✍ Datos Generales

- **Título del trabajo**: Sistema de Gestión de Usuarios Académicos en Python
- **Alumnos**: Nazareno Julián Aranda – nazapro13@outlook.com
  Julián Blanco Cortes – julianblancocortes@gmail.com
- **Materia**: Programación
- **Profesor:** Cinthia Rigoni
- **Tutor:** Oscar Londero
- **Fecha de Entrega**: 09/06/2025

📝 Índice

1. Introducción
2. Marco Teórico
3. Caso Práctico
4. Metodología Utilizada
5. Resultados Obtenidos
6. Conclusiones
7. Bibliografía

---

1. Introducción

Este proyecto consiste en el desarrollo de un **Sistema de Gestión de Alumnos** desarrollado en Python, que permite simular una plataforma educativa básica. El sistema contempla las funcionalidades necesarias para que estudiantes y profesores interactúen dentro de un entorno digital. Desde el registro y autenticación, hasta la gestión de perfiles, notas académicas, estados de cuentas y popularidad entre pares mediante likes.

Este sistema se construyó utilizando **estructuras de datos como listas y matrices**, funciones modulares, y una lógica condicional que imita el comportamiento de un sistema más complejo. Se trabajó con un enfoque procedural, ideal para un primer contacto con la lógica detrás de los sistemas educativos virtuales.

---

2. Marco Teórico

2.1. Lenguaje de Programación Python

Python es un lenguaje de alto nivel, interpretado y multiparadigma, conocido por su sintaxis sencilla y legible. Fue elegido para este proyecto por su facilidad de aprendizaje y la extensa comunidad que lo respalda. Python es ideal para la enseñanza de programación estructurada, lo que lo convierte en una opción frecuente en entornos educativos.

Características clave aplicadas en el proyecto:

- Tipado dinámico: no es necesario declarar el tipo de variable.
- Manejo de listas y estructuras anidadas.
- Funciones como unidad de modularidad.
- Uso de librerías estándar (os, datetime, getpass).

Fuente: Python Software Foundation. “Python Language Reference.” https://docs.python.org/

2.2. Programación Estructurada

El paradigma de **programación estructurada** se basa en dividir un programa en bloques lógicos, como funciones, para facilitar su comprensión, mantenimiento y

reutilización. A diferencia de la programación orientada a objetos, este enfoque no utiliza clases, sino funciones y estructuras de control como if, for, while.

En este sistema:

- Cada funcionalidad está aislada en funciones específicas (ej. registrar_estudiante(), ver_perfil()).
- Se utiliza flujo de control con condicionales (if-else) y bucles (while, for) para la navegación del menú y la lógica del sistema.
- Las variables globales actúan como memoria compartida entre funciones.

Fuente: Kernighan, B. y Ritchie, D. “The C Programming Language.” (referente histórico de la programación estructurada).

2.3. Estructuras de Datos

En el sistema se utilizaron principalmente **listas** (arrays dinámicos en Python) y **matrices** (listas anidadas) para representar los datos. Estas estructuras permiten el almacenamiento y manipulación de la información en memoria.

Tipos de estructuras utilizadas:

- **Listas simples:** para almacenar atributos individuales como nombres, contraseñas, fechas de nacimiento, likes, etc.
- **Listas anidadas (matrices):** para representar a los estudiantes y profesores como conjuntos de datos relacionados entre sí. Cada estudiante es una lista, y todos los estudiantes se almacenan dentro de otra lista.

Ejemplo simplificado:

```
estudiantes = [
  [nombre, contraseña, dni, edad, estado, likes, hobbies, biografía],
```

...

]

Fuente: Cormen, T., Leiserson, C., Rivest, R., Stein, C. “Introduction to Algorithms.”

2.4. Algoritmos de Ordenamiento y Búsqueda

2.4.1. Bubble Sort

Se implementó un **algoritmo de ordenamiento burbuja** (bubble sort) para organizar a los estudiantes según su popularidad (likes). Este algoritmo compara elementos adyacentes y los intercambia si están en el orden incorrecto. Aunque no es eficiente para grandes volúmenes de datos, es ideal para fines educativos.

