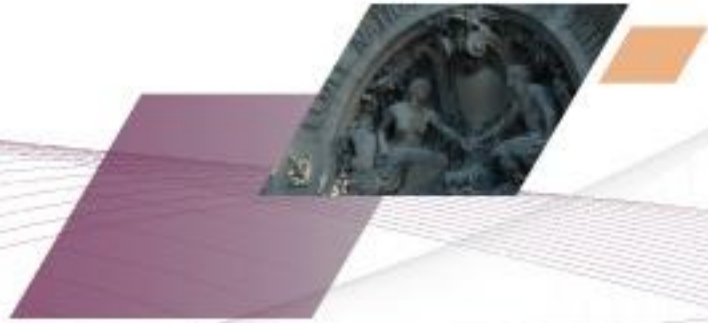
Mécatronique

Le langage Python

Version 3



Document 2 :

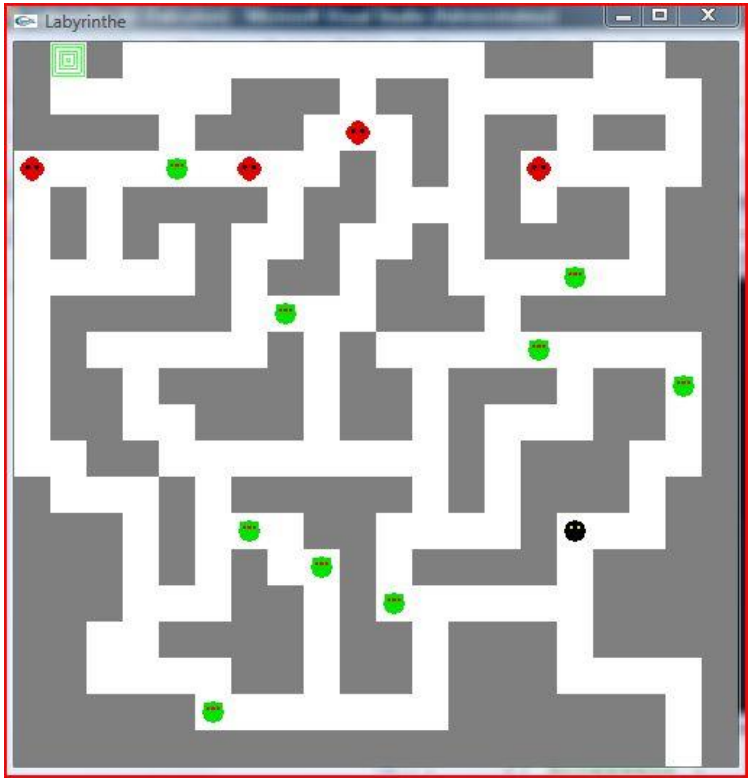
Les objets et les classes

Année 2016-2017, Centre de PARIS
rene.boulaire@ensam.eu

ARTS ET MÉTIERS ParisTech

Le Python pour programmer

Un exemple de programme qui pourra être réalisé facilement en Python.



Python, langage pour la POO

- POO, Programmation orientée objet
- Les types de base (int, float, str, …) sont des classes en Python. Elles sont définies par défaut dans le langage Python
- En Python il est possible de définir ses propres types qui dans ce cas seront appelés des classes. Le mot réservé class est utilisé pour définir une nouvelle classe
  Exemple :

```
class nom_de_la_classe:
    #definition de la classe
    ……..
    ……..
```

Python, les classes et les objets

- Une **classe** correspond donc à un type dérivé dont chaque élément est appelé **membre** de la classe et dispose de son propre type. Les membres données sont appelés des **propriétés**, les membres fonctions sont appelés **méthodes**.

- Une **classe** peut regrouper des **données**, des **fonctions**, des **opérateurs** et est analogue à un type élargi.

- Le processus de déclaration d'un objet (analogie avec la déclaration d'une variable sur un type prédéfini ou structuré) est également appelé **instanciation** d'une classe. Un **objet** est une **instance** d'une **classe**.

