## ACTIVITY NO. _

| The Scientific Calculator (GUI) | |
|---|---|
| **Course Code:** CPE009B | **Program:** Computer Engineering |
| **Course Title:** Object-Oriented Programming | **Date Performed:** December 2, 2024 |
| **Section:** CPE21S4 | **Date Submitted:** December 2, 2024 |
| **Name:**<br>- ESTEBAN, Prince Wally G.<br>- FERNANDEZ, Don Eleazar T.<br>- SANCHEZ, Christan Ray R.<br>- TANDAYU, Leoj Jeam B.<br>- VALLESER, Kenn L. | **Instructor:** Engr. Maria Rizette Sayo |

### 1. Objective(s)

**Develop a User Friendly Interface Python based Scientific Calculator**:

- Implement advanced mathematical operations using object oriented programming concepts.
- Integrate a PyQt5 based GUI to provide an interactive and visually appealing experience.
- Utilize efficient data structures and algorithms to manage and perform calculations seamlessly.

**Apply Object Oriented Programming Concepts in Development:**

- Class and Objects
- Inheritance
- Encapsulation
- Abstraction
- Polymorphism

**Implement Comprehensive Mathematical Functions**:

- Basic Operations: Addition, Subtraction, Multiplication, and Division
- Exponential and Logarithmic Function
- Trigonometric Function
- Permutation and Combination
- Factorial
- Derivative and Integral
- Geometry Function
- Kinematics and Physics Function
- Complex Function
- Numeral System Function

- History

**Aid Engineering Students**:

- Provide a tool tailored to support students, especially freshmen, in tackling complex math-related subjects such as physics, calculus, and geometry.

## 2. Intended Learning Outcomes (ILOs)

After this activity, the student should be able to:

- To develop a Python based scientific calculator program that can perform essential mathematical operations.
- To apply polymorphism by designing functions that allow different types of calculation to be handled in a uniform way.
- To create a console based interface that provides a user-friendly experience and responds dynamically to user input through event handling.
- To create a GUI using the functions of PyQt5, ensuring that the user has a full and interactive experience with the scientific calculator.
- To create a scientific calculator with GUI of PyQt5 with the functions of:
    - Basic Operations, such as addition, subtraction, division, and multiplication
    - Exponential Function
    - Logarithmic Function
    - Trigonometric Function
    - Permutation
    - Combination
    - Factorial
    - Derivative
    - Integration
    - Geometry
    - Kinematics and Physics Function
    - Complex Function
    - Numeral System Function
    - History

## 3. Discussion

In the modern age, mathematical expressions have become increasingly complex, which often require a tool to solve them. The use of calculators has extended from academic purposes to numerous industries across society.

Modern calculators are descendants of a digital arithmetic machine devised by Blaise Pascal in 1642. Later in the 17th century, Gottfried Wilhelm Leibniz created a more-advanced machine, and, especially in the late 19th century, inventors produced calculating machines that were smaller and smaller and less and less laborious to use (Augustyn, n.d.). The scientific calculator first appeared in the 1970s. They quickly made their way into schools in many countries, especially for students learning mathematics. The early calculators were called 'scientific' presumably because they met the immediate and everyday computational needs of scientists (and engineers); the name is still used today (Kissane, 2016).

Beyond mathematics, a calculator is an aid that facilitates the learning of mathematics and also facilitates the learning of other related subjects like Physics, Chemistry, Geography, and others that use mathematics. These subjects involve a lot of complex computations that can easily be solved using the calculator. Hence, the calculator facilitates learning in different fields (Nabie and Yidana, 2004).

The implementation of the scientific calculator ensure modularity and maintainability through the utilization of object oriented programming concepts, which include:

- Class And Objects - A class is a code template for creating objects. An object is created using the constructor of the class (Bhattacharjee, n.d.).
- Inheritance - The inheritance allows us to define a class that inherits all the methods and properties from another class (*Python Inheritance*, n.d.).
- Encapsulation - The encapsulation describes the idea of wrapping data and the methods that work on data within one unit (Jain, 2024).
- Abstraction - The abstraction is a fundamental concept in Python programming that allows us to simplify complex concepts and focus on the essential details. It involves hiding unnecessary details and exposing only the relevant information to the users (*Understanding Abstraction in Python: Simplifying Complex Concepts*, 2024).
- Polymorphism - The polymorphism refers to the occurrence of something in multiple forms (Duggal, 2024).

Functional Features of the Scientific Calculator

The scientific calculator provides a comprehensive set of functionalities to meet diverse computational requirements, including:

- Basic Operations, such as addition, subtraction, division, and multiplication
- Exponential Function
- Logarithmic Function
- Trigonometric Function
- Permutation
- Combination
- Factorial

- Derivative
- Integration
- Geometry
- Kinematics and Physics Function - The kinematics is a branch of physics and a subdivision of classical mechanics concerned with the geometrically possible motion of a body or system of bodies without consideration of the forces involved (*Kinematics*, n.d.).
- Complex Function
- Numeral System Function
- History

Testing will begin with unit testing to validate individual features such as arithmetic operations, trigonometric functions, and advanced tasks like differentiation and integration. Integration testing will ensure the GUI, computation logic, and history management work together seamlessly, verifying that inputs are processed correctly and results are accurately displayed and logged. Performance testing will evaluate the calculator's ability to handle complex calculations and large datasets efficiently.

