1 TD 5 : Carré magique minuté

(correction page ??)

Abordé lors de cette séance	
programmation	classe
algorithme	méthode de construction d'un carré magique

Un carré magique est un carré 3x3 dont chaque case contient un nombre entier et pour lequel les sommes sur chacune des lignes, chacune des colonnes, chacune des diagonales sont égales, soit huit sommes égales. Le plus simple des carrés magiques est celui dont tous les chiffres sont égaux même si un véritable carré magique contient des nombres a priori tous différents [1].

1	1	1
1	1	1
1	1	1

2	7	6
9	5	1
4	3	8

L'autre objectif de ce TD est de comprendre ce que sont les classes [2] au travers de l'exemple des carrés magiques. Le concept de classe est en pratique un moyen de créer ses propres types. Il existe des types prédéfinis en *Python* tels que les listes, les dictionnaires, les entiers... Par défaut, il n'existe pas de type matrice ni de type carré magique et mais les classes permettent de créer facilement ces types qui ne font pas partie du langage.

Première demi-heure : constructeur .

Une classe se crée invariablement de la même manière avec le mot-clé `class` et en ajoutant ce qu'on appelle un constructeur : c'est une fonction attachée à la classe et qui crée la classe.

```
class CarreMagique :
    def __init__ (self, ...) :
        ...
```

Dans le cas d'un carré magique cela pourrait donner :

```
class CarreMagique :
    def __init__ (self, a,b,c, d,e,f, g,h,i) :
        self.carre = [ [ a,b,c ], [ d,e,f ], [ g,h,i ] ]
```

Chaque fois que l'on écrit `self.` < `identifiant` >, c'est une façon d'attacher une variable à la classe. Les lignes précédentes définissent une classe, les lignes suivantes définissent une variable de type `CarreMagique`. On utilise aussi le terme d'*instance*.

```
c = CarreMagique ( 1,2,3, 4,5,6, 7,8,9 )
```

1. http://fr.wikipedia.org/wiki/Carr%C3%A9_magique_(math%C3%A9matiques)
2. http://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet

Pour afficher la variable ou *membre* `carre` de la classe `CarreMagique`, on écrit :

```
print c.carre
```

1) Modifier la classe de telle sorte que le constructeur accepte une liste plutôt que neuf paramètres. On souhaite, plutôt que de rentrer neuf paramètres à la suite, pouvoir les définir dans une liste. On souhaite par exemple pouvoir écrire cela :

```
l = range (1,10)
c = CarreMagique ( l )
```

2) Qu'affiche le programme suivant :

```
class CarreMagique :
    def __init__ (self, a,b,c, d,e,f, g,h,i) :
        self.carre = [ [ a,b,c ], [ d,e,f ], [ g,h,i ] ]

c = CarreMagique (2,7,6, 9,5,1, 4,3,8)
print c
```

Et celui-là :

```
class CarreMagique :
    def __init__ (self, a,b,c, d,e,f, g,h,i) :
        self.carre = [ [ a,b,c ], [ d,e,f ], [ g,h,i ] ]
    def __str__ (self) :
        return "carre magique : " + str (self.carre)

c = CarreMagique (2,7,6, 9,5,1, 4,3,8)
print c
```

Comment modifier le programme pour que l'instruction `print c` affiche le carré sur plusieurs lignes ?

Une indication avant de commencer :

```
s = "un deux"
print s
s = "un \n deux"  # ajout du caractère \n
print s
```

Seconde demi-heure : méthodes .

Une classe inclut des variables ou *membres*, des fonctions ou *méthodes*. Ces fonctions ont toutes en commun de s'appliquer à la classe. Le constructeur est une méthode. L'exemple suivant reprend la classe `CarreMagique` et ajouter une méthode qui compte la somme des nombres sur une ligne :

```
class CarreMagique :
    def __init__ (self, a,b,c, d,e,f, g,h,i) :
        self.carre = [ [ a,b,c ], [ d,e,f ], [ g,h,i ] ]
    def __str__ (self) :
        return "carre magique : " + str (self.carre)

    def somme_ligne (self, i) :                # ajout
        return sum ( self.carre [i] )          # ajout

c = CarreMagique (2,7,6, 9,5,1, 4,3,8)
print c.somme_ligne (0)   # somme des nombres sur la première ligne
```

3) Ajouter une méthode qui fait la somme des nombres sur une colonne.

4) Ajouter une méthode qui fait la somme des nombres sur une diagonale.

5) Ajouter une méthode qui détermine si un carré est magique ou non en complétant la méthode suivante :

```
class CarreMagique :
    ...

    def carre_est_magique (self) :
        li   = [ self.somme_ligne (i) for i in xrange (0,3) ]
        lj   = [ self.somme_colonne (j) for j in xrange (0,3) ]
        diag = [ self.somme_diagonal (k) for k in xrange (0,2) ]
        tout = li + lj + diag
        tout.sort ()
        return ..................
```

Troisième demi-heure : le damier crénelé .

