EJERCICIO FINAL

Students Management System

Presenta:
Rony Osman Carrillo Bermudez
0905 - 21 – 15472

Angel Gabriel Chavarría Yanes
0905-23-3809

Catedrático:
Ruldin Efrain Ayala Ramos

Sábado 24 de mayo del 2025

Mapa

Guía de uso

Introducción

El Sistema de Gestión de Estudiantes es una aplicación de escritorio desarrollada en Python con una interfaz gráfica de usuario (GUI) construida con Tkinter. Permite administrar la información de los estudiantes, incluyendo sus nombres, carnés únicos, materias inscritas y promedios. El sistema inicializa con 30 estudiantes de ejemplo para facilitar la prueba de sus funcionalidades.

2. Prerrequisitos

- Python 3.x : Asegúrate de tener Python 3 instalado en tu sistema.

- Tkinter : Tkinter es la biblioteca estándar de GUI para Python y generalmente viene incluida con las instalaciones de Python. Si por alguna razón no está disponible, puede que necesites instalarla por separado (por ejemplo, sudo apt-get install python3-tk en sistemas Debian/Ubuntu o verificar tu instalación de Python en Windows).

3. Ejecución de la Aplicación

Para ejecutar la aplicación:

1. Abre una terminal o línea de comandos.

2. Navega hasta el directorio donde guardaste el archivo sistema_gestion_estudiantes_gui.py .

```
cd ruta\al\directorio\propuestaRony
```

3. Ejecuta el siguiente comando:

```
python sistema_gestion_estudiantes_gui.py
```

Esto lanzará la ventana principal de la aplicación.

4. Interfaz Principal

Al iniciar la aplicación, verás la ventana principal que consta de:

- Barra de Botones (Superior) : Contiene botones para realizar las diferentes operaciones del sistema.

- Tabla de Estudiantes (Central) : Muestra la lista de estudiantes con su "Carné", "Nombre", "Materias" y "Promedio". Puedes desplazarte verticalmente si hay muchos estudiantes.

5. Funcionalidades Detalladas

A continuación, se describe la función de cada botón en la interfaz:

- Agregar Estudiante :

- Abre una nueva ventana (formulario) para ingresar los datos de un nuevo estudiante:

  - Nombre Completo.

  - Año de Inscripción (un número entre 20 y 25, que corresponde a los dos dígitos YY en el carné 0905-YY-XXXXX).

  - Materias (separadas por comas, ej: Matemáticas,Física,Programación).

  - Promedio (un número entre 0.0 y 10.0).

- Al guardar, se genera un carné único para el estudiante y se añade a la lista. La tabla principal se actualiza.

- Se realizan validaciones para asegurar que los datos sean correctos (ej. año dentro del rango, promedio numérico).

- Eliminar Estudiante :

- Solicita mediante un cuadro de diálogo que ingreses el carné del estudiante que deseas eliminar.

- Si el carné existe, el estudiante es eliminado del sistema. La tabla principal se actualiza.

- Muestra un mensaje de confirmación o error.

- Buscar Estudiante :

- Solicita mediante un cuadro de diálogo un término de búsqueda. Puedes buscar por carné exacto o por parte del nombre (no distingue mayúsculas/minúsculas).

- Si se encuentran coincidencias, la tabla principal se actualizará para mostrar solo los estudiantes que coincidan con la búsqueda.

- Si no se encuentran, la tabla se mostrará vacía o con un mensaje (dependiendo de la implementación exacta, usualmente se vacía y se espera a "Refrescar Lista"). Para ver todos los estudiantes de nuevo, usa el botón "Refrescar Lista".

- Promedio Superior a... :

- Solicita mediante un cuadro de diálogo que ingreses un umbral de promedio (ej. 7.5).

- La tabla principal se actualizará para mostrar solo los estudiantes cuyo promedio sea superior al umbral ingresado.

- Para ver todos los estudiantes de nuevo, usa el botón "Refrescar Lista".

- Materias de Estudiante :

- Solicita mediante un cuadro de diálogo el carné del estudiante cuyas materias deseas consultar.

- Si el estudiante es encontrado, se muestra una ventana con el nombre del estudiante y la lista de sus materias inscritas.

- Muestra un mensaje si el estudiante no es encontrado.

- Promedio General :

- Calcula el promedio de las calificaciones de todos los estudiantes registrados en el sistema.

- Muestra el resultado en un cuadro de mensaje.

- Si no hay estudiantes, indica que no se puede calcular.

- Refrescar Lista :

- Vuelve a cargar y mostrar todos los estudiantes registrados en la tabla principal. Es útil después de una búsqueda o filtro para ver la lista completa nuevamente.

6. Estructuras de Datos Utilizadas (Internas)

El sistema utiliza internamente las siguientes estructuras de datos de Python para gestionar la información, tal como se describe en los comentarios del código:

- Listas (estudiantes) : Para almacenar la colección principal de todos los estudiantes. Cada elemento de la lista es un diccionario que representa a un estudiante.

