concept de classe (2)

Lorsque l'on définit une classe comme la classe cycle, par exemple, l'accès aux attributs radius et color peut se faire directement à l'aide de l'opérateur d'accès (.)

```
class cycle:
    def __init__(self, radius, color): #Constructeur de la classe cycle
        self.radius = radius
        self.color = color

c = cycle(700, "green")   #Instanciation de l'objet c
print(c.color)          #accès à l'argument radius de l'objet c
```

>>> "green"

Mécanisme d'encapsulation

Dans certains cas, il peut s'avérer souhaitable que certains attributs de l'objet créé ne soient pas accessibles depuis l'extérieur de la classe. L'encapsulation est un mécanisme adapté, qui permet alors de cacher ou de protéger les données en question.

Imaginons que l'on souhaite ne plus rendre directement accessible l'attribut color au-dehors de la classe cycle. Dans ce cas, il convient de créer un attribut noté _color, le underscore _ signalant le caractère "privé" de l'attribut.

```
class cycle:
    def __init__(self, radius): #Constructeur de la classe cycle
        self.radius = radius
        self._color = "green"

c = cycle(700)          #Instanciation de l'objet c
```

Pour accéder à la couleur de l'instance c de la classe cycle, il faudrait insérer le code print(c._color). Ce qui fonctionnerait parfaitement. Cependant, la convention consistant à mettre un underscore _ suppose que la variable _color est privée et réservée à un usage interne dans la définition de la classe cycle.

Comment dès lors accéder à la valeur de l'attribut color et/ou comment modifier celle-ci ?

Accesseur et property

Pour accéder à la valeur de l'attribut protégé color, il est nécessaire d'implémenter une méthode spéciale, dite accesseur, donnant directement accès à cette valeur, puis d'affecter à l'attribut color par la méthode property la valeur de l'attribut privé _color. Quand on veut accéder à l'attribut color, Python tombe sur property qui redirige vers la méthode _get_color qui renvoie l'attribut protégé _color.

```
class cycle:
    def __init__(self, radius): #Constructeur de la classe cycle
        self.radius = radius
        self._color = "green"

    def _get_color(self):
```

```
        return self._color

    color = property(_get_color)

c = cycle(650)
print(c.color)
```

>>> "green"

Le problème qui se pose alors est celui de la modification de l'attribut color de l'objet instancié. Il est impossible de taper la ligne de code `c.color = "red"` pour obtenir une modification de l'attribut color inaccessible car protégé. La solution consiste en l'utilisation d'une méthode appelée mutateur permettant de modifier l'attribut.

Mutateur

Un mutateur est une méthode spéciale utilisée dans une classe pour permettre la modification d'un attribut protégé, en association avec les méthodes _get_ et property.

```
class cycle:
    def __init__(self, radius): #Constructeur de la classe cycle
        self.radius = radius
        self._color = "green"

    def _get_color(self):
        return self._color

    def _set_color(self, newcolor): #Définition du mutateur
        self._color = newcolor

    color = property(_get_color, _set_color)  #L'accesseur est toujours
suivi du mutateur

c = cycle(650)
print(c.color)
c.color = "red"
print(c.color)
c.color = "blue"
print(c.color)
```

Affichage obtenu dans la console :
green
red
blue

Descripteur

Le mécanisme qui utilise un accesseur, un mutateur et la méthode property est appelé descripteur. Ce mécanisme permet de protéger des données ou de contrôler celles-ci en paramétrant d'une manière adaptée le mutateur.

Exercice

Construire une classe Personne qui comporte les attributs publics nom et prenom, privé non protégé age = 16 et privé protégé lieu_residence (valeur par défaut : " ").

Correction de l'exercice

```python
class Personne:

    def __init__(self, nom, prenom): #Constructeur de la classe Personne
        self.nom = nom
        self.prenom = prenom
        self.age = 16
        self._lieu_residence = ""

    def _get_lieu_residence(self):
        return self._lieu_residence

    def _set_lieu_residence(self, nouveau_lieu_residence):
        self._lieu_residence = nouveau_lieu_residence

    lieu_residence = property(_get_lieu_residence, _set_lieu_residence)

E1 = Personne("DUPONT","Edouard") #Création d'une instance de la classe Personne
print(f"Monsieur {E1.prenom} {E1.nom} est un élève de {E1.age} ans.")
E1.lieu_residence = input("Quel est son lieu de résidence ? ")
print(f"Son lieu de résidence est {E1.lieu_residence}")
```