Programmation orientée objet avec Python

Partie 1

Historique

- La programmation objet est inventée dans les années 1970 comme une extension de la programmation procédurale.
- L'idée est de concevoir les programmes non plus comme des lignes de codes qui s'exécutent séquentiellement, mais comme des objets qui dialoguent.

Historique

- Simula 64 est le premier langage à utiliser ce paradigme mais de façon limitée.
- Smalltalk, développé à partir de 1971 par Alan Kay et publié en 1980, est le premier vrai langage objet.
- D'autres suivront : Eiffel en 1986, Python en 1991, ...
- Des versions objets de langages existants seront développées : C++ pour le C, ADA95 pour le ADA, Object Pascal pour le Pascal, PHP5 pour le PHP, ...
- Pour une histoire des langages, voir par exemple http://www.levenez.com/lang/

Historique

- La programmation objet devient à la fin des années 90 le paradigme dominant en programmation.

Un objet ? C’est quoi ?

Exemple 1:

Ali est une personne

Un rectangle est une forme géométrique

Une droite est une forme géométrique

TOYOTA Yaris est une voiture

Un objet ? C’est quoi ?

Un objet : quelque chose concrète

Classe : quelque chose abstraite, type de données, catégorie d’objets ou encore modèle à suivre pour la création des objets

➔ Un objet est une instance d’une classe

Un objet ? C’est quoi ?

Donc ,

Ali (objet) est une personne (classe)

Un rectangle (objet) est une forme géométrique (classe)

Une droite (objet) est une forme géométrique (classe)

TOYOTA Yaris (objet) est une voiture (classe)

des objets

Notion d’objet

Les objets qui peuvent avoir des contenus semblables et susceptibles d'être décrits de la même façon sont regroupés dans la même classe d’objets : Généralisation

Exemple 2:

La classe des véhicules permettant de transporter seulement des personnes est la classe « voiture »

Renault Megane est une voiture

Citroen DS4 est une voiture

Notion d'objet

Chaque objet en mémoire regroupe :

- des attributs : valeurs typées qui décrivent les caractéristiques de l'objet
- des méthodes : fonctions qui décrivent ce que l'objet peut faire et qui opèrent sur les attributs des objets

Un objet = données (attributs) + procédures (méthodes)

Notion d'objet

Exemple:

L’objet SCANIA G340 GNL est un camion, il peut avoir les données (attributs) suivants :

- Nombre de personnes : entier
- Poids vide : Réel
- Poids max : Réel
- Capacité de la remorque
- Moteur : un objet d’une autre classe (composition)

Notion d'objet

Exemple:

Et peut avoir les méthodes qui permettent de modifier l’état de ses attributs, donc changer l’état de l’objet:

- Démarrer moteur
- Arrêter moteur
- Avancer
- Reculer
- Verser la remorque

- Remarque :

Un attribut peut être un objet appartenant à la même classe

Exemple des attributs d'un objet de la classe personne

- Père : Personne
- Mère : personne

Notion d'objet

Les attributs des objets peuvent être des objets également.

Les paramètres des méthodes et leurs valeurs de retour peuvent également être des objets.

Notion d'objet

En résumé :

- Tous les objets ont un type (leur classe)
- Tous les classes sont des type (classe Type)
- Tous les types sont des objets (classe Object)

➔ En python tout est objet

Exemples :

5 : objet de la classe « int »

"abc" : objet de la classe « str »

[1,2] : objet de la classe « list »

list : objet de la classe « type »

type : objet de la classe « object »

Notion d'objet

Dans ce chapitre, nous allons créer :

- nos premières classes,
- nos premiers attributs
- et nos premières méthodes.

Nous allons aussi essayer de comprendre les mécanismes de la programmation orientée objet en Python.

Lignes directrices

Dans ce chapitre, nous allons créer :

- nos premières classes,
- nos premiers attributs
- et nos premières méthodes.

Nous allons aussi essayer de comprendre les mécanismes de la programmation orientée objet en Python.

Notion de classe

- Une classe est un peu un modèle suivant lequel on va créer des objets.
- Une classe est un ensemble d'attributs (ou variables) et de méthodes (ou fonctions).
- Un programme utilisant les classes est orienté objet.
- Il est possible de faire les mêmes choses avec ou sans classes mais leur utilisation rend d'ordinaire les grands programmes plus facile à comprendre et à construire

Notion de classe: Créer la première classe en Python

- Le mot réservé **class** suivi d'un identificateur permet de débuter la définition d'une classe.
- *Par convention : commencer les noms de classe par une majuscule.*

```
class point():
        """Les objets de cette classe sont vides pour le moment"""
        pass

>>> type(point)
<class 'type'>
```

Donc la classe "point" est un "type" en Python

Notion de classe: Créer la première instance en Python

- On peut instancier un élément de la classe, vérifier qu'il existe et même l'afficher.

```
class MaClasse(): # définition d'une classe "vide"
        pass
a = MaClasse() # création d'un nouvel objet a
print(type(a)) # quel est le type de a ?
<class '__main__.MaClasse'>
print(a) # qu'est ce que a ?
<__main__.MaClasse object at 0x01B50D70>
```

Sans autre précision, l'affichage donne le nom de la classe et l'adresse (en hexadécimal) de l'objet.

