**Getter dan Setter**

Sebelumnya kita pernah membuat getter dan setter tapi kita membuatnya seperti di java. Pada kali ini kita akan membuat getter dan setter yang di pakai di python.ini akan berbeda tapi kelakuannya akan sama dan hasilnya akan lebih bagus. berikut cara membuatnya :

Berikut kode standar yang kita buat :

class Hero:

    def \_\_init\_\_(self, name, health, armor) -> None:

*self*.\_\_name = name

*self*.\_\_health = health

*self*.\_\_armor = armor

*self*.info = f"name {*self*.\_\_name} : \n\thealth {*self*.\_\_health}"

    def getHealth(self):

        return *self*.\_\_health

sniper = Hero("sniper", 100, 10)

print(sniper.info)

ketika kode di atas di jalankan, maka berikut ouput yang di hasilkan :

***name sniper :***

***health 100***

yang menjadi masalah adalah ketika kita mencoba merubahnya seperti contoh perintah berikut :

sniper.info = 'info'

maka info itu akan otomatis berubah. Seharusnya user tidak bisa mengubahnya. Di python ada namanya decorator property. Kan sebelumnya kita sudah menggunakan decorator staticmethod dan classmethod, sekarang kita akan menggunakan decorator proprty. Sama seperti staticmethod dan classmethod, jika ingin menggunakannya hanya perlu menggunakan tanda @. Berikt perintahnya :

@property

Artinya adalah membuat method menjadi seperti variable. Berikut caranya :

    @property

    def info(self):

        return f"name {*self*.\_\_name} : \n\thealth {*self*.\_\_health}"

kode di atas akan membuat method **info** di anggap sebagai **variable**.dan anda bisa menghapus kode berikut yang ada di \_\_init\_\_ :

*self*.info = f"name {*self*.\_\_name} : \n\thealth {*self*.\_\_health}"

ketika anda menjalankan kode di atas dengan perintah berikut :

print(sniper.info)

maka hasilnya akan sama seperti sebelumnya, yaitu :

***name sniper :***

***health 100***

tapi bedanya anda tidak dapat mengubah isinya dengan cara yang tadi. Kalau anda tetap memaksa mengubahnya dengan cara yang tadi atau cara berikut :

sniper.info = "info"

maka hasilnya akan error.

**Getter and setter**

Sebelum kita masuk pada getter anda setter untuk encapsulasi itu, kita harus menganggap variable yang akan kita ambil sebagai property. Berikut contoh :

**getter**

    @property

    def armor(self):

        pass

ketika anda menajalankan kode di atas dengan perintah berikut

print(sniper.armor)

maka hasilnya :

***None***

**setter**

    @armor.getter

    def armor(self) :

        return *self*.\_\_armor

kode di atas harus di taruh di bawah kode getter di atas. Ketika kode berikut di jalankan :

print(sniper.armor)

maka outputnya adalah sebagai berikut :

***10***

Jika anda ingin merubah nilai armornya maka anda harus membuat kode setter di bawah kode getternya. berikut kode untuk membuatnya:

    @armor.setter

    def armor(self, input):

*self*.\_\_armor = input

jadi ketika anda ingi mengubah nilainya ubah dengan perintah seperti anda mengubah variabel pada umumnya. berikut contohnya :

sniper.armor = 50

jika anda menggunakan perintah berikut untuk mengubah nilainya

print(sniper.armor(60))

maka hasilnya akan error. Jadi dengan cara di atas kita akan lebih mudah untuk mengelolahnya.

**deleter**

di deleter kita akan membuatnya mendelete nilai yang telah ada atau yang telah kita masukkan. Berikut cara penulisannya ( tulis dibawah setter ):

    @armor.deleter

    def armor(self):

        print("armor di delete")

*self*.\_\_armor = None

berikut perintah untuk mendelete armor :

del sniper.armor

ketika anda menjalankannya, maka nilai atau value armor adalah None.

Kegunannya adalah ketika kita tidak membutuhkan armor, kita bisa langsung mendeletenya sehingga memori kita menjadi lebih lega atau luas.

Jadi di atas adalah encapsulasi dengan cara python, sebekumnya kita hanya meggunakan cara paradikma OOP biasa.