Práctico 6: Estructuras de datos complejas

Objetivo:

Dominar el uso de estructuras de datos complejas en Python para almacenar, organizar y manipular datos de manera eficiente, aplicando buenas prácticas y optimizando el rendimiento de las aplicaciones.

Resultados de aprendizaje:

1. Comprensión y aplicación de iterables: listas, tuplas y sets.
2. Introducción a estructuras de datos complejas: diccionarios y objetos.

Actividades

1) Dado el diccionario precios_frutas

```
precios_frutas = {'Banana': 1200, 'Ananá': 2500, 'Melón': 3000, 'Uva': 1450}
```

Añadir las siguientes frutas con sus respectivos precios:

- Naranja = 1200
- Manzana = 1500
- Pera = 2300

2) Siguiendo con el diccionario precios_frutas que resulta luego de ejecutar el código desarrollado en el punto anterior, actualizar los precios de las siguientes frutas:

- Banana = 1330
- Manzana = 1700
- Melón = 2800

3) Siguiendo con el diccionario precios_frutas que resulta luego de ejecutar el código desarrollado en el punto anterior, crear una lista que contenga únicamente las frutas sin los precios.

4) Crear una clase llamada Persona que contenga un método __init__ con los atributos nombre, pais y edad y el método saludar. El método saludar debe imprimir por pantalla un mensaje de salud que siga la estructura "¡Hola! Soy [nombre], vivo en [pais] y tengo [edad] años."

5) Crear una clase llamada Circulo que contenga el atributo radio y los métodos calcular_area y calcular_perimetro. Dichos métodos deben calcular el parámetro correspondiente.

Ayuda: el módulo math puede ser de utilidad para usar la constante π.

6) Dado un string con paréntesis `"()"`, `"{}"`, `"[]"`, verifica si están correctamente balanceados usando una **pila**.

Ejemplo de entrada y salida:

```
balanceado("({[]})") → True
balanceado("({[})") → False
```

7) Usa una **cola** para simular un sistema de turnos en un banco. La cola debe permitir:

- Agregar clientes (`encolar`).
- Atender clientes (`desencolar`).
- Mostrar el siguiente cliente en la fila.

8) Crea una **lista enlazada** que permita insertar nodos al inicio y recorrer la lista para mostrar los valores almacenados.

9) Dada una lista enlazada, implementa una función para invertirla.

Ejemplo de entrada y salida:

```
Lista original: 1 -> 2 -> 3 -> None
Lista invertida: 3 -> 2 -> 1 -> None
```