Notion de classe : attributs

- Les attributs sont des sous objets de types déjà définis qui composent notre objet plus complexe et que l'on va nommer.
- Les valeurs des attributs ne sont a priori pas définis dans la classe, car propres à chaque objet. Une valeur par défaut peut cependant être spécifiée.
- Les attributs sont déclarés le plus souvent
  - à l'intérieur du constructeur,
  - plus généralement à l'intérieur de toute méthode,
  - voire à l'extérieure de la classe.

Notion de classe : attributs

```
class point:
    """Modélise les points d'un plan muni d'un repère"""
    x=None # première composante
    y=None # deuxième composante
```

Notion de classe : attributs

- Exécuter ce script, puis dans le shell, saisir les commandes suivantes :

>>> ? point #afficher l'aide de cette classe

>>> P=point() #Créer un objet Point P

L'accès aux données d'un objet se fait par la notation pointée : nomObjet.nomAttribut

>>> P.x

>>> P.x=2; P.y=3

>>> P.x+P.y

- Les attributs d'un objet sont accessibles et modifiables après la construction de l'objet.

Notion de classe : attributs

Il est toujours possible en Python de définir après la définition d'une classe, des attributs pour un objet instance de cette classe.

Essayer les instructions suivantes dans le shell :

```
p = point()
p.truc="ceci est un attribut d'instance"
print(p.truc)
q = point()
q.truc
q=p
print(q.truc)
q.truc=" je modifie l'attribut truc"
print(p.truc)
```

Notion de classe : attributs

Donc, on distingue en Python deux types d'attributs :

les **attributs de classe** qui existent chez tous les objets d'une classe (x, y).

et les **attributs d'instance** qui sont définis après la construction d'un objet et qui peuvent différer d'un objet à l'autre d'une même classe.(truc)

Notion de classe : Méthodes

- Il s'agit de fonctions que l'on explicite, en général, lors de la définition de la classe et qui utilisent ou/et modifient, les objets de cette classe.

Notion de classe : Méthodes

- Par convention, le premier argument d'une méthode est souvent appelé self qui représente l'objet non encore instancié.
- Ce paramètre sera remplacé lors de l'appel de la fonction par l'objet lui-même.
- *De manière analogue, au moment de la définition d'une fonction (par exemple def f(x): return x*x) x ne désigne aucune variable, mais prendra corps seulement lorsque la fonction sera appelée par exemple a=3;f(a).*

Notion de classe : Méthodes

Les règles d'accès sont les mêmes que pour les attributs.

Elles acceptent également la récursivité et les paramètres par défaut à l'exception du premier.

Tester le script suivant :

```
class point:
        x=None
        y=None
        def initialise (self ,a,b):
                self.x=a
                self.y=b
P=point()
P. initialise (1,-1)
print(P.x,P.y)
```

Notion de classe : Méthodes

Ajouter maintenant les lignes suivantes dans la définition de la classe point :

```
def estPlusMeridionalQue (self ,Q):
        return self.y<=Q.y

def estPlusOccidentalQue (self ,Q):
        return self.x<=Q.x
```

Exécuter ces commandes dans le shell :

```
>>> P=point (); Q=point ()
>>> P. initialise (1 ,1); Q. initialise (-1,4)
>>> P. estPlusMeridionalQue (Q)
>>> P. estPlusOccidentalQue (Q)
```

Notion de classe : Méthodes

Plus généralement,

Si uneMethode(a) est une méthode de la classe MaClasse, et si X est un objet de type MaClasse, alors X.uneMethode(a) équivaut à MaClasse.uneMethode(X,a)

Exemple :

```
>>> P=point (); Q=point ()
>>> P. initialise (1 ,1); Q. initialise (-1,4)
>>> Point.estPlusMeridionalQue (P,Q)
>>> Point.estPlusOccidentalQue (P,Q)
>>> print(P is Q)
```

Notion de classe : Méthode init()

- Une classe peut définir une méthode spéciale nommée __init__(), joue le de constructeur, qui permet d'initialiser les objets de cette classe comme ceci :

```
def _ _init_ _(self):
        self.donnee = valeur
```

- Dans ce cas, l'instanciation de la classe appelle automatiquement _ _init_ _().
- La méthode __init__() peut avoir des arguments pour offrir plus de souplesse.

Exercice 1

1. Ajouter à la classe point une méthode echangexy qui échange les valeurs des attributs x et y d'un objet de cette classe.
2. Définir une méthode afficher de la classe point qui affiche pour un objet de cette classe la chaine (x; y) où x et y sont les valeurs des attributs x et y de l'objet.
3. Ecrire une méthode appartientDroite de la classe point telle que P.appartientDroite(A,B) renvoie True si P appartient à la droite (AB) et False sinon.

