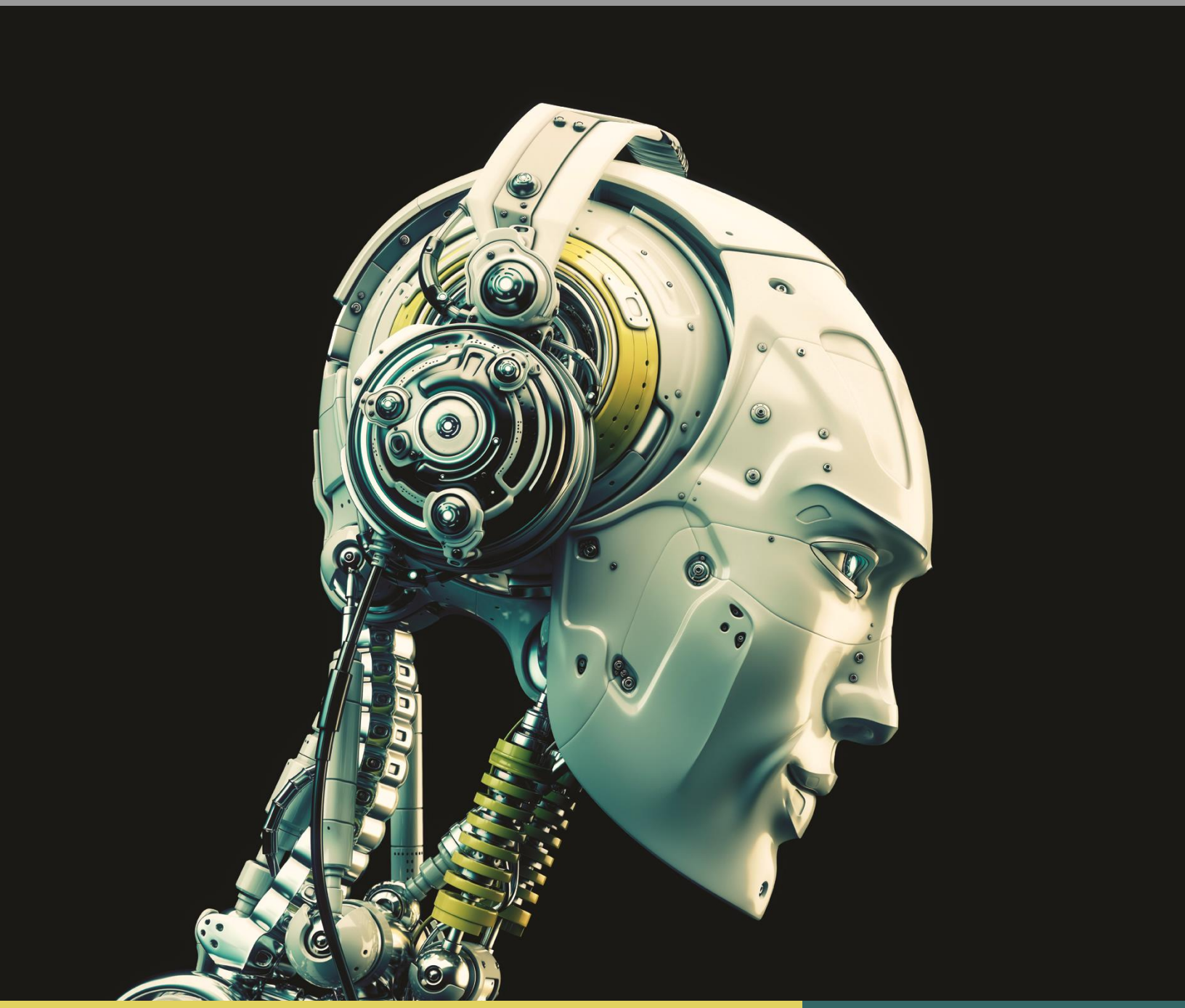
DON’T COMPROMISE



GESTOR DE NOTAS

BRANDON
ALEXANDER CHILE
POCÓN

Contenido

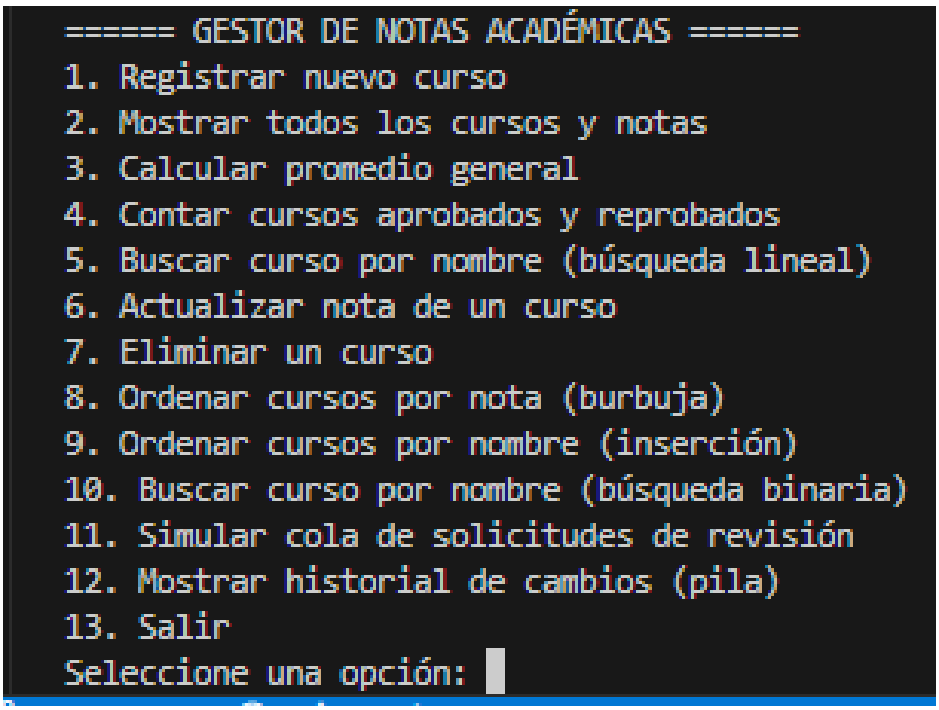
1. Descripción Técnica del Programa 3
1.1. Introducción 3
2. **Estructura General del Código** 4
2.3. Bloque de Procedimientos 4
2.1. Descripción General 4
2.2. Bloque de Funciones 4
2.4. Bloque del Programa Principal 5
3. Uso de Estructuras de Datos 5
3.1. Descripción General 5
3.2. Listas 5
3.3. Pilas 6
3.4. Colas 6
4. Justificación de los algoritmos de ordenamiento 7
4.1. Idea General 7
4.2. Ordenamiento Burbuja 7
4.3. Ordenamiento por Incerción 8
5. Documentación de Funciones y Módulos 8
5.1. Funciones de Cálculo y Procesamiento de Datos 8
5.2. Funciones de Búsqueda 9
5.3. Funciones de Ordenamiento 10
5.4. Procedimientos de Gestión de Cursos 10
5.5. Funciones Relacionadas con Pilas y Colas 11
5.6. Módulo del Menú Principal 12
6. Pseudo Código Principal Del Sistema 13
6.1. 13

1. Descripción Técnica del Programa

1.1. Introducción

El **Gestor de Notas** es un sistema desarrollado en **Python** que permite registrar, ordenar y analizar cursos junto con sus calificaciones.
Su diseño modular favorece la separación de responsabilidades y facilita la comprensión del código.

El programa implementa algoritmos clásicos de ordenamiento y búsqueda, así como estructuras de datos como **listas**, **pilas** y **colas**.
Su propósito es servir como herramienta educativa para demostrar cómo aplicar estos conceptos en un entorno práctico y controlado.



2. Estructura General del Código

2.1. Descripción General

El código del **Gestor de Notas** se encuentra estructurado de manera modular y organizada, siguiendo los principios de la programación estructurada y la buena práctica de separar las responsabilidades por secciones. Cada bloque tiene un propósito bien definido, lo cual facilita la lectura, el mantenimiento y la futura ampliación del sistema.

