Institut National Universitaire Champollion

Chapitre 4

Boucles imbriquées
Listes de listes

Plan du chapitre

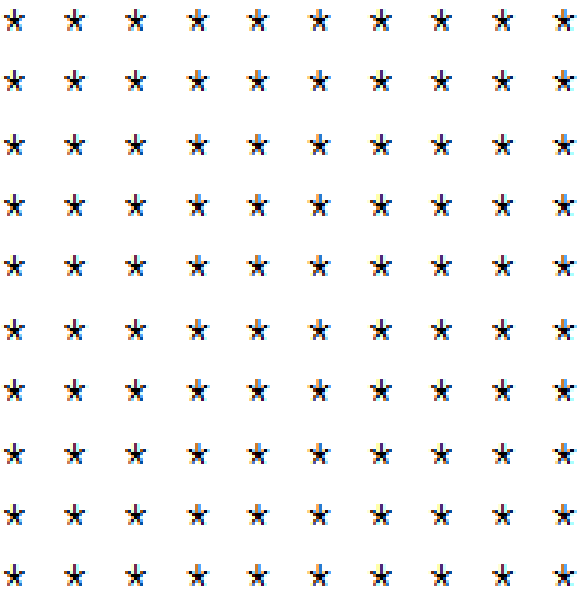
Boucles imbriquées

Boucles imbriquées

On a parfois le besoin de mettre une boucle dans une autre boucle.

Exemple :
écrire un algorithme affichant un carré de n lignes, chaque ligne contenant n étoiles.

Résultat attendu pour n = 10 :



Boucles imbriquées

idée :

on a besoin d'une boucle dite **externe** dont le corps permettra d'afficher une ligne, puis de passer à la ligne :

```
for i in range(n)
        afficher une ligne
        passer à la ligne
```

afficher une ligne boucle dite **interne** :

```
for j in range(n) :
        afficher une étoile
```

Boucles imbriquées

D’où le code :

```
1  n = 10
2  for i in range(n):
3      for j in range(n):
4          print("* ", end = "")
5      print("") # on passe à la ligne
6
```

Boucles imbriquées

Une borne de la boucle interne peut dépendre de l'indice de la boucle externe :

écrivons un programme affichant un triangle inférieur d'étoiles de n lignes.

Pour n = 10 :

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```

Boucles imbriquées

Même idée que précédemment sauf que cette fois la boucle interne doit afficher i étoiles :

afficher une ligne :

for j in range(i) :

afficher une étoile

d'où la solution :

```
1  n = 10
2  for i in range(n):
3      for j in range(i):
4          print("* ", end = "")
5      print("") # on passe à la ligne
6
```

Boucles imbriquées

Autre exemple :

On se donne deux listes L1 et L2.
On veut savoir si tous les éléments de L1 sont dans L2 (sans utiliser le in ...)

Idée : on prend chaque élément de L1 (boucle externe),
et on regarde si il est dans L2, en parcourant la liste L2. (boucle interne).
On arrête les parcours de listes dès qu'on repère un élément de L1 non présent dan L2 : utilisation boucle while …

ok = vrai (bouléen indiquant que L1 inclus dans L2)
tant que i est un indice de L1 et ok vrai : (boucle externe)
    si L1[i] n'est pas dans L2 alors
    tous les éléments de L1 ne sont pas dans L2
    donc ok = faux (boucle interne)

Boucles imbriquées

En pseudo code :

```
ok = vrai  (bouléen indiquant que L1 inclus dans L2)
i = 0
tant que < longueur(L1) et ok vrai :   (boucle externe)
      si L1[i] n'est pas dans L2 alors
            tous les éléments de L1 ne sont pas dans L2
            donc ok = faux (boucle interne)
       sinon i = i+1
```

savoir si L1[i] est dans L2 ou pas, boucle interne :

```
j = 0
présent = Faux
tant que j < longueur(L2) et pressent = Faux
      si L1[i] == L2[j]
            present = Vrai
      sinon
            j = j+1
si présent == Faux ok = faux
```

Boucles imbriquées

Entrée [14]:

```
1  L1 = [1,2,3]
2  L2 = [3,6,2,8,1]
3  ok = True
4  i = 0
5  while(i<len(L1) and ok):
6      j = 0
7      present = False
8      while j<len(L2) and not present:
9          if (L1[i] == L2[j]):
10             present = True
11         else:
12             j = j+1
13     if not present:
14         ok = False
15     i = i+1
16 print("L1 dans L2",ok)
```

```
L1 dans L2 True
```

attention au cas ou L[i] est en dernière position dans L2 …..

Listes de listes

Listes de listes

On a vu dans le chapitre précédent que les éléments d'une liste pouvaient être de type différent.

Ils peuvent même être une autre liste !!!

c'est pratique : on a peut obtenir ainsi des « tableaux » de dimension 2 …

exemple :

Entrée [15]:

```
1 L = [[1,2],[3],[6,7,8]]
2 print(L)
```

```
[[1, 2], [3], [6, 7, 8]]
```

Listes de listes

On peut accéder à la iéme ligne :

Entrée [17]:
```
1 L = [[1,2],[3],[6,7,8]]
2 print(L[2])
```

```
[6, 7, 8]
```

et aussi à l'élément situé en iémé ligne et j iéme colonne :

Entrée [18]:
```
1 L = [[1,2],[3],[6,7,8]]
2 print(L[2][1])
```

```
7
```

Listes de listes

Le plus souvent on utilise des tableaux ou toutes les lignes ont le même nombre d'éléments p, on obtient ainsi une « - matrice » à n lignes et p colonnes :

exemple avec une « matrice » de 4 lignes et 3 colonnes :

Entrée [19]:
```
1 L = [[1,6,5],[2,9,4],[5,1,3],[8,7,2]]
2 print(L)
3 print("nombre de lignes : ",len(L))
4 print("nombre de colonnes : ",len(L[0]))
```

```
[[1, 6, 5], [2, 9, 4], [5, 1, 3], [8, 7, 2]]
nombre de lignes :  4
nombre de colonnes :  3
```

le nombre de lignes est bien len(L),
le nombre de colonne est :

len(L[0]) = len(L[1]) = len(L[2]) = len(L[3])

Listes de listes

Affichage plus clair d'une liste de listes :

Entrée [20]:

```
1 L = [[1,6,5],[2,9,4],[5,1,3],[8,7,2]]
2 for i in range(len(L)):
3     print(L[i])
```

```
[1, 6, 5]
[2, 9, 4]
[5, 1, 3]
[8, 7, 2]
```

Exercices

Exercice 1 :
Un des tris (ordre croissant) les plus simples à mettre en oeuvre consiste à aller chercher la plus petite valeur présente et à la placer en première place. Ensuite, plus petite valeur qui reste est placée dans la seconde place et ainsi de suite.

Exercice 2 :
Ecrire un programme qui saisit n éléments et les range dans une liste. La liste doit être triée dans l'ordre croissant au fur et à mesure de la saisie.
On utilisera la fonction L.insert(i, elt) pour ajouter à la liste L, à l'indice i l'élément elt.