# Cours Introduction à la Programmation Python VI
# (IP1 Python)

Arnaud Sangnier
sangnier@irif.fr

Mercredi 22 Novembre 2017
MIASHS et MATHS

# Dans les épisodes précédents

- Présentation de certains aspects de Python 3
- Ce que nous avons vu :
  - Les données et leur type : int , str et bool
  - Les variables : affectation, lecture, modification
  - Toutes les instructions
  - Définition de fonctions
  - Les listes à une dimension et leur utilisation basique
  - La représentation en mémoire des listes

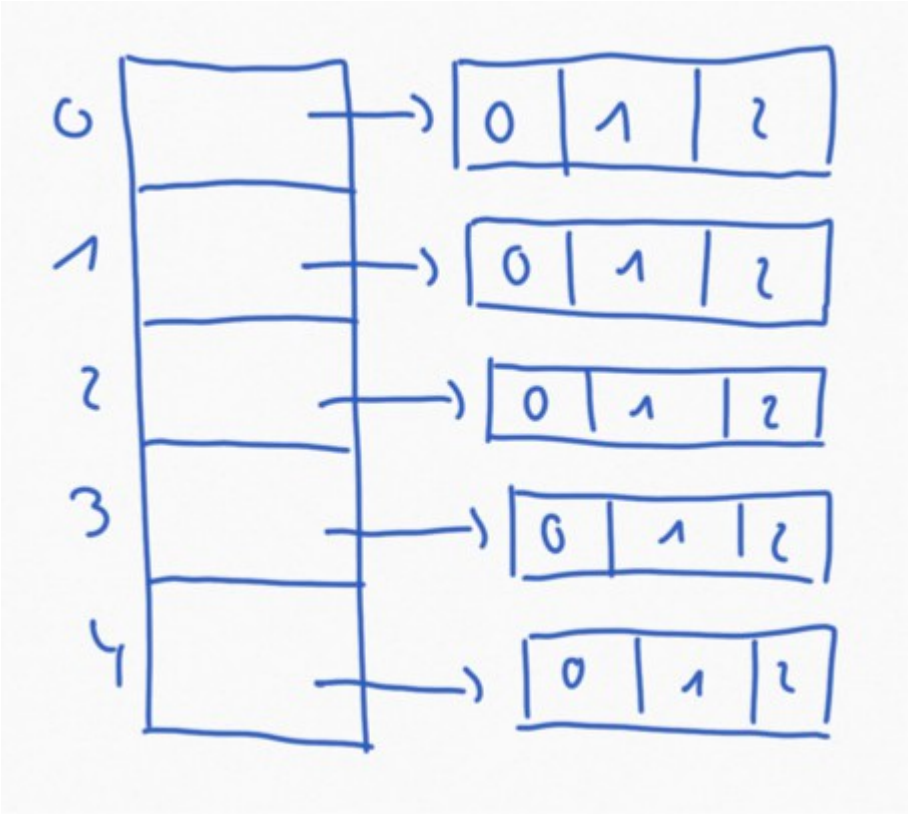
# Des listes à plusieurs dimensions

- Comme on a vu il est possible de créer des listes de listes d'entiers
- Même des listes de listes de listes d'entier
- etc
- Quelles sont les mécanismes utiles pour manipuler de telles listes
- Comment créer de telles listes ?
- Comment modifier de telles listes ?

# Un premier exemple

- Créer une liste de taille 5
- Chaque élément de la liste contient une liste de taille 3
- Chaque liste de taille 3 est de la forme {0,1,2}

