Algorithmique – Master MEEF, préparation CAPES Maths

Exercices Python – 23 Novembre 2020

Introduction

Ce document est un "notebook" Jupyter (https://jupyter.org). Il permet d'exécuter des bouts de code `Python` sans avoir besoin d'installer `Python` sur sa machine.

Vous avez donc deux possibilités pour faire ce TP. Vous pouvez installer l'une des nombreuses applications permettant de saisir et d'exécuter des programmes `Python`, comme par exemple :

- Thonny (https://thonny.org) ou
- Pyzo (https://pyzo.org), qui est le logiciel disponible au CAPES de Mathématiques.

Il est conseillé d'utiliser Pyzo (https://pyzo.org), pour se familiariser avec. Mais aujourd'hui, pour vous éviter d'installer quoi que ce soit sur votre machine, vous pouvez utiliser directement ce document dans votre navigateur web.

Ce document est composé de **cellules** qui peuvent contenir du **texte** comme celle que vous êtes en train de lire, ou du **code Python**. Les cellules pouvant contenir du code `python` se repèrent car elles sont marquées `In [ ]:` ou `Entrée [ ]:` à leur gauche.

Cliquer sur une cellule permet de la sélectionner. On repère la cellule sélectionnée par une barre à gauche (par défaut, bleue pour les cellules de texte et verte pour les cellules de code `python`).

Vous pouvez modifier toute cellule de code, en supprimant, changeant ou ajoutant du texte. Pour cela, il suffit de sélectionner la cellule. On peut ensuite l'exécuter directement dans ce document. On peut exécuter une cellule autant de fois que l'on souhaite.

Par exemple, la cellule suivante contient du code permettant juste d'afficher un message.
Pour l'exécuter : placez le curseur dans la cellule, et tapez `Shift-Entrée` (`Shift` est la touche pour passer temporairement en majuscules, et `Entrée` la touche pour passer à la ligne suivante).

Entrée [ ]:

```
1 print("Bonjour !\nÉvaluez moi en tapant Shift-Entrée")
```

En résumé : pour exécuter une cellule contenant du code `Python`, cliquez dans la cellule pour faire passer sa couleur en vert (elle devient alors éditable), modifiez-la, puis :

- soit utilisez le bouton `Run` (ou `Exécuter`) de la barre d'outils en haut de cette page web,
- soit tapez `Shift-Entrée` (`Shift` est la touche de passage temporaire en majuscules).

Ce raccourci clavier exécute la cellule, et positionne le curseur dans la cellule suivante. Vous pouvez utiliser ce même raccourci sur une cellule de texte pour passer à la cellule suivante (également si vous avez double-cliqué dans une cellule texte par erreur, pour revenir à l'affichage normal).

Attention ! Sauvegardez régulièrement votre travail (menu `File -> Download as -> Python` en anglais, ou `Fichier -> Télécharger au format -> Python` en français, qui enregistre tous les blocs `Python` que vous aurez saisis).

Algorithmes itératifs

Exercice 1

1. Écrire une fonction python non récursive `somme_puissances` qui prend en argument un entier n et un entier p, et qui renvoie la valeur $\sum_{k=0}^{n} k^p$ si $n \geqslant 0$, et 0 si $n < 0$.

Entrée [ ]:

```
def somme_puissances(n, p):
    pass

# Un test
for p in range(5):
    print(somme_puissances(5, p))
```

2. Écrire une version récursive de cette fonction.

Entrée [ ]:

```
def somme_puissances_rec(n, p):
    pass

# Un test
for p in range(5):
    print(somme_puissances_rec(5, p))
```

3. En utilisant l'une des fonctions précédentes, écrire :
   - une fonction `somme_entiers` qui prend en argument un entier n et qui renvoie $\sum_{k=0}^{n} k$ si $n \geqslant 0$, et 0 si $n < 0$.
   - une fonction `somme_cubes` qui prend en argument un entier n et qui renvoie $\sum_{k=0}^{n} k^3$ si $n \geqslant 0$, et 0 si $n < 0$.

