# Python Programming

Módulo 3

# Encapsulamiento

# Encapsulamiento

El encapsulamiento es un concepto relacionado con la programación orientada a objetos, y hace referencia al ocultamiento de los estados internos de una clase al exterior. Dicho de otra manera, encapsular consiste en hacer que los atributos o métodos internos a una clase no se puedan acceder ni modificar desde fuera, sino que tan solo el propio objeto pueda acceder a ellos.

Para la gente que conozca otros lenguajes, le resultará un término muy familiar, pero en Python es algo distinto.

Python por defecto no oculta los atributos y métodos de una clase al exterior.

```
class Clase:
    atributo_clase = "Hola"
    def __init__(self, valor):
        self.atributo_instancia = valor

mi_clase = Clase("Que tal")
print(mi_clase.atributo_clase) # 'Hola'
print(mi_clase.atributo_instancia) # 'Que
tal'
```

Ambos atributos son accesibles desde el exterior. Sin embargo, esto es algo que tal vez no queremos que suceda con ciertos métodos o atributos.

La encapsulación consiste en denegar el acceso a los atributos y métodos internos de la clase desde el exterior. En Python no se aconseja para no complejizar el código. Pero se puede simular.

Si que queremos que los atributos y métodos solo pertenezcan a la clase, y solamente puedan ser accedidos por la mecánica interna del objeto, podemos usar la doble __ (doble guion bajo) para nombrarlos. Esto hará que Python los interprete como “privados”, de manera que no podrán ser accedidos desde el exterior.

```
class Clase:
    atributo_clase = "Hola"   # Accesible desde el exterior
    __atributo_clase = "Hola" # No accesible

    # No accesible desde el exterior
    def __mi_metodo(self):
        print("Haz algo")
        self.__variable = 0

    # Accesible desde el exterior
    def metodo_normal(self):
        # El método si es accesible desde el interior
        self.__mi_metodo()

objeto = Clase()
print(objeto.atributo_clase)     # Ok!
objeto.metodo_normal()    # Ok!
#print(objeto.__atributo_clase)  # Error! Atributo no accesible
#objeto.__mi_metodo()     # Error! Método no accesible
```

# Acceder a atributos privados (GET-SET-DEL)

Muchas veces, el público que decide aprender Python viene con un bagaje de conocimiento de otro lenguaje. Y piensan como si Python funcionase igual que: C, C++, Java, etc. Y eso en algunos aspectos es válido y en otros no tanto. Python es Python. Java es Java.

En Python cuando queremos acceder a atributos privados podemos fabricarnos los métodos *get, set* y *del.*

Pero hay otra alternativa al uso de estos métodos basada en las propiedades Python que simplifica la tarea. Las propiedades en Python son un tipo especial de atributo a los que se accede a través de llamadas a métodos. Con ello, es posible ocultar los métodos *"get", "set"* y *"del"* de manera que sólo es posible acceder mediante estas propiedades por ser públicas.

No es obligatorio definir métodos *"get"*, *"set"* y *"del"* para todas las propiedades. Es recomendable sólo para aquellos atributos en los que sea necesario algún tipo de validación anterior a establecer, obtener o borrar un valor. Para que una propiedad sea sólo de lectura hay que omitir las llamadas a los métodos "set" y "del".

```
class Empleado:
    def __init__(self, nombre, salario):
        self.__nombre = nombre
        self.__salario = salario

    def getnombre(self):
        return self.__nombre

    def getsalario(self):
        return self.__salario

    def setnombre(self, nombre):
        self.__nombre = nombre

    def setsalario(self, salario):
        self.__salario = salario

    def delnombre(self):
        del self.__nombre

    def delsalario(self):
        del self.__salario
```

Vemos que nombre y salario son valores que se pasan al momento de generar la instancia. Pero luego esos valores son “privados”. ¿Entonces?

Si se quieren manipular, vamos a tener que armarnos métodos para poder manipularlos ya que internamente vamos a poder seguir trabajando con ellos.

En este caso los métodos *“get”* nos permiten obtener los valores, los *“set”* grabar valores y los *“del”* borrar.

¿Pero esto es lo que se busca en Python? No. Básicamente la filosofía del lenguaje es: “Simple es mejor que complejo”, no te compliques la vida.

La clase empleado sin tanta parafernalia:

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

No sólo es más corta, sino que su funcionalidad es completamente equivalente, es más fácil de leer porque es obvia (se puede leer de un vistazo), y hasta es más eficiente.

Lo único diferente es que en esta última versión los atributos son públicos y modificables sin ningún método especial. Como vimos en la primera parte teórica del módulo.

# ¡Muchas gracias!

¡Sigamos trabajando!