# **INTRODUCTION**

This portfolio displays three programming tasks which include a **Temperature Converter, Basic Calculator and a Contact Book.** The **Temperature Converter** allows users to input a temperature value, select a conversion type (Fahrenheit to Celsius or vice versa), and visualize the results using a bar chart.

The Objective is to showcase problem solving skills, algorithmic thinking, application of core programming principles and proficiency in software development practices. The tasks Incorporates graphical user interfaces (GUI), Object oriented programming (OOP) using python and data visualization. The portfolio highlights the whole software development lifecycle ranging from problem definition to implementation and testing.

# **2.0 PROBLEM SOLVING TECHNIQUES**

## **2.1 TEMPERATURE CONVERTER**

**Problem Statement**: The following problem involves developing a GUI based temperature converter that accepts User inputs and converts temperature from Celsius to Fahrenheit and vice versa and visualize conversion using a bar chart fully dependent on the user selection of conversion they wish to perform.

**Problem Breakdown:**

1. Create a GUI to accept User Input and allow user to choose conversion Type.
2. Conversion Logic that does the actual conversion depend on the user conversion choice.
3. Display User result on the GUI.
4. Visualize results in the Bar Chart.

**Requirement Analysis:**

**1. Functional Requirements**

Listed below are the core functionalities the system must provide:

1. **User Input**
   * Allow users to input a temperature value.
   * Ensure only numeric values are accepted.
   * Provide dropdown or radio buttons for selecting conversion type (Celsius to Fahrenheit or Fahrenheit to Celsius).
2. **Temperature Conversion**
   * Convert Celsius to Fahrenheit using: F=(C×9/5) +32F
   * Convert Fahrenheit to Celsius using: C=(F−32) ×5/9C
3. **GUI Interaction**
   * A **convert button** to trigger conversion.
   * A **clear button** to reset input fields.
   * Display the converted temperature on-screen.
4. **Graphical Representation**
   * Use a **bar chart** to visually compare the original and converted temperature values.
5. **Error Handling**
   * Prevent invalid inputs (e.g., letters, special characters).
   * Display appropriate error messages for incorrect inputs.

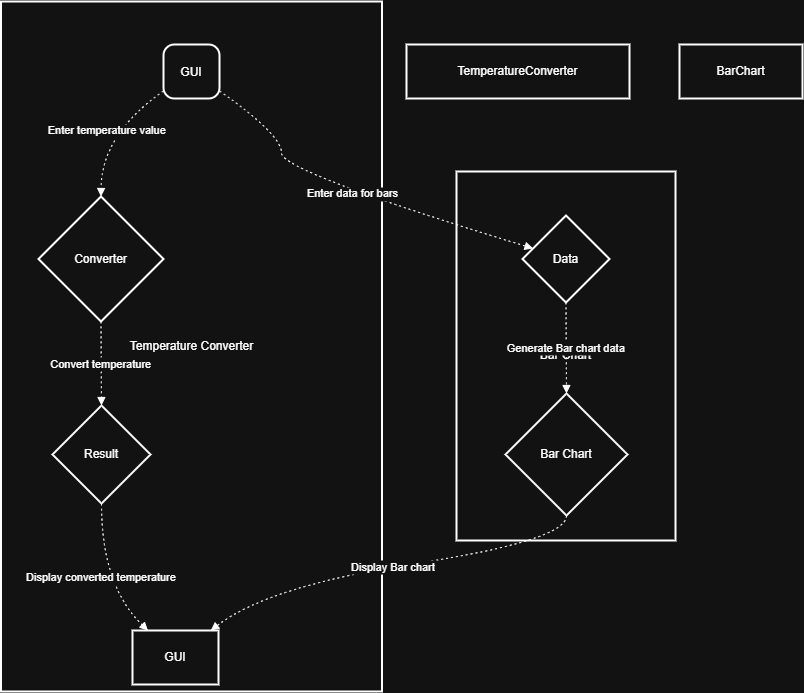
**2. Non-Functional Requirements**

These define the system’s quality attributes:

1. **Usability**
   * Simple and user-friendly graphical interface using **Tkinter**.
   * Clearly labeled buttons and text fields.
2. **Performance**
   * Fast conversion (O (1) time complexity).
   * Responsive graph updates using **Matplotlib**.

The diagram below shows a mind map of the temperature converter, it showcases and highlights the key features of the problem solution and how all these features interact such as:

1. GUI: Asks and accepts user input, also lets users specify the conversion type.
2. The Converter: This specifies the actual logic for the conversion, such as conversion formulae.
3. Result: This displays the answer after the logic has been result
4. The BAR CHART: for visualization of the answers on a bar chart.

****

## **2.2 BASIC CALCULATOR**

**Problem Statement:** The problem below aims to design a graphical user interface- based calculator which allows users to perform various mathematical operations such as addition, multiplication, division and subtraction.