- Diccionarios (para cada estudiante) : Para representar la información detallada de cada estudiante con pares clave-valor (ej. {"Nombre": "Ana", "Carné": "0905-23-01001", ...}).

- Sets (carnes_unicos) : Para garantizar que cada carné de estudiante sea único y para permitir una verificación rápida de la existencia de un carné.

- Tuplas (materias_disponibles_opciones) : Se utiliza para definir un catálogo fijo de opciones de materias (nombre y código) que se usan para poblar los datos iniciales de los estudiantes. Las materias asignadas a cada estudiante se guardan como una lista de strings (nombres de materias) dentro del diccionario del estudiante.

7. Notas Adicionales

- Población Inicial : Al iniciar, el sistema crea automáticamente 30 estudiantes con datos aleatorios (nombres, años de inscripción, materias y promedios) para que puedas probar las funcionalidades inmediatamente.

- Generación de Carnés : Los carnés se generan en el formato 0905-YY-XXXXX , donde YY es el año de inscripción (ultimos dos dígitos) y XXXXX es un número correlativo único.

- Manejo de Errores : La aplicación incluye manejo básico de errores y validaciones de entrada, mostrando mensajes al usuario a través de cuadros de diálogo.

Uso de Estructuras de Datos en el Sistema de Gestión de Estudiantes

Listas (estudiantes)

Propósito

La lista estudiantes sirve como la colección principal que almacena todos los registros de estudiantes en el sistema.

Características y Justificación

Ordenada y mutable: Permite mantener los estudiantes en un orden específico y modificar la colección según sea necesario

Flexibilidad

Facilita operaciones como agregar, eliminar y modificar estudiantes

Acceso por índice

Aunque no es el método principal de búsqueda, permite acceso secuencial cuando sea necesario

Almacena diccionarios

Cada elemento de la lista es un diccionario que representa un estudiante completo

Ejemplo de Uso

Diccionarios (para cada estudiante)

Propósito

Cada estudiante está representado por un diccionario que contiene toda su información estructurada.

Características y Justificación

Estructura clave-valor: Organiza la información del estudiante de forma clara y accesible

Legibilidad

Las claves descriptivas ('Nombre', 'Carné', etc.) hacen el código más comprensible

Acceso eficiente

Permite recuperar información específica de un estudiante en tiempo constante (O(1))

Flexibilidad

Puede expandirse fácilmente para incluir nuevos atributos si los requisitos cambian

Estructura Actual

```
estudiante = {
    "Nombre": nombre,
    "Carné": carne,
    "Materias": materias,
    "Promedio": promedio
```

Sets (carnes_unicos)

Propósito

Garantizar que todos los números de carné en el sistema sean únicos y permitir verificación rápida de existencia.

Características y Justificación

Elementos únicos: Automáticamente evita duplicados

Búsqueda eficiente

Operaciones de verificación de membresía (como comprobar si un carné existe) son O(1) en promedio

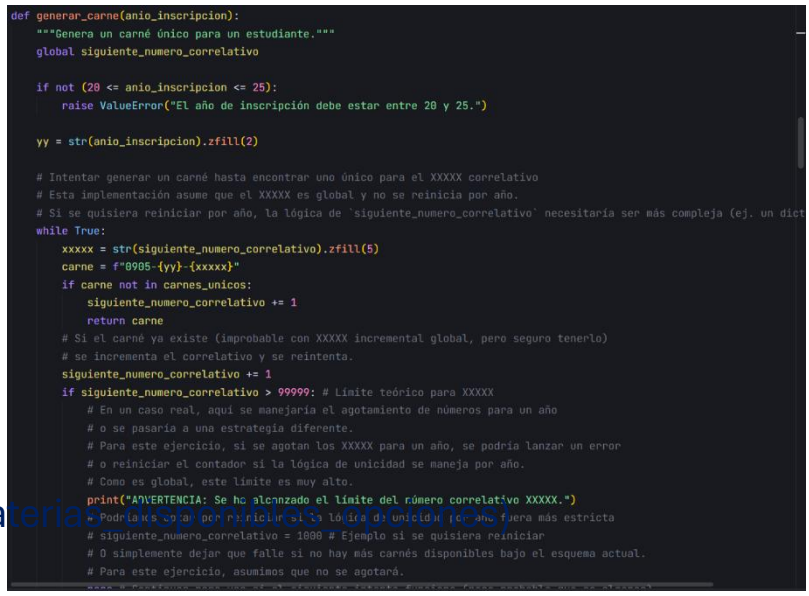
Optimización

Mucho más eficiente que buscar en una lista, especialmente cuando el número de estudiantes crece

Operaciones rápidas

Permite realizar operaciones como uniones, intersecciones y diferencias de forma optimizada

