4B. DESARROLLO DE UN SCRIPT EN PYTHON

Unidad 4. Programación Orientada a Objetos

Datos
Nombre: Keith Alexis Gutiérrez Ibarra
Docente: Silvia Guadalupe Victorio Aguilar
Fecha de elaboración: 29/09/2025

Introducción

En este trabajo se desarrollará la ampliación del proyecto de la calculadora realizada anteriormente, un proyecto que está basado en la Programación Orientada a Objetos que consiste en una calculadora básica en Python, para ello haremos uso de todos los elementos aprendidos durante el curso como la estructura de control, uso de funciones para facilitar la organización y estructura de datos para almacenar y manipular información de manera eficiente. A la calculará se le agregará la capacidad para realizar operaciones de potencia, además se implementará el uso de una lista para el manejo de varios valores y no solo un par, también debe interpretar estos cambios correctamente sin olvidar de hacer uso del modularidad para dividir en módulos la calculadora y permitir que las distintas clases tengas su propio archivo. Py.

El propósito de este ejercicio es hacer uso de todas las habilidades adquiridas en el curso mejorando un proyecto con los conocimientos vistos a través de este.

Desarrollo

Proyecto: Calculadora básica

a) Análisis del problema: Para el proyecto se nos piden las siguientes condiciones para ampliar la calculadora que se realizo en el proyecto anterior

- Debe soportar operaciones con varios valores, en lugar de que sean solamente un par.
- Reemplazar el sistema de dos valores individuales por una lista flexible, que permita almacenar múltiples valores y realizar operaciones sobre todos ellos.
- Crear un nuevo método llamado “potencia” dentro de la clase Calculadora que eleve el primer número a la potencia del segundo número.
- Modificar la función interpretar_expresion, para que reconozca el operador ^ como indicador de potencia.
- Actualizar el diccionario de operaciones en el programa principal, para incluir esta nueva funcionalidad. Asegúrate de que el sistema registre adecuadamente estas operaciones en el historial con el formato correcto.
- Usar el modularidad para dividir en módulos la calculadora y permitir que las clases tengan su propio archivo .Py.

b) Diseño del algoritmo

- Se crea en un archivo nombrado calculadora.py una clase llamada Calculadora como base para el programa
    a) Dentro de la clase se crea un constructor que reciba la lista para inicializar los valores en una lista protegida y una lista para almacenar operaciones
    b) Se crea una propiedad que permite consultar la lista de números sin modificarlo
    c) Se crea una propiedad que permite asignar un nuevo valor a la lista de números, verificando la entrada correcta de números enteros o decimales
    d) Se crea un método para cada una de las operaciones de suma, resta, multiplicación, división y potencia 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 un archivo llamado main.py que importa la clase Calculadora del archivo calculadora.py
    a) Se crea una función que recibe como lista una cadena de expresión e identifica los operadores también verifica si el operador actual se encuentra en la expresión y convierte los elementos a números float, elimina espacios y valida la entrada del formato correcto
    b) Creación de la función principal que inicializa un objeto de la clase "Calculadora
    c) Se dan instrucciones para el usuario
    d) 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.
    e) Se llama a la función main para iniciar la calculadora