La estructura general del código se divide en tres secciones principales:

2.2. Bloque de Funciones

En este bloque se agrupan todas las funciones puras del sistema, es decir, aquellas que **realizan cálculos o transformaciones sin depender de variables globales ni producir efectos secundarios**.
Entre las principales funciones se encuentran:

calcular_promedio_cursos()
Calcula el promedio general del grupo de cursos registrados, utilizando comprensión de listas para recorrer los valores de nota.

ordenar_burbuja()
Implementa el algoritmo de **ordenamiento burbuja**, el cual organiza los cursos de forma descendente según la nota.
Es útil para mostrar los cursos con mejores calificaciones primero.

ordenar_insercion()
Implementa el algoritmo de **ordenamiento por inserción**, utilizado para ordenar los cursos alfabéticamente por su nombre.
Este método es ideal cuando se trabaja con listas pequeñas o parcialmente ordenadas.

busqueda_lineal() y busqueda_binaria()
Permiten localizar un curso dentro de la lista.
La búsqueda lineal recorre secuencialmente cada elemento, mientras que la binaria realiza comparaciones dividiendo el rango de búsqueda, siendo más eficiente sobre listas ordenadas.

Estas funciones constituyen el **núcleo lógico del programa**, ya que permiten procesar la información sin interacción directa con el usuario.

2.3. Bloque de Procedimientos

Contiene las rutinas que se encargan de **la interacción con el usuario**, gestionando entradas, salidas y validaciones.
A diferencia de las funciones, estos procedimientos **sí generan efectos visibles** (como impresiones en pantalla o modificaciones en listas).

Entre los procedimientos más importantes destacan:

registrar_curso(): Solicita al usuario el nombre y la nota de un curso, valida los datos e inserta un nuevo registro en la lista de cursos.

mostrar_cursos(): Muestra todos los cursos registrados junto con su nota y promedio general.

contar_aprobados(): Evalúa cuántos cursos superan la nota mínima de aprobación (60 puntos) y cuántos no.

actualizar_nota(): Permite modificar la calificación de un curso existente.

eliminar_curso(): Elimina un curso de la lista, previa confirmación del usuario.

agregar_a_cola_revision() y procesar_cola_revision(): Simulan el flujo de revisión académica mediante una **cola (FIFO)**.

mostrar_historial(): Utiliza una **pila (LIFO)** para mostrar las últimas acciones realizadas, en orden inverso al que fueron ejecutadas.

Estos procedimientos utilizan internamente las funciones del bloque anterior para procesar los datos, manteniendo así un flujo lógico y coherente.

2.4. Bloque del Programa Principal

El programa principal se encuentra protegido por el bloque condicional:

```
if __name__ == "__main__":
```

Este bloque actúa como **punto de entrada del sistema**, garantizando que el código principal solo se ejecute cuando el archivo es ejecutado directamente, y no cuando es importado como módulo desde otro programa.

Dentro de este bloque se declara el **menú principal**, que controla el flujo completo de ejecución del sistema.

Cada opción del menú invoca las funciones y procedimientos correspondientes, permitiendo al usuario realizar operaciones como registrar, actualizar, eliminar o consultar información.

3. Uso de Estructuras de Datos

3.1. Descripción General

El **Gestor de Notas** emplea diversas estructuras de datos internas que permiten organizar la información de forma eficiente, según el tipo de operación que se desea realizar.

Todas estas estructuras están implementadas mediante **listas dinámicas de Python**, adaptadas para comportarse como listas, pilas o colas según la necesidad.

3.2. Listas

El **Gestor de Notas** emplea diversas estructuras de datos internas que permiten organizar la información de forma eficiente, según el tipo de operación que se desea realizar.

Las listas son el tipo de estructura de datos principal del sistema.

Se utilizan para almacenar los cursos como **tuplas** del tipo (nombre, nota) dentro del objeto principal o en la variable cursos.

Ejemplo:

```
cursos = [("Matemáticas", 90), ("Historia", 80), ("Programación", 95)]
```

Estas listas son dinámicas, lo que permite agregar, modificar o eliminar cursos sin necesidad de definir su tamaño previamente.

Además, son ideales para ser recorridas con bucles, ordenadas con algoritmos o filtradas según criterios específicos.

3.3. PILAS

Las pilas se simulan utilizando también listas, pero aplicando el principio **LIFO (Last In, First Out)**.
En el programa, la pila se utiliza para mantener un **historial de acciones realizadas**, como registros, actualizaciones o eliminaciones de cursos.

Ejemplo de uso:

```
historial.append("Actualizó nota de Programación")
ultima_accion = historial.pop()
```

El método append() inserta una acción al final de la lista, mientras que pop() la retira en orden inverso, garantizando que la última acción registrada sea la primera en mostrarse al consultar el historial.

