1 TD 5 : Classes, méthodes, attributs, opérateurs

(correction page ??)

Abordé lors de cette séance	
programmation	Classes, méthodes, attributs, opérateurs
algorithme	carré magique

La plupart du temps, les classes ne sont pas indispensables et l'expérience montre que la plupart des élèves choisissent de s'en dispenser lors de la réalisation de leur projet. Pourquoi les aborder ? Pour plusieurs raisons :

1. Un langage de programmation dispose des types standards : entier, réel, chaînes de caractères, tableaux. Hors de cette liste, il n'y a rien à moins de pouvoir le créer, c'est ce que permettent les classes.
2. Elles rendent le programme plus lisible : tous les projets conséquents utilisent les classes. C'est un peu comme si on disposait d'un vocabulaire enrich pour décrire un programme.

Prenons l'exemple d'un jeu de cartes. Une *carte* désigne l'objet physique : sa couleur, son numéro (ou figure), son atout... Dans un jeu, chaque carte vaut un certain nombre de points, elle est plus ou moins forte qu'une autre... A quoi correspondrait le mieux une carte de tarot pour vous ?

option 1	option 2	option 3
– une couleur – un numéro – un atout – tous trois indépendants	– une couleur – un numéro – un atout – un ensemble de configurations possibles	– une couleur – un numéro – un atout – un ensemble de configurations possibles – un nombre de points

Définir une classe dans votre programme vous permet de définir précisément ce que le concept signifie selon trois aspects :

1. les attributs : les données que la classe considère comme un tout,
2. les méthodes : des fonctions opérant sur les attributs,
3. les opérateurs : des fonctions spécifiques pour définir ce qu'est une addition, une soustraction...

Première demi-heure : Classes .

On veut définir une classe `Point` équivalent à un point dans le plan.

```
class Point :
    def __init__ (self, x,y) :
        self.x = x
        self.y = y
```

De cette façon, on peut définir un point de coordonnées (4,5) :

```
p = Point (4,5)
```

Vocabulaire :
- `p` est une instance de la classe `Point`. Il n'existe qu'une classe `Point` mais autant d'instances qu'on veut. Dans notre cas : instance = variable de type `Point`.
- `__init__` : est un constructeur. Il définit ce que le langage doit faire lorsqu'on crée une instance.
- `self.x`, `self.y` : sont des attributs (ou des variables à l'intérieur d'une classe).

La variable `self` peut être remplacée par n'importe quoi. C'est une convention de langage pour désigner l'instance manipulée :

```
class Point :
    def __init__ (moi, x,y) :
        moi.x = x
        moi.y = y

p1 = Point(4,5)  # moi désigne p1 ici
p2 = Point(6,7)  # moi désigne p2 ici
```

Si on utilise `print` :

```
print (p1)
# affiche <__main__.Point object at 0x0288E470>
```

Pour éviter cela, deux moyens :
- afficher directement `x` et `y` :

  ```
  print (p1.x, p1.y)
  ```

  Ou ajouter l'opérateur `__str__` :

  ```
  class Point :
      def __init__ (moi, x,y) :
          moi.x = x
          moi.y = y
      def __str__(moi):
          return "point: ({0},{1})".format(moi.x, moi.y)
  p = Point(4,5)
  print (p)  # affiche point: (4,5)
  ```

On peut également définir l'addition :

```
class Point :
    def __init__ (moi, x,y) :
        moi.x = x
        moi.y = y
    def __str__(moi):
        return "{0},{1}".format(moi.x, moi.y)
    def __add__(moi, autre_point) :
        return Point(moi.x + autre_point.x, moi.y + autre_point.y)
p = Point(4,5)
print (p + p)   # affiche 8,10
```

On peut redéfinir tous les opérateurs numériques [1] mais il en existe beaucoup d'autres comme l'opérateur [] [2].

Seconde demi-heure : Carré magique

Exercice

On souhaite appliqer ce qu'on vient de voir pour définir un carré magique qui contient neuf chiffres rangés dans un tableau à deux dimensions. On ajoutera l'opérateur `__str__`.

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

Troisième demi-heure : Méthodes

Une méthode est une fonction rattachée à une classe et qui s'applique aux données de la classe et celles envoyées en paramètres :

```
class Point :
    def __init__ (moi, x,y) :
        moi.x = x
        moi.y = y
    def norm(moi) :
        return (moi.x**2 + moi.y**2) ** 0.5
p = Point(4,5)
print (p.norm())
```

Avec un paramètre :

```
class Point :
    def __init__ (moi, x,y) :
        moi.x = x
        moi.y = y
    def norm(moi, lx = 2) :
        return (abs(moi.x)**lx + abs(moi.y)**lx) ** (1.0 / lx)
print (p.norm(1))
print (p.norm(2))
print (p.norm(3))
print (p.norm(100))
```

On peut bien sûr appeler une méthode de la classe depuis une autre méthode de la même classe :

```
class Point :
    def __init__ (moi, x,y) :
```

1. http://docs.python.org/3.3/reference/datamodel.html#emulating-numeric-types
2. http://docs.python.org/3.3/reference/datamodel.html#emulating-container-types

```
        moi.x = x
        moi.y = y
    def norm(moi, lx = 2) :
        if lx == 0 : return moi.est_nul()
        else : return (abs(moi.x)**lx + abs(moi.y)**lx) ** (1.0 / lx)
    def est_nul(moi):
        return  moi.x == 0 and moi.y == 0
```

Exercice à faire à trois :

Ajouter trois méthodes à la classe carré magique :

1. une méthode qui compte la somme des nombres sur chaque ligne, colonne, diagonale
2. une méthode qui dit si tous les chiffres du carrés sont uniques,
3. une méthode qui dit si le carré est magique.

Quatrième demi-heure : Trouver tous les carrés magiques .

On écrit une fonction qui trouve tous les carrés magiques composés des chiffres 1 à 9. On suit la méthode suivante :

1. Considérer un ensemble de carrés qui inclut l'ensemble des carrés magiques
2. Parcourir cet ensemble et mémoriser dans une liste ceux qui sont magiques.

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

La vitesse de la fonction dépend de l'ensemble de départ qui peut contenir 9^9 possibilités, ou encore $9!$. Peut-on faire plus rapide encore ?