**Problem Breakdown:**

* The program must accept user input through buttons and keyboard support, also provide clear and backspace options for error corrections
* Support basic arithmetic operations such as addition, multiplication, division and subtraction.
* Must show real-time input and results in a text-based display screen.
* A grid-based button layout.
* Display error messages for invalid calculations.
* Use object-oriented programming for structured and scalable code.

**Requirement Analysis:**

**Requirements Analysis for GUI-Based Calculator**

**1. Functional Requirements *(What the system should do)***

✔ **User Input:** Accepts numbers and mathematical operations via buttons and keyboard input.  
✔ **Basic Arithmetic:** Performs addition, subtraction, multiplication, and division.  
✔ **Equation Evaluation:** Processes multi-step expressions following mathematical precedence (PEMDAS).  
✔ **Real-time Display:** Shows the entered expression and result dynamically.  
✔ **Error Handling:** Prevents invalid operations like division by zero.

**2. Non-Functional Requirements *(Quality attributes and constraints)***

✔ **Usability:** Simple, intuitive interface resembling the Windows Calculator.  
✔ **Performance:** Fast, efficient calculations without lag.  
✔ **Scalability:** Structured using Object-Oriented Programming (OOP) for future feature expansion.  
✔ **Portability:** Runs on different operating systems (Windows, Linux, macOS).

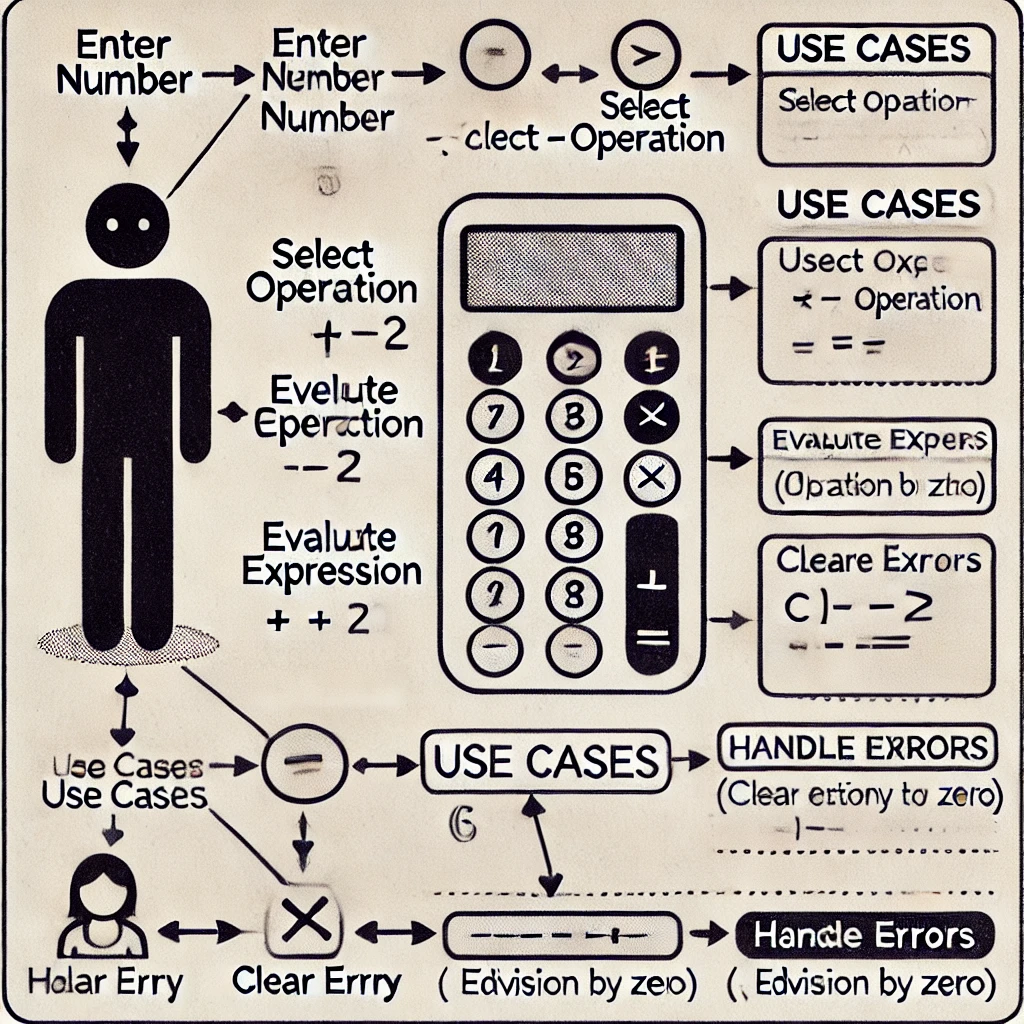
**3. User Interface (UI) Requirements**

✔ **Modern Layout:** Grid-based button layout for a clean, structured UI.  
✔ **Color-Coding:** Different colors for numbers, operators, and special functions.  
✔ **Responsive Design:** Dynamic resizing for different screen sizes.