Entrée [ ]:

```
def somme_entiers(n):
    pass

def somme_cubes(n):
    pass

# Tests
print(somme_entiers(5))
print(somme_cubes(5))
```

4. Écrire une fonction `verifier_identite` qui prend en argument un entier N et qui vérifie que pour tout entier n entre 0 et N, on a

$$\sum_{k=0}^{n} k^3 = \left(\sum_{k=0}^{n} k\right)^2.$$

Entrée [ ]:

```
def verifier_identite(N):
    pass

# Tests
print(verifier_identite(1000))
```

Exercice 2

On représente un point P dans le plan par un couple (x, y), où x est l'abscisse de P et où y est son ordonnée. En Python, si `P` est le couple `(x,y)`, on accède à la valeur de `x` par `P[0]` et à celle de `y` par `P[1]`.

1. Écrire une fonction `distance(P1, P2)` qui prend en arguments deux points P_1 et P_2 et qui retourne la distance entre ces points. Pour utiliser la fonction *racine carrée*, insérer la ligne `from math import *` avant l'utilisation. La fonction racine carrée est alors accessible par `sqrt`.

Entrée [ ]:

```
from math import *

def distance(P1, P2):
    pass

# Tests
print(distance((1,1), (1,1)))  # résultat attendu : 0.0
print(distance((0,1), (1,1)))  # résultat attendu : 1.0
print(distance((0,0), (1,1)))  # résultat attendu : 1.4142135623730951
```

Listes

On veut maintenant manipuler une **liste** de points. Pour cela, il faut comprendre que Python peut manipuler des variables représentant non pas une seule valeur, mais un nombre fini arbitraire de valeurs (qui peuvent être de même type, ou pas). Comme c'est une notion utile, voici les éléments principaux à connaître sur les listes, avant de passer à la question suivante. L'intérêt de manipuler des listes est de pouvoir,

- passer une liste de taille quelconque en argument à une fonction (et donc, simuler une fonction qui prend un nombre arbitraire d'arguments),
- retourner des listes de taille quelconque (et donc, simuler une fonction qui retourne un nombre arbitraire d'arguments),
- modifier les listes.

Création de listes. On peut donner explicitement une liste en insérant ses éléments entre crochets, séparés par des virgules. Il y a d'autres façons de créer des listes. Par exemple, on peut former les listes suivantes :

Entrée [ ]:

```
L1 = [2, 3, 5, 7, 11, 13]  # Liste des premiers nombres premiers, donnés explici
print(L1)

L2 = list(range(20))       # Liste L[i] = i pour i de 0 à 19, en utilisant la co
print(L2)

L3 = [x*x*x for x in L1]   # Liste des cubes des éléments de L1.
print(L3)

L4 = [(0,0), (0,1), (1,0)] # Liste de 3 couples d'entiers, donnés explicitement.
print(L4)

L5 = [1, 1.0, "Un", True]  # Liste d'éléments de types divers, donnés explicitem
print(L5)
```

Fonction longueur. La fonction `len` renvoie la longueur d'une liste :

Entrée [ ]:

```
for l in [L1, L2, L3, L4, L5]:
    print(len(l))
```

Parcours d'une liste par l'instruction `for` . Observez dans l'exemple précédent que la notation

```
for <nom_de_variable> in <liste>:
   instructions
```

permet d'exécuter les `instructions` en faisant prendre à la `variable` , successivement, chaque valeur des éléments de la `liste` .

Accès à un élément particulier. On peut accéder à l'élément d'indice `i` dans une liste `L` par `L[i]` . La numérotation, comme c'est l'usage en mathématiques pour les suites, commence à 0. Bien sûr, l'indice `i` doit être légal, donc pas supérieur ou égal à la longueur de la listes.

Entrée [ ]:

```
print(L5[2])
for p in L1:
    print(p, "est un nombre premier")

L5[2] = "One"
print("Nouvelle valeur de L5 :", L5)
```

Notez que l'on peut utiliser des indices négatifs : l'indice -1 désigne le dernier élément, l'indice -2 l'avant-dernier, etc.

Entrée [ ]:

```
print(L5[-1])
```

Tranches d'éléments successifs. On peut extraire d'une liste une sous-liste composée d'éléments situés à des indices consécutifs :

Entrée [ ]:

```
print(L1[1:4])  # éléments de L1 entre l'indice 1 et l'indice 3.
print(L1[2:])   # éléments de L1 de l'indice 2 jusqu'à la fin.
print(L1[:-3])  # éléments de L1 sauf les 3 derniers (c'est-à-dire jusqu'à l'inc
```

Concaténation de listes. On peut enfin concaténer des listes par l'opérateur `+` .