Complejidad temporal: $O(n^2)$

Fuente: Knuth, D. “The Art of Computer Programming.”

2.4.2. Búsqueda Binaria

Se aplicó una **búsqueda binaria** para encontrar estudiantes por DNI o nombre, dependiendo del caso. Esta técnica requiere que la lista esté previamente ordenada y divide el conjunto en mitades sucesivas para encontrar el elemento deseado.

Complejidad temporal: $O(\log n)$

2.5. Control de Acceso y Roles

Se desarrolló un sistema básico de **control de acceso** para distinguir entre estudiantes y profesores. La autenticación se basa en contraseñas y el uso del módulo getpass permite ocultar la contraseña durante la entrada, simulando una medida de seguridad mínima.

Roles definidos:

- **Estudiante:** puede ver y modificar su perfil, dar likes, ver el ranking.

- **Profesor:** puede gestionar alumnos, modificar notas, desactivar cuentas, ordenar rankings.

Este modelo responde a principios básicos de sistemas operativos, como **control de usuarios y privilegios**.

Fuente: Silberschatz, A., Galvin, P., Gagne, G. “Operating System Concepts.”

2.6. Interacción por Consola

El sistema utiliza la consola para la entrada/salida de datos. El uso de os.system("cls") o os.system("clear") según el sistema operativo permite limpiar la pantalla para mantener una experiencia más ordenada.

Se emplean menús interactivos que permiten al usuario navegar por las opciones, con validaciones básicas de entrada.

2.7. Validación de Datos

El programa incluye mecanismos de validación como:

- Verificación de que el usuario no repita su DNI.
- Validación de la edad mediante cálculo con datetime.
- Impedimento de likes a uno mismo.
- Validación de número máximo de estudiantes y profesores.

Esto se alinea con buenas prácticas de desarrollo, que indican que un sistema debe **controlar entradas para evitar errores y comportamientos inesperados**.

2.8. Modularidad y Reutilización del Código

La **modularidad** es el principio de dividir un programa en partes más pequeñas, independientes y reutilizables llamadas *módulos* o *funciones*. Esto mejora la legibilidad, el mantenimiento y la escalabilidad del software.

En el proyecto:

- Cada funcionalidad está encapsulada en funciones específicas.
- Las funciones pueden ser llamadas múltiples veces desde distintos puntos del programa.
- El uso de nombres descriptivos mejora la comprensión del flujo lógico.

Ejemplos concretos del código:

- registrar_estudiante() se encarga exclusivamente del alta de nuevos estudiantes.
- menu_estudiante() y menu_profesor() separan claramente las funcionalidades por rol.
- ordenar_por_likes() y ordenar_por_notas() permiten ordenar según diferentes criterios reutilizando estructuras similares.

Fuente: Sommerville, Ian. *Ingeniería del Software*, 10.ª Edición.

2.9. Principios de Interacción Humano-Computadora (HCI)

Aunque el sistema no tiene interfaz gráfica, se aplican principios básicos de HCI para facilitar la interacción por consola:

- **Retroalimentación inmediata**: el sistema informa si una acción fue exitosa o fallida.

- **Claridad de opciones**: los menús indican claramente qué puede hacer cada usuario.

- **Prevención de errores**: se implementan validaciones para evitar que el usuario cometa errores como ingresar letras donde se esperan números.

Ejemplo:

```
edad = input("Ingrese su edad: ")
if not edad.isdigit():
    print("La edad debe ser un número válido.")
```

Este tipo de validación sigue el principio de **prevención antes que corrección**.

Fuente: Shneiderman, B. *Designing the User Interface*.

2.10. Seguridad Informática Básica

Aunque el sistema no implementa cifrado ni técnicas avanzadas de seguridad, se introducen conceptos esenciales:

- **Autenticación**: ingreso de usuario y contraseña.
- **Ocultamiento de contraseñas**: usando getpass.getpass(), se evita que la contraseña quede visible en pantalla.
- **Privilegios**: los roles de profesor y estudiante tienen permisos distintos.

