Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Docente:

Ing. Coyla Idme Leonel

Alumno:

Ticona Miramira Roberto Angel

Trabajo Encargado 02

» 1. Clases y objetos (básico-intermedio)

Crear una clase CuentaBancaria que tenga:

- nombre del titular
- saldo
- métodos: depositar(), retirar(), mostrar_saldo()

Código

```python
class CuentaBancaria:
    def __init__(self, titular, saldo = 0):
        self.titular = titular
        self.saldo = saldo

    def depositar(self, monto):
        self.saldo += monto

    def retirar(self, monto):
        if monto > self.saldo:
            print("Fondos insuficientes")
        else:
            self.saldo -= monto

    def mostrarSaldo(self):
        print(f"El saldo de {self.titular} es {self.saldo}")

def prueba_cuenta_bancaria():
    c1 = CuentaBancaria("Ana", 500)
    c2 = CuentaBancaria("Luis", 300)

    c1.depositar(200)
    c1.retirar(100)
    c1.mostrarSaldo()
    c2.retirar(500)
    c2.mostrarSaldo()

def main():
    print("==== Probando CuentaBancaria ====")
    prueba_cuenta_bancaria()

if __name__=="__main__":
    main()
```

Ejecución

```
==== Probando CuentaBancaria ====
El saldo de Ana es 600
Fondos insuficientes
El saldo de Luis es 300
```

» 2. Encapsulamiento y propiedades

Crea una clase Producto con:

- atributos privados: _nombre, _precio
- una propiedad precio que valide que el precio nunca sea negativo
- método aplicar_descuento(porcentaje)

Código

```python
class Producto:
    def __init__(self, nombre, precio):
        self.__nombre = nombre
        self.__precio = precio

    def get_nombre(self):
        return self.__nombre

    def get_precio(self):
        return self.__precio

    def set_nombre(self, nuevoN):
        self.__nombre = nuevoN
        return self.__nombre

    def set_precio(self, nuevoP):
        if nuevoP > 0:
            self.__precio = nuevoP
        else:
            print("No puede ser negativo")

    def descuento(self, descuento):
        if 0 < descuento < 100:
            self.__precio *= (100 - descuento) / 100
        else:
            print("Descuento inválido")

def probarProductos():
    producto1 = Producto("Zapatilla", 250)
    producto2 = Producto("Casaca", 300)
    print(producto1.get_nombre())
    print(producto1.get_precio())
    producto1.descuento(10)
    producto1.descuento(120)
    print(producto1.get_nombre())
    print(producto1.get_precio())
    print(producto2.get_nombre())
    print(producto2.get_precio())
    producto2.descuento(50)
    print(producto2.get_nombre())
    print(producto2.get_precio())

def main():
    probarProductos()

if __name__=="__main__":
```

```
    main()
```

Ejecución

```
Zapatilla
250
Descuento inválido
Zapatilla
225.0
Casaca
300
Casaca
150.0
```

» 3. Herencia simple

Define una clase Empleado con atributos:

- nombre, salario

Luego crea dos subclases:

- EmpleadoTiempoCompleto
- EmpleadoPorHoras

Cada una debe redefinir un método calcular_pago() de manera distinta.

Escribe un programa que reciba una lista de empleados mixtos e imprima el pago de cada uno.

Código

```
class Empleado:
    def __init__(self, nombre, salario):
        self.nombre = nombre
        self.salario = salario


class EmpleadoTiempoCompleto(Empleado):
    def __init__(self, nombre, dias):
        Empleado.__init__(self, nombre, 0)
        self.dias = dias

    def calcular_pago(self):
        self.salario = 60 * self.dias
        return self.salario


class EmpleadoPorHoras(Empleado):
    def __init__(self, nombre, dias, horas):
        Empleado.__init__(self, nombre, 0)
        self.dias = dias
        self.horas = horas

    def calcular_pago(self):
        self.salario = 5 * self.dias * self.horas
        return self.salario

def main():
    empleados = [
        EmpleadoTiempoCompleto("Juan", 20),
        EmpleadoPorHoras("Ana", 10, 6),
        EmpleadoTiempoCompleto("Luis", 15),
        EmpleadoPorHoras("María", 12, 4)
```

```
    ]

    for emp in empleados:
        print(emp.nombre, "-> Pago:", emp.calcular_pago())

if __name__=="__main__":
    main()
```

Ejecución

```
Juan -> Pago: 1200
Ana -> Pago: 300
Luis -> Pago: 900
María -> Pago: 240
```

» 4. Herencia múltiple

Crea estas clases:

- Vehiculo(método acelerar())
- Volador(método volar())

Luego crea Avion, que hereda de ambas.

Escribe un programa que cree un avión y llame a ambos métodos

Código

```
class Vehiculo:
    def acelerar(self):
        print("El vehículo esta acelerando")

class Volador:
    def volar(self):
        print("Estamos volando")

class Avion(Vehiculo, Volador):
    pass

def main():
    avion1 = Avion()
    avion1.acelerar()
    avion1.volar()

if __name__ == "__main__":
    main()
```

Ejecución

```
El vehículo esta acelerando
Estamos volando
```

» 5. Polimorfismo

Crea una interfaz de dibujo usando una clase base Figura con un método area().

