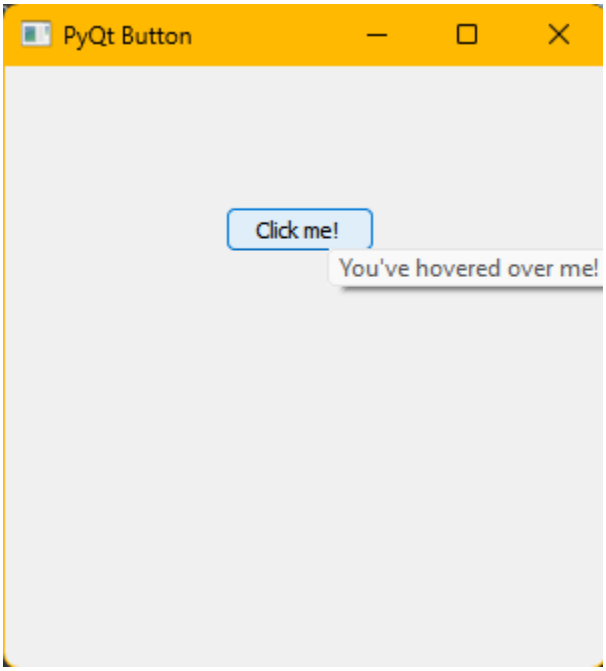
Laboratory Activity No. 5 - Introduction to Event Handling in GUI Development	
Sanchez, Christan Ray R.	21/10/2024
CPE 009B - CPE21S4	Prof. Ma. Rizette Sayo