Entrée [ ]:

```
L1 = L1 + [17, 19]
print("Nouvelle valeur de L1 :", L1)
```

2. Écrire une fonction `distance_minimale(L)` prenant en argument une liste `L` de points du plan et qui retourne `None` si cette liste a moins de deux éléments, et deux points P et Q de la liste situés à des positions distinctes dans `L` et dont la distance est minimale parmi toutes les distances entre points situés à des positions distinctes de `L` .

   Rappels. On accède à la longueur d'une liste `L` par `len(L)` . Connaissant une position `i` dans la liste, entre 0 et `len(L)-1` , on accède à l'élément à cette position par `L[i]` . Enfin, on peut parcourir toutes les positions grâce à une boucle `for i in range(len(L))` .

Entrée [ ]:

```
def distance_minimale(L):
    pass

# Tests
L = [(x, x**2) for x in range(1, 10)]
print(L)
print(distance_minimale(L))

L = L + [(6, 26)]
print(L)
print(distance_minimale(L))
```

3. Évaluer la complexité de votre fonction en fonction de la longueur de la liste.

Algorithmes récursifs

Exercice 3

La suite $(u_n)_{n\geqslant 0}$ est définie par les égalités suivantes:

$$u_0 = 1, \quad u_1 = 42 \quad \text{et pour } n \geqslant 0,\ u_{n+2} = u_{n+1} - 2u_n.$$

1. Écrivez une fonction récursive traduisant naïvement cette définition mathématique.

Entrée [ ]:

```
def u(n):
    pass
```

2. Évaluer le nombre d'appels récursifs de votre fonction.

3. En utilisant le fait que :

$$\begin{pmatrix} u_{n+2} \\ u_{n+1} \end{pmatrix} = M \cdot \begin{pmatrix} u_{n+1} \\ u_n \end{pmatrix}$$

pour une matrice M bien choisie, écrivez une fonction calculant u_n réalisant:

a. dans un premier temps, $O(n)$ opérations arithmétiques dans le cas le pire.

Entrée [ ]:

```
def u1(n):
    pass
```

b. dans un second temps, $(\log_2(n))$ opérations arithmétiques dans le cas le pire. **Indication** utilisez l'exponentiation rapide.

Entrée [ ]:

```
def u2(n):
    pass
```

4. Testez vos deux fonctions pour n valant 1, 10, 30, 40, 100, 100000000.

Entrée [ ]:

```

```

Module turtle

Dans les exercices 4, 5 et 6, on utilise le module `turtle` de python. Ce module permet de dessiner des segments de droites, en déplaçant un stylo (appelé "tortue") sur une fenêtre. Pour pouvoir utiliser ce module **sous Pyzo**, vous devez écrire:

```
from turtle import *
```

ou, ce qui est plus propre mais demande d'ajouter " `turtle.` " devant chaque nom de fonction:

```
import turtle
```

Si par contre, vous utilisez **ce notebook**, vous **devez** évaluer la cellule suivante.

Entrée [ ]:

```
from jturtle import *
```

L'exécution de la cellule précédente ne produit pas de résultat visible (à part que le compteur d'exécution des cellules, qui se trouve entre les crochets à gauche de la cellule, après le mot `Entrée` ou `In` , doit contenir maintenant le nombre de cellules évaluées, par exemple `[42]`). Mais une fois la cellule évaluée, vous avez accès aux instructions suivantes.
Rappel : tout ce qui se trouve **après un signe** `#` jusqu'à la fin de la ligne est **ignoré** par `Python` (sauf bien sûr si le signe `#` fait partie d'une chaîne de caractères). Cela permet d'insérer des commentaires en français dans les programmes.

Instruction Instruction	Signification Signification	Exemples d'utilisation Exemples d'utilisation
`Screen`	Ouvre un tableau graphique dont les dimensions (largeur et hauteur) peuvent être données en argument. Un seul tableau est visible à un instant : celui qui correspond à la dernière instruction `Screen` exécutée. Le tableau apparaît dans le navigateur à droite de ce texte. *Seulement sur cette page web* : ajustez la largeur et la hauteur si celle par défaut ne convient pas à la taille de votre écran.	`Screen()` `Screen(300, 750)  # jturtle seulement`
`forward`	Avance de la distance donnée en argument, dans la direction où la tortue regarde actuellement. Si la distance est négative, la tortue recule.	`forward(100)  # avance de 100 pixels` `avancer(-50)  # recule de 50 pixels`
`left`	Tourne de l'angle donné en argument. L'angle peut être positif ou négatif.	`left(45)  # 45 degrés vers la gauche` `left(-90) # 90 degrés vers la droite`
`setheading`	Positionne l'angle de la tortue (angle nul = vers l'Est).	`setheading(125)`
`reset`	Efface le tableau graphique et repositionne la tortue au centre.	`reset()`
`up`	Lève le crayon : les déplacements de la tortue ne produisent plus de marque jusqu'à ce que le crayon soit reposé.	`up()`
`down`	Pose le crayon : les déplacements de la tortue produisent une marque (sauf l'instruction `aller`), jusqu'à ce que le crayon soit levé.	`down()`

