Trabajo Práctico Integrador: Informe



Profesor: Gianluca Piriz

Materia: Programación Avanzada

Comisión 4

Tema: Evaluación de notas, Exploración de Conceptos Avanzados de Programación Orientada a Objetos (POO) en Python.

Integrantes:

Piriz Axel

Ruiz Franco Javier

Tapia Ferrufino Rocio Melina

Vargas Kevin Daniel

Índice

[**1.INTRODUCCIÓN 3**](#_llmvy7su4f2l)

[**2. DESARROLLO 3**](#_g8qp4rr0gulw)

[**3. DISEÑO Y ESTRUCTURA DE CÓDIGO 5**](#_bjws56622z7b)

[**4. RESULTADOS Y DIFICULTADES 7**](#_ci7ldklo87sc)

[**5. FUTURAS MEJORAS 9**](#_oiq8u3if7yh7)

[**6. CONCLUSIÓN FINAL 9**](#_m34620orfd9r)

# 

# 

# 1.**INTRODUCCIÓN**

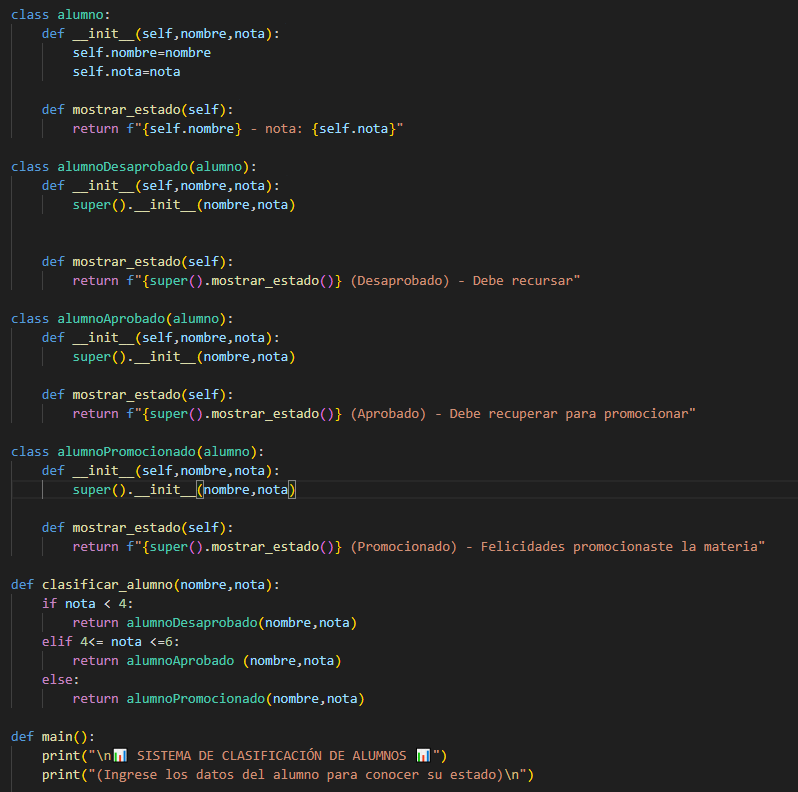
Para este proyecto, decidimos trabajar con el tema de **evaluación de notas**, aplicando los conocimientos adquiridos en la materia, en particular aquellos relacionados con la **Exploración de Conceptos Avanzados de Programación Orientada a Objetos (POO) en Python**. Este enfoque nos permitió no solo resolver un problema concreto relacionado con el ámbito educativo, sino también profundizar en el diseño y la implementación de clases, objetos, encapsulamiento, herencia, y otras herramientas clave de la programación orientada a objetos.

A través del desarrollo de este sistema, buscamos simular una aplicación capaz de gestionar el rendimiento académico de estudiantes, permitiendo registrar notas, calcular promedios y generar reportes automáticos.

# 2. **DESARROLLO**

Tuvimos reuniones y charlas mediante discord y whatsapp donde fuimos tomando las decisiones para ver cómo realizar el código de diferente manera, donde uno transmite la pantalla y los demás iban ayudando con el código para realizar el trabajo del mismo

En un principio, implementamos una solución con **1 clase padre y 3 herencias** para diferenciar las situaciones de los alumnos, junto con un bucle **while** al final para gestionar el flujo. Sin embargo, buscamos una alternativa más eficiente que redujera la complejidad, eliminando la necesidad del **while** y disminuyendo el número de herencias.



Esto nos llevó a **rediseñar el código utilizando 2 clases padre y 1 herencia múltiple**, logrando una estructura más simplificada y mantenible.

Posteriormente, **ampliamos la funcionalidad**: en lugar de solo calcular la nota final, agregamos un sistema para:

1. Calcular el **promedio de múltiples notas** del alumno.
2. **Clasificar** su estado académico en base al promedio:
   * **Desaprobado** (si no alcanza el mínimo).
   * **Aprobado** (si cumple con el requisito).
   * **Promocionado** (si supera un puntaje destacado).