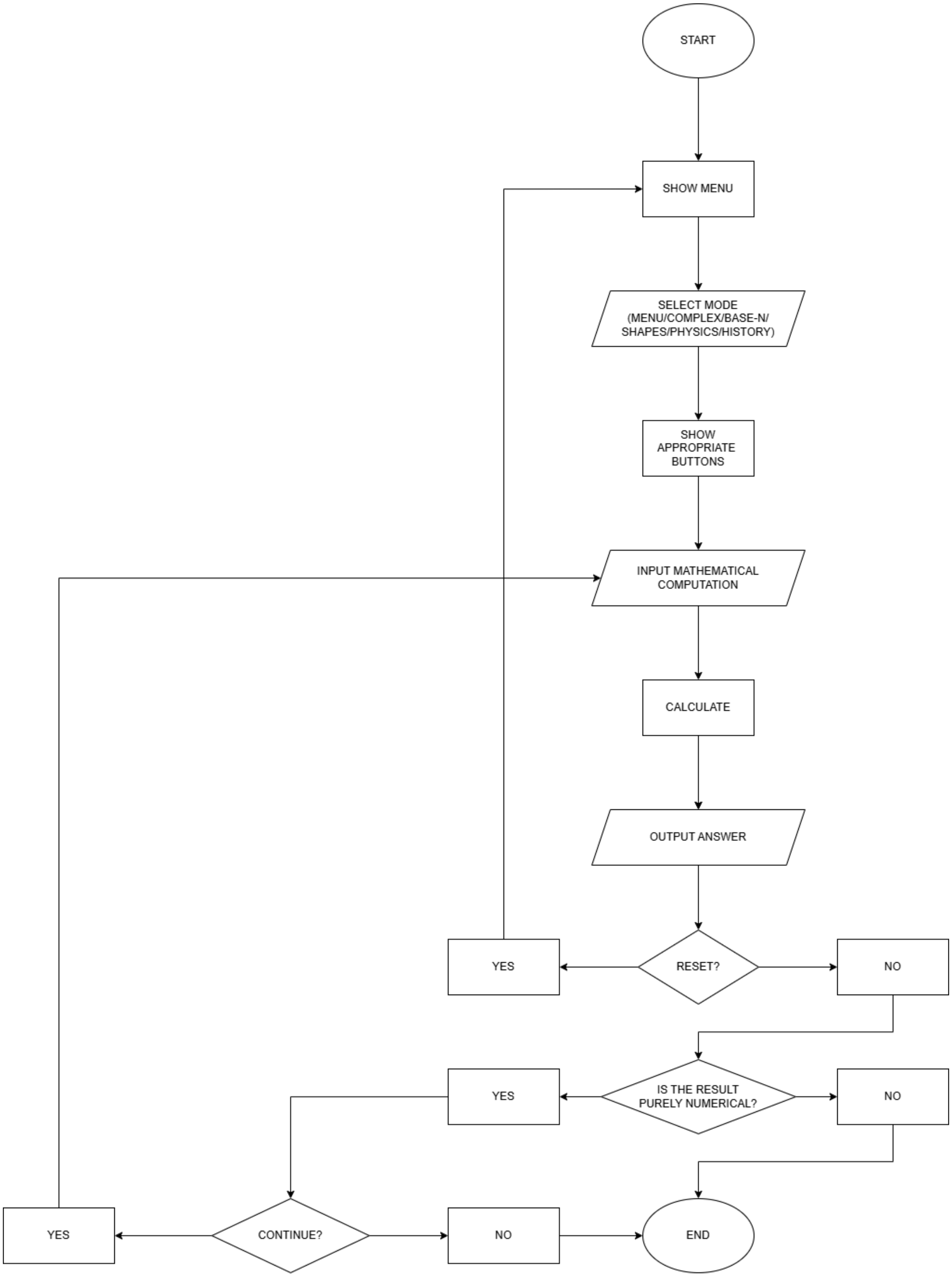
In the evaluation phase, the calculator's success will be measured by operational efficiency and user satisfaction. Efficiency will be assessed through the time complexity of key operations, while user feedback will determine the usability and intuitiveness of the interface. This approach ensures the calculator is both functional and user-friendly.

## 4. Materials and Equipment

- A Desktop Computer with Spyder
- A Laptop with Spyder
- A Windows Operating System

## 5. Procedure

### Flowchart


START
SHOW MENU
SELECT MODE (MENU/COMPLEX/BASE-N/ SHAPES/PHYSICS/HISTORY)
SHOW APPROPRIATE BUTTONS
INPUT MATHEMATICAL COMPUTATION
CALCULATE
OUTPUT ANSWER
YES
RESET?
NO
YES
IS THE RESULT PURELY NUMERICAL?
NO
YES
CONTINUE?
NO
END

## Algorithm

1. Initialize Application
   - Start the PyQt5 application using QApplication.
   - Create an instance of the ScientificCalculator class to initialize the calculator interface.
   - Set up the main application loop using app.exec_().

2. Define ScientificCalculator Class
   - Initialize the class with:
     - Attributes: history (to store past calculations) and memory (to store the last result).
       - Methods:
         - initUI: Sets up the graphical interface.
         - Button, ComplexButtons, BaseNButtons, PhysicsButton, ShapesButton: Creates buttons for each calculator mode.
         - Function, Result, Delete, negativeSign: Handles input and performs operations.
         - History: Manages the history of calculations.
         - ChangePage: Handles page navigation.

3. Set Up UI Components
   - Use QGridLayout for the main layout.
   - Add a stacked widget to support multiple calculator modes:
     - Main calculator.
     - Complex numbers.
     - Base-N conversions.
     - Physics-related calculations.
     - Geometry-related calculations.
     - History.
   - Add the following components:
     - Buttons: For mathematical operations (e.g., square root, trigonometry, logarithms).
     - ComboBox: To navigate between calculator modes.
     - TextLine: For user input and output.
     - History Panel: To display past calculations.

4. Button Functionality
   - Connect Button Clicks:
     - For each button, link its click event to the Function method.
     - Update the input text line with the button's value.
   - Special Buttons:
     - "Del": Removes the last character in the input.
     - "AC": Clears the input.
     - "=": Evaluates the input expression using the Result method.

