Exercice 1

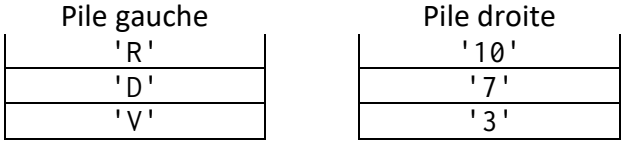
Thème abordé : structures de données : les piles

On cherche à obtenir un mélange d'une liste comportant un nombre **pair** d'éléments. Dans cet exercice, on notera N le nombre d'éléments de la liste à mélanger.

La méthode de mélange utilisée dans cette partie est inspirée d'un mélange de jeux de cartes :

- On sépare la liste en deux piles :
  - ⇨ à gauche, la première pile contient les N/2 premiers éléments de la liste ;
  - ⇨ à droite, la deuxième pile contient les N/2 derniers éléments de la liste.
- On crée une liste vide.
- On prend alors le sommet de la pile de gauche et on le met en début de liste.
- On prend ensuite le sommet de la pile de droite que l'on ajoute à la liste et ainsi de suite jusqu'à ce que les piles soient vides.

Par exemple, si on applique cette méthode de mélange à la liste `['V','D','R','3','7','10']`, on obtient pour le partage de la liste en 2 piles :



La nouvelle liste à la fin du mélange sera donc `['R','10','D','7','V','3']`.

1. Que devient la liste `['7','8','9','10','V','D','R','A']` si on lui applique cette méthode de mélange ?

On considère que l'on dispose de la structure de données de type pile, munie des seules instructions suivantes :

`p = Pile()` : crée une pile vide nommée `p`

`p.est_vide()` : renvoie Vrai si la liste est vide, Faux sinon

`p.empiler(e)` : ajoute l'élément `e` dans la pile

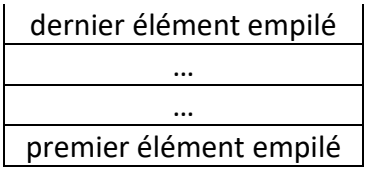
`e = p.depiler()` : retire le dernier élément ajouté dans la pile et le retourne (et l'affecte à la variable `e`)

`p2 = p.copier()` : renvoie une copie de la pile `p` sans modifier la pile `p` et l'affecte à une nouvelle pile `p2`

2. Recopier et compléter le code de la fonction suivante qui transforme une liste en pile.

```
def liste_vers_pile(L):
    '''prend en paramètre une liste et renvoie une
pile'''
    N = len(L)
    p_temp = Pile()
    for i in range(N):
      …….
    return …….
```

3. On considère la fonction suivante qui partage une liste en deux piles. Lors de sa mise au point et pour aider au débuggage, des appels à la fonction `affichage_pile` ont été insérés. La fonction `affichage_pile(p)` affiche la pile `p` à l'écran verticalement sous la forme suivante :




```
def partage(L):
    N = len(L)
    p_gauche = Pile()
    p_droite = Pile()
    for i in range(N//2):
        p_gauche.empile(L[i])
    for i in range(N//2,N):
        p_droite.empile(L[i])
    affichage_pile(p_gauche)
    affichage_pile(p_droite)
    return p_gauche, p_droite
```

Quels affichages obtient-on à l'écran lors de l'exécution de l'instruction : `partage([1,2,3,4,5,6])` ?

4.

4.a Dans un cas général et en vous appuyant sur une séquence de schémas, **expliquer** en quelques lignes comment fusionner deux piles `p_gauche` et `p_droite` pour former une liste `L` en alternant un à un les éléments de la pile `p_gauche` et de la pile `p_droite`.

4.b. **Écrire** une fonction `fusion(p1,p2)` qui renvoie une liste construite à partir des deux piles `p1` et `p2`.

5. Compléter la dernière ligne du code de la fonction `affichage_pile` pour qu'elle fonctionne de manière récursive.

```
def affichage_pile(p):
  p_temp = p.copier()
  if p_temp.est_vide():
    print('____')
  else:
    elt = p_temp.depiler()
    print('| ', elt, ' |')
    ...                          # ligne à compléter
```

EXERCICE 2

Cet exercice porte sur la programmation en général et la récursivité en particulier.

On s'intéresse dans cet exercice à un algorithme de mélange des éléments d'une liste.

1. Pour la suite, il sera utile de disposer d'une fonction `echange` qui permet d'échanger dans une liste `lst` les éléments d'indice `i1` et `i2`.
Expliquer pourquoi le code Python ci-dessous ne réalise pas cet échange et en proposer une modification.

```
def echange(lst, i1, i2):
    lst[i2] = lst[i1]
    lst[i1] = lst[i2]
```

2. La documentation du module `random` de Python fournit les informations ci-dessous concernant la fonction `randint(a,b)` :

```
Renvoie un entier aléatoire N tel que a <= N <= b. Alias
pour randrange(a,b+1).
```

Parmi les valeurs ci-dessous, quelles sont celles qui peuvent être renvoyées par l'appel `randint(0, 10)` ?

`0` `1` `3.5` `9` `10` `11`

3. Le mélange de Fischer Yates est un algorithme permettant de permuter aléatoirement les éléments d'une liste. On donne ci-dessous une mise en œuvre récursive de cet algorithme en Python.

```
from random import randint

def melange(lst, ind):
    print(lst)
    if ind > 0:
        j = randint(0, ind)
        echange(lst, ind, j)
        melange(lst, ind-1)
```

a. Expliquer pourquoi la fonction `melange` se termine toujours.

b. Lors de l'appel de la fonction `melange`, la valeur du paramètre `ind` doit être égal au plus grand indice possible de la liste `lst`.
Pour une liste de longueur n, quel est le nombre d'appels récursifs de la fonction `melange` effectués, sans compter l'appel initial ?

c. On considère le script ci-dessous :

```
lst = [v for v in range(5)]
melange(lst, 4)
```

On suppose que les valeurs successivement renvoyées par la fonction `randint` sont `2, 1, 2` et `0`.

Les deux premiers affichages produits par l'instruction `print(lst)` de la fonction `melange` sont :

```
[0, 1, 2, 3, 4]
[0, 1, 4, 3, 2]
```

Donner les affichages suivants produits par la fonction `melange`.

d. Proposer une version itérative du mélange de Fischer Yates.