Chapitre 1: Objet et Classe

Mr. CHAOULID

1. Encapsulation

- L'encapsulation est un concept important de la Programmation Orientée Objet.
- L’encapsulation n’autorise pas l’accès direct aux attributs de l’objet.
- L'objectif principal de l'encapsulation est de restreindre l'accès aux détails internes d’un objet et de permettre un contrôle plus précis sur l'utilisation de ses attributs et méthodes depuis l'extérieur.
- Les attributs d'une classe sont déclarés privés alors que ses méthodes et propriétés sont déclarées publiques. Cela signifie que l'utilisateur d'un objet (un autre programmeur) n'aura pas accès directement aux données privées de l'objet.
- Pour préciser explicitement la visibilité de chaque attribut et de chaque méthode:
  - **Public** : accessible depuis l’extérieur de la classe
  - **Private** : accessible uniquement à l’intérieur de la classe
  - **Protected** : accessible uniquement aux classes dérivées.

1. Encapsulation

- En Python, l'encapsulation est souvent mise en œuvre en utilisant des conventions de nommage et en définissant certains attributs et méthodes comme étant privés ou protégés.

  - **Attributs privés** : Les attributs qui sont préfixés par deux traits de soulignement (double underscore) sont considérés comme privés et ne devraient normalement pas être accessibles directement depuis l'extérieur de la classe.
  - **Attributs protégés** : Les attributs préfixés par un seul trait de soulignement sont considérés comme protégés, ce qui signifie qu'ils ne devraient pas être accédés directement depuis l'extérieur de la classe, mais cela n'est pas strictement interdit.

1. Encapsulation

- **Exemple :**

```
class Personne:
    #Constructeur de la classe
    def __init__(self,n,p,a):
        self.__nom=n     # attribut privé nom
        self.__prenom=p  # attribut privé nom
        self.__age=a     # attribut privé nom
    #Méthode publique de la classe
    def SePresenter(self):
        print("Je m'appelle",self.__nom,self.__prenom," et j'ai",self.__age," ans")
```

- Si les attributs sont verrouillés, on ne peut plus y avoir accès de l'extérieur de la classe. Il faut donc créer des méthodes publiques dédiées à l'accès pour chacun d'eux.

2. Méthodes d’accès

Pour accéder ou modifier des attributs privés depuis l'extérieur de la classe, on peut utiliser des méthodes publiques spéciales appelées "**getters**" (pour obtenir la valeur) et "**setters**" (pour définir la valeur).

Exemple:

```
class Personne:
    def __init__(self,n,p,a):
        self.__nom=n
        self.__prenom=p
        self.__age=a
    #getter de nom
    def get_nom(self):
        return self.__nom

    #setter de nom
    def set_nom(self,n):
        self.__nom= n

    def SePresenter(self):
        print("Je m'appelle",self.__nom,self.__prenom," et j'ai",self.__age," ans")


p1= Personne("Mohamedi","Ali",20)
p1.SePresenter()
print(p1.get_nom())   #Appel du getter de nom
p1.set_nom('Ahmadi')  #Appel du setter de nom
p1.SePresenter()
```

3. Les propriétés

- En Python, les propriétés (**properties**) sont une manière élégante d'implémenter des méthodes d'accès (getters et setters) pour les attributs d'une classe.
- L'utilisation de propriétés facilite l'encapsulation en permettant un accès contrôlé aux attributs.

Exemple:

```
class Personne:
    def __init__(self,n,p,a):
        self.__nom=n
        self.__prenom=p
        self.__age=a

    @property             #propriété pour l'attribut âge
    def age(self):
        return self.__age
    @age.setter
    def age(self,a):
        if(a>0):
            self.__age=a
        else:
            print("Age invalide")

    def SePresenter(self):
        print("Je m'appelle",self.__nom,self.__prenom," et j'ai",self.__age," ans")


p1= Personne("Mohamedi","Ali",20)
p1.SePresenter()
print(p1.age)   #Appel de la propriété âge (getter)
p1.age=30       #Appel de la propriété âge (setter)
p1.SePresenter()
p1.age=-20      #Appel de la propriété âge (setter)
p1.SePresenter()
```

4. Attribut statique

- un attribut statique est un attribut associé à une classe plutôt qu'à une instance particulière de cette classe. Cela signifie que cet attribut est partagé entre toutes les instances de la classe et peut être accédé via la classe elle-même ou via une instance de la classe.
- Pour déclarer un attribut statique en Python, on peut simplement le définir directement au niveau de la classe, en dehors de toute méthode.
- **Exemple :**

```
class Personne:
    pays="Maroc"      #attribut statique
    def __init__(self,n,p,a):
        self.__nom=n
        self.__prenom=p
        self.__age=a

    def SePresenter(self):
        print("Je m'appelle",self.__nom,self.__prenom," et j'ai",self.__age," ans")


p1= Personne("Mohamedi","Ali",20)
p2= Personne("Ahmadi","Karim",30)
p1.SePresenter()
p2.SePresenter()
print(Personne.pays)     #accés à l'attribut statique en lecture
Personne.pays='France'   #accés à l'attribut statique en écriture
print(p1.pays)
print(p2.pays)
```

5. Méthode statique

- Une méthode statique est une méthode associée à une classe plutôt qu'à une instance spécifique de cette classe. Elle est souvent utilisée pour des opérations qui ne dépendent pas de l'état d'une instance particulière, mais qui sont liées à la classe dans son ensemble.
- Pour déclarer une méthode statique en Python, on utilise le décorateur **@staticmethod.**
- **Exemple :**

```
class Personne:
    __pays="Maroc"      #attribut statique privé
    def __init__(self,n,p,a):
        self.__nom=n
        self.__prenom=p
        self.__age=a

    @staticmethod      #Méthode statique
    def getpays():
        return Personne.__pays

    @staticmethod      #Méthode statique
    def setpays(p):
        Personne.__pays=p

    def SePresenter(self):
        print("Je m'appelle",self.__nom,self.__prenom," et j'ai",self.__age," ans")


p1= Personne("Mohamedi","Ali",20)
p2= Personne("Ahmadi","Karim",30)
print(Personne.getpays())     #Appel de la méthode statique getpays
Personne.setpays('France')    #Appel de la méthode statique setpays
print(p1.getpays())
print(p2.getpays())
```

TP2

Mr. CHAOULID