5. Handle Input and Operations
    - Process Input:
        - Parse the input line to identify mathematical expressions or commands.
    - Perform Calculations:
        - Evaluate basic arithmetic using eval.
        - Handle advanced operations:
            - Trigonometry: sin, cos, tan.
            - Logarithmic: ln, log10.
            - Exponents and Roots: Square root, cube root, powers.
            - Factorial, Permutations, Combinations.
            - Calculus: Derivatives and integrals using SymPy.
            - Number Bases: Convert between binary, octal, decimal, and hexadecimal.
            - Complex Numbers: Magnitude, conjugate, polar form.
            - Physics: Velocity, displacement, force, power, momentum.
            - Geometry: Area and perimeter of shapes.

6. Store and Display History
    - Save each calculation's input and result to a history list.
    - Display the history in the dedicated panel.

7. Page Navigation
    - Use the ChangePage method to switch between modes based on the ComboBox selection.
    - Update the displayed page in the stacked widget.

8. Run the Application
    - Launch the application loop.
    - Continuously listen for button clicks or user input.
    - Close the application when the user exits.

**Pseudocode**

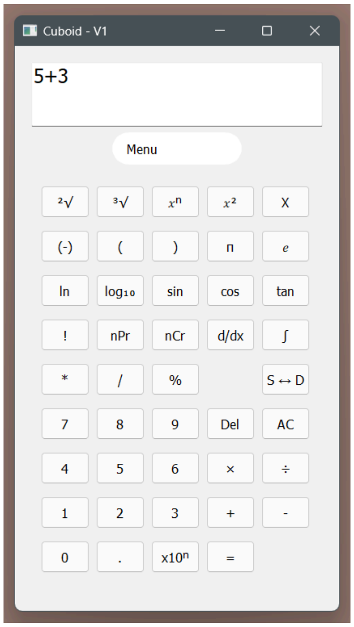
```
Define a class "ScientificCalculator" with:
    - Private attributes:
        - history (list): stores calculation history
        - memory (float): stores the last calculated result
    - Public methods:
        - Constructor: initializes UI and sets up pages
        - Button(grid): sets up calculator buttons and connects them to actions
        - ChangePage(): switches pages based on ComboBox selection
        - Function(): handles button input and updates the text line
        - Result(): evaluates the current expression and displays the result
        - Delete(): removes the last character from the input
        - negativeSign(): toggles negative sign for the current input
        - History(expression, result): saves and displays calculation history

Main function:
    - Initialize PyQt5 application.
    - Create an instance of "ScientificCalculator".
    - Start application loop.
    - On button click or input:
        - Parse input expression.
        - If valid:
            - Perform the calculation and display the result.
            - Save to history.
        - Else:
            - Display "Math ERROR".
    - Allow navigation between pages.
    - Exit the application when the user closes it.
```

