CHAPITRE 11 : PROGRAMMATION ORIENTÉE OBJET

Quand vous avez appris la programmation, on vous a montré comment stocker des données dans des structures de données:

- les listes
- les chaînes
- les entiers
- les dictionnaires
- etc

Et on vous a montré comment créer un comportement pour votre programme en utilisant des mots clés, puis plus tard en utilisant des fonctions pour regrouper ces mots clés.

En Python, à ce stade, vous utilisez des fonctions pour agir sur des structures de données.

La programmation orienté objet, est un style de programmation qui permet de regrouper au même endroit le comportement (les fonctions) et les données (les structures) qui sont faites pour aller ensemble.

En effet python est un langage orienté objet qui permet le traitement de :

- Classe
- Objet
- Encapsulation
- Héritage
- …..

I. Classes et objets :

I-1. La classe :

Une classe est un ensemble de données et de fonctions regroupées dans une même entité. Une classe est une description abstraite d'un objet. Les fonctions qui opèrent sur les données sont appelées des **méthodes**. Entre classe et objet il y a, en quelque sorte, le même rapport qu'entre type et variable.

I-2. L'objet :
On dit qu'un **objet est une instance de classe**.

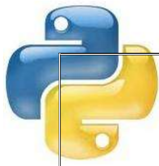
Si on prend la classe Voiture, nous pourrons avoir plusieurs voitures qui seront chacune des instances bien distinctes. Par exemple, la voiture de *Bennani*, qui est de couleur rouge avec une vitesse de 30 km/h, est une instance de la classe Voiture, c'est un **objet**. De même, la voiture de *Alaoui*, qui est de couleur grise avec une vitesse de 50 km/h, est un autre objet. Nous pouvons donc avoir plusieurs objets pour une même classe, en particulier ici deux objets (autrement dit : deux instances de la même classe). Chacun des objets a des valeurs qui lui sont propres pour les attributs.

II. Classes et objets sous python:

II-1. Création de classe sous python:

Syntaxe :

```
class Nom_classe:
        "Code à inserer"
```



Exemple: La création de la classe Point

```
class Point:
   "Definition d'un point geometrique"
```

II-2. Création d'objet sous python:

Pour créer un objet ou une instance d'une classe on utilise la syntaxe suivante:

```
Objet=Nom_classe(param1,param2,...)
```

Exemple:

```
P=Point()
```

Remarque:

- Comme pour les fonctions, lors de l'appel à une classe dans une instruction pour créer un objet, il faut toujours indiquer des parenthèses (même si aucun argument n'est transmis). Nous verrons un peu plus loin que ces appels peuvent se faire avec des arguments.
- On peut instancier autant d'instances qu'on veut.

II-3. Notion de constructeur:

Si lors de la création d'un objet nous voulons qu'un certain nombre d'actions soit réalisées (par exemple une initialisation), nous pouvons utiliser un **constructeur**.
Un constructeur n'est rien d'autre qu'une méthode, sans valeur de retour, qui porte un nom imposé par le langage Python : **__init__()**. Ce nom est constitué de **init** entouré avant et après par __ (**deux fois** le symbole **underscore** _, qui est le tiret sur la touche 8). Cette méthode sera appelée lors de la création de l'objet. Le constructeur peut disposer d'un nombre quelconque de paramètres, éventuellement aucun.
Il suit la syntaxe suivante:

```
class Nom:
        def __init__(self,param1,param2,...):
```

Exemple sans paramètre :

```
class Point:
   def __init__(self):
      self.x = 0
      self.y = 0

a = Point()
print("a : x =", a.x, "y =", a.y)
a.x = 1
a.y = 2
print("a : x =", a.x, "y =", a.y)
```

```
a : x = 0 y = 0
a : x = 1 y = 2
```

Dans cet exemple, nous avons pu définir des valeurs par défaut pour les attributs grâce au constructeur.

Exemple avec paramètres

```
class Point:
   def __init__(self, abs, ord):
      self.x = abs
      self.y = ord

a = Point(1, 2)
print("a : x =", a.x, "y =", a.y)
```

a : x = 1 y = 2

II-4. Autre méthode de création des attributs :

On utilise la syntaxe suivante pour définir un attribut dans une classe:

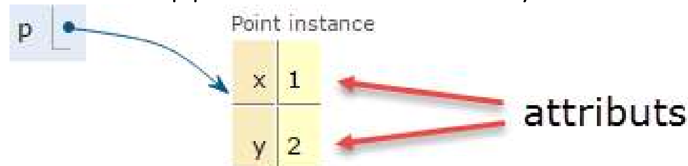
```
Nom_Objet.NomAttribut=valeur
```

Exemple:

```
class Point:
   "Definition d'un point geometrique"

p = Point()
p.x = 1
p.y = 2
print("p : x =", p.x, "y =", p.y)
```

L'objet dont la référence est dans p possède deux attributs : x et y.

