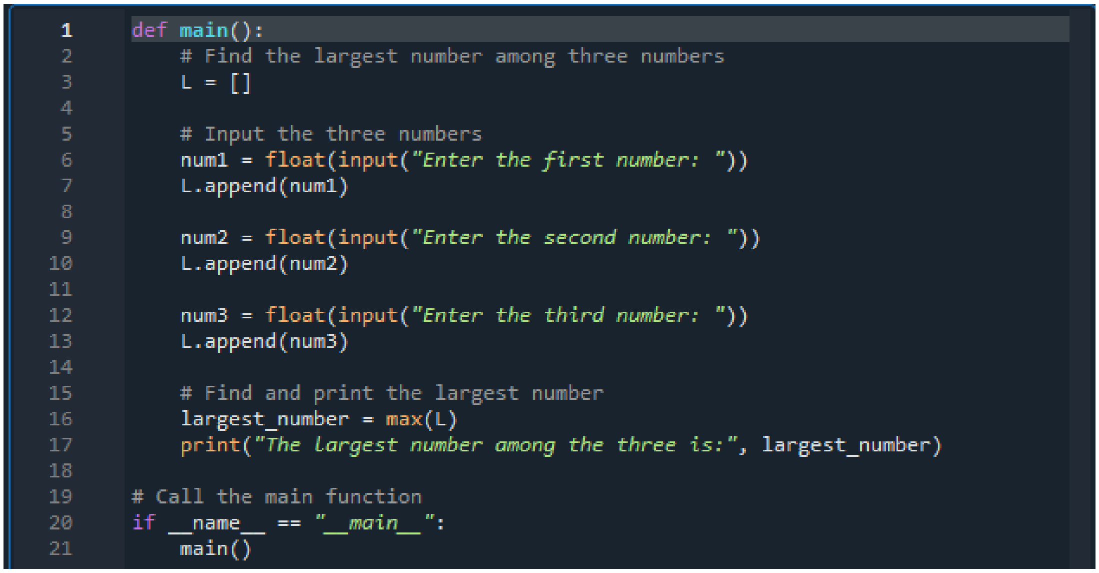
Lab Activity # 7 - TUI to GUI	
Group members: **Mamaril, Justin Kenneth, - Leader** Reyes, Alexzander, San Jose, Alexander, San Juan, Edson Ray, Titong, Lee Ivan	**Date Performed:** 11/11/2024
Course Code / Section: CPE009B - CPE21S4	**Instructor:** Prof. Sayo

5. Procedure

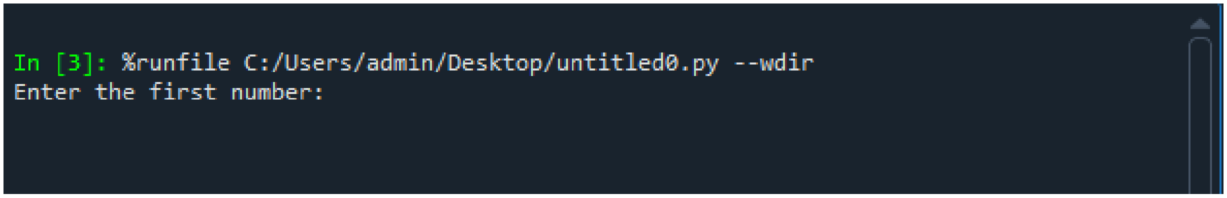
Method 1

1. Type the code in pycharm (we used Spyder as we don't have access to pycharm)



```python
def main():
    # Find the largest number among three numbers
    L = []

    # Input the three numbers
    num1 = float(input("Enter the first number: "))
    L.append(num1)

    num2 = float(input("Enter the second number: "))
    L.append(num2)

    num3 = float(input("Enter the third number: "))
    L.append(num3)

    # Find and print the largest number
    largest_number = max(L)
    print("The largest number among the three is:", largest_number)

# Call the main function
if __name__ == "__main__":
    main()
```

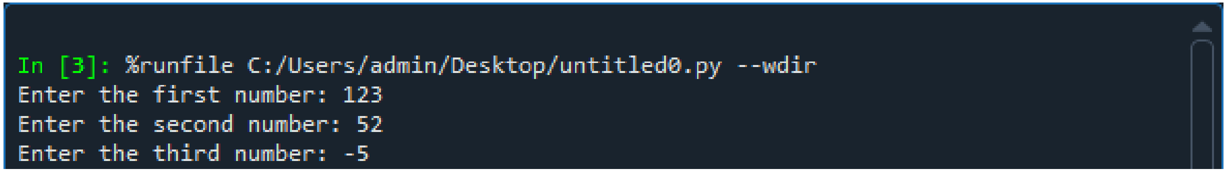
2. Run the program and observe the output



```
In [3]: %runfile C:/Users/admin/Desktop/untitled0.py --wdir
Enter the first number:
```

Figure 1. TUI Form

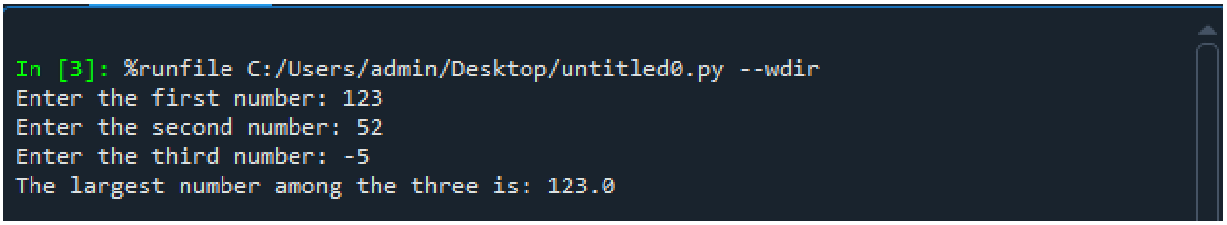
- The input() function prompts the user to enter the first number. The message "Enter the first number: " is displayed in the console.
- The input is converted to a float (a decimal number) so that the program can handle both integers and floating-point numbers.
- The entered number is then appended to the list L using the append() method.



```
In [3]: %runfile C:/Users/admin/Desktop/untitled0.py --wdir
Enter the first number: 123
Enter the second number: 52
Enter the third number: -5
```

