# Séance 8b: Listes imbriquées

Université Paris-Diderot

Objectifs:

- Apprendre à lire, modifier et créer des listes imbriquées (listes de listes) d'entiers ou de chaînes de caractère.
- Chercher si une condition est satisfaite par (au moins) des éléments d'une liste imbriquée.
- Compter le nombre d'élements vérifiant une condition donnée dans une liste imbriquée.
- Faire la liste imbriquée des positions des éléments vérifiants une condition donnée.
- Créer un tableau vérifiant certains motifs.

On se familiarise avec les listes de listes (ou liste à deux dimensions). Elles demandent naturellement l'utilisation de boucles imbriquées. Ce sujet est **long**. L'objectif est de traiter tous les thèmes en faisant le début de chaque exercice (les questions finales sont réservées à l'entraînement et au travail personnel).

**Exercice 1 (Affichage ligne par ligne, ⋆)**
*Écrire la procédure* `printLines` *qui attend une liste de listes et affiche son contenu ligne par ligne (un ligne est une* liste imbriquée*). On se servira de cette fonction pour tester certaines fonctions demandées dans les exercices ultérieurs.*

***Contrat:***
*Par exemple,* `printLines([[1, 2, 3], [4, 5], [6]])` *va afficher :*

```
1 2 3
4 5
6
```

□

**Exercice 2 (Listes imbriquées d'entiers, ⋆)**
*Dans cet exercice, on adapte des tâches déjà posées pour les listes à une dimension. Vous n'êtes pas obligés de répondre à toutes les questions ! Il faut travailler les exercices suivants aussi. Faites au moins les trois ou quatre premières. Tous les programmes de cet exercice prennent (au moins) une liste de listes d'entiers en argument.*

1. *Programmer* `sumListOfLists`*, la fonction qui attend une liste de listes d'entiers et retourne la somme de tous les entiers qu'elle contient.*

   ***Contrat:***
   *Par exemple,* `sumListOfLists([[1, 2, 3], [4, 5]])` *doit retourner* `15`*.*

2. *Programmer* `holdsOddInt`*, la fonction qui attend une liste de listes d'entiers et retourne* `True` *ssi au moins un entier est impair. Elle devra être écrite de façon à ce que le programme s'arrête dès qu'il rencontre un entier impair (si la liste en contient un).*

   ***Contrat:***
   *Par exemple,* `holdsOddInt([[2, 4, 6], [8, 10], [12]])` *doit retourner* `False` *et* `holdsOddInt([[4, 6, 3], [2], [9, 11]])` *doit retourner* `True`*.*

3. *Programmer* `numberOfOneDigitInt`, *la fonction qui attend une liste de listes d'entiers et retourne le nombre d'entiers (positifs) qui s'écrivent avec un seul chiffre.*

   ***Contrat:***
   *Par exemple,* `numberOfOneDigitInt([[11, 3, 12], [1, 100]])` *doit retourner* `2`.

4. *Programmer* `positionList`, *la fonction qui attend une liste de listes d'entiers et renvoie la liste des positions des multiples de 9 qu'elle contient. Alternativement, on peut retourner une liste de listes de booléens.*

   ***Contrat:***
   *Par exemple,* `positionList([[9, 4, 27], [81], [3, 45]])` *doit retourner* `[[1, 0, 1], [0], [0, 1]]`.

5. *(⋆⋆) Programmer* `sumOfEvenInt`, *la fonction qui attend une liste de listes d'entiers et retourne la somme de tous les entiers* pairs *qu'elle contient.*

   ***Contrat:***
   *Par exemple,* `sumOfEvenInt([[1, 2, 3], [4, 5]])` *doit retourner* `6`.

6. *(⋆⋆) Programmer* `rowSums`, *la fonction qui attend une liste de listes d'entiers et retourne la* liste *constituée des sommes de chaque ligne.*

   ***Contrat:***
   *Par exemple,* `rowSums([[2, 3], [5, 8, 13, 21], [34]])` *doit retourner* `[5, 47, 34]`.

7. *(⋆⋆) Programmer* `rowWiseCount`, *la fonction qui attend un entier* `n` *et une liste de listes d'entiers* `lis` *et retourne la* liste *du nombre d'entiers plus grands (strictement) que* `n` *dans chaque ligne.*

   ***Contrat:***
   *Par exemple,* `rowWiseCount(10,[[12,1], [37, 8, -1, 21], [0]])` *doit retourner* `[1, 2, 0]`.

8. *(⋆ ⋆ ⋆) Programmer* `columnSums`, *la fonction qui attend une liste de listes d'entiers et retourne la* liste *constituée des sommes de chaque colonne. Le programme devra tenir compte du fait que les lignes n'ont pas forcément la même longueur. Pour chaque colonne, il faut donc éventuellement sauter certaines lignes.*

   ***Contrat:***
   *Par exemple,* `columnSums([[2, 3], [5, 8, 13, 21], [34]])` *doit retourner* `[41, 11, 13, 21]`.

□

**Exercice 3 (Listes imbriquées de chaînes de caractère, ⋆⋆)**
*Tous les programmes de cet exercice prennent (au moins) une liste de listes de chaînes de caractères en argument.*

1. *(⋆) Programmer* `holdsCharlie`, *la fonction qui attend une liste de listes de chaînes de caractères et retourne* `True` *ssi la liste contient la chaîne* `"Charlie"`. *Elle devra aussi être écrite de façon à ce que le programme s'arrête dès qu'il rencontre* `"Charlie"` *(si la liste en contient cette chaîne).*

   ***Contrat:***
   *Par exemple,* `holdsCharlie([["Riri", "Fifi", "Loulou"], ["Charlie", "Georgio", "Valéry"], ["Franz"]]` *doit retourner* `True`.

