STRUCTURES DE DONNÉES :

PILES - LISTES - TABLEAUX

			4			8	7	
	4	7		9	2		5	
2			6				3	
9	7		5			2		3
5		8		2	4	7		6
6		4			7		8	5
	9		3		8			7
		3	2	4		1	6	
	1	2					9	

```
mat=np.zeros((9,9),int)

wb= load_workbook('sudoku.xlsx')
ws=wb['Grille-2']

for i in range(9):
    for j in range(9):
        val=ws.cell(row=i+1,column=j+1).value
        if val!=None:
            mat[i,j]=val
```

Objectifs

A la fin de la séquence d'enseignement l'élève doit pouvoir manipuler les structures de données suivantes :

- piles (ajout d'un élément, suppression du dernier)
- listes (création, ajout d'un élément, suppression d'un élément, accès à un élément, extraction d'une partie de liste)
- tableaux à une ou plusieurs dimensions
- chaînes de caractères

Table des matières

1 Introduction 2

2 Les piles et les files 2

2.1 Définitions . . . 2
2.2 Principales fonctions associées . . . 3
2.3 Piles et files définies par listes chaînées . . . 3
2.4 Pile définie par un tableau P de n éléments . . . 3
2.5 File définie par un tableau F de n éléments . . . 4

3 Les listes Python 5

4 Les ensembles 5

5 Les dictionnaires 5

6 Les tableaux Python 5

7 Chaînes de caractères 6

8 Manipulation de fichiers 6

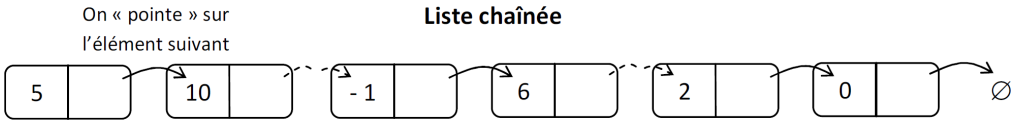
1 Introduction

Les listes (pas au sens python) sont des données composites. Elles rassemblent dans une seule structure un ensemble d'entités plus simples qui peuvent être de types différents. En général, on traite plutôt de listes d'éléments de même type, par exemple des listes de nombres.

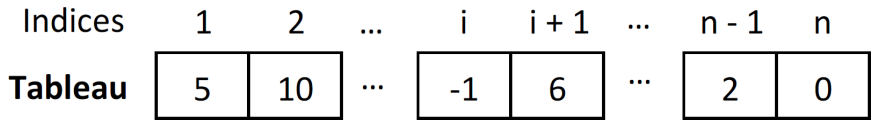
La liste est un objet de la catégorie des séquences, lesquelles sont des collections ordonnées d'éléments. Cela signifie simplement que les éléments d'une liste sont toujours disposés dans un certain ordre. Contrairement aux chaînes de caractères, les listes sont des séquences modifiables.

On peut implémenter une liste de deux manières :

- **par une liste chaînée :** Chaque maillon est un objet dans lequel on stocke l'élément et une information pour trouver le suivant (voire aussi le précédent). C'est une structure de données récursive. La liste chaînée n'a pas de taille fixée à l'avance. Seul l'objet ń maillon ż a une certaine dimension.



- **par un tableau :** Chaque élément de la liste peut être désigné par sa place dans la séquence, à l'aide d'un index. Pour accéder à un élément déterminé, on utilise le nom de la variable qui contient la liste et on lui accole entre deux crochets, l'index numérique qui correspond à sa position dans la chaîne. Le tableau a une dimension fixe. Certains langages (comme Python) permettent de modifier sa taille au cours de son utilisation.



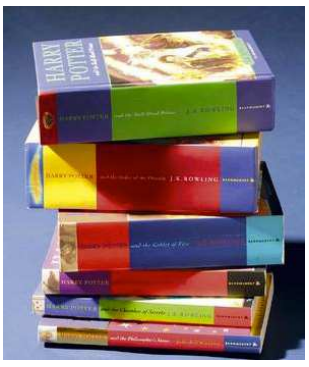
2 Les piles et les files

2.1 Définitions

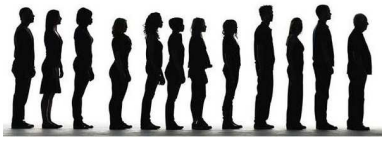
Les piles et les files sont des listes particulières : on accède aux éléments par les extrémités, c'est-à-dire au début ou à la fin.

