Diego Gonzales Taylor

A. Responder las siguientes preguntas

1. ¿Cómo se define un lenguaje de programación?

Un lenguaje de programación es como un idioma que usamos para hablar con la computadora y decirle qué hacer. Nos ayuda a escribir instrucciones que la computadora entiende para hacer cosas como crear programas, juegos, o aplicaciones que usamos todos los días.

2. ¿Qué es la sintaxis en un lenguaje de programación?

La sintaxis es como la gramática de un idioma, pero para computadoras. Son las reglas que seguimos para escribir código correctamente. Si no sigues las reglas, la computadora no entenderá lo que quieres que haga y marcará errores.

3. Explica la diferencia entre sintaxis y semántica en los lenguajes de programación.

La sintaxis es cómo debes escribir el código, como las reglas de escritura. La semántica es lo que significa lo que escribes. Puedes escribir algo bien, pero si no tiene sentido, la computadora hará algo diferente a lo que esperabas.

4. ¿Qué cambios trajo la evolución de la programación estructurada a la programación orientada a objetos?

La programación orientada a objetos hizo más fácil organizar el código en partes más pequeñas llamadas objetos. Ahora, es más fácil entender y reutilizar lo que ya hemos hecho en lugar de escribir todo desde cero. Ayuda a hacer proyectos grandes más simples.

5. ¿Cuáles son los pilares fundamentales de la Programación Orientada a Objetos (POO)?

Los pilares de la POO son: Encapsulación, para proteger los datos; Abstracción, para hacer las cosas más fáciles de entender; Herencia, para usar lo que ya hicimos; y Polimorfismo, para hacer cosas de diferentes formas con el mismo código.

6. ¿Qué son los modificadores de acceso y para qué se utilizan en una clase?

Los modificadores de acceso son reglas que usamos en una clase para decidir quién puede ver y usar sus partes. Algunos datos son privados y nadie más los puede tocar. Otros son públicos, y cualquiera los puede usar. Protege el código de cambios no deseados.

7. ¿Qué es un constructor en una clase y cuál es su propósito?

Un constructor es una función especial que se usa cuando creamos un objeto de una clase. Sirve para darle a ese objeto sus primeros valores y asegurarse de que esté listo para usar. Es como darle un nombre y una forma a algo que acabas de hacer.

8. Definición de Clase y Objeto

Una clase es como un molde que nos dice cómo será algo, y un objeto es lo que creamos usando ese molde. Por ejemplo, si una clase es un plano para hacer carros, un objeto sería un carro real hecho con ese plano.

9. ¿Que es UML?

UML es un lenguaje visual que usamos para dibujar cómo funcionará un programa o sistema. Nos ayuda a ver y entender mejor las partes del programa antes de construirlo. Es como hacer un mapa antes de empezar un proyecto grande de software.

10. ¿Que nos permite la modularidad?

La modularidad nos ayuda a dividir un programa en partes más pequeñas llamadas módulos. Así, podemos trabajar en cada parte por separado, lo que hace más fácil cambiar y mejorar el código sin romper todo. Es como armar un rompecabezas en partes.

B. Realizar el siguiente programa en Python o Java

Enunciado: Sistema de Gestión de Vehículos

Descripción:

Modelar un sistema simple para gestionar diferentes tipos de vehículos en un concesionario.

Requisitos:

1. Clase base `Vehiculo`:
   - Propiedades comunes a todos los vehículos:
     - `Marca` (string)
     - `Modelo` (string)
     - `Año` (int)

2. Clase derivada `Auto`:
   - Hereda de `Vehiculo`.
   - Propiedades específicas:
     - `NumeroPuertas` (int)
     - `EsAutomatico` (bool)

3. Clase derivada `Moto`:
   - Hereda de `Vehiculo`.
   - Propiedades específicas:
     - `Cilindraje` (int)
     - `Tipo` (string) // Ejemplo: Scooter, Deportiva

Funcionalidades:

- Crear cuatro objetos de diferentes tipos (Autos y Motos).

- Mostrar la información específica de cada objeto.

Instrucciones:

- Crear las clases mencionadas.

- Implementar la función principal que cree una lista con varios autos y motos y muestre un reporte en consola.

Entrega:

El código fuente del proyecto deberá ser subido a un repositorio en GitHub. Incluir un archivo README.md con una breve descripción del proyecto y las instrucciones para ejecutarlo.

