UNIVERSITÉ PARIS8 VINCENNES-SAINT-DENIS

Méthodologie de la Programmation

Palus Jean-Pascal

Licence IV - L1 - Semestre I

Bureau A184 - jpp@up8.edu

Séance III

La programmation orientée objet

Objets

- Python supporte nativement plusieurs types de données :

  1234 3.14159 "Hello word" [1, 5, 7, 11, 13]
  {"Univ" : "Paris 8", "Niveau" : "L1"}

- Chacun de ces types de données sont des **objets**.

Objets

- Chaque **objets** dispose de :
  - Un **type**
  - Une représentation de données interne, l'**implémentation**
  - Un ensemble de **méthodes** permettant d'intéragir avec lui, l'**interface**.
- Un objet est l'**instance** d'un type.
  - 1234 est l'instance d'un **int**.
  - "hello" est l'instance d'un **str**.

la Programmation orientée objet

- **En Python tout est un objet** (et a un type).
- Un langage permettant de faire de la POO est donc un langage permettant de :
  - Créer de nouveaux objets.
  - Manipuler des objets.
  - Détruire des objets (explicitement comme en C++ ou implicitement avec un garbage collector).

la Programmation orientée objet

- Tous les langages ne sont pas orientés objet même si on peut faire de la programmation objet dans n'importe lequel.
- Certains langages offre un support natif de la POO :
  - C++, Swift, Java, PHP, Go, Rust, Javascript, Python, ...
- Mais pas tous de la même façon :
  - Avec des classes qu'on instancie (C++, Java, Python).
  - Avec des prototypes qu'on clone (Self, JavaScript).
  - D'autres choses moins clairement nommées (Go).
- Avec différentes façons de typer :
  - Typage fort ou faible (Python vs. JavaScript)
  - Typage statique ou dynamique (OCaml vs. Racket)

Les objets

- Un objet représente un concept, une idée, ou toute entité abstraite.
- Il contient une **implémentation** interne.
- Des **méthodes** pour le manipuler.

Les objets

- Prenons l'exemple des listes :

Les objets

Prenons l'exemple des listes :

- **Implémentation** :

```
// listobject.h

#include <stdio.h>

typedef struct {
    PyObject_HEAD
    Py_ssize_t ob_size;

    /* Vector of pointers to list elements.  list[0] is ob_item[0], etc. */
    PyObject **ob_item;

    /* ob_item contains space for 'allocated' elements.  The number
     * currently in use is ob_size.
     * Invariants:
     *     0 <= ob_size <= allocated
     *     len(list) == ob_size
     *     ob_item == NULL implies ob_size == allocated == 0
     */
    Py_ssize_t allocated;
} PyListObject;
```

Les objets

Prenons l'exemple des listes :

- **Méthodes** :
  - L[i], L[i :j], +
  - len(), min(), max(), del(L[i])
  - L.append(), L.extend(), L.count(), L.index(), L.insert(), L.pop() ,L.remove(), L.reverse(), L.sort()

Les objets

- Les représentations internes sont généralement **privées**

Les avantages de la POO

- Permettre une meilleure modélisation des problèmes, et donc une faciliter leur programmation.
- Obtenir des briques logiciel facilement réutilisables en dissociant l'implémentation de l'interface.
  - On appelle cette pratique l'**encapsulation**.
  - De nombreux modules Python inplémentent de nouvelles classes.
  - Chaque classe à son propre environnement bien séparé qui évite les collision de noms.

Ne plus se concentrer sur les problèmes d'implémentation

```
P = Point(2,1)
Q = Point(6,4)
distance = P.Dist(Q)

print(distance)
# 5
```

- Ici, *Dist* est une **méthode** de la classe *Point* et *P.Dist(Q)* dit "calculer et renvoyer la distance du point P au point Q".

Ne plus se concentrer sur les problèmes d'implémentation

```
P = Point(2,1)
Q = Point(6,4)
distance = P.Dist(Q)

print(distance)
# 5
```

- Cela permet de se concentrer sur le concept de point plutôt que sur le concept de coordonnées.

Classes et Objets

- On fait la distinction entre **créer** une classe et **instancier** une classe.
  - Créer une classe consiste à définir une nouvelle classe, écrire ses attributs et méthodes.
  - Instancier une classe consiste à créer un exemplaire d'un objet. Par exemple *L=[1, 2]*.

Classes et Objets

- Une classe est en quelque sorte un plan permettant de créer un objet.
- Un **objet** est l'**instance** d'une **classe**.
  - **Classes** : int(), str(), list(), dict(), Point(), ...
  - **Objets** : 3, "hello", [1, 2, 3], ...

