Borrador de Manual Técnico y Manual de Usuario

Proyecto: Gestor de Notas Académicas

Índice

1. Introducción

Este documento presenta el **Manual Técnico** y **Manual de Usuario** para el proyecto "Gestor de Notas Académicas". El sistema es una aplicación de consola en Python que permite registrar cursos con sus notas, buscar, ordenar, actualizar, eliminar, manejar una cola de solicitudes de revisión y llevar un historial de cambios; además soporta persistencia en un archivo JSON y exportación a CSV.

2. Alcance

El Gestor de Notas está pensado para usarse desde la consola por estudiantes o administrativos que deseen llevar un control sencillo de cursos y notas. Está diseñado para ser ligero, extensible y fácil de entender.

3. Requisitos

Software

- Python 3.8 o superior.

Librerías estándar requeridas

- json (incluido en Python)
- csv (incluido en Python)

Archivo de datos

- datos_academicos.json (creado automáticamente en la primera ejecución si no existe)

4. Estructura de archivos y datos

Archivos principales

- gestor_notas.py — código fuente principal que contiene toda la lógica.
- datos_academicos.json — archivo de persistencia donde se guardan cursos, historial y revisiones.
- mis_notas.csv (o nombre que el usuario elija) — archivo opcional generado por la función de exportación.

Estructura de datos (en memoria)

- cursos: lista de diccionarios. Cada diccionario tiene las claves:
    - nombre (string)

    - nota (int)

- historial_cambios: lista (pila) de strings con descripciones de las acciones realizadas.
- solicitudes_revision: lista (cola) de strings con los nombres de cursos que requieren revisión.

5. Manual Técnico

5.1 Arquitectura general

Aplicación de consola: un único módulo con funciones agrupadas por responsabilidades (persistencia, operaciones CRUD, utilidades de ordenamiento y búsqueda, gestión de estructuras LIFO/FIFO, exportación). El flujo principal está en main() que muestra un menú y despacha a funciones según la opción seleccionada.

5.2 Descripción de módulos / funciones

A continuación, se listan las funciones con su propósito, entradas y salidas.

Funciones de persistencia

- guardar_datos()
    - Propósito: serializar las estructuras cursos, historial_cambios y solicitudes_revision a ARCHIVO_DATOS (datos_academicos.json).
    - Entradas: variables globales.
    - Salidas: archivo en disco, mensajes en consola en caso de error.
- cargar_datos()
    - Propósito: cargar datos desde ARCHIVO_DATOS al iniciar la aplicación.
    - Errores manejados: archivo no encontrado, JSON corrupto, otros errores generales.

Funciones de la aplicación (operaciones del menú)

- mostrar_menu() — Imprime el menú principal.
- registrar_curso() — Solicita nombre y nota; valida duplicados; añade a cursos y registra en historial_cambios.
- mostrar_cursos() — Lista los cursos y notas.
- calcular_promedio() — Calcula promedio aritmético de las notas.

- contar_cursos_estado() — Cuenta aprobados (nota >= 60) y reprobados.
- buscar_curso_lineal() — Búsqueda lineal por nombre (case-insensitive).
- actualizar_nota() — Actualiza la nota de un curso existente y registra la acción en historial_cambios.
- eliminar_curso() — Elimina un curso y registra la acción.
- ordenar_burbuja_nota() — Ordena una copia de cursos por nota usando Bubble Sort; pregunta si guardar el orden.
- ordenar_insercion_nombre() — Ordena una copia de cursos por nombre usando Insertion Sort; pregunta si guardar el orden.
- buscar_curso_binaria() — Realiza búsqueda binaria en una copia de cursos ordenada por nombre (case-insensitive).
- gestionar_cola_revision() — Submenú para encolar, procesar o listar solicitudes de revisión (FIFO).
- mostrar_historial_pila() — Muestra el historial_cambios en orden LIFO.
- exportar_a_csv() — Crea un archivo CSV con encabezados nombre_curso,nota.

Punto de entrada

- main() — Carga datos, ciclo principal de menú y guarda datos al salir.

5.3 Estructuras de datos

- cursos: List[Dict[str, Any]], por ejemplo: [{'nombre': 'Matemáticas', 'nota': 85}, ...].
- historial_cambios: List[str] donde cada entrada es una descripción de acción.
- solicitudes_revision: List[str] funcionando como cola FIFO.

5.4 Persistencia

- Formato: JSON con tres claves principales: cursos, historial, revisiones.
- Ventajas: legible, fácil de manipular.
- Consideraciones: no hay bloqueo de archivos; si dos instancias escriben simultáneamente pueden corromper el archivo. Para uso sencillo de un solo usuario en consola está bien.

5.5 Algoritmos implementados

- **Bubble Sort** (ordenar por nota): complejidad O(n^2). Implementado sobre una copia para no modificar la lista original salvo que el usuario lo confirme.
- **Insertion Sort** (ordenar por nombre): complejidad O(n^2). Implementado sobre una copia.
- **Búsqueda Lineal**: O(n), útil para listas no ordenadas.
- **Búsqueda Binaria**: O(log n), **requiere** lista ordenada alfabéticamente.