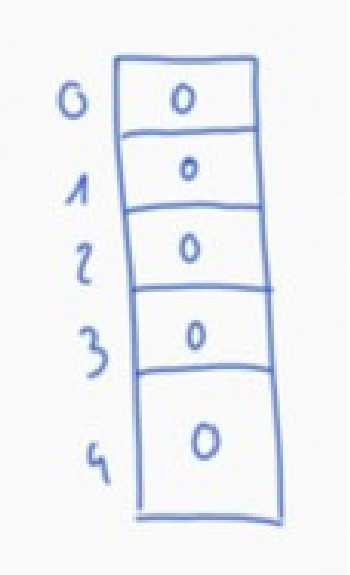
# Exemple



# Exemple

- On commence par créer la liste de taille 5

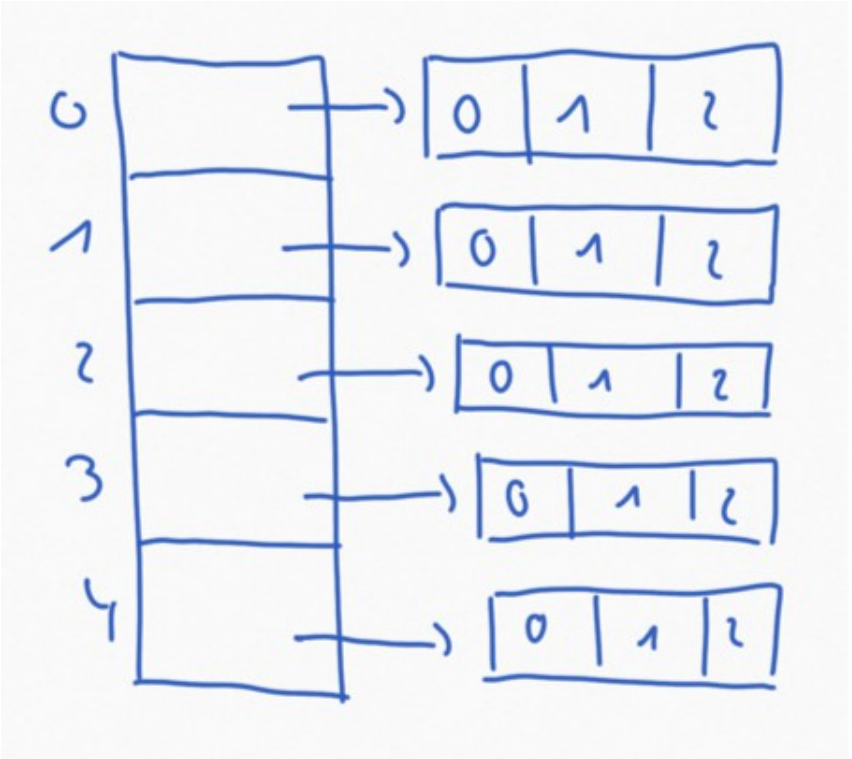
```
li=[0]*5
```



# Exemple

- Il faut ensuite remplir chacune des cases

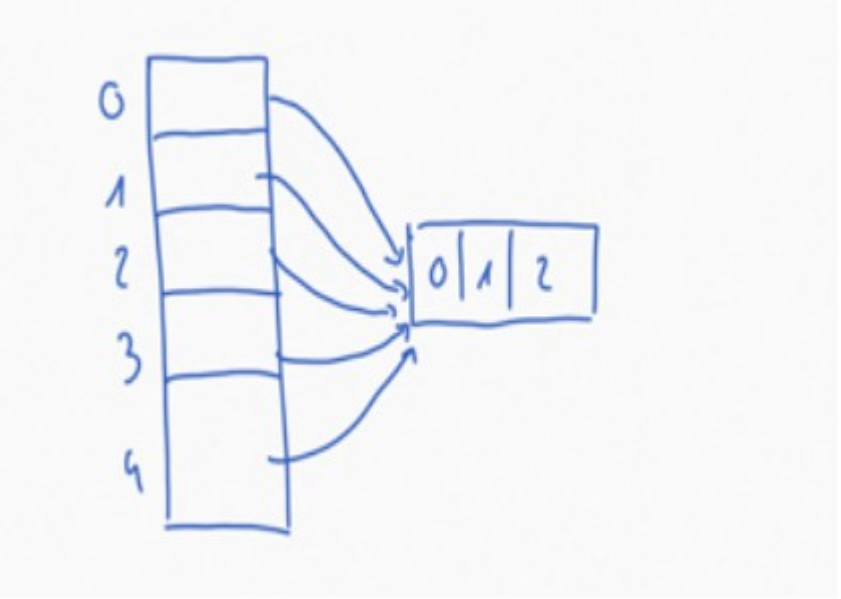
```
li=[0]*5
for i in range(0,5,1):
  li[i] = {0,1,2}
```



