

## Laboratory Activity 5 - Introduction to Event Handling in GUI Development

Solis, Paul Vincent M.

10/21/24

CPE 009 - CPE21S4

Prof Sayo

### PROCEDURE AND OUTPUT

#### gui\_buttonclicked.py

```
1 import sys
2 from PyQt5.QtWidgets import QWidget, QApplication, QMainWindow, QPushButton
3 from PyQt5.QtGui import QIcon
4 from PyQt5.QtCore import pyqtSlot
5
6 class App(QWidget):
7     def __init__(self):
8         super().__init__()
9
10
11         self.setWindowTitle("PyQt Button")
12         self.setGeometry(200, 200, 300, 300)
13         self.setWindowIcon(QIcon('pythonico.ico'))
14
15
16         self.button = QPushButton('Click me!', self)
17         self.button.setToolTip("You've hovered over me!")
18         self.button.move(100, 70)
19         self.button.clicked.connect(self.on_click)
20         self.show()
21
22     @pyqtSlot()
23     def on_click(self):
24         print("Button clicked!")
25
26 if __name__ == '__main__':
27     app = QApplication(sys.argv)
28     ex = App()
29     sys.exit(app.exec_())
```

## OUTPUT

```
IPython 8.15.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/TIPQC/.spyder-py3/temp.py', wdir='C:/Users/TIPQC/.spyder-py3')

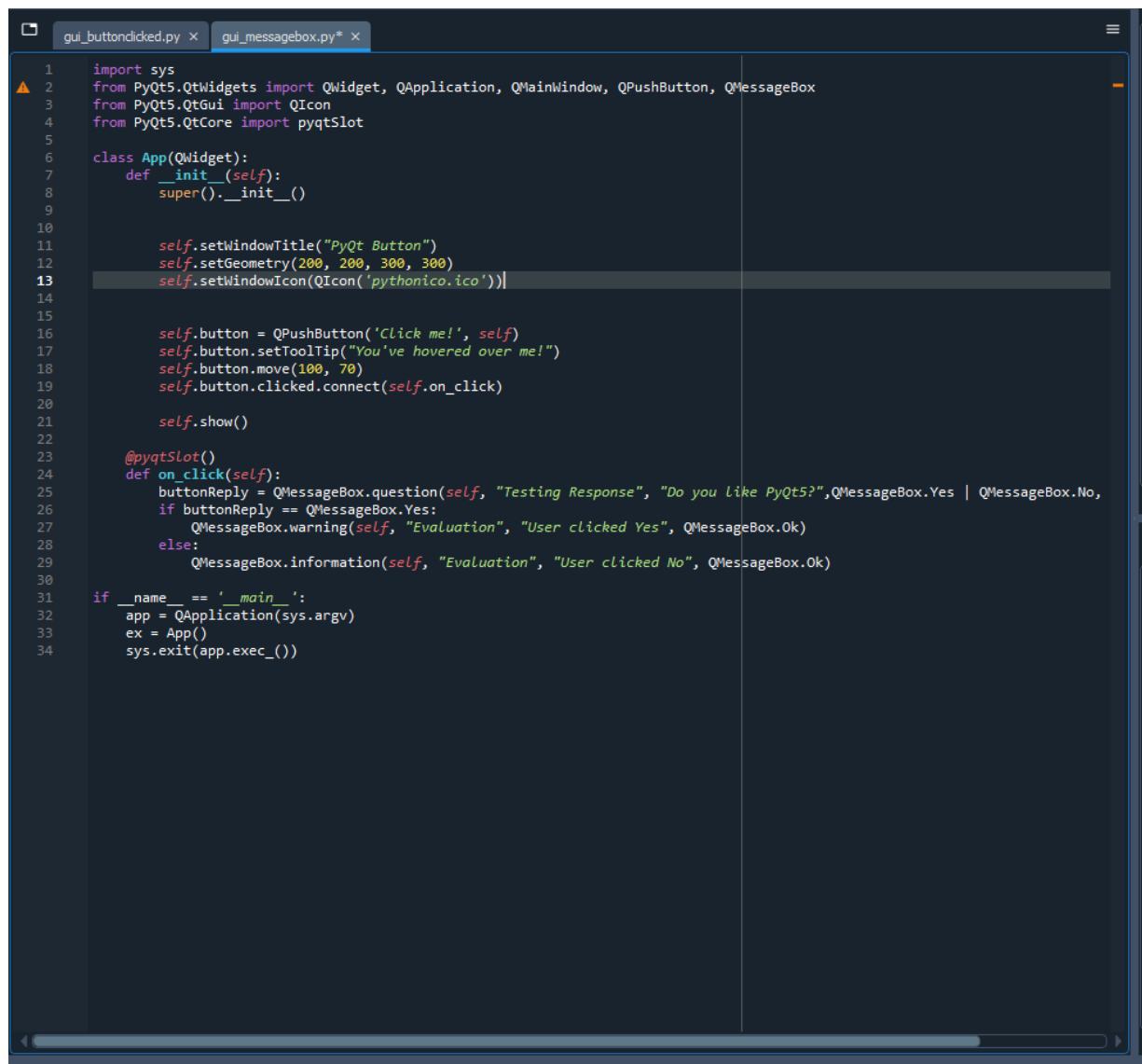
In [2]: runfile('C:/Users/TIPQC/Downloads/Lab5/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Downloads/Lab5')
Button clicked!

In [3]: runfile('C:/Users/TIPQC/untitled0.py', wdir='C:/Users/TIPQC')

In [4]: runfile('C:/Users/TIPQC/Downloads/Lab5/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Downloads/Lab5')
Button clicked!

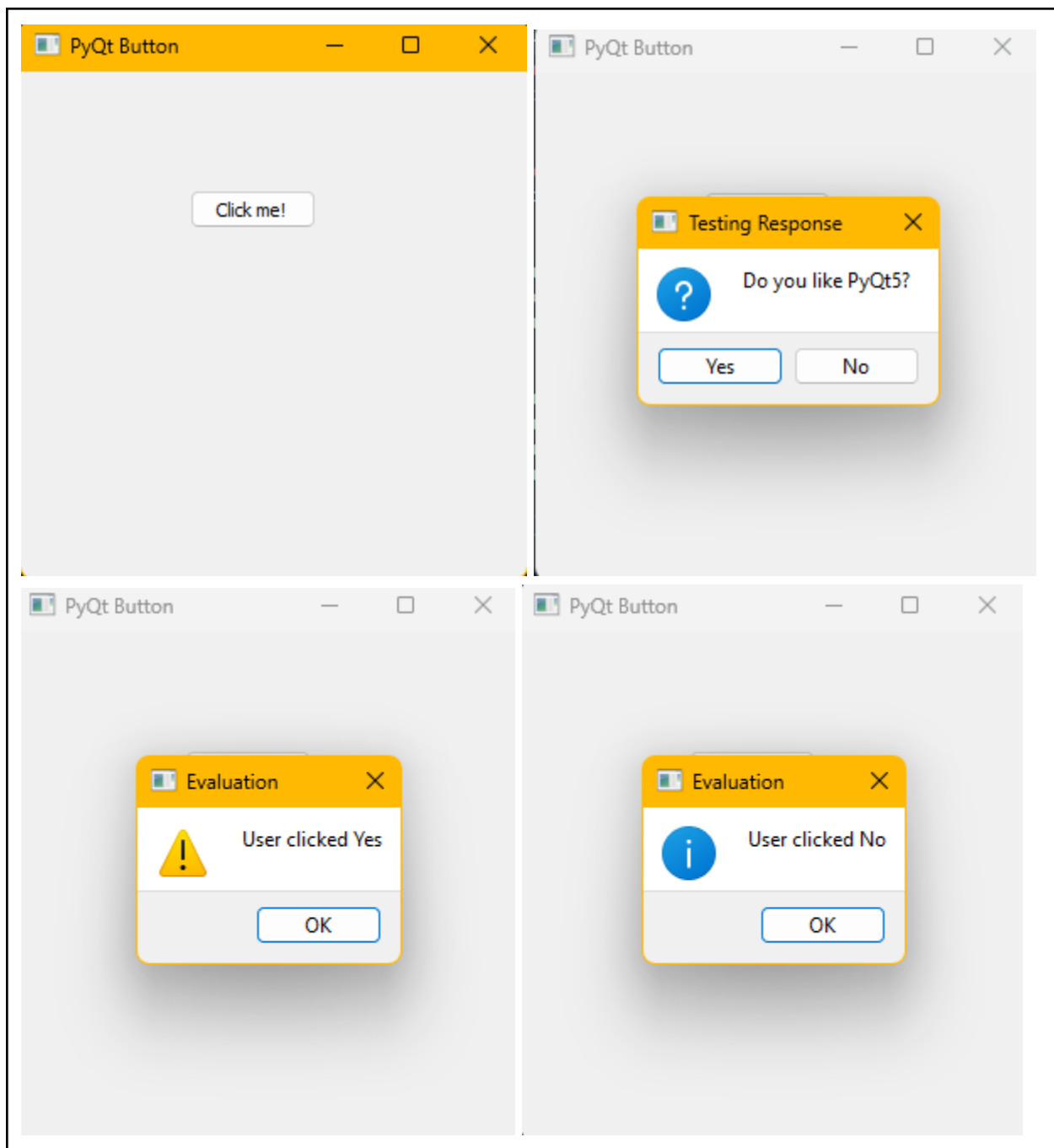
In [5]:
```