Implementa tres clases:

- Rectangulo
- Triangulo

- Circulo

Crea una lista de figuras y escribe un programa que imprima el área de cada una sin preguntar el tipo de objeto.

Código

```python
import math
class Figura:
    def area(self):
        pass

class Rectangulo(Figura):
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def area(self):
        return self.base * self.altura

class Triangulo(Figura):
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def area(self):
        return ((self.base * self.altura) / 2)

class Circulo(Figura):
    def __init__(self, radio):
        self.radio = radio

    def area(self):
        return math.pi * (self.radio ** 2)

def main():
    figuras = [Rectangulo(4,5), Circulo(3), Triangulo(5,2)]

    for figura in figuras:
        print(f"Área : {figura.area():.2f}")

if __name__=="__main__":
    main()
```

Ejecución

```
Área : 20.00
Área : 28.27
Área : 5.00
```

» 6. Sobrecarga de operadores

Crea una clase Vector2D que implemente:

- ___add___(suma de vectores)
- ___sub___(resta)
- ___mul___(multiplicación por escalar)

Escribe un programa que use estos operadores.

Código

```python
class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, otro):
        return Vector2D(self.x + otro.x, self.y + otro.y)

    def __sub__(self, otro):
        return Vector2D(self.x - otro.x, self.y - otro.y)

    def __mul__(self, escalar):
        return Vector2D(self.x * escalar, self.y * escalar)

    def __str__(self):
        return f"({self.x}, {self.y})"

def main():
    v1 = Vector2D(3, 4)
    v2 = Vector2D(1, 2)

    print("v1 =", v1)
    print("v2 =", v2)

    print("Suma:", v1 + v2)
    print("Resta:", v1 - v2)
    print("Multiplicación:", v1 * 3)

if __name__ == "__main__":
    main()
```

Ejecución

```
v1 = (3, 4)
v2 = (1, 2)
Suma: (4, 6)
Resta: (2, 2)
Multiplicación: (9, 12)
```

» 7. Composición

Crea una clase Motor y una clase auto que contenga un motor.

El motor debe tener un método encender().

El auto debe tener un método arrancar() que use su motor.

Escribe un programa que construya un auto y lo arranque.

Código

```python
class Motor:
    def __init__(self, tipo):
        self.tipo = tipo

    def encender(self):
        print(f"Motor: {self.tipo} encendido")

class Auto:
    def __init__(self, marca):
        self.marca = marca
        self.motor = Motor("Gasolinero")

```

```
    def arrancar(self):
        print(f"Auto: {self.marca} arrancando")
        self.motor.encender()

def main():
    miAuto = Auto("Toyota")
    miAuto.arrancar()

if __name__=="__main__":
    main()
```

Ejecución

```
Auto: Toyota arrancando
Motor: Gasolinero encendido
```

» 8. Métodos de clase y métodos estáticos

Crea una clase ConversorTemperatura con:

- Un método de clase desde_celsius(cls, c) que cree un objeto a partir de grados Celsius
- Un método estático celcius_a_fahrenheit(c)
- Un atributo que almacene la temperatura en Fahrenheit

Prueba la clase.

Código

```
class ConversorTemperatura:
    def __init__(self, fahrenheit):
        self.fahrenheit = fahrenheit

    def celsius_a_fahrenheit(self, c):
        return (c * 9/5) + 32

    def desde_celsius(self, c):
        f = self.celsius_a_fahrenheit(c)
        return ConversorTemperatura(f)

def main():
    conversor = ConversorTemperatura(0)
    t1 = conversor.desde_celsius(25)
    print("Temperatura en Fahrenheit:", t1.fahrenheit)

if __name__ == "__main__":
    main()
```

Ejecución

```
Temperatura en Fahrenheit: 77.0
```

» 9. Uso de excepciones en POO

Crea una clase CalculadoraSegura con un método:

- dividir(a, b) que lance una excepción personalizada DivisionPorCeroError si b == 0

Pruebe tu clase capturando la excepción.

Código

```python
class CalculadoraSegura:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def dividir(self):
        try:
            resultado = self.a / self.b
            return resultado
        except ZeroDivisionError:
            return "Error: No se puede dividir entre cero."
        except Excepcion as e:
            return f"Ocurrió un error: {e}"
        finally:
            print("Operación finalizada.")

operacion = CalculadoraSegura(10, 0)
print(operacion.dividir())
```

Ejecución

```
Operación finalizada.
Error: No se puede dividir entre cero.
```

» 10. Clases abstractas

Usa el módulo abc para crear una clase abstracta Animal con el método abstracto hacer_sonido().

Implementa dos animales: Perro y Gato.

Crea una lista de animales y haz que cada uno haga su sonido.

Código

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def hacer_sonido(self):
        pass

class Perro(Animal):
    def hacer_sonido(self):
        print("Guau guau!")


class Gato(Animal):
    def hacer_sonido(self):
        print("Miau!")


def main():
    animales = [Perro(), Gato(), Perro()]

    for animal in animales:
        animal.hacer_sonido()


if __name__ == "__main__":
    main()
```

Ejecución

```
Guau guau!
Miau!
Guau guau!
```