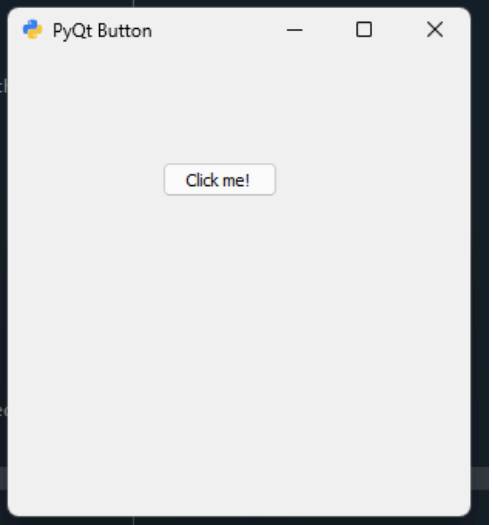
Activity Name # 5 - Introduction to Event Handling in GUI Development	
Name: Virtucio, DOminic Joseph P. Virtucio	10/21/24
Course/Section: OOP 009B / CPE 21S4	Ma'am Sayo

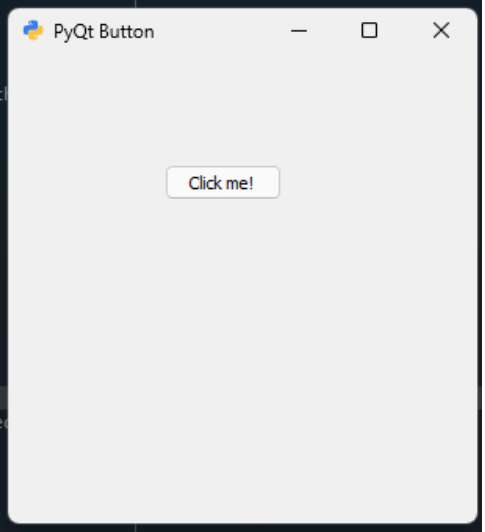
Output:

Event Handling:

```python
import sys
from PyQt5.QtWidgets import QWidget, QApplication, QMainWindow, QPushButton
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import pyqtSlot

class App(QWidget):
    def __init__(self):
        super().__init__()  # initializes the main window like in th
        # window = QMainWindow()
        self.title = "PyQt Button"
        self.x = 200  # or left
        self.y = 200  # or top
        self.width = 300
        self.height = 300
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.x, self.y, self.width, self.height)
        self.setWindowIcon(QIcon('pythonico.ico'))

        # In GUI Python, these buttons, textboxes, labels are calle
        self.button = QPushButton('Click me!', self)
        self.button.setToolTip("You've hovered over me!")
        self.button.move(100, 70)  # button.move(x,y)

        self.show()


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



```python
import sys
from PyQt5.QtWidgets import QWidget, QApplication, QMainWindow, QPushButton
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import pyqtSlot

class App(QWidget):
    def __init__(self):
        super().__init__()  # initializes the main window like in t
        # window = QMainWindow()
        self.title = "PyQt Button"
        self.x = 200  # or left
        self.y = 200  # or top
        self.width = 300
        self.height = 300
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.x, self.y, self.width, self.height)
        self.setWindowIcon(QIcon('pythonico.ico'))

        # In GUI Python, these buttons, textboxes, labels are calle
        self.button = QPushButton('Click me!', self)
        self.button.setToolTip("You've hovered over me!")
        self.button.move(100, 70)  # button.move(x,y)
        self.button.clicked.connect(self.on_click)

        self.show()

    @pyqtSlot()
    def on_click(self):
        print('You clicked me!')

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

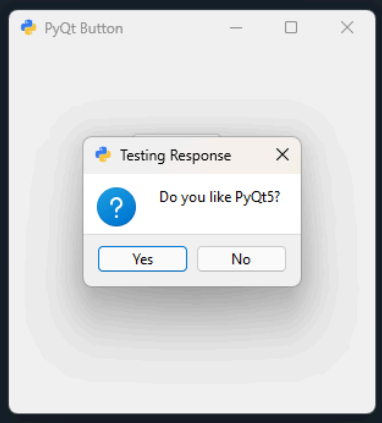


Output at the terminal:

