Exercice 1

jeu de Dominos

1.1 version classique

Le jeu de *Dominos* est un jeu très simple, ou, pour gagner, il faut être le premier joueur à avoir posé tous ses dominos. Une fois le premier domino placé sur la table, le joueur suivant doit à son tour poser un domino ayant le même nombre de points sur au moins un côté du domino précédemment posé.

Un domino est constitué de côtés, droite/gauche ou haut/bas selon comment la pièce sera disposée.

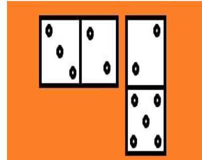

Figure 1 – début de partie

La disposition importe peu : il faut que la chaine reste ouverte.

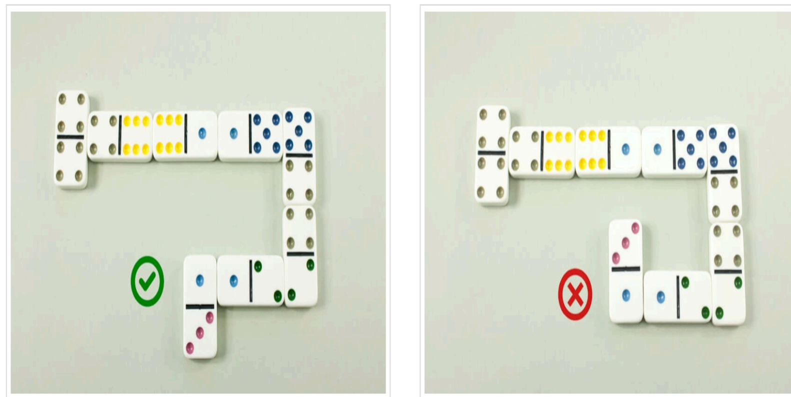

Figure 2 – Exemple de disposition juste (à gauche) et fausse (à droite)

On utilise la définition de classes suivantes :

```
class Domino:
    def __init__(self,val1,val2):
        self.val1 = val1
        self.val2 = val2
        self.suiv = None


class Partie:
    def __init__(self,first):
        self.first = first

    def last(self):
        M = self.first
        while not M.suiv is None:
            M = M.suiv
        return '{}:{}'.format(M.val1,M.val2)

```

```python
    def atteindre_domi(self,val1,val2):
        D = self.first
        while not D.suiv is None and (D.val1,D.val2) != (val1,val2):
            D = D.suiv
        if (D.val1,D.val2) == (val1,val2):
            return D
        else:
            return self.first

    def inserer(self,D_place,D_a_inserer):
            #à completer

    def __repr__(self):
        M = self.first
        s = '{}:{} '.format(M.val1,M.val2)
            # à completer
        return s
```

Qu a. On cherche à représenter la partie de l'image de gauche (voir plus haut). Les dominos seront instanciés à l'aide des noms D1, D2, D3, ... Ecrire les instructions qui instancient tous les dominos de la partie, avec, pour chacun, leurs valeurs et le domino suivant.

Qu b. Ecrire l'instruction qui doit créer l'objet `partie1` à partir de ce plateau de jeu. (classe `Partie`)

Qu c. Compléter la méthode de classe `__repr__` qui surcharge la fonction `print`

Qu d. Commenter la méthode de classe `atteindre_domi`. A quoi sert-elle ? Quelle est sa complexité asymtotique ?

Qu e. Imaginons que l'état de la partie soit celui-ci :

```python
print(partie1)
4:4 => 4:6 => 6:1 => 1:5 => 5:4 => 4:2 => 2:1 => 1:3
```

Compléter la méthode de classe `inserer` qui permet d'insérer un domino (double) dans la chaine de dominos à partir des instructions suivantes :

```python
D = partie1.atteindre_domi(4,2)
D10 = Domino(2,2)
partie1.inserer(D,D10)
```

Le nouvel état de la partie devrait alors être :

```python
print(partie1)
4:4 => 4:6 => 6:1 => 1:5 => 5:4 => 4:2 => 2:2 => 2:1 => 1:3
```

1.2 Variante du *Train mexicain*

Le but du jeu est pour un joueur de jouer tous les dominos de sa main sur une ou plusieurs chaînes, ou “trains”, émanant d'un hub central ou “station”. Train mexicain

Les différentes chaînes sont placées comme bifurcation lorsque l'on a posé un domino double. Le pliage du train se fait d'un angle qui est souvent différent de 90°, pour ne pas chevaucher avec les autres chaînes. On supposera que l'on ne peut poser qu'une seule bifurcation sur un même domino.