Figure 1(a) TUI form with three input numbers

- The same process is repeated for the second and third numbers:



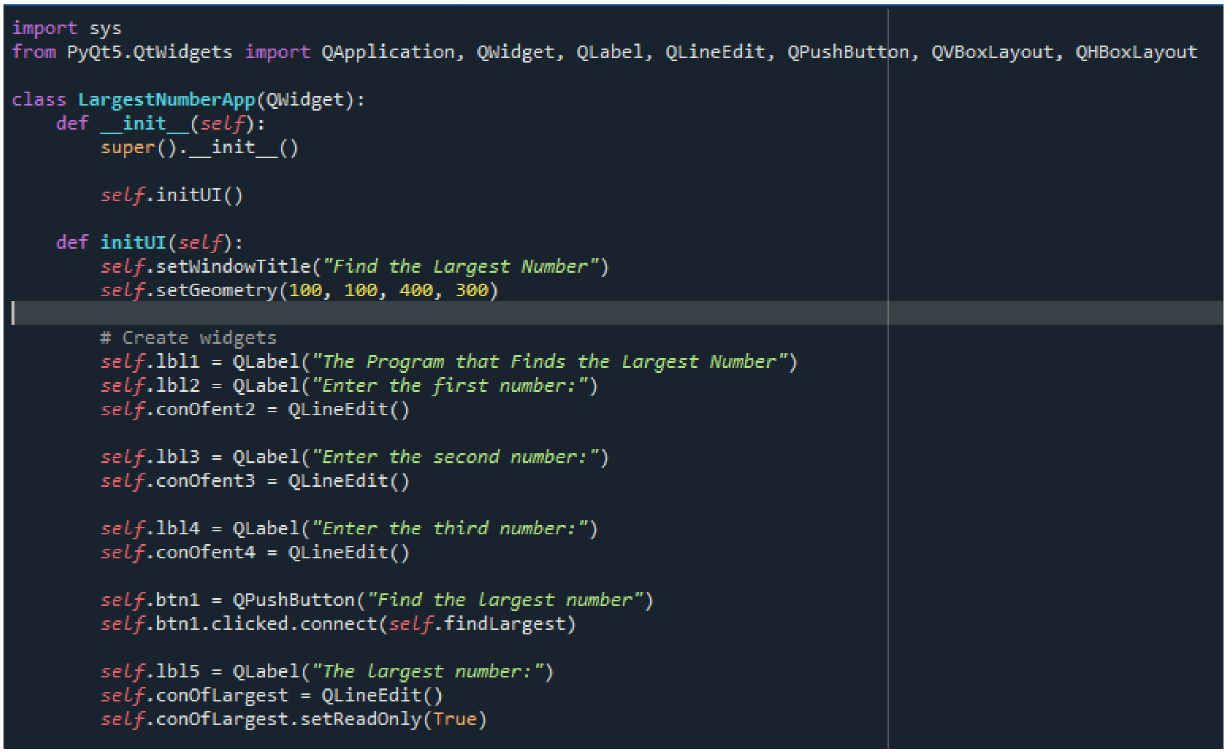
```
In [3]: %runfile C:/Users/admin/Desktop/untitled0.py --wdir
Enter the first number: 123
Enter the second number: 52
Enter the third number: -5
The largest number among the three is: 123.0
```

Figure 1(b) TUI form with output “The largest number among the three”

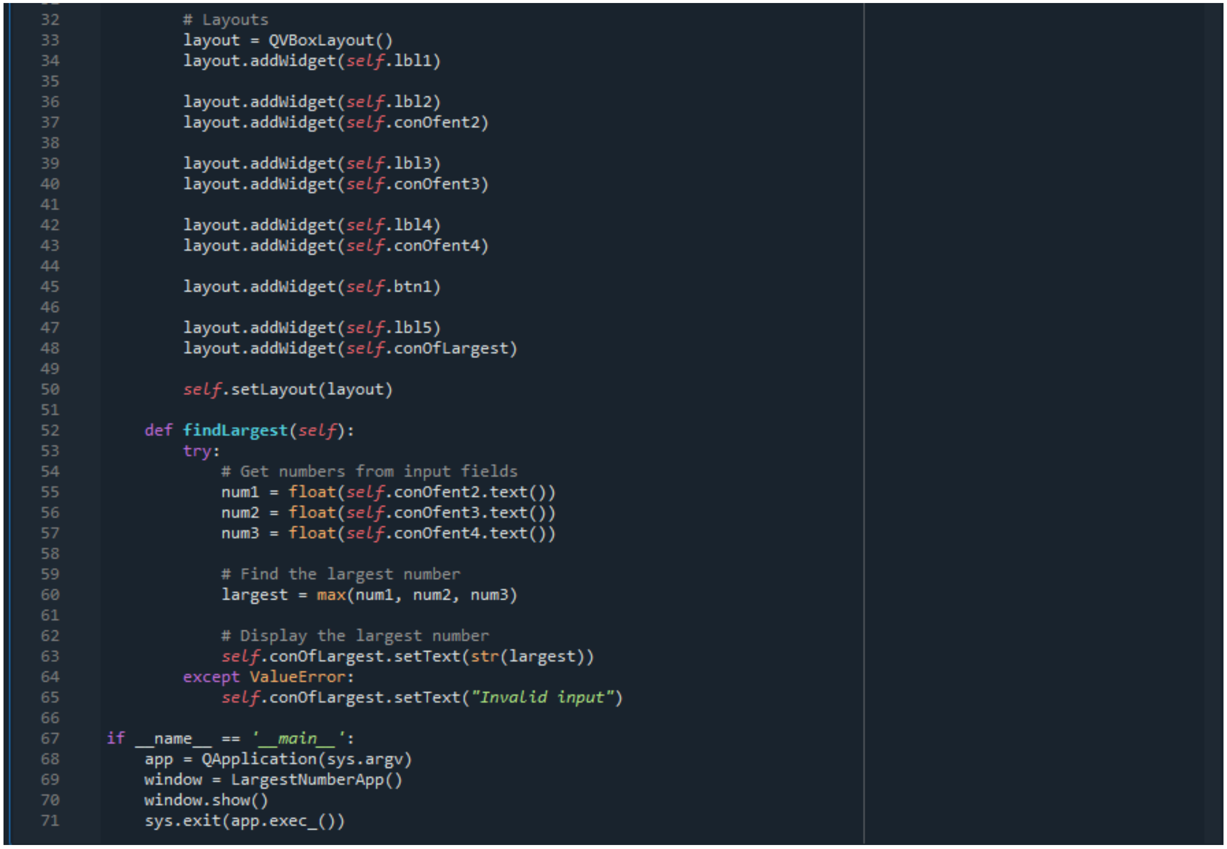
- Once all three numbers are collected in the list L, the max() function is called. This built-in function returns the largest number from the list. The result is stored in the variable largest_number.
- The print() function outputs the specified message along with the value of largest_number.

