

Trabajo 02 – Repaso de POO en Python

Alumno: Puma Huanca Anthony Rusbel
Docente: Ing. Coyla Idme Leonel
Lenguajes de Programación II – FINESI
Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Ejercicio 1 Clases y objetos (básico–intermedio)

Crea una clase CuentaBancaria con:

- atributos: titular, saldo
- métodos: depositar(), retirar(), mostrar_saldo()
- el método retirar() debe evitar que el saldo quede negativo.

```
1 class CuentaBancaria:
2     def __init__(self, titular, saldo=0):
3         self.titular = titular
4         self.saldo = saldo
5
6     def depositar(self, cantidad):
7         if cantidad > 0:
8             self.saldo += cantidad
9             print(f"Dep sito de {cantidad} realizado.")
10
11     def retirar(self, cantidad):
12         if cantidad > self.saldo:
13             print("Fondos insuficientes.")
14         else:
15             self.saldo -= cantidad
16             print(f"Retiro de {cantidad} realizado.")
17
18     def mostrar_saldo(self):
19         print(f"Titular: {self.titular}, Saldo: {self.saldo}")
20
21 cuenta1 = CuentaBancaria("Anthony", 1000)
22 cuenta2 = CuentaBancaria("Nayelin", 500)
23
24 cuenta1.depositar(200)
25 cuenta1.retirar(1500)
26 cuenta1.mostrar_saldo()
27
```

```
28 cuenta2.retirar(100)
29 cuenta2.mostrar_saldo()
```

Listing 1: Clase CuentaBancaria

Ejecución:

Depósito de 200 realizado.
Fondos insuficientes.
Titular: Anthony, Saldo: 1200
Retiro de 100 realizado.
Titular: Nayelin, Saldo: 400

Ejercicio 2 Encapsulamiento y propiedades

Crea una clase Producto con:

- atributos privados: `_nombre`, `_precio`
- propiedad `precio` que evite valores negativos
- método `aplicar_descuento(porcentaje)`

```
1 class Producto:
2     def __init__(self, nombre, precio):
3         self._nombre = nombre
4         self._precio = precio
5
6     @property
7     def precio(self):
8         return self._precio
9
10    @precio.setter
11    def precio(self, valor):
12        if valor < 0:
13            print("El precio no puede ser negativo.")
14        else:
15            self._precio = valor
16
17    def aplicar_descuento(self, porcentaje):
18        if 0 < porcentaje <= 100:
19            descuento = self._precio * (porcentaje / 100)
20            self._precio -= descuento
21        else:
22            print("Porcentaje inv lido.")
23
24    p = Producto("Laptop", 1000)
25    p.precio = -50
26    print(f"Precio actual: {p.precio}")
27
28    p.aplicar_descuento(10)
29    print(f"Precio con descuento: {p.precio}")
30
```

```
31 p.aplicar_descuento(200)
```

Listing 2: Clase Producto con propiedad

Ejecución:

El precio no puede ser negativo.

Precio actual: 1000

Precio con descuento: 900.0

Porcentaje inválido.

Ejercicio 3 Herencia simple

Define Empleado con atributos nombre y salario. Crea subclases:

- EmpleadoTiempoCompleto
- EmpleadoPorHoras

Cada una debe reimplementar calcular_pago().

```
1 class Empleado:
2     def __init__(self, nombre, salario_base):
3         self.nombre = nombre
4         self.salario_base = salario_base
5
6     def calcular_pago(self):
7         return self.salario_base
8
9 class EmpleadoTiempoCompleto(Empleado):
10     def calcular_pago(self):
11         return self.salario_base * 1.10
12
13 class EmpleadoPorHoras(Empleado):
14     def __init__(self, nombre, tarifa_hora, horas_trabajadas):
15         self.nombre = nombre
16         self.salario_base = 0
17         self.tarifa_hora = tarifa_hora
18         self.horas_trabajadas = horas_trabajadas
19
20     def calcular_pago(self):
21         return self.tarifa_hora * self.horas_trabajadas
22
23 empleados = [
24     EmpleadoTiempoCompleto("Luis", 2000),
25     EmpleadoPorHoras("Ana", 20, 80)
26 ]
27
28 for emp in empleados:
29     print(f"{emp.nombre}: Pago total = {emp.calcular_pago()}")
```

Listing 3: Herencia con pago diferenciado

Ejecución:

Luis: Pago total = 2200.0

Ana: Pago total = 1600

Ejercicio 4 Herencia múltiple

Crea:

- Vehículo con método `acelerar()`
- Volador con método `volar()`
- Avion que herede de ambas

```
1 class Vehiculo:
2     def acelerar(self):
3         print("El veh culo est  acelerando.")
4
5 class Volador:
6     def volar(self):
7         print("El objeto est  volando.")
8
9 class Avion(Vehiculo, Volador):
10     pass
11
12 mi_avion = Avion()
13 mi_avion.acelerar()
14 mi_avion.volar()
```

Listing 4: Herencia múltiple: Avion

Ejecución:

El vehículo está acelerando.

El objeto está volando.

