MANUAL TÉCNICO

GESTOR DE NOTAS ACADÉMICAS

Versión 1.0

INDICE

INTRODUCCIÓN

1.1 Propósito del Documento

Este manual técnico proporciona información detallada sobre la implementación, arquitectura y funcionamiento interno del Gestor de Notas Académicas. Está dirigido a desarrolladores, programadores y personal técnico que requiera comprender, mantener o extender el sistema.

1.2 Alcance del Sistema

El Gestor de Notas Académicas es una aplicación de consola desarrollada en Python que permite la gestión completa de cursos y calificaciones mediante una interfaz de menú interactivo.

1.3 Tecnologías Utilizadas

- Lenguaje: Python 3.x
- Paradigma: Programación Procedural
- Interfaz: Consola/Terminal
- Estructuras de Datos: Listas, Tuplas

ARQUITECTURA DEL SISTEMA

1.4 Diagrama de Arquitectura

El sistema sigue una arquitectura modular y procedural, organizada en tres capas funcionales:

- Interfaz De Usuario
- Capa De Funciones
- Capa De Datos

1.5 Patrón de Diseño

El sistema utiliza un enfoque procedural con las siguientes características:

- Modularidad: Cada funcionalidad está encapsulada en una función independiente.
- Estado Global: Las variables **notas** e **historial_pila** mantienen el estado de la aplicación.
- Controlador Principal: Un bucle **while** gestiona la interacción con el usuario.

REQUISITOS DEL SISTEMA

3.1 Requisitos de Software

Sistema Operativo: Windows, Linux, macOS

Python: Versión 3.6 o superior

Terminal/Consola: Cualquier emulador de terminal estándar

3.2 Dependencias

El sistema utiliza únicamente funciones built-in de Python, por lo que no requiere librerías externas

ESTRUCTURA DE DATOS

4.1 Variable Global: notas

Es una lista mutable que almacena tuplas con información de cursos y sus respectivas notas.

```
notas = []  # List[Tuple[str, float]]
```

Ejemplo de Estructura: [("Matemáticas", 85.0), ("Física", 72.5)]

4.2 Variable Global: historial_pila

Es una lista (List[str]) que funciona como Pila (LIFO) para almacenar el historial de cambios.

Operación de Push: Se utiliza historial_pila.append(mensaje).

Acceso LIFO: Se accede usando índices negativos historial_pila[-i] para mostrar el más reciente primero.

Formatos de Mensaje:

Actualización: "ACTUALIZADO: 'NombreCurso' de NotaAnterior a NotaNueva"

Eliminación: "ELIMINADO: 'NombreCurso' con nota Nota"

DESCRIPCION DE FUNCIONES

5.1 mostrarTitulo()

- Propósito: Muestra el título del sistema y menú principal.
- Algoritmo/Lógica: Imprime texto en consola.
- Complejidad: O(1)

5.2 registrar_nuevo_curso()

- Propósito: Registra un nuevo curso con su nota.
- Algoritmo/Lógica: Bucle **while** para validar que la nota esté en rango $[0, 100]$ y sea numérica.
- Complejidad: O(1)

5.3 mostrar_cursosYnotas()

- Propósito: Muestra todos los cursos y notas.
- Algoritmo/Lógica: Itera sobre la lista notas y evalúa si está vacía.
- Complejidad: O(n)

5.4 promedio_de_notas()

- Propósito: Calcula y muestra el promedio general.
- Algoritmo/Lógica: Itera sobre las notas, suma y cuenta. Incluye *Return Anticipado* si la lista está vacía.
- Complejidad: O(n)

5.5 contar_aprobadosYreprobados()

- Propósito: Cuenta cursos aprobados (nota ≥ 60) y reprobados
- Algoritmo/Lógica: Iteración con condicional simple. Utiliza _ para ignorar el nombre del curso.
- Complejidad: O(n)

5.6 buscar_curso()

- Propósito: Busca un curso específico por nombre.
- Algoritmo/Lógica: Búsqueda lineal secuencial con comparación *case-insensitive* (.lower()). Usa break al encontrar.
- Complejidad: O(n)

5.7 actualizar_nota()

- Propósito: Actualiza la nota de un curso existente.
- Algoritmo/Lógica: Itera con índice (range(len(notas))). Registra el cambio en historial_pila.
- Complejidad: O(n)

