Introduction à la programmation orientée-objet

Yannick Prié
UFR Informatique – Université Lyon 1

Master SIB M1 – 2005-2006
UE1.2 *Organisation de l'information documentaire et programmation*

Objectifs du (petit) module

- Découvrir ce qu'est la programmation orientée-objet
  - pour des étudiants qui ont déjà appris la programmation impérative
  - Cours → 2 x 1h30
  - TP → 2 x 1h30 + 1 x 3h
- Découvrir le langage Python

Organisation du module

- CM1
  - Objets / classes
  - Attributs
  - Méthodes
  - Collaboration d'objets
- TP1
  - Prise en main de l'environnement et du langage python (syntaxe python), manipulation d'objets
- CM2
  - Hiérarchie de classe et héritage
  - Relations entre classes
  - Conception objet
- TP2
  - Réalisation d'un petit programme impliquant plusieurs objets et classes.
- TP3
  - Conception et réalisation d'un programme un peu plus ambitieux

Objets

- Objets du monde
  - Objets « concrets », plus ou moins coopératifs : cette pierre, ma télévision, ta voiture
  - Objets « abstraits », « conceptuels » : mon compte bancaire, le langage de programmation que j'utilise
- Classes d'objets
  - les pierres, les télévisions, les langages de programmation, les comptes bancaires, etc.
- Objets et classes d'objets sont toujours considérés dans un contexte

Abstraction

- Objets
  - tout ce qui nous permet de réfléchir, parler, manipuler des concepts du domaine, avec
    - un certain nombre de propriétés les caractérisant
    - un certain nombre de comportements connus
    - des classes d'objets avec des propriétés et des comportements similaires
- Abstraction
  - passage du particulier au général
  - « abstraire » des propriétés, comportements

En informatique

- Programme classique
  - structures de données (tableau, arbre, etc.)
  - opérations sur ces structures de données (fonctions)
- Difficultés
  - faire *évoluer* structures de données et fonctions en même temps
  - *réutiliser* des structures/fonctions en les spécialisant
  - ...

Idée objet

- Regrouper dans un composant
  - des caractéristiques qui concernent une entité informatique
    - structure de données
    - ensemble d'attributs
      - variables avec nom, type, valeur
  - les opérations liées à cette entité
    - ensemble de fonctions
    - appelées *méthodes*
      - avec : nom, valeur de retour, paramètres

Objet informatique

- Etat

  Ce qu'est l'objet à un instant donné
- \+ Comportement

  Comment l'objet réagit aux sollicitations
- \+ ...

Etat d'un objet

- Ensemble des valeurs des attributs de l'objet à un instant donné
- L'état d'un objet change pendant sa vie

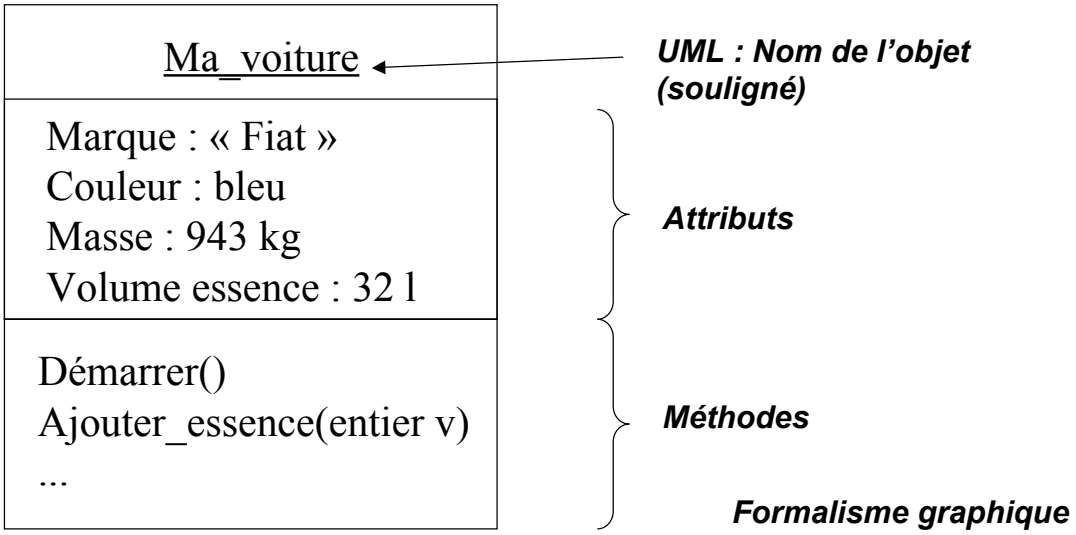
Ma_voiture
Marque : « Fiat » Couleur : bleu Masse : 943 kg Volume essence : 32 l