On distingue :

les piles (*stacks*)
le premier empilé est le dernier à être dépilé
LIFO (Last In first Out)



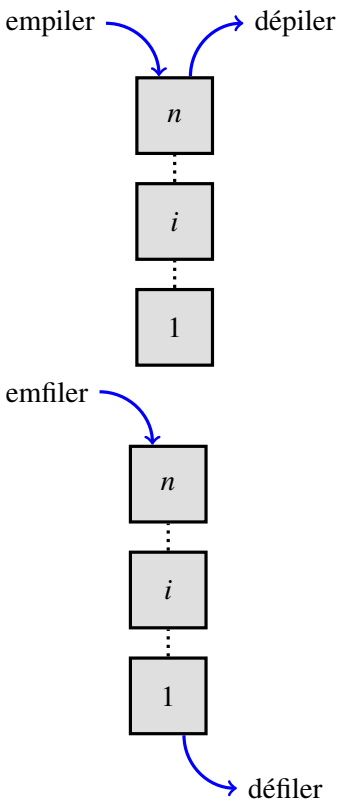
les files
le premier entré est le premier à sortir
FIFO (First In First Out)



2.2 Principales fonctions associées

Pour les piles, les principales fonctions associées aux piles sont :

- ajouter un sommer un élément `x`: **empiler** `Pile.append(x)`
- supprimer du sommet le dernier élément de la pile : **dépiler** `Pile.pop()`.
  REMARQUE: on peut récupérer cet élément en l'affectant à une variable : `x=Pile.pop()`
- tester si la pile est vide : **est_vide** `Pile==[]`
- tester si la pile est plein : **est_pleine** avertissement par `overflow` ou `stackoverflow`



Pour les files, les principales fonctions associées aux piles sont :

- ajouter en queue : **enfiler**
- supprimer en tête : **défiler**
- tester si la pile est vide : **est_vide** `File==[]`
- tester si la pile est plein : **est_pleine** avertissement par `overflow` ou `stackoverflow`

2.3 Piles et files définies par listes chaînées

Voir Tp *ALGO-PROG-TP3 : structures de données*.

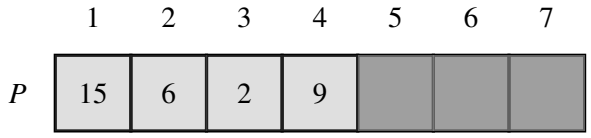
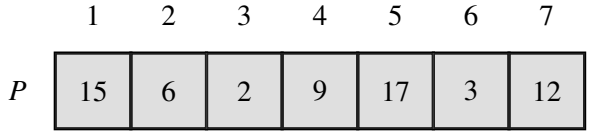
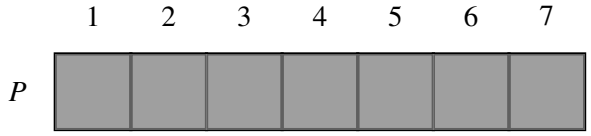
2.4 Pile définie par un tableau P de *n* éléments

- l'ordre est déterminé par les indices des éléments du tableau `P[1:n]`
- le tableau possède un attribut `sommet[P]` qui indexe l'élément le plus récemment inséré
- la pile se constitue des éléments `P[1:sommet[P]]` où `P[1]` est l'élément de base de la pile (le plus ancien) et `P[sommet[P]]` est l'élément situé au sommet (le plus récent)
- quand sommet[P] = 0, la pile est vide
- Si **sommet[P]** est supérieur à *n*, on dit que la pile déborde
  REMARQUE: Dépiler une pile vide provoque une erreur ; on parle de débordement négatif.

