Initiation à la programmation en Python

Damien Vergnaud

École Normale Supérieure

4 mai 2011

L'approche orientée-objet : Objectifs

La programmation orientée objet (**POO**) est un paradigme de programmation informatique qui consiste en la *définition* et l'*assemblage* de briques logicielles appelées objet

Elle vise à produire des programmes possédant des bonnes qualités de modularité.

Objectifs :

- développer une partie d'un programme sans qu'il soit nécessaire de connaître les détails internes des autres parties;
- apporter des modifications locales à un module, sans que cela affecte le reste du programme;
- réutiliser des fragments de code développés dans un cadre différent.

L'approche orientée-objet: Principes

- Les **objets** sont des structures de données présentes dans la mémoire de l'ordinateur au cours de l'exécution d'un programme ;
- Un objet combine des **données** et du **code exécutable**. Ce dernier correspond à un certain nombre d'opérations que l'objet est capable d'effectuer ;
- Un objet possède une interface et une implémentation. **Seule l'interface** est accessible par l'environnement de l'objet;
- Les données gérées par un objet sont retenues dans un ensemble de variables internes à cet objet;
- Les opérations réalisables par un objet sont représentées par un ensemble de méthodes;

L'approche orientée-objet: Principes

- L'interface d'un objet caractérise chaque méthode par un en-tête, composé
  - d'un **nom**,
  - d'une **liste d'arguments**,
  - du **type** d'une valeur de retour éventuelle;

- L'implémentation d'un objet regroupe
  - ses **variables**,
  - le **corps de ses méthodes**, formées d'instructions exécutables;

- L'exécution d'un programme est vue comme une suite d'interactions entre objets. Ces interactions prennent la forme de messages.

  Un message émis d'un objet A à un objet B représente une requête d'effectuer une opération.

  La réception d'un message par un objet provoque l'invocation d'une méthode.

L'approche orientée-objet: Principes

Les instructions exécutées lors de l'invocation d'une méthode peuvent

- **manipuler** les données de l'objet;
- **envoyer** des messages;
- **créer** de nouveaux objets.

Définition d'une classe élémentaire

- Pour créer une nouvelle classe d'objets Python, on utilise l'instruction `class`.

```
>>> class Point:
        "Définition d'un point mathématique"
```

Les définitions de classes peuvent être situées n'importe où dans un programme (en général au début ou dans un module)

- Une fois une classe définie, nous pouvons créer des objets de ce type :

```
>>> p = Point()
```

- Si la première ligne suivant l'instruction `class` est une chaîne de caractères, celle-ci est incorporée dans un dispositif de documentation des classes :

```
>>> print p.__doc__
Définition d'un point mathématique
```

Attributs (ou variables) d'instance

- L'objet que nous venons de créer est une coquille vide.
  Nous pouvons ajouter des composants à cet objet par simple assignation, en utilisant le système de qualification des noms par points :

```
>>> p.x = 3.0
>>> p.y = 4.0
```

- On peut utiliser les attributs d'un objet dans n'importe quelle expression

```
>>> print p.x
3.0
>>> print p.x**2 + p.y**2
25.0
```

Égalité d'objets

```
>>> p1=Point() ; p1.x=5 ; p1.y=3
>>> p2=Point() ; p2.x=5 ; p2.y=3
>>> print p1==p2
False
```

```
>>> p1=Point() ; p1.x=5 ; p1.y=3
>>> p2=p1
>>> print p1==p2
True
```

Objets composés d'objets

```
>>> class Rectangle:
        "Définition d'une classe de rectangles"
```

```
>>> boite = Rectangle()
>>> boite.largeur = 50.0
>>> boite.hauteur = 35.0
```

```
>>> boite.coin = Point()
>>> boite.coin.x = 12.0
>>> boite.coin.y = 27.0
```

Objets comme arguments/valeurs de retours d'une fonction

- Les fonctions peuvent utiliser des objets comme paramètres

```
>>> def affiche_point(p):
        print "coord. horiz. =", p.x, "coord. vert. =", p.y
>>> affiche_point(p)
coord. horiz. = 3.0 coord. vert. = 4.0
```

- Les fonctions peuvent également transmettre une instance comme valeur de retour

```
>>> def trouveCentre(box):
        p = Point()
        p.x = box.coin.x + box.largeur/2.0
        p.y = box.coin.y + box.hauteur/2.0
        return p
```