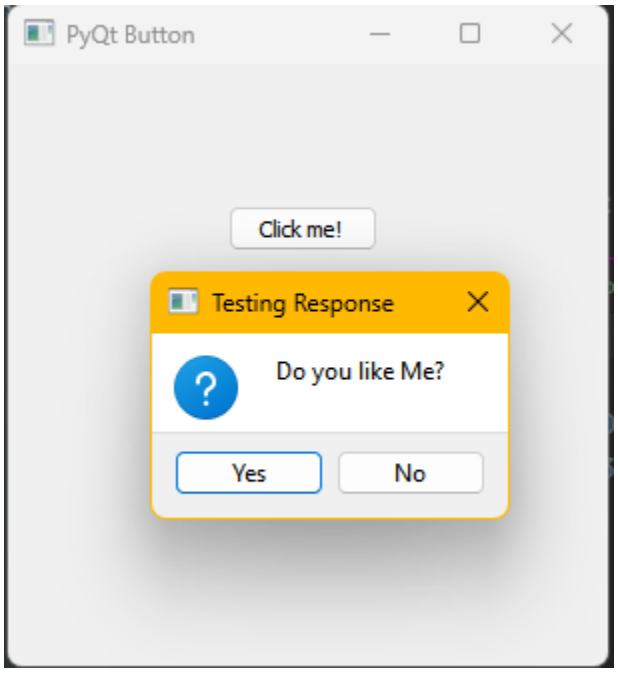
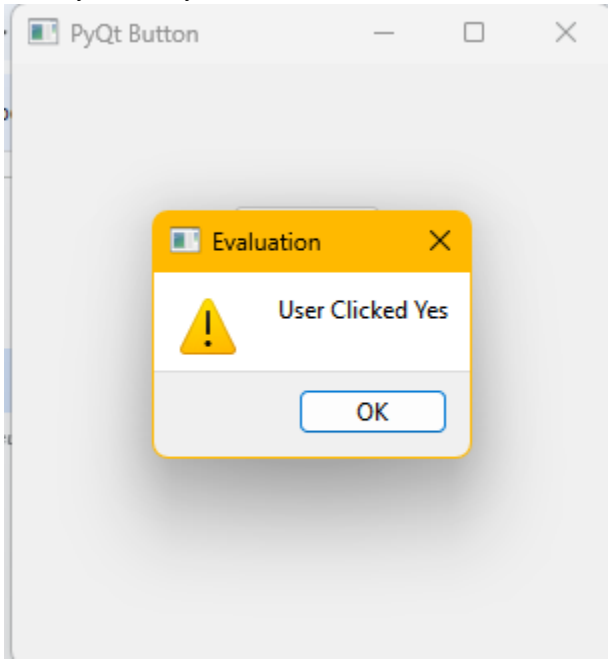
TABLE FOR THE 1ST AND 2ND PART OF THE PROCEDURE	
SOURCE	

```
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__()
        self.title = "PyQt Button"
        self.x = 200
        self.y = 200
        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'))

        self.button = QPushButton('Click me!',self)
        self.button.setToolTip("You've hovered over me!")
        self.button.move(110,70)
        self.button.clicked.connect(self.on_click)
        self.show()

    @pyqtSlot()
    def on_click(self):
        print('You touch my tralala')

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

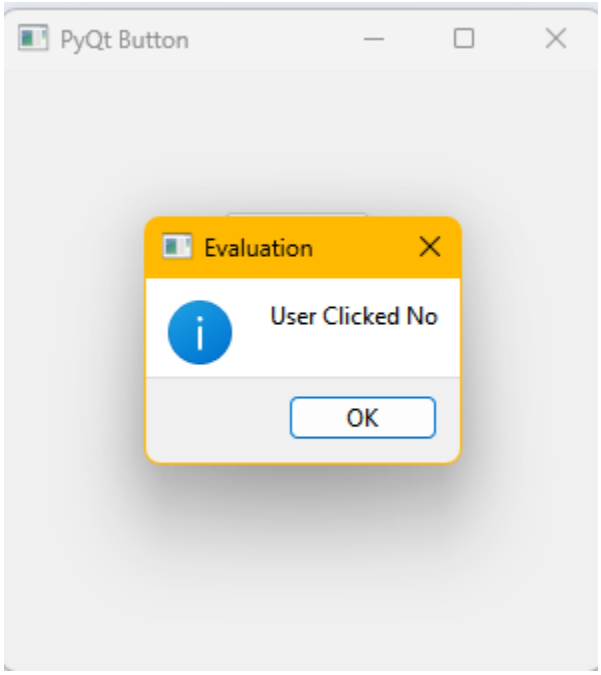
OUTPUT	When hovered over:  When the button is pushed: `You touch my tralala`
OBSERVATIONS	The setToolTip method provides the user with information about what the button will do, which is why it is called a 'Tip. The pyqtSlot is used to connect a function to a button. In this example, it prints out my chosen words because that is the function I assigned to it.

Last part of the procedure which is adding a message box	
SOURCE	

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

class App(QWidget):

    def __init__(self):
        super().__init__()
        self.title = "PyQt Button"
        self.x = 200
        self.y = 200
        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'))

        self.button = QPushButton('Click me!', self)
        self.button.setToolTip("You've hovered over me!")
        self.button.move(110, 70)
        self.button.clicked.connect(self.clickMe)
        self.show()

    @pyqtSlot()
    def clickMe(self):
        buttonReply = QMessageBox.question(self, "Testing
Response", "Do you like Me?",
                                           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_())
```

OUTPUT	When the button is clicked:  When you click yes/ 

When you clicked No:



OBSERVATIONS

II added a message box with two options, allowing the user to choose either 'Yes' or 'No.' This is useful for confirming important actions.

A message box helps ensure that the user does not make mistakes when pressing buttons. basically it provides a more detailed information for the use of the button