3.4. COLAS

Las colas también se implementan con listas, pero siguiendo el principio **FIFO (First In, First Out)**.
Esta estructura se emplea para **simular una cola de revisión académica**, donde los cursos se atienden en el mismo orden en que fueron agregados.

Ejemplo de uso:

```
cola_revision.append(("Historia", 70))
curso_revisado = cola_revision.pop(0)
```

De esta manera, el primer curso en entrar es el primero en ser procesado, imitando el comportamiento de una fila de solicitudes.

3.5. Integracion de estructuras

El sistema combina estas estructuras para mantener un flujo de información coherente:
Las **listas** almacenan la información principal (los cursos).
La **pila** conserva un registro histórico de los cambios realizados.
La **cola** gestiona los procesos pendientes de revisión.
Cada una cumple un propósito distinto dentro del mismo entorno de ejecución, lo que refuerza el concepto de modularidad y separación de responsabilidades.

4. Justificación de los algoritmos de ordenamiento

4.1. Idea General

El **Gestor de Notas** incorpora dos algoritmos clásicos de ordenamiento:
el **método burbuja (Bubble Sort)** y el **método de inserción (Insertion Sort)**.
Ambos fueron seleccionados por su **simplicidad**, **legibilidad del código**, y su **valor didáctico**, siendo ideales para proyectos educativos en los que se busca comprender la lógica de comparación e intercambio entre elementos.

A continuación, se detalla su funcionamiento, ventajas y justificación de uso en el contexto del proyecto:

4.2. Ordenamiento Burbuja

El método burbuja es uno de los algoritmos de ordenamiento más conocidos y sencillos de implementar.
Su funcionamiento consiste en **recorrer la lista varias veces**, comparando pares de elementos adyacentes y **intercambiándolos si están en el orden incorrecto**.
Con cada pasada, el elemento más grande (o más pequeño, según el criterio) "burbujea" hasta su posición final.

Aplicación en el proyecto:

En el Gestor de Notas, el ordenamiento burbuja se utiliza para **ordenar los cursos por su nota de manera descendente**.
Esto permite visualizar primero las calificaciones más altas, facilitando la interpretación general del rendimiento académico.

Ejemplo:

Si los cursos tienen las notas [75, 90, 60], el algoritmo realizará los intercambios necesarios hasta obtener [90, 75, 60].

Ventajas:

De fácil comprensión e implementación.
Ideal para listas pequeñas o medianas.
Permite visualizar claramente el proceso de ordenamiento paso a paso.
Favorece la enseñanza del concepto de intercambio y comparación.

Desventajas:

Tiene una **complejidad temporal de $O(n^2)$**, lo que lo vuelve ineficiente para listas grandes.
Realiza muchas comparaciones y pasadas incluso si la lista ya está casi ordenada.

Justificación:

Dado que el número de cursos en este sistema es reducido (usualmente menor a 20), el costo computacional es mínimo.
Además, el algoritmo refuerza los fundamentos de la programación algorítmica, por lo que su implementación cumple un propósito formativo más que de optimización.

4.3. Ordenamiento por Incerción

El ordenamiento por inserción es un algoritmo que **construye una lista ordenada de forma incremental**.
Toma un elemento de la lista no ordenada y lo inserta en su posición correcta dentro de la parte ya ordenada.
Su proceso es similar a cómo una persona ordena las cartas de una baraja en la mano.

Aplicación en el proyecto:

En el Gestor de Notas, este algoritmo se emplea para **ordenar los cursos alfabéticamente por su nombre**.
Esto facilita la búsqueda visual y mejora la legibilidad del listado, especialmente cuando se muestran muchos cursos.

Ejemplo:

Si los cursos son [“Historia”, “Matemáticas”, “Arte”], el algoritmo los reordenará como [“Arte”, “Historia”, “Matemáticas”].

Ventajas:

Muy eficiente para listas pequeñas o casi ordenadas.
Requiere menos intercambios que el método burbuja.
Mantiene la estabilidad del orden (no altera el orden de elementos con valores iguales).
Fácil de adaptar a diferentes criterios de ordenamiento (por nombre, nota, etc.).

Desventajas:

También posee **complejidad O(n²)** en el peor caso.
No es adecuado para listas muy grandes, aunque en este proyecto su impacto es despreciable.

