


Le principe de l'encapsulation

PLAN

- Le principe d'encapsulation
- Les modificateurs d'accès
- Les getters et les setters

L'encapsulation

- Le mot encapsulation vient du mot capsule 
- Le rôle d'une capsule est de protéger son contenu
- L'encapsulation consiste à protéger **l'information contenue dans un objet**
- L'encapsulation consiste à **protéger** ou à **cacher** les données de certains objets.

L'encapsulation

- L'encapsulation en pratique, c'est le fait de rendre un élément **privé (private)**, que ce soit un **attribut** ou même une **méthode** .
- Un attribut privé ne peut être utilisée qu'à l'intérieur de la classe où il est définit.
- Une méthode privée aussi est une méthode qui n'est accessible que à l'intérieur de la classe où elle a été définit.

Les modificateurs d'accès

- Les modificateurs d'accès sont des mots clés utilisés pour restreindre l'accès aux variables et aux méthodes de la classe.
- La plupart des langages de programmation ont trois modificateurs d'accès, qui sont :
 1. **Public** : un élément (attribut ou méthode) public est un élément qui **peut être utilisé à l'intérieur de la classe ou en dehors de la classe** où elle a été défini.

Tous les attributs et méthodes membres d'une classe sont **publics par défaut**.
 2. **Private** : un élément (attribut ou méthode) privé n'est utilisé **qu'à l'intérieur** de la classe où ils sont définis.
 3. **Protected** : (à voir plus tard)

Modificateur d'accès public en python

```
class Voitures:

    def __init__(self,m1="",m2="",p=0) :
        #attributs public
        self.matricule=m1
        self.marque=m2
        self.prix=p

    #méthode public
    def Infos(self):
        #accès aux attributs public
        print("la matricule :",self.matricule, " la marque :",self.marque,

v1=Voitures("A1254","ford",90000)
#on peut accéder en dehors de la classe à l'attribut public matricule
print(v1.matricule)
#on peut accéder en dehors de la classe à la méthode public Infos
v1.Infos()
```

Modificateur d'accès privé en python

- Le modificateur d'accès privé est le modificateur d'accès le plus **sécurisé**.
- Les membres de données d'une classe sont déclarés privés en ajoutant un **double trait de soulignement** `'__'` avant **le membre de données de cette classe**.

Modificateur d'accès privé en python

```
class Voitures:
    def __init__(self,m1="",m2="",p=0) :
        #attributs d'instance privés
        self.__matricule=m1
        self.__marque=m2
        self.__prix=p

    def Infos(self):
        #accès aux attribut privé à l'interieur
        print("la matricule :",self.__matricule, " la marque :",
              #print(Voitures.fournisseur)

v1=Voitures("A1254","ford",80000)
print(v1.__matricule)
```

L'attribut est accessible mais il ne le connaît pas lors de l'exécution

Modificateur d'accès privé pour les méthodes

```
class Voitures:
    def __init__(self,m1="",m2="",p=0) :
        #attributs d'instance privés
        self.__matricule=m1
        self.__marque=m2
        self.__prix=p

    def __Inofs(self):
        #accès aux attribut privé à l'interieur
        print("la matricule :",self.__matricule, " la marque :",
        #print(Voitures.fournisseur)

v1=Voitures("A1254","ford",80000)
print(v1.__Inofs())
```

La méthode est accessible par l'objet mais lors de l'exécution il ne la connaît pas

Modificateur d'accès privé pour les méthodes

- Lorsque on mets nos attributs ou certaines de nos méthodes privée, c'est-à-dire qu'on applique le principe de **l'encapsulation**
- Mais comme déjà vu, les éléments privés sont accessibles en dehors de la classe, même si ils ne sont pas connus lors de l'exécution, dans d'autres langages OO , un élément privé n'est même pas accessible en dehors de la classe.

Question

- Si les attributs sont privé et ne sont pas utilisés en dehors de la classe, comment pourrais-je modifier la valeur d'un attribut?
- Pour cela , on se sert de certaines méthodes spéciales, qu'on appelle, **les getters et les setters**.

Les getters (les accesseurs)

- Est une méthode permettant de retourner la valeur d'un attribut privé.

```
def getMatricule(self):  
    return self.__matricule
```

- Chaque attribut privé, on peut lui créer un getter (accesseur)
- Pour son appel, c'est comme n'importe quelle méthode:

```
v1.getMatricule()
```

Les setters (modificateurs)

- Est une méthode permettant de modifier la valeur d'un attribut privé:

```
def setMatricule(self,m):  
    self.__matricule=m
```

- Chaque attribut privé, on peut lui créer un setter (modificateur)
- On appelle un setter comme suit:

```
a=input("Saisir la nouvelle matricule ")  
v1.setMatricule(a)
```