GESTOR DE NOTAS ACADÉMICAS

ANA IZABEL ESTRADA RUIZ

CARNET: NO. 7590-25-7280

OCTUBRE 2025

MANUAL TÉCNICO

1) GESTOR DE NOTAS ACADÉMICAS

Descripción:

El Gestor de Notas Académicas es un sistema interactivo desarrollado en el programa Python, diseñado para que los estudiantes lleven un control completo y personalizado de sus calificaciones obtenidas en cada uno de sus cursos. Funciona exclusivamente en una consola, sin necesidad de usar librerías externas ni estructuras complejas, lo que lo hace compatible con cualquier entorno de Python.

El sistema facilita la administración, actualización, análisis y seguimiento del rendimiento académico mediante un menú interactivo. Entre sus principales funcionalidades se incluyen registro de cursos y notas, cálculo de promedio, conteo de cursos aprobados/reprobados, búsqueda de cursos, actualización y eliminación de registros, ordenamiento de cursos y simulación de solicitudes de revisión.

El programa está dirigido a estudiantes de nivel medio y universitario que desean un seguimiento personalizado y eficiente de su desempeño académico.

El sistema simula operaciones reales de un gestor académico mediante el uso de:

- Listas → para almacenar cursos y notas.
- Pilas (LIFO) → para el historial de cambios.
- Colas (FIFO) → para la simulación de solicitudes de revisión.

El código está dividido en funciones específicas por tarea, lo que facilita su mantenimiento, comprensión y prueba individual.

2) ESTRUCTURA GENERAL DEL CÓDIGO

El código está organizado por funciones independientes, cada una con su propósito claro, acompañado de una pequeña documentación PRE y POST, y comentarios explicativos sobre algunas funciones o módulos. EL menú contara con las siguientes funciones:

1. **Registro_Curso():** Esta función registrara un nuevo curso y su nota.
2. **Mostrar_cursos_notas():** Muestra todos los cursos y notas, registradas en las listas.
3. **Calcular_promedio():** Calculara el promedio general de todas las notas registradas.
4. **Aprobados_Reprobados():** Realiza el recuento de cuántos cursos fueron aprobados o reprobados, y los muestra al final.
5. **Buscar_curso():** Permite buscar un curso por nombre o una parte del nombre, dentro de la lista de cursos que ya está guardada en el programa.
6. **Actualizar_Nota_De_Cursos():** Permite actualizar la nota de un curso en específico que ya existe en el sistema. Por ejemplo, si el estudiante mejora su nota puedes modificarla fácilmente.
7. **Eliminar_Curso():** Elimina un curso y su nota.
8. **ordenar_cursos_por_nota(cursos, notas):** Ordena los cursos según sus notas, de forma descendente (de la nota más alta a la más baja). A este ordenamiento se le conoce como Burbuja.
9. **ordenar_cursos_por_nombre(cursos, notas):** Ordena los cursos por nombre (en orden alfabético, usando el método de inserción. Además, asegura que las notas sigan siempre anidadas al curso correcto después del ordenamiento.
10. **buscar_curso_binario(cursos, notas):** Buscar un curso específico por su nombre dentro de la lista de cursos, usando el algoritmo de búsqueda binaria.
11. **simular_cola_revision():** Simula una cola de solicitudes donde el estudiante pide una revisión de sus cursos. Se revisa en el orden en que fue recibida, es decir, el primero en entrar es el primero en salir (FIFO).
12. **Historial_de_cambios():** Muestra el historial de modificaciones hechas en el sistema, como, eliminar un curso. Utiliza una pila (LIFO: Last In, First Out), donde el último cambio registrado es el primero en mostrarse.

3) PILAS Y COLAS

Pila: Una pila (stack) es una estructura de datos LIFO (Last In, First Out), es decir, el último elemento que entra es el primero que sale. Esto se asemeja a una pila de platos: el último que colocas arriba es el primero que retiras.

En el gestor la pila se utiliza para registrar las acciones más recientes del usuario o estudiante que usa el sistema, como actualizaciones, eliminaciones o registros de cursos. Del mismo modo el sistema podrá mostrar los últimos cambios realizados en orden inverso al que fueron ejecutados, previamente en el gestor de notas; podemos ver su estructura en la función número 12, donde se simula el uso de la pila.

Los cambios se almacenan en esta lista:

```
historial_cambios = []
```

Estructura de uso en el Gestor de Notas Académicas:

```
def Historial_de_cambios():
    if not historial_cambios:
        print("No hay cambios registrados.\n")
        return

    print("Historial de cambios recientes:")
    for i, cambio in enumerate(reversed(historial_cambios), 1):
        print(f"{i}. {cambio}")
    input("Presione enter para continuar")
```

- Permite rastrear las acciones más recientes realizadas durante su funcionamiento.