Method 2

- We converted the code into Spyder IDE so we used PyCharm for the code to work

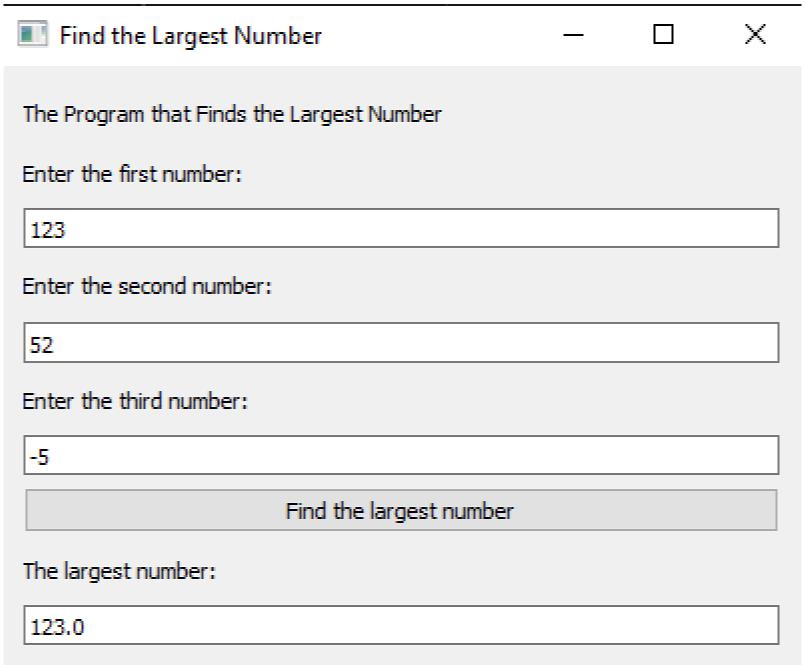


```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout

class LargestNumberApp(QWidget):
    def __init__(self):
        super().__init__()

        self.initUI()

    def initUI(self):
        self.setWindowTitle("Find the Largest Number")
        self.setGeometry(100, 100, 400, 300)

        # Create widgets
        self.lbl1 = QLabel("The Program that Finds the Largest Number")
        self.lbl2 = QLabel("Enter the first number:")
        self.conOfent2 = QLineEdit()

        self.lbl3 = QLabel("Enter the second number:")
        self.conOfent3 = QLineEdit()

        self.lbl4 = QLabel("Enter the third number:")
        self.conOfent4 = QLineEdit()

        self.btn1 = QPushButton("Find the largest number")
        self.btn1.clicked.connect(self.findLargest)

        self.lbl5 = QLabel("The largest number:")
        self.conOfLargest = QLineEdit()
        self.conOfLargest.setReadOnly(True)
```



```
        # Layouts
        layout = QVBoxLayout()
        layout.addWidget(self.lbl1)

        layout.addWidget(self.lbl2)
        layout.addWidget(self.conOfent2)

        layout.addWidget(self.lbl3)
        layout.addWidget(self.conOfent3)

        layout.addWidget(self.lbl4)
        layout.addWidget(self.conOfent4)

        layout.addWidget(self.btn1)

        layout.addWidget(self.lbl5)
        layout.addWidget(self.conOfLargest)

        self.setLayout(layout)

    def findLargest(self):
        try:
            # Get numbers from input fields
            num1 = float(self.conOfent2.text())
            num2 = float(self.conOfent3.text())
            num3 = float(self.conOfent4.text())

            # Find the largest number
            largest = max(num1, num2, num3)

            # Display the largest number
            self.conOfLargest.setText(str(largest))
        except ValueError:
            self.conOfLargest.setText("Invalid input")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = LargestNumberApp()
    window.show()
    sys.exit(app.exec_())
```

Output:



The Program that Finds the Largest Number

Enter the first number:

123

Enter the second number:

52

Enter the third number:

-5

The largest number:

123.0

- This code demonstrates how to create a basic PyQt application for numerical input and processing, showcasing essential GUI programming concepts such as event handling, widget management, and layout organization. It serves as a practical example for beginners to understand how to build interactive applications using PyQt.
- User Input: The application consists of three input fields (QLineEdit) where users can enter numbers.
- Button Interaction: A button (QPushButton) is provided for users to trigger the computation. When clicked, it invokes the findLargest method.
- Logic for Finding the Largest Number: The findLargest method retrieves the numbers from the input fields, converts them to floating-point values, and uses the built-in max() function to determine the largest number. If the input is invalid (e.g., non-numeric), it handles the error gracefully by displaying an appropriate message.
- Output Display: The result, which is the largest number, is displayed in a read-only input field, preventing users from altering it.
- Layout Management: The application utilizes a vertical layout to organize the widgets in a user-friendly manner, ensuring a clean and intuitive interface.

