**AI-Powered Calculator  
Understanding the Problem Statement**  
 **Define the Problem**

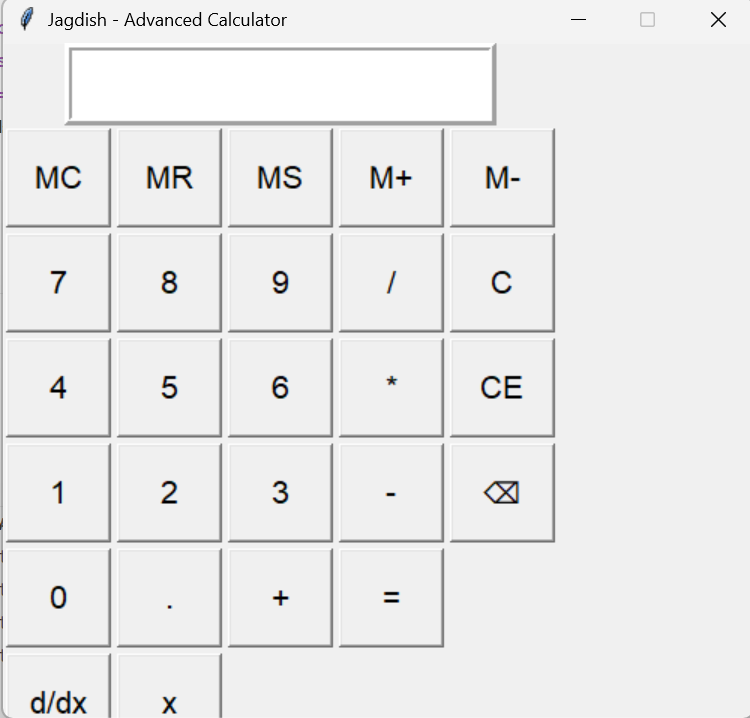
**Mathematical Challenges:**

Algebraic equations of complexity and calculus problems resist manual solution methods.

Need for quick and accurate basic arithmetic and derivative calculations.

**Research Insights**

The program becomes more effective when it includes Python Libraries NumPy and SymPy alongside Tkinter for mathematical operations.

Users will find it easier to access the system due to its friendly Graphical User Interface design.  


**Problem Statement**

Create an AI-based arithmetic and calculus (derivatives) calculator which operates effectively through Python programming.

**Research Insights & Tool Selection**

**Mathematical Concepts & Applications**

**Basic Arithmetic:** Addition (+), subtraction (-), multiplication (\*), division (/).

The system uses equation solving algebraic operations and polynomials management features within mathematical concepts for its operations.

The system uses AI computation that automatically resolves derivative problems by utilizing (d/dx).

**Selected Python Libraries**

**NumPy** – Efficient numerical computations.

**SymPy** – Symbolic mathematics for algebra and calculus.

**Tkinter** – GUI development for user-friendly interaction.  
  
**Outcome (FA-2)**

**Visual Representation of Mathematical Concepts**

The mathematical symbols +, −, ×, ÷, = and d/dx are common features of this process.

**Input example**: x² + 3x - 5

Expected output (derivative): 2x + 3

**Pain Points & Solutions**

Slow manual calculations → Instant AI-powered results

Limited support for symbolic computation → Advanced symbolic algebra with SymPy

**Representation of AI Tools & Libraries**

The Tkinter,sympy and numpy application develops diagrams with human-friendly interfaces which contain fields for user text admission and text display areas.