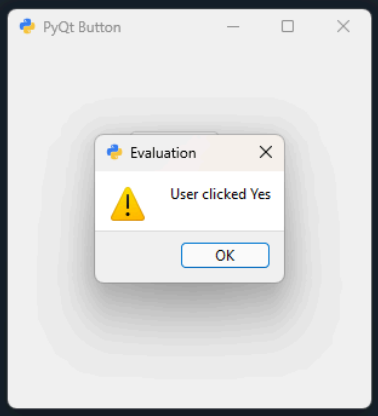
```
In [49]: runfile('C:/Users/tipqc.Q3203-06/Downloads/
cpe009fa1_Virtucio_DominicJoseph_lab5/untitled6.py',
wdir='C:/Users/tipqc.Q3203-06/Downloads/
cpe009fa1_Virtucio_DominicJoseph_lab5')
You clicked me!
You clicked me!
You clicked me!
```

Adding a message box:

```python
import sys
from PyQt5.QtWidgets import QWidget, QApplication, QMainWindow, QPushButton
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMessageBox

class App(QWidget):
    def __init__(self):
        super().__init__()  # initializes the main window like in the previous one
        # window = QMainWindow()
        self.title = "PyQt Button"
        self.x = 200  # or left
        self.y = 200  # or top
        self.width = 300
        self.height = 300
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.x, self.y, self.width, self.height)
        self.setWindowIcon(QIcon('pythonico.ico'))

        # In GUI Python, these buttons, textboxes, labels are called Widgets
        self.button = QPushButton('Click me!', self)
        self.button.setToolTip("You've hovered over me!")
        self.button.move(100, 70)  # button.move(x,y)
        self.button.clicked.connect(self.on_click)

        self.show()

    @pyqtSlot()
    def on_click(self):
        print('You clicked me!')
        self.clickMe()

    @pyqtSlot()
    def clickMe(self):
        buttonReply = QMessageBox.question(self, "Testing Response", "Do you like PyQt5?", QMessageBox.Yes | QMessageBox.No, QMessageBox.Yes)
        if buttonReply == QMessageBox.Yes:
            QMessageBox.warning(self, "Evaluation", "User clicked Yes", QMessageBox.Ok, QMessageBox.Ok)
        else:
            QMessageBox.information(self, "Evaluation", "User clicked No", QMessageBox.Ok, QMessageBox.Ok)

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



```python
import sys
from PyQt5.QtWidgets import QWidget, QApplication, QMainWindow, QPushButton
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMessageBox

class App(QWidget):
    def __init__(self):
        super().__init__()  # initializes the main window like in the previous one
        # window = QMainWindow()
        self.title = "PyQt Button"
        self.x = 200  # or left
        self.y = 200  # or top
        self.width = 300
        self.height = 300
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.x, self.y, self.width, self.height)
        self.setWindowIcon(QIcon('pythonico.ico'))

        # In GUI Python, these buttons, textboxes, labels are called Widgets
        self.button = QPushButton('Click me!', self)
        self.button.setToolTip("You've hovered over me!")
        self.button.move(100, 70)  # button.move(x,y)
        self.button.clicked.connect(self.on_click)

        self.show()

    @pyqtSlot()
    def on_click(self):
        print('You clicked me!')
        self.clickMe()

    @pyqtSlot()
    def clickMe(self):
        buttonReply = QMessageBox.question(self, "Testing Response", "Do you like PyQt5?", QMessageBox.Yes | QMessageBox.No, QMessageBox.Yes)
        if buttonReply == QMessageBox.Yes:
            QMessageBox.warning(self, "Evaluation", "User clicked Yes", QMessageBox.Ok, QMessageBox.Ok)
        else:
            QMessageBox.information(self, "Evaluation", "User clicked No", QMessageBox.Ok, QMessageBox.Ok)