Questions:

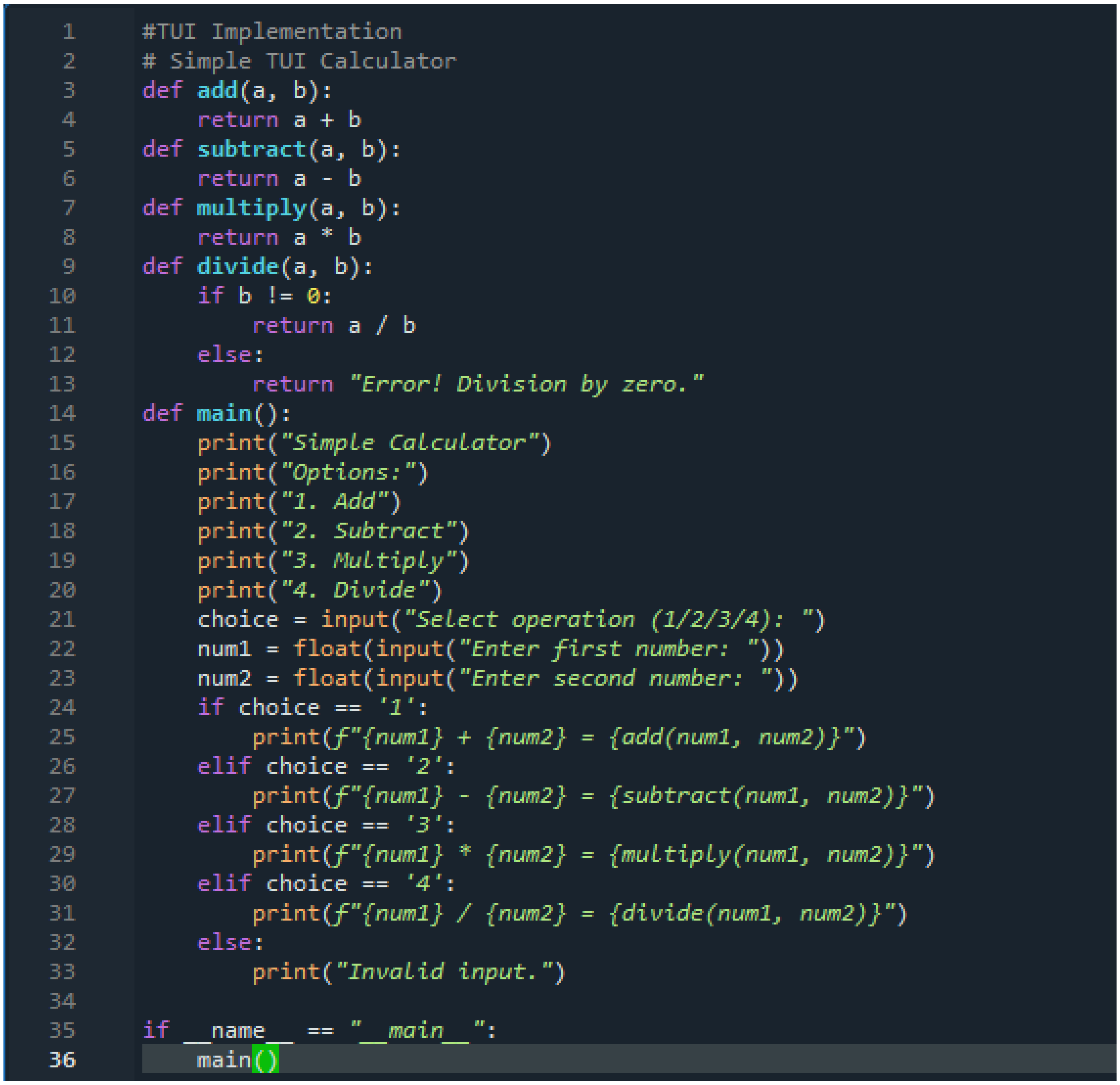
1. **What is TUI in Python?**
   Python's Text User Interface (TUI) enables users to interact with programs via text-based input and output via a terminal or console. When a graphical user interface (GUI) is not feasible, it is the best option because it is lightweight and resource-efficient. A number of Python packages are available for generating TUIs, such as Prompt Toolkit for interactive command-line capabilities like autocompletion, Rich for writing aesthetically pleasing text displays, and Curses for basic terminal handling. File managers and monitoring apps are examples of system programs that frequently employ TUIs since they offer a quick and effective method of completing tasks without a graphical interface.

2. **How to make a TUI in Python?**
   You may use Python's built-in curses package to develop a basic Text User Interface (TUI), which facilitates controlling text display and terminal user input. The curses module must be imported first, and then a main function handling the TUI must be defined. To clean the screen, use stdscr.clear(); to show text at precise locations, use addstr(); and to update the terminal display, use refresh(). Getch() finally waits for a key to be pressed before ending the program. Using curses.wrapper(main), the TUI is initialized and cleanup is handled automatically. You may use this configuration to create a simple interactive text interface that shows a message, waits for user input, and then closes.

