Cours SI6

Python – POO par l'exemple

python

Principe de base : Instanciation

- Objet créé à partir d'un moule
- Moule -> une classe
- Objet = instance d'une classe
- Classe=
  - Structure -> attributs
  - Comportement -> méthodes

- Cacher des données dans l'objet
- Accès contrôlé par les accesseurs
- Objet = fournisseur de service
- Python -> pas d'encapsulation

Principe de base : Encapsulation

Principe de base : Polymorphisme

- Objets de même interface manipulés de manière générique
- En Python :
  - Polymorphisme = même signature

Principe de base : Héritage

- Réutilisation d'une classe en y ajoutant:
  - Nouveaux comportements
  - Nouveaux attributs
- Classe fille « est-un » classe mère
- Un chat « est un » Animal
- Un moteur n'est pas une voiture
- Une voiture contient un moteur

- Python : uniformité
- Tout est objet
  - Chaîne
  - Entier
  - Liste
  - Fonction
  - Classes
  - Module
  - …
- Objet manipulé par référence
- Hierarchie :
  - Module contient
    - Classes contiennent
      - Méthodes

Principe de base : Objets et références

Les classes

- Définition d'une classe avec class
- Suit la règle des blocs
- Héritage entre parenthèses
- Méthode : fonction avec au moins 1 argument:
  - self

```
>>> class Dummy(object):
...     def hello(self):
...         print 'hello world!'
```

- Instanciation sans new

```
>>> d = Dummy()
>>> d
<Dummy object at 0x...>
>>> d.hello()
hello world!
```

Attributs et constructeurs

- Attributs définis à la première affectation
- Attribut = self.variable
- Méthode vue comme attribut particulier
- Constructeur -> méthode __init__:
  - Unique donc non surchargeable
  - Mais valeurs par défaut autorisées

```
>>> class Compteur(object):
...     def __init__(self, v=0):
...         self.val = v
...     def value(self):
...         return self.val
```

Visibilité

- Par défaut -> attributs et méthodes:
  - publique en Python
- Par convention, « privée » -> _var:
  - Attribut quand même accessible
  - Mais il ne faut pas y accéder
- Double underscore __var:
  - Interdit la lecture/ecriture
  - Levée d’exception
- Philosophie Python : Faire confiance

```
>>> class Foo(object):
...     def __init__(self):
...         self._a = 0
...         self.__b = 1
...
>>> f = Foo()
>>> f._a
0
>>> f.__b
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: Compteur instance has no attribute '__b'
```

Démonstration Visibilité

Exo01:Classe Telephone

- Définir la classe Telephone
- Attributs
  - numero: le numéro de tél
  - type: domicile, mobile, travail
- Le constructeur prend les 2 paramètres
- Faire les 2 getters
- Surcharger __str__ qui est la méthode d'affichage dans la console

Héritage

- Python supporte héritage multiple
- Pour héritage:

```
>>> class A(object):
...     def __init__(self, n='none'):
...         self._name = n
...     def name(self):
...         return self._name

>>> class B(A):
...     def __init__(self, val=0, n='none'):
...         A.__init__(self, n)
...         self._wheels = val
...     def wheels(self):
...         return self._wheels
```

- Attention : appeler le constructeur de la classe mère!!!

```
>>> class C(object):
...     def __init__(self, t=''):
...         self._title = t
...     def title(self):
...         return self._title

>>> class D(A, C):
...     def __init__(self, n='none', t=''):
...         A.__init__(self, n)
...         C.__init__(self, t)
...     def fullname(self):
...         return self._title + ' ' + self._name
```

- NB: toute classe hérite de la classe object

Exemple héritage multiple

Les propriétés

- Permettent de manipuler un attribut de manière transparente
- Déguisement de méthode en attribut
- Propriété définit grâce à la fonction property
- Syntaxe de property:
  - Param1 : fct de lecture
  - Param2 : fct d’écriture

