Herencia de datos Públicos, Privados y Protegidos

Type	Exposicion
Reviewed	☐

Herencia de datos Públicos Privados y Protegidos en Python

Introducción

Herencia, Nos ayuda a que una clase hija reutilice código de una clase padre

Elementos públicos = al publico?? libre interacción al, pueden ser modificados facilmente

Elementos privados = se utilizan para proteger los datos, se solicita acceso a la clase en donde lo estas utilizando

Elementos protegidos = solo se accede desde la clase en la que son definidos y de sus clases derivadas, puedes modificarlo si tienes permiso de una clase hija

Como se diferencias los elementos públicos, se diferencian de las demas por que no tienen ningún guion bajo **(solo un guion bajo) ("_")**

Como se diferencia el elemento protegido, solo se puede acceder desde la misma clase y sus clases hijas **(doble guion bajo) ("__")**

metodo protegido??

decorador property ??

setters y getters en python

1. **¿Cual es la diferencia entre atributos publico, protegidos y privados?**

   La diferencia es su accesibilidad en el código, públicos pueden ser accesibles desde cualquier parte del código mientras que los protegidos solo desde la clase donde se declaran y subclases o clases hijas, y los atributos privados, no pueden ser accesados desde ningún otro lugar que no sea la misma clase donde son declarados

2. **¿Por que es importante usar atributos privados?**

   El uso de atributos privados facilitaria para los datos que no queremos que sean accesibles desde ningún otro lado en el código mas que en la clase en donde son implementados, resguardando así información sensible y que no pueda ser editada

3. **¿Puedo acceder a un atributo privado desde una subclase?**

   No no se puede accesar a los atributos privados desde ningún otro lado en el código mas que en sus propios métodos de la clase en la que se declaran

4. **¿Cual es la diferencia entre un atributo privado y uno protegido en una clase?**

   La diferencia principal es su acceso desde las diferentes partes del código, los elementos **protegidos** tienen un acceso *solo* desde la clase en donde son declarados y desde clases hijas que estén utilizando su herencia, a diferencia de los elementos **privados** los cuales solo son accesibles desde la clase declarada y sus métodos.

5. **¿Cual es la diferencia de poner o no el @property?**

La diferencia radica totalmente en su funcionamiento, ya que si no se especifica el @ el programa no podrá identificarlo como este método especial que es.

Ejercicios

1. Crear una clase llamada `Animal` que tenga el atributo nombre y un metodo `hacer_sonido` que devuelva “Sonido Generico”. Luego, crea una subclase llamada `Perro` que sobreescriba el metodo `hacer_sonido` para devolver “Guau Guau” y otra subclase llamada Gato que lo sobreescriba para devolver “Miau”. Finalmente, crea instancias de las clases `Perro` y `Gato`, asigna nombres e imprime el nombre y sonido que hace cada uno.

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hacer_sonido(self):
        print("Sonido Generico")


class Perro(Animal):
    def hacer_sonido(self):
        print(f"{self.nombre} hace: Guau, Guau")


class Gato(Animal):
    def hacer_sonido(self):
        print(f"{self.nombre} hace: Miau")


perro1 = Perro("robin")
gato1 = Gato("willow")
```

```
perro1.hacer_sonido()
gato1.hacer_sonido()
```

2. Crea una programa con una clase `Persona` que tenga un atributo publico nombre, uno protegido `_edad` y uno privado `_identifiacion` .

   Agrega método público `mostrar_datos` para mostrar el nombre y la edad. Luego, crea una clase hija `Empleado` que herede de `Persona`, agregue un atributo puesto y tenga un método mostrar información que use los datos de la clase padre. Crea una instancia de `Empleado` y prueba el acceso a los atributos y métodos de cada nivel de acceso.

```
class Persona:
    def __init__(self, nombre, _edad):
        self.nombre = nombre #Público
        self.edad = _edad    #Protegido
        self.__identificacion = "12345"  #Privado

    def mostrar_datos(self):
        return ("Nombre: (self.nombre), Edad: {self. edad}")

    def mostrar_identificacion(self):
        return f"ID: {self.__identificacion}"


class Empleado(Persona):
    def __init__(self, nombre, edad, puesto):
        super().__init__(nombre, edad)
        self.puesto = puesto #Publico

    def mostrar_informacion(self) :
        #Acceso a atributos públicos y protegidos
        return f"{self.mostrar_datos()}, Puesto: {self.puesto}"
```

```
# Crear una instancia de Empleado
empleado = Empleado("Iván", 30, "Desarrollador")
print(empleado.mostrar_informacion())

# Intentar acceder a los atributos protegidos y privados
print(empleado._edad) # Acceso permitido auque no recomendado
# print(empleado.__identifiacion)
```