Python, les classes et les objets

- Un objet existe en mémoire en terme de données (appelées attributs d'instance). Cet objet peut accéder à ses fonctions et divers opérateurs définis au sein de sa classe suivant le statut de ses membres.

- Les membres ont un statut (visibilité) par défaut publique.
  Ce statut peut être privé et donc non visible par les objets instanciés sur la classe. Pour cela il est nécessaire de préfixer par deux soulignés le nom de l'attribut. (__).

  En P.O.O. ceci correspond à la notion d'**encapsulation**.

- Une propriété (attribut de classe) peut être partagée par toutes les instances de classe (objets). Cet attribut est dit **statique**.

Exemple de classe

```
class MaClasse():

    def __init__(self):              # constructeur de la classe
      self.a=0                       # a membre de la classe comme propriété
      self._b=1                      # a est un attribut d'instance
      self.__c=2                     # membre non visible par une instance de la classe

    def affiche(self):               # affiche() membre de la classe comme méthode
      print('a=',self.a)
      print('b=',self._b)
      print('c=',self.__c)

# programme utilisant la classe
c=MaClasse()
c.affiche()
print(c.a)
print(c._b)
#print(c.__c)
```

Python, les classes et les objets

Autre exemple de classe

```
class Personne():

   cp=0                          # cp est une propriété ou attribut de classe
                                 # partagée par toutes les instances de la classe
   def __init__(self):
         self.cp=0
         Personne.cp += 1

# programme utilisant la classe
a=Personne()
print(a.cp)
b=Personne()
print(b.cp)
print(Personne.cp)
```

Python, le constructeur

- Il existe une méthode (fonction membre) spéciale appelée automatiquement à la création (déclaration) de l'objet. C'est la méthode __init__ lorsqu'elle existe dans la définition de la classe.
- Cette méthode est souvent appelée le **constructeur** pour certains langages (C++, etc …).
- Une classe contient un seul constructeur.
- Il est toujours déclaré de la même façon avec un premier paramètre dont le nom est en général « self » et correspond à une référence d'objet appelant pour les différents membres de la classe.

```
def __init__(self):
```

- Cette méthode ne retourne aucune valeur ou objet.

Python, le destructeur

- le **destructeur** est une fonction membre appelée automatiquement lors de la destruction de l'objet.
  Une classe peut ne pas avoir de constructeur, cas le plus courant

- La destruction d'un objet peut avoir lieu de plusieurs façons :

  - Appel explicite avec le mot clé :

    ```
    del nom_de_l'objet
    ```

  - Par la définition de la méthode de classe :

    ```
    def __del__(self):
    ```

  - Lorsque l'espace de noms contenant l'objet est détruit (au sein d'une fonction par exemple).

Python, les méthodes

- Les méthodes de classe (y compris le constructeur) peuvent prendre des paramètres avec des valeurs par défauts.

- self représente l'instance elle-même, elle doit apparaître en première position dans la définition de toutes les méthodes.

- Pour une bonne lecture de la classe, il est préférable d'énumérer les champs de la classe (propriétés). Ils sont typés lors de l'affectation.

Python, les opérateurs

- Certains opérateurs peuvent être redéfinis pour les besoins de la classe. Ainsi, comme pour les types prédéfinis, des opérations seront possibles sur les instances de classes (objets).

+	def **__add__**(self, objet_a_ajouter)
-	def **__sub__**(self, …)
*	def **__mul__**self, …)
/	def **__truediv__**(self, …)
……	
==	def **__eq__**(self, objet_a_comparer)
!=	def**__ne__**(self, …)
>	def **__gt__**(self, …)
<	def**__lt__**(self, …)
……	

Python, les méthodes

- Le « . » permet l'accès aux différents membres.

- Au sein d'une classe, il est possible d'instancier un objet sur cette même classe.

- Une méthode peut renvoyer un objet (instance de classe).