# Exemple

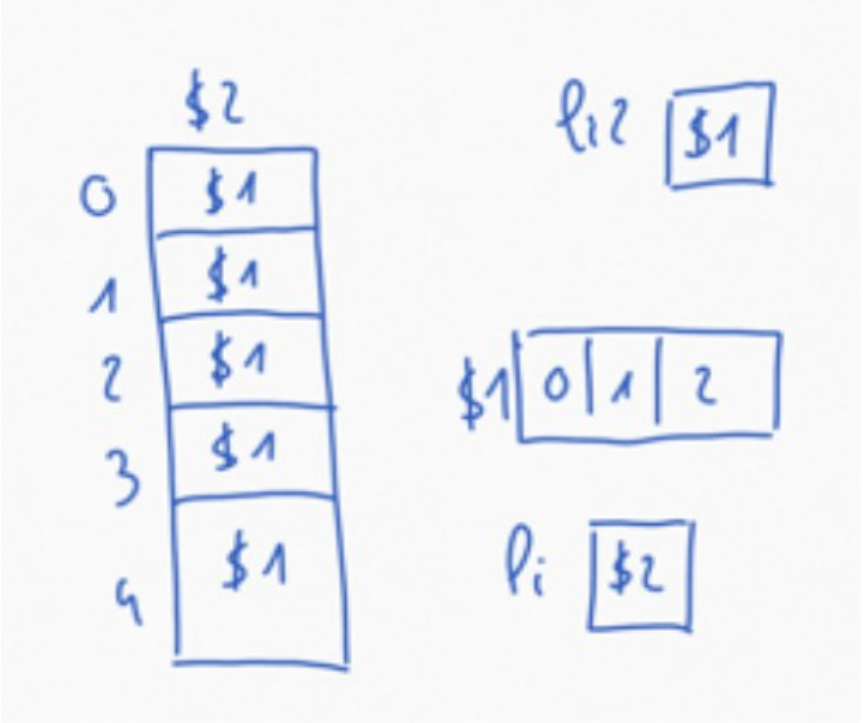
- Il faut faire attention aux nombres de listes créées

```
li2={0,1,2}
li=[0]*5
for i in range(0,5,1) :
   li[i]=li2
```



# Représentation en mémoire

```
li2={0,1,2}
li=[0]*5
for i in range(0,5,1) :
   li[i]=li2
```



# Exemple

```
li2={0,1,2}
li=[0]*5
for i in range(0,5,1) :
   li[i]=li2

li[0][0]=5
print(li[1][0])
```

**Ce programme affiche 5**
**Toutes les listes li[0],li[1],li[2],li[3] et li[4] sont en fait les mêmes**

# Récapitulatif

- Quand on a une liste de liste d'entiers li (on appelle cela une liste à dimension 2)
- Chaque élément li[0], li[1],…, li[len(li)-1] est aussi une liste
- Donc li[j] (pour j entre 0 et len(li)-1) est une liste
  - Elle a une longueur len(li[j])
  - On peut accéder à ses élements et les modifier grâce à li[j][k] (pour k entre 0 et len(li[j])-1)
- Cela se généralise aux listes de listes de listes d'entiers, si lis est une liste d'entiers de dimension 3, alors les li[i][j] sont des listes et on accède à leurs éléments en faisant