## Actual Code

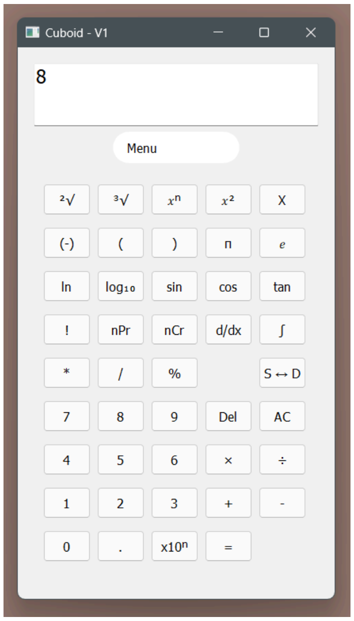
```
import sys
import math
import cmath
import re
import numpy as np
from PyQt5.QtWidgets import QGridLayout, QLineEdit, QPushButton, QWidget, QApplication,
QMessageBox, QStackedWidget, QComboBox, QTextEdit
from PyQt5.QtCore import Qt
from sympy import symbols, diff, sympify, integrate
from fractions import Fraction

class ScientificCalculator(QWidget):
    def __init__(self):
        super().__init__()
        self.history = []
        self.initUI()
        self.memory = None

    def initUI(self):
        mainLayout = QGridLayout(self)
        mainLayout.setContentsMargins(20, 20, 20, 20)

        # The Page
        self.StackedWidget = QStackedWidget(self)
        self.MainPage = QWidget()
        self.ComplexPage = QWidget()
        self.BaseNPage = QWidget()
        self.PhysicsPage = QWidget()
        self.ShapesPage = QWidget()
        self.HistoryPage = QWidget()

        self.MainGrid = QGridLayout()
        self.ComplexGrid = QGridLayout()
        self.BaseNGrid = QGridLayout()
        self.PhysicsGrid = QGridLayout()
        self.ShapesGrid = QGridLayout()
        self.HistoryGrid = QGridLayout()

        self.Button(self.MainGrid)
        self.MainPage.setLayout(self.MainGrid)

        self.ComplexButtons(self.ComplexGrid)
        self.ComplexPage.setLayout(self.ComplexGrid)

        self.BaseNButtons(self.BaseNGrid)
        self.BaseNPage.setLayout(self.BaseNGrid)

        self.PhysicsButton(self.PhysicsGrid)
        self.PhysicsPage.setLayout(self.PhysicsGrid)

        self.ShapesButton(self.ShapesGrid)
        self.ShapesPage.setLayout(self.ShapesGrid)

        self.HistoryPanel = QTextEdit(self)
        self.HistoryPanel.setReadOnly(True)
        self.HistoryPanel.setStyleSheet("font-size: 16px;")
        self.HistoryGrid.addWidget(self.HistoryPanel)
        self.HistoryPage.setLayout(self.HistoryGrid)

        self.StackedWidget.addWidget(self.MainPage)
        self.StackedWidget.addWidget(self.ComplexPage)
        self.StackedWidget.addWidget(self.BaseNPage)
        self.StackedWidget.addWidget(self.PhysicsPage)
        self.StackedWidget.addWidget(self.ShapesPage)
        self.StackedWidget.addWidget(self.HistoryPage)
        # The Combobox
        self.ComboBox = QComboBox(self)
        self.ComboBox.addItem("Menu")
        self.ComboBox.addItem("Complex")
        self.ComboBox.addItem("Base - N")
        self.ComboBox.addItem("Physics")
        self.ComboBox.addItem("Shapes")
        self.ComboBox.addItem("History")
        self.ComboBox.currentIndexChanged.connect(self.ChangePage)
        self.ComboBox.setFixedSize(160, 40)
        self.ComboBox.setStyleSheet("QComboBox {font-size: 16px; border-radius: 20px; padding-left: 18px;} QComboBox::drop-down {border: 0px;width: 0px;}")

        self.textLine = QLineEdit(self)
        self.textLine.setFixedHeight(80)
        self.textLine.setStyleSheet("font-size: 24px;")
        self.textLine.setAlignment(Qt.AlignLeft | Qt.AlignTop)

        mainLayout.addWidget(self.textLine, 0, 0, 1, 5)
        mainLayout.addWidget(self.ComboBox, 1, 2)
        mainLayout.addWidget(self.StackedWidget, 2, 0, 1, 5)


        self.setGeometry(1400, 200, 400, 700)
        self.setWindowTitle('Cubeoid - V1')
        self.setLayout(mainLayout)
        self.show()

    def Button(self, grid):
        functions = [
            ('²√', 2, 0), ('³√', 2, 1), ('𝑥ⁿ', 2, 2), ('𝑥²', 2, 3), ('X', 2, 4),
            ('(-)', 3, 0), ('(', 3, 1), (')', 3, 2), ('π', 3, 3), ('𝑒', 3, 4),
            ('ln', 4, 0), ('log₁₀', 4, 1), ('sin', 4, 2), ('cos', 4, 3), ('tan', 4, 4),
            ('!', 5, 0), ('nPr', 5, 1), ('nCr', 5, 2), ('d/dx', 5, 3), ('∫', 5, 4),
            ('*', 6, 0), ('/', 6, 1), ('%', 6, 2), ('S ↔ D', 6, 4)
        ]

        keypad = [
            '7', '8', '9', 'Del', 'AC',
            '4', '5', '6', '×', '÷',
            '1', '2', '3', '+', '-',
            '0', '.', 'x10ⁿ', '='
        ]

        for name, row, col in functions:
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.clicked.connect(self.Function)
            grid.addWidget(button, row, col)

        keypadPositions = [(i, j) for i in range(7, 11) for j in range(5)]
        for position, name in zip(keypadPositions, keypad):
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.clicked.connect(self.Function)
            grid.addWidget(button, *position)

    def ComplexButtons(self, grid):
        functions = [
            ('²√', 2, 0, ''), ('³√', 2, 1, ''), ('𝑥ⁿ', 2, 2, ''), ('𝑥²', 2, 3, ''), ('*', 2, 4, ''),
            ('(-)', 3, 0, ''), ('(', 3, 1, ''), (')', 3, 2, ''), ('π', 3, 3, ''), ('𝑒', 3, 4, ''),
            ('ln', 4, 0, ''), ('log₁₀', 4, 1, ''), ('sin', 4, 2, ''), ('cos', 4, 3, ''), ('tan', 4, 4, ''),
            ('Arg', 5, 0, 'Input: Arg(a + bi)'), ('Conjg', 5, 1, 'Input: Conjg(a + bi)'), ('∠', 5, 2, 'Input:
R∠θ'), ('i', 5, 3, 'Input: a + bi')
        ]

        keypad = [
            '7', '8', '9', 'Del', 'AC',
            '4', '5', '6', '×', '÷',
            '1', '2', '3', '+', '-',
            '0', '.', '×10ⁿ', '='
        ]

        for name, row, col, tooltip in functions:
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.setToolTip(tooltip)
            button.clicked.connect(self.Function)
            grid.addWidget(button, row, col)

        keypadPositions = [(i, j) for i in range(7, 11) for j in range(5)]
        for position, name in zip(keypadPositions, keypad):
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.clicked.connect(self.Function)
            grid.addWidget(button, *position)

    def BaseNButtons(self, grid):
        functions = [
            ('²√', 2, 0), ('³√', 2, 1), ('𝑥ⁿ', 2, 2), ('𝑥²', 2, 3), ('*', 2, 4),
            ('(-)', 3, 0), ('(', 3, 1), (')', 3, 2), ('π', 3, 3), ('𝑒', 3, 4),
            ('ln', 4, 0), ('log₁₀', 4, 1), ('sin', 4, 2), ('cos', 4, 3), ('tan', 4, 4),
            ("BIN", 5, 0), ("OCT", 5, 1), ("DEC", 5, 2), ("HEX", 5, 3)
        ]

        keypad = [
            '7', '8', '9', 'Del', 'AC',
            '4', '5', '6', '×', '÷',
            '1', '2', '3', '+', '-',
            '0', '.', '×10ⁿ', '='
        ]

        for name, row, col in functions:
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.clicked.connect(self.Function)
            grid.addWidget(button, row, col)

        keypadPositions = [(i, j) for i in range(7, 11) for j in range(5)]
        for position, name in zip(keypadPositions, keypad):
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.clicked.connect(self.Function)
            grid.addWidget(button, *position)

    def PhysicsButton(self, grid):
        functions = [
            ('²√', 2, 0, ''), ('³√', 2, 1, ''), ('𝑥ⁿ', 2, 2, ''), ('𝑥²', 2, 3, ''), ('*', 2, 4, ''),
            ('(-)', 3, 0, ''), ('(', 3, 1, ''), (')', 3, 2, ''), ('π', 3, 3, ''), ('𝑒', 3, 4, ''),
            ('ln', 4, 0, ''), ('log₁₀', 4, 1, ''), ('sin', 4, 2, ''), ('cos', 4, 3, ''), ('tan', 4, 4, ''),
            ('𝑣', 5, 0, 'Final Velocity Input: v,a,t,𝑣'), ('Δ𝑑', 5, 1, 'Displacement Input: v,a,t,Δ𝑑'), ('𝐹', 5, 2,
'Force Input: m,a,𝐹'), ('𝑃', 5, 3, 'Power Input: W,t,𝑃'), ('𝑝', 5, 4, 'Momentum Input: m,v,𝑝'),
            ('Δ𝑈', 6, 0, 'Change in Internal Energy Input: Q,W,Δ𝑈'), ('𝑇(IGL)', 6, 1, 'Temperature in
Ideal Gas Law Input: P,V,n,𝑇(IGL)'), (',', 6, 2, '')
        ]

        keypad = [
            '7', '8', '9', 'Del', 'AC',
            '4', '5', '6', '×', '÷',
            '1', '2', '3', '+', '-',
            '0', '.', '×10ⁿ', '='
        ]

        for name, row, col, tooltip in functions:
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.setToolTip(tooltip)
            button.clicked.connect(self.Function)
            grid.addWidget(button, row, col)

        keypadPositions = [(i, j) for i in range(7, 11) for j in range(5)]
        for position, name in zip(keypadPositions, keypad):
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.clicked.connect(self.Function)
            grid.addWidget(button, *position)

    def ShapesButton(self, grid):
        functions = [
            ('²√', 2, 0, ''), ('³√', 2, 1, ''), ('𝑥ⁿ', 2, 2, ''), ('𝑥²', 2, 3, ''), ('*', 2, 4, ''),
            ('(-)', 3, 0, ''), ('(', 3, 1, ''), (')', 3, 2, ''), ('π', 3, 3, ''), ('𝑒', 3, 4, ''),
            ('ln', 4, 0, ''), ('log₁₀', 4, 1, ''), ('sin', 4, 2, ''), ('cos', 4, 3, ''), ('tan', 4, 4, ''),
            ('◯', 5, 0, 'Circle Input: R◯(A or P)'), ('□', 5, 1, 'Square Input: S□(A or P)'), ('△', 5, 2,
'Triangle Input: B,H△(A or P)'), ('▭', 5, 3, 'Rectangle Input: L,W▭(A or P)'), (',', 5, 4, ''),
            ('A', 6, 0, 'Area'), ('P', 6, 1, 'Perimeter')
        ]

        keypad = [
            '7', '8', '9', 'Del', 'AC',
            '4', '5', '6', '×', '÷',
            '1', '2', '3', '+', '-',
            '0', '.', '×10ⁿ', '='
        ]

        for name, row, col, tooltip in functions:
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.setToolTip(tooltip)
            button.clicked.connect(self.Function)
            grid.addWidget(button, row, col)

        keypadPositions = [(i, j) for i in range(7, 11) for j in range(5)]
        for position, name in zip(keypadPositions, keypad):
            button = QPushButton(name)
            button.setFixedSize(60, 40)
            button.setStyleSheet("font-size: 16px;")
            button.clicked.connect(self.Function)
            grid.addWidget(button, *position)

    def Function(self):
        sender = self.sender()

        Button = sender.text()
        if Button == 'Del':
            self.Delete()
        elif Button == 'AC':
            self.textLine.clear()
        elif Button == '=':
            self.Result()
        elif Button == '(-)':
            self.negativeSign()
        elif Button == 'S ↔ D':
            expression = self.textLine.text().strip()
            if '/' in expression:
                numerator, denominator = map(int, expression.split('/'))
                if denominator == 0:
                    return "Math ERROR"
                result = numerator / denominator
                self.textLine.setText(f"{result:.2f}")
            else:
                decimal = float(expression)
                result = Fraction(decimal).limit_denominator()
                self.textLine.setText(str(result))
        else:
            inputText = self.textLine.text()
            self.textLine.setText(inputText + Button)