Créer ses propres classes

- Grâce au mot clef ***class***

Créer ses propres classes

```
class Point(object):
    """
    attributs et méthodes
    """
```

- Le mot clef ***class*** déclare une nouvelle classe.

Créer ses propres classes

```
class Point(object):
    """
    attributs et méthodes
    """
```

- Le nom de la classe est ici **Point**.

Créer ses propres classes

```
class Point(object):
    """
    attributs et méthodes
    """
```

- L'argument est la **classe parente** (**super-classe**) dont la nouvelle classe **hérite**.

Créer ses propres classes

```
class Point(object):
    """
    attributs et méthodes
    """
```

- Comme pour les fonction, le scope est défini par l'indentation.

```
class Point(object):
    """
    attributs et méthodes
    """
```

- Les attributs de la classes sont les données qui vont être partagées par toutes les instances de cette classe.

Créer ses propres classes

Attributs

```
class Point(object):
    x = 0
    y = 42
```

- Les **attributs de la classes** sont les données qui vont être partagées par toutes les instances de cette classe.

! Attention à ce que vous faites.

Créer ses propres classes

Constructeur

```
class Point(object):
    def __init__(self,x,y):
    """ crée un objet point """
        self.x = x
        self.y = y
```

- On peut initialiser l'objet on utilise une méthode spéciale nommée **constructeur**.

Créer ses propres classes

Constructeur

```
class Point(object):
    def __init__(self, x, y):
    """ crée un objet point """
        self.x = x
        self.y = y
```

- Lorsqu'une méthode est appelée sur une instance, elle reçoit cette instance en argument.
  - Dans certains langages il est implicite, dans d'autre il est souvent nommé ***this*** ou ***self*** (ou les deux).
  - En python il est de convention de le nommer ***self***.

Créer ses propres classes

Constructeur

```
class Point(object):
    def __init__(self, x, y):
    """ crée un objet point """
        self.x = x
        self.y = y
```

- x et y sont des **attributs d'instance** de la classe ***Point***.
- Elles reçoivent ici les valeurs passées en paramètres du constructeur.

Créer ses propres classes

Appeler le constructeur

```
class Point(object):
    def __init__(self, x, y):
    """ crée un objet point """
        self.x = x
        self.y = y

point = Point(3, 4)
```

- Le constructeur est appelé lors de l'instanciation de la classe.

- On interagit avec les attributs d'un objet à l'aide d'un point.

```
class Point(object):
    def __init__(self, x, y):
    """ crée un objet point """
        self.x = x
        self.y = y

point = Point(3, 4)
```

- Le constructeur est appelé lors de l'instanciation de la classe.

Créer ses propres classes

méthodes spéciales

- Il existe un certain nombre de methodes spéciales qui permettent d'homogénéiser le fonctionnement des objets en python.
- Par exemple vous trouvez sûrement intuitif de pouvoir faire :

```
i = 42
print(i)
```

- Ou encore :

```
L = ['a', 'b', 'c']

for i in L:
    print(i)
```

Créer ses propres classes

méthodes spéciales

- Pour cela il existe dans le langage une syntaxe permettant d'écrire des méthodes spéciales permettant ce genre de comportement.
  - __init__()
  - __str__()
  - __sizeof__()
  - __len__()
  - __pow__()
  - __floor__()
  - __iter__()
  - __repr__()
  - ...

- On peut maintenant utiliser les classes créées comme un type standard.

```
# Renvoie un point random dans le carré
# de diagonale [x, y]
def point_random(xa, ya, xb, yb):
    x = random.randint(xa,xb)
    y = random.randint(ya, yb)
    return Point(x, y)

# Renvoie le milieu du segment [P1, P2]
def point_milieu(Pa, Pb):
    x = (Pa.x + Pb.x) // 2
    y = (Pa.y + Pb.y) // 2
    return Point(x, y)
```

- On peut maintenant également implémenter ses propres méthode.

```
class Point(object):

    # Constructeur
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, destination):
        """ Implémentons cette méthode """
```

- **self** est toujours le premier argument des méthodes de la classe.

- Les méthodes et les fonctions sont donc fondamentalement de même nature.
  - On les appelle méthodes quand elles sont implémentées dans une classe...
  - et fonctions dans le reste du code.
- En python les fonctions sont généralement, comme en mathématiques, préfixes : ***fonction(argument)***.
- Et les méthodes sont généralement postfixes : ***objet.méthode()***

Créer ses propres classes

- Créer des alias est transparant quand cela concerne les types de base immutables, ça l'est moins pour les classes que l'on crée soit même.

```
i = 3
j = i

# Alias
P = Point(3, 4)
Q = P

# Copie
Q = Point(3, 4)
P = copy(Q)
```

Créer ses propres classes

- Essayons d'écrire une classe **Fraction**...