3. **What is the difference between TUI and GUI?**
   The main difference between a Text User Interface (TUI) and a Graphical User Interface (GUI) is in how users interact with the program. TUIs rely on text-based input and output, allowing users to type commands in a terminal or console window. They are generally faster for experienced users and require fewer system resources, making them suitable for low-performance environments or tasks requiring automation. In contrast, GUIs provide a visual interface with elements like buttons, icons, and menus, allowing users to interact using mouse clicks or touch gestures. GUIs are more intuitive and user-friendly, especially for beginners, but they consume

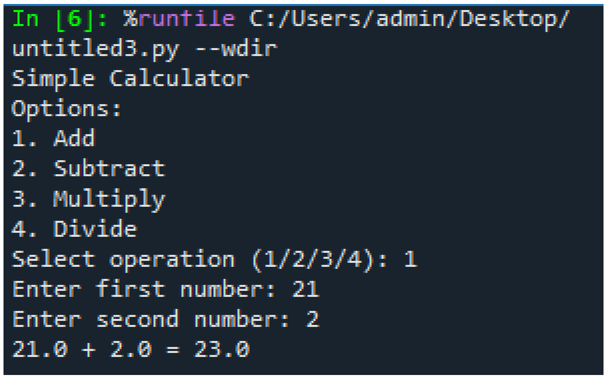
more resources due to graphical rendering. TUIs are often used in command-line tools and system utilities, while GUIs are common in desktop applications like web browsers and word processors.

6. Supplementary Activity



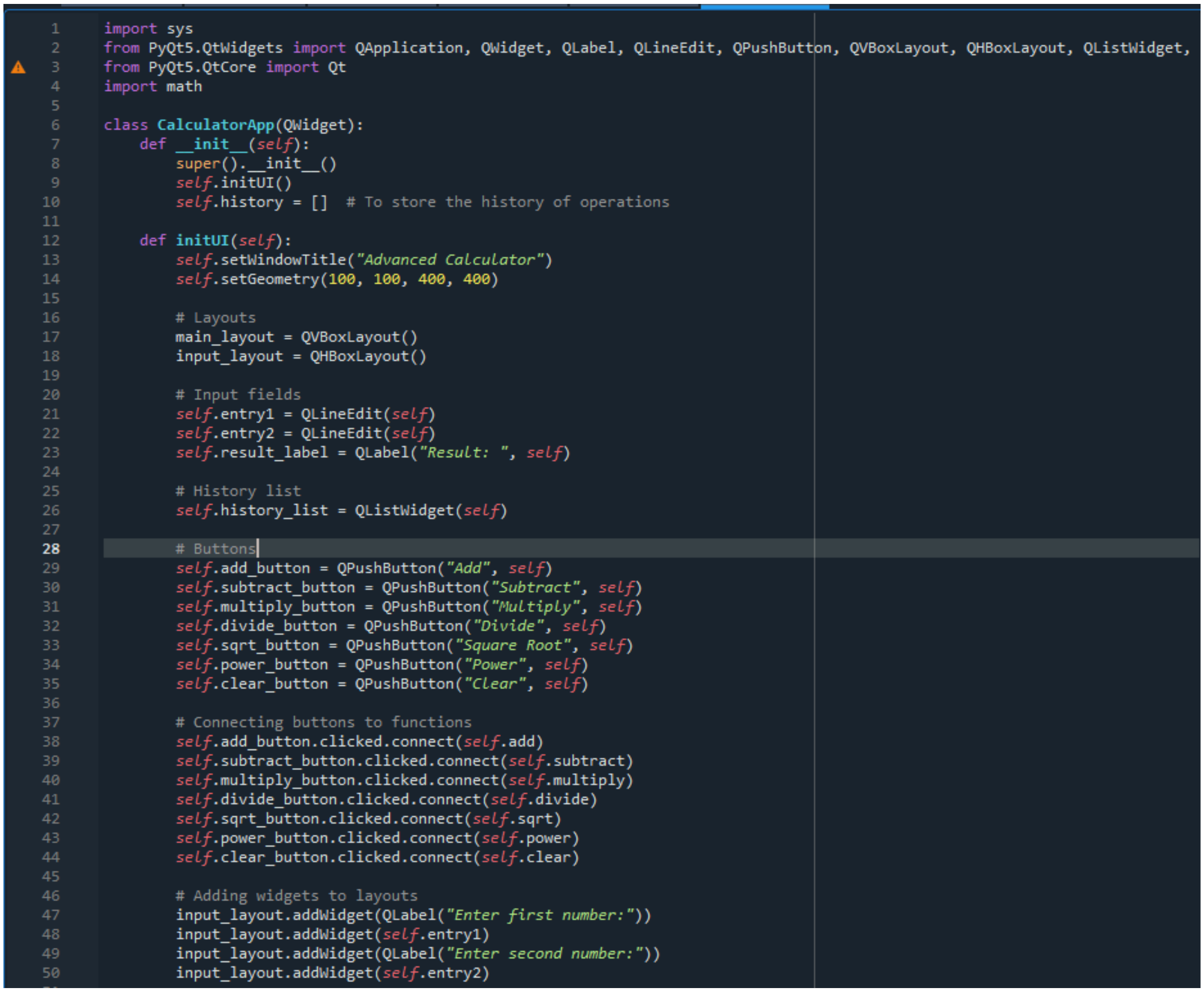
```python
#TUI Implementation
# Simple TUI Calculator
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    if b != 0:
        return a / b
    else:
        return "Error! Division by zero."
def main():
    print("Simple Calculator")
    print("Options:")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")
    choice = input("Select operation (1/2/3/4): ")
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    if choice == '1':
        print(f"{num1} + {num2} = {add(num1, num2)}")
    elif choice == '2':
        print(f"{num1} - {num2} = {subtract(num1, num2)}")
    elif choice == '3':
        print(f"{num1} * {num2} = {multiply(num1, num2)}")
    elif choice == '4':
        print(f"{num1} / {num2} = {divide(num1, num2)}")
    else:
        print("Invalid input.")

if __name__ == "__main__":
    main()
```

Output :

