EXERCICE 2 (6 points)

Cet exercice porte sur la programmation Python, la programmation orientée objet, les tests et la structure de données pile.

Le *mélange faro* consiste à partager un jeu de cartes en deux moitiés et intercaler les cartes de ces deux moitiés.

Pour tout l'exercice on notera `n` le nombre de cartes et on considérera qu'il est pair.

Pour modéliser le jeu de cartes, on décide d'utiliser une pile qui sera une instance de la classe `Pile` dont on donne ici l'interface.

- Le constructeur Pile ne prend pas de paramètres et renvoie une pile vide.
  `jeu = Pile()` *`# crée une pile vide référencée par jeu`*
- La méthode `empile` prend en paramètre une valeur et l'empile sur la pile.
  `jeu.empile(1)` *`# empile la valeur 1 sur la pile jeu`*
- La méthode `depile` ne prend pas de paramètres et retire le dernier élément empilé d'une pile non vide et renvoie sa valeur.
  `print(jeu.depile())` *`# dépile 1 et affiche cette valeur`*
- La méthode `est_vide` ne prend pas de paramètres et renvoie un booléen indiquant si la pile est vide.
  `print(jeu.est_vide())` *`# affiche True puisque la pile est vide`*

Le jeu de cartes est alors modélisé par une pile appelée `jeu` de sommet 1, puis 2 en dessous, *et cætera* jusqu'au bas de la pile qui contient n, comme illustré sur la figure ci-dessous.

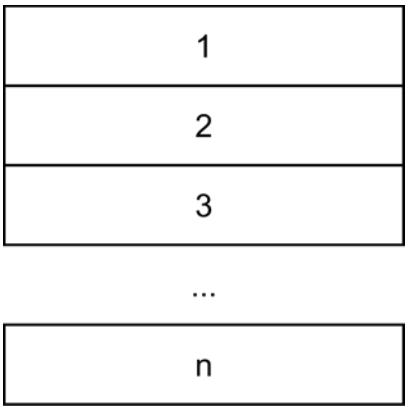

Figure 1. Pile représentant un jeu de cartes

Le mélange faro est réalisé ainsi :

- **Étape 1** : on dépile la moitié de `jeu` et chaque élément dépilé est empilé dans une deuxième pile appelée `moitie1` ;
- **Étape 2** : on dépile le reste de `jeu` et chaque élément dépilé est empilé dans une troisième pile appelée `moitie2` ;
- **Étape 3** : on empile alternativement dans `jeu` et dans cet ordre un élément de `moitie1` puis un élément de `moitie2` jusqu'à vider ces 2 piles.

Dans l'exemple suivant les contenus initiaux de `jeu`, `moitie1` et `moitie2` sont représentés ci-dessous :

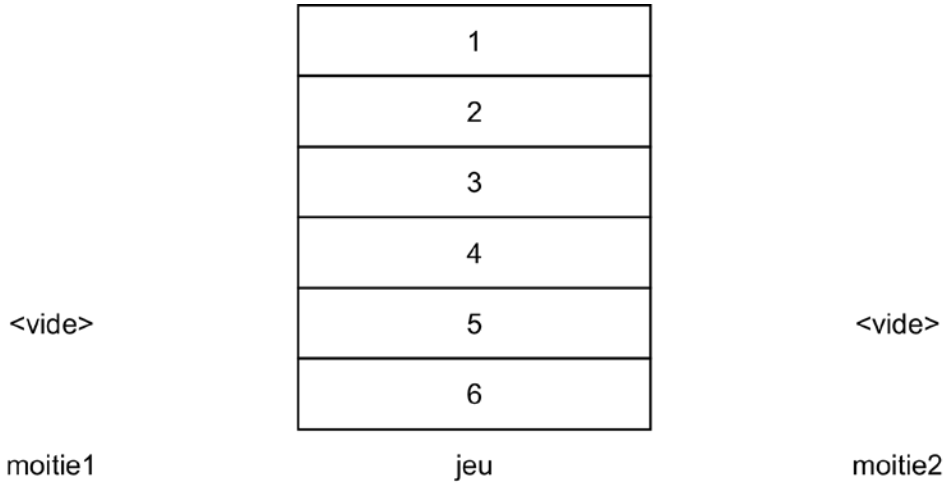

Figure 2. Contenus initiaux des 3 piles

1. Représenter sur votre copie les contenus de ces trois piles à la fin de chaque étape du mélange faro.

Voici le code de la fonction `produire_jeu` qui prend en paramètre un entier `n` supposé pair et qui renvoie une instance de la classe `Pile` qui représente le jeu de cartes.

```
1 def produire_jeu(n):
2     resultat = Pile()
3     for i in range(...):
4         resultat.empile(...)
5     return resultat
```

2. Recopier et compléter sur votre copie le code de la fonction `produire_jeu`.

Ci-après figure le code de la fonction `scinder_jeu` qui prend en paramètres une instance de taille paire de la classe `Pile` qui est le jeu que l'on veut partager en 2 moitiés, un entier `n` qui est la taille de la pile et qui renvoie deux piles qui sont les deux moitiés du jeu.

```
1 def scinder_jeu(p, n):
2     m1 = Pile()
3     m2 = Pile
4     for i in range(n):
5         m1.empile(p.depile())
6     for i in range(n):
7         m2.empile(p.depile())
8     return m1, m2
```

3. Ce code comporte des erreurs. Indiquer les numéros de lignes à rectifier ainsi que les rectifications à apporter.
4. Écrire une fonction `recombiner` qui prend en paramètres deux instances `m1` et `m2` de la classe `Pile` qui sont respectivement la première et la deuxième moitié d'un jeu de cartes et qui renvoie une instance de la classe `Pile` qui est le jeu obtenu en y empilant alternativement et dans cet ordre les éléments de `m1` et de `m2`.
5. Écrire une fonction `faro` qui prend en paramètres une instance de la classe `Pile` qui est le jeu que l'on veut mélanger, un entier `n` qui est la taille de la pile et qui renvoie une instance de la classe `Pile` qui contient le jeu obtenu en appliquant le mélange faro.

Une propriété mathématique assure qu'étant donné un jeu de n cartes (n pair), en répétant suffisamment de fois le mélange faro, on finira par remettre le jeu dans l'ordre initial. On aimerait trouver, pour un entier n donné, ce nombre minimal de répétitions nécessaires. Pour cela, on considère une fonction `identiques` qui prend en arguments deux instances de la classe `Pile` et qui renvoie un booléen indiquant si ces deux piles ont les mêmes éléments, en même nombre et dans le même ordre.

La fonction `identiques` ne modifie pas les piles données en entrée.

Pour s'assurer que la fonction `identiques` fonctionne correctement, on a commencé à produire un jeu de tests :

```
1 p1 = Pile()
2 p1.empile(1)
3 p2 = Pile()
4 assert not identiques(p1, p2)
```

6. Compléter ce jeu de tests pour s'assurer que l'on couvre les cas suivants : les piles sont différentes, mais de même taille ; les piles sont identiques.
7. Écrire une fonction `ordre_faro` qui prend en paramètres un entier `n` pair et qui renvoie le plus petit nombre de répétitions du mélange faro pour qu'un jeu de n cartes soit remis dans son ordre initial.