Ejemplo de Uso



```python
def generar_carne(anio_inscripcion):
    """Genera un carné único para un estudiante."""
    global siguiente_numero_correlativo

    if not (20 <= anio_inscripcion <= 25):
        raise ValueError("El año de inscripción debe estar entre 20 y 25.")

    yy = str(anio_inscripcion).zfill(2)

    # Intentar generar un carné hasta encontrar uno único para el XXXXX correlativo
    # Esta implementación asume que el XXXXX es global y no se reinicia por año.
    # Si se quisiera reiniciar por año, la lógica de `siguiente_numero_correlativo` necesitaría ser más compleja (ej. un dict
    while True:
        xxxxx = str(siguiente_numero_correlativo).zfill(5)
        carne = f"8905-{yy}-{xxxxx}"
        if carne not in carnes_unicos:
            siguiente_numero_correlativo += 1
            return carne
        # Si el carné ya existe (improbable con XXXXX incremental global, pero seguro tenerlo)
        # se incrementa el correlativo y se reintenta.
        siguiente_numero_correlativo += 1
        if siguiente_numero_correlativo > 99999: # Límite teórico para XXXXX
            # En un caso real, aquí se manejaría el agotamiento de números para un año
            # o se pasaría a una estrategia diferente.
            # Para este ejercicio, si se agotan los XXXXX para un año, se podría lanzar un error
            # o reiniciar el contador si la lógica de unicidad se maneja por año.
            # Como es global, este límite es muy alto.
            print("ADVERTENCIA: Se ha alcanzado el límite del número correlativo XXXXX.")
            # Podríamos optar por reiniciar si la lógica de unicidad lo permite, o fallar
            # siguiente_numero_correlativo = 1000 # Ejemplo si se quisiera reiniciar
            # O simplemente dejar que falle si no hay más carnés disponibles bajo el esquema actual.
            # Para este ejercicio, asumimos que no se agotará.
```

Tuplas (materias_disponibles)

Propósito

Almacenar un catálogo fijo de materias disponibles que no cambiará durante la ejecución del programa.

Características y Justificación

Inmutabilidad

Una vez creadas, no pueden modificarse, lo que las hace ideales para datos constantes

Seguridad

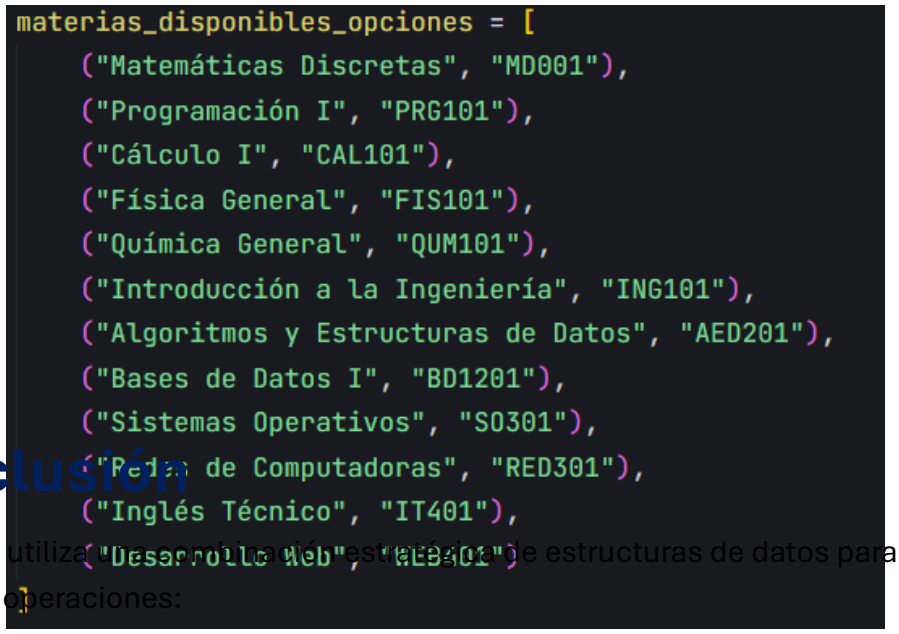
Previene modificaciones accidentales de datos críticos

Eficiencia

Más ligeras que las listas para datos que no requieren cambios

Estructura compuesta: Permite almacenar pares de valores relacionados (nombre de materia y código)

Ejemplo de Implementación



```
materias_disponibles_opciones = [
    ("Matemáticas Discretas", "MD001"),
    ("Programación I", "PRG101"),
    ("Cálculo I", "CAL101"),
    ("Física General", "FIS101"),
    ("Química General", "QUM101"),
    ("Introducción a la Ingeniería", "ING101"),
    ("Algoritmos y Estructuras de Datos", "AED201"),
    ("Bases de Datos I", "BD1201"),
    ("Sistemas Operativos", "SO301"),
    ("Redes de Computadoras", "RED301"),
    ("Inglés Técnico", "IT401"),
    ("Desarrollo Web", "DES301")
]
```

Conclusión

El sistema utiliza una combinación estratégica de estructuras de datos para optimizar diferentes operaciones:

Listas para la colección principal por su flexibilidad

Diccionarios para la representación individual de estudiantes por su claridad y acceso rápido

Sets para verificación de unicidad por su eficiencia en búsquedas

Tuplas para datos inmutables por su seguridad y rendimiento

Esta combinación asegura que el sistema sea eficiente, mantenible y escalable a medida que crece la cantidad de estudiantes.