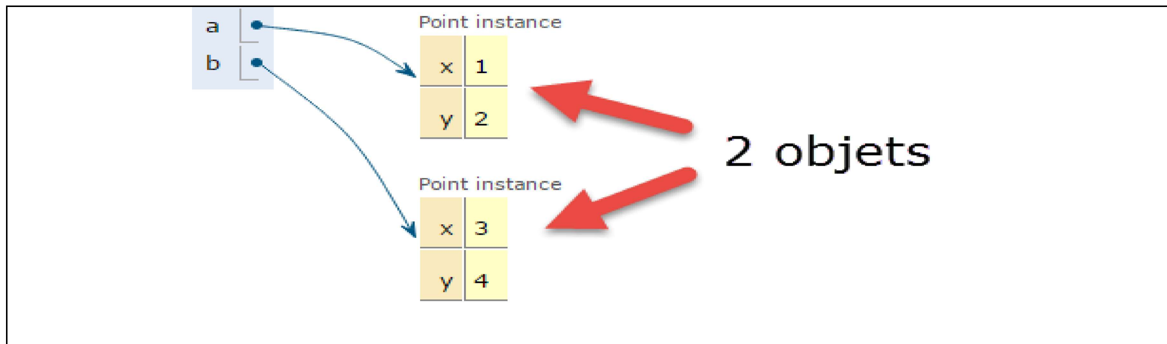


Exemple: On peut créer plusieurs objets.

```
class Point:
   "Definition d'un point geometrique"

a = Point()
a.x = 1
a.y = 2
b = Point()
b.x = 3
b.y = 4
print("a : x =", a.x, "y =", a.y)
print("b : x =", b.x, "y =", b.y)
```

On a 2 instances de la classe Point, c'est-à-dire 2 objets de type Point. Pour chacun d'eux, les attributs prennent des valeurs qui sont propres à l'instance.

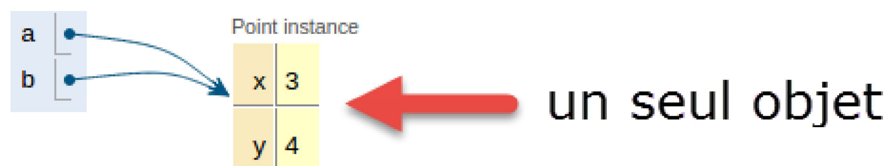


II-5. Distinction entre variable et objet :

L'exemple suivant montre bien la distinction entre variable et objet :

```
class Point:
   "Definition d'un point geometrique"

a = Point()
a.x = 1
a.y = 2
b = a
print("a : x =", a.x, "y =", a.y)
print("b : x =", b.x, "y =", b.y)
a.x = 3
a.y = 4
print("a : x =", a.x, "y =", a.y)
print("b : x =", b.x, "y =", b.y)
```



Ici les variables a et b font référence au même objet. En effet, lors de l'affectation b = a, on met dans la variable b la référence contenue dans la variable a. Par conséquent, toute modification des valeurs des attributs de l'objet dont la référence est contenue dans a entraîne une modification pour b.

II-6. Définition des méthodes:

Les classes possèdent en plus des attributs des fonctions aussi. Pour définir ces fonctions on suit la même syntaxe habituelle.
Cependant, il ya deux types de méthodes:

- **Méthodes statiques:** Ce sont des méthodes partagées par tous les objets. Elles appartiennent à la classe et non aux instances.
- **Méthodes d'instances:** Ce sont des méthodes qui sont propre à chaque objet.