Ma_voiture
Marque : « Fiat » Couleur : bleu Masse : 943 kg Volume essence : 31 l

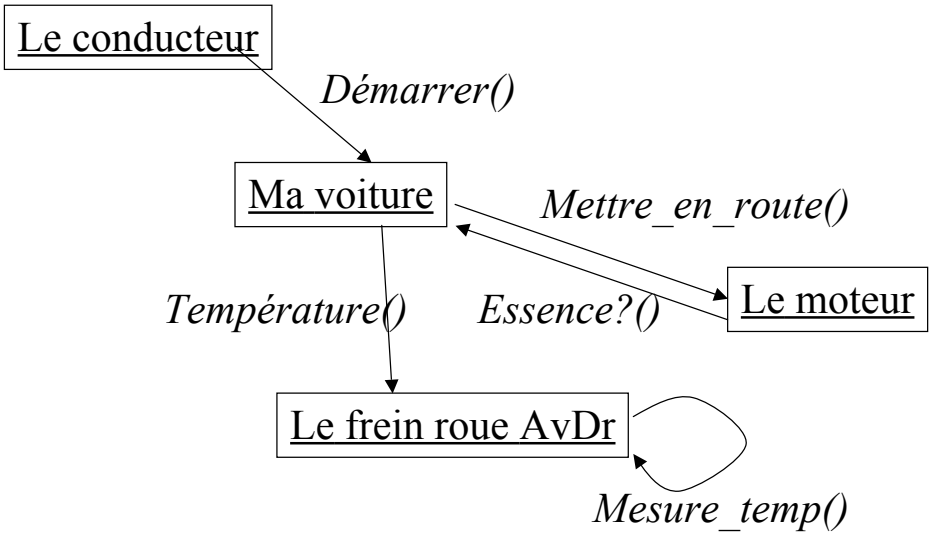
Comportement d'un objet

- Actions et réactions possibles
  - ensemble d'*opérations / méthodes*
  - exemple automobile
    - *démarrer, rouler, stopper, ajouter_essence*
- Stimulation
  - demander à un objet d'effectuer une méthode = lui envoyer un message
  - Exemple
    - ok = ma_voiture.démarrer()
    - qtte = ma_voiture.ajouter_essence(15 l.)
- L'état dépend des opérations effectuées
  - Ex. ma_voiture.
- Les opérations dépendent de l'état courant
  - Ex. ma_voiture.démarrer() ne marchera pas si ma_voiture.volume_essence == 0

Représentation d'un objet



Messages et collaboration d'objets



Accès aux attributs/méthodes

- Accès depuis un autre objet
  - Attribut/méthode publics
    - tout objet peut y accéder
  - Attribut/méthode privés
    - aucun autre objet ne peut y accéder
    - seul l'objet lui-même peut utiliser ses attributs et méthodes
      - comme un programme « indépendant »
  - Attribut/méthode protégé
    - accès limité

Objet informatique

- Etat

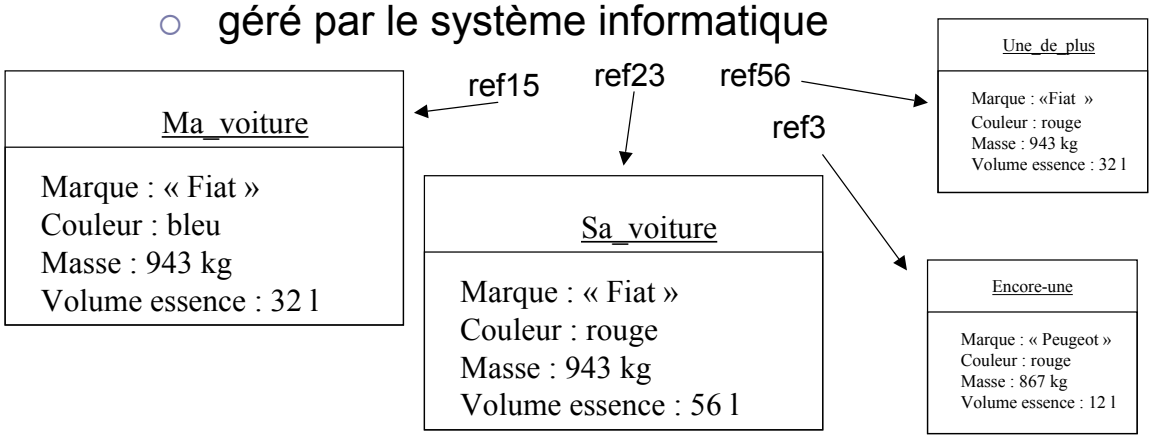
  Ce qu'est l'objet à un instant donné
- \+ Comportement