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



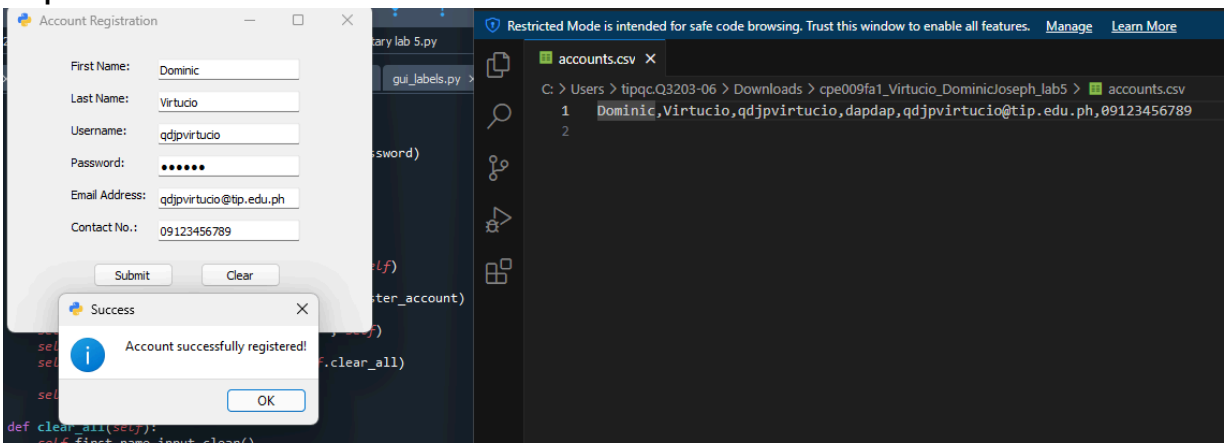
Supplementary Activity

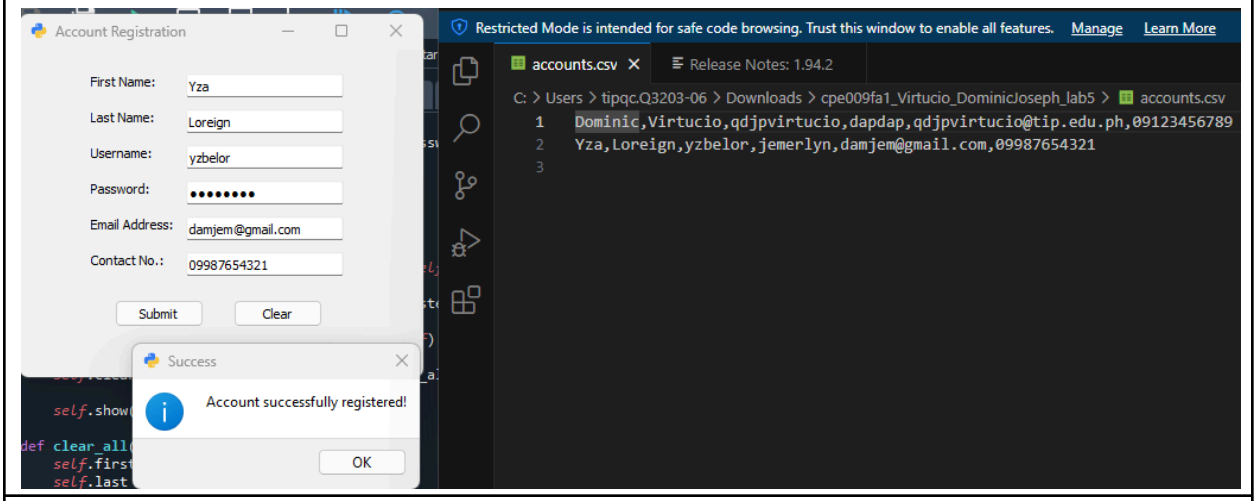
Python

```python
import sys
from PyQt5.QtWidgets import QWidget, QApplication, QLabel, QLineEdit, QPushButton, QMessageBox
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import pyqtSlot
import csv

class RegistrationForm(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Account Registration")
        self.setGeometry(100, 100, 340, 275)
        self.setWindowIcon(QIcon('pythonico.ico'))
        self.initUI()

    def initUI(self):
        # Labels
        self.first_name_label = QLabel("First Name:", self)
        self.first_name_label.move(60, 20)
        self.last_name_label = QLabel("Last Name:", self)
        self.last_name_label.move(60, 50)
        self.username_label = QLabel("Username:", self)
        self.username_label.move(60, 80)
        self.password_label = QLabel("Password:", self)
        self.password_label.move(60, 110)
        self.email_label = QLabel("Email Address:", self)
        self.email_label.move(60, 140)
```

```
        self.contact_label = QLabel("Contact No.:", self)
        self.contact_label.move(60, 170)

        # Text Fields
        self.first_name_input = QLineEdit(self)
        self.first_name_input.move(140, 20)
        self.last_name_input = QLineEdit(self)
        self.last_name_input.move(140, 50)
        self.username_input = QLineEdit(self)
        self.username_input.move(140, 80)
        self.password_input = QLineEdit(self)
        self.password_input.move(140, 110)
        self.password_input.setEchoMode(QLineEdit.Password)
        self.email_input = QLineEdit(self)
        self.email_input.move(140, 140)
        self.contact_input = QLineEdit(self)
        self.contact_input.move(140, 170)

        # Buttons
        self.submit_button = QPushButton("Submit", self)
        self.submit_button.move(80, 210)
        self.submit_button.clicked.connect(self.register_account)

        self.clear_button = QPushButton("Clear", self)
        self.clear_button.move(180, 210)
        self.clear_button.clicked.connect(self.clear_all)

        self.show()

    def clear_all(self):
        self.first_name_input.clear()
        self.last_name_input.clear()
        self.username_input.clear()
        self.password_input.clear()
        self.email_input.clear()
        self.contact_input.clear()

    @pyqtSlot()
    def register_account(self):
        first_name = self.first_name_input.text()
        last_name = self.last_name_input.text()
        username = self.username_input.text()
        password = self.password_input.text()
        email = self.email_input.text()
        contact = self.contact_input.text()

        if not all([first_name, last_name, username, password, email, contact]):
            QMessageBox.warning(self, "Error", "Please fill in all fields.")
            return

        try:
            with open("accounts.csv", "a", newline="") as csvfile:
                writer = csv.writer(csvfile)
                writer.writerow([first_name, last_name, username, password, email, contact])

            QMessageBox.information(self, "Success", "Account successfully registered!")
            self.clear_all()

        except Exception as e:
            QMessageBox.critical(self, "Error", f"Registration failed: {e}")

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

Output:





Questions:

1. **What are the other signals available in PyQt5? (give at least 3 and describe each)**