SUPPLEMENTARY ACTIVITY

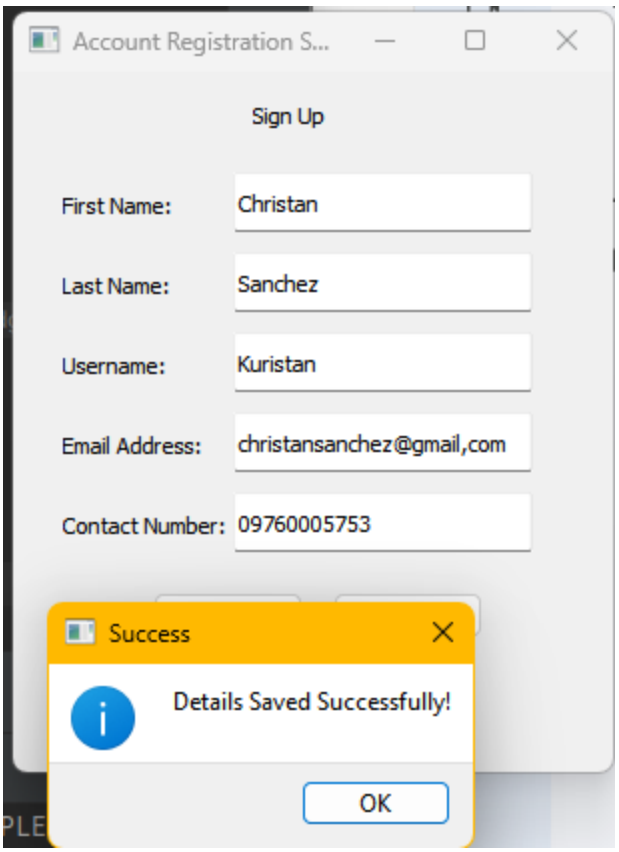
SOURCE CODE

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

class App(QWidget):
    def __init__(self):
        super().__init__()
        self.title = "Account Registration System"
        self.x = 200
        self.y = 200
        self.width = 300
        self.height = 350  # Adjusted for better fit
```

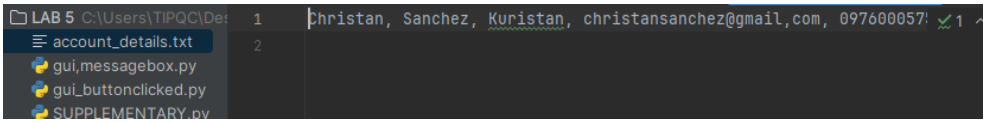
```
        self.initUI()

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

        self.textboxbl = QLabel("Sign Up", self)
        self.textboxbl.move(120, 15)

        # First Name
        self.textboxbl2 = QLabel("First Name: ", self)
        self.textboxbl2.move(25, 60)
        self.firstNameInput = QLineEdit(self)
        self.firstNameInput.move(110, 50)
        self.firstNameInput.resize(150, 30)

        # Last Name
        self.textboxbl3 = QLabel("Last Name: ", self)
        self.textboxbl3.move(25, 100)
        self.lastNameInput = QLineEdit(self)
        self.lastNameInput.move(110, 90)
        self.lastNameInput.resize(150, 30)

        # Username
        self.textboxbl4 = QLabel("Username: ", self)
        self.textboxbl4.move(25, 140)
        self.usernameInput = QLineEdit(self)
        self.usernameInput.move(110, 130)
        self.usernameInput.resize(150, 30)

        # Email Address
        self.textboxbl5 = QLabel("Email Address: ", self)
        self.textboxbl5.move(25, 180)
        self.emailInput = QLineEdit(self)
        self.emailInput.move(110, 170)
        self.emailInput.resize(150, 30)

        # Contact Number
        self.textboxbl6 = QLabel("Contact Number: ", self)
        self.textboxbl6.move(25, 220)
        self.contactInput = QLineEdit(self)
        self.contactInput.move(110, 210)
        self.contactInput.resize(150, 30)

        # Buttons
        self.submitButton = QPushButton("Submit", self)
        self.submitButton.move(70, 260)

self.submitButton.clicked.connect(self.save_account_details)