Justificación:

El método de inserción fue elegido por su **balance entre simplicidad y claridad**, y porque **permite demostrar el concepto de comparación y desplazamiento** dentro de una lista de datos.
Además, su aplicación en orden alfabético complementa al método burbuja (que ordena por valor numérico), mostrando dos enfoques distintos del mismo problema.

5. Documentación de Funciones y Módulos

5.1. Funciones de Cálculo y Procesamiento de Datos

a) calcular_promedio_general(cursos)

Propósito:

Calcula el promedio general del grupo de cursos, sumando todas las notas registradas y dividiendo el resultado entre la cantidad total de cursos.

Parámetros:

- cursos: lista de tuplas o diccionarios con la estructura [(nombre, nota), ...].

Retorno:

- Promedio general del grupo (tipo float).

Precondición:

- La lista cursos no debe estar vacía.
- Las notas deben estar en el rango [0, 100].

Postcondición:

- Retorna un valor numérico representando el promedio total.

b) contar_aprobados_reprobados(cursos)

Propósito:

Determina la cantidad de cursos aprobados y reprobados en base a la nota mínima de aprobación (60 puntos).

Parámetros:

- cursos: lista de tuplas o diccionarios.

Retorno:

- Una tupla (aprobados, reprobados) con los conteos respectivos.

Precondición:

- Las notas deben ser válidas (0–100).

Postcondición:

- Retorna los totales correspondientes sin modificar la lista original.

5.2. Funciones de Búsqueda

a) busqueda_lineal(cursos, nombre)

Propósito:

Busca un curso dentro de la lista recorriéndola secuencialmente.

Parámetros:

- cursos: lista de cursos registrados.
- nombre: cadena con el nombre del curso a buscar.

Retorno:

- El índice del curso si se encuentra; de lo contrario, -1.

Justificación:

La búsqueda lineal se implementa por su sencillez y porque no requiere que la lista esté ordenada.

b) busqueda_binaria(cursos, nombre)

Propósito:

Localiza un curso mediante el método de búsqueda binaria, reduciendo el rango de comparación en cada iteración.

Parámetros:

- cursos: lista ordenada alfabéticamente.
- nombre: nombre del curso buscado.

Retorno:

- El índice del curso si existe; -1 en caso contrario.

Precondición:

- La lista debe estar **ordenada** alfabéticamente antes de ejecutar la búsqueda.

Justificación:

Ofrece un mejor rendimiento que la búsqueda lineal para listas grandes, reduciendo el número de comparaciones necesarias.

5.3. Funciones de Ordenamiento

a) ordenar_burbuja(cursos)

Propósito:

Ordena la lista de cursos en orden descendente según la nota obtenida.

Método:

Comparación e intercambio sucesivo de elementos adyacentes.
Complejidad temporal: $O(n^2)$.

Justificación:

Se eligió por su claridad y valor didáctico, ideal para comprender la lógica de ordenamiento paso a paso.

b) ordenar_insercion(cursos)

Propósito:

Ordena los cursos alfabéticamente por nombre, utilizando el algoritmo de inserción.

Método:

Reubica cada elemento en la posición correcta dentro de la parte ya ordenada.
Complejidad temporal: $O(n^2)$.

Justificación:

Más eficiente que burbuja en listas parcialmente ordenadas, y excelente para mostrar conceptos de desplazamiento controlado.

5.4. Procedimientos de Gestión de Cursos

a) registrar_curso(cursos, historial)

Propósito:

Solicita al usuario ingresar el nombre y la nota de un curso, valida los datos y registra el nuevo curso.

Precondición:

- La nota debe ser numérica y estar dentro del rango [0, 100].

Postcondición:

- Se agrega el nuevo curso a la lista y se registra la acción en la pila de historial.

b) mostrar_cursos(cursos)

Propósito:

Imprime todos los cursos registrados junto con su nota y el promedio general del grupo.

Comportamiento adicional:

Si no existen cursos registrados, muestra un mensaje de advertencia.

c) actualizar_nota(cursos, historial)

Propósito:

Permite modificar la calificación de un curso existente.
Actualiza la lista y registra el cambio en la pila de historial.

Precondición:

- El curso debe existir.
- La nueva nota debe ser válida.

Postcondición:

- La nota del curso se actualiza correctamente.

d) eliminar_curso(cursos, historial)

Propósito:

Elimina un curso de la lista principal tras confirmación del usuario.
También registra la eliminación en la pila de historial.