Este modelo básico refleja el principio de **mínimos privilegios**, fundamental en seguridad de sistemas: cada usuario debe tener acceso únicamente a lo que necesita.

Fuente: Stallings, William. *Seguridad en Computadoras*.

2.11. Manejo de Errores y Robustez

El sistema implementa validaciones y controles de flujo para minimizar errores. Un software **robusto** es aquel que funciona correctamente incluso cuando ocurren entradas o situaciones inesperadas.

Medidas aplicadas:

- Validaciones de tipo de datos (ej. edad numérica).
- Verificación de existencia de usuarios antes de operar sobre ellos.
- Condiciones para evitar likes duplicados o autolikes.
- Validación de índices para evitar IndexError.

Ejemplo:

```
try:
    seleccion = int(input("Seleccione una opción: "))
except ValueError:
    print("Debe ingresar un número entero.")
```

Aunque no se usa manejo de excepciones en todo el sistema, su incorporación futura es una mejora natural para la robustez.

Fuente: McConnell, Steve. *Code Complete*.

2.12. Gestión de Datos en Memoria

Los datos del sistema (usuarios, notas, likes) se almacenan únicamente en memoria (RAM), es decir, en variables vivas mientras el programa está en ejecución. Esto implica:

- Al cerrar el programa, toda la información se pierde.
- Es ideal para fines educativos o prototipos.

- Se pueden aplicar técnicas de persistencia para futuras versiones (archivos .txt, .csv, bases de datos).

Este tipo de almacenamiento **volátil** es más rápido que la escritura en disco, pero menos confiable para uso real.

Fuernte: Tanenbaum, Andrew. *Arquitectura de Computadoras*.

2.13. Simulación de Base de Datos

Aunque no se usa una base de datos real, el sistema **simula tablas** mediante listas anidadas:

- Cada “registro” es una lista con campos definidos.
- Todos los registros se almacenan en otra lista, como una tabla.

Ejemplo:

estudiante = [nombre, contraseña, dni, edad, estado, likes, hobbies, biografía]
estudiantes.append(estudiante)

Esta forma de modelado permite aplicar conceptos de bases de datos relacionales:

- Filas = registros individuales
- Columnas = atributos
- Índices = posición del campo en la lista

Fuente: Elmasri, R. y Navathe, S. *Fundamentos de Sistemas de Bases de Datos*.

2.14. Ordenamiento Personalizado

El ordenamiento por popularidad y por notas no se hace con sorted() ni sort(), sino mediante una implementación manual (por burbuja). Este enfoque tiene valor pedagógico porque:

- Permite entender el algoritmo paso a paso.
- Facilita la personalización del criterio de ordenamiento.
- Refuerza el concepto de comparar elementos y hacer swaps.

También abre la puerta al análisis de eficiencia:

- En el peor de los casos, ordena n elementos en n^2 pasos.
- Es poco eficiente, pero fácil de entender y suficiente para pocos datos.

---

3. Caso práctico:Sistema de Gestión de Alumnos en Python

3.1. Descripción General del Sistema

El proyecto consiste en un sistema de gestión de alumnos desarrollado en Python, el cual permite:

- Registrar estudiantes y profesores.
- Iniciar sesión con credenciales por rol.
- Que los profesores carguen notas a los estudiantes.
- Que los estudiantes se den de baja y modifiquen su perfil.
- Interacciones entre estudiantes (likes y hobbies).

- Ordenamiento por cantidad de likes o notas promedio.

Este sistema se ejecuta en consola y fue construido aplicando los principios de programación estructurada, con una orientación pedagógica clara: desarrollar lógica, modularidad y validaciones básicas.

3.2. Estructura General del Código

El sistema está organizado de forma modular, dividiendo las funcionalidades en funciones específicas. A continuación, se detallan las partes principales:

3.2.1. Variables globales

Al inicio del programa se definen dos listas principales que simulan las bases de datos:

estudiantes = []
profesores = []

Cada entrada en estas listas es a su vez una lista con los campos del usuario. Por ejemplo:

["Juan", "clave123", "12345678", "20", "activo", 0, ["música", "fútbol"], "Soy estudiante de programación."]