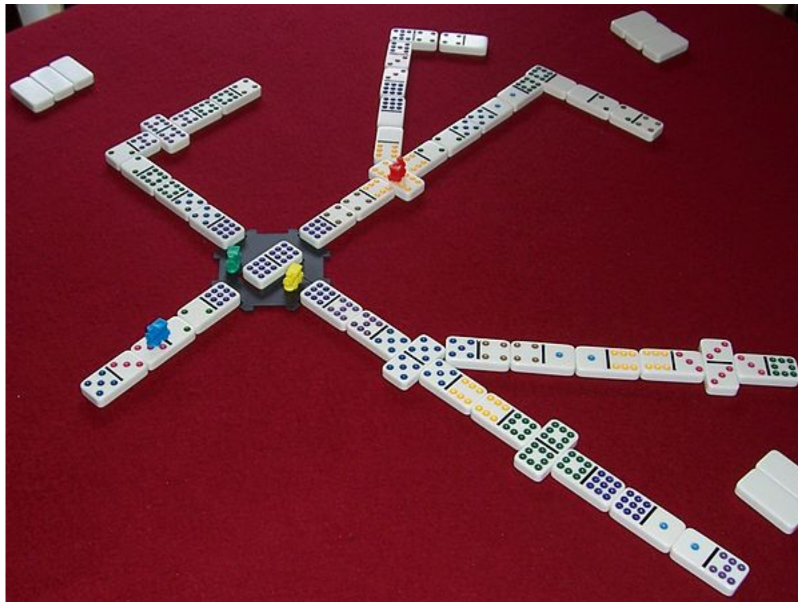

Figure 3 – variante du domino : le train mexicain - wikipedia

Qu f. Modifier la classe `Domino` pour l'adapter à ce nouveau jeu.

Qu g. On souhaite atteindre le domino (8 :8) qui se trouve dans la diagonale inférieure droite sur le schéma. On voudrait realiser une bifurcation, en posant le domino (8 :2). Quelles instructions faut-il ajouter à la partie pour réaliser ceci ?

Qu h. La fonction `__repr__` va-t-elle faire la liste de tous les dominos disposés sur le plateau ? Sinon, comment faudrait-il adapter l'algorithme ? (*on ne demande pas d'expliquer l'algorithme en détail, seulement le principe général de cet algorithme*).

Exercice 2

Bac 2022 Polynesie : Exercice 4

Cet exercice traite du thème « structures de données », et principalement des piles.

La classe Pile utilisée dans cet exercice est implémentée en utilisant des listes Python et propose quatre éléments d'interface :

- Un constructeur qui permet de créer une pile vide, représentée par `[]` ;
- La méthode `est_vide()` qui renvoie `True` si l'objet est une pile ne contenant aucun élément, et `False` sinon ;
- La méthode `empiler` qui prend un objet quelconque en paramètre et ajoute cet objet au sommet de la pile. Dans la représentation de la pile dans la console, cet objet apparaît à droite des autres éléments de la pile ;
- La méthode `depiler` qui renvoie l'objet présent au sommet de la pile et le retire de la pile.

Exemples :

```
>>> mapile = Pile()
>>> mapile.empiler(2)
>>> mapile
[2]
>>> mapile.empiler(3)
```

```
6  >>> mapile.empiler(50)
7  >>> mapile
8  [2, 3, 50]
9  >>> mapile.depiler()
10 50
11 >>> mapile
12 [2, 3]
```

La méthode `est_triee` ci-dessous renvoie `True` si, en dépilant tous les éléments, ils sont traités dans l'ordre croissant, et `False` sinon.

```
1 def est_triee(self):
2   if not self.est_vide() :
3     e1 = self.depiler()
4     while not self.est_vide():
5       e2 = self.depiler()
6       if e1 ... e2 :
7         return False
8       e1 = ...
9     return True
```

2.1 Question 1

Recopier sur la copie les lignes 6 et 8 en complétant les points de suspension.