2. *Programmer* `holdsE`, *la fonction qui attend une liste de listes de chaînes de caractères et retourne* `True` *ssi une des chaînes de la liste contient la voyelle* `e`. *Elle devra aussi être écrite de façon à ce que le programme s'arrête dès qu'il rencontre un* `e` *(si une chaîne en contient un).*

   ***Contrat:***
   *Par exemple,* `holdsE([["Il", "abandonna", "son", "roman", "sur", "son", "lit"], ["Il", "alla", "à", "son", "lavabo''"]]` *doit retourner* `False`.

3. *Programmer* `fWordPositions`, *la fonction qui attend un chaîne de caractère* `s` *et une liste de listes de chaînes de caractères* `lis` *et retourne les positions de la chaîne* `s` *dans la liste* `lis`. *Alternativement, on peut retourner une liste de listes de booléens.*

***Contrat:***
*Par exemple,* `fWordPositions("son",[["Il", "abandonna", "son", "roman", "sur", "son", "lit"], ["Il", "alla", "à", "son", "lavabo''']]` *doit retourner* `[[0, 0, 1, 0, 0, 1, 0], [0, 0, 0, 1, 0]]`.

□

**Exercice 4 (Création de listes imbriquées, ⋆⋆)**

1. *(⋆) Programmer* `squareOfZeros`*, la fonction qui attend un entier* `n` *et renvoie la liste de listes de dimensions* `n` $\times$ `n` *et ne contenant que des 0.*

   ***Contrat:***
   *Par exemple,* `squareOfZeros(3)` *va retourner la liste* `[[0, 0, 0], [0, 0, 0], [0, 0, 0]]`
2. *(⋆) Programmer* `rectOfInt`*, la fonction qui attend deux entiers* `n` *et* `p`*, et et renvoie la liste de listes de dimensions* `n` $\times$ `p` *contenant, dans l'ordre de lecture,* 1, 2, 3*...*

   ***Contrat:***
   *Par exemple,* `rectOfInt(3, 2)` *va retourner la liste* `[[1, 2, 3], [4, 5, 6]]`
3. *Programmer* `triangleOfInt`*, la fonction qui attend un entier* `n`*, et renvoie la liste triangulaire de* `n` *lignes, qui contient 1 sur sa 1ère ligne, 2 et 3 sur sa 2ème ligne, 4, 5, 6 sur sa 3ème ligne, 7, 8, 9, 10 sur sa 4ème ligne, etc.*

   ***Contrat:***
   *Par exemple,* `triangleOfInt(3)` *va retourner la liste* `[[1], [2, 3], [4, 5, 6]]`

*(⋆⋆⋆) Programmer* `target`*, la fonction qui attend un entier* `n` *et renvoie la liste d'entiers carrée de taille* `n`*, gbordée par des 1 et telle que, à chaque fois qu'on se rapproche d'une case vers le centre (horizontalement ou verticalement), on augmente de 1.*

***Contrat:***
*Par exemple,* `target(6)` *va retourner la liste de listes :*

```
1 1 1 1 1 1
1 2 2 2 2 1
1 2 3 3 2 1
1 2 3 3 2 1
1 2 2 2 2 1
1 1 1 1 1 1
```

□

**Exercice 5 (Bonus : applications aux mathématiques, ⋆⋆⋆)**

1. *Le triangle de Pascal apparaît en algèbre et en probabilités. Il est donné par une famille d'entiers* $\binom{n}{k}$*, où* $n$ *et* $k$ *sont des entiers naturels tels que* $k \leq n$.
   *Le triangle de Pascal se calcule par* $\binom{n}{n} = \binom{n}{0} = 1$ *et* $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$.
   *Programme la fonction* `pascal`*, qui attend un entier* `sz` *et retourne la liste triangulaire d'entiers tels que le terme à la* `n`*-ème ligne et la* `k`*-ème colonne est* $\binom{\texttt{n}}{\texttt{k}}$*, pour* `k` $\leq$ `n` $\leq$ `sz`

   ***Contrat:***
   *Par exemple,* `pascal(4)` *va retourner* `[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]`.

2. *On rappelle que la suite de Fibonacci est commence par 1, 1, 2, 3, 5, 8, 13, 21, 34... : en général, chaque terme est obtenu en effectuant la somme des deux précédents.*
   *Programmer* `fiboByDigits`, *la fonction qui attend un entier* `n` *et retourne la liste de* `n` *premiers termes de la suite de Fibonacci avec la condition suivante : les termes de 1 chiffre seront sur la 1ère ligne, les termes de 2 chiffres seront sur la 2ème ligne, les termes de 3 chiffres sur la 3ème et ainsi de suite.*

   ***Contrat:***

   *Par exemple,* `fiboByDigits(13)` *va retourner* `[[1, 1, 2, 3, 5, 8], [13, 21, 34, 55, 89], [144, 233]]`.

   *Afficher* `fiboByDigits(43)`. *Que remarque-t-on ?*

□