4A. CALCULADORA BÁSICA CON POO

Unidad 4. Programación Orientada a Objetos

Datos

Nombre: Keith Alexis Gutiérrez Ibarra
Docente: Silvia Guadalupe Victorio Aguilar
Fecha de elaboración: 28/09/2025

Introducción

En este trabajo se desarrollarán un proyecto basado en la Programación Orientada a Objetos que consistirá en una calculadora básica en Python, que nos permitirá hacer uso de los principios de la POO mediante la creación de clases y objetos sin olvidar el uso de estructura de datos y de control, funciones y algoritmos, la calculadora. La calculará contará con el uso de cuatro operaciones aritméticas fundamentales: suma. Resta, multiplicación y división, además debe leer las expresiones en formato de texto e interpretarlas correctamente, mostrar el historial de operación, debe ejecutarse en un bucle continuo hasta que el usuario se lo indique y por último el programa de manejar una interacción con el usuario con mensajes claros.

El propósito de este ejercicio es mejorar la habilidad de construcción de software con el modelo POO, haciéndolo modular, reutilizable y escalable, asegurando que la calculadora funcione correctamente y realice las operaciones de manera eficiente.

Desarrollo

Proyecto: Calculadora básica

a) Análisis del problema: Para el proyecto se nos piden las siguientes condiciones que debe tener la calculadora

- Debe manejar las cuatro operaciones aritméticas fundamentales: suma, resta, multiplicación y división.
- El sistema debe leer las expresiones matemáticas en formato de texto, interpretar correctamente los números y operadores, validar que los datos ingresados sean correctos, ejecutar el cálculo correspondiente y mostrar el resultado inmediatamente al usuario.
- Debe incluir funcionalidades de historial, que permitan al usuario visualizar todas las operaciones realizadas durante la sesión.
- Debe ejecutarse en un bucle continuo, hasta que el usuario decida salir, mediante un comando especial.
- El programa debe manejar la interacción con el usuario con mensajes claros, procesar comandos especiales y gestionar errores (división entre cero, entradas no válidas, etc.).

b) Diseño del algoritmo

- Se crea una clase llamada Calculadora como base para el programa
    a) Dentro de la clase se crea un constructor que reciba los dos parámetros para inicializar los valores en atributos protegidos y una lista para almacenar operaciones
    b) Se crea una propiedad que permite consultar el primer y segundo número sin modificarlo
    c) Se crea una propiedad que permite asignar un nuevo valor al atributo del primer y segundo número, verificando la entrada correcta de números enteros o decimales
    d) Se crea un método para cada una de las operaciones de suma, resta, multiplicación y división que guarde los resultados retornándolos
    e) Se crea el método privado para registro en historial que agrega un diccionario a la lista _historial
    f) Creación de un método para visualizar historial que valida que la lista no este vacía y cuando esto se cumple devuelve el historial
- Se crea una función que recibe como parámetros una cadena de expresión e identifica los operadores también verifica si el operador actual se encuentra en la expresión y la divide en partes convirtiendo los elementos a números float, elimina espacios y valida la entrada
- Creación de la función principal que inicializa un objeto de la clase "Calculadora"
    a) Se dan instrucciones para el usuario
    b) Se crea un bucle que permite ingresar al usuario operaciones continuamente con condición para salir de la calculadora, ver el historial, validar la expresión, asigna los valores devueltos por interpretar_expresion y valida el tipo de operación para su posterior resolución.
- Se llama a la función main para iniciar la calculadora