5.8 eliminar_curso()

- Propósito: Elimina un curso con confirmación.
- Algoritmo/Lógica: Requiere confirmación "SI". Utiliza notas.remove(curso) y registra en el historial.
- Complejidad: O(n)

5.9 ordenar_por_nota()

- Propósito: Muestra cursos ordenados por nota.
- Algoritmo/Lógica: Utiliza el algoritmo de Ordenamiento de Burbuja (Bubble Sort). Compara e intercambia elementos adyacentes repetidamente hasta que la lista está ordenada. Este método modifica la lista global notas in-place.
- Complejidad: O(n^2)

5.10 ordenar_por_nombre()

- Propósito: Muestra cursos ordenados alfabéticamente.
- Algoritmo/Lógica: Utiliza el algoritmo de Ordenamiento por Inserción (Insertion Sort). Recorre la lista, tomando cada elemento y colocándolo en su posición correcta dentro de la sublista ya ordenada. Este método también modifica la lista global notas in-place. La comparación es case-sensitive (distingue mayúsculas y minúsculas).
- Complejidad: O(n^2)

5.11 cola_revision()

- Propósito: Gestiona una **Cola (FIFO)** de solicitudes.
- Algoritmo/Lógica: Implementada con lista local. **Enqueue**: append(). **Dequeue**: pop(0).
- Complejidad: O(n)

5.12 historial()

- Propósito: Muestra historial de cambios en orden LIFO.
- Algoritmo/Lógica: Itera usando índices negativos (historial_pila[-i]).
- Complejidad: O(n)

FLUJO DE EJECUCIÓN

6.1 Diagrama de Flujo Principal

El control del programa reside en un bucle principal que procesa las opciones del menú:

1. INICIO: Inicialización de variables globales y llamada a mostrarTitulo().
2. Bucle WHILE True: Muestra el menú, captura la opción, y ejecuta la función correspondiente.
3. FIN: El bucle termina (BREAK) solo cuando el usuario selecciona la opción 12.

6.2 Ciclo de Vida de una Operación

El ciclo de vida de una operación sigue el patrón: Usuario -> Menú -> Validación -> Ejecución -> Feedback -> Menú

MANEJO DE ERROES Y VALIDACIONES

7.1 Validaciones de Entrada

Validación de Notas: Se utiliza un bucle while para asegurar que la entrada cumpla con el tipo numérico (.isdigit()) y el rango válido (0 <= float(nota_str) <= 100).

Búsquedas Case-Insensitive: Las comparaciones de nombres de cursos se realizan utilizando .lower() para tolerar errores de mayúsculas/minúsculas del usuario.

7.2 Manejo de Listas Vacías

Todas las funciones de cálculo y consulta que operan sobre notas verifican si len(notas) == 0.

Prevención de División por Cero: Esta verificación previene implícitamente la división por cero en promedio_de_notas mediante un return anticipado.

CONSIDERACIONES TECNICAS

8.1 Limitaciones Conocidas

1. Persistencia: Los datos residen solo en memoria RAM y no se guardan entre ejecuciones
2. Escalabilidad: Las búsquedas y eliminaciones utilizan un enfoque logarítmico y lineal, que puede degradarse con muchos cursos
3. Duplicados: El sistema no previene el registro de cursos con nombres duplicados

8.2 Mejoras Futuras Recomendadas

- Persistencia de Datos: Implementar serialización JSON o integrar una base de datos
- Estructura de Datos: Usar diccionarios o hash tables para mejorar la complejidad de búsqueda a O(1)
- Interfaz: Implementar una GUI (ej. con Tkinter) o añadir colores a la consola

GLOSARIO TÉCNICO

Tupla (Tuple): Estructura de datos inmutable en Python.

Lista (List): Estructura de datos mutable en Python.

FIFO (First In, First Out): Principio de cola donde el primer elemento en entrar es el primero en salir.

LIFO (Last In, First Out): Principio de pila donde el último elemento en entrar es el primero en salir.

Case-insensitive: Comparación de cadenas sin distinción entre mayúsculas y minúsculas.

O(n): Notación Big O que indica complejidad lineal respecto al tamaño de entrada.

Timsort: Algoritmo de ordenamiento híbrido usado por Python internamente