**4. Software & Hardware Requirements**

✔ **Programming Language:** Python 3+  
✔ **GUI Library:** Tkinter (or PyQt for an advanced UI)  
✔ **Mathematical Operations:** Python’s built-in eval() function or custom parsing for security  
✔ **Hardware:** Works on any system with Python installed (low resource usage).

Below is a use case diagram generated by ChatGPT using graphviz

:

# **3.0 ALGORITHIM DESIGN AND GENERALIZED PROBLEM SOLUTION**

## **3.1 Temperature Converter Algorithm**

**PSEUCODE**

BEGIN

INPUT temperature, conversion type

IF conversion type is "F to C" THEN

converted\_temp = (temperature - 32) \* 5/9

ELSE

converted\_temp = (temperature \* 9/5) + 32

ENDIF

DISPLAY converted\_temp

PLOT bar chart

END

**FLOWCHART**

**A diagram of a flowchart

Description automatically generated**

**Modification or Generalization of algorithm using OOP concept, Inheritance**

Say, we were to write a code that does conversion for not just temperature but weight and distance. Rather than write separate functions for each conversion (Fahrenheit to Celsius, Kilogram to Pounds, Kilometer to Miles), we create a “template” class that will provide a structure for conversions, we then create specialized classes that inherit from this template and define their own conversion rules. This approach is known as **inheritance** and it makes code clean, reusable and easy to extend.

**Time Complexity of the Temperature Converter**

Time Complexity is a way of measuring how fast a program runs as the input size increases. Some programs slow down when you give them more input while some others stay fast no matter what.

The Temperature converter performs a simple mathematical calculation:

* Fahrenheit to Celsius

*(*

* Celsius to Fahrenheit

*+32*

These are **basic math operations** (addition, subtraction, multiplication, division), which **always take the same amount of time**, no matter what the input is.

**Big O Notation (O (1))**

* The temperature converter **only takes one step** to compute the result.
* It does **not** use loops or complex calculations.
* Because it always runs in a **fixed amount of time**, we say its time complexity is **O (1)** (constant time).

## **3.2 A BASIC CALCULATOR ALGORITHIM**

**PSEUDOCODE:**

BEGIN

DISPLAY Calculator GUI

FUNCTION perform\_operation(operator, num1, num2)

IF operator is "+" THEN

RETURN num1 + num2

ELSE IF operator is "-" THEN

RETURN num1 - num2

ELSE IF operator is "\*" THEN

RETURN num1 \* num2

ELSE IF operator is "/" THEN

IF num2 is NOT zero THEN

RETURN num1 / num2

ELSE

DISPLAY "Error: Division by zero"

ENDIF

END FUNCTION

FUNCTION on\_button\_click(button\_value)

IF button\_value is a number THEN

APPEND number to current input

ELSE IF button\_value is an operator (+, -, \*, /) THEN

STORE current input as first number

STORE operator

CLEAR input field

ELSE IF button\_value is "=" THEN

STORE current input as second number

CALL perform\_operation(operator, first number, second number)

DISPLAY result

ELSE IF button\_value is "C" (Clear) THEN

RESET input field

CLEAR stored values

ENDIF

END FUNCTION

LOOP UNTIL user closes the application

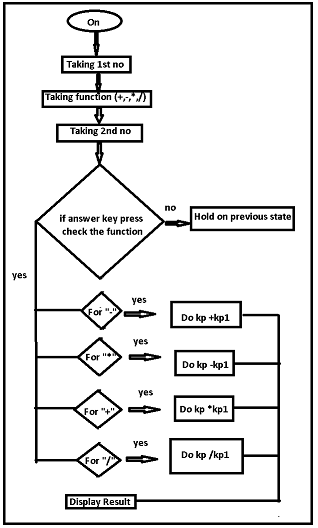
WAIT for user input (button click)

CALL on\_button\_click(button\_value)