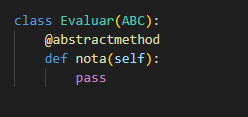
Esta mejora hizo que el código fuera más completo y útil para evaluar el rendimiento de los estudiantes de manera integral.

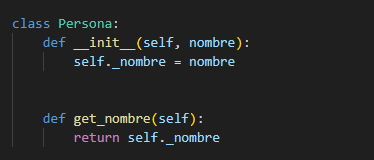
# 3. **DISEÑO Y ESTRUCTURA DE CÓDIGO**

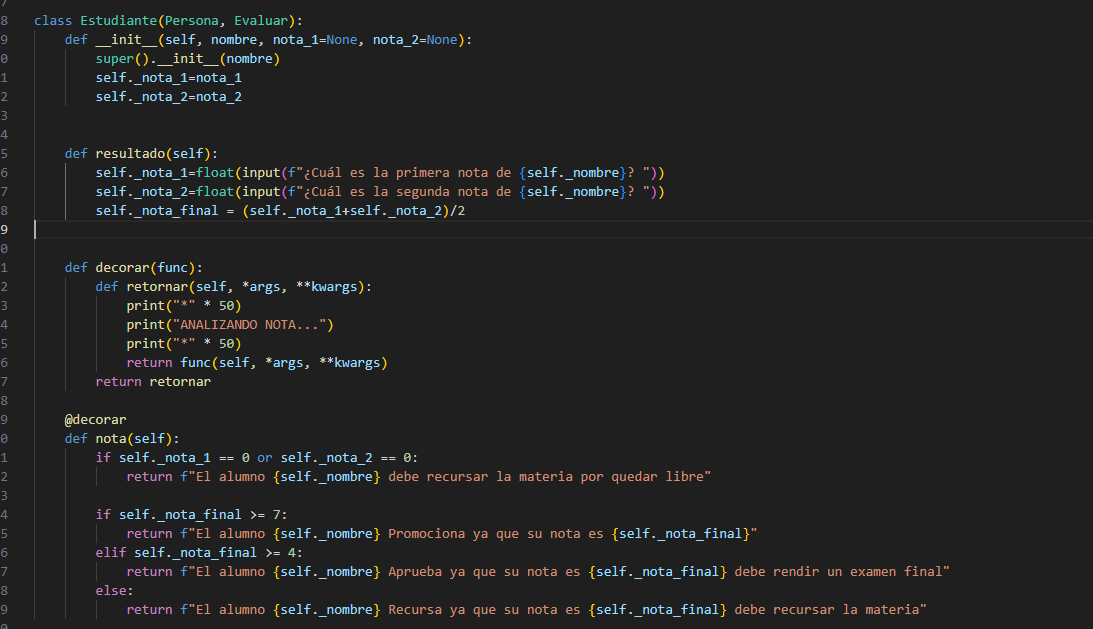
La estructura de nuestro proyecto está compuesta por tres clases principales: **Evaluar** y **Persona**, que actúan como clases base, y **Estudiante**, que hereda de ambas. Cada una de estas clases cumple un rol específico dentro del sistema, implementando funcionalidades particulares que permiten organizar el programa de forma modular.

Este enfoque favorece la reutilización de código, facilita el mantenimiento y promueve una mejor organización del proyecto. Al mantener las responsabilidades separadas en clases distintas, logramos un código más limpio, claro y escalable, creando así un entorno más eficiente y agradable para el desarrollo y futuras mejoras a continuación se destaca la utilización de cada clase.

* Clase Evaluar:
  + Se trata de la clase padre donde actúa como clase abstracta y define una interfaz que obliga a utilizar la clase hija que lo heredera en este caso es **nota** que sirve para poder sacar al final la nota del alumno y definir si éste promociona o no.

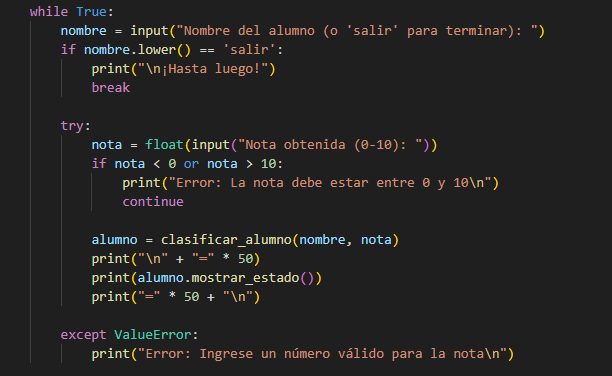


* Clase Persona:
  + Se trata de una clase padre donde en este se utiliza para recopilar la información del alumno para introducir el nombre del mismo.
* Clase Estudiante:
  + Es una clase hija que hereda de **Evaluar y Persona** mediante herencia múltiple. Hereda el nombre del alumno y está obligada a implementar el método **nota**, definido como abstracto en la clase **Evaluar**. En esta clase se inicializan dos notas (*nota\_1* y *nota\_2*), se calcula un promedio en el método **resultado**, y finalmente en **nota** se evalúa si el alumno promociona, aprueba o recursa.



