Universidad Mariano Gálvez

Curso: Algoritmos

Manual Técnico Gestor De Notas Académicas

Estudiante: Oscar Mauricio Chavez Uyú

Carne: 7590-24-13951

índice

Introducción:

El desarrollo de este proyecto se basó en la aplicación práctica de los fundamentos de programación aprendidos, lo que incluye el uso de estructuras de control esenciales como las estructuras repetitivas, condicionales y la modularización mediante funciones, además de la implementación de estructuras de datos fundamentales para la gestión eficiente de la información: se utilizaron listas, pilas y colas para organizar los datos, destacando el uso de una pila para mantener un historial de cambios que permite revertir operaciones; asimismo, se incorporaron algoritmos de ordenamiento y técnicas de búsqueda eficiente (lineal y binaria) para localizar cursos rápidamente. El propósito de este manual es, precisamente, ofrecer una visión técnica exhaustiva del *software*, detallando las estructuras internas y el código fuente con la finalidad de facilitar su comprensión, garantizar un mantenimiento efectivo y sentar las bases para futuras mejoras y expansiones.

Descripción Técnica General Del Sistema

El proyecto “Gestor De Notas Académicas” fue desarrollado en el lenguaje de programación de Python, utilizando Únicamente herramientas básicas del lenguaje, sin librerías externas. El sistema fue diseñado para ejecutarse complemente en la consola y esta estructurado de manera modular, es decir dividido en diferentes funciones que se encargan de realizar tareas específicas dentro del programa.

El sistema inicia con un menú principal que muestra todas las opciones disponibles para el usuario. Este menú se repite constantemente dentro de un ciclo while, hasta que el estudiante decide salir del programa seleccionando la opción correspondiente. Cada opción del menú está relacionada con una función específica, lo que permite mantener el código ordenado y fácil de comprender.

Los datos principales del programa se almacenan en una lista llamada cursos, la cual guarda todos los cursos registrados junto con sus respectivas notas. Cada curso se almacena como un diccionario, con dos claves: "nombre" y "nota". Esta estructura facilita la manipulación de la información, ya que se pueden agregar, buscar, modificar o eliminar elementos fácilmente mediante las funciones implementadas. Por ejemplo, cuando el usuario registra un nuevo curso, el programa solicita el nombre y la nota. Luego, esta información se guarda dentro de la lista como un nuevo diccionario. Posteriormente, otras funciones pueden acceder a estos datos para calcular el promedio, contar cuántos cursos están aprobados o buscar un curso en específico.

El sistema también aplica estructuras de datos dinámicas como pilas y colas, las cuales fueron simuladas usando listas de Python. La pila se utiliza para almacenar el historial de cambios realizados en el sistema, como cuando se actualiza o se elimina una nota. En una pila, el último elemento que se agrega es el primero en salir (principio LIFO: “Last In, First Out”). Esto permite que el programa muestre los cambios más recientes primero, tal como ocurre en un historial real.

Excelente contenido. Aquí tienes la redacción solicitada en dos párrafos grandes, manteniendo y expandiendo la información de manera fluida:

El proyecto implementa una variedad de estructuras y algoritmos esenciales, destacando el uso de la **Cola (Queue)** para simular de manera práctica la gestión de **solicitudes de revisión académica**, donde los cursos ingresados se procesan estrictamente bajo el principio *First In, First Out* (FIFO), replicando un proceso real de flujo de trabajo. Adicionalmente, el sistema integra varios **algoritmos de búsqueda y ordenamiento** para garantizar la eficiencia y la correcta presentación de los datos. En cuanto a la búsqueda, se utiliza la **búsqueda lineal** para recorrer la lista de cursos cuando no es necesario que estén ordenados, mientras que la **búsqueda binaria**, mucho más rápida y eficiente, se reserva para aquellos momentos en que la lista ha sido previamente