`pencolor`	Change la couleur du crayon. On doit donner 3 arguments, entre 0 et 255 :
	- la composante de rouge,
	- la composante de vert,
	- la composante de bleu.
	
	La couleur obtenue est un mélange des trois couleurs. 0 signifie l'absence de couleur et 255 la présence maximale.
	Sous Pyzo, préalablement : `colormode(255)`
		`# Passe en rouge`
		`pencolor(255, 0, 0)`
		`# Passe en vert`
		`pencolor(0, 255, 0)`
		`# Passe en bleu`
		`pencolor(0, 0, 255)`
		`# Passe en jaune`
		`pencolor(252, 258, 41)`
		`# Passe en noir`
		`pencolor(0, 0, 0)`

`speed`	(Pyzo seulement) : règle la vitesse	`speed('fastest')` `speed(1)`

Exercice préliminaire

Familiarisez-vous avec ces instructions. Pour cela, exécutez, modifiez et ré-exécutez autant de fois que nécessaire les cellules suivantes.

Entrée [ ]:

```
# Ajustez les dimensions pour avoir un tableau de taille convenable.
# Une fois créé, vous pourrez déplacer le tableau à la souris, à l'intérieur
# de la fenêtre de votre navigateur.
Screen(450,750)
```

Entrée [ ]:

```
# Modifiez éventuellement la distance
forward(150)
```

Entrée [ ]:

```
1 left(-60) # angle en degré, vers la droite si négatif
```

Entrée [ ]:

```
1 forward(50)
```

Entrée [ ]:

```
1 reset()
```

Entrée [ ]:

```
1 setheading(180)
```

Entrée [ ]:

```
1 up()
```

Entrée [ ]:

```
1 forward(100)
```

Entrée [ ]:

```
1 down()
```

Entrée [ ]:

```
1 forward(100)
```

Entrée [ ]:

```
1 left(120)
```

Entrée [ ]:

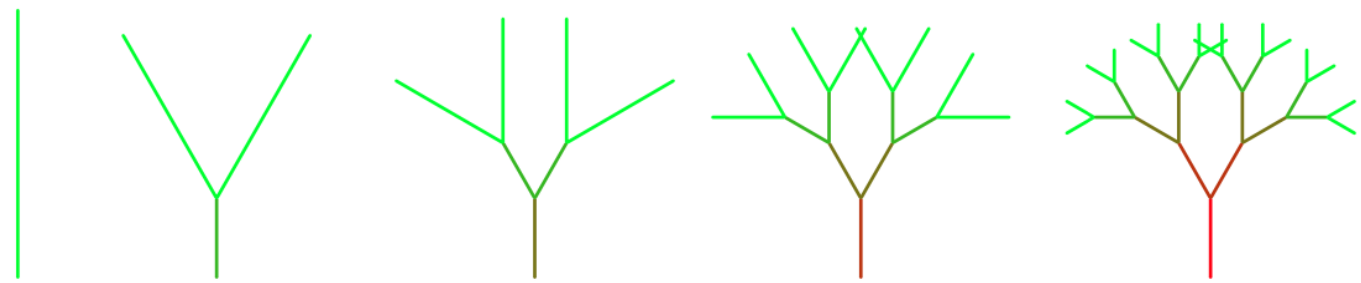
```
1 pencolor(250,50,200)
```

Entrée [ ]:

```
1 forward(100)
```

Exercice 4

On veut dessiner des arbres obtenus récursivement, de la façon suivante:



L'arbre de rang 0 (à gauche) est réduit à un segment vertical. Pour $n \geqslant 1$, l'arbre de rang n est composé d'un segment vertical et de deux arbres de rang $n-1$ commençant chacun en haut du tronc, et inclinés d'un angle donné (par exemple 30°) symétriquement par rapport au tronc.