## gui\_messagebox.py PROCEDURE

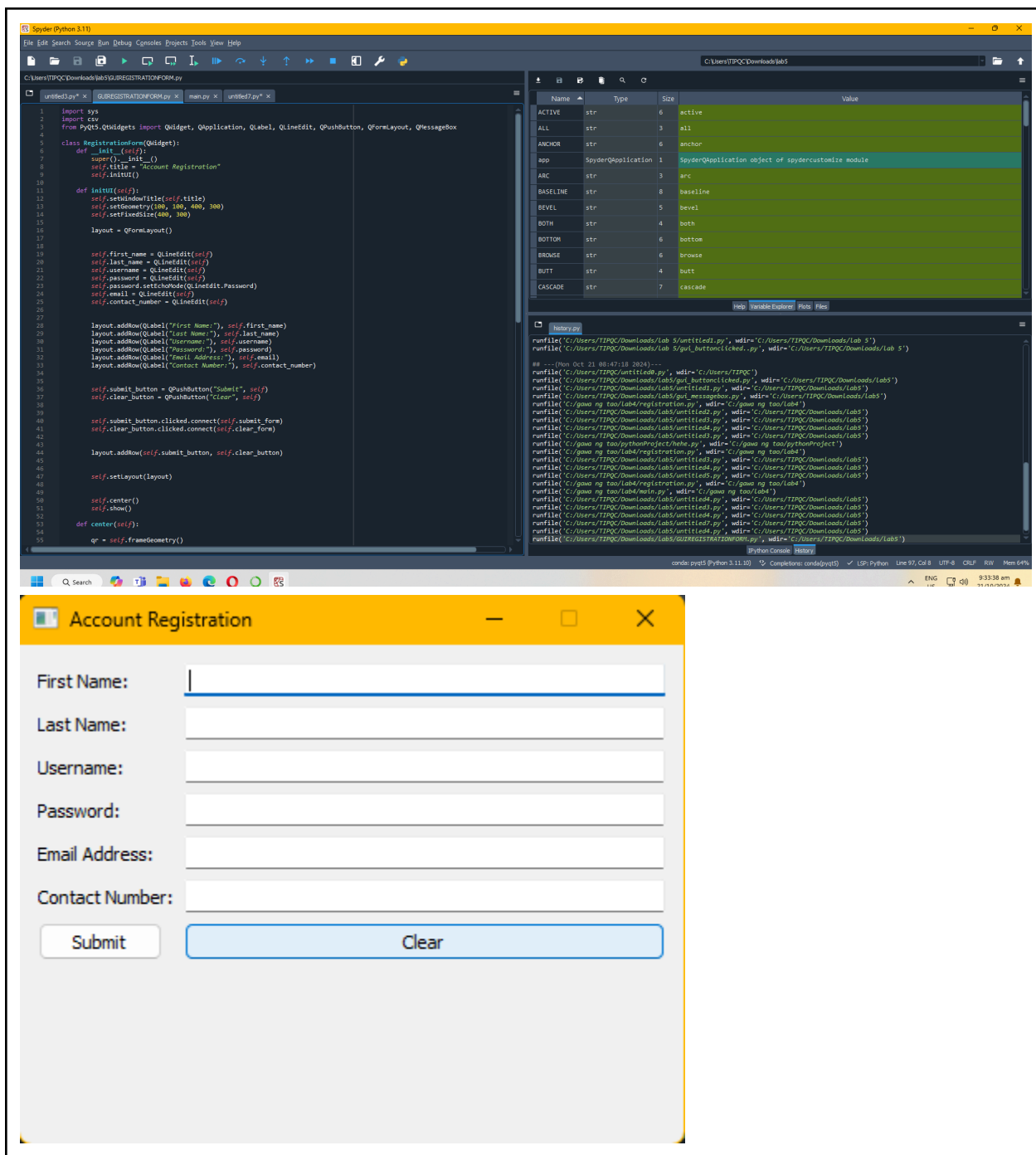


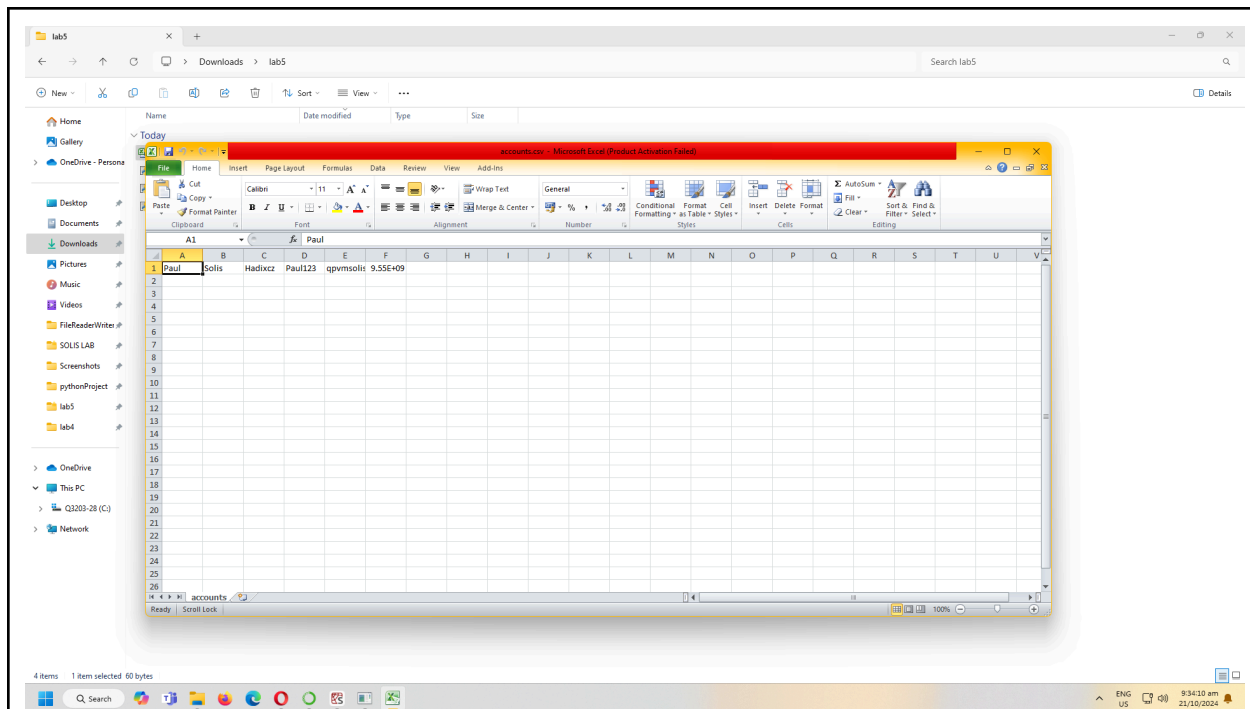
```
1 import sys
2 from PyQt5.QtWidgets import QWidget, QApplication, QMainWindow, QPushButton, QMessageBox
3 from PyQt5.QtGui import QIcon
4 from PyQt5.QtCore import pyqtSlot
5
6 class App(QWidget):
7     def __init__(self):
8         super().__init__()
9
10
11         self.setWindowTitle("PyQt Button")
12         self.setGeometry(200, 200, 300, 300)
13         self.setWindowIcon(QIcon('pythonico.ico'))
14
15
16         self.button = QPushButton('Click me!', self)
17         self.button.setToolTip("You've hovered over me!")
18         self.button.move(100, 70)
19         self.button.clicked.connect(self.on_click)
20
21         self.show()
22
23     @pyqtSlot()
24     def on_click(self):
25         buttonReply = QMessageBox.question(self, "Testing Response", "Do you Like PyQt5?", QMessageBox.Yes | QMessageBox.No,
26         if buttonReply == QMessageBox.Yes:
27             QMessageBox.warning(self, "Evaluation", "User clicked Yes", QMessageBox.Ok)
28         else:
29             QMessageBox.information(self, "Evaluation", "User clicked No", QMessageBox.Ok)
30
31 if __name__ == '__main__':
32     app = QApplication(sys.argv)
33     ex = App()
34     sys.exit(app.exec_())
```