c) Código en Python

- Calculada.py

calculadora.py × main.py

calculadora.py > Calculadora > lista_numero

```python
# Se crea una clase llamada Calculadora como base para el programa
class Calculadora:
    # Creacion de una lista de numeros
    lista_numero = []
    # Se crea un constructor que reciba la lista para inicializar los valores
    def __init__(self, lista_numero):
        # lista protegida para guardar los valores del parametro
        self._lista_numero = lista_numero
        # Lista para almacenar el historial de operaciones
        self._historial = []
    # Propiedad que permite consultar la lista de numeros sin modificarlos
    @property
    def lista_numero(self):
        return self._lista_numero
    # Propiedad que permite asignar un nuevo valores a la lista de numeros
    @lista_numero.setter
    def lista_numero(self, nueva_lista):
        # Validacion para asegurar entrada de numeros enteros o decimales en toda la lista
        if not all(isinstance(n, (int, float)) for n in nueva_lista):
            raise ValueError("Todos los elementos de la lista deben ser números.")

        self._lista_numero = nueva_lista
    # Metodo que realiza la operacion de suma entre lista de numeros, guardando en resultados
    def sumar(self):
        resultado = sum(self._lista_numero)
        # Se utiliza el metodo _registrar_operacion pasando como argumento el simbolo y resultado
        self._registrar_operacion("+", resultado)
        return resultado
    # Metodo que realiza la operacion de resta entre lista de numeros, guardando en resultados
    def restar(self):
        resultado = self.lista_numero[0]
        for n in self.lista_numero[1:]:
            resultado -= n
        # Se utiliza el metodo _registrar_operacion pasando como argumento el simbolo y resultado
        self._registrar_operacion("-", resultado)
        return resultado
    # Metodo que realiza la operacion de multiplicacion entre lista de numeros, guardando en resultados
    def multiplicar(self):
        resultado = self.lista_numero[0]
        for n in self.lista_numero[1:]:
            resultado *= n
        # Se utiliza el metodo _registrar_operacion pasando como argumento el simbolo y resultado
        self._registrar_operacion("*", resultado)
        return resultado
    # Metodo que realiza la operacion de division entre lista de numeros, guardando en resultados
    def dividir(self):
        # Se hace uso de try y except para solucionar el error de indeterminacion
        try:
            resultado = self._lista_numero[0]
            for n in self._lista_numero[1:]:
                resultado /= n
            # Se utiliza el metodo _registrar_operacion pasando como argumento el simbolo y resultado
            self._registrar_operacion("/", resultado)
            return resultado
        except ZeroDivisionError:
            print("No se puede dividir por cero.")
    # Metodo que realiza la operacion de potencias entre lista de nuemros, guardando en resultados
    def potencia(self):
        resultado = self.lista_numero[0]
        for n in self.lista_numero[1:]:
            resultado **= n
        # Se utiliza el metodo  registrar operacion pasando como argumento el simbolo y resultado
```

```
63            self._registrar_operacion("^", resultado)
64            return resultado
65        # Metodo privado para registro en historial
66        def _registrar_operacion(self, operador, resultado):
67            # Se convierte los daros de la lista en tipo string  para uso del metodo join
68            lista_str = [str(n) for n in self._lista_numero]
69            # Se agrega un diccionario a la lista _historial
70            self._historial.append({
71                "operacion": f" {operador} ".join(lista_str),
72                "resultado": resultado
73            })
74        # Metodo para visualizar historial
75        def ver_historial(self):
76            # Validacion para asegurar que la lista no esta vacia
77            if not self._historial:
78                print("No hay operaciones en el historial")
79                return
80            # Si no está vacia se imprime el historial
81            else:
82                print("\n--- Historial de Operaciones ---")
83                contador = 1
84                for operacion in self._historial:
85                    print(f"{contador}. {operacion['operacion']} = {operacion['resultado']}")
86                    contador += 1
87
```

- Main.py

calculadora.py | main.py

main.py > ...

```
1   from calculadora import Calculadora
2   #Funcion que recibe como lista una cadena de expresion e indentifica los operadores
3   def interpretar_expresion(expresion):
4       for operador in ["+", "-", "*", "/", "^"]:
5           # Verifica si el operador se encuentra en la expresion
6           if operador in expresion:
7               # Funcion que elimina los operadores de la cadena de expresion
8               partes = expresion.split(operador)
9               # Se agrega un try/excep que retorna un valor de None para una respuesta
10              try:
11                  # Se convierte la expresion en una lista, eliminando los espacios vacios y retorno la lista y operador
12                  lista = [float(p.strip()) for p in partes]
13                  return lista, operador
14              except ValueError:
15                  return None
16  # Funcion principal que inicializa un objeto de la clase "Calculadora"
17  def main():
18      calc = Calculadora([])
19      # Instrucciones iniciales para el usuario
20      print("Calculadora Básica. Escribe 'salir' para terminar o  'historial' para ver operaciones.\n")
21      # Bucle que permite ingresar al usuario operaciones continuamente
22      while True:
23          entrada = input("Ingresa la operacion (ejemplo 5 + 5 + 5): ")
24          # Condicion para salir de la calculadora
25          if entrada.strip().lower() == "salir":
26              print("¡Hasta pronto!")
27              break
28          # Condicion para ver el historial de la calculadora
29          if entrada.strip().lower() == "historial":
30              calc.ver_historial()
31              continue
32          # 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 + 5)\n")
            continue
        # Asignacion de los valores devueltos por interpretar_expresion
        lista, operador = resultado
        calc.lista_numero = lista
        # Validada el tipo de operacion a realizar segun la expresion del usuario
        resultado1 = None
        if operador == "+":
            resultado1 = calc.sumar()
        elif operador == "-":
            resultado1 = calc.restar()
        elif operador == "*":
            resultado1 = calc.multiplicar()
        elif operador == "/":
            resultado1 = calc.dividir()
        elif operador == "^":
            resultado1 = calc.potencia()
        # Valida que la operacion se realizo correctamente y no mando ningun error
        if resultado1 is not None:
            print("Resultado:", resultado1)
# 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 tres números:

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

Ingresa la operacion (ejemplo 5 + 5 + 5): 2 + 2 + 2
Resultado: 6.0
Ingresa la operacion (ejemplo 5 + 5 + 5):
```