Tout d'abord une astuce. Il n'existe pas d'élément d'indice $(1, -1)$ dans une matrice mais cela est tout-à-fait possible avec un dictionnaire :

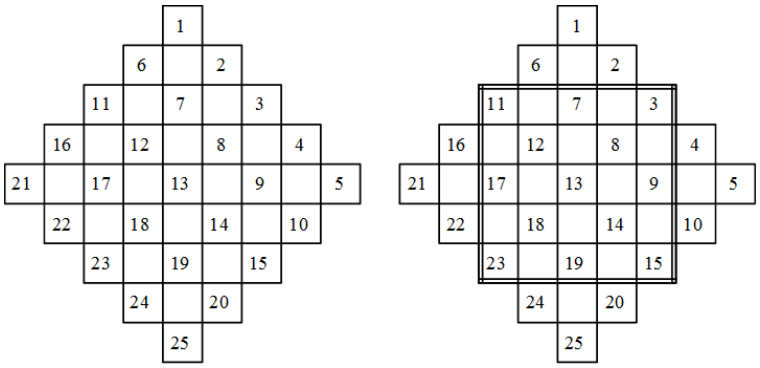
```
d = {}
d [ 1,-1 ] = 1
```

Seconde astuce, l'opérateur modulo ou % :

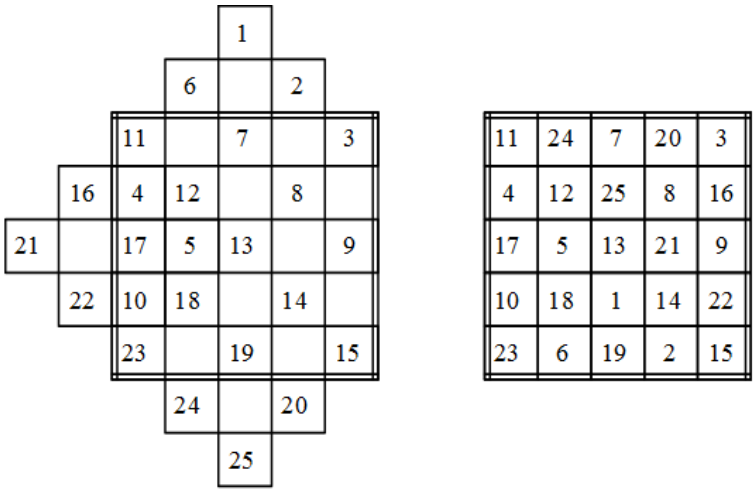
```
for i in xrange (0,9) :
    print i % 3           # affiche 0,1,2, 0,1,2, 0,1,2
```

On souhaite implémenter la méthode du damier crénelé [3]. Les illustrations suivantes sont extraites de Wikipédia :

3. http://fr.wikipedia.org/wiki/Carr%C3%A9_magique_(math%C3%A9matiques)#M.C3.A9thode_du_damier_cr.C3.A9nel.C3.A9



Premières étapes de construction d'un carré magique d'ordre 5. Chaque diagonale allant de gauche à droite comporte un entier unique en ordre croissant. Ensuite, le contour du carré magique final est esquissé.



Dernières étapes de construction d'un carré magique d'ordre 5. Chaque case qui se trouve en-dehors du contour est "glissé" de 5 places en direction du centre. Une fois les déplacements effectués, le carré magique final est complet.

6) Ecrire une méthode de la classe `CarreMagique` qui positionne les nombres de la première étape du damier crénelé dans un dictionnaire :

```
class CarreMagique :
    ...
    def etape_damier_1 (self, dimension) :
        d = { }
        ...
        return d
```

Quatrième demi-heure : le damier crénelé, seconde partie

7) Ecrire une méthode de la classe `CarreMagique` qui implémente la seconde étape du damier crénelé.

```
class CarreMagique :
    ...
    def damier_crenele (self, dimension) :
        d = self.etape_damier_1 ( dimension )
        self.carre = [ [ 0 for i in xrange (0, ... ) ] for j in xrange (0, ... ) ]
        ...

c = CarreMagique ([])
c.damier_crenele ()
print c
print c.carre_est_magique ()
```

Si votre algorithme est correct, la dernière ligne devrait afficher `True`.

8) Créer un carré magique selon cette méthode pour toutes les dimensions de 3 à 10. Sont-ils tous magiques ?

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

9) Ecrire une fonction qui permute la première et la dernière colonne du carré magique créé avec la méthode du damier crénelé. Ce carré est-il magique ?

10) Déterminer une condition nécessaire qui permette de conclure que le magique sera magique si on permute la première et la dernière colonne ?

11) Démontrez que cette condition est toujours vérifiée lorsque le carré est construit selon la méthode du damier crénelé ?

12) Est-ce que cela vous inspire une autre permutation sur les colonnes qui aboutit à un nouveau carré magique ? Vérifier le en l'implémentant.