END LOOP

END

**Flowchart**

****

**Modification or Generalization of algorithm using OOP concept, Inheritance**

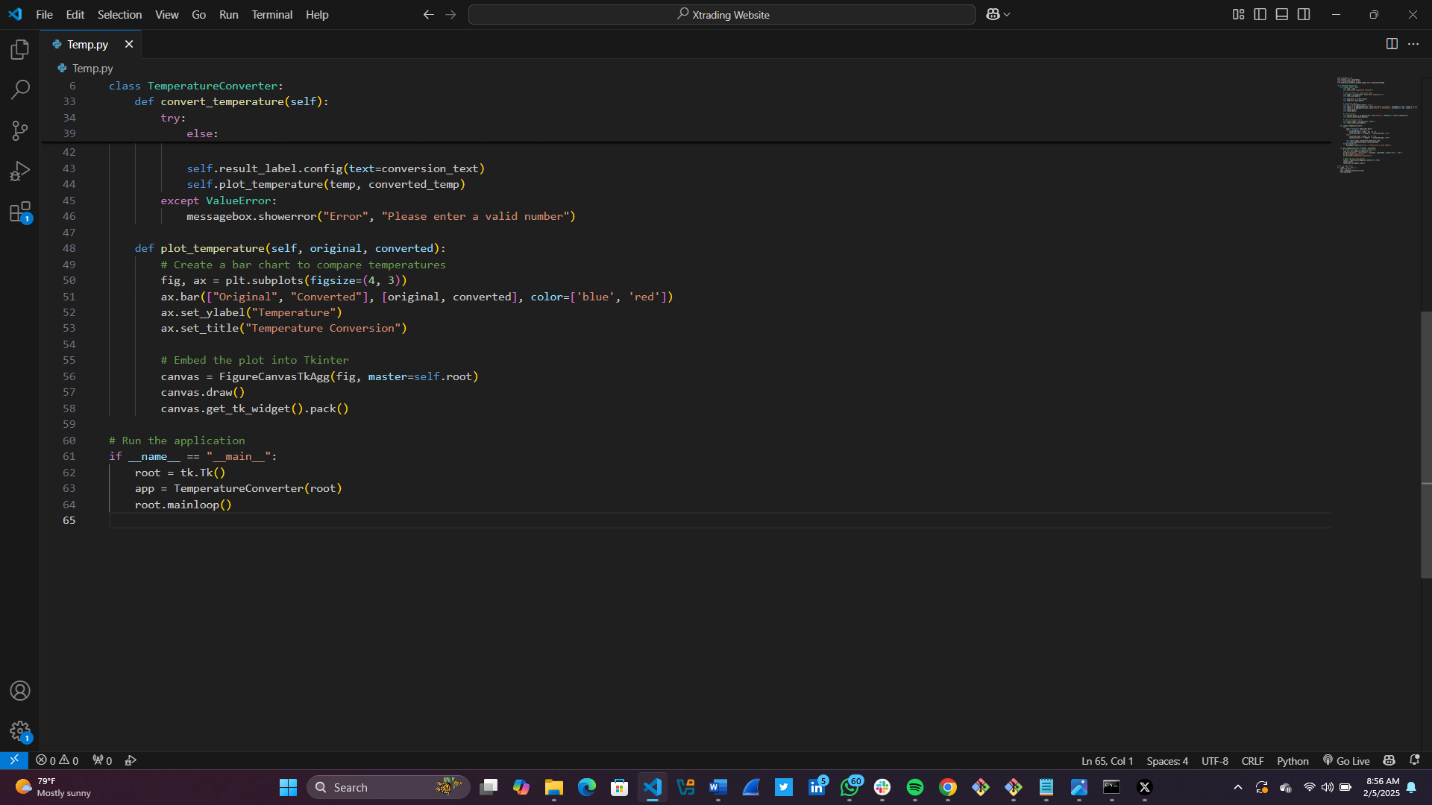
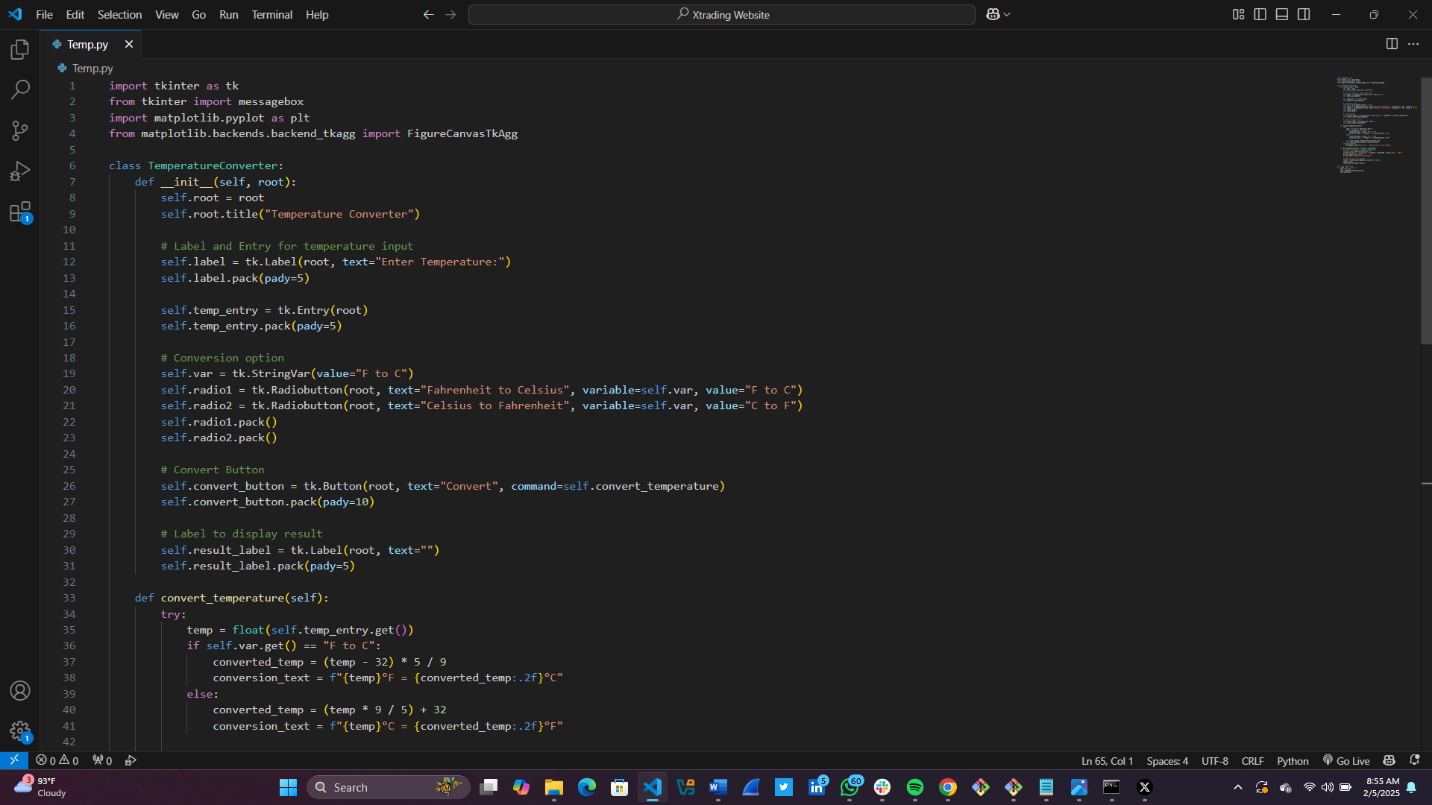
The algorithm above only works on arithmetic operations. However, to make it more adaptable to various use cases e.g. scientific, financial or programmer’s calculator. We could use inheritance, i.e. instead of writing separate calculators for different kinds of calculations, we could create a base class ***BasicCalculator*** *and* then extend it into different calculators.