- Resta de cuatro números:

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

Ingresa la operacion (ejemplo 5 + 5 + 5): 7 - 3 - 1
Resultado: 3.0
Ingresa la operacion (ejemplo 5 + 5 + 5):
```

- Multiplicación de tres números:

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

Ingresa la operacion (ejemplo 5 + 5 + 5): 2 * 4 * 3
Resultado: 24.0
Ingresa la operacion (ejemplo 5 + 5 + 5):
```

- División de tres números:

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

Ingresa la operacion (ejemplo 5 + 5 + 5): 2 / 2 / 2
Resultado: 0.5
Ingresa la operacion (ejemplo 5 + 5 + 5):
```

- Potencia de tres números:

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

Ingresa la operacion (ejemplo 5 + 5 + 5): 2 ^ 2 ^ 2
Resultado: 16.0
Ingresa la operacion (ejemplo 5 + 5 + 5):
```

- Visualizar el historial de operaciones:

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

Ingresa la operacion (ejemplo 5 + 5 + 5): 2 ^ 2 ^ 2
Resultado: 16.0
Ingresa la operacion (ejemplo 5 + 5 + 5): 2 + 2 + 2
Resultado: 6.0
Ingresa la operacion (ejemplo 5 + 5 + 5): 6 - 3 - 4
Resultado: -1.0
Ingresa la operacion (ejemplo 5 + 5 + 5): 5 / 2
Resultado: 2.5
Ingresa la operacion (ejemplo 5 + 5 + 5): historial

--- Historial de Operaciones ---
1. 2.0 ^ 2.0 ^ 2.0 = 16.0
2. 2.0 + 2.0 + 2.0 = 6.0
3. 6.0 - 3.0 - 4.0 = -1.0
4. 5.0 / 2.0 = 2.5
Ingresa la operacion (ejemplo 5 + 5 + 5):
```

- Expresión no valida:

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

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

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

- Error de indeterminación en división:

```
Ingresa la operacion (ejemplo 5 + 5 + 5): 2/0
No se puede dividir por cero.
Ingresa la operacion (ejemplo 5 + 5 + 5):
```

- Salir del programa:

```
Ingresa la operacion (ejemplo 5 + 5 + 5): 2/0
No se puede dividir por cero.
Ingresa la operacion (ejemplo 5 + 5 + 5): salir
¡Hasta pronto!
PS C:\Users\keith\OneDrive\Documentos\Pyrhon\4B. Proyecto final. Desarrollo de un script en Python>
```

Cuestionario

- **¿Qué fue lo más retador al momento de ampliar la calculadora y cómo lo resolviste?**
  R = Lo más problemático fue implementar la lista en todo el funcionamiento de la calculadora, ya que se pedía el aumentar la cantidad de valores a calcular, esto implico una modificación de todos los métodos y funciones para lograr que funcionaran correctamente, para resolverlo fue necesario buscar información respecto a estos casos particulares para poder aplicarlos con el uso en listas.

- **¿Cómo podrías mejorar esta calculadora en el futuro (nuevas funciones o características)?**
  R = Se podría hacer uso de una interfaz grafica para un mejor diseño para el usuario, borrar ciertos caracteres solamente de la expresión, borrar historial, agregar más operaciones como raíz cuadrada o el uso de números negativos.

- **¿Qué estrategias utilizaste para depurar o corregir errores en tu código?**
  R = Primero se corrigió el código con las modificaciones que requería para posteriormente aplicar pruebas de funcionamiento y según los resultados arrojados en el terminal ir corrigiendo en la marcha.

Conclusiones

En este trabajo se aplicaron todos los conocimientos vistos en el curso logrando que la calculadora se transformara en una herramienta potente y con mas funciones, el uso de listas fue el mayor cambio que le permitió hacer uso de múltiples valores además de implementar el uso de potencias y con la separación de clases por archivo logro que se tuviera un mejor modularidad.

Durante estas modificaciones el mayor reto fue refactorizar el código para la implementación de listas, por lo cual fue necesario una mayor investigación de distintas funciones que rodean a las listas.

Por último este ejercicio permitió fortalecer las habilidades que se obtuvieron a lo largo del curso demostrando de manera practica el uso de la POO, haciendo así un programa modular, reutilizable y escalabre en todos los aspectos.