**Informe del proyecto de Lenguajes de Programación**

**Mauricio J. Avalo Tamayo 3ro Ciencias de la Computación**

El presente informe describe un programa desarrollado en Python el cual permite gestionar tus tareas diarias, organizarlas mediante carpetas, categorías, fechas de caducidad, realización y pendientes, así como la modificación y eliminación de las mismas.

1. Programación Estructurada

#### La programación estructurada se observa en el uso de funciones y estructuras de control para organizar el flujo del programa.

#### Ejemplos:

• Funciones para manejo de tareas:

• crear\_tarea: Crea un diccionario con los datos de una tarea.

• guardar\_tareas: Guarda las tareas en un archivo JSON.

• cargar\_tareas: Carga las tareas desde un archivo JSON.

• es\_tarea\_vencida: Verifica si una tarea está vencida.

- Control de flujo:

• Uso de estructuras como try-except para manejar errores (e.g., archivo no encontrado en cargar\_tareas).

• Ciclos como while en verificar\_recordatorios para ejecutar tareas periódicas.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Programación Orientada a Objetos (POO)

El paradigma POO es el núcleo de la aplicación, ya que se utiliza una clase principal para encapsular la lógica y los datos de la aplicación:

• Clase ToDoApp:

• Representa la aplicación completa.

• Hereda de ctk.CTk para personalizar la ventana principal.

• Contiene atributos como self.tareas (lista de tareas) y self.tareas\_filtradas (tareas visibles).

• Métodos como mostrar\_tareas, crear\_tarea, eliminar\_tarea, y guardar\_tareas encapsulan funcionalidades específicas.

- Encapsulación:

• Los métodos y atributos están organizados dentro de la clase, lo que facilita la modularidad y el mantenimiento.

- Herencia:

• La clase ToDoApp hereda de ctk.CTk, reutilizando funcionalidades de la biblioteca customtkinter.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Programación Funcional

##### El paradigma funcional se utiliza en ciertas partes del código, especialmente en el manejo de listas y filtros:

- Uso de funciones de orden superior:

• filter en filtrar\_tareas: Se utiliza para filtrar tareas según un criterio proporcionado como función lambda.

- Ejemplo:

• lambda t: t["completada"] para filtrar tareas completadas.

- Funciones lambda:

• Se emplean para definir funciones anónimas en tiempo de ejecución, como en los botones de filtro (btn\_filtro\_completadas, btn\_filtro\_pendientes).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Programación Concurrente

#### *La concurrencia se implementa para manejar tareas en segundo plano, como los recordatorios:*

- Uso de hilos (threading):

• El método iniciar\_recordatorios ejecuta verificar\_recordatorios en un hilo separado para no bloquear la interfaz gráfica.

- Ejemplo:

• threading.Thread(target=self.iniciar\_recordatorios, daemon=True).start().

- Uso de asyncio:

• El método verificar\_recordatorios utiliza programación asíncrona para verificar recordatorios periódicamente sin bloquear el hilo principal.

- Ejemplo:

• await asyncio.sleep(60) para pausar la ejecución durante un minuto.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Programación Basada en Eventos

##### La aplicación utiliza eventos para interactuar con el usuario:

- Botones y comandos:

• Cada botón tiene un evento asociado que ejecuta un método específico. Ejemplo:

• self.btn\_crear: Llama al método crear\_tarea para abrir una ventana de creación de tareas.

• self.btn\_guardar: Llama a guardar\_tareas para guardar las tareas en el archivo.

- Interacción con la interfaz gráfica:

• Los eventos de clic en botones o cierre de ventanas desencadenan acciones específicas.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_