**Time Complexity of A Basic Calculator**

The time and space complexity analysis of the calculator application can be broken down as follows:  
  
Time Complexity:  
  
1. Button Click Handling: Each button click triggers a function that updates the expression or evaluates it. The time complexity for updating the expression (adding a character or clearing it) is O (1) since it involves simple string concatenation or assignment.  
  
2. Expression Evaluation: The evaluation of the expression when the "=" button is pressed can vary in complexity depending on the implementation of the evaluation logic. If a simple method like `eval () ` is used, it operates in O(n) time, where n is the length of the expression string. More complex parsing methods (like Shunting Yard or similar algorithms) may have a time complexity of O(n log n) or O(n) depending on the specific implementation.  
  
Overall, the most time-consuming operation is likely the evaluation of the expression, leading to an average time complexity of O(n).

# **4.0 PROGRAMMING LANGUAGE CONCEPT**

## **4.1 Temperature Converter**



1. **Tkinter for GUI:**

**Tkinter** is a standard Python library for creating graphical user interfaces (**GUIs**). It provides various widgets like labels, buttons, entry fields, and more to build interactive applications.

1. **Class Definition**:

The “**TemperatureConverter**” class encapsulates all the functionality related to the temperature conversion application. This includes initializing the GUI components, handling user input, performing the conversion, and displaying the results.

1. **Widgets:**
   * + **Label**: Used to display text or information to the user
     + **Entry**: A single-line text field for user input
     + **Radiobutton**: Allow the user to select one option from a set of mutually exclusive options
     + **Button**: A clickable button that triggers an action
     + **StringVar**: A **Tkinter** variable class used to manage the value of a widget like a Radiobutton
2. **Event Handling:**

The ***command*** parameter in the button widget specifies the function to be called when the button is clicked

1. **Exception Handling:**

The ***try-except*** block is used to handle potential errors, such as invalid user input