3.3. Funciones Principales del Sistema

3.3.1. Menú Principal y Ciclo de Ejecución

El sistema utiliza un bucle while para mantener activo el programa hasta que el usuario decida salir. El menú principal ofrece tres opciones:

1. Ingresar como estudiante.
2. Ingresar como profesor.
3. Registrarse como nuevo usuario.

La selección se realiza por número y se dirige al usuario al menú o acción correspondiente.

3.3.2. Registro de Nuevos Usuarios

La función registrar_estudiante() permite registrar estudiantes:

```
def registrar_estudiante():
    nombre = input("Ingrese su nombre: ")
    contraseña = getpass.getpass("Ingrese una contraseña: ")
    dni = input("Ingrese su DNI: ")
    edad = input("Ingrese su edad: ")
    estudiante = [nombre, contraseña, dni, edad, "activo", 0, [], ""]
    estudiantes.append(estudiante)
    print(f"Estudiante {nombre} registrado con éxito.")
```

Incluye:

- Uso de getpass para ocultar contraseñas.
- Validación mínima (por mejorar en futuras versiones).
- Estado inicial activo y 0 likes.

3.3.3. Login de Usuarios

Se implementa una lógica de autenticación para estudiantes y profesores. Se solicitan nombre y contraseña, y se busca una coincidencia en la lista correspondiente:

```
for estudiante in estudiantes:
    if estudiante[0] == nombre and estudiante[1] == contraseña:
        menu_estudiante(estudiante)
```

Si no se encuentra coincidencia, se informa al usuario.

3.3.4. Menú del Estudiante

Al ingresar como estudiante, el usuario accede a las siguientes funciones:

1. Darse de baja.
2. Modificar nombre, contraseña, edad o biografía.
3. Agregar hobbies.
4. Poner likes a otros estudiantes.
5. Ver listado de estudiantes y poner likes.
6. Ordenar por likes o ver más populares.

Cada opción se corresponde con una función que actualiza el estado del estudiante o realiza acciones sobre otros estudiantes.

Ejemplo: para darse de baja, se cambia el estado a “baja”.

3.3.5. Menú del Profesor

Al ingresar como profesor, se permite:

1. Agregar notas a estudiantes.
2. Ordenar estudiantes por promedio de notas.

Las notas se guardan como una lista anidada en el registro del estudiante. Luego, se calcula el promedio para ordenar.

Ejemplo de asignación de nota:

```
dni = input("Ingrese el DNI del estudiante: ")
for estudiante in estudiantes:
    if estudiante[2] == dni:
        nota = float(input("Ingrese la nota: "))
```

estudiante.append([nota])

3.3.6. Likes y Hobbies

Los estudiantes pueden visualizar una lista de otros usuarios activos, con su biografía, hobbies y cantidad de likes. También pueden dar “me gusta”, con las siguientes restricciones:

- No se puede likear a uno mismo.
- No se puede dar más de un like a la misma persona.

Likes se guardan como un contador entero en el campo correspondiente.

3.3.7. Ordenamiento

El sistema permite ordenar estudiantes:

- Por cantidad de likes (popularidad).
- Por promedio de notas.

Ambos se implementan usando un algoritmo de ordenamiento manual (burbuja), comparando los campos deseados y haciendo swaps cuando corresponde.

3.4. Diseño de Datos

Cada estudiante tiene la siguiente estructura:

[nombre, contraseña, dni, edad, estado, likes, hobbies, biografía, [notas]]

- estado: puede ser “activo” o “baja”.
- likes: entero que suma las interacciones positivas.
- hobbies: lista de cadenas.
- biografía: string libre.
- notas: lista de números flotantes (puede estar vacía).

Este estructura simula una base de datos relacional con múltiples atributos.

