Manual de Gestor de Notas

Tabla de contenidos

---

1. Introducción

Este documento describe un programa en Python para gestionar un listado de cursos y sus notas, con funcionalidades para registrar, mostrar, ordenar, buscar, actualizar, eliminar, y simular una cola de solicitudes de revisión. Contiene un **Manual Técnico** (para desarrolladores/mantenedores) y un **Manual de Usuario** (para usuarios finales / docentes / administradores).

2. Alcance

El programa es una utilidad de consola (CLI) diseñada para administrar de forma simple cursos y sus notas en memoria. No persiste datos en disco ni utiliza bases de datos; todas las estructuras residen en listas y se pierden al cerrar el programa.

3. Requisitos

- Python 3.8+ (recomendado 3.10+)
- Sistema operativo: multiplataforma (Windows, macOS, Linux)

- No requiere librerías externas (usa solo la biblioteca estándar)
- Permisos para ejecutar scripts en la terminal

4. Instalación y ejecución

1. Guardar el código en un archivo: `gestor_cursos.py`.
2. Abrir terminal / cmd en la carpeta del archivo.
3. Ejecutar:

```
python gestor_cursos.py
```

4. Seguir las opciones del menú que aparecen en pantalla.

5. Estructura del código y datos

Variables principales (en memoria):

- `cursos`: lista de diccionarios con estructura `{"nombre": str, "nota": float}`. Inicialmente contiene varios cursos ejemplo.
- `cola_revisiones`: lista que actúa como cola FIFO de nombres de cursos solicitados para revisión.
- `historial_cambios`: lista que funciona como pila (se muestran invertidos) con strings que documentan cambios de notas.

Arquitectura:

- Cada operación del menú es una función independiente.
- `opciones`: diccionario que mapea la entrada numérica del usuario a la función correspondiente.
- Loop principal: muestra el menú, lee opción y ejecuta la función asociada hasta salir.

6. Descripción de funciones (Manual técnico)

A continuación se describen las funciones principales, su propósito, inputs esperados, outputs y comportamiento interno.

`mostrar_menu()`

- Propósito: imprimir las opciones disponibles al usuario.
- Entrada: ninguna.
- Salida: texto en consola.

`registrar_curso()`

- Propósito: registrar uno o varios cursos nuevos.
- Flujo:
  - Solicita la cantidad `n` de cursos a registrar.
  - Para cada curso pide `nombre` y `nota`.
  - Validaciones:
    - `n` debe ser entero.
    - `nombre` no puede existir ya (comparación case-insensitive).
    - `nota` debe ser float entre 0 y 100.
- Efecto: agrega nuevos diccionarios a `cursos`.
- Errores: maneja `ValueError` al parsear números.

`mostrar_cursos()`

- Propósito: listar todos los cursos y notas.
- Si `cursos` vacío muestra mensaje apropiado.

`promedio()`

- Propósito: calcular promedio aritmético simple de las notas.
- Salida: `Promedio general: X.XX` o mensaje si no hay cursos.

`cursos_aprobados_reprobados()`

- Propósito: separar cursos aprobados (`nota >= 61`) y reprobados (`nota < 61`) y mostrarlos.
- Salida: listas impresas en consola.

`buscar_curso()` **(búsqueda lineal)**

- Propósito: buscar por nombre (lineal).
- Flujo: solicita nombre, itera sobre `cursos`, comparación `lower()` para case-insensitive.
- Salida: muestra nombre, nota y estado (Aprobado/Reprobado) o mensaje si no existe.

`actualizar_nota()`

- Propósito: actualizar la nota de un curso existente.
- Flujo:
  - Solicita nombre (case-insensitive).
  - Si encuentra el curso:
    - Muestra nota actual.
    - Solicita `nueva` nota y valida rango.
    - Agrega entrada a `historial_cambios` con el formato `"Nombre: vieja -> nueva"`.