Surcharge des comparateurs

- Possibilité de surcharger tous les comparateurs:
  - == : __eq__(self,autreObjet)
  - != : __ne__(self,autreObjet)
  - > : __gt__(self,autreObjet)
  - >= : __ge__(self,autreObjet)
  - < : __lt__(self,autreObjet)
  - >= : __le__(self,autreObjet)

```
class Personne:
    def __init__(self,nom,taille):
        self._nom = nom
        self._taille = taille
    def __eq__(self,autrePersonne):
        return self._taille == autrePersonne._taille
    def getNom(self):
        return self._nom
    def __gt__(self,autrePersonne):
        return self._taille > autrePersonne._taille

if __name__ == '__main__':
    p1 = Personne("Toto",175)
    p2 = Personne("Titi",170)
    if p1>p2:
        print(p1.getNom()+" est plus grand que "+p2.getNom())
```

- Toto est plus grand que Titi

Exemple surcharge op

- Créer la classe Etudiant composée de:
  - Nom de l'étudiant
  - Prénom de l'étudiant
  - Liste de téléphones
  - La méthode pour ajouter un téléphone
  - La méthode __str__ de l'étudiant:
    - Son nom + son prénom
    - La liste des téléphones avec leur type
  - Les 3 getters
- Créer la classe EtudiantBoursier:
  - Qui hérite de Etudiant
  - Qui contient le niveau de bourse de l'étudiant:
    - B1, B2, …
  - Constructeur + __str__

Exo02:

Attributs de classe

- On déclare un attribut de classe (statique) directement dans la classe

```
class A():
    var = 5
    def __init__(self):
        print ("Objet de type A cree")
```

- On y accède en écriture en appelant la classe et non l'objet, sinon…:
  - l'attribut n'est plus vu comme attribut de classe

```
if __name__ == '__main__':
    a1 = A()
    print ("A.var : " +  str(A.var))
    print ("a1.var : " + str(a1.var))
    A.var = 10
    print ("A.var : " + str(A.var))
    print ("a1.var : " + str(a1.var))
    a1.var = 15
    print ("A.var : " + str(A.var))
    print ("a1.var : " + str(a1.var))
```

```
Objet de type A cree
A.var : 5
a1.var : 5
A.var : 10
a1.var : 10
A.var : 10
a1.var : 15
```

Démo attributs de classe

- 2 possibilités:
  - ma_methode = staticmethod(ma_methode)

```
class A():
    var = 5
    def __init__(self):
        print ("Objet de type A cree")

    def get3():
        return 3
    get3 = staticmethod(get3)
```

  - ma_methode = classmethod(ma_methode)

```
class A():
    var = 5
    def __init__(self):
        print ("Objet de type A cree")

    def get3(cls):
        return 3
    get3 = classmethod(get3)
```

  - Appel

```
print (A.get3())
```

Méthode de classe (sans décorateur)

- Regroupement:
  - De classes ou de fonctions dans un module
    - Correspond à un fichier
  - De modules dans un package
    - Correspond à un répertoire qui contient les modules
- Un module est référencé par:
  - nomPackage.nomModule
- Chaque package contient un fichier:
  - __init.py__
  - La liste des fichiers des modules

Hiérarchie des classes

Utilisation d’un module

- 2 méthodes:
  - import …
    - import nomPackage.nomModule
    - Puis nomPackage.nomModule.nomClasse
  - from … import …
    - from nomPackage.nomModule import nomClasse

Module ou programme

- Un module peut-être:
  - Importé
  - Et/ou exécuté
- Pour être exécuté, un module doit contenir un bloc d'exécution:

```
if __name__ == '__main__':
    a = 1
    b = 2
    c = a + b
    print (c)
```

- **sys**: fournit les paramètres et fonctions liées à l'environnement d'exécution
- **string**: fournit des opérations courantes sur les chaînes de caractères
- **re**: fournit le support des expressions régulières
- **os**: fournit l'accès aux services génériques du système d'exploitation.

Qqs modules standards

Fin