Cola: una cola (queue) es una estructura de datos **FIFO** (First In, First Out), el primer elemento que entra es el primero en salir. Es como una fila en un banco: quien llega primero, se atiende primero.

En el Gestor, la cola se utiliza para simular solicitudes de revisión de varias notas. Los estudiantes agregan sus nombres o cursos a la cola cuando desean pedir una revisión. Luego el sistema procesa cada solicitud en el mismo orden en que fueron registradas. La usamos en la función número 11.

Cada curso revisado se almacenará en:

```
cola = []
```

Estructura de la cola en el Gestor de Notas Académicas:

```
def simular_cola_revision():
    if not cursos:
        print("No hay cursos registrados en el sistema\n")
        return
    cola = []  # lista vacía que funcionará como cola

    print("Ingrese curso para revisión (escriba 'fin' para terminar):")
    while True:
        curso = input("> ").strip()
        if curso.lower() == "fin":
            break
        cola.append(curso)  # se agrega al final de la cola

    print("\nProcesando solicitudes:")
    while cola:  # mientras haya elementos en la cola
        curso = cola.pop(0)  # sacar el primero en entrar
        print(f"Revisando: {curso}")
    input("Presione enter para continuar")
```

- Logro mejorar la comprensión del manejo dinámico de listas como colas.

4) ALGORITOS DE ORDENAMIENTO IMPLEMENTADOS EN EL SISTEMA:

En el Gestor de Notas Académicas, se utilizaron dos algoritmos de ordenamiento distintos, cada uno fue seleccionado según su utilidad eficiencia y aplicabilidad práctica dentro del sistema. Estos algoritmos permiten organizar y visualizar los datos almacenados en las listas de cursos y calificaciones de manera clara, estructurada y comprensible para el usuario que use el sistema.

El uso de algoritmos de ordenamiento fue esencial dentro del programa, ya que nos garantiza que la información académica del usuario se presente de forma ordenada, facilitando el análisis del rendimiento del estudiante del mismo modo mejorando la interacción con el sistema.

Los algoritmos que se implementaron son:

Ordenamiento Burbuja (Bubble Sort):

Se basa en comparar pares de elementos adyacentes dentro de una lista y realizar intercambios si los elementos cuando están en el orden incorrecto; en cada iteración el algoritmo desplaza el elemento más grande o más pequeño hasta su posición final, repitiendo el proceso hasta que todos los elementos estén ordenados.

Dentro del Gestor de Notas Académicas, este método de ordenamiento se utiliza para ordenar los cursos según las calificaciones obtenidas, de mayor a menor nota. De esta forma, el estudiante puede identificar rápidamente cuáles son sus cursos con mejor rendimiento académico y cuáles necesitan mejorar.

Este es un algoritmo simple y muy fácil de implementar, lo que lo hace apropiado para entornos educativos o sistemas académicos de práctica. Nos presenta una complejidad temporal adecuada para el volumen de datos limitado que maneja el sistema, ofreciendo una respuesta rápida y ordenada ante cada operación de ordenamiento. Permite visualizar los resultados de forma inmediata y ordenada según el criterio del usuario, por su claridad lógica, bajo nivel de complejidad y compatibilidad con listas pequeñas, se adapta perfectamente a las necesidades de un sistema de gestión de calificaciones personales.

Estructura del Algoritmo Burbuja:

```
def ordenar_cursos_por_nota(cursos, notas):
#PRE: Debe haber cursos y notas registrados, en las listas globales, notas y Cursos
#POST:Las listas se reordenan y se muestran de mayor a menor nota, en la pantalla principal.
    if not cursos:
        print("No hay cursos registrados para ordenar.\n")
        return
    n = len(notas)
    for i in range(n):
        for j in range(0, n - i - 1):
            if notas[j] < notas[j + 1]:
                #Intercambiar ambos notas
                notas[j], notas[j + 1] = notas[j + 1], notas[j]
                #Intercambiar cursos
                cursos[j], cursos[j + 1] = cursos[j + 1], cursos[j]

    print("Cursos ordenados por nota (descendente):")
    for i in range(n):
        print(f"{i+1}. {cursos[i]} - Nota: {notas[i]}")
    input("Presione enter para continuar")
```

Ordenamiento por Inserción (Insertion Sort):

El algoritmo de inserción divide la lista en dos secciones: una parte ordenada y otra parte desordenada. Durante este proceso, el algoritmo toma cada nuevo elemento de la parte desordenada y lo inserta en la posición correcta dentro de la parte ordenada, desplazando los demás elementos según las veces que sea necesario.

En el Gestor de Notas Académicas: el algoritmo se utiliza para ordenar los cursos de forma alfabéticamente por su nombre, junto con sus notas.
Esto permite que la información se presente de una manera organizada y sea más fácil de buscar manualmente, facilitando la búsqueda de cursos dentro del gestor.

