Manual Técnico - Gestor de Notas Académicas

¿Qué hace el programa?

El programa te deja:

- Agregar cursos con sus notas
- Ver todos los cursos que tienes
- Calcular tu promedio
- Ver cuántos cursos aprobaste o reprobaste
- Buscar cursos
- Actualizar notas
- Eliminar cursos
- Ordenar los cursos de diferentes formas
- Llevar un historial de los cambios que haces

Estructura del Proyecto

El proyecto está organizado en varios archivos `.py` (archivos de Python). Cada archivo tiene una función específica. Esto se llama "modularización" y ayuda a que el código esté más organizado.

¿Por qué separar el código en varios archivos?

En lugar de tener todo el código en un solo archivo gigante, lo dividimos en archivos más pequeños. Esto tiene varias ventajas:

1. **Es más fácil de entender:** Cada archivo hace una cosa específica, entonces cuando necesitas buscar algo, sabes dónde ir.
2. **Es más fácil de mantener:** Si algo falla en la búsqueda, solo revisas `buscar_curso.py`, no tienes que revisar 500 líneas de código.
3. **Se puede reutilizar:** Si en el futuro quieres usar la función de ordenamiento en otro proyecto, solo copias ese archivo.
4. **Trabajo en equipo:** Varias personas pueden trabajar en diferentes archivos sin estorbarse.
5. **Está más ordenado:** Es como tener tu cuarto organizado con cajones, en lugar de todo tirado en el piso.

Archivos principales:

```
python/
├── menu.py                    # El archivo principal que ejecuta todo
├── almacenamiento_datos.py    # Donde se guardan los cursos en memoria
├── registrar_curso.py         # Para agregar nuevos cursos
├── mostrar_cursos.py          # Para ver todos los cursos
├── buscar_curso.py            # Para buscar un curso (búsqueda lineal)
├── buscar_binario.py          # Para buscar un curso (búsqueda binaria)
├── actualizar_nota.py         # Para cambiar la nota de un curso
├── eliminar_curso.py          # Para borrar un curso
├── promedio_general.py        # Para calcular el promedio de todas las notas
├── contar_aprobados.py        # Para contar aprobados y reprobados
├── ordenar_por_nota.py        # Para ordenar cursos por nota
├── ordenar_por_nombre.py      # Para ordenar cursos alfabéticamente
├── cola_revision.py           # Simula una cola de revisiones
└── historial_cambios.py       # Guarda el historial de cambios
```

Resumen de qué hace cada archivo:

Archivo	¿Qué hace?	¿Cuándo se usa?
menu.py	Es el cerebro del programa. Muestra el menú y coordina todo	Es el archivo que ejecutas para iniciar el programa
almacenamiento_datos.py	Guarda la lista de cursos en memoria	Todos los demás archivos lo usan para acceder a los cursos
registrar_curso.py	Agrega un nuevo curso a la lista	Cuando quieres registrar un curso nuevo
mostrar_cursos.py	Muestra todos los cursos en pantalla	Cuando quieres ver qué cursos tienes
buscar_curso.py	Busca un curso de forma simple (uno por uno)	Cuando quieres encontrar un curso específico
buscar_binario.py	Busca un curso de forma rápida (dividiendo a la mitad)	Cuando tienes muchos cursos y quieres buscar rápido

Archivo	¿Qué hace?	¿Cuándo se usa?
actualizar_nota.py	Cambia la nota de un curso que ya existe	Cuando te equivocaste o cambió una nota
eliminar_curso.py	Borra un curso de la lista	Cuando ya no necesitas un curso
promedio_general.py	Calcula el promedio de todas tus notas	Cuando quieres saber cómo vas en general
contar_aprobados.py	Cuenta cuántos cursos aprobaste y reprobaste	Para ver tu rendimiento general
ordenar_por_nota.py	Ordena los cursos de menor a mayor nota	Para ver cuáles son tus peores/mejores notas
ordenar_por_nombre.py	Ordena los cursos alfabéticamente	Para ver los cursos en orden de A-Z
cola_revision.py	Simula una fila de solicitudes de revisión	Para practicar el concepto de colas (FIFO)
historial_cambios.py	Guarda un registro de todos los cambios	Para ver qué modificaciones has hecho

Explicación del Código

1. almacenamiento_datos.py

Este es el archivo más simple pero súper importante. Aquí se guarda la lista de cursos.

```
cursos = []
```

Es una lista vacía que va a guardar diccionarios. Cada diccionario tiene esta estructura:

```
{"nombre": "Matemáticas", "nota": 85.5}
```

Nota importante: Los datos solo se guardan mientras el programa está corriendo. Cuando lo cierras, se pierden. Esto es porque estamos usando memoria RAM, no un archivo o base de datos.