Exemple :

```
class Personne:
    ……….
    def copie(self):
        q = Personne()
        q.nom = self.nom
        q.age = self.age
        q.salaire = self.salaire
        return q

# fin définition classe Personne
```

Python, les décorateurs

- Le décorateur @property permet de définir une propriété en lecture.
  - En interne à la classe c'est une méthode.
  - L'objet voit cette en seule lecture comme étant une propriété.

```
@property
def xvalue(self): return self.__x
```

- Le décorateur @name.setter (name sera remplacé par le nom de la méthode) permet de définir une méthode en écriture.

```
@xvalue.setter
def xvalue(self, value): self.__x=value
```

- Le décorateur @staticmethod permet de définir une méthode de classe accessible par la classe elle-même (pas d'argument self).

Python, bonnes pratiques

- Définir les classes dans des modules (fichier .py).
- Importer le module contenant la ou les classes.
- Pour explorer le contenu d'une classe ou d'un type prédéfini, il faut utiliser la fonction **dir()** sur l'objet ou le type concerné.
- Il n'est pas conseillé d'accéder directement en lecture/écriture aux propriétés de la classe mais plutôt mettre en place des accesseurs et des mutateurs qui sont des méthodes de classe.

Python, Composition et Héritage

- **L'héritage** et la **composition** vont permettre de construire de nouvelles classes à partir de classes existantes.
- La composition correspond à l'utilisation d'une ou plusieurs classes dans la définition d'une autre classe (agrégation).
- L'héritage correspond à la spécialisation de classe existantes.

Python, héritage ou dérivation

- L'**héritage** permet la création d'une nouvelle classe enrichie à partir d'une classe existante (dite classe de base). Ceci est aussi appelée **spécialisation ou dérivation**.
  Cette notion, au niveau de la modélisation, impose qu'un objet de la classe dérivée soit un objet de la classe de base, l'inverse n'étant pas vrai. La dérivation est publique et ne change pas le statut des membres.

```
class A
class B(A):
    ……
    ……
```

L'objet instancié sur une classe dérivée dispose de tous les membres de la classe de base ou de la hiérarchie plus les membres de sa propre classe. L'accès aux membres respecte le statut des membres.

Python, héritage ou dérivation

- L'**héritage** peut être multiple.

- Si la classe de base dispose d'un constructeur, la classe qui hérite doit se charger de passer les paramètres au constructeur de la classe de base.

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

    def name(self):
        return self._name
# Fin définition classe A

class B(A):
    def __init__(self, val=0, n='none'):
        A.__init__(self, n)                    # Appel du constructeur de la classe A
        self._valeur = val

    def Valeur(self):
        return self._valeur

class C(A,B)
        ……
```

Python, héritage et surcharge

- Surcharge de méthode : lorsqu'une classe B hérite de la classe A et redéfinit une méthode de la classe A portant le même nom, on dit qu'elle surcharge cette méthode. La méthode surchargée est exécutée par défaut sauf si la méthode de la classe de base est appelée explicitement par l'objet de la classe dérivée.

- Surcharge d'attributs : la surcharge d'attributs n'est pas possible. Si une classe de base possède un attribut a, les classes dérivées le possèdent aussi et ne peuvent en déclarer un autre du même nom.

- Pour que les attributs déclarés dans le constructeur de la classe de base soient des attributs pour la classe dérivée, le constructeur de la classe dérivée doit appelé le constructeur de la classe de base.

Python, héritage et surcharge

```
class A:                        # classe de base A
   def __init__(self):
      self.a=5
   def affiche(self):
      return ('a=',str(self.a))

class B(A):                      # classe B dérivée de la classe de base A
   def __init__(self):
      A.__init__(self)          # attribut a déclaré dans la classe de base
      self.a +=1                # devient un attribut dans la classe dérivée
   def affiche(self):
      A.affiche(self)
      return ('a=',str(self.a))

x=A()
print(x.affiche())
y=B()
print(y.affiche())
```

Python, Polymorphisme

- Le polymorphisme est la possibilité pour une méthode portant le même nom mais appartenant à des classes distinctes héritées d'effectuer un travail différent. Cette technique est acquise par la surcharge.

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