

**Fundamentos de Python**

Unidad 4. PROGRAMACION ORIENTADA A OBJETOS

PROJECTOS

1. Desarrollo de un script en Python

ESTUDIANTE

Armando Gabriel Jacinto López

INSTITUCION

INFOTEC

FECHA

24/09/2025

**1.- Introducción**

El siguiente reporte detalla un programa de calculadora interactiva desarrollado en Python, siguiendo los principios de la **Programación Orientada a Objetos (POO)**. La aplicación está estructurada en dos módulos separados, calculadora\_POO.py y calculator.py, para mejorar la modularidad y la organización del código. Este diseño permite que la lógica principal de la aplicación esté separada de la funcionalidad de la calculadora, facilitando su mantenimiento y escalabilidad.

**2.- Desarrollo**

El programa se compone de dos archivos principales, cada uno con una función específica. A continuación, se detalla la descripción y el propósito de cada parte del código.

**2.1.- Calculator.py**

Este archivo define la clase central del programa, la clase Calculator. Esta clase encapsula toda la lógica de cálculo y la gestión del historial de operaciones.

* **Método \_\_init\_\_**: Este es el constructor de la clase. Inicializa las variables privadas (denotadas con un guion bajo, \_) que almacenan el estado de la calculadora. Estas variables son \_numbers (lista de números en la operación actual), \_operators (lista de operadores), \_records (historial de todas las operaciones completadas) y \_total (el resultado acumulado de la operación en curso).
* **Métodos de Operación (sum, sub, multiply, divide, power)**: Cada uno de estos métodos realiza una operación matemática específica. Toman un valor como argumento, actualizan el \_total y, lo más importante, llaman al método register\_calculation para registrar tanto el operador como el valor en las listas correspondientes (\_operators y \_numbers).
* **Método register\_calculation**: Este método privado se encarga de añadir los números y operadores de una operación a las listas internas de la clase.
* **Método assign\_value**: Este método es crucial para iniciar una nueva operación. Recibe el primer número y lo asigna tanto al \_total como a la lista \_numbers, preparando la calculadora para la siguiente operación.
* **Método register\_operation**: Este método es responsable de formatear la operación completa (por ejemplo, "5 + 3 - 2 = 6") y añadirla al historial (\_records).
* **Método look\_records**: Muestra el historial de operaciones almacenado en \_records, imprimiendo cada operación en una nueva línea.
* **Método get\_total**: Finaliza la operación actual, llama a register\_operation para guardar el cálculo en el historial y devuelve el resultado final (\_total).
* **Método clear**: Reinicia las listas y el total a sus valores iniciales, preparando la calculadora para un nuevo cálculo sin perder el historial.

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 1 Definición de la clase Calculator

**Texto

El contenido generado por IA puede ser incorrecto.**

Ilustración 2 Métodos de la clase Calculator

**Texto

El contenido generado por IA puede ser incorrecto.**

Ilustración 3 Métodos para obtener el total y limpiar los datos

**2.2.- Calculadora\_POO.py**

Este es el programa principal que interactúa con el usuario y utiliza la clase Calculator del otro archivo.

* **Importación**: La primera línea del código, from calculator import Calculator, importa la clase Calculator del archivo calculator.py, demostrando la **modularidad** del programa.
* **Funciones de Validación (validate\_number, validate\_operation)**: Estas dos funciones son esenciales para la robustez del programa.
  + validate\_number: Usa un bloque try-except para convertir la entrada del usuario a un número flotante, manejando errores si la entrada no es un valor numérico.
  + validate\_operation: Verifica si la entrada del usuario es uno de los operadores válidos (+, -, \*, /, ^, result).
* **Función main**: Esta es la función principal que controla el flujo del programa.
  + **Bucle while True**: Mantiene la aplicación en funcionamiento hasta que el usuario decide salir.
  + **Entrada del Usuario**: El programa pide al usuario que elija entre "exit", "records" u "operation".
  + **Manejo de Opciones**: Utiliza sentencias if-elif-else para dirigir el flujo del programa según la entrada del usuario.
    - Si la opción es 'exit', el bucle se rompe y el programa termina.
    - Si la opción es 'records', se llama al método look\_records de la instancia de la calculadora.
    - Si la opción es 'operation', se inicia un segundo bucle while True para gestionar los cálculos.
  + **Bucle de Operaciones**: Dentro de la opción 'operation', este bucle permite al usuario realizar múltiples operaciones con un resultado acumulado hasta que ingresa 'result'. Valida cada número y operación antes de llamar al método correspondiente de la clase Calculator.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

Ilustración 4 Funciones de validación

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 5 Función main()

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 6 Opciones principales

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 7 Ciclo para ejecutar las operaciones

**Texto

El contenido generado por IA puede ser incorrecto.**

Ilustración 8 Selección de operaciones

****

Ilustración 9 Ejecución de la función main()

**3.- Conclusión**

El programa de calculadora es un excelente ejemplo de la aplicación de la **Programación Orientada a Objetos**. La división del código en dos módulos (calculator.py para la lógica y calculadora\_POO.py para la interfaz de usuario) demuestra un diseño limpio y mantenible.

La clase Calculator encapsula de manera efectiva la funcionalidad de cálculo, manteniendo el estado de la operación y el historial. Las validaciones de entrada en el programa principal aseguran que la aplicación sea robusta y capaz de manejar errores del usuario de manera elegante. La adición de la funcionalidad de potencia (^) y el registro de operaciones en el historial demuestran la flexibilidad y la capacidad de expansión del código, confirmando un diseño bien pensado y modular.