2. menu.py - El Archivo Principal

Este es el archivo más importante del proyecto. Es el que ejecutas para iniciar el programa completo.

¿Por qué es el archivo principal?

- Es el punto de entrada del programa (el que arranca todo)
- Contiene el bucle principal que mantiene el programa corriendo
- Coordina todas las demás funciones del proyecto
- Es como el "cerebro" que decide qué hacer según lo que el usuario elija

¿Qué hace?

- Muestra un menú con 13 opciones
- Lee lo que el usuario escribe
- Llama a la función correcta según la opción elegida
- Se repite hasta que el usuario decida salir

Conceptos importantes:

- **Bucle while True:** Hace que el menú se repita infinitamente hasta que elijas salir
- **try-except:** Atrapa errores cuando el usuario escribe algo que no es un número
- **Imports:** Al inicio importa todas las funciones de los otros archivos

```
from registrar_curso import ejecutar as opcion_1
```

Esto significa: "trae la función ejecutar del archivo registrar_curso y llámala opcion_1"

Para ejecutar el programa completo, solo necesitas correr este archivo:

```
python python/menu.py
```

3. registrar_curso.py

Aquí se agregan nuevos cursos a la lista.

Pasos que sigue:

1. Pide el nombre del curso
2. Pide la nota (tiene que ser entre 0 y 100)
3. Verifica que el curso no exista ya
4. Si todo está bien, lo agrega a la lista

5. Guarda el cambio en el historial

Validaciones importantes:

- Verifica que el nombre no esté vacío
- Verifica que la nota sea un número válido
- Verifica que la nota esté entre 0 y 100
- Verifica que el curso no esté duplicado (compara en minúsculas para evitar problemas)

4. mostrar_cursos.py

Muestra todos los cursos que tienes registrados.

Usa `enumerate(cursos, 1)` que es una función de Python que te da el índice y el elemento al mismo tiempo. El `1` significa que empieza a contar desde 1 en vez de 0.

5. buscar_curso.py

Busca un curso usando **búsqueda lineal**. Esto significa que revisa uno por uno todos los cursos hasta encontrar el que buscas.

Algoritmo:

```
Para cada curso en la lista:
    Si el nombre coincide:
        Lo encontré, termino
    Si no:
        Sigo buscando
```

Es simple pero puede ser lento si tienes muchos cursos.

6. buscar_binario.py

Busca un curso usando **búsqueda binaria**. Es más rápida que la búsqueda lineal, pero necesita que la lista esté ordenada.

¿Cómo funciona?

1. Primero ordena los cursos alfabéticamente
2. Mira el curso del medio
3. Si es el que buscas, listo
4. Si el que buscas es "menor" (alfabéticamente), busca en la mitad izquierda
5. Si es "mayor", busca en la mitad derecha
6. Repite hasta encontrarlo o hasta que no queden cursos

Es como buscar una palabra en el diccionario: no empiezas desde la A, sino que abres por la mitad.

7. actualizar_nota.py

Cambia la nota de un curso que ya existe.

Proceso:

1. Busca el curso por nombre
2. Pide la nueva nota
3. Valida que sea un número entre 0 y 100
4. Guarda la nota anterior y la nueva en el historial
5. Actualiza la nota

Usa `enumerate()` para obtener el índice del curso y poder modificarlo.

8. eliminar_curso.py

Borra un curso de la lista.

Usa `del cursos[i]` para eliminar el elemento en la posición `i`. Antes de eliminarlo, guarda la información en el historial.

9. promedio_general.py

Calcula el promedio de todas las notas.

Fórmula:

```
promedio = suma de todas las notas / cantidad de cursos
```

Usa un bucle `for` para sumar todas las notas y luego divide entre la cantidad de cursos.

