Les propriété en python

Plan

- La fonction property()
- Le décorateur @property

Avantage des propriétés

- Les propriétés vont nous permettre d'accéder aux attributs d'instances qui sont déclaré privé, ils jouent le rôle des getters et les setters , sauf que les **propriétés ne sont pas des fonctions**.
- L'avantage des propriétés par rapport aux getters et setters c'est d'utiliser un **seul mot**, comme étant un attribut
- Il y'a deux manières de définir une propriété:
  1. En utilisant la fonction **property()**
  2. En utilisant le décorateur **@property**

La fonction property()

- La méthode **property**() prend les méthodes get, set et delete comme arguments et renvoie un objet de la classe property.

- La syntaxe de la fonction **property**() :

property (fget, fset, fdel,doc)

  - **fget** est une fonction pour récupérer la valeur de l'attribut
  - **fset** est une fonction pour définir la valeur de l'attribut
  - **fdel** est une fonction pour supprimer la valeur de l'attribut
  - **doc** est une chaîne contenant la documentation de l'attribut

Exemple d'utilisation

```
class Person:
    def __init__(self):
        self.__age = 10

    def setAge(self, age):
        self.__age = age

    def getAge(self):
        return self.__age

    age = property(getAge, setAge)

p = Person()
p.age = 46
print(p.age)
```

La fonction **property**() possède ici deux paramètres qui sont les deux fonctions: setAge et getAge

Appel automatique du setter : setAge

Appel automatique du getter: getAge

Exemple d'utilisation

```
class Person:
    def __init__(self):
        self.__age = 10

    def setAge(self, age):
        self.__age = age

    def getAge(self):
        return self.__age

    def delAge(self):
        del self.__age

    age = property(getAge, setAge, delAge)

p = Person()
p.age = 40
print(p.age)
del p.age
print(p.age)
```

Ici on a ajouté la méthode **delAge**

Supprimer l'age et appel de la méthode **delAge**

Le décorateur @property

- **@property** est un décorateur qui évite d'utiliser la fonction getter explicite: c'est-à-dire que ce décorateur permet de **définir une propriété qui retourne la valeur d'un attribut privé**

- **@Nompropriété.setter :** est un décorateur qui évite d'utiliser la fonction setter explicite: c'est-à-dire qu'elle permet d'utiliser **le même nom de la propriété mais comme un setter, donc permet de modifier la valeur d'un attribut privé.**

Exemple d'utilisation

```
class Person:
    def __init__(self):
        self.__age = 10

    @property
    def Age(self):
        return self.__age

    @Age.setter
    def Age(self, age):
        self.__age = age

p = Person()
p.Age = 40
print(p.Age)
```

On définit une propriété **Age** qui retourne la valeur de l'attribut privé __age (joue le rôle d'un getter)

On définit une propriété **Age** qui modifie la valeur de l'attribut privé __age (joue le rôle d'un setter)

Appel de la propriété **Age** comme getter et setter

Exemple

```python
class Person:
    def __init__(self):
        self.__age = 25


    @property
    def Age(self):
        return self.__age

    @Age.setter
    def Age(self, age):
        if age>=18 :
            self.__age = age



p = Person()
p.Age = 10
print(p.Age)
```

On peut ajouter des conditions ou des traitements dans le setter selon le cas qu'on a à étudier.

Exemple :

Si la valeur de âge est supérieur à 18 , il va être affecté à l'attribut __age
Mais si la valeur est inférieur à 18, la valeur de l'attribut __age sera la valeur par défaut qui est **25**

Vérifier la valeur du paramètre **age** avant de l'affecter à notre attribut __age

Les propriétés

- Dans une classe, tous les attributs privés doivent avoir des propriétés correspondantes (getter et setter)
- Il existe parfois des **propriété dites en lecture seule:** c'est une propriété qui permet de retourner la valeur d'un attribut privé mais ne fait pas la modification (getter uniquement)
- On parle aussi de **propriété en écriture seule**: propriété qui permet de modifier la valeur d'un attribut sans possibilité de la retourner (setter uniquement)