    def Delete(self):
        inputText = self.textLine.text()
        if inputText:
            deletedText = inputText[:-1]
            self.textLine.setText(deletedText)

    def negativeSign(self):
        inputText = self.textLine.text()
        if inputText:
            if inputText.startswith('-'):
                negativeText = inputText[1:]
            else:
                negativeText = '-' + inputText
            self.textLine.setText(negativeText)

    def Result(self):
        expression = self.textLine.text()

        # The Function
        if '%' in expression:
            expression = expression.split('%')
            if len(expression) == 2:
                base = eval(expression[0].strip())
                result = (1 / 100) * base
                self.textLine.setText(str(result))
        elif '²√' in expression or '³√' in expression:
            expression = expression.replace('²√', 'np.sqrt')
            expression = expression.replace('³√', 'np.cbrt')
        elif 'xⁿ' in expression or 'x²' in expression:
            expression = expression.replace('xⁿ', '**')
            expression = expression.replace('x²', '**2')
        elif 'ln' in expression or 'log₁₀' in expression:
            expression = expression.replace('ln', 'np.log')
            expression = expression.replace('log₁₀', 'np.log10')
        elif 'sin' in expression or 'cos' in expression or 'tan' in expression:
            expression = expression.replace('sin', 'np.sin(np.radians')
            expression = expression.replace('cos', 'np.cos(np.radians')
            expression = expression.replace('tan', 'np.tan(np.radians')
            expression = expression.replace(')', '))', 1)
        elif '!' in expression:
            parts = expression.split('!')
            expression = parts[0].strip()
            if expression:
                expression = int(expression)
                if expression < 0:
                    self.textLine.setText("Math ERROR: Factorial of negative number is undefined.")
                else:
                    result = math.factorial(expression)
                    self.textLine.setText(f"{result}")
        elif 'π' in expression or 'e' in expression:
            expression = expression.replace('π', str(math.pi))
            expression = expression.replace('e', str(math.e))
        elif 'nPr' in expression:
            n, r = map(int, expression.split('nPr'))
            if n < 0 or r < 0 or n < r:
                self.textLine.setText("Math ERROR")
            else:
                result = math.factorial(n) // math.factorial(n - r)
                self.textLine.setText(f"{result}")
        elif 'nCr' in expression:
            n, r = map(int, expression.split('nCr'))
            if n < 0 or r < 0 or n < r:
                self.textLine.setText("Math ERROR")
            else:
                result = math.factorial(n) // (math.factorial(r) * math.factorial(n - r))
                self.textLine.setText(f"{result}")
        elif 'd/dx' in expression:
            expression = expression.split('d/dx')[1].strip()
            x = symbols('X')
            try:
```

```
            func = sympify(expression)
            derivative = diff(func, x)
            self.textLine.setText(f"{derivative}")
            return
        except:
            self.textLine.setText("Math ERROR")
            return
    elif 'ʃ' in expression:
        if 'ʃ' in expression:
            expression = expression.split('ʃ')[1].strip()
            x = symbols('X')
            expression = sympify(expression)
            result = integrate(expression, x)
            self.textLine.setText(f"{result} + C")
            return
        expression = expression.replace('×', '*').replace('÷', '/')
        result = eval(expression)
        self.textLine.setText(str(result))
    elif '×' in expression or '÷' in expression:
        expression = expression.replace('×', '*').replace('÷', '/')
    elif 'x10ⁿ' in expression:
        expression = re.sub(r'(\d+|\d+\.\d+)\s*x\s*10\ⁿ(\d+)', r'\1 * (10**\2)', expression)
    elif 'BIN' in expression:
        num = int(expression.split("BIN")[1].strip())
        result = bin(num)[2:]
        self.textLine.setText(f"{result}")
    elif 'OCT' in expression:
        num = int(expression.split("OCT")[1].strip())
        result = oct(num)[2:]
        self.textLine.setText(f"{result}")
    elif 'DEC' in expression:
        num = int(expression.split("DEC")[1].strip())
        result = str(num)
        self.textLine.setText(f"{result}")
    elif 'HEX' in expression:
        num = int(expression.split("HEX")[1].strip())
        result = hex(num)[2:].upper()
        self.textLine.setText(f"{result}")
    elif "Arg" in expression:
        expression = re.match(r'Arg\((([-+]?\d*\.?\d+)([-+]\d*\.?\d+)i\)', expression)
        a = float(expression[1])
        b = float(expression[2])
        rad = math.atan2(b, a)
        theta = math.degrees(rad)
        self.textLine.setText(f"{theta:.2f}°")
    elif "Conjg" in expression:
        expression = re.match(r'Conjg\(\s*([-+]?\d*\.?\d+)\s*([-+])\s*(\d*\.?\d+)i\s*\)',
expression)
        a = float(expression[1])
        sign = expression[2]