# 4. **RESULTADOS Y DIFICULTADES**

El primer enfoque que implementamos **no producía los resultados esperados**, ya que el código no imprimía correctamente los datos requeridos. Esto nos obligó a introducir un bucle **while** como solución temporal para lograr que el programa funcionase, aunque añadió complejidad innecesaria.



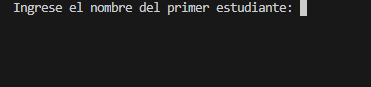
Sin embargo, al analizar el problema con más profundidad, tomamos la decisión clave de replantear la estructura del código por completo. En lugar de depender de parches como el **while**, optamos por:

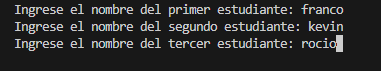
1. Rediseñar la lógica del programa para eliminar dependencias redundantes.
2. Simplificar el flujo de datos, asegurando que el código generará salidas sin necesidad de bucles externos.
3. Priorizar claridad sobre conveniencia, incluso si eso implicaba rehacer partes del trabajo inicial.

Este proceso nos enseñó que las soluciones rápidas no siempre son las óptimas, y que un problema de implementación (como la falta de impresión de resultados) suele ser síntoma de un diseño imperfecto. La versión final no solo resolvió el error original, sino que resultó más eficiente, legible y fácil de escalar.

**RESULTADO**

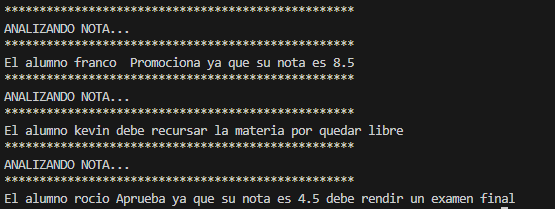
\*Primero la consola pide el nombre de los 3 estudiantes para poder identificarlos:





\*Segundo la consola va pidiendo el resultado de la primera de nota del primer alumno



\*Tercero pide el resultado de la segunda nota del primer alumno  
  
  
\*Cuarto repite el mismo proceso para cada alumno restante  
  
\*Quinto aparece el estado de cada alumno y como quedo al final del cuatrimestre  


# 5. FUTURAS MEJORAS

Si bien el sistema desarrollado cumple con los objetivos planteados, durante el proceso de implementación surgieron diversas ideas que podrían incorporarse en futuras versiones para mejorar su funcionalidad, flexibilidad y experiencia de uso. Carga dinámica de estudiantes: Actualmente el sistema está preparado para evaluar una cantidad fija de estudiantes. Una posible mejora sería permitir al usuario ingresar la cantidad deseada de alumnos, gestionando el ingreso de datos mediante estructuras iterativas que adapten el proceso automáticamente. Promedios con más notas: El sistema podría extenderse para admitir una cantidad variable de notas por alumno, ajustando el cálculo del promedio según los datos ingresados. Esta modificación permitiría una evaluación académica más detallada y realista. Mejor presentación de resultados: Se podrían aplicar mejoras en la forma en que se muestran los resultados por pantalla, incorporando divisores, encabezados o incluso resaltando información clave para facilitar la lectura y comprensión de los datos. Validación de entradas: Es posible incorporar controles que verifiquen que las notas ingresadas sean valores numéricos válidos dentro del rango establecido. Esto aportaría mayor robustez al sistema y evitaría errores en el tiempo de ejecución. Estructura de menú interactivo: Se puede mejorar la interacción con el usuario mediante un menú inicial que ofrezca opciones como ingresar estudiantes, calcular resultados o salir del programa. Esto le daría al sistema una estructura más organizada y cercana a la de una aplicación real. Estas posibles ampliaciones permitirían no solo optimizar el funcionamiento del sistema, sino también profundizar en el uso de estructuras y conceptos clave de la programación, aplicando buenas prácticas que favorecen la mantenibilidad y escalabilidad del código.

# 6. **CONCLUSIÓN FINAL**

A lo largo de la cursada, profundizamos en los conceptos fundamentales de la Programación Orientada a Objetos (POO), aplicando sus pilares básicos en el desarrollo de nuestro proyecto. También exploramos el uso de decoradores lo cual nos permitió escribir código más flexible y reutilizable.

Más allá de lo técnico, aprendimos la importancia del trabajo en equipo, la organización y la división de tareas, simulando un entorno de desarrollo colaborativo. En conclusión, este proyecto no solo fortaleció nuestros conocimientos de Python y POO, sino que también nos brindó una experiencia integral que combina habilidades técnicas y de colaboración, fundamentales para cualquier entorno profesional.