La création d'une méthode se fait par la syntaxe suivante:

```
class nom:
        def methode_instance(self,param1,param2,...):
                "Traitement "
        @classmethod
        def Metode_Static(cls):
                "Traitement"
```

Exemple:

```
class A:
        nb = 0
        def __init__(self):
            print("creation objet de type A")
            A.nb = A.nb + 1
            self.x=1
            self.y=8
            print("il y en a maintenant ", A.nb)
        def deplace(self, dx, dy):
            self.x = self.x + dx
            self.y = self.y + dy

        @classmethod
        def get_nb(cls):
            return A.nb
print("Partie 1 : nb objets = ", A.get_nb())
a = A()
a.deplace(5,8)
print("Partie 2 : nb objets = ", A.get_nb())
b = A()
print("Partie 3 : nb objets = ", A.get_nb())
print("le deplacement de A est X=",a.x," Y=",a.y)
```

II-7. Définition d'attributs privés et public et de classe :

Il ya deux types d'attributs qui sont liés à l'instance:

- **Privé:** Ce type d'attribut est visible uniquement dans l'objet.il est déclaré suivant la syntaxe suivante:

```
self.__Nom=valeur
```

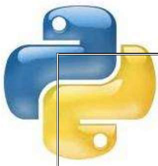
- **Public:** Ce type d'attribut est visible à l'intérieur et à l'extérieur. .il est déclaré suivant la syntaxe suivante:

```
self.Nom=valeur
```

- **Attribut de classe:** Cet attribut est statique. Il est lié à la classe et non aux objets. Il suffit de mettre la variable directement derrière la déclaration de la classe.

```
class nom:
    var_class=valeur
```

Il faut donc disposer de méthodes qui vont permettre par exemple de modifier ou d'afficher les informations associées à ces variables.



Exemple:

```
class Point:
 nb=80 #Variable statique
 def __init__(self, x, y):
     self.__x = x
     self.__y = y

  def deplace(self, dx, dy):
     self.__x = self.__x + dx
     self.__y = self.__y + dy

  def affiche(self):
     print("abscisse =", self.__x, "ordonnee =", self.__y)

a = Point(2, 4)
a.affiche()
a.deplace(1, 3)
a.affiche()
```

II-8. accesseurs et mutateurs :

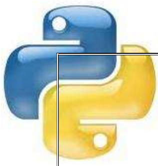
Parmi les différentes méthodes que comporte une classe, on a souvent tendance à distinguer :

- les **constructeurs**.
- les **accesseurs** (en anglais *accessor*) qui fournissent des informations relatives à l'état d'un objet, c'est-à-dire aux valeurs de certains de ses attributs (généralement privés) sans les modifier ;
- les **mutateurs** (en anglais *mutator*) qui modifient l'état d'un objet, donc les valeurs de certains de ses attributs.

On rencontre souvent l'utilisation de noms de la forme get_XXXX() pour les accesseurs et set_XXXX()pour les mutateurs, y compris dans des programmes dans lesquels les noms de variable sont francisés. Par exemple, pour la classe Point sur laquelle nous avons déjà travaillé on peut définir les méthodes suivantes :

Exemple :

```
class Point:
  def __init__(self, x, y):
     self.__x = x
     self.__y = y
  def get_x(self):
     return self.__x
  def get_y(self):
     return self.__y
  def set_x(self, x):
     self.__x = x
  def set_y(self, y):
     self.__y = y
```




```
a = Point(3, 7)
print("a : abscisse =", a.get_x())
print("a : ordonnee =", a.get_y())
a.set_x(6)
a.set_y(10)
print("a : abscisse =", a.get_x())
print("a : ordonnee =", a.get_y())
```

III. L'héritage de classe :

L'Héritage est le mécanisme fondamental pour faire évoluer un modèle, un programme. Il possède les propriétés suivantes:

- simple au multiple : une classe peut dériver d'une ou plusieurs classes de base.
- Souvent arborescent : une classe dérivée peut à son tour être une classe de base pour une autre dérivation (hiérarchie de classes).
- N'est pas réflexif : une classe ne peut pas hériter d'elle-même.

· La classe dérivée : ne peut pas accéder aux membres privés de la classe de base

- possède ses propres attributs/méthodes, que la classe de base ne connaît pas peut redéfinir (améliorer, spécialiser....) les méthodes héritées de la classe de base.

La syntaxe de l'héritage est:

```
class DerivedClassName(Classe_de_base):
        pass
```

Exemple :

```
class Person:
    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last
    def Name(self,num):
        return self.firstname + " " + self.lastname + " " + num

class Employee(Person):
    def __init__(self, first, last, staffnum):
        Person.__init__(self,first, last)
        self.staffnumber = staffnum
    def GetEmployee(self,num):
        return self.Name(num) + ", " + self.staffnumber
    def Getemp(self,nm):
        return Person.Name(self,nm)

x = Person("Marge", "Simpson")
y = Employee("Homer", "Simpson", "1007")

print(x.Name("ab"))
print(y.GetEmployee("cd"))
print(y.Getemp("2"))
```