Estructura General Del Código

El código del proyecto “Gestor de Notas Académicas” está organizado de forma ordenada y lógica, para que sea fácil de leer, entender y mantener. El programa está dividido en funciones, cada una con una tarea específica. Esta estructura ayuda a que el código no sea repetitivo y que las acciones estén bien separadas, lo que mejora la claridad y el control del sistema. El archivo principal del programa se llama gestor_notas.py. En este archivo se encuentra todo el código fuente que controla las operaciones del sistema. Desde el inicio, se declara una lista principal llamada cursos, donde se almacenan los cursos registrados junto con sus notas. Cada curso se guarda como un diccionario con dos elementos: el nombre y la nota. Por ejemplo:

```
{"nombre": "Matemáticas", "nota": 85}
```

Después viene la función leer_opcion(), la cual lee la opción del usuario y valida que sea un número correcto. Si el usuario ingresa un valor vacío o un texto que no sea un número, el sistema muestra un mensaje de error y vuelve a pedir la opción. Esta validación se realiza con un ciclo while que se repite hasta que el usuario ingresa un valor válido.

La función ejecutar_opcion() actúa como el “cerebro” del programa. Su función es decidir qué acción se debe realizar según la opción elegida. Por ejemplo, si el usuario elige el número 1, se llama a la función registrar_curso(), que es la encargada de agregar un nuevo curso a la lista. Si elige la opción 2, se ejecuta mostrar_cursos(), y así sucesivamente.

1.registrar_curso():

Esta función sirve para **agregar un nuevo curso** al sistema junto con su nota. Primero, el programa pide al usuario que escriba el nombre del curso y luego la nota obtenida.
Antes de guardar los datos, el sistema revisa que:

- El nombre no esté vacío.
- La nota sea un número entre **0 y 100**.

Si todo está bien, los datos se guardan en la lista principal llamada cursos.
Cada curso se guarda así:

2.mostrar_cursos():

Esta función **muestra todos los cursos registrados** con sus respectivas notas.
Si la lista está vacía, el sistema informa al usuario que **no hay cursos aún registrados**.
Por ejemplo, al ejecutar esta función podría mostrarse en pantalla

```
📚 Cursos Registrados:
1. Matemáticas → Nota: 85
2. Computación → Nota: 90
3. Inglés → Nota: 78
```

Esto permite al usuario ver un resumen de todo lo que lleva ingresado.

3. calcular_promedio():

Aquí el sistema suma todas las notas de los cursos y las divide entre la cantidad total.
El resultado es el promedio general del estudiante.

Ejemplo:

```
Matemáticas: 85
Computación: 90
Inglés: 78
```

Cálculo del promedio:
(85 + 90 + 78) / 3 = **84.3**

El programa mostrará:

```
El promedio general de tus cursos es: 84.3
```

```
{"nombre": "Matemáticas", "nota": 85}
```

4. contar_aprobados_reprobados():

Esta función revisa todas las notas y cuenta cuántos cursos están aprobados (nota ≥ 60) y cuántos están reprobados (nota < 60).
Así el usuario puede saber cuántos cursos necesita mejorar.

Ejemplo de salida:

```
Cursos aprobados: 3
Cursos reprobados: 1
```

5. buscar_curso_lineal():

Esta función usa un método llamado búsqueda lineal.
Significa que el programa revisa uno por uno los cursos hasta encontrar el que el usuario busca.

Ejemplo:
El usuario busca “Matemáticas”.
El sistema recorre la lista:

1. Computación → No coincide
2. Matemáticas → ¡Coincide!

Resultado:

```
Curso encontrado: Matemáticas → Nota: 85
```

6. actualizar_curso():

Permite **modificar la nota** de un curso que ya existe.
El programa busca el curso por su nombre y luego pide ingresar la nueva nota.

Antes de guardar, verifica que el curso exista.
Ejemplo:

```
Ingrese el nombre del curso: Matemáticas
Nueva nota: 90
✅ Nota actualizada con éxito.
```

7. eliminar_curso():

Esta función borra un curso de la lista.
Primero, el sistema pide confirmar antes de eliminar para evitar errores.

Además, el curso eliminado se guarda en una pila llamada historial, lo que permite llevar un registro de lo que se borró.
Ejemplo:

```
¿Seguro que desea eliminar Matemáticas? (S/N): S
✅ Curso eliminado y agregado al historial.
```

8. ordenar_por_nota():

Utiliza el algoritmo de la burbuja (Bubble Sort) para ordenar los cursos por nota, del más alto al más bajo.

