Laboratory Activity 5 - Introduction to Event Handling in GUI Development	
Reyes, Alexzander J. Reyes	21/10/2024
CPE 009B/CPE21S4	Mrs. Maria Rizette Sayo

Procedure:

C:\Users\TIPQC\Downloads\lab5\gui_buttonclicked.py

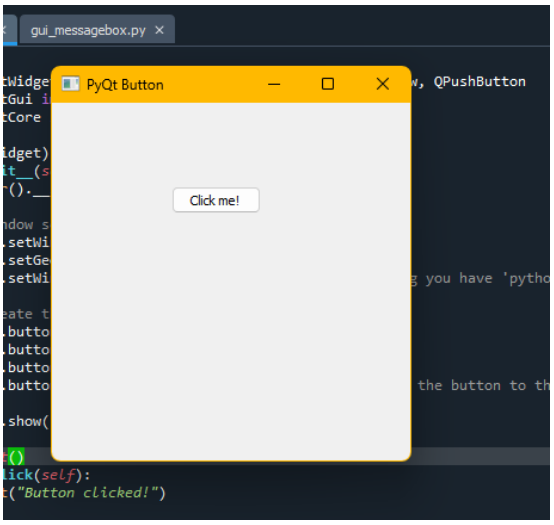
gui_buttonclicked.py × gui_messagebox.py* ×

