**Universidad Mariano Gálvez de Guatemala, Jutiapa**

**Docente:** [**Ruldin Efraín Ayala Ramos**](https://miumg.instructure.com/courses/139006/users/77404)

**Curso: Programación I**



**Tema: Validaciones de clase**

**Alumnos: Geofrey Florián Carné: 0905-24-17570**

**Sección: “A”**

**LLM Utilizado:**

He utilizado **ChatGPT**, un modelo de lenguaje basado en **GPT-4** para realizar las tareas que se me pidieron. Este modelo tiene la capacidad de entender y generar texto en base a preguntas y peticiones complejas.

**Prompt Utilizado:**

El prompt que utilicé para generar la respuesta fue:

"**Ayúdame con lo siguiente: modificar clases en Python para un ejercicio de programación, implementando validaciones, métodos de aceleración, frenos, y gestión de vehículos. También, implementa una clase derivada para la moto.**"

**Mejor Forma de Hacerlo:**

La mejor forma de hacerlo fue ir paso a paso, abordando cada parte del ejercicio de manera clara. Primero, analicé los requisitos para saber qué cambios hacer en las clases, luego implementé los métodos como acelerar, frenar, apagar, y me aseguré de que las clases derivadas (como la moto) se ajustaran a las especificaciones. Cada cambio se realizó teniendo en cuenta la lógica de programación y cómo estas funciones interactúan entre sí.

A lo largo del proceso, me aseguré de que el código fuera flexible y fácil de modificar para futuras mejoras, y utilicé las características del modelo de lenguaje para realizar el código de manera eficiente.

Para resolver el problema de validar el tipo de licencia cada vez que se cambie la propiedad en la clase Chofer, la mejor práctica en Python es utilizar el decorador @property junto con un método setter. Esto permite controlar el acceso a los atributos y asegurarnos de que, al modificar el tipo de licencia, se valide correctamente según la edad del conductor.

Primero, hice una investigación sobre cómo manejar este tipo de validaciones dinámicas en las clases de Python. Lo que descubrí es que al usar propiedades, podemos encapsular la lógica de validación dentro de los métodos getter y setter, lo cual nos proporciona un control más fino sobre cómo los atributos pueden ser modificados, sin permitir asignaciones directas que puedan ser erróneas.

Entonces, lo que propongo es usar un setter para el atributo tipo\_licencia en la clase Chofer. Al hacer esto, cada vez que el tipo de licencia sea modificado, se ejecutará automáticamente una validación para asegurarse de que el chofer cumpla con la edad mínima para ese tipo de licencia. De esta forma, no solo validamos el tipo de licencia en el constructor cuando se crea el objeto, sino que también lo validamos cada vez que intentamos cambiarlo.

**Ejemplo:**

class Chofer:

def init (self, nombre, edad, tipo\_licencia): self.nombre = nombre

self.edad = edad self.\_tipo\_licencia = None

self.tipo\_licencia = tipo\_licencia # Usamos el setter para validar

@property

def tipo\_licencia(self): return self.\_tipo\_licencia

@tipo\_licencia.setter

def tipo\_licencia(self, valor): self.validar\_licencia(valor) self.\_tipo\_licencia = valor

def validar\_licencia(self, tipo\_licencia):

if tipo\_licencia == 'A' and self.edad < 18:

raise ValueError("No cumple con la edad mínima para licencia tipo

A")

B")

C")

elif tipo\_licencia == 'B' and self.edad < 21:

raise ValueError("No cumple con la edad mínima para licencia tipo

elif tipo\_licencia == 'C' and self.edad < 25:

raise ValueError("No cumple con la edad mínima para licencia tipo

print(f"Licencia tipo {tipo\_licencia} asignada correctamente a

{self.nombre}.")

**Explicación de la solución:**

1. **Propiedad tipo\_licencia:** Utilizo el decorador @property para crear un getter y un setter para el atributo tipo\_licencia. El getter simplemente devuelve el valor de la propiedad privada

\_tipo\_licencia, mientras que el setter valida el tipo de licencia antes de asignarlo a la propiedad privada.

1. **Método validar\_licencia:** Este método se encarga de hacer la validación de acuerdo con la edad del chofer. Si el chofer no cumple con la edad mínima para el tipo de licencia, lanza un ValueError, lo que impide que se cambie el valor de la licencia.
2. **Ventajas de este enfoque:**
   * **Encapsulamiento:** La validación se centraliza dentro de un método, lo que mejora la organización del código.
   * **Control de acceso:** Asegura que el tipo de licencia solo se pueda modificar si se cumplen las condiciones.
   * **Flexibilidad:** Si en el futuro se desean agregar más validaciones

o cambiar la lógica, se puede hacer de manera sencilla en un solo lugar sin tener que modificar todo el código.

Este enfoque, utilizando @property con un setter, es una de las mejores prácticas en Python para manejar este tipo de situaciones porque garantiza que

cualquier cambio en un atributo pase por un proceso de validación o transformación, lo cual es más seguro y ordenado.

Al final, lo más importante es mantener el código limpio, fácil de leer y extender, y este patrón de propiedades en Python permite hacerlo de manera efectiva y con buenas prácticas.