1. Écrivez une fonction

   ```
   def arbre(n, longueur, angle, ratio):
   ```

   qui dessine l'arbre de rang n. La longueur (de la base du tronc aux extrémités des branches) est donnée par le second argument. Le troisième argument est l'angle des sous-arbres par rapport au tronc. Le dernier argument est le ratio entre la longueur du tronc et la longueur totale.

Entrée [ ]:

```
1
```

2. Combien d'appels récursifs sont-ils effectués par cette fonction?

Exercice 5

Les flocons de Koch sont obtenus en répétant récursivement un motif. Les premiers flocons sont les suivants:

Le flocon de rang 0 (figure de gauche) est simplement un segment. Pour $n \geqslant 1$, le flocon de rang n est obtenu en juxtaposant 4 flocons de rang $n-1$: le premier horizontal, le second incliné de 60°, le troisième incliné de -60° et le quatrième également horizontal.

1. Écrivez une fonction `koch(n, longueur)` qui dessine le flocon de rang n, dont la longueur de l'extrémité gauche à l'extrémité droite est donnée par le second paramètre.

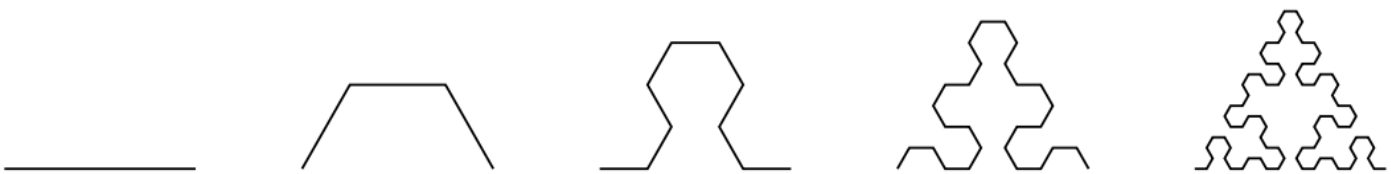
Entrée [ ]:

```
1
```

2. Combien d'appels récursifs sont-ils effectués par cette fonction?

Exercice 6

Même exercice pour le triangle de Sierpinski dont les premiers rangs sont représentés sur la figure suivante.



Entrée [ ]:

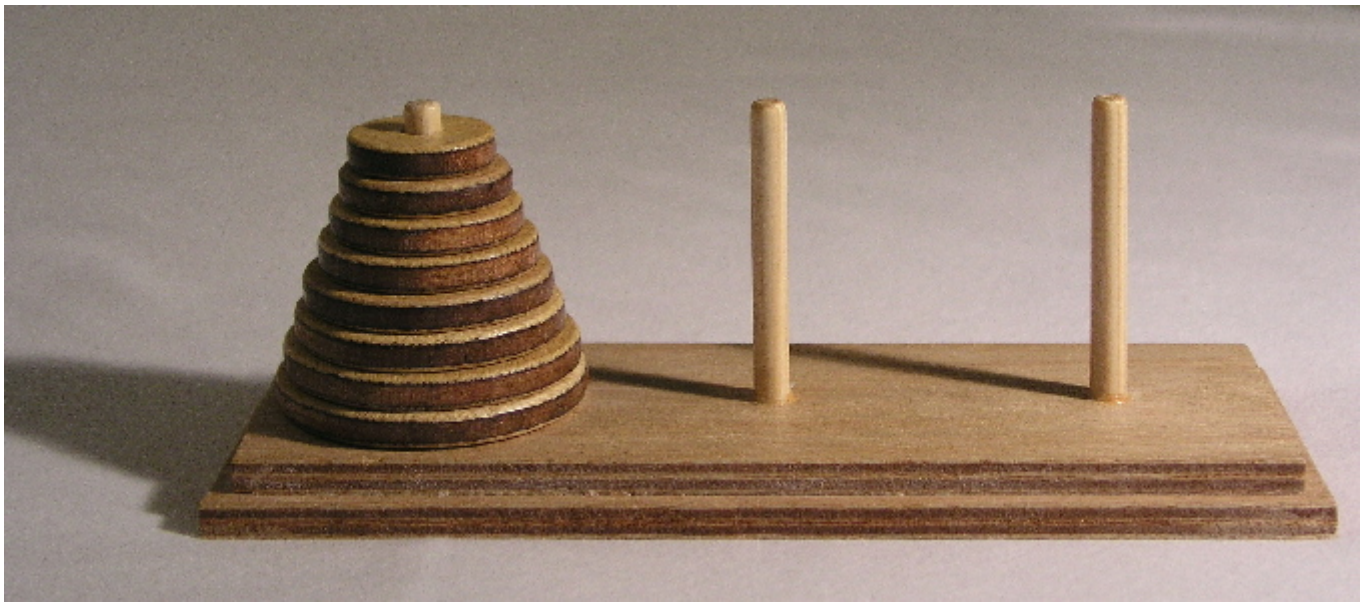
```
1
```

Testez la fonction pour l'itération 8.