10. contar_aprobados.py

Cuenta cuántos cursos están aprobados y cuántos reprobados.

Criterio: Un curso está aprobado si la nota es >= 60

Usa dos contadores que van sumando según la condición.

11. ordenar_por_nota.py

Ordena los cursos de menor a mayor nota usando el **algoritmo de burbuja**.

¿Cómo funciona el algoritmo de burbuja?

- Compara cada par de elementos consecutivos
- Si están en el orden incorrecto, los intercambia
- Repite el proceso varias veces hasta que todo esté ordenado

Es como cuando tienes fichas y vas comparando de dos en dos, moviendo la más grande hacia el final.

Ejemplo visual:

```
[85, 60, 95, 70]
[60, 85, 95, 70]  # Comparó 85 y 60, los intercambió
[60, 85, 70, 95]  # Comparó 95 y 70, los intercambió
[60, 70, 85, 95]  # Sigue comparando hasta ordenar todo
```

12. ordenar_por_nombre.py

Ordena los cursos alfabéticamente usando el **algoritmo de inserción**.

¿Cómo funciona el algoritmo de inserción?

- Toma un elemento
- Lo compara con los anteriores
- Lo inserta en la posición correcta

Es como cuando ordenas cartas en tu mano: tomas una carta nueva y la pones en su lugar entre las que ya tienes ordenadas.

13. cola_revision.py

Simula una **cola** (estructura de datos tipo FIFO: First In, First Out).

¿Qué es una cola? Es como la fila del banco: el primero que llega es el primero que se atiende.

Operaciones:

- **Agregar:** Pone una solicitud al final de la cola
- **Atender:** Atiende (elimina) la primera solicitud de la cola
- **Ver cola:** Muestra todas las solicitudes pendientes

Usa `cola_revisiones.pop(0)` para sacar el primer elemento (el más antiguo).

14. historial_cambios.py

Implementa una **pila** (estructura de datos tipo LIFO: Last In, First Out).

¿Qué es una pila? Es como una pila de platos: el último que pones es el primero que sacas.

Guarda todos los cambios que haces (registrar, actualizar, eliminar) y los muestra en orden inverso (del más reciente al más antiguo).

Funciones:

- `agregar_cambio()` : Agrega un cambio al historial
- `ejecutar()` : Muestra el historial de más reciente a más antiguo

Estructuras de Datos Usadas

Listas

Las listas son la estructura principal. En Python se crean con corchetes `[]`.

```
cursos = []
cursos.append(elemento)  # Agregar al final
del cursos[i]            # Eliminar en posición i
```

Diccionarios

Cada curso es un diccionario con dos claves: "nombre" y "nota".

```
curso = {"nombre": "Matemáticas", "nota": 85.5}
print(curso["nombre"])  # Acceder al valor
```

Cola (Queue)

Implementada con una lista, pero usando operaciones específicas:

- `append()` para agregar al final
- `pop(0)` para sacar del inicio

Pila (Stack)

Implementada con una lista:

- `append()` para agregar al final
- Se recorre de atrás hacia adelante para mostrar

Algoritmos Implementados

1. Búsqueda Lineal

¿Qué hace? Busca un curso revisando uno por uno desde el inicio hasta encontrarlo.

¿Cuándo es útil?

- Cuando tienes pocos cursos (menos de 20-30)
- Cuando la lista no está ordenada
- Es el método más simple de entender

¿Qué tan rápido es? Si tienes 10 cursos, en el peor caso revisa los 10. Si tienes 100, revisa los 100. Mientras más cursos tengas, más tiempo tarda.

2. Búsqueda Binaria

¿Qué hace? Busca un curso dividiéndolo a la mitad cada vez (como buscar en el diccionario).

¿Cuándo es útil?

- Cuando tienes muchos cursos (más de 30)
- La lista DEBE estar ordenada alfabéticamente
- Es mucho más rápida que la búsqueda lineal

¿Qué tan rápido es? Si tienes 100 cursos, solo necesita revisar como 7 cursos aproximadamente. Si tienes 1000, solo revisa como 10. Es súper eficiente con muchos datos.