1. **Matplotlib for plotting:**

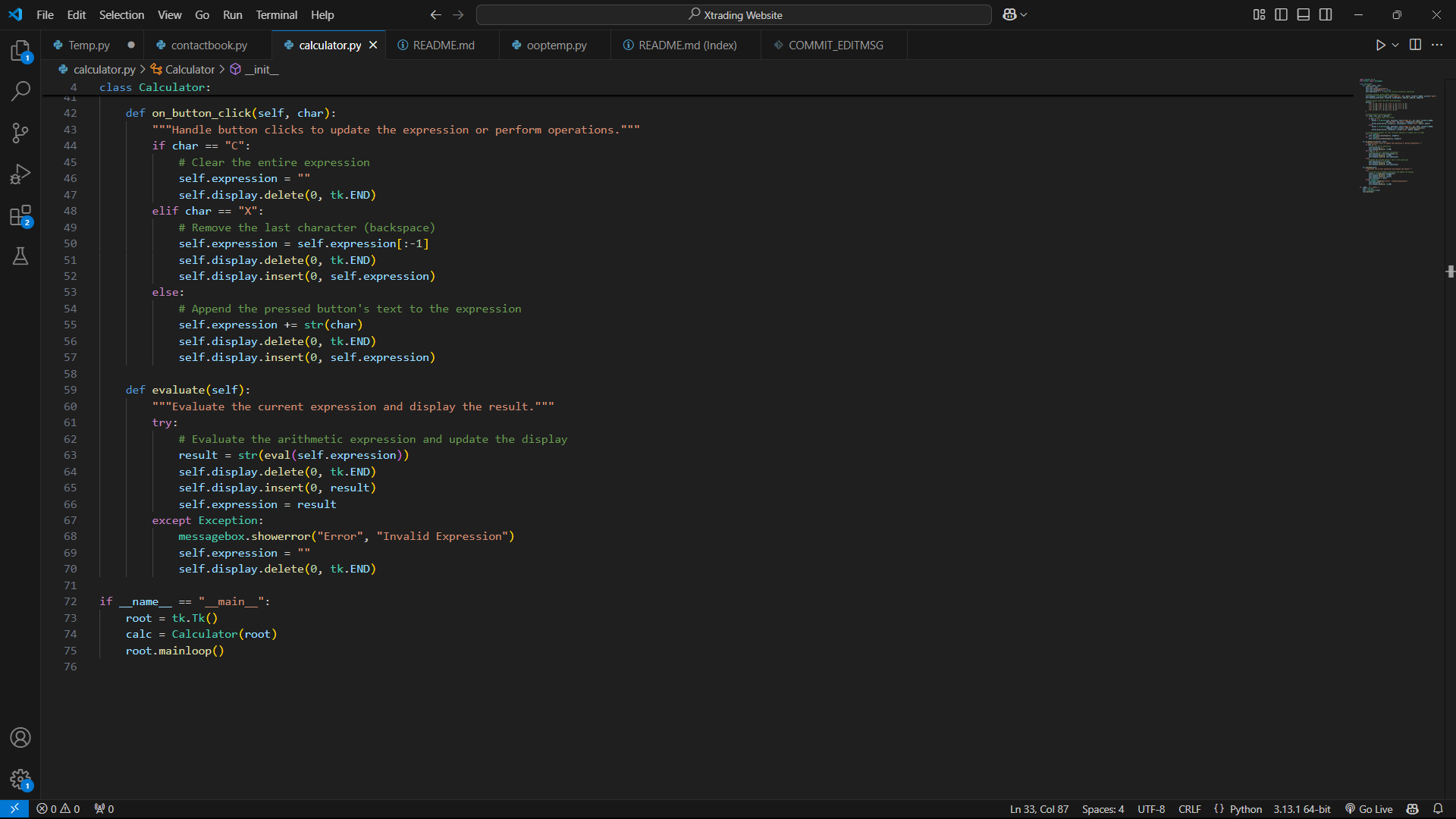
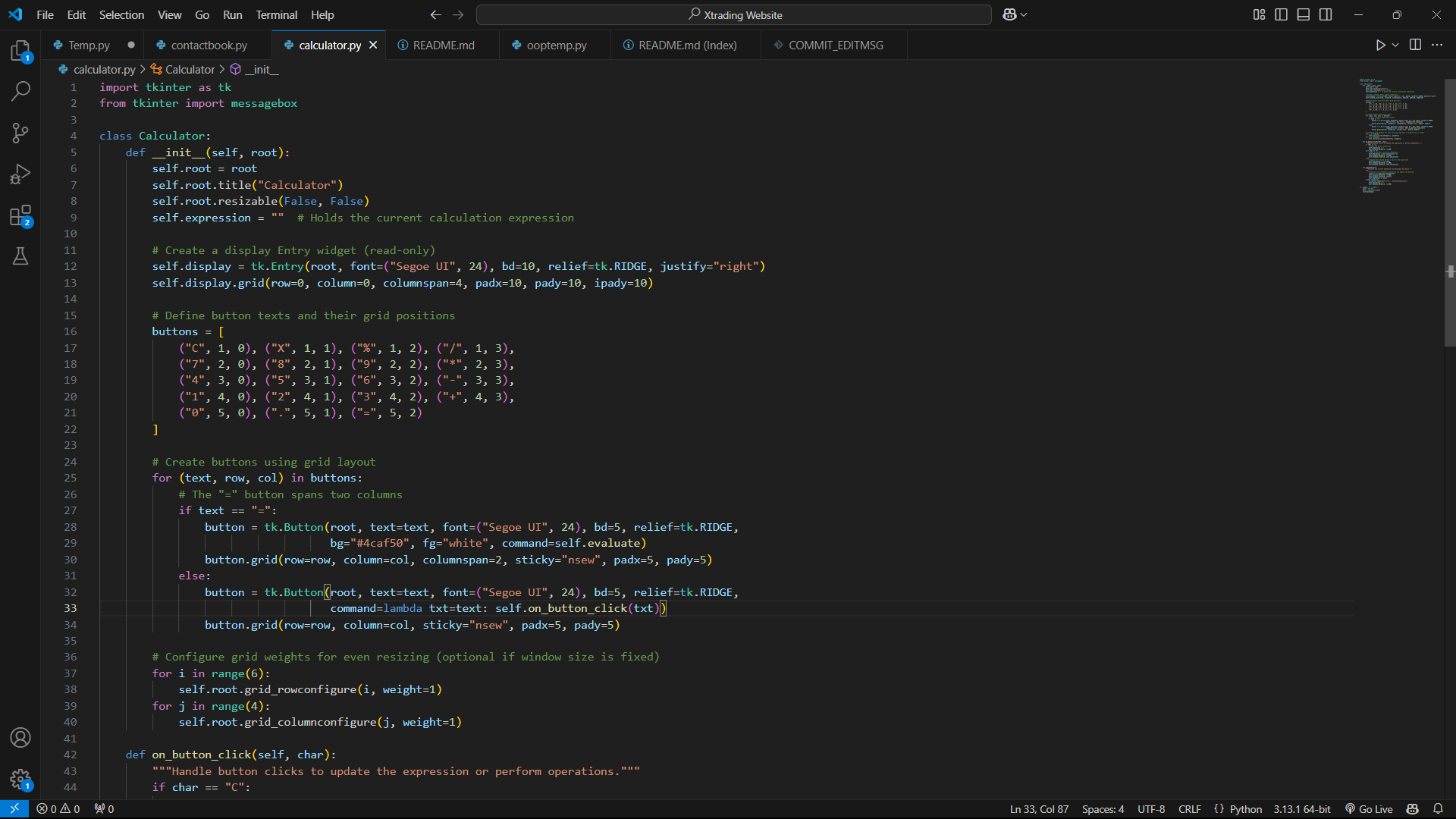
**Matploitlib** is a plotting library that is used to create interactive visualizations in python