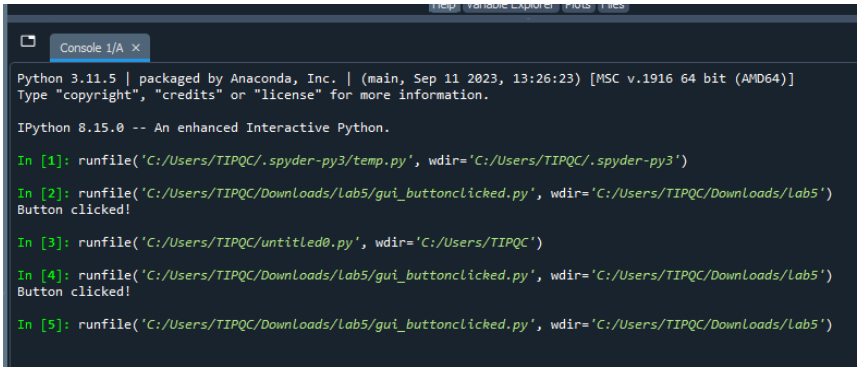
```python
import sys
from PyQt5.QtWidgets import QWidget, QApplication, QMainWindow, QPushButton
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import pyqtSlot

class App(QWidget):
    def __init__(self):
        super().__init__()  # Initialize the main window

        # Window settings
        self.setWindowTitle("PyQt Button")
        self.setGeometry(200, 200, 300, 300)
        self.setWindowIcon(QIcon('pythonico.ico'))  # Assuming you have 'pythonico.ico'

        # Create the button
        self.button = QPushButton('Click me!', self)
        self.button.setToolTip("You've hovered over me!")
        self.button.move(100, 70)
        self.button.clicked.connect(self.on_click)  # Connect the button to the slot

        self.show()

    @pyqtSlot()
    def on_click(self):
        print("Button clicked!")

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

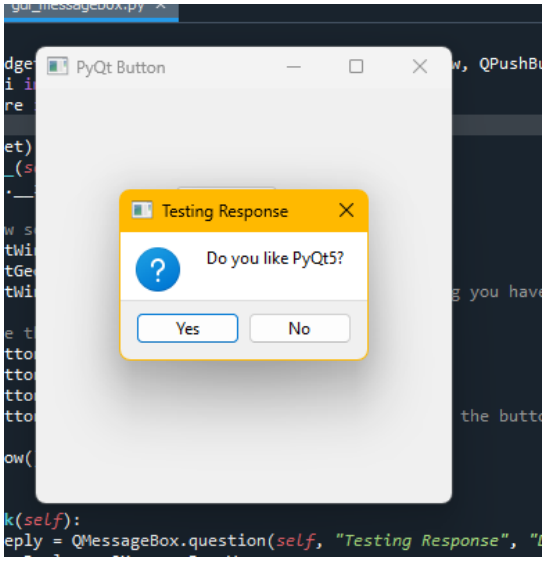
gui_buttonclicked.py × gui_messagebox.py* ×

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

class App(QWidget):
    def __init__(self):
        super().__init__()  # Initialize the main window

        # Window settings
        self.setWindowTitle("PyQt Button")
        self.setGeometry(200, 200, 300, 300)
        self.setWindowIcon(QIcon('pythonico.ico'))  # Assuming you have 'pythonico.ico'

        # Create the button
        self.button = QPushButton('Click me!', self)
        self.button.setToolTip("You've hovered over me!")
        self.button.move(100, 70)
        self.button.clicked.connect(self.on_click)  # Connect the button to the slot

        self.show()

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

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

OUTPUT:


gui_messagebox.py
PyQt Button
Click me!
, QPushButton
you have 'pytho
the button to th
Button clicked!"


Console 1/A
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
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]: runfile('C:/Users/TIPQC/Downloads/lab5/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Downloads/lab5')


PyQt Button
Testing Response
Do you like PyQt5?
Yes
No
, QPushBu
you have
the butto
k(self):
eply = QMessageBox.question(self, "Testing Response",

6. Supplementary Activity:

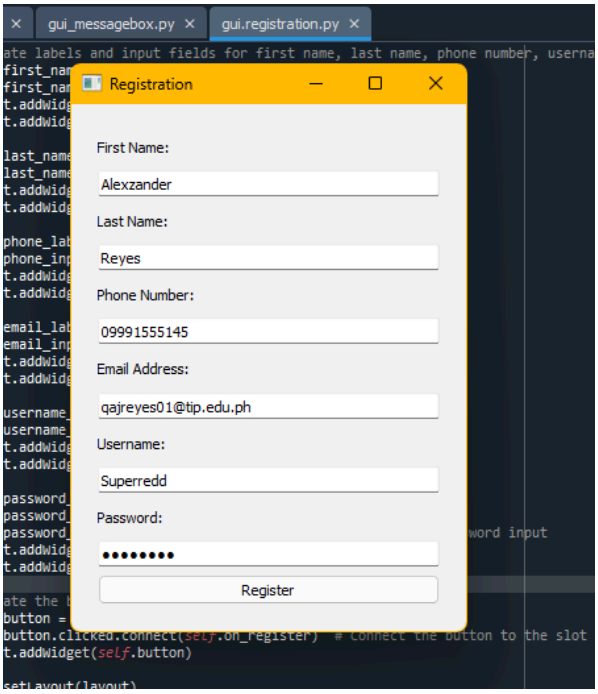
Input:

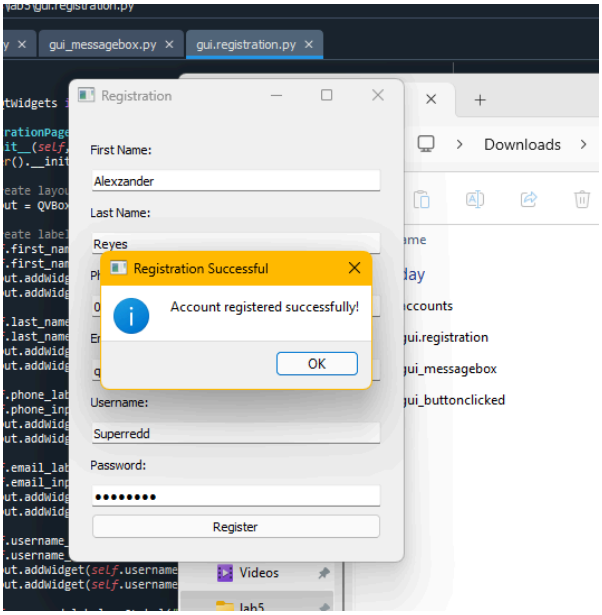
```
import sys
import csv
from PyQt5.QtWidgets import QWidget, QApplication, QPushButton, QMessageBox, QLineEdit, QLabel,
QVBoxLayout

class RegistrationPage(QWidget):
    def __init__(self, parent):
        super().__init__(parent)

        # Create layout for registration page
        layout = QVBoxLayout()

        # Create labels and input fields for first name, last name, phone number, username, password, and
email address
        self.first_name_label = QLabel("First Name:")
        self.first_name_input = QLineEdit(self)
        layout.addWidget(self.first_name_label)
        layout.addWidget(self.first_name_input)

        self.last_name_label = QLabel("Last Name:")
        self.last_name_input = QLineEdit(self)
        layout.addWidget(self.last_name_label)
        layout.addWidget(self.last_name_input)

        self.phone_label = QLabel("Phone Number:")
        self.phone_input = QLineEdit(self)
        layout.addWidget(self.phone_label)
        layout.addWidget(self.phone_input)

        self.email_label = QLabel("Email Address:")
        self.email_input = QLineEdit(self)
        layout.addWidget(self.email_label)
        layout.addWidget(self.email_input)

        self.username_label = QLabel("Username:")
        self.username_input = QLineEdit(self)
        layout.addWidget(self.username_label)
        layout.addWidget(self.username_input)

