = 83$; TI : `u(n)=3u(n-1)-4, u(nMin)=5`. **2.** `u(n)=2u(n-1)-(n-1)` ; termes 3, 6, 11, 20, 37 ; en Python, `L.append(2*L[-1] - k)` pour `k in range(n)`. **3.** `[n**2 + 1 for n in range(11)]`. **4.** Dans la boucle `for k in range(n):` écrire `a = a/(k+1)` (TI : `Pour(K,0,N-1)` puis `A/(K+1)→A`) ; $a_5 = \frac{1}{120} \approx 0{,}00833$: on reconnaît $a_n = \frac{1}{n!}$.