        - Actualiza `c["nota"]`.
- Validaciones y errores cubiertos.

`borrar_curso()`

- Propósito: eliminar un curso del listado.
- Flujo: solicita nombre; si existe, elimina con `cursos.remove(c)`.

`ordenar_por_nota()` **(burbuja)**

- Propósito: ordenar la lista `cursos` por `nota` ascendente usando algoritmo burbuja.
- Implementación: doble bucle; intercambio de objetos completos en la lista.
- Nota técnica: algoritmo $O(n^2)$. Para listas grandes, considerar `list.sort()` con `key=lambda c: c['nota']`.

`ordenar_por_nombre()` **(burbuja)**

- Propósito: ordenar `cursos` alfabéticamente por `nombre` (case-insensitive).
- Misma observación $O(n^2)$.

`buscar_curso_binario()`

- Propósito: búsqueda binaria por `nombre`.
- Flujo:
    - Llama a `ordenar_por_nombre()` para garantizar orden previo.
    - Realiza búsqueda binaria por `nombre` (lowercase).
- Nota técnica: requiere que la lista esté ordenada por `nombre`. Complejidad O(log n) tras ordenamiento.

`simular_cola_revision()`

- Propósito: simular cola FIFO de solicitudes de revisión de notas.
- Submenú interno con opciones:
    1. Agregar solicitud: valida que curso exista y `append` a `cola_revisiones`.
    2. Atender solicitud: `pop(0)` de `cola_revisiones`, encuentra curso y solicita nueva nota (registra en `historial_cambios`).
    3. Ver cola: imprime contenido.
    4. Salir del submenú.
- Nota: eliminación con `pop(0)` es O(n) para listas grandes; usar `collections.deque` para eficiencia en producción.

`mostrar_historial()`

- Propósito: mostrar `historial_cambios` en orden cronológico inverso (último cambio primero).
- Muestra cada entrada.

`salir()`

- Propósito: mostrar mensaje y romper el loop enviando `False`.

`opciones` y loop principal

- `opciones`: diccionario que asocia números a funciones.
- Loop: muestra menú y ejecuta `opciones[op]()`. Si la función devuelve `False`, se rompe el loop.

7. Flujo de uso (Manual de usuario)

A continuación se describe cómo usar la aplicación paso a paso (sin entrar en implementación interna).

Inicio

- Ejecuta `python gestor_cursos.py`.
- Verás el menú numerado. Introduce el número de la acción que quieres ejecutar.

Acciones comunes

1. **Registrar nuevo curso**: elegir opción `1`, indicar cuántos cursos y luego nombre y nota. No se permiten duplicados.
2. **Mostrar todos los cursos**: opción `2`.
3. **Calcular promedio**: opción `3`.
4. **Contar aprobados/reprobados**: opción `4`. Se muestran listados.
5. **Buscar curso (lineal)**: opción `5`, escribe el nombre exacto (mayúsculas/minúsculas no importan).
6. **Actualizar nota**: opción `6`, escribe nombre del curso, ingresa nueva nota válida (0-100).
7. **Eliminar curso**: opción `7`, escribe nombre del curso a eliminar.
8. **Ordenar por nota**: opción `8`. Ordena ascendente por nota.
9. **Ordenar alfabéticamente por nombre**: opción `9`.
10. **Buscar (binaria)**: opción `10`. Internamente ordena por nombre y busca más rápido.
11. **Simular cola de revisiones**: opción `11`. Submenú para agregar, atender o ver la cola.
12. **Mostrar historial**: opción `12`. Muestra cambios de nota recientes.
13. **Salir**: opción `13`.

Consejos de uso

- Escribir el nombre completo del curso tal como se registró (aunque la comparación es insensible a mayúsc/minúsc).
- Las notas deben ser valores numéricos entre 0 y 100.

- Para búsquedas rápidas por nombre, usar opción 10 (binaria) si hay muchos cursos.