On créé dans la console la pile A représentée par `[1, 2, 3, 4]`

2.2 Question 2

A. Donner la valeur renvoyée par l'appel `A.est_triee()`.

B. Donner le contenu de la pile A après l'exécution de cette instruction.

On souhaite maintenant écrire le code d'une méthode `depileMax`d'une pile non vide ne contenant que des nombres entiers et renvoyant le plus grand élément de cette pile en le retirant de la pile.

Après l'exécution de `p.depileMax()`, le nombre d'éléments de la pile p diminue donc de 1.

```
1  def depileMax(self):
2    assert not self.est_vide(), "Pile vide"
3    q = Pile()
4    maxi = self.depiler()
5    while not self.est_vide() :
6      elt = self.depiler()
7      if maxi < elt :
8        q.empiler(maxi)
9        maxi = ...
10     else :
11       ...
12   while not q.est_vide():
13     self.empiler(q.depiler())
14   return maxi
```

2.3 Question 3

Recopier sur la copie les lignes 9 et 11 en complétant les points de suspension.

On créé la pile `B` représentée par `[9, -7, 8, 12, 4]` et on effectue l'appel `B.depileMax()`.

2.4 Question 4

A. Donner le contenu des piles `B` et `q` à la fin de chaque itération de la boucle `while` de la ligne 5.

B. Donner le contenu des piles `B` et `q` avant l'exécution de la ligne 14.

C. Donner un exemple de pile qui montre que l'ordre des éléments restants n'est pas préservé après l'exécution de `depileMax`.

On donne le code de la méthode `traiter()` :

```
def traiter(self):
q = Pile()
while not self.est_vide():
q.empiler(self.depileMax())
while not q.est_vide():
self.empiler(q.depiler())
```

2.5 Question 5

A. Donner les contenus successifs des piles `B` et `q`

- avant la ligne 3,
- avant la ligne 5,
- à la fin de l'exécution de la fonction traiter lorsque la fonction traiter est appliquée sur la pile B contenant `[1, 6, 4, 3, 7, 2]`.

B. Expliquer le traitement effectué par cette méthode.

Exercice 3

Corrections

3.1 Jeu de dominos

3.1.1 Jeu classique

```
class Partie:
    def __init__(self,first):
        self.first = first

    def last(self):
        M = self.first
        while not M.suiv is None:
            M = M.suiv
        return '{}:{}'.format(M.val1,M.val2)

    def atteindre_domi(self,val1,val2):
        D = self.first
        while not D.suiv is None and (D.val1,D.val2) != (val1,val2):
            D = D.suiv
        if (D.val1,D.val2) == (val1,val2):
            return D
        else:
            return self.first

    def inserer(self,D_place,D_a_inserer):
        D_a_inserer.suiv = D_place.suiv
        D_place.suiv = D_a_inserer

    def doubles(self):
        n = 0
        M = self.first
        if M.val1 == M.val2 : n+=1
        while not M.suiv is None:
            M = M.suiv
            if M.val1 == M.val2 : n+=1
        return n

    def __repr__(self):
        M = self.first
        s = '{}:{} '.format(M.val1,M.val2)
        while not M.suiv is None:
            M = M.suiv
            s += '=> {}:{} '.format(M.val1,M.val2)
        return s

D1 = Domino(4,4)
D2 = Domino(4,6)
D3 = Domino(6,1)
D4 = Domino(1,5)
D5 = Domino(5,4)
D6 = Domino(4,2)
```

```
D7 = Domino(2,1)
D8 = Domino(1,3)
D1.suiv = D2
D2.suiv = D3
D3.suiv = D4
D4.suiv = D5
D5.suiv = D6
D6.suiv = D7
D7.suiv = D8
partie1 = Partie(D1)
>>> partie1.doubles()
1
```