        self.clearButton = QPushButton("Clear", self)
        self.clearButton.move(160, 260)
        self.clearButton.clicked.connect(self.clear_fields)
```

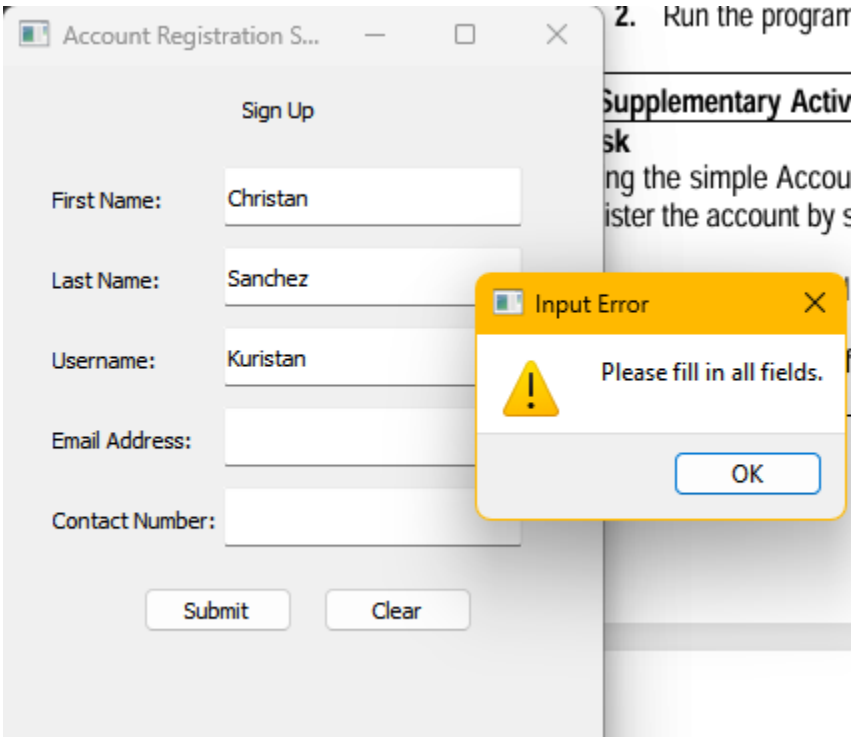
```
        self.center()
        self.show()

    def center(self):
        # Centers the window on the screen
        qr = self.frameGeometry()
        cp =
QApplication.desktop().availableGeometry().center()
        qr.moveCenter(cp)
        self.move(qr.topLeft())

    def save_account_details(self):
        details = [
            self.firstNameInput.text(),
            self.lastNameInput.text(),
            self.usernameInput.text(),
            self.emailInput.text(),
            self.contactInput.text()
        ]

        # Check if all fields are filled
        if any(not detail for detail in details):
            QMessageBox.warning(self, "Input Error", "Please
fill in all fields.")
            return

        with open('account_details.txt', 'a') as f:
            f.write(', '.join(details) + '\n')
        QMessageBox.information(self, "Success", "Details Saved
Successfully!")

    def clear_fields(self):
        self.firstNameInput.clear()
        self.lastNameInput.clear()
        self.usernameInput.clear()
        self.emailInput.clear()
        self.contactInput.clear()
        QMessageBox.information(self, "Cleared", "Fields
Cleared Successfully!")

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

OUTPUT



it made a text file containing the details I inputted



	It made sure that all details must be filled out 
OBSERVATIONS	The supplementary activity is similar to the last one, but it helped me understand more about using event handling in GUI development I have finished the necessary functions, such as confirming details and saving the details in a .txt file or .csv file. The most crucial part of the activity is the event handling, which ensures that the user cannot submit the details unless all the textboxes are filled out."

Questions:

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

clicked - as performed in the procedure and the supplementary activity it is used to trigger a function when a user interacts with a button

textChanged - it is emitted when a text in the line edit is changed.

released - is when the button is released after being pushed by the user.

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

To be more efficient and flexible. Signals work when events occur, and slots are functions that handle these events. This will help your code to be more responsive to user interactions

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?

Since not many people are proficient in using technology, this can help provide a more detailed explanation of what the buttons can do. This will help secure the worries of the user if their input is wrong or not or when something is not filled out.

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

Error handling is used to manage errors during a program's execution and prevent crashes. While the task performed doesn't specifically focus on error handling, it uses message boxes to guide users and prevent mistakes. Additionally, a function was added in the supplementary activity to ensure users cannot proceed without filling out the required text boxes, further enhancing user experience and preventing errors.

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

Implementing error handling is essential to prevent programs from crashing unexpectedly, ensuring a smoother user experience. It also guides users by providing clear feedback when something goes wrong, helping them understand and correct their mistakes.

CONCLUSION

This lab activity helped me understand how event handling works in GUI development using PyQt5. I learned how to connect signals and slots, making the app interactive and responsive. Adding tooltips and message boxes showed me how important it is to give clear instructions and feedback to the user.

I also realized that error handling is key to making applications easy to use and reliable. By catching mistakes, we can stop the app from crashing and help users fix their input.

Overall, this activity taught me valuable skills in building GUIs and handling events, which are essential for creating user-friendly software.