**ENCAPSULAMIENTO EN PYTHON**

Se conoce como atributos encapsulados a los atributos que “no deberían ser modificados desde fuera de la clase”. Esto puede “indicarse” o señalizarse anteponiendo un \_ al nombre del atributo en la declaración del mismo.

Aunque podría accederse desde fuera de la clase al atributo, no es lo recomendable.

        self.\_nombre = nombre

        self.\_apellidos = apellidos

        self.\_edad = edad

Esta sería la forma de “indicar” que no se deberían modificar estos valores desde fuera de la clase.

La única forma de restringir este acceso en Python es anteponiendo un doble guion-bajo al atributo en la declaración de la clase.

class Persona:

    def \_\_init\_\_(self, nombre, edad) -> None:

        self.\_\_nombre = nombre

        self.\_\_edad = edad

    def mostrar\_detalles(self):

        print(f'{self.\_\_nombre, self.\_\_edad}')

stri = Persona('Agustin', 10)

stri.mostrar\_detalles()

stri.\_\_nombre = 'asd'

stri.mostrar\_detalles()

Por tanto este código no modifica el atributo **\_\_nombre**, y en la ultima impresión el atributo **\_\_nombre** se imprime con el valor **‘Agustin’.**

**ENCAPSULAMIENTO TOTAL**

Para lograr un encapsulamiento “total” podemos utilizar los decoradores @property y @attr.setter

Estos decoradores se establecen en la línea de código anterior a la definición del método **get** y **set** respectivamente.

Con estos, podemos acceder a los atributos con la sintaxis que se usaría para acceder a un atributo “directamente” pero no lo estaríamos haciendo, sino que llamaríamos a un método dentro de la clase que realiza las modificaciones. De este modo, no se modifican los atributos “por fuera de la clase”.

class Persona:

    def \_\_init\_\_(self, nombre, edad) -> None:

        self.\_nombre = nombre

        self.\_edad = edad

    def mostrar\_detalles(self):

        print(f'{self.\_nombre, self.\_edad}')

    @property

    def nombre(self):

        return self.\_nombre

    @property

    def edad(self):

        return self.\_edad

    @nombre.setter

    def nombre(self, nombre):

        self.\_nombre = nombre

    @edad.setter

    def edad(self, edad):

        self.\_edad = edad

stri = Persona('Agustin', 10)

print(stri.nombre)

stri.edad = 19

print(stri.edad)

Este es el código de ejemplo. Como se ve, los métodos get anteponen @property y los set, el nombre del atributo.setter.

Es importante, para que funcione correctamente que el método property y en el setter, se definan con el mismo nombre.

En las ultimas 3 lineas de código, no se esta accediendo a los atributos directamente, ya que eso se haría con **\_nombre o \_edad**. Lo que en realidad se hace es acceder a los métodos de la clase para realizar esta tarea y solo estos métodos pueden modificar los atributos en cuestión.

**ATRIBUTOS READ-ONLY**

SI quisiésemos que un attr fuese de solo lectura, solo bastaría con agregar el método getter con el decorador @property y omitir el setter, de este modo, al intentar acceder a este método, se encontrará con un error. Aunque todavía sería posible acceder desde el atributo directamente, la sintaxis no es la recomendada.

**MODULOS EN PYTHON**

Los modulos son secciones de código, como pueden ser una Clase dentro de un archivo.

Para importar un modulo desde otro archivo (para por ej utilizarle) se utiliza la siguiente sintaxis:

From **archivo** import **Modulo**

from Coche import Coche

puede usarse esta otra para importar todos los modulos:

from Coche import \*

Una complicación que puede surgir es que, una vez que importemos el modulo se ejecute en el archivo que importamos, código “de prueba” o innecesario del modulo importado.

Para solucionar esto, puede utilizarse la propiedad \_\_name\_\_ que devuelve el nombre del archivo en que se está ejecutando el archivo. En caso de que sea el actual, el valor de esta propiedad es \_\_**main\_\_** pero en caso de que no, se imprime el nombre del archivo.

Podríamos agg estas líneas en el modulo a importar para omitir este código

if \_\_name\_\_ == 'main':

    # codigo a omitir en importaciones