

BORRADOR DE MANUAL TÉCNICO V1.0

Gestor de Notas Académicas

Josué Tepet
Proyecto Final de Algoritmos

Octubre de 2025
Cobertura: Avances 1 al 5

Índice

1. Introducción	3
1.1. Propósito y Audiencia del Documento	3
1.2. Alcance del Proyecto (Avances 1-5)	3
2. Arquitectura del Sistema y Modularización (Avance 4)	3
2.1. Estructura de Control Principal	3
2.2. Diseño de Subrutinas (Modularización)	3
3. Estructuras de Datos e Implementación (Avances 2 y 5)	3
3.1. Listas Paralelas: cursos y notas	3
3.2. Pila (Stack) para Historial de Acciones	4
4. Algoritmos Fundamentales	5
4.1. Búsqueda Lineal (Avances 3 y 4)	5
4.1.1. Flujo del Algoritmo	5
4.2. Algoritmos de Ordenamiento (Avance 5)	5
4.2.1. Ordenamiento por Burbuja (Bubble Sort)	5
4.2.2. Ordenamiento por Inserción (Insertion Sort)	5
5. Manejo de Errores y Excepciones Técnicas	6
5.1. Validación de Datos y Robustez del Flujo	6
5.2. Gestión de Riesgos Técnicos y Limitaciones	6

1. Introducción

1.1. Propósito y Audiencia del Documento

Este manual técnico tiene como objetivo documentar en detalle la arquitectura de software, las estructuras de datos fundamentales y los algoritmos implementados en el **Gestor de Notas Académicas**. Está dirigido a desarrolladores y personal técnico para facilitar el mantenimiento y futuras ampliaciones.

1.2. Alcance del Proyecto (Avances 1-5)

El proyecto implementa la totalidad de las funcionalidades del gestor, incluyendo las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y la implementación de estructuras de datos y algoritmos avanzados.

2. Arquitectura del Sistema y Modularización (Avance 4)

El sistema utiliza un enfoque de programación **modularizada** basado en funciones de Python, separando las responsabilidades de control de la lógica de negocio.

2.1. Estructura de Control Principal

El flujo de ejecución es centralizado en la función `menu_principal()`, que actúa como el controlador principal mediante un **bucle while** y estructuras **condicionales** (`if/elif`).

Diagrama de Flujo del Controlador:

```
[INICIO] -> [Inicializar Variables Globales]
|
V
[BUCLE MIENTRAS Opción != 9]
|
V
[Mostrar Menú] -> [LEER Opción]
|
V
[CONDICIONAL IF/ELIF]
|
V
[Llamar a Subrutina (e.g., actualizar_nota())]
|
V
[Fin de Bucle] -> [FIN]
```

2.2. Diseño de Subrutinas (Modularización)

Cada requisito funcional se encapsula en una función independiente, operando sobre las variables globales (`cursos`, `notas`, `historial_pila`).

3. Estructuras de Datos e Implementación (Avances 2 y 5)

3.1. Listas Paralelas: cursos y notas

El almacenamiento principal se realiza mediante dos listas de Python. La coherencia de los datos se mantiene asegurando que la operación de manipulación (inserción, eliminación, intercambio) se aplique en el mismo **índice** (`i`) en ambas listas.

3.2. Pila (Stack) para Historial de Acciones

- **Implementación:** Una lista simple de Python (`historial_pila`).
- **Operación PUSH (Apilar):** Se simula con el método nativo `pila.append(accion)`.
- **Función:** Registrar cada evento significativo del usuario, respetando el principio LIFO (Last-In, First-Out).

Ejemplo de Uso de Pila en Registro:

```
def registrar_nota(cursos, notas, pila):  
    ...  
    cursos.append(nombre)  
    notas.append(nota)  
    apilar_accion(pila, f"Registr {nombre}")
```

4. Algoritmos Fundamentales

4.1. Búsqueda Lineal (Avances 3 y 4)

Este algoritmo es esencial para las funcionalidades de Actualización y Eliminación. Su propósito es obtener la posición (índice) del elemento a manipular.

