Pyramides [it10] - Exercice résolu

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 17 mai 2018

Table des matières

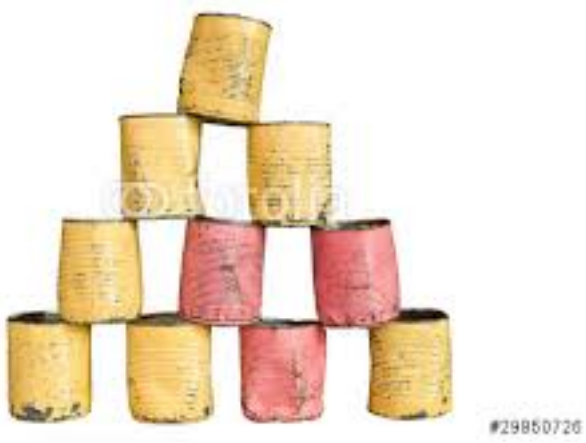
Python - Pyramides (Solution)

Mots-Clés Schéma itératif ■
Requis Structures de base, Structures répétitives ■
Difficulté • ○ ○

Objectif
Cet exercice calcule et affiche le nombre de boîtes à poser sur la rangée de base pour construire la plus haute pyramide complète possible étant donné le nombre de boîtes disponibles.

1 Énoncé

Cet exercice concerne des murs de boîtes de conserves construits sous forme de pyramide (image : http://google/images) :



Une première rangée est posée par terre. La rangée suivante est construite de manière à ce que chaque boîte soit posée entre deux boîtes du dessous, si bien qu'elle contient une boîte de moins que la rangée du dessous. La hauteur de la pyramide se mesure en nombres de rangées.

Problème 1
Écrire un algorithme, qui **sans utiliser** la formule de la somme d'une suite arithmétique, calcule et affiche le nombre de boîtes de conserves nécessaires pour former une pyramide dont la hauteur est demandée à l'utilisateur. Par exemple, il faut 10 boîtes pour une pyramide de hauteur 4 et 2 boîtes pour une pyramide de hauteur 6.

Problème 2
Écrire un algorithme qui demande à l'utilisateur de combien de boîtes il dispose et calcule et affiche combien de boîtes il faut poser sur la rangée du dessous pour construire la plus grande pyramide complète possible avec ses boîtes. L'algorithme affichera aussi le nombre de boîtes non utilisées. Par exemple, avec 18 boîtes il faut poser 5 boîtes sur la base et il reste 3 boîtes.

...(suite page suivante)...

2 Algorithmique, Programmation

2.1 Problème 1 : Nombre de boites

Choix des variables

Une première lecture de l'énoncé conduit à déclarer deux variables entières : le nombre total de boîtes (`nboites`) et le nombre de rangées (`hauteur`).

Modèle de l'algorithme

Le premier réflexe est de se représenter la construction de la pyramide comme la réalité, en commençant par le bas. Puisqu'on enlève une boîte à chaque rangée, le nombre de boîtes posées sur la base est égal à la hauteur de la pyramide. Le nombre de répétitions est donc égal à la hauteur : il est donc connu et conduit vers l'utilisation d'une boucle `Pour`.

Construction de la pyramide

Chaque événement de la construction consiste à ajouter une rangée d'un nombre de boîtes inférieur d'une unité au nombre de boîtes de la rangée du dessous. Le corps de la répétitive contiendra donc, dans un ordre convenable, le calcul du nombre de boîtes de la nouvelle rangée à partir de la précédente et le cumul du nombre de boîtes de la nouvelle rangée.

Initialisations

Il y a deux manières d'initialiser qui conduisent à deux variantes d'algorithmes :

- Soit on part d'une pyramide vide.
- Soit on part de la base de la pyramide.

Dans les deux cas, la boucle pose la rangée (cumule les boîtes) avant de préparer la rangée suivante. Cependant la boucle est croissante dans le premier cas, tandis qu'elle est décroissante dans le deuxième.

Écrivez une procédure qui traduit cette analyse en partant d'une pyramide vide.

Validez votre procédure avec la solution.

Solution Python @[pgpyramide.py]

```
def afficherPyramide1a(hauteur):
    """  Problème 1

     :param hauteur: hauteur de la pyramide
    """
    nboites = 0
    for rangee in range(1, hauteur+1):
        nboites += rangee
        print(rangee, "e rangee Total = ", nboites, sep="")
    print()
```

```
def afficherPyramide1b(hauteur):
    """  Problème 1

     :param hauteur: hauteur de la pyramide
    """
    nboites = 0
    for rangee in range(hauteur, 1-1, -1):
        nboites += rangee
        print(rangee, "e rangee Total = ", nboites, sep="")
    print()
```

Solution simple

Les valeurs du compteur de boucle et de la variable `rangee` sont identiques lors d'un même passage dans la boucle. On peut donc en éliminer une et remplacer la variable de boucle par `rangee`.