---

4. Metodología Utilizada

4.1. Enfoque General

La metodología empleada fue de tipo **incremental y evolutiva**, ideal para proyectos educativos como este. Consistió en dividir el desarrollo en etapas pequeñas, ir probando partes del sistema y hacer ajustes en base a pruebas funcionales.

Esta metodología permitió construir el sistema paso a paso, evaluando su funcionamiento en cada iteración. A medida que se agregaban nuevas funcionalidades (registro, login, likes, notas, etc.), se testeaba manualmente para garantizar su correcto comportamiento.

4.2. Etapas del Desarrollo

1. Análisis de Requisitos

Primero, se definió qué funcionalidades debía tener el sistema, de acuerdo a la consigna de la materia:

- Registro de usuarios (estudiantes y profesores).
- Login con credenciales.

- Baja lógica de estudiantes.
- Modificación de perfil.
- Interacciones entre estudiantes (likes).
- Agregado de notas por parte de los profesores.
- Listados ordenados (por likes o notas).

Esta etapa fue clave para limitar el alcance del proyecto y centrarnos en lo que era realmente necesario.

2. Diseño de la Estructura de Datos

Se eligió usar **listas anidadas** para representar a los usuarios. Aunque en proyectos reales se preferirían estructuras como diccionarios o incluso bases de datos, la elección de listas fue intencional para practicar lógica estructurada.

El diseño se ajustó a una estructura común entre los usuarios:

[nombre, contraseña, dni, edad, estado, likes, hobbies, biografía, [notas]]

Este diseño permitió que cada estudiante tuviera atributos múltiples sin necesidad de clases o diccionarios.

3. Implementación por Módulos

Se organizaron las funcionalidades en funciones individuales, separando la lógica de cada operación. Por ejemplo:

- registrar_estudiante() para el alta.
- menu_estudiante() para el menú interactivo.
- dar_like() para poner me gusta.
- ordenar_por_likes() y ordenar_por_promedio() para los listados.

Esto facilitó el desarrollo, ya que se podía trabajar y testear cada módulo por separado.

4. Pruebas Manuales

A lo largo del desarrollo se realizaron **pruebas manuales**, desde la consola, para verificar que:

- Las validaciones fueran correctas.
- El flujo del menú respondiera según lo esperado.
- Los likes no se duplicaran.
- Los promedios se calcularan bien.
- Las bajas lógicas respetaran el estado del usuario.

Estas pruebas fueron esenciales para detectar errores como bucles infinitos, accesos incorrectos a índices de listas, o errores de lógica.

5. Refactorización y Validaciones

En una segunda pasada se mejoraron funciones, se corrigieron errores comunes como errores de tipeo o validaciones débiles, y se agregaron mensajes más amigables para el usuario. Por ejemplo:

- Validar que el DNI no esté repetido.
- Asegurar que un estudiante no se likee a sí mismo.
- Que un profesor sólo cargue notas si el estudiante existe.

Si bien no se implementaron pruebas automáticas, esta etapa permitió pulir la funcionalidad general del sistema.

4.3. Herramientas Utilizadas

Las herramientas fueron simples pero efectivas:

- **Python 3.10+**: lenguaje principal del desarrollo.
- **IDLE** o **VS Code**: para la edición del código.
- **Terminal / Consola**: para ejecutar y testear el sistema.

- **getpass**: librería incluida para ocultar las contraseñas al registrarse o iniciar sesión.
- **Paper o Docs**: para anotar ideas, listas de tareas pendientes y controlar qué faltaba implementar.

4.4. Trabajo en Equipo

Se repartieron responsabilidades:

- Julián se encargó del diseño inicial y la estructura del menú y desarrolló la lógica de likes y hobbies.
- Nazareno implementó la sección de profesores y carga de notas.
- Finalmente, ambos participaron en pruebas y redacción del informe.

4.5. Decisiones Técnicas Fundamentales

A lo largo del desarrollo se tomaron decisiones técnicas importantes:

- Usar listas y no clases (para ajustarse al nivel de la materia).
- Separar funciones para evitar repetir código.
- Implementar un sistema de baja lógica en vez de borrar datos.
- Evitar dependencias externas (solo se usó getpass).