Exercice 7

Le jeu des tours de Hanoi se joue avec 3 piquets, appelés d (comme départ), a (comme arrivée), et i (comme intermédiaire), et n disques de tailles différentes que l'on met sur ces piquets.

On part de la configuration où les n disques sont disposés sur le piquet d de départ, comme sur la photo suivante (avec $n = 8$).



L'objectif est de déplacer les disques du piquet d vers le piquet a, en déplaçant un seul disque à chaque étape. La contrainte à respecter est qu'on n'a pas le droit de placer un disque sur un disque plus petit que lui.

1. En supposant que vous savez déplacer les $(n-1)$ disques les plus petits d'un piquet vers un autre en utilisant une suite de mouvements, comment faire pour déplacer les n disques du piquet d vers le piquet a?
2. Décrire un algorithme récursif permettant de résoudre le jeu, en affichant la suite des déplacements de disques.
3. Écrire cet algorithme en `Python`, en faisant une fonction récursive `hanoi` qui prend en paramètre le nombre n de disques et affiche la suite des déplacements de disques à effectuer. Un déplacement du disque 1 vers le disque 3 peut par exemple être affiché, par l'instruction `print`, comme suit :

```
print("1 -> 3")
```

Entrée [ ]:

```
1
```

4. Combien cet algorithme fait-il de déplacements de disques :? Justifiez votre réponse.

Exercice bonus : sommes de trois cubes

On considère un entier k entre 0 et 99, et l'équation :

$$x^3 + y^3 + z^3 = k.$$

On veut savoir pour quelles valeurs de k cette équation à une solution pour x, y, z entiers (positifs ou négatifs).

1. Écrire une fonction `generate` qui prend en paramètre un entier n, et génère la **liste de tous les entiers** k pour lesquels l'équation ci-dessus à une solution avec $x, y, z \in \mathbb{Z}$ et $|x| \leqslant n$, $|y| \leqslant n$ et $z \leqslant n$.

Entrée [ ]:

```
def generate(n):
    pass
```

2. Testez votre fonction pour $n = 100$, $n = 200$, $n = 500$. Vous devriez obtenir que l'équation a une solution pour les entiers k suivants :
$0, 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28, 29, 34, 35, 36, 37, 38, 43, 44, 48, 53, 54, 55, 56, 57, 60, 61, 62, 63, 64, 65, 66, 69, 70, 71, 72, 73, 78, 79, 80, 81, 82, 83, 88, 89, 90, 91, 97, 98, 99.$

Entrée [ ]:

```
print(generate(100))
print(generate(200))
print(generate(800)) # soyez patient(e)
```

Si vous avez optimisé le programme et que vous êtes assez patient(e) (entre 5 et 6 minutes pour un programme intelligemment optimisé, beaucoup plus sinon), vous détecterez que la valeur $k = 51$ fournit aussi une solution :

$$602^3 + 659^3 - 796^3 = 51.$$

Mais il manque de nombreuses valeurs dans la liste ci-dessus. Les chercheurs ont en fait montré que l'équation a une solution pour **toute** valeur de k entre 0 et 99, mais

- on ne connaît pas d'algorithme prenant un entier k en entrée et indiquant si l'équation $x^3 + y^3 + z^3 = k$ a une solution.
- certaines valeurs de k demandent des efforts mathématiques, algorithmiques et de programmation conséquents. C'est le cas pour les valeurs 33 et 42, dernières à avoir résisté, et dont les plus petites solutions sont gigantesques :

Entrée [ ]:

```
x = 8866128975287528
y = -8778405442862239
z = -2736111468807040

print(x**3 + y**3 + z**3)

x = -80538738812075974
y = 80435758145817515
z = 12602123297335631

print(x**3 + y**3 + z**3)
```