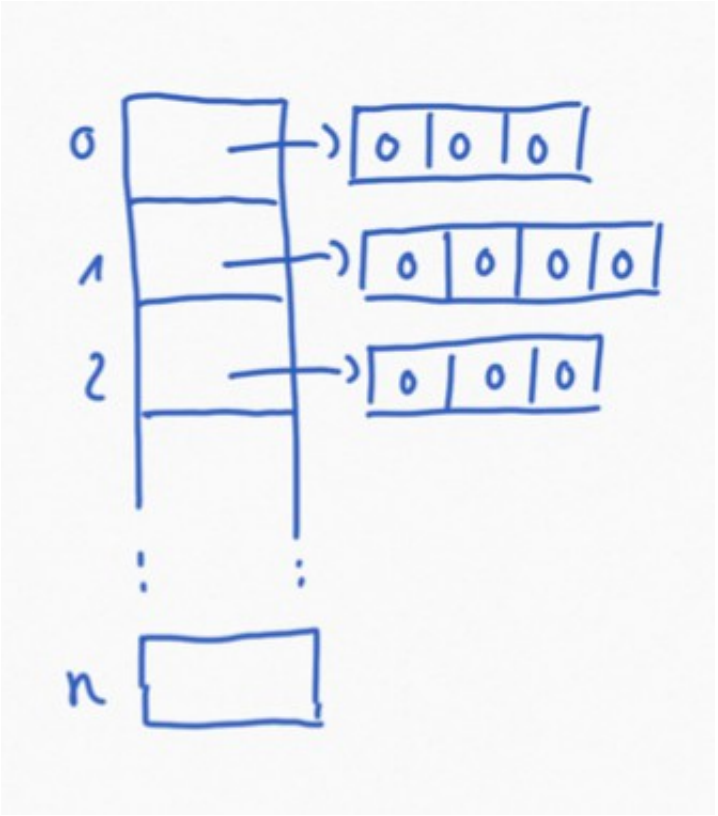
  l[i][j][k]

# À propos des tailles des sous-listes

- Il faut toujours faire attention à la taille de la liste qu'on ait en train de parcourir
- Les sous-listes d'une liste d'entiers de dimension 2 n'ont pas forcément toute la même taille
- Ainsi on peut avoir une liste de liste d'entiers li remplie de 0 où la taille de la liste li[i]
  - est de 3 si i est paire
  - est de 4 si i est impaire
- Écrivons une fonction oddEvenList qui renvoie une telle liste de taille n

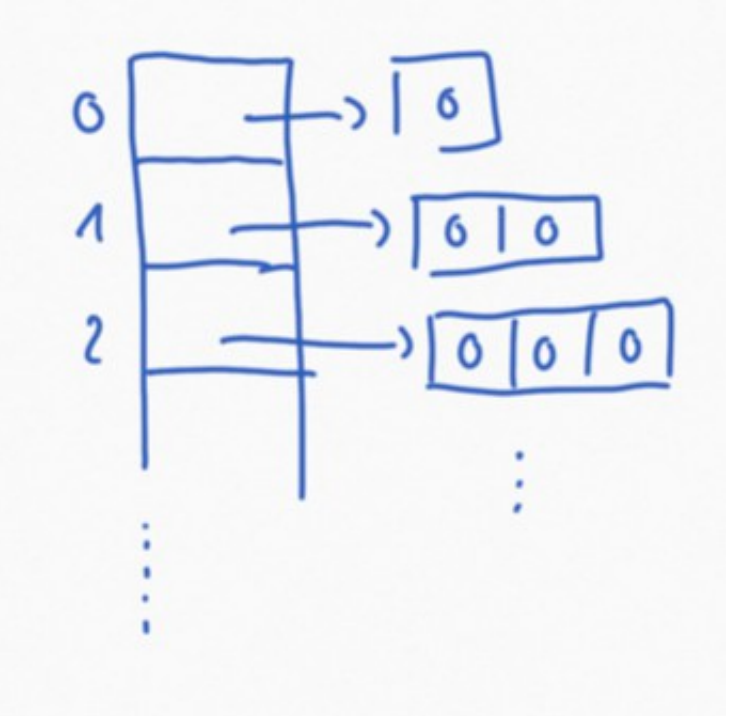
# Exemple

```
def oddEvenLis(n):
    li=[0]*n
    for i in range(0,n,1):
        if(i%2==0):
            li[i]=[0]*3
        else:
            li[i]=[0]*4
    return li
```