Notion de classe : Copie d'instances

- Les instances de classes sont des objets modifiables, comme pour les listes, une simple affectation ne signifie pas une copie mais un second nom pour désigner le même objet.
- Exemple :

```
a  = Point(1,2)
b = a
b.x = -1
print (a.x) # affiche 1
print (b.x) # affiche ?
print(a is b)
```

Notion de classe : Copie d'instances

- Il faut donc copier explicitement l'instance pour obtenir le résultat souhaité.

```
a = Point (1,2)
import copy
b = copy.copy (a)
b.x = -1
print (a.x) # affiche 1
print (b.x) # affiche ?
print(a is b)
```

- Lorsque une classe inclut une variable de type classe, il faut utiliser la fonction deepcopy et non copy.

Notion de classe : Dériver une classe

Exemple:

- Un Bus et une Véhicule
- On dit que la classe Bus hérite ou dérive la classe Véhicule
- Grâce à cet héritage, tous les attributs et les méthodes de la classe Véhicule peuvent être utilisés par les instances de la classe Bus et par toutes les classes qui sont descendantes de la classe Bus (ex : Mini-Bus)

Notion de classe : Dériver une classe

- L’héritage est l’intérêt majeur des classes et de la programmation orientée objet.

Avantages :

- La classe dérivée possède alors les attributs et méthode de la classe dont elle dérive et ses propres attributs et méthodes.
- Le mécanisme d’héritage permet la multiplicité des classes de base.

Notion de classe : Dériver une classe

- Lorsque l'on veut que l'héritage provienne d'une classe précise, on doit le préciser dans la définition de la nouvelle classe.

Syntaxe:

class nom_classe_Fille (nom_classe_Mère):

Exemple:

class Bus (Véhicule):

Notion de classe : Dériver une classe

Dérivons par exemple la classe point:

- Créer une classe PointMobile dérivée de la classe Point
- Ajouter une méthode appelée bouger qui prend deux paramètres, appelés dx et dy. Cette méthode déplace le point dans la direction x et y le nombre d'unités donné.
- Ajouter une méthode info qui permet d'afficher par exemple le message suivant :

« A l'instant 10s, le mobile se trouve en (4; 8) avec une vitesse égale à 1,2 ms^{-1}»

Rappel : vitesse entre deux points a et b = distance(a,b) / temps nécessaire au déplacement entre a et b.

Distance = racine_carrée $((x\_b - x\_a)^2 + (y\_b - y\_a)^2)$

Notion de classe : Dériver une classe

Dérivons par exemple la classe point:

```
class pointMobile (point):
        """ Cette classe hérite de la classe point """
        t = 0
        v = 0 #Vitesse initiale
        def info(self):
                print(" A l'instant ",self.t,"s, le mobile se trouve en ")
                self.affiche ()
                print(" avec une vitesse égale à :", self.v ,"m/s")
        def bouger(self, dx, dy):
                self.x + =dx
                self.y + =dy
                from math import sqrt
                self.v = sqrt( dx**2 + dy**2) / self.t
```

Notion de classe : Dériver une classe

Dans le shell, on peut alors saisir :

```
>>> P= pointMobile ()
>>> P. initialise (2 ,3)
>>> P. bouger(4 ,5)
>>> P.t=10
>>> P.info ()
```

Notion de classe : Dériver une classe

- *La bonne question à se poser pour décider si B doit hériter de A est : B is A ou bien B has A ?*
  - *Si la réponse est B is A, alors B doit hériter de A,*
  - *Si la réponse est B has A, alors B doit avoir un attribut A*.

Notion de classe : Dériver une classe

- Un cercle POSSÈDE un centre, donc possèdera un attribut qu'on appellera probablement centre.
- La classe Cercle s'écrira sans héritage:

```
class Cercle(object) :
        def __init__(self, centre, rayon) :
                self.centre = centre #attribut
                self._rayon = rayon
        def perimetre(self) : pass
        def aire(self) :pass
```

Notion de classe : Dériver une classe

- Par contre un triangle EST un polygone. La classe Triangle s'écrira avec héritage.
- Supposant que l'on dispose d'une classe Polygone, comme celle-ci :

```
class Polygone(object):
        def __init__(self, liste_de_points) :
                self._points=liste_de_points
        def isClosed(self) :
                return self._points[0] == self._points[-1]
        def close(self) :
                if not self.isClosed() :
                        self._points.append(self._points[0])
```

Notion de classe : Dériver une classe

On pourra définir une classe Triangle comme ceci :

```
class Triangle(Polygone):
        def __init__(self, a, b, c) :
                Polygone.__init__(self,[a,b,c])
        def hauteur(self):
        #formule de calcul d'hauteur  spécifique à Triangle
        def isRectangle(self) : pass
        #etc...
```

Notion de classe : Dériver une classe

et l'utiliser ainsi :

t=Triangle(Point(1,0), Point (0,1), Point (1,1))

t.isClosed() # méthode de Polygone

t.hauteur() # méthode spécifique à Triangle