Así, el estudiante puede ver rápidamente en qué cursos obtuvo mejor o peor rendimiento.

Ejemplo:

```
1. Computación → 95
2. Matemáticas → 85
3. Inglés → 70
```

9. ordenar_por_nombre():

Usa el **algoritmo de inserción (Insertion Sort)** para **ordenar los cursos alfabéticamente** por su nombre.
Esto ayuda a encontrar fácilmente un curso dentro de la lista.

Ejemplo:

```
1. Computación
2. Inglés
3. Matemáticas
```

10. simular_cola_revision():

Esta función simula una **cola (estructura FIFO)** donde los cursos se agregan al final y se revisan en el mismo orden en que entraron.
Representa el proceso de **revisión de tareas o exámenes**, donde el primero en llegar es el primero en ser atendido.

11. mostrar_historial():

Simula una **pila (estructura LIFO)** que guarda los **últimos cursos eliminados o modificados**.
Permite revisar los cambios recientes.
Ejemplo:

```
Últimos cursos eliminados:
1. Historia
2. Química
```

Explicación Del Uso De Listas, Pilas y Colas

En el programa **Gestor de Notas Académicas**, se utilizan diferentes estructuras de datos de Python para guardar, organizar y manipular la información de los cursos. Las más importantes son las **listas**, las **pilas** y las **colas**.
A continuación se explica cómo y por qué se usan cada una:

1. Uso de Listas

Las **listas** son la base principal del sistema.
Se usan para **guardar todos los cursos registrados** junto con sus notas.
En Python, una lista es un conjunto de elementos ordenados que pueden cambiar o crecer en cualquier momento.

En este proyecto, la lista principal se llama cursos:

```
cursos = []
```

Cada vez que el usuario registra un nuevo curso, se agrega un elemento a esa lista.
Cada elemento es un **diccionario**, que guarda dos datos importantes: el nombre y la nota.

Por ejemplo:

```
{"nombre": "Matemáticas", "nota": 85}
```

Si el estudiante registra varios cursos, la lista quedaría así:

```
cursos = [
    {"nombre": "Matemáticas", "nota": 85},
    {"nombre": "Computación", "nota": 90},
    {"nombre": "Historia", "nota": 70}
]
```

De esta forma, el sistema puede **recorrer la lista** para mostrar todos los cursos, calcular el promedio o buscar alguno en específico.

Las listas también se usan para:

- Contar cuántos cursos están aprobados o reprobados.
- Ordenar los cursos por nombre o por nota.
- Guardar los datos actualizados o modificados.

En resumen, las **listas** son el “corazón” del sistema porque ahí se almacena toda la información principa

2. Uso de Pilas (Historial de Cambios)

Una pila es una estructura que funciona con la lógica LIFO (Last In, First Out), que significa “el último en entrar es el primero en salir”.
En este proyecto, la pila se usa para guardar el historial de los cambios realizados, como cuando un curso se elimina o cuando se actualiza su nota.

Por ejemplo, si se elimina el curso “Historia”, ese cambio se guarda así:

```
historial.append("Se eliminó: Historia - Nota: 70")
```

Si después se actualiza “Matemáticas”, se guarda:

```
historial.append("Se actualizó: Matemáticas - De 85 a 90")
```

Entonces la pila (historial) quedaría así:

```
historial = [
    "Se eliminó: Historia - Nota: 70",
    "Se actualizó: Matemáticas - De 85 a 90"
]
```

Cuando el usuario elige **“Mostrar historial”**, el sistema mostrará los cambios **del más reciente al más antiguo**, respetando el principio de una pila.

```
Historial de cambios:
1. Se actualizó: Matemáticas - De 85 a 90
2. Se eliminó: Historia - Nota: 70
```

Esto es útil para llevar control y evitar perder información importante sobre lo que se ha modificado.

3. Uso de Colas (Simulación de revisión de notas)

Una **cola** funciona al revés de una pila: sigue la regla **FIFO (First In, First Out)**, o sea, “el primero en entrar es el primero en salir”.
En este proyecto se usa para **simular el proceso de revisión de cursos**.