2.2 Problème 2 : Nombre de boites sur la base

Choix des variables

Reprenons l'idée du premier problème qui consiste à commencer la construction par le sommet et à ajouter, tant qu'on peut le faire, une rangée avec une boîte de plus.

A chaque rangée construite, il faut puiser dans les boîtes disponibles. Nous sommes donc amenés à redéfinir le rôle de la variable `nboites` qui représentera le nombre de boîtes **encore disponibles** et non celui des boîtes déjà placées dans la pyramide.

La variable `rangee` représentera le nombre de boîtes qui composent la rangée en construction.

Modèle de l'algorithme

Il est clair que cette fois nous ne connaissons pas le nombre de répétitions puisque si nous connaissions la hauteur, nous connaîtrions en même temps le nombre de boîtes de la base.

Pour que l'algorithme puisse aussi s'appliquer au cas limite où il y a 0 boîte, il est préférable d'utiliser une répétitive `TantQue`. L'algorithme se compose en suivant l'action de construction virtuelle de la pyramide à partir du sommet :

```
TantQue assez de boites pour la prochaine rangee Faire
  construire la nouvelle rangee avec une boite de plus
  mettre à jour le nombre de boites restantes
FinTantQue
```

La **prochaine** rangée est égale à la rangée précédente augmentée de 1, ce qui explique la condition d'entrée dans la boucle.

Initialisations

Le nombre de boîtes disponibles sera saisi par l'utilisateur et le nombre de boîtes sur la rangée initialisé à 0 (pour couvrir le cas limite d'une pyramide vide).

Écrivez une procédure qui traduit cette analyse.

Validez votre procédure avec la solution.

Solution Python @[pgpyramide.py]

```
def afficherPyramide2a(nboites):
    """  Problème 2

     :param nboites: nombre de boites
    """
    rangee = 0
    while nboites >= rangee + 1:
        rangee += 1
        nboites -= rangee
        print(rangee, "e rangee Reste = ", nboites, sep="")
    print("Il faut ", rangee, " rangee(s) sur la base", sep="")
    print("Il reste ", nboites, " boite(s)", sep="")
    print()
```

```
def afficherPyramide2b(nboites):
    """  Problème 2

     :param nboites: nombre de boites
    """
    rangee = 1
    while nboites >= rangee:
        nboites -= rangee
        print(rangee, "e rangee Reste = ", nboites, sep="")
        rangee += 1
    rangee -= 1
    print("Il faut ", rangee, " rangee(s) sur la base", sep="")
    print("Il reste ", nboites, " boite(s)", sep="")
    print()
```

Solution commentée

L'expression booléenne du `TantQue` correspond à l'acte d'évaluer s'il y a assez de boîtes pour construire la nouvelle rangée. On pourrait être tenté de mettre à jour la nouvelle rangée avant d'entrer dans la boucle ce qui entraînerait une inversion des deux instructions du corps de boucle. Tout se passe bien tant qu'on a assez de boîtes. Mais lorsque la rangée devient strictement supérieure au nombre de boîtes restantes, la boucle va s'arrêter et la rangée gardera sa valeur avec une boîte de trop. Il faudra donc faire « marche arrière » en ajoutant une instruction qui remet la rangée à sa dernière valeur acceptable. C'est ce qui est fait dans la deuxième méthode.

Testez.

Validez votre script avec la solution.

Solution Python @[pgpyramide.py]

```
def PGPyramide():
    h = int(input("Combien de rangees? "))
    afficherPyramide1a(h)
    afficherPyramide1b(h)

    n = int(input("Combien de boites? "))
    afficherPyramide2a(n)
    afficherPyramide2b(n)
```

3 Que retenir de cet exercice?

Il met en évidence deux niveaux d'analyse :

- Quelle boucle choisir.
- Comment organiser le corps de la boucle (corps des instructions).

Le premier problème montre deux versions de la construction qui induisent les deux variantes de la boucle `Pour`. Dans le cas de la boucle décroissante, il faut vérifier l'incrément et le test d'arrêt.

Le deuxième problème montre comment un problème peut être résolu en simulant des événements pour trouver quelle est la situation favorable.

4 Références générales

Comprend [Tartier-AL1 :c5 :ex16] ■