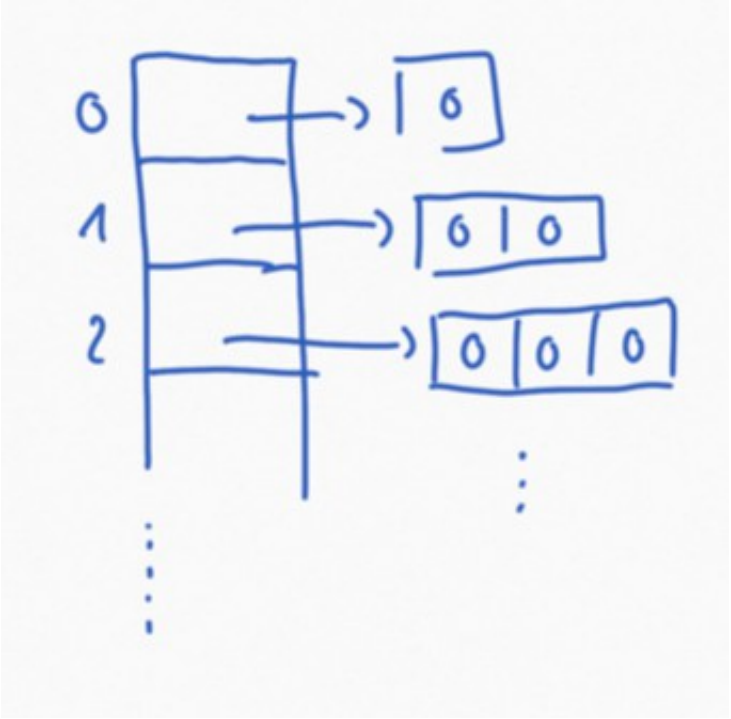
# Un exemple plus complexe

- Écrire une fonction pyramide qui crée une liste de liste d'entiers lis de taille n (donné en argument) remplie de 0 telle que la taille de chaque sous-liste lis[i] est égale à i+1



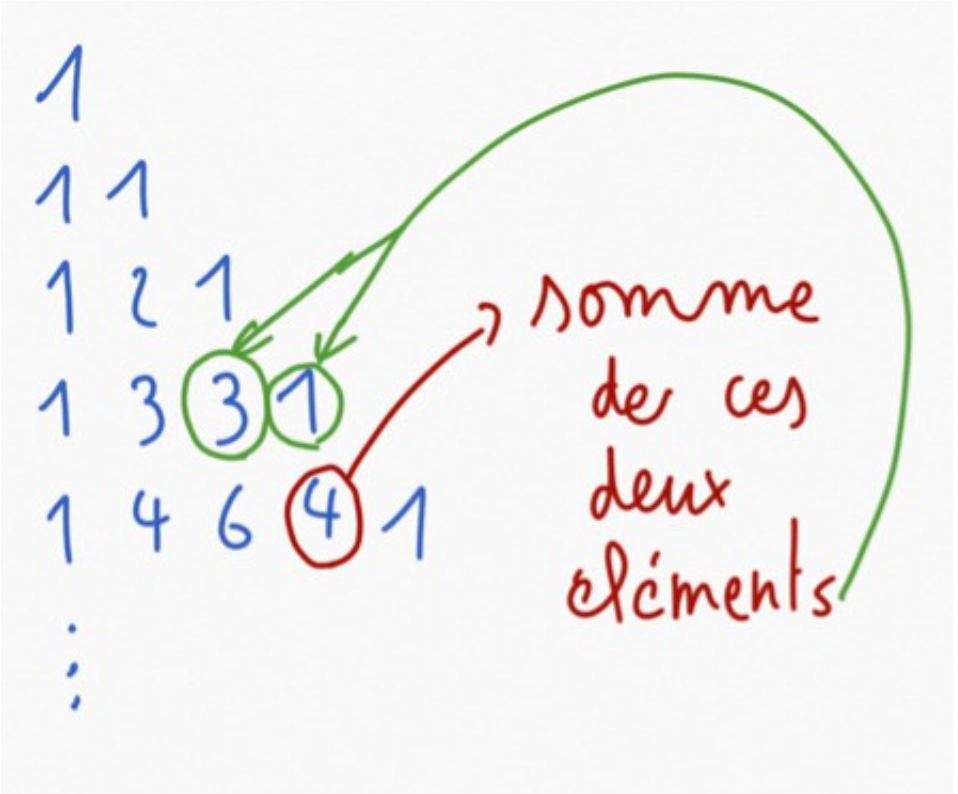
# Exemple

```
def pyramide(n) :
    lis=[0]*n
    for i in range(0,n,1) :
        lis[i]=[0]*(i+1)
    return lis
```