Postcondición:

- El curso es removido definitivamente de la lista.

5.5. Funciones Relacionadas con Pilas y Colas

a) agregar_a_cola_revision(cola_revision)

Propósito:

Simula el proceso de revisión de notas agregando cursos a una cola de solicitudes.
La estructura sigue el principio **FIFO (First In, First Out)**.

Postcondición:

- El curso se encola correctamente y queda en espera de revisión.

b) procesar_cola_revision(cola_revision)

Propósito:

Atiende las solicitudes de revisión en el orden en que fueron recibidas.
Cada curso revisado se desencola y se notifica al usuario.

Postcondición:

- La cola se vacía a medida que se procesan los elementos.

c) mostrar_historial(historial)

Propósito:

Muestra el registro de acciones almacenadas en la **pila (LIFO)**.
Cada acción reciente se muestra primero, en orden inverso al que fue ejecutada.

Precondición:

- Debe haberse realizado al menos una acción modificadora (registro, actualización o eliminación).

Postcondición:

- Se despliega la secuencia histórica sin alterar el contenido de la pila.

5.6. Módulo del Menú Principal

menu_principal()

Propósito:

Controla la navegación general del sistema y permite al usuario acceder a todas las funcionalidades mediante un menú numérico.

Opciones disponibles:

1. Registrar nuevo curso
2. Mostrar todos los cursos y notas
3. Calcular promedio general
4. Contar cursos aprobados y reprobados
5. Buscar curso por nombre (búsqueda lineal)
6. Actualizar nota de un curso
7. Eliminar un curso
8. Ordenar cursos por nota (burbuja)
9. Ordenar cursos por nombre (inserción)
10. Buscar curso por nombre (búsqueda binaria)
11. Simular cola de solicitudes de revisión
12. Mostrar historial de cambios (pila)
13. Salir del sistema

Flujo interno:

Cada opción invoca funciones y procedimientos específicos.
Al finalizar cada operación, el sistema regresa al menú principal, permitiendo al usuario continuar con nuevas acciones sin reiniciar el programa.

Postcondición:

- El usuario puede interactuar con todas las funciones desde una única interfaz.

6. Pseudo Código Principal Del Sistema

6.1.

```
ALGORITMO GestorNotasAcademicas
   # Inicialización de estructuras
   cursos ← lista vacía
   historial ← lista vacía
   cola_revision ← lista vacía

   REPETIR
      MOSTRAR menú principal:
         1. Registrar nuevo curso
         2. Mostrar cursos y notas
         3. Calcular promedio general
         4. Contar aprobados y reprobados
         5. Buscar curso (búsqueda lineal)
         6. Actualizar nota
         7. Eliminar curso
         8. Ordenar por nota (burbuja)
         9. Ordenar por nombre (inserción)
         10. Buscar curso (búsqueda binaria)
         11. Simular cola de revisión
         12. Mostrar historial de acciones
         13. Salir

      LEER opcion

      SEGÚN opcion HACER
         CASO 1:
            LLAMAR registrar_curso(cursos, historial)
         CASO 2:
            LLAMAR mostrar_cursos(cursos)
         CASO 3:
            promedio ← calcular_promedio_general(cursos)
            IMPRIMIR promedio
         CASO 4:
            (aprob, reprob) ← contar_aprobados_reprobados(cursos)
            IMPRIMIR resultados
         CASO 5:
            LLAMAR busqueda_lineal(cursos, nombre)
         CASO 6:
            LLAMAR actualizar_nota(cursos, historial)
         CASO 7:
            LLAMAR eliminar_curso(cursos, historial)
```

```
            CASO 8:
               LLAMAR ordenar_burbuja(cursos)
            CASO 9:
               LLAMAR ordenar_insercion(cursos)
            CASO 10:
               LLAMAR busqueda_binaria(cursos, nombre)
            CASO 11:
               LLAMAR agregar_a_cola_revision(cola_revision)
               LLAMAR procesar_cola_revision(cola_revision)
            CASO 12:
               LLAMAR mostrar_historial(historial)
            CASO 13:
               IMPRIMIR "Saliendo del sistema..."
               SALIR DEL PROGRAMA
            CASO CONTRARIO:
               IMPRIMIR "Opción inválida, intente nuevamente."
         FIN SEGÚN

      HASTA opcion = 13
   FIN ALGORITMO
```