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.

```python
# Clase Calculator
class Calculator:
    # Definir el metodo constructor
    def __init__(self):
        # Definir los valores privados
        self._numbers = []
        self._operators = []
        self._records = []
        self._total = 0
    # Asignar el primer valor, y agregarlo a la lista de numeros
    def assign_value(self, first_value):
        self._total = first_value
        self._numbers.append(first_value)
    # Metodo para registrar los numeros, y operaciones
    def register_calculation(self, operation, value):
        self._numbers.append(value)
        self._operators.append(operation)
    # Metodo para registrar las operaciones en el historial
    def register_operation(self):
        if len(self._operators) == 0:
            print('There are not operations')
            return
        operation = ''
        total_numbers = len(self._numbers)
        for i in range(total_numbers):
            if i == total_numbers - 1:
                operation += f' {self._numbers[total_numbers - 1]} = {self._total}'
            else:
                operation += f'{self._numbers[i]} {self._operators[i]} '
        # Agregar la operacion al historial
        self._records.append(operation)
```

Ilustración 1 Definición de la clase Calculator

```python
# Metodo para mostrar los valores del historial al usuario
def look_records(self):
    print('these are your operations')
    for operation in self._records:
        print(operation)
# Metodo para ejecutar la suma, y llamar al metodo 'register_calculation'
def sum(self, value):
    self._total += value
    self.register_calculation('+', value)
    return self
# Metodo para ejecutar la resta, y llamar al metodo 'register_calculation'
def sub(self, value):
    self._total -= value
    self.register_calculation('-', value)
    return self
# Metodo para ejecutar la multiplicacion, y llamar al metodo 'register_calculation'
def multiply(self, value):
    self._total *= value
    self.register_calculation('*', value)
    return self
# Metodo para ejecutar la division, y llamar al metodo 'register_calculation'
def divide(self, value):
    if value == 0:
        raise ValueError('Cannot divide by zero')
    self._total /= value
    self.register_calculation('/', value)
    return self
```

Ilustración 2 Métodos de la clase Calculator

```python
# Metodo para ejecutar la potencia, y llamar al metodo 'register_calculation'
def power(self, value):
    self._total **= value
    self.register_calculation('^', value)
    return self
# Obtener el total de la operacion
def get_total(self):
    self.register_operation()
    return self._total
# Limpiar los valores privados
def clear(self):
    self._total = 0
    self._numbers = []
    self._operators = []
    return self
```

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.

```python
# Funcion para validar los numeros
def validate_number(entry):
    try:
        number = float(entry)
        return number
    except ValueError:
        print('Error. Invalid number')
        return None
# Funcion para validar las operaciones
def validate_operation(entry):
    if entry in ['+', '-', '*', '/', '^', 'result']:
        return entry
    return None
```

Ilustración 4 Funciones de validación

```python
# Clase principal para ejecutar el programa
def main():
    # Mensaje de presentacion
    print('Basic calculator. "Exit" to end the program, "records" to take a look at the previous operations')
    # Crear la instancia 'calc'
    calc = Calculator()
```

Ilustración 5 Función main()

```python
# Bucle while para controlar las ejecuciones del programa
while True:
    # Pedir la operacion a realizar
    entry = input('Insert an option (exit, records, operation):')
    # exit, salir del ciclo while, y terminar el programa
    if entry.strip().lower() == 'exit':
        print('See you later!')
        break
    # records, mostrar el historial
    if entry.strip().lower() == 'records':
        calc.look_records()
        continue
    # operation, realizar una operacion matematica
    if entry.strip().lower() == 'operation':
        # Pedir el primer numero
        num1_entry = input('Number: ')
        # Validar la entrada
        num1 = validate_number(num1_entry)
        if num1 is None:
            print('Invalid Number. Please insert a valid number')
            continue
        # Asignar el primer valor
        calc.assign_value(num1)
        # i, este valor sirve para saber el numero del ciclo
        i = 0
```

Ilustración 6 Opciones principales

```python
# Ciclo para controlar el numero de operaciones
while True:
    # Mensaje para mostrar las opciones
    message = 'Operation(+, -, *, /, ^) or result: '
    # Cambiar el mansaje en el primer ciclo
    if i == 0:
        message = 'Operation(+, -, *, /, ^): '
        i += 1
    # Pedir la operacion
    operation_entry = input(message)
    # Validar la operacion
    operation = validate_operation(operation_entry)
    if not operation:
        print('Invalid operation. Please insert a valid operation')
        continue
    # result, mostrar el resultado de los calculos
    if operation.strip().lower() == 'result':
        print(f'result: {calc.get_total()}\n')
        calc.clear()
        break
    # Pedir el segundo numero
    num2_entry = input('Number: ')
    # Validar las entrada
    num2 = validate_number(num2_entry)
    if not num2:
        print('Invalid Number. Please insert a valid number')
        continue
```

Ilustración 7 Ciclo para ejecutar las operaciones

```
        # Realizar la operacion seleccionada
        if operation == '+':
            calc.sum(num2)
        elif operation == '-':
            calc.sub(num2)
        elif operation == '*':
            calc.multiply(num2)
        elif operation == '/':
            calc.divide(num2)
        elif operation == '^':
            calc.power(num2)
# si la opcion es diferente, pedir ingresar una opcion valida
else:
    print('Invalid operation. Insert a valid one')
    continue
```

Ilustración 8 Selección de operaciones

```
# Ejecucion de la funcion principal
main()
```

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.