5.6 Manejo de errores y validaciones

- Validación de nombre no vacío al registrar.
- Prevención de duplicados (comparación case-insensitive).
- Validación de rango de nota (0-100) y manejo de ValueError en conversiones a entero.
- Manejo de FileNotFoundError y JSONDecodeError al cargar datos.
- Manejo de IOError al guardar o exportar CSV.

5.7 Pruebas y casos de prueba

Proponer pruebas unitarias y manuales.

Pruebas unitarias sugeridas

- Registrar curso válido → verificar que se agrega y que historial crece.
- Registrar curso duplicado → no debe agregarse.
- Actualizar nota válida → nota cambia y historial registra cambio.
- Eliminar curso existente → se elimina y historial registra.
- Ordenamiento por nota y por nombre → comprobar estabilidad (según implementación) y resultado.
- Búsqueda binaria → encontrar y no encontrar casos.
- Exportar CSV → archivo creado con cabecera correcta.
- Carga y guardado de JSON → persistencia consistente.

Casos de prueba manual

- Intentar ingresar notas no numéricas.

- Intentar ordenar cuando la lista está vacía.
- Procesar cola vacía.

5.8 Extensiones sugeridas y mejoras futuras

- Validación avanzada de nombres (caracteres, longitud máxima).
- Manejo de usuarios (login) para varios perfiles.
- Concurrencia y bloqueo de archivo o uso de base de datos ligera.
- Reportes estadísticos.

5.9 Notas de implementación y convenciones

- Convención de nombres: snake_case.
- Uso de variables globales para sencillez — en proyectos más grandes se recomienda encapsular en clases.
- Documentar con docstrings y comentar bloques críticos.

6. Manual de Usuario

6.1 Introducción rápida

El Gestor de Notas permite llevar un registro sencillo de cursos y notas desde la consola. Permite ordenar, buscar, actualizar notas, solicitar revisiones y exportar a CSV.

6.2 Instalación y ejecución

1. Tener Python 3.8+ instalado.
2. Guardar el archivo gestor_notas.py en una carpeta.
3. Abrir la terminal/símbolo del sistema y navegar a la carpeta.
4. Ejecutar:

python gestor_notas.py

Al iniciar el programa, éste intentará cargar datos_academicos.json si existe.

6.3 Flujo de trabajo (paso a paso)

- Al arrancar verá el menú numérico.
- Seleccione la opción que desee escribiendo el número y presionando Enter.
- Después de cada acción, el programa volverá al menú (presione Enter para continuar cuando se le solicite).
- Para salir, elija la opción 14 (el programa guardará los datos automáticamente antes de cerrar).

6.4 Descripción de cada opción del menú

(Resumido — consulte el manual técnico para detalles de implementación)

1. **Registrar nuevo curso**: Ingrese el nombre del curso y la nota (0–100). No acepta duplicados.
2. **Mostrar todos los cursos y notas**: Lista numerada de cursos.
3. **Calcular promedio general**: Muestra promedio con dos decimales.
4. **Contar cursos aprobados y reprobados**: Considera aprobado nota >= 60.
5. **Buscar curso (Búsqueda Lineal)**: Busqueda por nombre (no sensible a mayúsculas).
6. **Actualizar nota de un curso**: Ingrese nombre, luego la nueva nota.

7. **Eliminar un curso**: Elimina curso por nombre.
8. **Ordenar cursos por nota (Burbuja)**: Ordena y pregunta si desea guardar el nuevo orden.
9. **Ordenar cursos por nombre (Inserción)**: Ordena alfabéticamente y pregunta si desea guardar.
10. **Buscar curso (Búsqueda Binaria)**: Opera sobre una copia ordenada por nombre.
11. **Gestionar cola de solicitudes de revisión**: Sub-opciones para agregar, procesar o ver la cola.
12. **Mostrar historial de cambios (Pila)**: Muestra acciones recientes en orden inverso.
13. **Exportar notas a CSV**: Crea un archivo .csv con dos columnas: nombre_curso y nota.
14. **Salir**: Guarda datos y cierra la aplicación.

6.5 Ejemplos de uso

- Registrar curso:
    - Opción 1 → nombre: Física → nota: 78.
- Actualizar nota:
    - Opción 6 → nombre: Física → nueva nota: 82.
- Exportar:
    - Opción 13 → nombre archivo: notas_semestre.csv → archivo en la misma carpeta.

6.6 Exportar e importar datos

- **Exportar a CSV**: Desde la opción 13 se crea un CSV; abrirlo con Excel o LibreOffice.
- **Importar**: Actualmente no hay una opción explícita de importación desde CSV, pero puede editar datos_academicos.json manualmente respetando la estructura JSON (ver anexo) o ampliar la aplicación para incluir una función de importación.

6.7 Solución de problemas comunes (FAQ)

- **El programa no encuentra datos_academicos.json**: Es normal si es la primera ejecución; el programa crea uno al salir.

- **Error al guardar CSV**: Verifique permisos de escritura y que el archivo no esté abierto por otra aplicación.
- **Se ingresó una nota no válida**: Ingrese un número entero entre 0 y 100.
- **Curso no encontrado al actualizar/eliminar**: Asegúrese de escribir el nombre correctamente; la búsqueda no hace tolerancia por errores tipográficos.