Ejercicio 5 Polimorfismo

Crea una clase base `Figura` con método `area()`. Implementa:

- Rectangulo
- Triangulo
- Circulo

Imprime el área de cada figura sin usar `isinstance`.

```
1 import math
2
3 class Figura:
4     def area(self):
5         pass
6
7 class Rectangulo(Figura):
8     def __init__(self, base, altura):
9         self.base = base
10        self.altura = altura
```

```

11     def area(self):
12         return self.base * self.altura
13
14 class Triangulo(Figura):
15     def __init__(self, base, altura):
16         self.base = base
17         self.altura = altura
18     def area(self):
19         return (self.base * self.altura) / 2
20
21 class Circulo(Figura):
22     def __init__(self, radio):
23         self.radio = radio
24     def area(self):
25         return math.pi * (self.radio ** 2)
26
27 figuras = [Rectangulo(10, 5), Triangulo(10, 5), Circulo(3)]
28
29 for f in figuras:
30     print(f" rea : {f.area():.2f}")

```

Listing 5: Polimorfismo con figuras

Ejecución:

Área: 50.00

Área: 25.00

Área: 28.27

Ejercicio 6 Sobrecarga de operadores

Crea Vector2D con:

- `__add__`, `__sub__`, `__mul__`

```

1 class Vector2D:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     def __add__(self, other):
7         return Vector2D(self.x + other.x, self.y + other.y)
8
9     def __sub__(self, other):
10        return Vector2D(self.x - other.x, self.y - other.y)
11
12    def __mul__(self, escalar):
13        return Vector2D(self.x * escalar, self.y * escalar)
14
15    def __str__(self):
16        return f"Vector({self.x}, {self.y})"
17

```

```

18 v1 = Vector2D(2, 3)
19 v2 = Vector2D(1, 4)
20
21 print(f"Suma: {v1 + v2}")
22 print(f"Resta: {v1 - v2}")
23 print(f"Multiplicaci n : {v1 * 3}")

```

Listing 6: Vector2D con operadores sobrecargados

Ejecución:

Suma: Vector(3, 7)
 Resta: Vector(1, -1)
 Multiplicación : Vector(6, 9)

Ejercicio 7 Composición

Crea:

- Motor con método encender()
- Auto que contiene un motor y tiene arrancar()

```

1 class Motor:
2     def encender(self):
3         print(" El motor ha arrancado correctamente.")
4
5 class Auto:
6     def __init__(self):
7         self.motor = Motor()
8
9     def arrancar(self):
10        print("Girando la llave del auto...")
11        self.motor.encender()
12        print(" Auto  listo para avanzar!")
13
14 mi_auto = Auto()
15 mi_auto.arrancar()

```

Listing 7: Composición: Auto contiene Motor

Ejecución:

Girando la llave del auto...
 El motor ha arrancado correctamente.
 ¡Auto listo para avanzar!

Ejercicio 8 Métodos de clase y estáticos

Crea ConversorTemperatura con:

- método de clase desde_celsius(cls, c)
- método estático celsius_a_fahrenheit(c)

```

1 class ConversorTemperatura:
2     def __init__(self, fahrenheit):
3         self.fahrenheit = fahrenheit
4
5     @staticmethod
6     def celsius_a_fahrenheit(c):
7         return (c * 9/5) + 32
8
9     @classmethod
10    def desde_celsius(cls, c):
11        f = cls.celsius_a_fahrenheit(c)
12        return cls(f)
13
14 temp_obj = ConversorTemperatura.desde_celsius(25)
15 print(f"Objeto creado con temperatura en F: {temp_obj.fahrenheit}")
16
17 f_calc = ConversorTemperatura.celsius_a_fahrenheit(0)
18 print(f"0 C en Fahrenheit es: {f_calc}")

```

Listing 8: Métodos estáticos y de clase

Ejecución:

Objeto creado con temperatura en F: 77.0
0°C en Fahrenheit es: 32.0

Ejercicio 9 Excepciones en POO

Crea CalculadoraSegura con método dividir(a, b) que lance una excepción personalizada si $b == 0$.

```

1 class DivisionPorCeroError(Exception):
2     pass
3
4 class CalculadoraSegura:
5     def dividir(self, a, b):
6         if b == 0:
7             raise DivisionPorCeroError("No se puede dividir por
8             cero.")
9             return a / b
10
11 calc = CalculadoraSegura()
12 try:
13     print(calc.dividir(10, 2))
14     print(calc.dividir(5, 0))
15 except DivisionPorCeroError as e:
16     print(f"Excepci n capturada: {e}")

```

Listing 9: Excepción personalizada

Ejecución:

5.0
Excepción capturada: No se puede dividir por cero.

Ejercicio 10 Clases abstractas

Usa `abc` para crear una clase abstracta `Animal` con método abstracto `hacer_sonido()`. Implementa `Perro` y `Gato`.

```
1 from abc import ABC, abstractmethod
2
3 class Animal(ABC):
4     @abstractmethod
5     def hacer_sonido(self):
6         pass
7
8 class Perro(Animal):
9     def hacer_sonido(self):
10         return "Guau"
11
12 class Gato(Animal):
13     def hacer_sonido(self):
14         return "Miau"
15
16 animales = [Perro(), Gato()]
17
18 for animal in animales:
19     print(animal.hacer_sonido())
```

Listing 10: Clase abstracta con `abc`

Ejecución:

Guau
Miau