El algoritmo de ordenamiento por inserción fue seleccionado por ser eficiente y estable en el manejo de listas pequeñas o medianas, acordes con el volumen de datos que manejara el

sistema. Su complejidad **O(n²)** resulta adecuada en este contexto, mostrando un desempeño bastante excelente cuando los datos están parcialmente ordenados, reduciendo el número de comparaciones necesarias. Además, es un algoritmo fácil de comprender y mantener, lo hace ideal para entornos educativos o de práctica académica; fue elegido por su simplicidad, precisión y capacidad de mantener la coherencia entre los cursos y sus notas, garantizando una organización alfabética clara y eficiente de la información académica.

Estructura del Algoritmo de Insercion:

```
def ordenar_cursos_por_nombre(cursos, notas):
#PRE:Debe haber cursos y notas registrados.
#POST: Los cursos quedan ordenados alfabéticamente junto con sus notas.
    if not cursos:
        print("No hay cursos registrados para buscar.\n")
        return
    n = len(cursos)
    for i in range(1, n):
#Se guardan temporalmente el curso y la nota que se van a insertar en la posición correcta.
        curso_actual = cursos[i]
        nota_actual = notas[i]
        j = i - 1
        # Mover los cursos mayores que curso_actual una posición adelante
        while j >= 0 and cursos[j].lower() > curso_actual.lower():
            #Se mueve el curso y la nota en la posición j una posición a la derecha
            cursos[j + 1] = cursos[j]
            notas[j + 1] = notas[j]
            j -= 1
        cursos[j + 1] = curso_actual
        notas[j + 1] = nota_actual
    print("Cursos ordenados por nombre:")
    for i in range(n):
        print(f"{i+1}. {cursos[i]} - Nota: {notas[i]}")
    input("Presione enter para continuar")
```

Ambos algoritmos fueron seleccionados por su facilidad de comprensión, su simplicidad lógica, y por ser totalmente compatibles con los requerimientos del sistema.

5) DOCUMENTACIÓN DE FUNCIONES

Mostrar_menu()

Muestra el menú principal del sistema en consola y permite al usuario seleccionar la acción que desea realizar.

Es llamada continuamente dentro del bucle principal del programa para controlar el flujo de ejecución.

Funcionamiento:

Despliega una lista numerada de opciones (agregar, mostrar, actualizar, ordenar, etc.). Solicita una entrada del usuario (input ()), que se válida para evitar errores. Retorna la opción seleccionada al flujo principal del programa.

Parámetros: No recibe parámetros.

Valor de retorno: Devuelve un número entero o cadena con la opción elegida por el usuario.

1) **Función: def Registro_Curso().**

   Esta función permite al usuario registrar una nueva nota junto con el nombre del curso. Debe ingresar un nombre de curso válido no puede estar vacío y una nota numérica que se encuentre entre 0 y 100. El curso y su nota quedan almacenados en las listas principales del sistema: cursos = [] y notas = [].

   Se registran los cambios de las listas globales en la pila de historial.

   Historial de cambios = []

   Parámetros: no recibe parámetros directos, solo modifica las listas notas y cursos.

   Retorno: Actualiza las listas globales.

2) **Función: def Mostrar_cursos_notas().**

   Muestra todas las notas registradas junto con los nombres de los cursos que este registrados en las listas principal, en consola en una lista numerada.

   Parámetros: No hay entradas, trabaja con los datos que ya están registrados.

   Retorno: No retorna datos; solo imprime información en la consola.

3) **Función: def Calcular_promedio().**

   Calcula el promedio general de las notas registradas.

   Suma los valores almacenados en la lista notas, luego divide el total con la cantidad de elementos de la lista.

Parámetros: No recibe parámetros, trabaja con la lista de notas.

Retorno: Devuelve el promedio calculado con 2 decimales o muestra un mensaje si no hay notas registradas.

4) **Función: def Aprobados_Reprobados().**

   Cuenta y Muestra los cursos aprobados y reprobados, que se encuentren en el sistema. Deben existir notas registradas. La función compara que los cursos aprobados estén (>=60) y los reprobados (<60).

   Parámetros: no recibe parámetros externos, trabaja con la lista Notas y lista cursos.

   Retorno: Se imprime la cantidad de cursos aprobados y reprobados.

   Cursos aprobados (>=60) y reprobados (<60)

5) **Función: def Buscar_curso().**

   Permite al usuario localizar cursos dentro del sistema a partir de su nombre o parte del mismo, mostrando las coincidencias encontradas junto con sus respectivas notas. Solicita al usuario que ingrese el nombre (o parte del nombre) del curso a buscar. Recorre la lista cursos si encuentra coincidencias parciales o completas, almacena el curso y su nota correspondiente en la lista encontrados.