1. **Plotting Data:**

The ***Plot***\_***temperature*** method creates a bar chart to compare the original and converted temperature

The actual conversion logic is encapsulated in the “***def convert\_temperature(self)***” function, the function first receives the argument of the user input from the GUI and stores it in a variable, then from the Gui, it also checks what kind of conversion the user wants to perform from whatever selection the user makes from the GUI. Then uses an IF-ELSE statement to calculate depending on the user selection. The actual code will be added to the appendix of this document.

## **4.2 A Basic Calculator**



**Explanation of Key Programming Concepts in the GUI Calculator**

**1. Key Syntax Elements**

* **import tkinter as tk** → Imports Tkinter, Python’s standard library for GUI applications.
* **class BasicCalculator:** → Defines a class to encapsulate calculator functionality.
* **def \_\_init\_\_(self, root):** → The class constructor initializes the calculator’s attributes and GUI components.
* **self.expression = ""** → Stores the mathematical expression entered by the user.
* **self.display = tk.Entry(...)** → Creates an input field to display calculations.
* **self.display.insert(tk.END, self.expression)** → Updates the display with the latest expression.
* **lambda t=text: self.on\_button\_click(t)** → Uses lambda functions to pass button values dynamically.
* **if \_\_name\_\_ == "\_\_main\_\_":** → Ensures that the script runs only when executed directly.

**2. Classes and Object-Oriented Programming (OOP)**

**Classes Used:**

* **BasicCalculator** → A base class for a simple calculator with addition, subtraction, multiplication, and division.
* .

**3. Functions Used**

**Core Functions**

* **on\_button\_click(self, value)**
  + Handles button presses and updates the display.
  + Uses **eval(self.expression)** to evaluate the mathematical expression safely.

self.display.delete(0, tk.END)

self.display.insert(tk.END, self.expression)

except:

self.display.delete(0, tk.END)

self.display.insert(tk.END, "Error")

**4. Data Structures Used**

The program mainly uses **strings and lists** to store input and output.

* **Strings (self.expression)** → Used to store user input as an equation (e.g., "5+3\*2").
* **Lists (button grid)** → Used to store button labels, positions, and references dynamically.

This **list of tuples** helps create the buttons dynamically.

**5. Control Structures**

Control structures manage the program flow:

**(i) Conditional Statements (if-elif-else)**

Used for handling button interactions, calculations, and error checking.

**(ii) Loop Structures**

Loops are used for **creating buttons dynamically** instead of writing multiple lines for each button.

# **5.0 SOFTWARE DEVELOPMENT**

## 5.1 Software Development process for the Temperature Converter

**The Agile Methodology** was followed for this project because it accommodates incremental development, meaning at every stage of this program-continuous changes were made after tests. This ensures that the developed features-for instance, GUIs, functions, graphs were refined through the results of tests.

Agile Methodology has five major stages which are:

* Requirement gathering
* Design Phase
* Implementation/Coding
* Testing
* Deployment.

This project will however focus on the first four phases as deployment is beyond the scope of this project.