Cuando el estudiante quiere enviar varios cursos a revisión, los va ingresando uno por uno, y cada curso se agrega **al final de la cola**.

Ejemplo:

```
cola_revision = []
cola_revision.append("Matemáticas")
cola_revision.append("Historia")
cola_revision.append("Computación")
```

Luego, el sistema procesa la cola **en el mismo orden**:

4. Uso de Búsquedas y Ordenamientos

Además de listas, pilas y colas, el sistema también utiliza algoritmos de búsqueda y ordenamiento básicos que son parte de la lógica del programa.

- **Búsqueda lineal:** Recorre toda la lista para encontrar un curso.
- **Búsqueda binaria:** Se usa cuando los cursos están ordenados alfabéticamente y permite encontrar un curso más rápido.
- **Ordenamiento burbuja:** Sirve para ordenar por nota (de mayor a menor).
- **Ordenamiento por inserción:** Ordena los cursos por nombre (alfabéticamente).

Justificación De Los Algoritmos de Ordenamiento Implementados

En el proyecto Gestor de Notas Académicas, se implementaron dos algoritmos de ordenamiento clásicos:

- Ordenamiento Burbuja (Bubble Sort)
- Ordenamiento por Inserción (Insertion Sort)

Estos métodos fueron elegidos porque son fáciles de entender, no requieren librerías externas y se pueden programar paso a paso utilizando listas, que es justo lo que aprendimos en clase.

1. Ordenamiento Burbuja (para las notas)

Este método se llama así porque las notas “van subiendo” o “bajando” poco a poco, igual que burbujas en el agua.
Sirve para ordenar los cursos según la nota, desde la más alta hasta la más baja.

El programa va comparando dos cursos a la vez, y si uno tiene una nota más baja, los cambia de lugar.
Así, las notas se van acomodando hasta que toda la lista queda ordenada.

Ejemplo:

Si tengo estas notas:

```
[70, 90, 80]
```

Primero compara 70 con 90 → los deja igual.
Luego compara 90 con 80 → los cambia.
Y al final queda así:

```
[90, 80, 70]
```

Este proceso se repite varias veces hasta que todas las notas están bien ordenadas.
Aunque no es el método más rápido del mundo, **es fácil de entender** y suficiente para este proyecto.

2. Ordenamiento por Inserción (para los nombres)

Este otro método se usa para ordenar los cursos alfabéticamente, por ejemplo:

```
["Historia", "Matemáticas", "Programación"]
```

Funciona parecido a cuando **ordenamos cartas o papeles**:
vamos tomando uno por uno y lo colocamos en el lugar que le corresponde.

Por ejemplo, si tengo:

```
["Programación", "Matemáticas", "Historia"]
```

Primero reviso “Matemáticas” y veo que va antes que “Programación”.
Luego reviso “Historia” y veo que va antes que las dos.
Al final queda así:

```
["Historia", "Matemáticas", "Programación"]
```

Documentación De Las Funciones Del Sistema

A continuación, se explica para qué sirve cada función del programa **Gestor de Notas Académicas**, cómo trabaja y un ejemplo sencillo de su uso.
Todas las funciones fueron creadas con el objetivo de que el código sea más ordenado, fácil de leer y de modificar en el futuro.

1. mostrar_menu():

Función:
Muestra todas las opciones disponibles del sistema al usuario.
Es como el “mapa” del programa, donde se puede ver todo lo que se puede hacer.

Ejemplo:
Cuando se ejecuta el programa, aparece algo así:

```
##### GESTOR DE NOTAS ACADÉMICAS #####
1. Registrar nuevo curso
2. Calcular promedio general
3. Contar cursos aprobados y reprobados
...
12. Salir
```

Por qué se usa:
Sirve para guiar al usuario paso a paso y repetir el menú cada vez que elija una opción.

2. leer_opcion():

Función:
Permite que el usuario escriba un número de opción.
Verifica que no esté vacío ni que sea texto, para evitar errores.

