Manual Técnico

Gestor de Notas Académico

Tito Alexander Pirir

Algoritmos

24 de octubre de 2025

Descripción Técnica General del Sistema

El Gestor de Notas Académicas es una aplicación de interfaz en línea de comandos desarrollada íntegramente en Python 3. Esta diseñado para ser autocontenido y así no depender de librerías externas.

La arquitectura del programa es de sesión única por lo que es únicamente para pruebas ya que se mantiene en la memoria RAM durante su ejecución. El flujo principal está controlado por un bucle While que presenta el menú interactivo al usuario infinitamente para gestionar las llamadas a las funciones correspondientes o hasta que se desee cerrar el programa.

Estructura General del Código

El código fuente está contenido en el archivo **gestor_notas.py**, y sigue un paradigma de programación modular para promover la legibilidad y el mantenimiento.

La estructura de datos principal está definida en una lista global `cursos = []` al inicio del programa. Esta lista funciona como la base de datos en memoria para todo el programa.

Se incluyen dos listas globales adicionales: `cola_revision = []` para simular la estructura de Cola (FIFO) y `lista_historial = []` para simular la estructura de Pila (LIFO).

Un bucle `while True` al final del archivo actúa como motor de la aplicación, encargado de mostrar el menú, leer la opción del usuario y llamar a cada función mediante una estructura condicional `if/elif/else`.

Explicación de Estructura de Datos

Lista de Diccionarios (Datos Principales)

```
cursos = [
    {'nombre': 'Precalculo', 'nota': 80.0},
    {'nombre': 'Algoritmos', 'nota': 75.0}
]
```

Lista: Se utiliza como el contenedor principal. Al ser una estructura dinámica y mutable, permite agregar (`.append()`) y eliminar (`.pop()`) elementos de forma eficiente.

Diccionario: Cada elemento de la lista es un diccionario. Se prefirió esta estructura por su claridad semántica. Las claves (`'nombre'`, `'nota'`) hacen que el acceso a los

datos sea auto-descriptivo y robusto (`curso['nota']`). Permite una fácil expansión futura (ejemplo: agregar una clave `créditos`).

Simulación de Pila (Historial)

Se utiliza la lista `lista_historial` para simular una Pila (LIFO).

Apilar: Se utiliza ‘lista_historial.append(cambio)’ para agregar un nuevo registro al final de la lista.

Ver: Para mostrar el historial, se usa `reversed(lista_historial)`, recorriendo la lista desde el último elemento ingresado hasta el primero.

Simulación de Cola (Revisión)

Se utiliza la lista ‘cola_revision’ para simular una Cola (FIFO).

Encolar (Enqueue): Se utiliza ‘cola_revision.append(curso)’ para agregar una nueva solicitud al final de la “fila”.

Desencolar (Dequeue): Se utiliza ‘cola_revision.pop(0)’ para eliminar y procesar el elemento en el índice 0 (el "frente" de la fila).

Justificación de Algoritmos

Búsqueda Lineal: Se eligió para las funciones de ‘actualizar_nota’ y ‘eliminar_curso’ por su simplicidad. Dado que la lista principal no se mantiene ordenada por defecto, la búsqueda lineal (complejidad O(n)) es el método más directo para encontrar un elemento.

Búsqueda Binaria: Se implementó para demostrar un algoritmo de búsqueda eficiente (complejidad O(log n)). Se usa creando una copia ordenada de la lista solo en el momento de la búsqueda, cumpliendo el requisito sin alterar la lista original.

Ordenamiento Burbuja e Inserción: Sirven para demostrar la capacidad de reorganizar los datos bajo diferentes criterios (nota y nombre) y operan sobre copias para no afectar el orden de registro.