  Comment l'objet réagit aux sollicitations
- \+ Identité

  Ce qui identifie l'objet

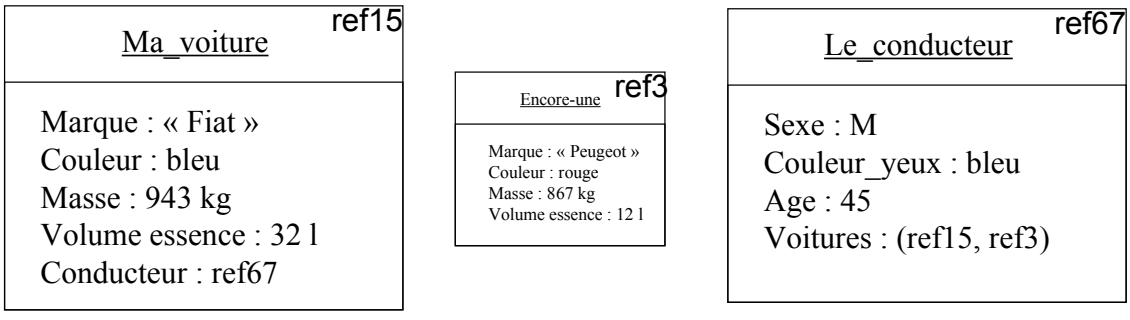
Identité d'un objet

- Existence propre de l'objet
  - identification non ambiguë
  - indépendante de l'état
  - géré par le système informatique



Liens entre objets

- Pour pouvoir envoyer un message à un objet, il faut le « connaître »
  - Ex. l'objet *Le_conducteur* connaît l'objet *Ma_voiture*
- Connaître un objet revient à avoir une référence qui lui correspond
  - Certains attributs d'un objet sont des références vers d'autres objets



En bref

- Cohérence interne des objets
  - données + traitements
- Faible couplage entre l'objet et l'environnement
  - envoi de messages
- Insertion dans un scénario de communication par envoi de messages
  - objets acteurs : à l'origine d'une interaction
  - objets serveurs : répondent à la sollicitation
  - objets agents : les deux

Que nous manque-t-il ?

- Soient 2 objets :
  - même structures de données (attributs)
  - même comportement (opérations)
- Il faut les décrire asbtraitement de la même manière → classes

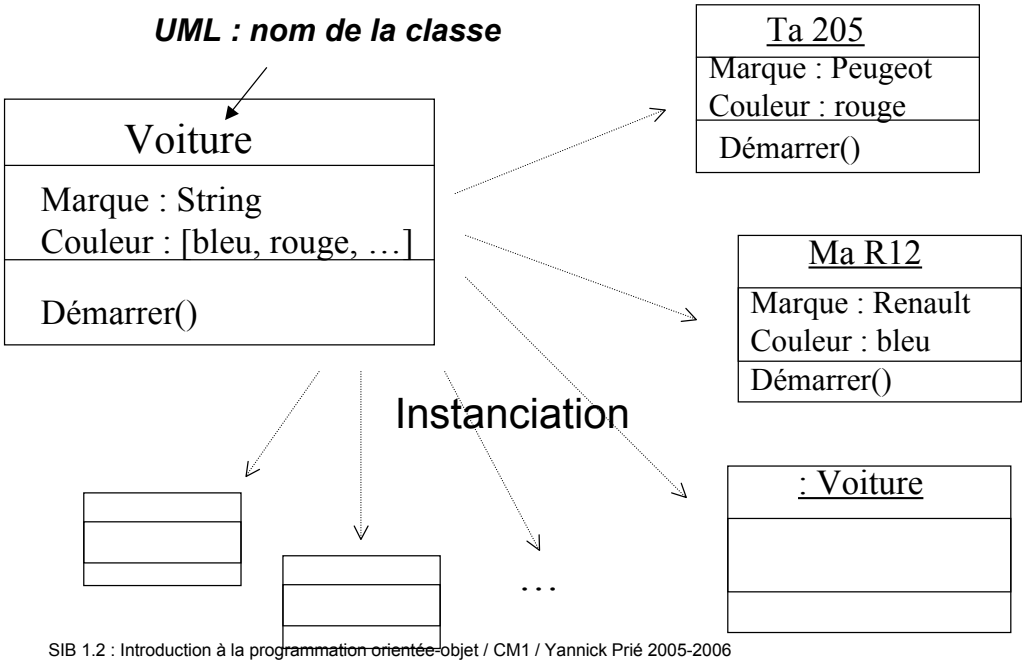
Ma R12
Marque : Renault Couleur : bleu
Démarrer()