PileVide (P)
1: **si** sommet[P] = 0 **alors**
2: **renvoi:** Vrai
3: **sinon**
4: **renvoi:** Faux
5: **fin si**

Empiler (P,x)
1: Sommet[P] ←sommet[P]+1
2: P[Sommet[P]] ←x

Dépiler (P)
1: **si** PileVide(P) **alors**
2: "débordement négatif" (Erreur)
3: **sinon**
4: sommet[P] ←sommet[P]-1
5: **renvoi:** P[sommet[P]+1]
6: **fin si**

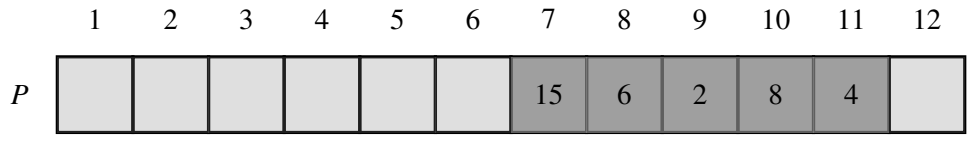
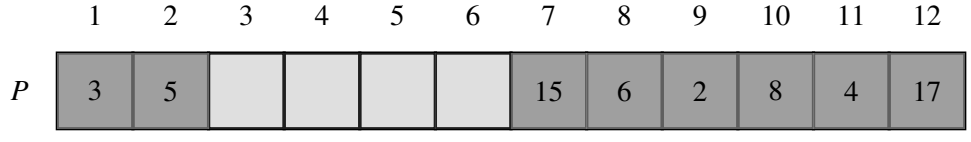
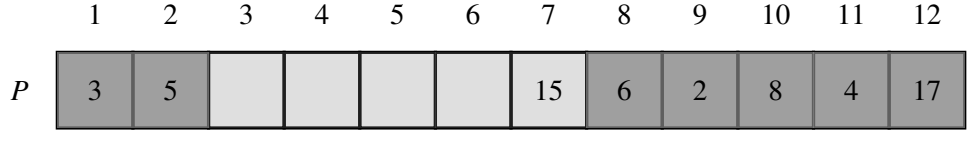
Pile	sommet[P]	Empiler	Dépiler
	4	on place en 5	on récupère 9
	7	débordement !	on récupère 12
	0	on place en 1	débordement négatif !

2.5 File définie par un tableau F de n éléments

- la file a un attribut `tête[F]` qui repère sa tête
- la file a un attribut `queue[F]` qui indexe le prochain emplacement pouvant conserver un nouvel élément.
- les éléments de la file sont repérés par `tête[F]`, `tête[F+1]`,...,`queue[F]-1`
- l'emplacement 1 suit l'emplacement n dans un ordre circulaire
- quand `tête[F] = queue[F]`, la file est vide
  REMARQUE: au départ, on a `tête[F]=queue[F]=1`
- quand `tête[F] = queue[F]+1`, la file est pleine.

Enfiler(F,x)
1: F[queue[F]] ←x
2: **si** queue[F] = longueur[F] **alors** queue[F] ←1
3: **sinon** queue[F] ←queue[F]+1
4: **fin si**

Défiler(F)
1: x ←F[tête[F]]
2: **si** tête[F] = longueur[F] **alors** tête[F] ←1
3: **sinon** tête[F] ←tête[F]+1
4: **fin sirenvoi:** x

File	tête	queue	action précédente
	7	12	Les éléments 15, 6, 2, 8 et 4 ont été enfilés
	7	3	Les éléments 17, 3 et 5 ont été enfilés
	8	3	l'élément 15 à été défilé

3 Les listes Python

Beaucoup plus pratique que les listes chaînées qui sont briques de bases de la programmation, les listes en python sont des mutables permettant de faire :

- ajouter un élément en fin de liste : `L.append(5)`
- récupérer le dernier élément mais en l'enlevant : `x=L.pop()`
- supprimer l'élément *i* d'une liste `L` : `del L[i]`
- accéder directement à l'élément *n* d'une liste : `L[n]`
- changer directement (mutabilité) un élément *n* d'une liste : `L[n]=3`
- accéder à plusieurs éléments consécutifs (slicing) : `L[1:3]`
- connaître la longueur de la liste: `len(L)`
- assembler des listes bout à bout (concaténation) : `L+[1,2,3]`
- obtenir une liste composée de *n* fois la liste `L` mise bout à bout : `L*3`
- des listes de ce qu'on veut :
  - liste de chaînes de caractère : `L=['Je ne ','Peppa Pig','pas','suis']`
  - listes de listes : `[1,2,[3,4,[5,8],[2,3,4],3],1]`