8. Ejemplos de interacción

Registrar un curso

```
Elija una opción: 1
¿Cuántos cursos desea registrar? 1
Ingrese el nombre del curso 1: Física
Ingrese la nota de Física: 92
Solicitud completada.
```

Buscar curso

```
Elija una opción: 5
Ingrese el curso a buscar: física
Física - Nota: 92 (Aprobado)
```

Simular cola

```
Elija una opción: 11

1. Agregar solicitud
2. Atender solicitud
3. Ver cola
4. Salir
Opción: 1
Curso a revisar: física
Solicitud agregada.

Opción: 2
Revisando Física (nota actual 92)
Nueva nota: 95
Nota revisada.
```

9. Validaciones y manejo de errores

- Todas las entradas numéricas (cantidad de cursos, notas, selección del menú) están envueltas en `try/except ValueError`.
- Validaciones adicionales:
  - Nota entre 0 y 100.
  - Nombre no duplicado al registrar.
  - Búsquedas y actualizaciones devuelven mensaje si el curso no existe.
- Posibles fallos no cubiertos:
  - Entrada vacía para nombres (podrías agregar `if not nombre: continuar`).
  - Manejo de caracteres Unicode o acentos si se desean normalizar.

10. Pruebas sugeridas

- Pruebas unitarias (pytest) recomendadas para:
  - Registrar curso nuevo (caso normal y duplicado).
  - Actualizar nota (válido/ inválido).
  - Búsqueda lineal (existe/no existe).
  - Búsqueda binaria con lista vacía / un elemento / muchos elementos.
  - Ordenamiento por nota y nombre — verificar estabilidad y correcto orden.
  - Cola: agregar múltiples solicitudes y atender todas.
- Casos borde:
  - Notas límite (0, 61, 100).
  - Nombres con mayúsculas/acentos.
  - Intentar eliminar curso inexistente.

11. Registro de cambios / historial

- `historial_cambios` guarda cadenas con formato `"Nombre: vieja -> nueva"`.
- Para mayor trazabilidad, sugerir almacenar estructuras con timestamps: `{"curso":..., "antigua":..., "nueva":..., "ts": "YYYY-MM-DD HH:MM:SS"}`.

12. Seguridad y buenas prácticas

- No almacenar datos sensibles en texto plano (no aplica directamente aquí).
- Evitar `input()` sin saneamiento cuando se integre en sistemas más grandes.
- Para persistencia, migrar a base de datos o ficheros con control de acceso.
- Validar y sanitizar nombres si la app se expone a usuarios no confiables.

13. Posibles mejoras (priorizadas)

1. **Persistencia**: guardar `cursos` y `historial_cambios` en JSON/CSV o base de datos SQLite al cerrar/abrir programa.
2. **UI**: crear interfaz web simple (Flask/FastAPI) o GUI (Tkinter) para facilitar uso.
3. **Optimización de cola**: usar `collections.deque` para O(1) en pop/append.
4. **Uso de `sort()`**: reemplazar burbuja por `cursos.sort(key=...)` para eficiencia.
5. **Internacionalización (i18n)**: soporte para varios idiomas.
6. **Export/Import**: funciones para exportar/importar cursos en CSV/JSON.
7. **Reportes**: generar PDF/CSV con resúmenes y estadísticas.
8. **Pruebas automatizadas**: añadir suite de tests (pytest) y CI.

14. Glosario

- CLI: Interfaz de línea de comandos.
- FIFO: First In First Out — comportamiento de la cola.
- $O(n^2)$: Complejidad cuadrática (algoritmo burbuja).
- `historial_cambios`: registro de modificaciones (tipo pila en la presentación).

15. Contacto / Soporte

- Autor del script: Jorge Pablo Tepet Tzul
- Mail: jorgepablo48@gmail.com

- Recomendación: mantener copia del archivo `gestor_cursos.py` y versionarlo con git.