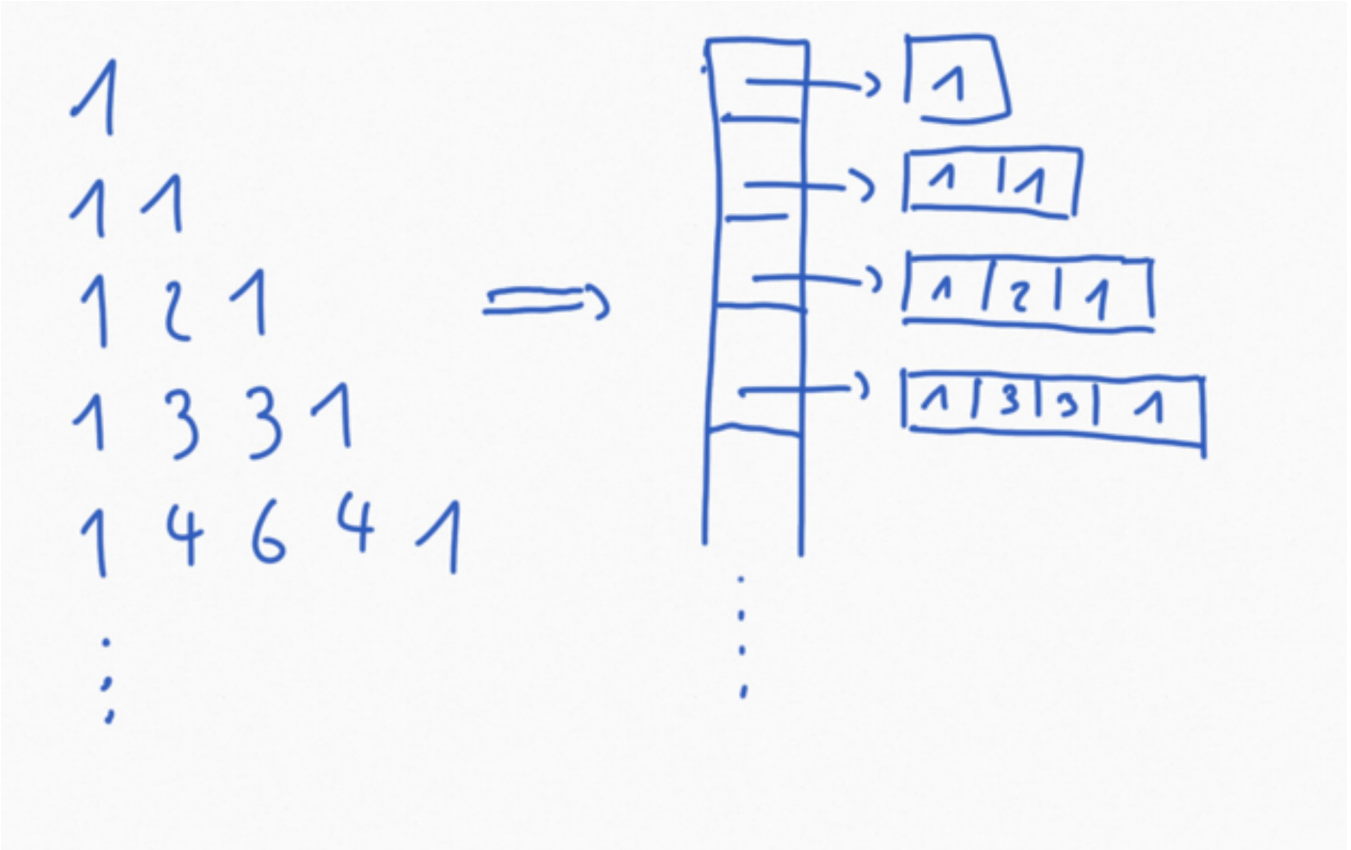
# Un classique: le triangle de Pascal

- Il donne les coefficients binomiaux $C_n^k = n!/(k!(n-k)!)$
- À la ligne i et la colonne j on trouve $C_i^j$
- On sait que $C_i^j = C_{(i-1)}^{j} + C_{(i-1)}^{(j-1)}$



# Un classique: le triangle de Pascal

- Faire une fonction qui prend en argument un entier n et qui crée une liste de n lignes où chaque ligne représente une ligne du triangle de Pascal