## OUTPUT



**SUPPLEMENTARY ACITIVITY:**  
**OUTPUT:**





### Questions:

1. What are the other signals available in PyQt5? (give at least 3 and describe each)
  - clicked: emitted when a button is clicked, often used to trigger actions or functions.
  - textChanged: emitted when the text in a QLineEdit or QTextEdit widget is modified, useful for validating or reacting to user input.
  - currentTextChanged: emitted when the current text in a QComboBox or QLineEdit widget is changed, often used to update other widgets or perform actions based on the selected text.
2. Why do you think that event handling in Python is divided into signals and slots?
  - Event handling in Python is divided into signals and slots to promote a clear and organized way of managing events and their responses. Signals represent events emitted by objects, while slots are the functions that respond to these events. This decoupling allows for a flexible design, where different components can communicate without being tightly bound to each other, facilitating easier maintenance and scalability in GUI applications. Overall, this architecture enhances code readability and modularity.
3. How can message boxes be used to provide a better User Experience or how can message boxes be used to make a GUI Application more user-friendly?
  - Message boxes can improve user experience in a GUI application by providing clear information and feedback to users. They can alert users to important events, confirm actions, or warn about potential issues, making interactions more intuitive. Additionally, using message boxes to ask for user confirmation before critical actions helps prevent mistakes. Overall, they serve as helpful communication tools that guide users through the application effectively.
4. What is Error-handling and how was it applied in the task performed?
  - Error management is a technique for maintaining a program's smooth operation by managing errors or issues. When something goes wrong during a task, it assists in identifying faults and displays a helpful notice rather than crashing the entire program. Users will find it easier to comprehend what went wrong and how to remedy it as a result.
5. What maybe the reasons behind the need to implement error handling?

- Error management is essential because it keeps programs from crashing or ceasing to function by identifying and fixing errors. When something goes wrong, it helps users understand what to do next by displaying helpful messages, which makes the application safer and easier for them to use.

**CONCLUSION:**

In conclusion, programs become nicer and easier to use when they incorporate message boxes and error handling. While message boxes provide users with clear instructions and information, error handling identifies errors and maintains the program's functionality. When used in tandem, they enhance users' overall experience and provide them greater confidence when utilizing the program.