**PHASE 1: REQUIREMENTS GATHERING**

This phase involves trying to understand the problem at hand and defining the requirements to solve the problem. The requirements for the temperature converter include:

* Convert temperatures between Celsius and Fahrenheit and vice versa
* Provide a user-friendly GUI (Graphical user interface)
* Use graphs to visualize temperature change
* Implement object-oriented programming (OOP)

**PHASE 2: DESIGN PHASE**

This Phase involves using algorithms and flowcharts to and mind maps to design the overall architecture of the system, basically to visualize the working of the system.

* **Flowcharts** were created to visualize:
  + The **temperature conversion process**.
  + The **logic for checking palindromes**.
* **Pseudocode** was written before actual coding to structure the logic.
* **OOP Design Considerations**
  + **Created a base class** (Converter) for general conversion.
  + **Used inheritance** to extend the class for **temperature conversion**.
  + Allowed easy future extensions (e.g., weight or distance conversion).

**PHASE 3: IMPLEMENTATION/CODING**

In this phase, we will Write Python codes following best practices.

* **Programming concepts used:**
  + **Functions** for modularity.
  + **Classes and Objects (OOP)** for maintainability.
  + **Tkinter** for GUI development.
  + **Matplotlib** for graphs and data visualization.
* **Coding Approach:**
  + Developed core functionalities **first**.
  + Integrated **GUI elements** after core logic was working.
  + Added **graphical representations** to enhance user experience.

**PHASE 4: TESTING & DEBUGGING**

In this phase, correctness is ensured, and errors are fixed. Various kinds of tests are carried out in this phase, some have been highlighted below:

* **Unit Testing**
  + Tested individual functions (e.g., temperature conversion formula).
  + Checked **GUI responsiveness** with different inputs.
* **Manual Testing**
  + Input **edge cases** (e.g., negative temperatures, very large numbers).
  + **Debugging Approach**: Employed Python’s **exception handling** (try-except) to prevent crashes.

## 5.2 Software Development process for the Basic Calculator

For the **GUI-based calculator**, we followed the **Agile software development methodology** because of its **iterative, flexible, and user-centric approach**. Agile is well-suited for projects that require continuous feedback and improvements, such as a calculator where **new features (e.g., scientific operations, history tracking) can be added over time**.

**Phases of Agile Software Development in This Project**

**1. Requirement Analysis (Understanding the Problem)**

* **Goal:** Identify what the calculator should do.
* **Activities:**
  + Gather user requirements (basic arithmetic, scientific functions, clean UI).
  + Define core functionalities:
    - Perform arithmetic operations.
    - Provide an easy-to-use graphical interface.

**2. Planning (Sprint Planning & Task Breakdown)**

* **Goal:** Define project scope and plan iterations (sprints).
* **Activities:**
  + Break down the project into **sprints**:
    - Sprint 1: Develop basic calculator with GUI.
    - Sprint 2: Implement arithmetic operations.

**3. Design (Algorithm Design)**

* **Goal:** Create the system architecture and UI design.
* **Activities:**
  + Write **pseudocode** for core functions before coding.
* **Tools Used:** Pseudocode.

**4. Implementation (Coding the Application)**

* **Goal:** Write and develop the actual software based on the design.
* **Activities:**
  + Set up the project environment (tkinter for GUI, math module for calculations).
  + Implement **basic functionalities** (buttons, display, input handling).
  + Ensure **modular programming** by using classes and functions.
* **Key Decisions:**
  + **Use OOP concepts**: Created BasicCalculator class.
  + **Use lists for dynamic button generation** to reduce hardcoded elements.

**5. Testing (Bug Fixing & Performance Testing)**

* **Goal:** Ensure the software works as expected without errors.
* **Activities:**
  + **Unit Testing**: Test individual functions (e.g., addition, division).
  + **Integration Testing**: Verify how different components interact.
* **Tools Used:** Python unit test module, Manual Testing.

# 6.0 CONCLUSION

Developing this portfolio has been a valuable experience, allowing me to apply programming concepts to real-world scenarios. The following are the key takeaways:

**Learnings Derived:**

* Modular programming is essential: Dividing the application into functions and classes made debugging and feature addition easier.
* User experience (UX) design matters: Properly designed UI enhances usability and user satisfaction.
* Agile approach is a success: Iterative development helped to iron out the application through ongoing testing and feedback.

**Challenge Encountered:**

* Debugging difficult mathematics: There were certain operations, like dealing with floating-point precision, which required additional error handling.
* Graphical User Interface (GUI) constraints: Ensuring correct responsiveness and layout across different screen sizes was a nightmare.
* Time estimation: Planning time for testing, coding, research, and documentation was a learning experience.

**Enhancement of Problem-Solving and Programming Skills:**

* Improved coding effectiveness: Creating reusable classes and functions improved my ability to structure code effectively.
* Improved debugging techniques: Working with and solving runtime errors improved my debugging technique.
* Improved algorithmic thinking: Creating flowcharts, pseudocode, and algorithms helped structuring complex functionalities in a methodical way.

This project not only strengthened my knowledge of Python, GUI programming, and software development procedures but also prepared me for more intricate projects in the future.