Poliformismo

1. Ejercicio: Figuras geométricas

Crea una clase base Figura con métodos abstractos area() y perimetro().

Luego crea clases Circulo, Rectangulo, y Triangulo que hereden de Figura y sobrescriben estos métodos.

Tarea:

Crear una función que reciba una lista de figuras y muestre su área y perímetro sin importar qué tipo de figura sea (polimorfismo).

Crear varias figuras diferentes y pasarlas a esa función.

2. Ejercicio: Vehículos y sonoros

Crea una clase base Vehiculo con un método sonido() que devuelve un string.

Luego crea clases Coche, Moto, y Camion, sobrescribiendo sonido() para que devuelvan sonidos típicos: "Vroom", "Brrr", "Rrrr".

Tarea:

Escribir una función que acepte una lista de vehículos y muestre el sonido de cada uno, sin importar el tipo. Mostrar los sonidos usando polimorfismo.

3. Ejercicio: Empleados y beneficios

Crea una clase base Empleado con método calcular_beneficio().

Se derivan clases EmpleadoFijo, EmpleadoTemporal y EmpleadoFreelance, cada una con su propia implementación del método.

Tarea:

Crear una función que calcule y muestre los beneficios de una lista de empleados, sin preocuparse por su clase específica.

Crear una lista con diferentes tipos de empleados y pasarla a esa función.

4. Ejercicio: Animales en un zoológico

Crea una clase Animal con método hablar().

Deriva clases Perro, Gato, y Pato, sobrescribiendo hablar() para devolver sonidos como "Guau", "Miau", "Cuac".

Tarea:

Crear una función que recibe una lista de Animal y llama al método hablar() en cada uno, demostrando polimorfismo.

Agrega más animales y prueba la función.

5. Ejercicio: Trabajadores y tarifa horaria

Supón una clase base Trabajador con un método calcular_pago().

Derivan clases TrabajadorPorHora y TrabajadorPorProyecto con diferentes implementaciones.

Tarea:

Crear una lista de diferentes trabajadores y, usando un solo método, mostrar cuánto deben pagarles sin importar el tipo de trabajador.

Recorre la lista y muestra el pago de cada uno.

Excepciones

1. División segura

Crea una función que tome dos números y retorne su división.
Maneja excepciones para evitar divisiones por cero y reportar un mensaje adecuado si ocurre un error.

---

2. Leer entero del usuario

Escribe un programa que pida al usuario ingresar un número entero.
Utiliza try-except para capturar errores si el usuario ingresa algo que no sea un entero y solicita el ingreso nuevamente hasta que sea correcto.

---

3. Acceder a elementos de una lista

Dada una lista de números, pide al usuario ingresar un índice y muestra el valor almacenado en esa posición.
Maneja excepciones como IndexError si el índice es inválido y reporta un mensaje amigable.

4. Manejo de excepciones con división

Escribe una función que reciba dos números y retorne su división.
La función debe verificar si el divisor es caracter, y si es así, lanzar intencionadamente una excepción ValueError con un mensaje personalizado usando raise.
Luego, en un bloque try-except, captura esa excepción y muestra un mensaje informando del error.

Instrucciones adicionales:

- La función debe lanzar la excepción solamente cuando detecta división por un caracter.
- La captura del error debe mostrar un mensaje amigable.