- The following are the other signals seen in PyQt5 and in the conducted laboratory above:

a) clicked(): This signal is emitted when a button is clicked; it is frequently used in QPushButton, QToolButton, and other button-like widgets. You can see it in the activities above. This signal can be connected to a slot that initiates an action, like launching a new window or submitting a form.

b) textChanged(): this signal is used, or will function, if the user emits text in a way that is largely visible in the activity above, which is evident in the way that QLineEdit is implemented, as demonstrated by my input during account registration.

c) valueChanged(): When a QSpinBox, QSlider, or comparable widget's value changes, this signal is released. It presents the reasoning for the new value. It can be used to instantly refresh a display or carry out computations based on the updated value.

2. **Why do you think that event handling in Python is divided into signals and slots?**

- PyQt5 divides event handling into signals and slots since they serve distinct purposes. Signals serve as alerts, notifying my application when anything happens in the interface, such as when a button is clicked or text is placed into an input field. This makes it easier for the program to remain responsive to user inputs. On the other side, slots are the functions that handle these

notifications. When a signal is sent, the associated slot activates and performs the required tasks, such as refreshing the display or processing user input.

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?**

- When users do activities, message boxes provide rapid feedback, verifying whether the activity was successful or unsuccessful. They can also notify users about prospective difficulties, which can help to avoid blunders. In a lab activity, for example, a message box was utilized to affirm the user's preference for PyQt5. Depending on the user's reaction, another message box appears, indicating whether they clicked "Yes" or "No." This type of interaction helps the program appear more responsive and user-friendly.

4. **What is Error-handling and how was it applied in the task performed?**

- Error handling is an important part of software development that entails anticipating and addressing potential problems or exceptions that may arise during program execution. In the lab, error handling is applied in the account registration task. We used Empty field validation, which verifies if all fields are filled before attempting to store the account data. If any of the fields are left blank, an error message box appears, instructing the user to fill them out completely.

5. **What maybe the reasons behind the need to implement error handling?**

- Error handling is an important component of stable and dependable apps because it keeps users informed and stops programs from crashing. Error management was essential to the lab activity because it prevented inaccurate or incomplete data from being saved, ensured data integrity, and made the user experience seamless by preventing crashes and providing users with helpful error messages.

Conclusion:

This activity helped me get more familiar with event handling in a PyQt5 GUI application. I gained knowledge on how to handle button clicks and changes in user input using signals and slots to create an interactive and responsive interface. I obtained practical experience in linking signals to slots, verifying user input, and giving unambiguous feedback through message boxes—all of which improve user experience—by building a basic account registration system.