Ejemplo:

```
Seleccione una opción: 1
```

Si el usuario escribe algo incorrecto (como “abc”), el sistema le muestra un mensaje de error.

Por qué se usa:
Para controlar que la entrada sea correcta antes de continuar.

3. registrar_curso():

Función:
Permite guardar un nuevo curso con su nota.
Primero pide el nombre del curso y después la nota (entre 0 y 100).

Ejemplo:

```
Ingrese el nombre del curso: Matemáticas
Ingrese la nota del curso (0-100): 85
Curso registrado con éxito.
```

Qué hace internamente:
Guarda la información así:

y la añade a la lista principal de cursos.

Por qué se usa:
Es la base del sistema, ya que sin esta función no habría datos que mostrar ni analizar.

4. mostrar_cursos():

Función:
Muestra en pantalla todos los cursos que han sido registrados junto con sus notas.

Ejemplo:

```
Cursos registrados:
1. Matemáticas - Nota: 85
2. Programación - Nota: 90
```

Por qué se usa:
Ayuda al usuario a tener una vista general de todas las notas guardadas.

5. calcular_promedio():

Función:
Suma todas las notas de los cursos registrados y las divide entre la cantidad de cursos.
Así obtiene el promedio general del estudiante.

Ejemplo:

```
Promedio general: 87.5
```

Por qué se usa:
Para saber cómo van las calificaciones en general.

6. contar_aprobados_reprobados():

Función:
Cuenta cuántos cursos están aprobados (nota ≥ 60) y cuántos están reprobados.

Ejemplo:

```
Aprobados: 3
Reprobados: 1
```

Por qué se usa:
Permite analizar de forma rápida el desempeño del estudiante.

7. buscar_curso_lineal():

Función:
Busca un curso escribiendo su nombre.
Compara cada elemento de la lista uno por uno hasta encontrar coincidencia.

Ejemplo:

```
Ingrese el nombre del curso a buscar: matemáticas
Curso encontrado: Matemáticas - Nota: 85
```

Por qué se usa:
Es un método fácil para buscar algo cuando no hay orden en la lista.

8. actualizar_curso():

Función:
Permite cambiar la nota de un curso ya existente.

Ejemplo:

```
Ingrese el nombre del curso: Matemáticas
Ingrese la nueva nota: 95
Nota actualizada con éxito.
```

Por qué se usa:
Porque a veces una nota puede cambiar (por revisión o corrección), y el programa debe reflejarlo.

9. eliminar_curso():

Función:
Elimina un curso de la lista, pero antes pide confirmación al usuario.

Ejemplo:

```
Ingrese el curso a eliminar: Historia
¿Está seguro que desea eliminarlo? (s/n): s
Curso eliminado correctamente.
```

```
Cursos ordenados por nota:
1. Programación - 95
2. Matemáticas - 85
3. Historia - 72
```

Por qué se usa:
Para mantener la lista actualizada solo con los cursos válidos.

10. ordenar_por_nota():

(Usa el método **Burbuja**)

Función:
Ordena los cursos de mayor a menor nota.

Ejemplo:

```
Cursos ordenados por nombre:
1. Historia - 72
2. Matemáticas - 85
3. Programación - 95
```

Por qué se usa:
Para identificar fácilmente cuáles son las mejores y peores notas.

11. ordenar_por_nombre():

(Usa el método **Inserción**)

Función:
Ordena los cursos alfabéticamente.

Ejemplo:

Por qué se usa:
Facilita encontrar cursos por nombre y mejora la organización.

12. buscar_curso_binaria():

Función:
Busca un curso por nombre más rápido, pero solo funciona si ya está ordenado por nombre.

Ejemplo:

```
Ingrese el curso a buscar: Programación
Curso encontrado: Programación - Nota: 95
```

Por qué se usa:
Para mostrar una forma más eficiente de búsqueda en listas ordenadas.

13. simular_cola_revision():

Función:
Crea una “cola” de cursos que el estudiante quiere enviar a revisión.
Los cursos se procesan en el mismo orden en que se agregan.