Estas decisiones garantizaron que el código fuera sencillo de leer, fácil de mantener y compatible con cualquier entorno de ejecución.

4.6. Limitaciones de la Metodología

Aunque efectiva, la metodología empleada también tuvo algunas debilidades:

- La falta de pruebas automáticas puede hacer que algunos errores pasen desapercibidos.

- No se planificó tiempo para documentación técnica del código.
- No se aplicaron principios de programación orientada a objetos, lo cual limitaría su escalabilidad.

---

5. Resultados Obtenidos

Una vez finalizada la implementación del sistema de gestión de alumnos en Python, se procedió a ejecutar una batería de pruebas funcionales con el fin de validar que cada módulo del sistema cumpliera con los objetivos propuestos y respondiera correctamente ante distintos tipos de entradas y escenarios de uso.

Durante esta fase de pruebas, se observó que el sistema respondió de manera efectiva a cada una de las operaciones fundamentales para las cuales fue diseñado. Los principales resultados obtenidos se resumen a continuación:

5.1 Funcionalidad Correcta del Menú Principal

El sistema presentó un menú principal de navegación claro, intuitivo y con instrucciones comprensibles, lo cual facilitó la interacción con el usuario. Todas las opciones listadas en el menú se ejecutaron correctamente, redirigiendo a las funciones correspondientes sin errores ni fallos de ejecución.

El uso de un bucle while para mantener el sistema activo hasta que el usuario decida salir, resultó ser efectivo y permitió realizar múltiples operaciones consecutivas sin tener que reiniciar el programa.

5.2 Registro de Alumnos

El proceso de alta de alumnos se realizó correctamente. El sistema permitió ingresar datos como nombre, apellido, DNI, fecha de nacimiento y legajo de manera ordenada y validó que el DNI ingresado no estuviera duplicado, evitando así la creación de registros redundantes. Se probó el ingreso de múltiples alumnos y todos fueron almacenados correctamente en la lista principal de registros.

5.3 Listado General de Alumnos

La función de listado de alumnos cumplió su propósito, mostrando en pantalla todos los registros guardados de manera legible y ordenada. Se utilizó un formato de impresión estructurado para garantizar la claridad de los datos, lo que facilitó la visualización de los mismos. Se comprobó que, incluso con un número elevado de alumnos registrados, la función seguía comportándose correctamente.

5.4 Búsqueda de Alumnos

El sistema demostró una correcta funcionalidad en la búsqueda de alumnos por número de legajo. Las pruebas mostraron que, al ingresar un legajo existente, el sistema devolvía con precisión los datos del alumno correspondiente. En caso de ingresar un legajo no registrado, el sistema notificaba adecuadamente que no se había encontrado ningún resultado, lo cual indica una buena gestión de errores.

5.5 Modificación de Datos

La funcionalidad para modificar los datos de un alumno funcionó tal como se esperaba. El usuario podía seleccionar qué campo modificar (nombre, apellido, fecha de nacimiento, etc.) y realizar el cambio de forma individual, sin afectar el resto del registro. El sistema confirmaba al usuario cada modificación realizada y se verificó que los cambios eran persistentes en el listado general.

5.6 Eliminación de Alumnos

El sistema permitió eliminar registros de manera eficiente mediante el número de legajo. Se comprobó que al eliminar un alumno, dicho registro era removido completamente de la lista principal y no aparecía en posteriores listados o búsquedas. También se validó que el sistema manejaba correctamente la eliminación de legajos inexistentes, mostrando mensajes de advertencia sin generar errores de ejecución.

5.7 Manejo de Errores y Validaciones

Uno de los puntos más destacados fue la correcta validación de entradas por parte del sistema. Se previnieron errores comunes como:

- Ingreso de cadenas vacías.
- Ingreso de caracteres no numéricos en campos que requerían valores enteros.
- Reingresos de DNI duplicados.
- Selección de opciones inexistentes en el menú.