Un autre exemple

```
>>> class Time:
        "Définition d'une classe temporelle"
```

```
>>> instant = Time()
>>> instant.h = 11
>>> instant.m = 34
>>> instant.s = 25
```

```
>>> def affiche_heure(t):
        print str(t.h) + ":" + str(t.m) + ":" + str(t.s)
>>> print affiche_heure(instant)
11:34:25
```

Définition d'une méthode

- On peut vouloir encapsuler une fonction dans une classe pour qu'elle soit toujours disponible pour tous les objets de la classe
- Une fonction qui est ainsi encapsulée dans une classe s'appelle une méthode.
- La définition d'une méthode est placée à l'**intérieur** de la définition d'une classe
- Le premier paramètre d'une méthode doit être une référence d'**instance** (par convention on lui donne le nom `self`).

```
>>> class Time:
        "Nouvelle classe temporelle"
        def affiche_heure(self):
            print str(self.h) + ":" + str(self.m) \
                  + ":" + str(self.s)
```

Essai de la méthode dans une instance

```
>>> maintenant = Time()
```

```
>>> maintenant.affiche_heure()
AttributeError: 'Time' instance has no attribute 'h'
```

```
>>> maintenant.h = 13
>>> maintenant.m = 34
>>> maintenant.s = 21
>>> maintenant.affiche_heure()
13:34:21
```

La méthode constructeur

- L'exemple précédent montre que les variables d'instance doivent être prédéfinies elles aussi à l'intérieur de la classe
- Une méthode constructeur est une méthode qui est exécutée automatiquement lorsque l'on instancie un nouvel objet à partir de la classe.
- Elle doit s'appeler `__init__`

```
>>> class Time:
        "Encore une nouvelle classe temporelle"
        def __init__(self):
            self.h, self.m, self.s =0,0,0

        def affiche_heure(self):
            print str(self.h) + ":" + str(self.m) \
                 + ":" + str(self.s)

>>> tstart = Time()
>>> tstart.affiche_heure()
0:0:0
```

La méthode constructeur

```
def __init__(self, hh =0, mm =0, ss =0):
            self.h = hh
            self.m = mm
            self.s = ss
```

```
>>> debut = Time(18)
>>> debut.affiche_heure()
18:0:0
>>> fin = Time(20,10)
>>> fin.affiche_heure()
20:10:0
```

Espaces de noms des classes et instances

```
>>> class Espaces:
        aa = 33
        def affiche(self):
            print aa, Espaces.aa, self.aa

>>> aa = 12
>>> essai = Espaces()
>>> essai.aa = 67
>>> essai.affiche()
12 33 67
>>> print aa, Espaces.aa, essai.aa
12 33 67
```

Héritage

- L'héritage permet de se servir d'une classe existante pour en creer une nouvelle avec des fonctionnalités différentes ou supplémentaires.

```
class Rectangle:
    def __init__(self, longueur=30, largeur=15):
        self.L, self.l, self.nom = longueur, largeur, "rectangle"
class Carre(Rectangle):
    def __init__(self, cote=10):
        Rectangle.__init__(self, cote, cote)
        self.nom = "carré"

r = Rectangle()
print r.nom # 'rectangle'
c = Carre()
print c.nom # 'carré'
```

Modules contenant des bibliothèques de classes

```
class Rectangle:
    "Classe de rectangles"
    def __init__(self, longueur =30, largeur =15):
        self.L,self.l  = longueur, largeur
        self.nom ="rectangle"

    def perimetre(self):
        return "(%s + %s) * 2 = %s" % (self.L, self.l, (self.L + self.l)*2)
    def surface(self):
        return "%s * %s = %s" % (self.L, self.l, self.L*self.l)

    def mesures(self):
        print "Un %s de %s sur %s" % (self.nom, self.L, self.l)
        print "a une surface de %s" % (self.surface(),)
        print "et un périmètre de %s\n" % (self.perimetre(),)

class Carre(Rectangle):
    "Classe de carrés"
    def __init__(self, cote =10):
        Rectangle.__init__(self, cote, cote)
        self.nom ="carré"
```

Modules contenant des bibliothèques de classes

```
>>> import formes
>>> f1 = formes.Rectangle(27, 12)
>>> f1.mesures()
Un rectangle de 27 sur 12
a une surface de 27 * 12 = 324
et un périmètre de (27 + 12) * 2 = 78

>>> f2 = formes.Carre(13)
>>> f2.mesures()
Un carré de 13 sur 13
a une surface de 13 * 13 = 169
et un périmètre de (13 + 13) * 2 = 52
```