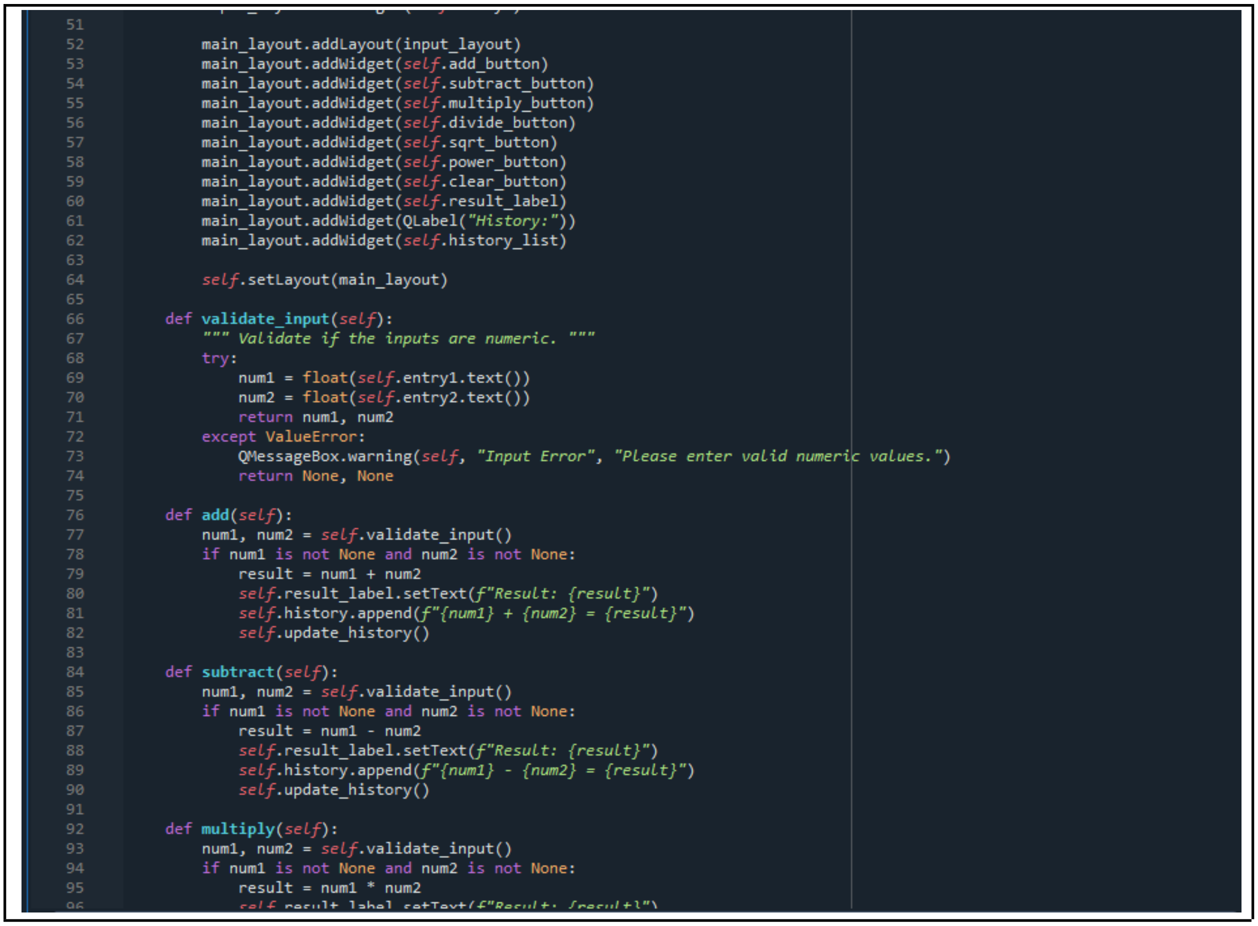


```
In [6]: %runfile C:/Users/admin/Desktop/
untitled3.py --wdir
Simple Calculator
Options:
1. Add
2. Subtract
3. Multiply
4. Divide
Select operation (1/2/3/4): 1
Enter first number: 21
Enter second number: 2
21.0 + 2.0 = 23.0
```