4.1.1. Flujo del Algoritmo

- Se utiliza `enumerate(cursos)` para recorrer la lista mientras se obtiene el índice (`i`).
- El ciclo se rompe (`break`) al encontrar una coincidencia con el nombre buscado.

Diagrama de Flujo de la Búsqueda Lineal para Eliminación:

```
[INICIO SUBROUTINA ELIMINAR]
|
V
[FOR i IN ENUMERATE(cursos)]
|
[cursos[i] == Buscado]?
/      \
[No]    [Sí]
|        |
V        V
[i++]    [ELIMINAR: cursos.pop(i)]
|        |
V        V
[FIN BUCLE]  [ELIMINAR: notas.pop(i)]
|
V
[RETORNAR]
```

4.2. Algoritmos de Ordenamiento (Avance 5)

Ambos algoritmos tienen una complejidad temporal de $O(n^2)$ y se implementan con la restricción de realizar los intercambios de datos en paralelo en las listas `cursos` y `notas`.

4.2.1. Ordenamiento por Burbuja (Bubble Sort)

- **Mecanismo:** Compara pares de elementos adyacentes y los intercambia si están desordenados.
- **Principio Técnico:** El *swap* (intercambio) utiliza la asignación paralela de Python.

Mecanismo de Intercambio Paralelo:

```
# Intercambio de notas (Burbuja)
if notas[j] > notas[j + 1]:
    # Swap de notas
    notas[j], notas[j + 1] = notas[j + 1], notas[j]
    # Swap de cursos (paralelo e indispensable)
    cursos[j], cursos[j + 1] = cursos[j + 1], cursos[j]
```

4.2.2. Ordenamiento por Inserción (Insertion Sort)

- **Mecanismo:** Construye la lista ordenada un elemento a la vez, insertando cada elemento en su posición correcta en la sublista ya ordenada.
- **Principio Técnico:** El desplazamiento de elementos mayores y la inserción de la clave se replican en ambas listas.

5. Manejo de Errores y Excepciones Técnicas

El manejo de errores en este prototipo se centra en garantizar la estabilidad del flujo de ejecución y en la validación de la entrada de usuario para prevenir fallos críticos.

5.1. Validación de Datos y Robustez del Flujo

La robustez del sistema se mantiene mediante mecanismos de control de flujo en puntos críticos de interacción con el usuario o de manipulación de estructuras de datos.

- **Validación Numérica (try/except):** Se utiliza el bloque `try-except ValueError` en las funciones de registro y actualización para atrapar y gestionar entradas no numéricas, previniendo la terminación abrupta del programa.
- **Validación de Rango:** Se implementa una condición `if (0 <= nota <= 100)` para asegurar la integridad de los datos académicos.
- **Control de Estructuras Vacías:** Las funciones `calcular_promedio` y `mostrar_notas` contienen condicionales que verifican si las listas están vacías para evitar errores de tipo `ZeroDivisionError` (división por cero) o `IndexError`.

5.2. Gestión de Riesgos Técnicos y Limitaciones

Los siguientes puntos identifican riesgos potenciales en la arquitectura y las soluciones implementadas para mitigar su impacto en la ejecución.

Riesgo Técnico	Causa y Contexto de Fallo	Mecanismo de Mitigación Implementado
Pérdida de Sincronización de Datos	Error en la manipulación de las listas paralelas donde una operación (<i>swap</i> , <i>pop</i>) solo afecta a notas o cursos .	La arquitectura de las funciones de Ordenamiento (Burbuja/Inserción) y Eliminación obliga a realizar el intercambio/eliminación en líneas adyacentes de forma paralela.
Fallo por Búsqueda Inválida	El usuario busca o intenta eliminar un curso que no está registrado.	La Búsqueda Lineal se ejecuta completamente. Si el elemento no se encuentra, la función retorna un mensaje de error sin intentar acceder a un índice inválido (IndexError).
Complejidad $O(n^2)$	Uso de los algoritmos de Ordenamiento por Burbuja e Inserción. La eficiencia se degrada notablemente con un gran volumen de datos ($N > 1000$).	Es una limitación aceptada para este prototipo. Se cumple con el requisito funcional, y el rendimiento es adecuado dado el bajo volumen de datos que se espera en un entorno de consola.