¿Qué es el encapsulamiento en Python?

El encapsulamiento en Python y en la POO en general, se trata de esconder partes internas de un objeto definido en el código, de tal forma que no cualquier persona pueda cambiarlas.

Esta forma de programar permite ocultar detalles internos de una clase y controlar cómo se accede y usan los atributos y métodos.

Sigo sin entender que significa

Un ejemplo sencillo puede ser como algún amigo que tiene un coche tuneado, tú lo puedes ver desde fuera, lo conduces y funciona, pero **NO** puedes meterle mano al motor ni a los cables porque seguro que lo rompes, y si quieres cambiar algo hay que pedírselo al dueño para evitar romper nada.

Si se pudiera cambiar de todo al coche, un duraría ni una semana, es por eso la idea del “encapsulamiento en el código”. Al evitar cambios brucos, se garantiza que el código funcione.

En el código de la siguiente página se explica paso a paso como se implementa esta idea del coche en un programa en Python usando métodos privados.

```
class Coche:
    def __init__(self, modelo):
        self.modelo = modelo
        self.__velocidad = 0        # 🔓 Velocidad actual
        self.__max_velocidad = 300  # 🔓 Tope máximo

    def acelerar(self, gas):
        if gas <= 0:
            print("No puedes acelerar con negativos ❌")
            return

        self.__velocidad += gas
        if self.__velocidad > self.__max_velocidad:
            self.__velocidad = self.__max_velocidad
            print(f"\n⚠️ - El {self.modelo} alcanzó el límite de {self.__max_velocidad} km/h y no puede más.\n")
        else:
            print(f"\n🚀 - El {self.modelo} acelera {gas} km/h | Velocidad actual: {self.__velocidad} km/h")

    def ver_velocidad(self):
        print(f"🔎 - El {self.modelo} va a {self.__velocidad} km/h\n")
        return self.__velocidad

    def iniciar_velocidad(self):
        print(f"\n🚦 - El {self.modelo} inicia desde 0 km/h")
        return self.__velocidad

# Coches
coche = Coche("Tesla Roaster")

coche.iniciar_velocidad()

coche.acelerar(50)

coche.acelerar(250)

coche.acelerar(100)
```

Lo que sucede en el código es que el coche tiene una velocidad máxima que no puede sobrepasar, de lo contrario se dañará, el coche inicia desde 0 km/h y a medida que el programa avanza, el coche aumenta su velocidad, en un punto del código, el coche se pone a la velocidad máxima e intenta sobrepasarla, pero el programa lo protege poniéndolo justo al límite máximo que soporta.

Explicación del código paso a paso

Este primer fragmento contiene la clase 'Coche', después se muestran sus atributos (características del objeto), podemos ver el modelo, la velocidad actual del coche y la velocidad máxima que alcanza, ambas velocidades son métodos privados que so se deben manipular.

```python
class Coche:
    def __init__(self, modelo):
        self.modelo = modelo
        self.__velocidad = 0          # 🔒 Velocidad actual
        self.__max_velocidad = 300    # 🔒 Tope máximo
```

Después tenemos este otro fragmento que define un método llamado 'acelerar', este método se usará para darle velocidad al coche usando 'gas', que es la cantidad de velocidad por km/h que se le añade al coche.

```python
def acelerar(self, gas):
    if gas <= 0:
        print("No puedes acelerar con negativos ❌")
        return

    self.__velocidad += gas
    if self.__velocidad > self.__max_velocidad:
        self.__velocidad = self.__max_velocidad
        print(f"\n⚠️  - El {self.modelo} alcanzó el límite de {self.__max_velocidad} km/h y no puede más.\n")
    else:
        print(f"\n🚀 - El {self.modelo} acelera {gas} km/h | Velocidad actual: {self.__velocidad} km/h")
```

Primero se asegura de que no se pueda acelerar el coche con números negativos ni con 0 km/h, porque el coche no avanzaría o se quedaría parado.

```
def acelerar(self, gas):
    if gas <= 0:
        print("No puedes acelerar con negativos ❌")
        return
```

Después se hace una suma de la velocidad actual del coche junto con 'gas', pero si al realizar la suma, la velocidad actual del coche supera la velocidad máxima permitida, el coche vuelve justo a la velocidad máxima permita, impidiendo que ésta suba más.

```
self.__velocidad += gas
if self.__velocidad > self.__max_velocidad:
    self.__velocidad = self.__max_velocidad
    print(f"\n⚠️  - El {self.modelo} alcanzó el límite de {self.__max_velocidad} km/h y no puede más.\n")
else:
    print(f"\n🚀 - El {self.modelo} acelera {gas} km/h | Velocidad actual: {self.__velocidad} km/h")
```

Después de todo este rollo, tenemos estos últimos 2 métodos que simplemente devuelven mensajes de la velocidad actual del coche y cuando el coche inicia desde 0 para posteriormente aumentar su velocidad.

```
def ver_velocidad(self):
    print(f"🔎 - El {self.modelo} va a {self.__velocidad} km/h\n")
    return self.__velocidad

def iniciar_velocidad(self):
    print(f"\n🚦 - El {self.modelo} inicia desde 0 km/h")
    return self.__velocidad
```

Y finalmente quedan estas últimas líneas de código que simplemente llaman a los métodos para que el programa pueda iniciar y funcionar.

El coche elegido a sido un **Tesla Roadster,** y la primera velocidad que alcanza el coche es de 50 km/h, después sube otros 250 km/h, eso hace que el coche llegue a su límite de 300 km/h y para terminar se intentan añadir otros 100 km/h adicionales, pero el programa protege al coche y lo vuelve a poner en sus 300 km/h como límite.

```
# Coches
coche = Coche("Tesla Roadster")

coche.iniciar_velocidad()

coche.acelerar(50)

coche.acelerar(250)

coche.acelerar(100)
```

```
miguel@DESKTOP-93SQ670:/mnt/c/Users/miguel/Desktop$ python3 file.py
🚦 - El Tesla Roaster inicia desde 0 km/h
🚀 - El Tesla Roaster acelera 50 km/h | Velocidad actual: 50 km/h
🚀 - El Tesla Roaster acelera 250 km/h | Velocidad actual: 300 km/h
⚠️ - El Tesla Roaster alcanzó el límite de 300 km/h y no puede más.
```