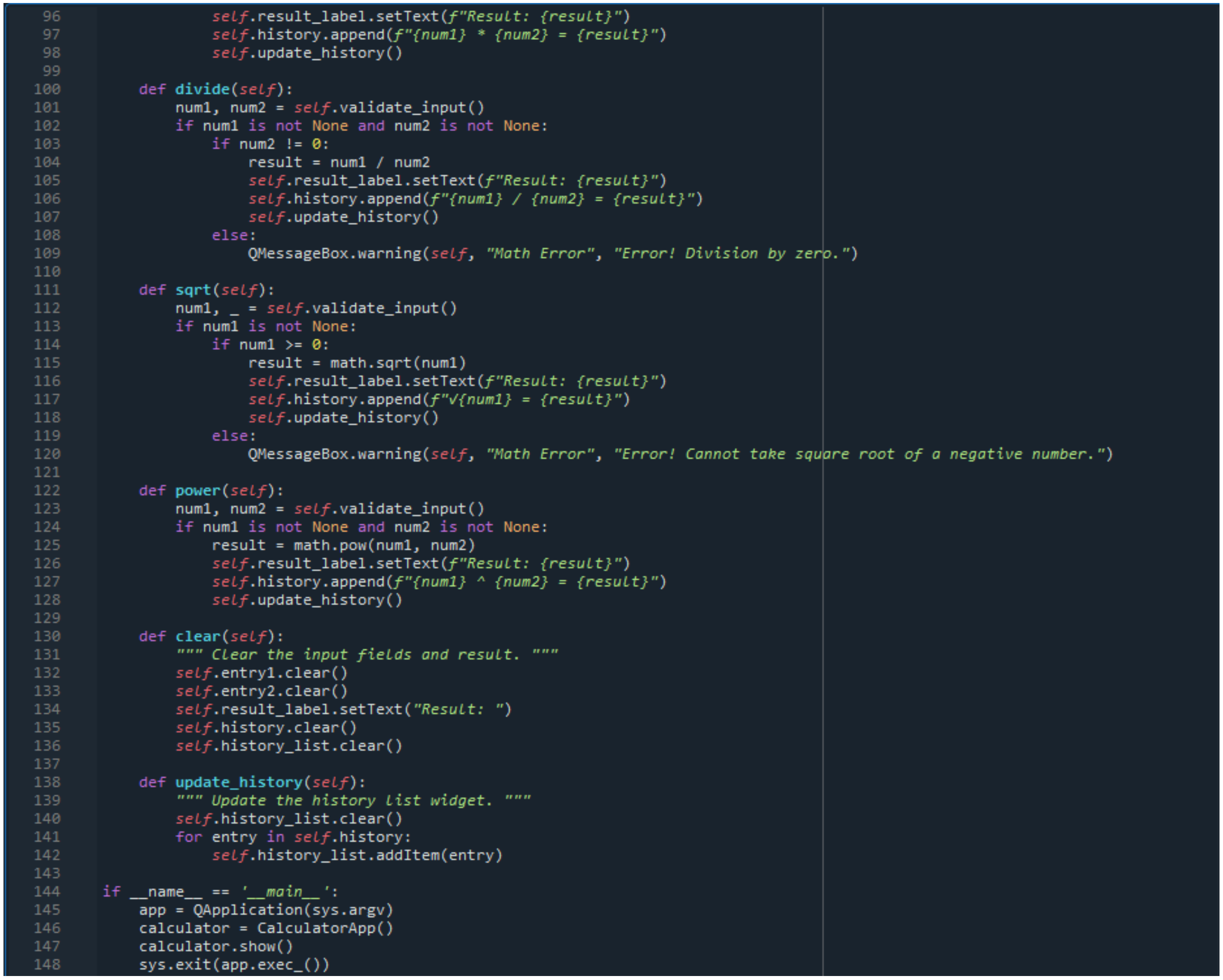
GUI Version of calculator with added features



```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout, QListWidget,
from PyQt5.QtCore import Qt
import math

class CalculatorApp(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
        self.history = []  # To store the history of operations

    def initUI(self):
        self.setWindowTitle("Advanced Calculator")
        self.setGeometry(100, 100, 400, 400)

        # Layouts
        main_layout = QVBoxLayout()
        input_layout = QHBoxLayout()

        # Input fields
        self.entry1 = QLineEdit(self)
        self.entry2 = QLineEdit(self)
        self.result_label = QLabel("Result: ", self)

        # History list
        self.history_list = QListWidget(self)

        # Buttons
        self.add_button = QPushButton("Add", self)
        self.subtract_button = QPushButton("Subtract", self)
        self.multiply_button = QPushButton("Multiply", self)
        self.divide_button = QPushButton("Divide", self)
        self.sqrt_button = QPushButton("Square Root", self)
        self.power_button = QPushButton("Power", self)
        self.clear_button = QPushButton("Clear", self)

        # Connecting buttons to functions
        self.add_button.clicked.connect(self.add)
        self.subtract_button.clicked.connect(self.subtract)
        self.multiply_button.clicked.connect(self.multiply)
        self.divide_button.clicked.connect(self.divide)
        self.sqrt_button.clicked.connect(self.sqrt)
        self.power_button.clicked.connect(self.power)
        self.clear_button.clicked.connect(self.clear)

        # Adding widgets to layouts
        input_layout.addWidget(QLabel("Enter first number:"))
        input_layout.addWidget(self.entry1)
        input_layout.addWidget(QLabel("Enter second number:"))
        input_layout.addWidget(self.entry2)
```

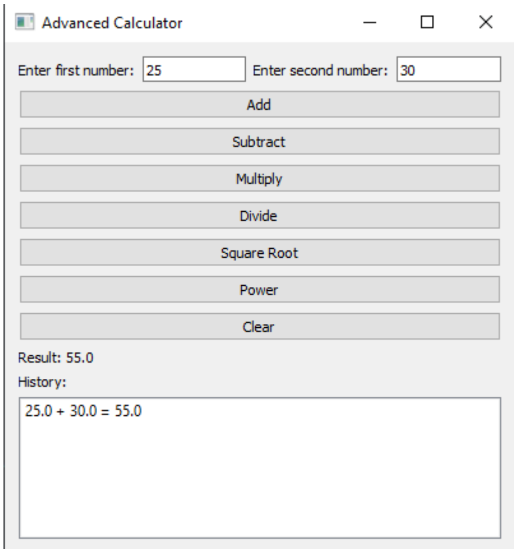


```
        main_layout.addLayout(input_layout)
        main_layout.addWidget(self.add_button)
        main_layout.addWidget(self.subtract_button)
        main_layout.addWidget(self.multiply_button)
        main_layout.addWidget(self.divide_button)
        main_layout.addWidget(self.sqrt_button)
        main_layout.addWidget(self.power_button)
        main_layout.addWidget(self.clear_button)
        main_layout.addWidget(self.result_label)
        main_layout.addWidget(QLabel("History:"))
        main_layout.addWidget(self.history_list)

        self.setLayout(main_layout)

    def validate_input(self):
        """ Validate if the inputs are numeric. """
        try:
            num1 = float(self.entry1.text())
            num2 = float(self.entry2.text())
            return num1, num2
        except ValueError:
            QMessageBox.warning(self, "Input Error", "Please enter valid numeric values.")
            return None, None

    def add(self):
        num1, num2 = self.validate_input()
        if num1 is not None and num2 is not None:
            result = num1 + num2
            self.result_label.setText(f"Result: {result}")
            self.history.append(f"{num1} + {num2} = {result}")
            self.update_history()

    def subtract(self):
        num1, num2 = self.validate_input()
        if num1 is not None and num2 is not None:
            result = num1 - num2
            self.result_label.setText(f"Result: {result}")
            self.history.append(f"{num1} - {num2} = {result}")
            self.update_history()

    def multiply(self):
        num1, num2 = self.validate_input()
        if num1 is not None and num2 is not None:
            result = num1 * num2
            self.result_label.setText(f"Result: {result}")
```



```python
            self.result_label.setText(f"Result: {result}")
            self.history.append(f"{num1} * {num2} = {result}")
            self.update_history()

    def divide(self):
        num1, num2 = self.validate_input()
        if num1 is not None and num2 is not None:
            if num2 != 0:
                result = num1 / num2
                self.result_label.setText(f"Result: {result}")
                self.history.append(f"{num1} / {num2} = {result}")
                self.update_history()
            else:
                QMessageBox.warning(self, "Math Error", "Error! Division by zero.")

    def sqrt(self):
        num1, _ = self.validate_input()
        if num1 is not None:
            if num1 >= 0:
                result = math.sqrt(num1)
                self.result_label.setText(f"Result: {result}")
                self.history.append(f"√{num1} = {result}")
                self.update_history()
            else:
                QMessageBox.warning(self, "Math Error", "Error! Cannot take square root of a negative number.")

    def power(self):
        num1, num2 = self.validate_input()
        if num1 is not None and num2 is not None:
            result = math.pow(num1, num2)
            self.result_label.setText(f"Result: {result}")
            self.history.append(f"{num1} ^ {num2} = {result}")
            self.update_history()

    def clear(self):
        """ Clear the input fields and result. """
        self.entry1.clear()
        self.entry2.clear()
        self.result_label.setText("Result: ")
        self.history.clear()
        self.history_list.clear()

    def update_history(self):
        """ Update the history list widget. """
        self.history_list.clear()
        for entry in self.history:
            self.history_list.addItem(entry)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    calculator = CalculatorApp()
    calculator.show()
    sys.exit(app.exec_())
```