Ta 205
Marque : Peugeot Couleur : rouge
Démarrer()

Regroupement en classes

- Les objets sont regroupés dans une *classe*
- Une classe est une *abstraction* décrivant les propriétés communes des objets qui en sont des *instances*
- Une classe décrit une infinité d'instances
- Un objet sait toujours de quelle classe il fait partie

Classification



Dans un programme OO

- On définit des classes
  - leur attributs, privés et publics
  - leurs méthodes, privées et publiques
- On instancie des objets à partir des classes
- On lance/gère la collaboration
  - envoi de messages à des objets
- Exécution du programme : objets
  - qui s'envoient des messages
  - qui changent d'état

Pour la suite

- Concepts objets : la suite
  - CM2
- Découverte du langage OO Python
  - Fin de ce cours + TP1

Python

- Langage
  - Interprété
  - Interactif
  - Portable
  - Orienté-objet
- Inventé en 1990 par Guido Van Rossum
- Avec pour objectifs
  - Simplicité et puissance
  - Programmation modulaire
  - Lisibilité du code
  - Développement rapide d'application
  - Facilité de fonctionnement avec d'autres langages

Installation et utilisation

- Disponible sur http://www.python.org/download (gratuit, Open Source)
- Utilisation
  - interactive (shell)
  - batch (programme)
- Editeur intégré
  - IDLE : Integrated DeveLopment Environment
- Pour nous
  - DrPython : plus pratique

Eléments du langage

- Les espaces sont importants
  - Indentation d'un bloc (tabulation)
- Commentaires
  - *# mon commentaire*
- Booléens
  - Tout ce qui vaut 0, null, vide, est de longeur nulle, etc. est évalué à Faux
  - Le reste est évalué à Vrai

Nombres

- décimal e.g. 631, 3.14
- octal e.g. O631
- hexadécimal e.g. oxABC
- complexe e.g. 1 + 3j
- long e.g. 122233445656455L

- Opérateurs arithmétiques et logiques standards
- Division entière : 1/2 = 0

Chaînes (1/2)

- Concaténation
  - "Hello" + "World" -> "HelloWorld"
- Répétition
  - "UMBC" * 3 -> "UMBCUMBCUMBC"
- Indexation
  - "UMBC"[0] -> "U"
- Tranches
  - "UMBC"[1:3] -> "MB"
- Taille
  - len("UMBC") -> 4

Chaînes (2/2)

- Comparaison
  - "UMBC" < "umbc" -> 0
- Recherche
  - "M" in "UMBC" -> 1
- Possibilité d'utiliser des « simples quotes » ou des triples guillemets
  - e.g. 'UMBC' ou """Du texte "compliqué" avec des caractères spéciaux """
- Non mutable
  - on ne modifie pas, on recopie
  - c1 = c1 + c3 → on ne modifie pas le c1 original

Listes

- Ensemble de valeurs quelconques
  - Ma_liste = [124, "Python", [32, "bonjour"]]
  - Quelques opérations
    - Opérations de chaînes (accès, tranche, etc.)
    - Ajout ma_liste.append(342)
    - Insertion ma_liste.insert(2, "toto")
    - Inversion ma_liste.reverse()
    - Tri ma_liste.sort()
    - ...
- Tuples = listes non mutables
- Mon_tuple = (124, " Python ", "bonjour")

Dictionnaires

- Ensemble de couples clé/valeur
  - Mon_dict = {“Guido”: “Python”, “Ullman”: “ML”}
  - Quelques opérations
    - Insertion Map[“Ritchie”] = “C”
    - Accès Map[“Guido”]
    - Effacement del Map[“Ullman”]
    - Itération keys() values() items()
    - Présence has_key(“Guido”)
- Les valeurs peuvent être n’importe quoi
- Les clés doivent être non mutables

Variables

- Pas besoin de les déclarer
- Déduction du type à l'initialisation

  ex. F = 2 * 4.5 → f de type float
- Variables globales et locales

Références

- a = b
  - ne fait pas de copie de b
  - a et b réfèrent au même objet

E.g.