Documentación Breve de cada Función

registrar_curso() Responsable de la entrada de nuevos datos. Valida que el nombre no esté vacío. Para la nota, utiliza un bloque try-except ValueError para asegurar que la entrada sea numérica y esté en el rango (0-100). Si es válido, crea el diccionario y lo agrega a cursos y registra la acción en lista_historial.

mostrar_cursos(lista_a_mostrar) Función reutilizable que recibe una lista (la original o una copia ordenada) y la imprime en un formato de tabla. Verifica primero si la lista está vacía.

actualizar_nota() Combina búsqueda y modificación. Usa un bucle for (búsqueda lineal) para encontrar el diccionario del curso. Si lo encuentra, pide y valida una nueva nota (reutilizando la lógica de try-except). Registra el cambio (nota anterior y nueva) en lista_historial.

eliminar_curso() Usa enumerate() para obtener el índice (i) del curso a eliminar. Pide confirmación (s/n) antes de proceder. Si se confirma, usa cursos.pop(i) para eliminar el elemento por su índice y registra la eliminación en lista_historial.

calcular_promedio() Suma las notas de todos los cursos y las divide por el total de cursos (len(cursos)). Muestra el resultado formateado a dos decimales usando una f-string ({promedio:.2f}).

contar_cursos() Recorre la lista y usa contadores para clasificar los cursos según la condición nota >= 60.

buscar_curso_lineal() Implementa una búsqueda secuencial simple. Recorre la lista y compara el nombre (usando .lower() para ser insensible a mayúsculas) con el término buscado.

ordenar_por_nombre() (Inserción) Implementa el algoritmo de **Ordenamiento por Inserción** sobre una copia de la lista cursos para ordenarla alfabéticamente.

ordenar_por_nota() (Burbuja) Implementa el algoritmo de **Ordenamiento Burbuja** sobre una copia de la lista cursos para ordenarla por nota de forma descendente.

buscar_curso_binaria() Implementa la **Búsqueda Binaria**. Primero, crea una copia ordenada de la lista usando sorted(). Luego, aplica el algoritmo de división y conquista (punteros bajo, medio, alto) para encontrar el curso.

solicitudes_revision() Presenta un sub-menú para gestionar la cola_revision, permitiendo agregar (append), procesar (pop(0)) o ver las solicitudes.

historial_cambios() Muestra los contenidos de lista_historial usando reversed() para cumplir con el orden LIFO (Pila).

Diagrama General del Sistema

```
INICIO
    Crear lista global: cursos
    Crear lista global: cola_revision
    Crear lista global: lista_historial

    MIENTRAS (Verdadero)
        Imprimir "SISTEMA DE GESTIÓN ACADÉMICA"
        Imprimir "1. Agregar nuevo curso"
        Imprimir "2. Ver todos los cursos"
        Imprimir "3. Actualizar nota"
        Imprimir "4. Eliminar curso"
        Imprimir "5. Calcular promedio general"
        Imprimir "6. Cursos aprobados y reprobados"
        Imprimir "7. Buscar curso por nombre (Búsqueda Lineal)"
        Imprimir "8. Ordenar por nombre"
        Imprimir "9. Ordenar por nota"
        Imprimir "10. Buscar curso por nombre (Búsqueda Binaria)"
        Imprimir "11. Cola de solicitudes de revisión"
        Imprimir "12. Historial de cambios"
        Imprimir "13. Salir del sistema"

        Leer opcion

        SI (opcion == "1") ENTONCES
            Llamar a registrar_curso()
        SINO SI (opcion == "2") ENTONCES
            Llamar a mostrar_cursos(cursos)
        SINO SI (opcion == "3") ENTONCES
            Llamar a actualizar_nota()
        SINO SI (opcion == "4") ENTONCES
            Llamar a eliminar_curso()
        SINO SI (opcion == "5") ENTONCES
            Llamar a calcular_promedio()
        SINO SI (opcion == "6") ENTONCES
            Llamar a contar_cursos()
        SINO SI (opcion == "7") ENTONCES
            Llamar a buscar_curso_lineal()
        SINO SI (opcion == "8") ENTONCES
```

```
        Llamar a ordenar_por_nombre()
      SINO SI (opcion == "9") ENTONCES
        Llamar a ordenar_por_nota()
      SINO SI (opcion == "10") ENTONCES
        Llamar a buscar_curso_binaria()
      SINO SI (opcion == "11") ENTONCES
        Llamar a solicitudes_revision()
      SINO SI (opcion == "12") ENTONCES
        Llamar a historial_cambios()
      SINO SI (opcion == "13") ENTONCES
        Imprimir "Gracias por usar el Gestor de Notas Académicas."
        Romper bucle
      SINO
        Imprimir "Opción inválida. Intente de nuevo."
      FIN SI
    FIN MIENTRAS
FIN
```