c) Código en Python

calculadora_POO.py

calculadora_POO.py > Calculadora > numero1

```python
# Se crea una clase llamada Calculadora como base para el programa
class Calculadora:
    numero1 = 0
    numero2 = 0
    # Se crea un constructor que reciba los dos parametros para inicializar los valores
    def __init__(self, numero1, numero2):
        # Atributos protegidos para guardar los valores del parametro
        self._numero1 = numero1
        self._numero2 = numero2
        # Lista para almacenar el historial de operaciones
        self._historial = []
    # Propiedad que permite consultar el primer numero sin modificarlo
    @property
    def numero1(self):
        return self._numero1
    # Propiedad que permite asignar un nuevo valor al atributo del primer numero
    @numero1.setter
    def numero1(self, nuevo_numero1):
        # Validacion para asegurar entrada de numeros enteros o decimales
        if type(nuevo_numero1) in (int, float):
            self._numero1 = nuevo_numero1
        # Si no es un numero muestra un error
        else:
            raise ValueError("Debe ser un numero")
    # Propiedad que permite consultar el segundo numero sin modificarlo
    @property
    def numero2(self):
        return self._numero2
    # Propiedad que permite asignar un nuevo valor al atributo del segundo numero
    @numero2.setter
    def numero2(self, nuevo_numero2):
        # Validacion para asegurar entrada de numeros enteros o decimales
        if type(nuevo_numero2) in (int, float):
            self._numero2 = nuevo_numero2
        # Si no es un numero muestra un error
        else:
            raise ValueError("Debe ser un numero")
    # Metodo que realiza la operacion de suma entre atributos, guardando en resultados
    def sumar(self):
        resultado = self._numero1 + self._numero2
        self._registrar_operacion("+", resultado)
        return resultado
    # Metodo que realiza la operacion de resta entre atributos, guardando en resultados
    def restar(self):
        resultado = self._numero1 - self._numero2
        self._registrar_operacion("-", resultado)
        return resultado
    # Metodo que realiza la operacion de multiplicacion entre atributos, guardando en resultados
    def multiplicar(self):
        resultado = self._numero1 * self._numero2
        self._registrar_operacion("*", resultado)
        return resultado
    # Metodo que realiza la operacion de division entre atributos, guardando en resultados
    def dividir(self):
        resultado = self._numero1 / self._numero2
        # Se utiliza el metodo _registrar_operacion pasando como argumento el simbolo y resultado
        self._registrar_operacion("/", resultado)
        return resultado
    # Metodo privado para registro en historial
    def _registrar_operacion(self, operador, resultado):
        # Se agrega un diccionario a la lista _historial
        self._historial.append({
```

```
            "operacion": f"{self._numero1} {operador} {self._numero2}",
            "resultado": resultado
        })
    # Metodo para visualizar historial
    def ver_historial(self):
        # Validacion para asegurar que la lista no esta vacia
        if not self._historial:
            print("No hay operaciones en el historial")
            return
        # Cuando no esta vacia se devuelve el historial
        else:
            print("\n--- Historial de Operaciones ---")
            contador = 1
            for operacion in self._historial:
                print(f"{contador}. {operacion['operacion']} = {operacion['resultado']}")
                contador += 1
#Funcion que recibe como parametros una cadena de expresion e indentifica los operadores
def interpretar_expresion(expresion):
    for operador in ["+", "-", "*", "/"]:
        # Verifica si el operador actual se encuentra en la expresion y la divide en partes
        if operador in expresion:
            partes = expresion.split(operador)
            # Si son dos partes, convierte los elementos a numeros float y elimina espacios
            if len(partes) == 2:
                # El codigo original se rompe en caso de ingresar caracteres por ello se agrega un try/except
                # Que retorna un valor de None para una respuesta
                try:
                    num1 = float(partes[0].strip())
                    num2 = float(partes[1].strip())
                except ValueError:
                    return None
                return num1, num2, operador
# Funcion principal que inicializa un objeto de la clase "Calculadora"
def main():
    calc = Calculadora(0,0)
    # Instrucciones iniciales para el usuario
    print("Calculadora Básica. Escribe 'salir' para terminar o  'historial' para ver operaciones.\n")
    # Bucle que permite ingresar al usuario operaciones continuamente
    while True:
        entrada = input("Ingresa la operacion (ejemplo 5 + 5): ")
        # Condicion para salir de la calculadora
        if entrada.strip().lower() == "salir":
            print("¡Hasta pronto!")
            break
        # Condicion para ver el historial de la calculadora
        if entrada.strip().lower() == "historial":
            calc.ver_historial()
            continue
        # Validacion de la expresion en caso de la existencia de un error
        resultado = interpretar_expresion(entrada)
        if not resultado:
            print("Expresion no valida. Usa el formato: numero operador numero (ej. 5 + 5)\n")
            continue
        # Asignacion de los valores devueltos por interpretar_expresion
        num1, num2, operador = resultado
        calc.numero1 = num1
        calc.numero2 = num2
        # Validada el tipo de operacion a realizar segun la expresion del usuario
        if operador == "+":
            print("Resultado:", calc.sumar())
        elif operador == "-":
            print("Resultado:", calc.restar())
        elif operador == "*":
            print("Resultado:", calc.multiplicar())
        elif operador == "/":
            print("Resultado:", calc.dividir())
# Se llama a la funcion main  para iniciar la calculadora
main()
```

d) Ejecución y pruebas

Ejemplos del funcionamiento del código en diversos escenarios:

- Suma de dos números:

```
Calculadora Básica. Escribe 'salir' para terminar o  'historial' para
ver operaciones.

Ingresa la operacion (ejemplo 5 + 5): 5 + 5
Resultado: 10.0
```

- Resta de dos números:

```
Calculadora Básica. Escribe 'salir' para terminar o  'historial' para ver operaciones
.

Ingresa la operacion (ejemplo 5 + 5): 3-2
Resultado: 1.0
Ingresa la operacion (ejemplo 5 + 5):
```

- Multiplicación de dos números:

```
Calculadora Básica. Escribe 'salir' para terminar o  'historial' para ver operaciones
.

Ingresa la operacion (ejemplo 5 + 5): 2*2
Resultado: 4.0
Ingresa la operacion (ejemplo 5 + 5):
```

- División de dos números:

```
Calculadora Básica. Escribe 'salir' para terminar o  'historial' para ver operaciones
.

Ingresa la operacion (ejemplo 5 + 5): 4 / 7
Resultado: 0.5714285714285714
Ingresa la operacion (ejemplo 5 + 5):
```

- Visualizar el historial de operaciones:

```
Calculadora Básica. Escribe 'salir' para terminar o  'historial' para ver operaciones
.

Ingresa la operacion (ejemplo 5 + 5): 4 / 7
Resultado: 0.5714285714285714
Ingresa la operacion (ejemplo 5 + 5): 2 + 2
Resultado: 4.0
Ingresa la operacion (ejemplo 5 + 5): 4 - 1
Resultado: 3.0
Ingresa la operacion (ejemplo 5 + 5): historial

--- Historial de Operaciones ---
1. 4.0 / 7.0 = 0.5714285714285714
2. 2.0 + 2.0 = 4.0
3. 4.0 - 1.0 = 3.0
Ingresa la operacion (ejemplo 5 + 5):
```

- Expresión no valida:

```
Ingresa la operacion (ejemplo 5 + 5): -2 + 1
Resultado: -1.0
Ingresa la operacion (ejemplo 5 + 5): 2 - -1
Expresion no valida. Usa el formato: numero operador numero (ej. 5 + 5)

Ingresa la operacion (ejemplo 5 + 5):
```

- Uso de símbolos o caracteres:

```
Calculadora Básica. Escribe 'salir' para terminar o  'historial' para ver operaciones
.

Ingresa la operacion (ejemplo 5 + 5): Yes - 2
Expresion no valida. Usa el formato: numero operador numero (ej. 5 + 5)
```

- Salir del programa:

```
Calculadora Básica. Escribe 'salir' para terminar o  'historial' para ver operaciones
.

Ingresa la operacion (ejemplo 5 + 5): 5+5
Resultado: 10.0
Ingresa la operacion (ejemplo 5 + 5): salir
¡Hasta pronto!
PS C:\Users\keith\OneDrive\Documentos\Pyrhon\Proyectos>
```

Cuestionario

- **¿Debe la calculadora permitir operaciones inválidas, como la división por cero?**
  R = No, las calculadoras no deben permitir operaciones invalidas como la que se menciona, porque al no tener un valor definitivo se pueden generar errores que afectan los resultados y por lo tanto se deben manejar con mensajes de error o finalizando la operación.

- **¿Dónde está el límite sobre lo que debe hacer la calculadora de forma automática?**
  R= El limite depende de las necesidades del usuario, ya que es este el que realiza las operaciones, con lo cual la calculadora no puede asumir resultados ni realizar acciones que el usuario no solicite o requiera.

Conclusiones

En este trabajo se aplicaron los principios de la Programación Orientada a Objetos para desarrollar una calculadora básica en Python, a través del proyecto se consolidó el uso de clases, objetos, propiedades, métodos y estructuras de control, así como la gestión de datos y el historial de operaciones, la calculadora cumple con las indicaciones mencionadas siendo un código modular, reutilizable y escalabre

Este ejercicio permitió fortalecer las habilidades de construcción de software demostrando cómo la POO facilita el desarrollo de programas más organizados, fáciles de mantener y capaces de manejar operaciones complejas de manera segura y controlada con un enfoque que busca modelar con conceptos del mundo real.