Mg. Diana Marcela Ramirez Rodriguez

Asignatura: Programación Orientada a Objetos
Horas de Trabajo: Presenciales 3 horas
Trabajo Independiente: 6 horas

Contenido: Mecanismos de abstracción. Análisis de problemas y diseño de clases y relaciones entre ellas. Análisis de programas de otros programadores.

RA3.1: Mapea un problema real en un conjunto de objetos con sus relaciones, usando los 3 tipos de polimorfismo (sobrecarga, subtipado y tipo abstracto de dato) para evitar estructuras condicionales, y para desacoplar objetos.

Indicador de logro: *Usa una metodología orientada a objetos para descomponer un problema en los objetos que lo componen y sus relaciones*

Indicaciones:

El código realizado alójalo en Github en un repositorio denominado metodología orientada a objetos

- **En el campo de respuestas añade el enlace del código correspondiente al ejercicio planteado.**
- **Documenta el código**
- **Sube el documento al Curso en el campo clase 10: polimorfismo**

Mg. Diana Marcela Ramirez Rodriguez
Asignatura: Programación Orientada a Objetos
Horas de Trabajo: Presenciales 3 horas
Trabajo Independiente: 6 horas

Tipos de Polimorfismo

El **polimorfismo** se refiere a la capacidad de un objeto de adoptar diferentes formas. Existen tres formas comunes de implementar polimorfismo en Python:

Ejercicio 1: Polimorfismo basado en funciones (Ad-hoc)

Utilizando polimorfismo basado en funciones (Ad-hoc), implementa una función que pueda sumar diferentes tipos de objetos (números, listas, y cadenas de texto)

Ejercicio2: Polimorfismo basado en herencia (Inclusión)

En este ejercicio, implementa clases para diferentes tipos de vehículos, donde cada clase hija sobrescribe el comportamiento de la clase base Vehiculo.

Ejercicio3: Polimorfismo por sobrecarga de operadores

Utilizando una clase para **puntos en un espacio tridimensional (3D)** sobrecargando el operador de resta - para restar dos puntos

Mg. Diana Marcela Ramirez Rodriguez

Asignatura: Programación Orientada a Objetos
Horas de Trabajo: Presenciales 3 horas
Trabajo Independiente: 6 horas

DESARROLLO

Actividad #1

1. def operar(a, b): Definí la función operar que recibe dos parámetros, a y b.
2. suma = a + b Sumé a y b y guardé el resultado en la variable suma.
3. if type(a) == type(b) and type(a) in [int, float]: Verifiqué si ambos parámetros son del mismo tipo y si son números (int o float).
4. resta = a - b Si la condición se cumple, realicé la resta de a y b.
5. multiplicacion = a * b Multipliqué a y b
6. division = a / b Dividí a entre b.
7. return suma, resta, multiplicacion, division Devolví los resultados de la suma, resta, multiplicación y división.
8. else: Si los tipos no coinciden o no son números:
9. return suma, None, None, None Devolví la suma y None para las otras operaciones. El None se usa para evitar errores de operaciones incompatibles.
10. print(operar(10, 2)) Probé la función con dos enteros.
11. print(operar(10.0, 2)) Probé la función con un flotante y un entero.
12. print(operar("Hola", " Univalle")) Probé la función con dos cadenas de texto.
13. print(operar([1, 2], [3, 4])) Probé la función con dos listas.

```
14. print("_____")
15. print("Acitividad #1")
16. def operar(a, b):
17.     suma = a + b
18.     if type(a) == type(b) and type(a) in [int, float]:
19.         resta = a - b
20.         multiplicacion = a * b
21.         division = a / b
22.         return suma, resta, multiplicacion, division
23.     else:
24.         return suma, None, None, None
25.
26. print(operar(10, 2))
27. print(operar(10.0, 2))
28. print(operar("Hola", " Univalle"))
29. print(operar([1, 2], [3, 4]))
```

Mg. Diana Marcela Ramirez Rodriguez
Asignatura: Programación Orientada a Objetos
Horas de Trabajo: Presenciales 3 horas
Trabajo Independiente: 6 horas

Actividad #2

1. class Vehiculo: Definí la clase base Vehiculo.
2. def mover(self): Definí un método vacío mover el Vehiculo.
3. class Carro(Vehiculo): Creé una subclase Carro que hereda de Vehiculo.
4. def mover(self): Redefiní el método mover en Carro.
5. return 'El carro está conduciendo' El método mover en Carro devuelve esta cadena.
6. class Moto(Vehiculo): Creé una subclase Moto que hereda de Vehiculo.
7. def mover(self): Redefiní el método mover en Moto.
8. return 'La moto está conduciendo' El método mover en Moto devuelve esta cadena.
9. class Monopatin(Vehiculo): Creé una subclase Monopatin que hereda de Vehiculo.
10. def mover(self): Redefiní el método mover en Monopatin.
11. return 'El monopatín está rodando' El método mover en Monopatin devuelve esta cadena.
12. vehiculos = [Carro(), Moto(), Monopatin()] Creé una lista de instancias Carro, Moto y Monopatin
13. for vehiculo in vehiculos: Iteré sobre la lista de vehículos.
14. print(vehiculo.mover()) Llamé al método mover de cada objeto e imprimí el resultado.

```
15. print("_____")
16. print("Acitividad #2")
17. class Vehiculo:
18.     def mover(self):
19.         pass
20.
21. class Carro(Vehiculo):
22.     def mover(self):
23.         return 'El carro está conduciendo'
24.
25. class Moto(Vehiculo):
26.     def mover(self):
27.         return 'La moto está conduciendo'
28.
29. class Monopatin(Vehiculo):
30.     def mover(self):
31.         return 'El monopatín está rodando'
32.
33. vehiculos = [Carro(), Moto(), Monopatin()]
34. for vehiculo in vehiculos:
35.     print(vehiculo.mover())
```

Mg. Diana Marcela Ramirez Rodriguez
Asignatura: Programación Orientada a Objetos
Horas de Trabajo: Presenciales 3 horas
Trabajo Independiente: 6 horas

Actividad #3

1. class Punto: Definí la clase Punto.
2. def init(self, x, y): Definí el constructor que inicializa los atributos x e y.
3. self.x = x Asigné el valor de x al atributo x.
4. self.y = y Asigné el valor de y al atributo y.
5. def sumar(self, otro): Definí el método sumar que toma otro objeto Punto.
6. return Punto(self.x + otro.x, self.y + otro.y) El método sumar devuelve un nuevo Punto con las coordenadas sumadas.
7. def repr(self): Definí el método __repr__ para la representación en cadena del objeto.
8. return f'Punto({self.x}, {self.y})' El método __repr__ devuelve una cadena con el formato (x, y).
9. p1 = Punto(1, 2) Creé una instancia de Punto p1 con coordenadas (1, 2).
10. p2 = Punto(3, 4) Creé una instancia de Punto p2 con coordenadas (3, 4).
11. p3 = p1.sumar(p2) Sumé p1 y p2 usando el método sumar, guardando el resultado en p3.
12. print(p3) Imprimí p3, mostrando las coordenadas sumadas (4, 6).

```
13. print("_____")
14. print("Acitividad #3")
15. class Punto:
16.     def __init__(self, x, y):
17.         self.x = x
18.         self.y = y
19.
20.     def sumar(self, otro):
21.         return Punto(self.x + otro.x, self.y + otro.y)
22.
23.     def __repr__(self):
24.         return f'Punto({self.x}, {self.y})'
25.
26. p1 = Punto(1, 2)
27. p2 = Punto(3, 4)
28. p3 = p1.sumar(p2)
29. print(p3)
30.
```