Output :



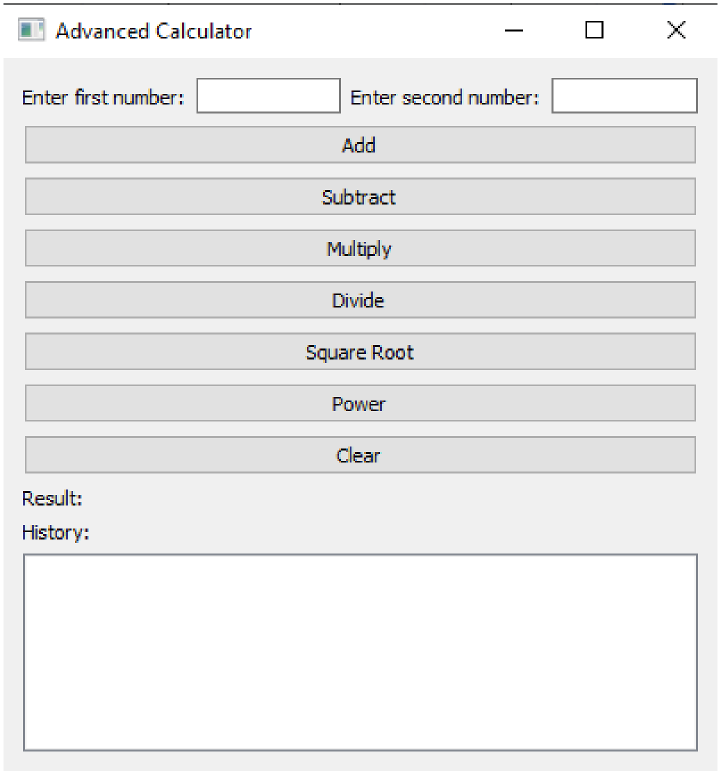
Enter first number: 25 Enter second number: 30

Result: 55.0

History:

25.0 + 30.0 = 55.0

Input



Clear Function

- We can clearly see that the program functions clearly, and i will explain now what the functions of the program are.
- Imports: The code imports necessary modules from PyQt5 and the math module for advanced calculations.
- CalculatorApp Class: This class inherits from QWidget and contains all the logic for the calculator.
- UI Initialization:
- The initUI() method sets up the main window, input fields, buttons, and layout.
- A vertical layout (QVBoxLayout) is used to arrange widgets vertically, and a horizontal layout (QHBoxLayout) is used for input fields.
- Input Validation:
- The validate_input() method checks if the input fields contain valid numeric values. If not, it shows a warning message.
- Basic Operations:
- The methods add(), subtract(), multiply(), and divide() perform the corresponding arithmetic operations. They validate inputs, perform calculations, update the result label, and add the operation to the history.
- Advanced Operations:
- The sqrt() method calculates the square root of the first input. It checks for negative input and shows an error if necessary.
- The power() method calculates the power of the first number raised to the second number.
- Clear Functionality:
- The clear() method clears the input fields, resets the result label, and clears the history list.

- History Feature:
- The update_history() method updates the history list widget to display all previous calculations.
- Main Execution:
- The code checks if the script is being run directly and creates an instance of the CalculatorApp, then starts the application's event loop.

7. Conclusion

Conclusion:

In this task, we used Python's PyCharm IDE to successfully transform a Text User Interface (TUI) software into a Graphical User Interface (GUI) program using the Tkinter package. By switching from TUI to GUI, we improved user experience by offering a more engaging and aesthetically pleasing method of data input and output. We discovered that the fundamental elements of a GUI application such as labels, entry fields, and buttons help to replace the conventional input and print instructions seen in TUI systems. This modification adds event-driven programming, where functions are triggered by user events (such as pressing a button). Overall, this exercise provided hands-on experience with GUI design, emphasizing the advantages of GUI over TUI in terms of usability and user experience. By learning to convert a program from TUI to GUI in PyCharm, we gained a deeper understanding of how to build interactive and user-friendly applications in Python.