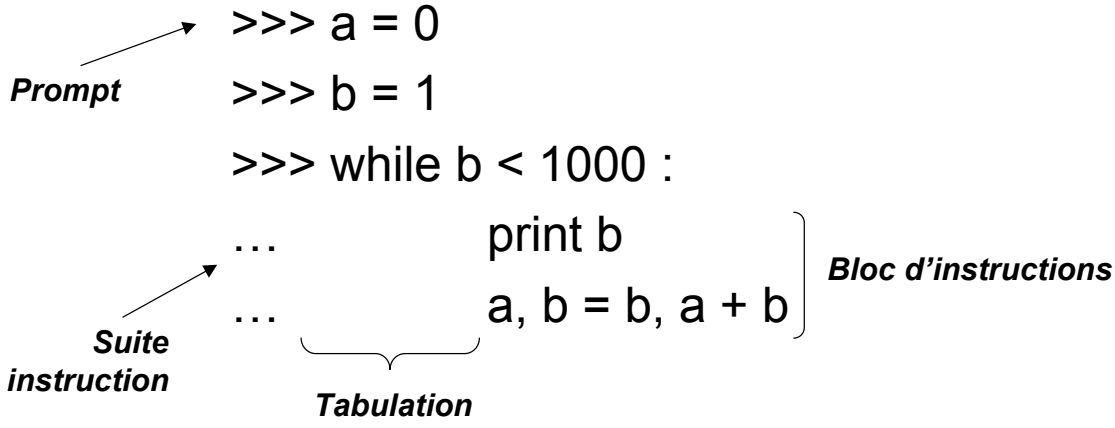
```
>>> a = [1,2,3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

Contrôle du flot

- if condition : statements
  [elif condition : statement]
  [else : statement]
- while condition : statements
- for var in sequence : statements
- break
- continue

Exemple

- (suite de Fibonacci)



Fonctions and procédures

- Forme générale

```
def name(arg1, arg2, …):
    Statements
    return                  # from procedure          OR
    return expression       # from function
```

- Exemple

```
>>> def fib(n):    # write Fibonacci series up to n
…                   """Print a Fibonacci series up to n."""
...                 a, b = 0, 1
...                 while b < n:
...                         print b
...                         a, b = b, a+b
...                 return a
```

Documentation

Print

- Pour afficher n'importe quoi
  - directement

```
>>> toto = 99
>>> print toto , 5
99 5
```

  - En formattant

```
>>> chaine = "Bonjour"
>>> print "J'affiche %s et %i" % (chaine, toto)
J'affiche bonjour et 99
```

Modules

- Sortes de librairies
- Fichier contenant des définitions et des instructions Python
- Les fichiers ont un suffixe
  - mon_module.py
- Le module a un nom
  - mon_module
- On peut importer le contenu d'un module
  - import mon_module
- Certains modules sont livrés avec Python
  - exemple : sys

Paquetages

- Collections de modules
- Espace de nom permettant d'accéder à des modules
  - A.B réfère au module B dans le paquetage A
- Pour importer le module B
  - import A.B
    - puis utiliser A.B.xxx pour accéder aux objets, fonctions
  - from A.B import *
    - puis utiliser directement and use only the module name
  - possibilité d'importer seulement quelques éléments
  - from A.B import xxx, yyy, zzz

Classes et objets

- Classes

```
class MaClasse:
        instructions
class MaClasse(ClasseBase1, ClasseBase2):
        instructions
```

Héritage (cf. CM2)

- Objets
  - x = MaClasse()
  - Crée un nouvelle instance de la classe MaClasse, et l'assigne à la variable x

Exemple de classe

```
class Pile:
    "Une structure de données bien connue."
    def __init__(self) :
            self.items = []
    def push(self, x) :
            self.items.append(x)
    def pop(self) :
            x = self.items[-1]
            del self.items[-1]
            return x
    def empty(self):
            return len(self.items) == 0
```

Constructeur (cf. CM2)

une méthode prend toujours self comme argument

self = l'instance elle-même

Exceptions

Mécanismes de gestion des erreurs

```
try:
    Print 1/x
except ZeroDivisionError, message:
    print “Can’t divide by zero”
    print message

f = open(file)
try:
    process_file(f)
finally :
    f.close()
print “OK”
```


Lever des exceptions

- Raise ZeroDivisionException
- Raise ZeroDivisionException(“can’t divide by zero”)
- Raise ZeroDivisionException, “can’t divide by zero”

- Python permet de définir ses propres exceptions

Domaines d'application

- *Glue language*
- Applications graphiques
- Applications basés sur des protocoles internet
- Applications de bases de données
- Applications web
- Applications multimédia

La suite au prochain épisode...

Remerciements

- Quelques transparents sont directement adaptés du cours « Python » par K. Naik, M. Raju et S. Bhatkar (2002)