        b = float(expression[3])
        if sign == '+':
            conjugate = f"{a} - {b}i"
        else:
            conjugate = f"{a} + {b}i"
        self.textLine.setText(f"{conjugate}")
    elif '∠' in expression:
        angle = expression.split('∠')
        r = float(angle[0].strip())
        result = float(angle[1].strip())
        result = r * (math.cos(math.radians(result)) + 1j * math.sin(math.radians(result)))
        self.textLine.setText(f"{result:.4f}")
    elif 'i' in expression:
        expression = expression.replace(' ', ").replace('i', ")
        if '+' in expression:
            expression = expression.split('+')
        elif '-' in expression[1:]:
            expression = expression.split('-', 1)
            expression[1] = '-' + expression[1]
        a = float(expression[0])
        b = float(expression[1])
        r = math.sqrt(a**2 + b**2)
        theta = math.degrees(math.atan2(b, a))
        result = f"{r:.2f}∠{theta:.2f}"
        self.textLine.setText(f"{result}")
    elif 'v' in expression:
        u = float(self.textLine.text().split(',')[0])
        a = float(self.textLine.text().split(',')[1])
        t = float(self.textLine.text().split(',')[2])
        v = u + a * t
        self.textLine.setText(f"{v:.2f} m/s")
    elif 'Δd' in expression:
        u = float(self.textLine.text().split(',')[0])
        a = float(self.textLine.text().split(',')[1])
        t = float(self.textLine.text().split(',')[2])
        s = u * t + 0.5 * a * t ** 2
        self.textLine.setText(f"{s:.2f} m")
    elif 'F' in expression:
        m = float(self.textLine.text().split(',')[0])
        a = float(self.textLine.text().split(',')[1])
        F = m * a
        self.textLine.setText(f"{F:.2f} N")
    elif 'P' in expression:
        W = float(self.textLine.text().split(',')[0])
        t = float(self.textLine.text().split(',')[1])
        P = W / t
        self.textLine.setText(f"{P:.2f} W")
    elif 'p' in expression:
        m = float(self.textLine.text().split(',')[0])
        v = float(self.textLine.text().split(',')[1])
        p = m * v
        self.textLine.setText(f"{p:.2f} kg·m/s")
    elif 'ΔU' in expression:
        Q = float(self.textLine.text().split(',')[0])
        W = float(self.textLine.text().split(',')[1])
        ΔU = Q - W
        self.textLine.setText(f"{ΔU:.2f} J")
    elif 'T(IGL)' in expression:
        P = float(self.textLine.text().split(',')[0])
        V = float(self.textLine.text().split(',')[1])
        n = float(self.textLine.text().split(',')[2])
        R = 8.314
        T = (P * V) / (n * R)
        self.textLine.setText(f"{T:.2f} K")
    elif '◯' in expression:
        parts = expression.split('◯')
        radius = float(parts[0].strip())
        if 'A' in expression:
            area = math.pi * (radius ** 2)
            self.textLine.setText(f"{area:.2f}")
        elif 'P' in expression:
            circumference = 2 * math.pi * radius
            self.textLine.setText(f"{circumference:.2f}")
        return
    elif '□' in expression:
        parts = expression.split('□')
        side = float(parts[0].strip())
        if 'A' in expression:
            area = side ** 2
            self.textLine.setText(f"{area:.2f}")
        elif 'P' in expression:
            perimeter = 4 * side
            self.textLine.setText(f"{perimeter:.2f}")
    elif '▭' in expression:
        if ',' in expression:
            length, rest = expression.split(',')
            width, operation = rest.split('▭')
            length = float(length)
            width = float(width)
        else:
            parts = expression.split('▭')
            length = float(parts[0].strip())
            width = length
        if 'A' in expression:
            area = length * width
            self.textLine.setText(f"{area:.2f}")
        elif 'P' in expression:
            perimeter = 2 * (length + width)
            self.textLine.setText(f"{perimeter:.2f}")
    elif '△' in expression:
        if ',' in expression:
            parts = expression.split('△')
            base_height = parts[0].strip().split(',')
            if len(base_height) == 2:
                base = float(base_height[0])
                height = float(base_height[1])
                if 'A' in expression:
                    area = 0.5 * base * height
                    self.textLine.setText(f"{area:.2f}")
                elif 'P' in expression:
                    hypotenuse = math.sqrt(base**2 + height**2)
                    perimeter = base + height + hypotenuse
                    self.textLine.setText(f"{perimeter:.2f}")
    else:
        self.textLine.setText("Math ERROR")

    result = eval(expression)
    result = round(result, 2)
    self.textLine.setText(str(result))
    self.memory = result
    self.History(expression, result)

def History(self, expression, result):
    expression = f"{expression} = {result}"
    self.history.append(expression)
    self.HistoryPanel.append(expression)

def ChangePage(self):
    Index = self.ComboBox.currentIndex()
    self.StackedWidget.setCurrentIndex(Index)

if __name__ == '__main__':
app = QApplication(sys.argv)
calc = ScientificCalculator()
sys.exit(app.exec_())
```