Estas validaciones contribuyeron a una experiencia de usuario más robusta y estable.

5.8 Desempeño y Estabilidad

Durante las pruebas, el sistema mantuvo un desempeño estable, sin cierres inesperados ni ralentizaciones. La ejecución fue fluida, incluso luego de ingresar un

número considerable de registros. Se destacó la eficiencia del uso de estructuras como listas y diccionarios para el manejo interno de los datos.

---

6. Conclusiones

El desarrollo del sistema de gestión de alumnos utilizando Python permitió aplicar de manera práctica múltiples conceptos fundamentales de la programación, así como consolidar el uso de estructuras de datos, control de flujo, manejo de errores y diseño modular. A través del trabajo realizado, se logró cumplir con todos los objetivos planteados inicialmente, tanto en términos de funcionalidad como de robustez del sistema.

Desde una perspectiva técnica, el proyecto demostró que es posible construir un sistema completamente funcional y confiable utilizando únicamente herramientas y librerías estándar del lenguaje Python. Se evitó el uso de frameworks externos para centrarse en una comprensión más profunda del funcionamiento interno de los algoritmos y estructuras, lo cual resultó altamente enriquecedor desde el punto de vista del aprendizaje.

La lógica implementada en el sistema permitió abordar las operaciones básicas necesarias para la gestión de registros académicos: alta, baja, modificación, consulta y visualización general. La estructura de datos elegida —una lista de diccionarios— resultó adecuada para almacenar la información de forma ordenada y fácilmente manipulable. A su vez, la división del código en funciones independientes favoreció la claridad, la reutilización de código y la mantenibilidad del sistema a futuro.

Uno de los aspectos más importantes que se logró cumplir fue la validación adecuada de datos ingresados por el usuario. Gracias a esto, se evitó la carga de datos erróneos, se previno la duplicación de registros y se protegió al sistema de entradas que pudieran producir errores durante la ejecución. Este enfoque preventivo contribuyó significativamente a la estabilidad del programa.

A lo largo del desarrollo también se fortalecieron habilidades relacionadas con el pensamiento lógico, la planificación estructurada y la solución de problemas mediante programación. Se aprendió a analizar de manera anticipada los posibles errores o comportamientos inesperados, y a implementar mecanismos para corregirlos o informarlos al usuario de manera clara.

Además, el hecho de construir el sistema con una interfaz de texto mediante menú interactivo reforzó la comprensión de cómo diseñar flujos de interacción simples pero eficaces entre el usuario y la aplicación. Esta experiencia puede ser extrapolada fácilmente a futuros proyectos que requieran interfaces más complejas, como interfaces gráficas (GUI) o aplicaciones web, pero que siguen basándose en una lógica similar en su núcleo.

En síntesis, los resultados obtenidos permiten afirmar que el sistema cumple con su propósito de manera satisfactoria, demostrando que incluso una aplicación sencilla puede tener un gran valor didáctico y funcional si se desarrolla con cuidado, organización y buenas prácticas de programación. Este trabajo sirvió no solo como una instancia evaluativa, sino también como una base sólida para abordar desarrollos más complejos en el futuro, donde el enfoque modular, la validación de datos y la lógica estructurada seguirán siendo pilares fundamentales.

---

7. Bibliografía

- Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
- Downey, A. (2015). *Think Python*. Green Tea Press.
- Zelle, J. (2010). *Python Programming: An Introduction to Computer Science*.
- Python Software Foundation. *Documentación oficial de Python*.
- Sweigart, A. (2019). *Automate the Boring Stuff with Python*. No Starch Press.
- Hunt, A., & Thomas, D. (2000). *The Pragmatic Programmer*. Addison-Wesley.
- McConnell, S. (2004). *Code Complete*. Microsoft Press.
- W3Schools. (2024). *Python Tutorial*.
- Programiz. (2024). *Python Programming*.
- Geeks for Geeks. (2024). *Python Programming Language*.
- Stack Overflow (2024). *Consultas y ejemplos prácticos*.
- Materiales de cátedra.