Ejemplo: Si tienes 1000 cursos:

- Búsqueda lineal: revisa hasta 1000 cursos
- Búsqueda binaria: revisa solo unos 10 cursos

3. Ordenamiento Burbuja

¿Qué hace? Ordena comparando pares de elementos y moviéndolos si están en el orden incorrecto.

¿Cuándo es útil?

- Cuando tienes pocos cursos
- Es fácil de entender y programar
- Funciona bien para aprender cómo funcionan los ordenamientos

¿Qué tan rápido es? Con 10 cursos es rápido. Con 100 cursos empieza a ser lento. Con 1000 cursos es muy lento.

4. Ordenamiento por Inserción

¿Qué hace? Ordena tomando cada elemento y poniéndolo en su posición correcta (como ordenar cartas).

¿Cuándo es útil?

- Cuando tienes pocos cursos
- Cuando la lista ya está casi ordenada
- Es más eficiente que burbuja en algunos casos

¿Qué tan rápido es? Similar al burbuja: rápido con pocos datos, lento con muchos datos.

Validaciones y Manejo de Errores

El programa tiene varias validaciones para evitar errores:

1. **Validación de entrada vacía:** Verifica que el usuario escriba algo
2. **Try-except:** Atrapa errores cuando se espera un número

3. **Validación de rango:** Verifica que las notas estén entre 0 y 100
4. **Validación de duplicados:** No deja agregar cursos con el mismo nombre
5. **Validación de lista vacía:** Verifica que haya cursos antes de hacer operaciones

¿Por qué usamos `if not` en lugar de `if` con `else`?

En muchas partes del código verás esto:

```
if not cursos:
    print("No hay cursos registrados")
    return
# Resto del código continúa aquí...
```

En lugar de esto:

```
if cursos:
    # Todo el código aquí adentro
else:
    print("No hay cursos registrados")
```

¿Por qué lo hacemos así?

Esto se llama **"validación temprana"** o **"return early"**. La idea es:

1. **Primero verificamos si algo está mal** (lista vacía, dato inválido, etc.)
2. **Si está mal, salimos inmediatamente** con `return`
3. **Si está bien, continuamos con el código normal**

Ventajas de este método:

- **Código más limpio:** No tienes que indentar (meter espacios) todo el código principal
- **Más fácil de leer:** Las validaciones están al inicio, separadas de la lógica principal
- **Menos errores:** Es más difícil olvidarte de validar algo
- **Menos anidamiento:** Evitas tener muchos `if` dentro de `if` dentro de `if`

Ejemplo comparativo:

❌ **Forma complicada (con else):**

```
def ejecutar():
    if cursos:
        nombre = input("Nombre: ")
        if nombre:
            nota = input("Nota: ")
            if nota.isdigit():
                # Código principal aquí (muy adentro)
            else:
                print("Nota inválida")
        else:
            print("Nombre inválido")
    else:
        print("No hay cursos")
```

✅ **Forma simple (con if not y return):**

```
def ejecutar():
    if not cursos:
        print("No hay cursos")
        return

    nombre = input("Nombre: ")
    if not nombre:
        print("Nombre inválido")
        return

    nota = input("Nota: ")
    if not nota.isdigit():
        print("Nota inválida")
        return

    # Código principal aquí (sin tanto indentado)
```

Como puedes ver, la segunda forma es mucho más fácil de leer y entender.

Conceptos de Programación Aplicados

Variables

```
nombre = "Matemáticas"  # String (texto)
nota = 85.5             # Float (número decimal)
aprobados = 0           # Integer (número entero)
```

Condicionales

```
if nota >= 60:
    print("Aprobado")
else:
    print("Reprobado")
```

Bucles

```
# Bucle for - para recorrer listas
for curso in cursos:
    print(curso)

# Bucle while - se repite mientras una condición sea verdadera
while True:
    # código
```

Funciones

```
def ejecutar():
    # código de la función
```

Importaciones

```
from archivo import funcion
```

Cómo Ejecutar el Programa

1. Asegúrate de tener Python instalado (versión 3.x)
2. Abre una terminal o cmd
3. Navega a la carpeta del proyecto
4. Ejecuta: `python python/menu.py`
5. Sigue las instrucciones del menú