**SymPy:** Handles differentiation (d/dx) for algebraic equations.

**NumPy:** Optimizes numerical calculations for efficiency.

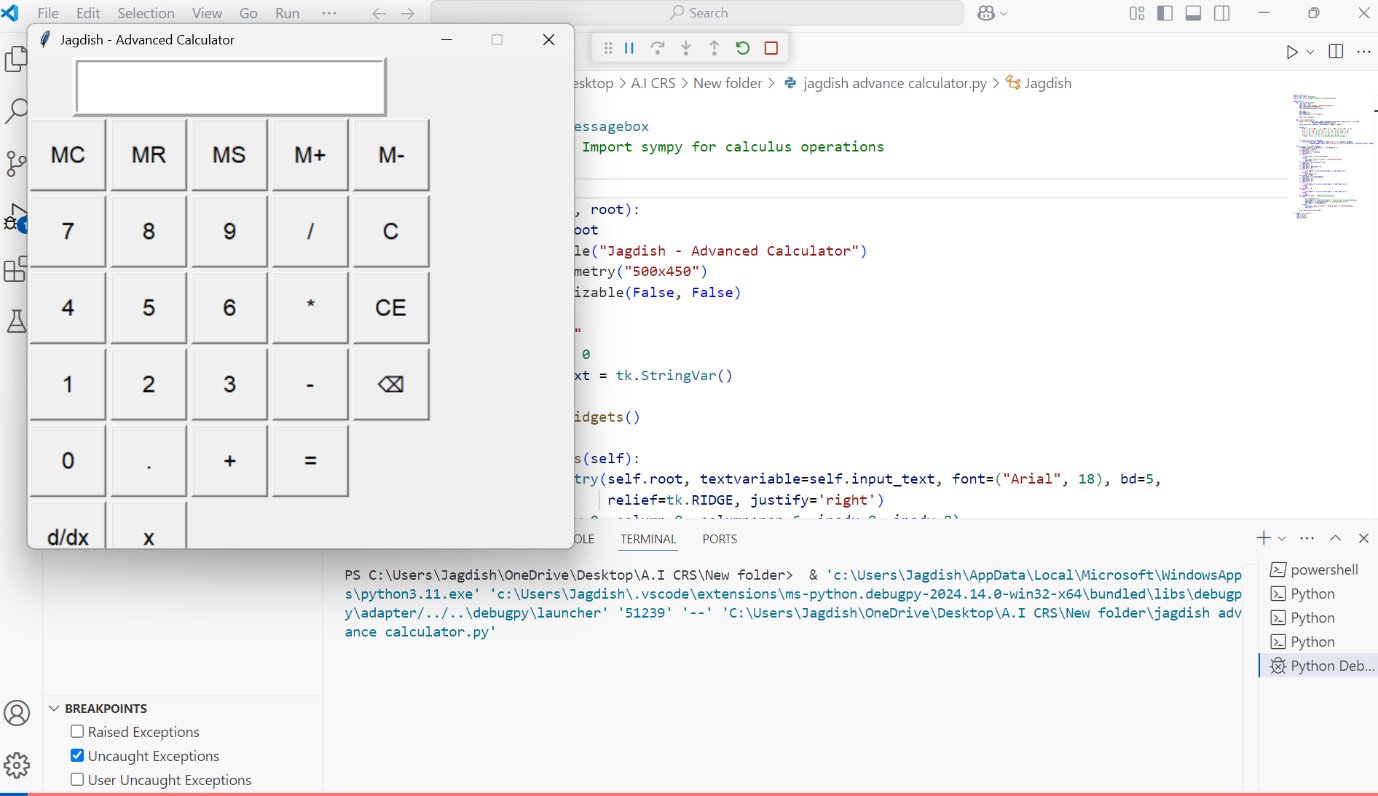
**How It Works**

1. The user provides an input of the mathematical expression x² + 3x - 5 so the system starts its processing phase.

2. Presses the = button for arithmetic calculations or d/dx for derivatives.

3. SymPy.diff() performs calculus functions by accepting processed input and the eval() module enables arithmetic operation management.

4. The computing system presents its calculation results as soon as the program finishes through its graphical user interface.

**Screen shot of working  
**

**Next Steps & Future Enhancements**

* The d/dx operation enables users to perform evaluation of definite and indefinite integrals on the system.
* The key mandatory function in the application lets users see their functions by creating graphical representations.
* Voice input becomes a requirement in the application whenever users want to perform calculations without manual input.

**Calculator programming code**  
import tkinter as tk

from tkinter import messagebox

import sympy as sp  # Import sympy for calculus operations

class Jagdish:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Jagdish - Advanced Calculator")

        self.root.geometry("500x450")

        self.root.resizable(False, False)

        self.expr = ""

        self.memory = 0

        self.input\_text = tk.StringVar()

        self.create\_widgets()

    def create\_widgets(self):

        entry = tk.Entry(self.root, textvariable=self.input\_text, font=("Arial", 18), bd=5,

                         relief=tk.RIDGE, justify='right')

        entry.grid(row=0, column=0, columnspan=6, ipadx=8, ipady=8)

        buttons = [

            ('MC', 1, 0), ('MR', 1, 1), ('MS', 1, 2), ('M+', 1, 3), ('M-', 1, 4),

            ('7', 2, 0), ('8', 2, 1), ('9', 2, 2), ('/', 2, 3), ('C', 2, 4),

            ('4', 3, 0), ('5', 3, 1), ('6', 3, 2), ('\*', 3, 3), ('CE', 3, 4),

            ('1', 4, 0), ('2', 4, 1), ('3', 4, 2), ('-', 4, 3), ('⌫', 4, 4),

            ('0', 5, 0), ('.', 5, 1), ('+', 5, 2), ('=', 5, 3),

            ('d/dx', 6, 0), ('x', 6, 1)  # Derivative button & variable x

        ]

        for (text, row, col) in buttons:

            tk.Button(self.root, text=text, font=("Arial", 16), width=5, height=2,

                      command=lambda t=text: self.on\_button\_click(t)).grid(row=row, column=col, padx=2, pady=2)

    def on\_button\_click(self, button):

        if button.isdigit() or button == '.' or button == 'x':

            self.expr += button

        elif button in '+-\*/':

            self.expr += f' {button} '

        elif button == '=':

            try:

                self.expr = str(eval(self.expr))

            except:

                messagebox.showerror("Error", "Invalid Expression")

                self.expr = ""

        elif button == 'C' or button == 'CE':

            self.expr = ""

        elif button == '⌫':

            self.expr = self.expr[:-1]

        elif button == 'MS':

            try:

                self.memory = float(eval(self.expr)) if self.expr else 0

            except:

                self.memory = 0

        elif button == 'MR':

            self.expr += str(self.memory)

        elif button == 'MC':

            self.memory = 0

        elif button == 'M+':

            try:

                self.memory += float(eval(self.expr)) if self.expr else 0

            except:

                pass

        elif button == 'M-':

            try:

                self.memory -= float(eval(self.expr)) if self.expr else 0

            except:

                pass

        elif button == 'd/dx':  # Derivative Calculation

            try:

                x = sp.symbols('x')

                expr\_sympy = sp.sympify(self.expr)  # Convert input to sympy expression

                derivative = sp.diff(expr\_sympy, x)  # Compute derivative

                self.expr = str(derivative)

            except:

                messagebox.showerror("Error", "Invalid Input for Differentiation")

                self.expr = ""

        self.input\_text.set(self.expr)

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    Jagdish(root)

    root.mainloop()