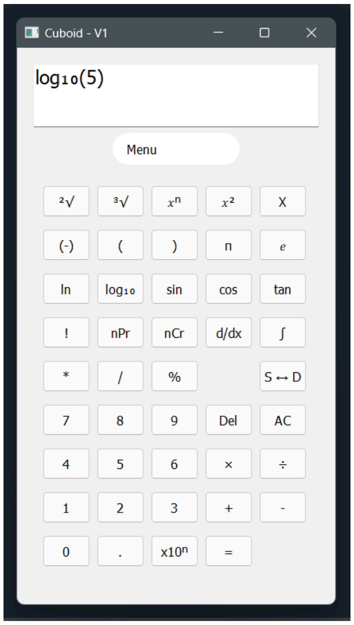
## 6. Output



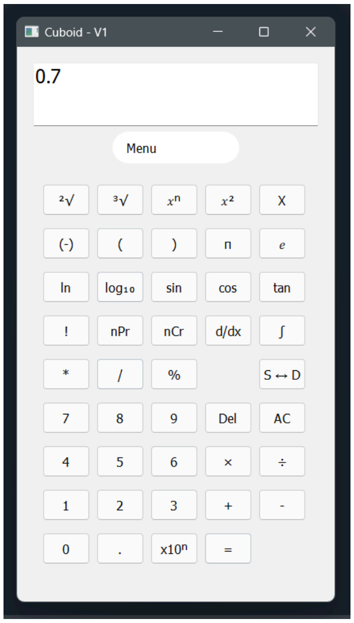
5+3



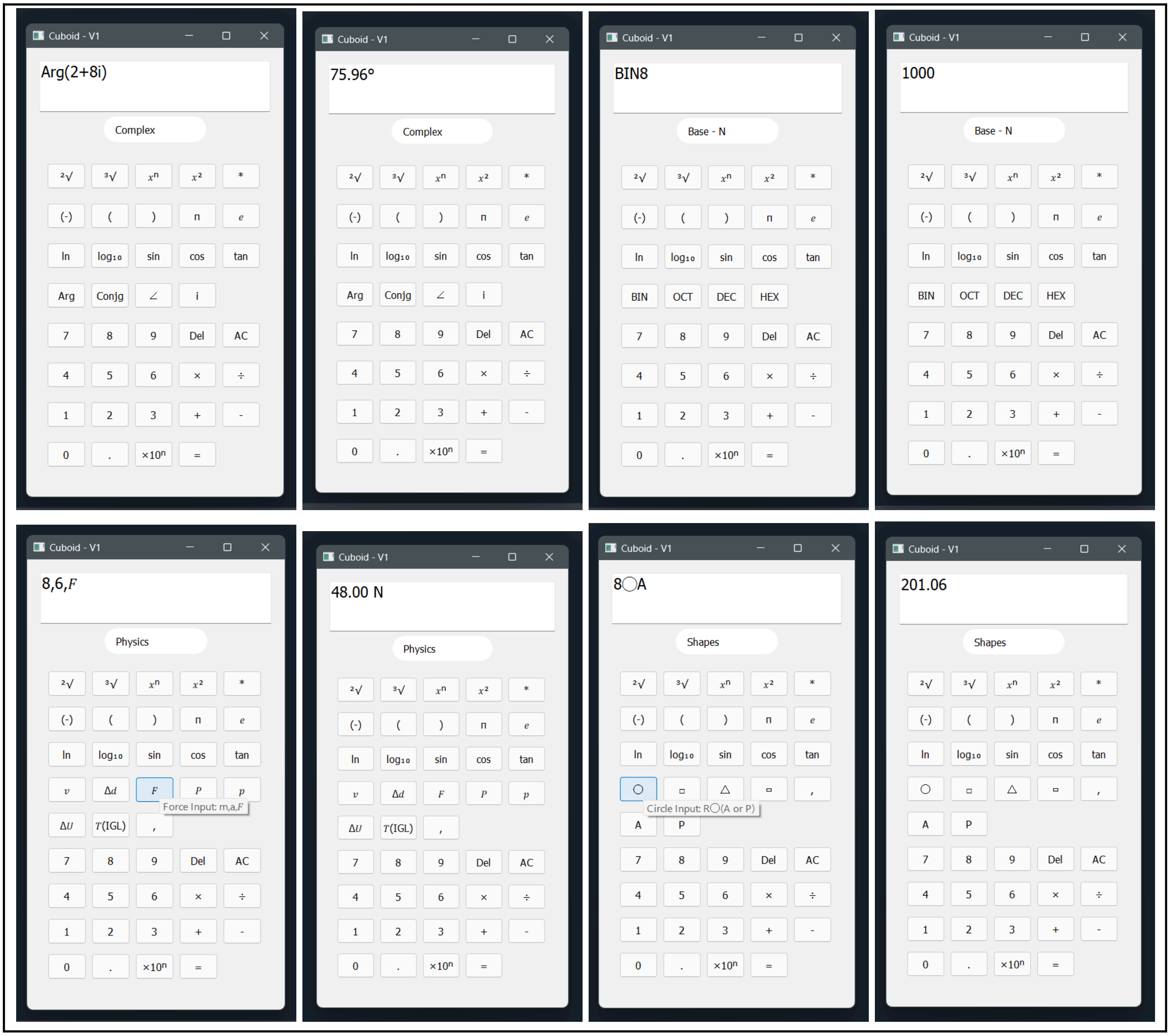
8



log$_{10}$(5)



0.7



Cuboid - V1

Arg(2+8i)

Complex

Cuboid - V1

75.96°

Complex

Cuboid - V1

BIN8

Base - N

Cuboid - V1

1000

Base - N

Cuboid - V1

8,6,$F$

Physics

Force Input: m,a,$F$

Cuboid - V1

48.00 N

Physics

Cuboid - V1

8◯A

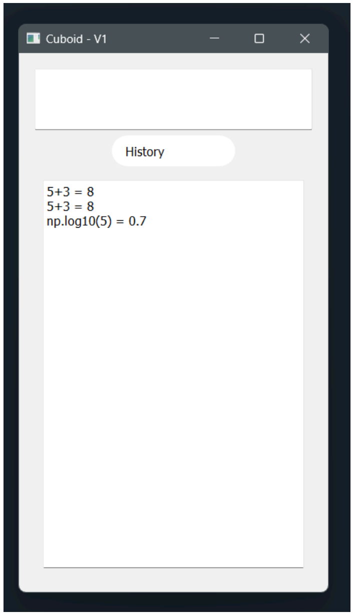
Shapes

Circle Input: R◯(A or P)

Cuboid - V1

201.06

Shapes



```
5+3 = 8
5+3 = 8
np.log10(5) = 0.7
```

## 7. Conclusion

In conclusion, the **Scientific Calculator project** effectively demonstrates the application of object oriented programming concepts, specifically in the development of a modular, user friendly, and versatile computational tool. By highlighting the concepts such as encapsulation, inheritance, abstraction, and polymorphism, the calculator ensures scalability and maintainability in its codebase while providing a smooth user experience.

The project achieved its objectives by integrating a **PyQt5 based graphical user interface** that supports real time interaction and provides a set of features. The intuitive interface allows users to navigate through various functionalities, such as physics based computations and complex number handling, making it highly versatile.

Key functionalities like **basic operations, exponential and logarithmic functions, trigonometric functions, permutation and combination functions, factorials, derivatives, integration, geometry functions, kinematics and other physics functions, complex functions, and numeral system functions** were successfully implemented, meeting the intended learning outcomes of creating an efficient and interactive calculator. Rigorous testing ensured the reliability and accuracy of all operations, making the tool suitable for academic and practical use.

In summary, the Scientific Calculator project highlights the practical implementation of object oriented programming concepts in creating real world software solutions, particularly in supporting engineering students. It not only reinforces the theoretical concepts of object oriented programming but also provides a tool for solving complex mathematical problems, which achieves academic and practical efficiency.

## 8. References

Augustyn, A. (n.d.). *Calculator | Definition, History, Types, & Facts*. Britannica. Retrieved November 11, 2024, from https://www.britannica.com/technology/calculator

Bhattacharjee, J. (n.d.). *Classes and Objects*. HackerEarth. Retrieved December 1, 2024, from https://www.hackerearth.com/practice/python/object-oriented-programming/classes-and-objects-i/tutorial/

Duggal, N. (2024, February 9). *Learn Polymorphism in Python with Examples*. Simplilearn.com. Retrieved December 1, 2024, from https://www.simplilearn.com/polymorphism-in-python-article

Jain, S. (2024, July 1). *Private Methods in Python*. GeeksforGeeks. Retrieved December 1, 2024, from https://www.geeksforgeeks.org/private-methods-in-python/?ref=lbp

*Kinematics*. (n.d.). Britannica. Retrieved December 1, 2024, from https://www.britannica.com/science/kinematics

Kissane, B. (2016). The Scientific Calculator and School Mathematics. *Southeast Asian Mathematics Education Journal*, *6*(1), pp. 29-48.

Nabie, M. J., & Yidana, I. (2004, January 28). The Scientific Calculator as a Tool for Mathematics Teaching and Learning. *Mathematics Connection*, *2*, pp. 34-38.

*Python Inheritance*. (n.d.). W3Schools. Retrieved December 1, 2024, from https://www.w3schools.com/python/python_inheritance.asp

*Understanding Abstraction in Python: Simplifying Complex Concepts*. (2024, February 28). Analytics Vidhya. Retrieved December 1, 2024, from https://www.analyticsvidhya.com/blog/2024/02/understanding-abstraction-in-python-simplifying-complex-concepts/