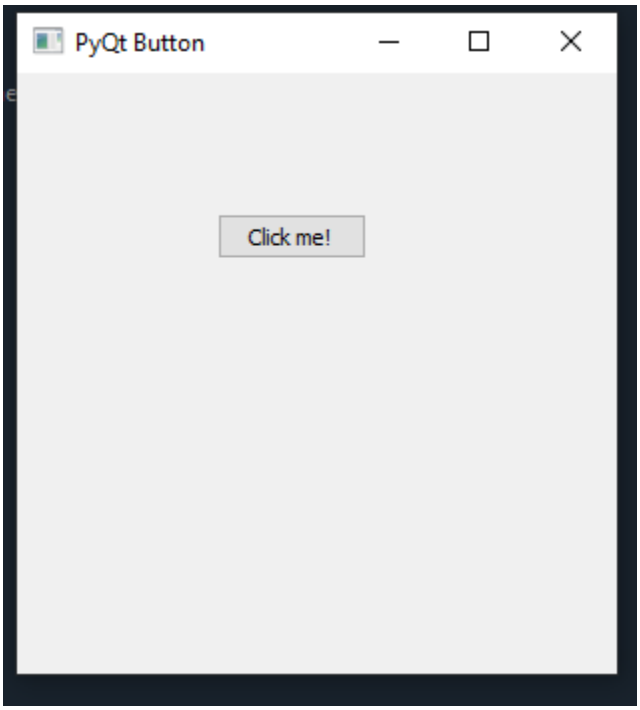
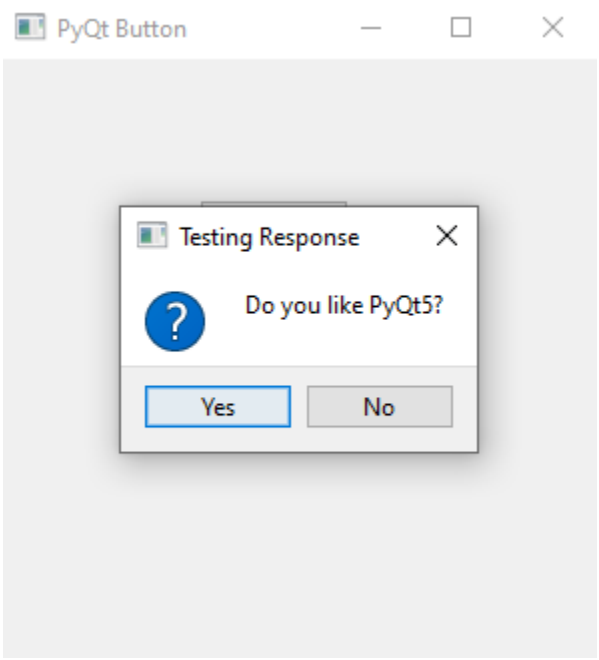
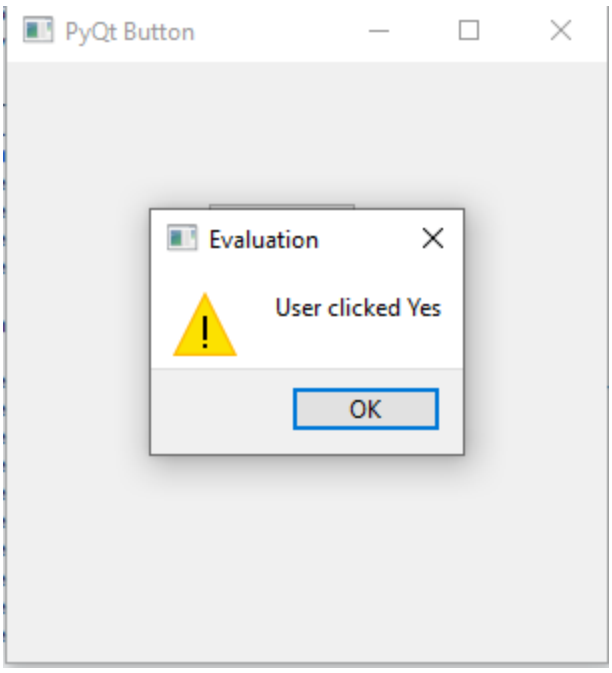
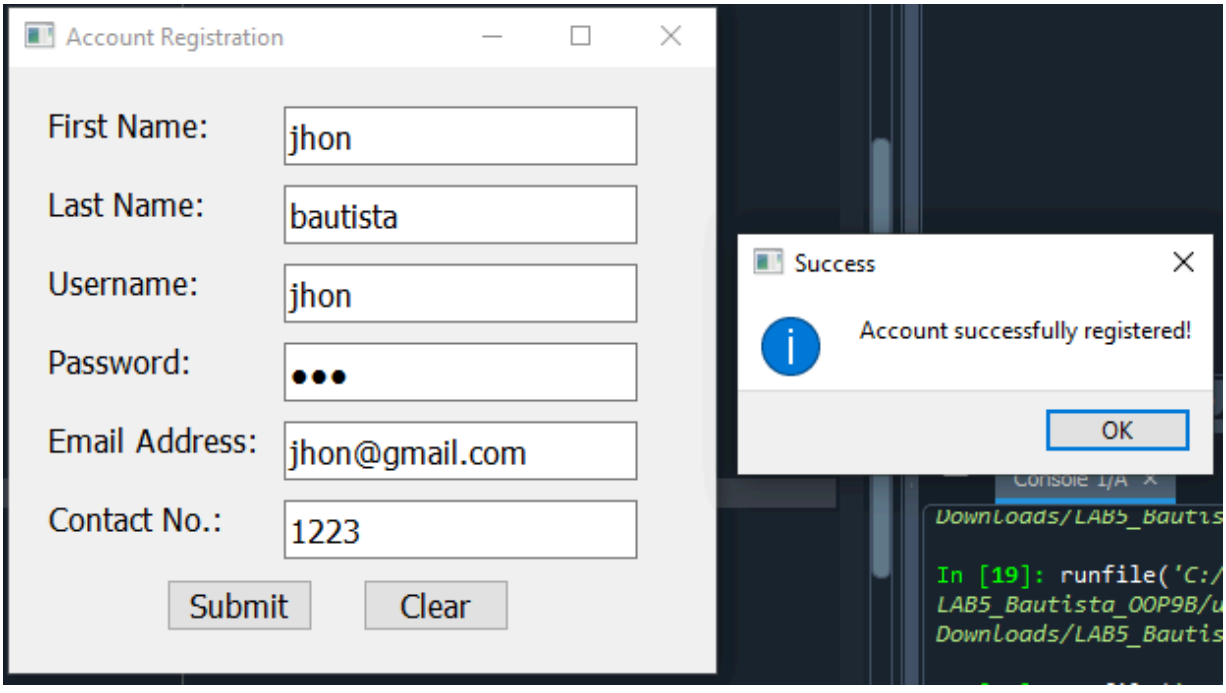
Laboratory Activity 5 - Introduction to Event Handling in GUI Development	
Bautista, Jhon Hendricks T.	10/29/2024
CPE 009B / CPE21S1	Mrs. Maria Rizette Sayo