   Parámetros: No recibe parámetros, trabaja con las listas, y la parte del curso que desea buscar

   Retorno: Muestra en consola los cursos encontrados o un mensaje si no se halla ninguno.

6) **Función: def Actualizar_Nota_De_Cursos().**

   Actualiza la nota de un curso ya existente.

   Muestra los cursos disponibles en el sistema, solicita el nombre del curso a modificar, busca el curso ignorando mayúsculas/minúsculas; si lo encuentra, actualiza la nota y guarda el cambio en el historial.

   Parámetros: No recibe parámetros externos, solo cambia un dato por otro.

   Retorno: Actualiza las listas globales y el historial.

7) **Función: def Eliminar_Curso().**

   Permite eliminar un curso y su nota del sistema.

Muestra los cursos actuales registrados, solicita al usurario cual es el curso a eliminar; verifica su existencia y lo elimina de las listas cursos y notas. Registra la acción en historial_cambio

Parámetro: No recibe parámetros, elimina valores de las listas.

Retorno: Modifica las listas globales para eliminar el rastro del curso eliminado.

8) **Función: def ordenar_cursos_por_nota(cursos, notas).**

Ordena los cursos según sus notas (mayor a menor) usando el método de burbuja.

Recorre ambas listas la de cursos y notas, compara pares de elementos adyacentes; Si están en orden incorrecto, los intercambia, repite el proceso hasta que la lista quede completamente ordenada.

Parámetros: Trabaja con las listas globales.

Retorno: Modifica el orden interno de las listas.

9) **Función: def ordenar_cursos_por_nombre(cursos, notas).**

Ordena los cursos alfabéticamente usando el método de inserción.

Divide la lista en dos una parte ordenada y otra desordenada. Toma cada elemento de la parte desordenada y lo inserta en su posición correcta dentro de la parte ordenada. Reordena simultáneamente la lista notas para mantener coherencia con los cursos.

Parámetro: No recibe valores, busca dentro de las listas.

Retorno: No devuelve datos, solo modifica el orden de las listas de forma alfabética.

10) **Función: def buscar_curso_binario(cursos, notas).**

Busca un curso usando búsqueda binaria (se requiere que la lista esta ordenada).

Solicita el nombre exacto del curso a buscar. Define los índices min y max como los extremos de la lista. Calcula la posición media (medio) y compara el curso en esa posición con el nombre buscado.Si el nombre buscado es mayor, ajusta el límite inferior (min), y si es menor, ajusta el límite superior (max). Repite el proceso hasta encontrar el curso o hasta que el rango de búsqueda se cierre.

Parámetro: lista cursos y lista notas.

Retorno: Muestra el resultado directamente en la consola, sino devuelve -1 si no existe coincidencia.

11) **Función: def simular_cola_revision().**

Simula una cola de solicitudes de revisión de notas, aplicando el principio FIFO (First In, First Out), donde el primer curso en ingresar es el primero en ser atendido.

Comienza comprobando si existen cursos registrados en el sistema. Se crea una lista vacía llamada cola, para almacenar los cursos que serán revisados. Se solicita al usuario que ingrese el nombre de los cursos que desea enviar a revisión, Cada curso se añade al final de la cola, y se extraen desde el inicio. El proceso continúa hasta que el usuario escriba la palabra clave "fin".

Parámetro: Utiliza la lista global cursos para verificar que existan cursos en el sistema.

Retorno: Muestra en pantalla el proceso de atención de las solicitudes de revisión.

12) F**unción: def Historial_de_cambios().**

Muestra el historial de cambios realizados (estructura tipo pila)

Accede a la lista historial_cambios. Si está vacía, muestra un mensaje de advertencia. Si contiene datos, los recorre en orden inverso (el último cambio va a ser el primero). Permite visualizar las modificaciones más recientes.

Parámetros: No recibe parámetros.

Retorno: Imprime el historial.

6) PSEUDOCÓDIGO PRINCIPAL

INICIO

FUNCIÓN mostrar_menu

IMPRIMIR "====== GESTOR DE NOTAS ACADÉMICAS ======"

IMPRIMIR "1. Registrar Curso"

IMPRIMIR "2. Mostrar curso"

IMPRIMIR "3. Calcular promedio"

IMPRIMIR "4. Contar Cursos Aprobados/Reprobados"

IMPRIMIR "5. Buscar Curso por nombre"

IMPRIMIR "6. Actualizar nota de un curso"

IMPRIMIR "7. Eliminar un curso"

IMPRIMIR "8. Ordenar cursos por nota (ordenamiento burbuja)"

IMPRIMIR "9. Ordenar cursos por nombre (ordenamiento 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"
FIN FUNCIÓN
FIN
```

DIAGRAMA GENERAL DEL SISTEMA