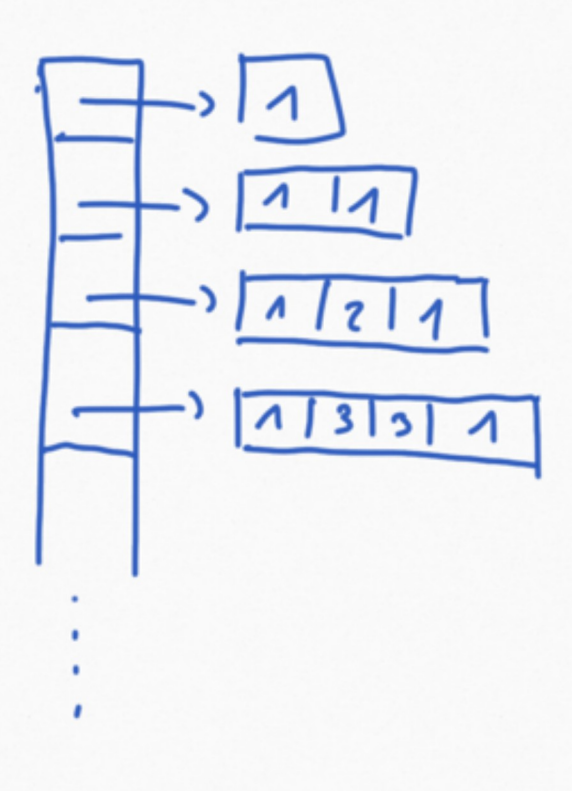


# Un classique: le triangle de Pascal

- Il faut faire attention à la façon dont on construit le triangle
- Chaque ligne dépend de la ligne précédente
- Il faut faire attention aux premiers et derniers éléments de chaque ligne qui valent 1
- Supposons que la liste créée s'appelle lp
- Alors la longueur de lp[i] est i+1
- lp[i][0]=1 et lp[i][len(lp[i])-1]=1
- si j est entre 1 et len(lp[i])-2 et i >0 alors :
  - lp[i][j]=lp[i-1][j]+lp[i-1][j-1]

# Solution triangle de Pascal

```
def pascal(n) :
  lp=[0]*n
  for i in range(0,n,1) :
    lp[i]=[0]*(i+1)
    lp[i][0]=1
    lp[i][i]=1
    for j in range(1,i,1) :
      lp[i][j]=lp[i-1][j]+lp[i-1][j-1]
  return lp
```



# Comment parcourir une liste de listes de façon sûre

- Supposons que nous ayons une liste de liste d'entiers li,mais non ne savons rien sur ce qu'elle contient
- Nous voulons faire un programme  qui affiche sur chaque ligne les éléments de chacune des sous-listes séparés par des virgules
- Il faut pour cela faire une boucle pour i allant 0 à len(li)-1 et dans cette boucle
  - faire une boucle pour j allant de 0 à len(li[i])-1
  - afficher une  virgule après chaque élément sauf le dernier où la on affiche un retour à la ligne

# Solution parcours

```
def parcours(li) :
  for i in range(0, len(li), 1) :
    for j in range(0,len(li[i]),1) :
       print(l[i][j],end="")
       if (j==len(li[i])-1) :
          print()
       else :
          print(",",end="")'
```

# Solution parcours : erreur classique

```
def parcours(li) :
  l=li[0]
  for i in range(0, len(li), 1) :
    for j in range(0,l,1) :
      print(l[i][j],end="")
      if (j==len(li[i])-1) :
        print()
      else :
        print(",'",end="")'
```

**Ne marche pas car :**
**1) Toutes les sous-listes n'ont pas la même taille**
**2) Si ça trouve la liste li est vide**

# Lien avec les matrices

- Une matrice à n lignes et m colonnes peut être facilement représentée par une liste de taille n dont chaque élément est une liste d'entiers de taille m
- Si on appelle liMat cette liste,liMat[i][j] est l'élément à la i-ème et j-ème colonne

# Exemple

- Une matrice à n lignes et m colonnes peut être facilement représentée par une liste de taille n dont chaque élément est une liste d'entiers de taille m