Supplementary Activity



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

class RegistrationForm(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Account Registration")
        self.setGeometry(200, 200, 360, 300)
        self.initUI()

    def initUI(self):
        font = QFont()
        font.setPointSize(12)

        self.first_name_label = QLabel("First Name:", self)
        self.first_name_label.setFont(font)
        self.first_name_label.move(20, 20)

        self.last_name_label = QLabel("Last Name:", self)
        self.last_name_label.setFont(font)
        self.last_name_label.move(20, 60)

        self.username_label = QLabel("Username:", self)
        self.username_label.setFont(font)
        self.username_label.move(20, 100)

        self.password_label = QLabel("Password:", self)
        self.password_label.setFont(font)
        self.password_label.move(20, 140)

        self.email_label = QLabel("Email Address:", self)
        self.email_label.setFont(font)
        self.email_label.move(20, 180)

        self.contact_label = QLabel("Contact No.:", self)
        self.contact_label.setFont(font)
        self.contact_label.move(20, 220)


        self.first_name_input = QLineEdit(self)
        self.first_name_input.setFont(font)
        self.first_name_input.move(140, 20)
        self.first_name_input.resize(180, 30)

        self.last_name_input = QLineEdit(self)
        self.last_name_input.setFont(font)
        self.last_name_input.move(140, 60)
        self.last_name_input.resize(180, 30)
```

```
        self.first_name_input = QLineEdit(self)
        self.first_name_input.setFont(font)
        self.first_name_input.move(140, 20)
        self.first_name_input.resize(180, 30)

        self.last_name_input = QLineEdit(self)
        self.last_name_input.setFont(font)
        self.last_name_input.move(140, 60)
        self.last_name_input.resize(180, 30)

        self.username_input = QLineEdit(self)
        self.username_input.setFont(font)
        self.username_input.move(140, 100)
        self.username_input.resize(180, 30)

        self.password_input = QLineEdit(self)
        self.password_input.setFont(font)
        self.password_input.move(140, 140)
        self.password_input.setEchoMode(QLineEdit.Password)
        self.password_input.resize(180, 30)

        self.email_input = QLineEdit(self)
        self.email_input.setFont(font)
        self.email_input.move(140, 180)
        self.email_input.resize(180, 30)

        self.contact_input = QLineEdit(self)
        self.contact_input.setFont(font)
        self.contact_input.move(140, 220)
        self.contact_input.resize(180, 30)

        # Buttons
        self.submit_button = QPushButton("Submit", self)
        self.submit_button.setFont(font)
        self.submit_button.move(80, 260)
        self.submit_button.clicked.connect(self.register_account)

        self.clear_button = QPushButton("Clear", self)
        self.clear_button.setFont(font)
        self.clear_button.move(180, 260)
        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
```

```
    @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_())
```

1. What are the other signals available in PyQt5? (give at least 3 and describe each)
   - clicked() - A signal is sent when a widget is clicked and is commonly known for triggering a new window.
   - textChanged() - When a user modifies the text in a text-editing widget (such as a QLineEdit or QTextEdit), this signal is released.
   - hover() - A signal is sent when you hover the mouse on a widget. It is frequently used to alter the widget's look, such as altering the color of a button

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

The division allows the system to have modular design where we can link slots into different signals and signals from different sources. The related slot activates and completes the necessary action when a signal is sent. Even though they are divided, they work together in order to create a good user experience.

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 in GUI applications provide users with important feedback by providing notifications. They can also alert consumers to potential challenges, which can aid in preventing errors. The message box used to provide

clear feedback and verification of users based on their actions. This can help in giving the user an easy time using the UI.

4. What is Error-handling and how was it applied in the task performed?
Error handling is a procedure used to deal with unexpected program behavior or runtime issues. It guarantees that problems may be handled by the software without crashing, giving the user feedback and preserving system stability. The program utilized validation for empty fields,which checks that every field is filled out before trying to save the account information.

5. What maybe the reasons behind the need to implement error handling?
A program's ability to handle errors is crucial for preventing crashes and generating inaccurate results. It's an essential component of programming that aids developers in foreseeing and resolving possible problems before they arise. With this, programs can have a better user experience and improve the stability of the program.

Conclusion:

After this laboratory I was able to utilize the PyQt framework in creating a simple UI. I was able to learn the code and syntax for error handling and event handling in Python. I made an understanding of how to use signals and slots to manage button clicks and user input changes to produce an interface that is responsive and interactive. I gained hands-on experience connecting slots to signals to create a simple UI. All of my learnings I implemented it in a simple UI of account registration.