REMARQUE: Attention à la copie :

- écrire `L1=L2` conduit à changer les éléments de `L1` lorsqu'on change les éléments de `L2`
- pour éviter ce problème, on peut écrire :
  - `L1=L.copy()`
  - `L1=list(L)`
- attention la copie est superficielle.

```
>>> L=[[1],[2],[3]]
>>> L1=L.copy()
>>> L[0][0]=0
>>> L
[[0], [2], [3]]
>>> L1
[[0], [2], [3]]
>>> L[0]=[7]
>>> L1
[[0], [2], [3]]
>>> L
[[7], [2], [3]]
```

4 Les ensembles

On dispose en python d'un type ensemble : set. Comme en maths, l'ordre des éléments n'importe pas, et il n'apparaissent qu'une seule fois. C'est un type mutable.

- **Ensemble vide :** {} ou `set()`
- **Affectation :** `s={1,2,3}` affecte l'ensemble `{1,2,3}`.
- **Cardinal :** `len(s)`
- **Appartenance :** `x in s`, `x not in s`.
- **Inclusion :** `s1.issubset(s2)` ou `s1 <= s2`
- **Union :** `s1.union(s2)` ou `s1|s2`
- **Intersection :** `s1.intersection(s2)` ou `s1&s2`
- **Différence :** `s1.difference(s2)` ou `s1-s2`
- **et aussi :** `s.add(x)`, `s.remove(x)`...

Pour la copie, les mêmes règles que pour les listes s'appliquent !

5 Les dictionnaires

Ça existe en python mais c'est hors programme.

6 Les tableaux Python

Les tableaux python s'obtiennent avec **numpy** à partir de la commande `array`. Nous les étudierons quand vous aurez quelques notions d'algèbre linéaire ...

En attendant, on utilisera des listes de listes à la place de tableau à plusieurs dimension. Ainsi, pour accéder à l'élément de la ligne *i* du tableau `T` et de la colonne *j*, on utilisera la liste `L` telle que :

`T[i,j]=L[i][j]`

7 Chaînes de caractères

Chaîne de caractères (*str* de string en anglais) sont des suites finies de caractères. Elles sont délimitées par des apostrophes `'`, des guillemets `"` ou des triples guillemets `"""` si on veut aller à la ligne à l'intérieur.

Les renvois à la ligne sont obtenus avec `\n` , les sauts de ligne avec `\r` et `\t` affiche une tabulation. De même, `\'` et `\"` permettent de faire apparaître les caractères.

Les éléments ne sont pas mutables mais beaucoup de propriétés des listes sont partagées avec les chaines de caractères :

- accéder au $i^{\text{ème}}$ caractère d'une chaîne `s` en plaçant l'indice entre crochets `s[i]`.
  REMARQUE: la numérotation commence à 0 et elle accepte des indices négatifs : -1 donne le dernier caractère, -2 l'avant dernier,...
- extraire des caractères d'une chaine pour en faire une sous-chaine `s[i:j]`
- concaténer des chaînes (les mettre bout-à-bout) grâce à l'opérateur +
- calculer la longueur d'une chaîne grâce à l'opérateur `len()`
- convertir des objets en chaîne de caractère grâce à l'instruction `str()`
- évaluer des chaines de caractères grâces à l'instruction `eval()`
- faire des test d'inclusion avec l'opérateur `in`

Enfin, pour une chaîne de caractère `chaine` :

- supprimer les retours à la ligne : `chaine.rstrip()`
- mettre en majuscules : `chaine.upper()`
- mettre en minuscules : `chaine.lower()`

8 Manipulation de fichiers

Pour lire ou sauvegarder du contenu, il est possible d'avoir recours à des fichiers :

- ouvrir en lecture : `monfichier = open('lenom.txt','r')`
- ouvrir en écriture : `monfichier = open('lenom.txt','w')`
- fermeture : `monfichier.close()`
- lire la ligne suivante : `monfichier.readline()`
- lire toutes les lignes : `for ligne in monfichier.readlines():`
- écrire dans le fichier : `monfichier.write('blahblah.. blah')`
- écrire dans le fichier et passer à la ligne suivante : `monfichier.write('blahblah.. blah'+'\n')`