VEHICLEMANAGEMENTSYSTEM.PY

```
class Vehiculo:
    def __init__(self, marca, modelo, año):
        self.marca = marca
        self.modelo = modelo
        self.año = año

    def mostrar_informacion(self):
        return f"Marca: {self.marca}, Modelo: {self.modelo}, Año: {self.año}"


class Auto(Vehiculo):
    def __init__(self, marca, modelo, año, numero_puertas, es_automatico):
        super().__init__(marca, modelo, año)
        self.numero_puertas = numero_puertas
        self.es_automatico = es_automatico

    def mostrar_informacion(self):
        automatico_texto = "Automático" if self.es_automatico else "Manual"
        return f"{super().mostrar_informacion()}, Puertas: {self.numero_puertas},
Transmisión: {automatico_texto}"


class Moto(Vehiculo):
    def __init__(self, marca, modelo, año, cilindraje, tipo):
        super().__init__(marca, modelo, año)
        self.cilindraje = cilindraje
        self.tipo = tipo
```

```
    def mostrar_informacion(self):
        return f"{super().mostrar_informacion()}, Cilindraje: {self.cilindraje} cc,
Tipo: {self.tipo}"


def gestion_vehiculos():

    auto1 = Auto("BMW", "Serie 3", 2021, 4, True)
    auto2 = Auto("Bugatti", "Chiron", 2022, 2, True)
    auto3 = Auto("Rolls-Royce", "Phantom", 2020, 4, True)

    moto1 = Moto("Honda", "CBR600RR", 2021, 600, "Ninja")
    moto2 = Moto("BMW", "R 1250 GS", 2022, 1250, "Cruiser")

    vehiculos = [auto1, auto2, auto3, moto1, moto2]

    for vehiculo in vehiculos:
        print(vehiculo.mostrar_informacion())

gestion_vehiculos()

def menu_interactivo():
    vehiculos = [
        Auto("BMW", "Serie 3", 2021, 4, True),
        Auto("Bugatti", "Chiron", 2022, 2, True),
        Auto("Rolls-Royce", "Phantom", 2020, 4, True),
        Moto("Honda", "CBR600RR", 2021, 600, "Ninja"),
        Moto("BMW", "R 1250 GS", 2022, 1250, "Cruiser")
    ]

    while True:
        print("\n--- Sistema de Gestión de Vehículos ---")
        print("\n--- ============================== ---")
        print("1 -> Ingresar nuevo vehículo")
        print("2 -> Mostrar lista de vehículos")
        print("3 -> Salir")

        opcion = input("Seleccione una opción: ")

        if opcion == "1":

            tipo_vehiculo = input("Ingrese el tipo de vehículo (Auto/Moto):
").lower()

            marca = input("Marca: ")
            modelo = input("Modelo: ")
            año = int(input("Año: "))

            if tipo_vehiculo == "auto":
                numero_puertas = int(input("Número de puertas: "))
                es_automatico = input("¿Es automático? (si/no): ").lower() == "si"
```

```
                vehiculo = Auto(marca, modelo, año, numero_puertas, es_automatico)

            elif tipo_vehiculo == "moto":
                cilindraje = int(input("Cilindraje: "))
                tipo = input("Tipo (Ninja, Cruiser, etc.): ")
                vehiculo = Moto(marca, modelo, año, cilindraje, tipo)

            else:
                print("Tipo de vehículo no válido. Inténtelo de nuevo.")
                continue

            vehiculos.append(vehiculo)
            print(f"{tipo_vehiculo.capitalize()} ingresado correctamente.")

        elif opcion == "2":

            if not vehiculos:
                print("Lista vacía.")
            else:
                print("\n--- Lista de Vehículos ---")
                print("\n--- ================== ---")
                for vehiculo in vehiculos:
                    print(vehiculo.mostrar_informacion())

        elif opcion == "3":
            print("Saliendo del sistema...")
            break

        else:
            print("Opción no válida. Inténtelo de nuevo.")

menu_interactivo()
```

README.md

```
# VehicleManagementSystem

Este es un proyecto simple de Python que modela un sistema para gestionar diferentes tipos de vehículos en un concesionario. El sistema incluye una clase base `Vehiculo` y clases derivadas `Auto` y `Moto`. Además, se incluye un menú interactivo en consola para permitir al usuario ingresar nuevos vehículos o mostrar la lista existente de vehículos predefinidos y los que se vayan agregando.

## Instrucciones para ejecutar el proyecto

1. Clona este repositorio en tu máquina local:
   ```bash
   git clone https://github.com/Wolftch47/VehicleManagementSystem.git
```
```