Ejemplo:

```
Ingrese curso para revisión (escriba 'fin' para terminar):
> Matemáticas
> Historia
> fin

Procesando solicitudes:
Revisando: Matemáticas
Revisando: Historia
```

Por qué se usa:
Muestra cómo funciona una cola: el primero que entra es el primero que se atiende.

14. mostrar_historial_cambios():

(Usa una **pila**)

Función:
Guarda los cambios hechos (como actualizaciones o eliminaciones) y los muestra en orden inverso.

Ejemplo:

```
Historial de cambios recientes:
1. Se eliminó: Historia - Nota: 72
2. Se actualizó: Matemáticas - 85 → 95
```

Por qué se usa:
Permite llevar un registro de las acciones realizadas y practicar el uso de pilas.

Diagrama General Del sistema/ Pseudocódigo

El siguiente pseudocódigo representa cómo funciona el programa **Gestor de Notas Académicas** paso a paso.
Está pensado para mostrar de forma sencilla la lógica general del sistema antes de convertirlo en código Python.

Pseudocódigo del sistema

```
INICIO
    CREAR lista cursos ← []
    CREAR lista historial ← []
    CREAR lista cola_revision ← []
    MIENTRAS VERDADERO HACER
        IMPRIMIR "===== GESTOR DE NOTAS ACADÉMICAS ====="
        IMPRIMIR "1. Registrar nuevo curso"
        IMPRIMIR "2. Mostrar todos los cursos y notas"
        IMPRIMIR "3. Calcular promedio general"
        IMPRIMIR "4. Contar cursos aprobados y reprobados"
        IMPRIMIR "5. Buscar curso por nombre (búsqueda lineal)"
        IMPRIMIR "6. Actualizar nota de un curso"
        IMPRIMIR "7. Eliminar un curso"
        IMPRIMIR "8. Ordenar cursos por nota (burbuja)"
        IMPRIMIR "9. Ordenar cursos por nombre (inserción)"
```

```
IMPRIMIR "10. Buscar curso por nombre (búsqueda binaria)"
IMPRIMIR "11. Simular cola de solicitudes de revisión"
IMPRIMIR "12. Mostrar historial de cambios (pila)"
IMPRIMIR "13. Salir"

LEER opcion

SI opcion = 1 ENTONCES
   LLAMAR registrar_curso()
SINO SI opcion = 2 ENTONCES
   LLAMAR mostrar_cursos()
SINO SI opcion = 3 ENTONCES
   LLAMAR calcular_promedio()
SINO SI opcion = 4 ENTONCES
   LLAMAR contar_aprobados_reprobados()
SINO SI opcion = 5 ENTONCES
   LLAMAR buscar_curso_lineal()
SINO SI opcion = 6 ENTONCES
   LLAMAR actualizar_curso()
SINO SI opcion = 7 ENTONCES
   LLAMAR eliminar_curso()
SINO SI opcion = 8 ENTONCES
   LLAMAR ordenar_por_nota()
SINO SI opcion = 9 ENTONCES
   LLAMAR ordenar_por_nombre()
SINO SI opcion = 10 ENTONCES
   LLAMAR buscar_curso_binaria()
SINO SI opcion = 11 ENTONCES
```

```
        LLAMAR simular_cola_revision()
    SINO SI opcion = 12 ENTONCES
        LLAMAR mostrar_historial_cambios()
    SINO SI opcion = 13 ENTONCES
        IMPRIMIR "Gracias por usar el Gestor de Notas Académicas"
        SALIR DEL PROGRAMA
    SINO
        IMPRIMIR "Opción no válida, intente de nuevo."
    FIN_SI
FIN_MIENTRAS
```

Conclusión

El **Gestor de Notas Académicas** es un sistema funcional que aplica los fundamentos de la programación estructurada y las estructuras de datos clave: utiliza **Listas** para almacenar cursos, **Pilas** para el historial de cambios y **Colas** para gestionar solicitudes de revisión (FIFO). La lógica se controla mediante condicionales y bucles, e integra algoritmos de **búsqueda y ordenamiento** para maximizar la eficiencia. El proyecto demuestra que con estos conceptos básicos se puede crear una aplicación útil, funcional y de fácil manejo para el estudiante.