        self.password_label = QLabel("Password:")
        self.password_input = QLineEdit(self)
        self.password_input.setEchoMode(QLineEdit.Password)  # Hide password input
```

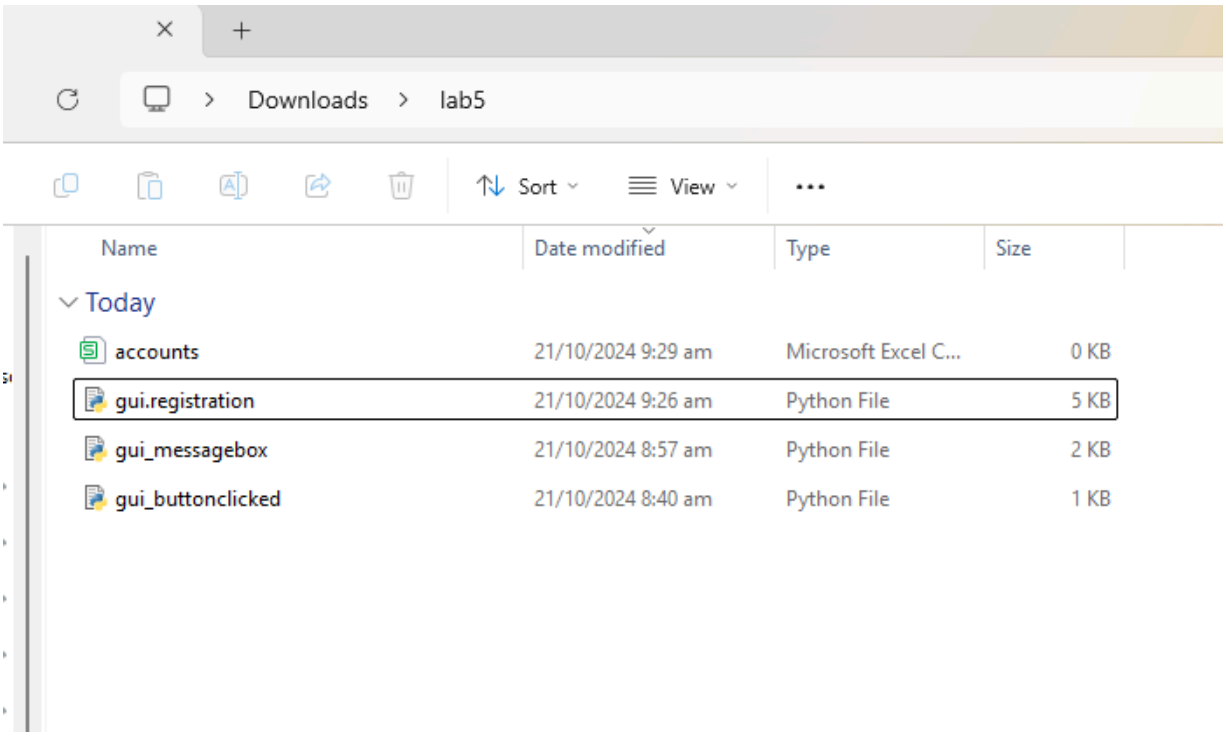
```
        layout.addWidget(self.password_label)
        layout.addWidget(self.password_input)

        # Create the button to register
        self.button = QPushButton('Register', self)
        self.button.clicked.connect(self.on_register)  # Connect the button to the slot
        layout.addWidget(self.button)

        self.setLayout(layout)

    def on_register(self):
        # Retrieve input values
        first_name = self.first_name_input.text().strip()
        last_name = self.last_name_input.text().strip()
        phone = self.phone_input.text().strip()
        username = self.username_input.text().strip()
        password = self.password_input.text().strip()
        email = self.email_input.text().strip()

        # Validate input
        if not (first_name and last_name and phone and username and password and email):
            QMessageBox.warning(self, "Input Error", "Please fill in all fields.", QMessageBox.Ok)
            return

        # Save to CSV file
        try:
            with open('accounts.csv', mode='a', newline='') as file:
                writer = csv.writer(file)
                # Check if the username already exists
                existing_users = []
                try:
                    with open('accounts.csv', mode='r') as existing_file:
                        existing_users = [row[0] for row in csv.reader(existing_file)]
                except FileNotFoundError:
                    pass  # File doesn't exist, no users to check

                if username in existing_users:
                    QMessageBox.warning(self, "Registration Error", "Username already exists.",
QMessageBox.Ok)
                else:
                    # Save first name, last name, username, password, phone number, and email address
                    writer.writerow([first_name, last_name, username, password, phone, email])
                    QMessageBox.information(self, "Registration Successful", "Account registered
successfully!", QMessageBox.Ok)

                    # Terminate the application after successful registration
```

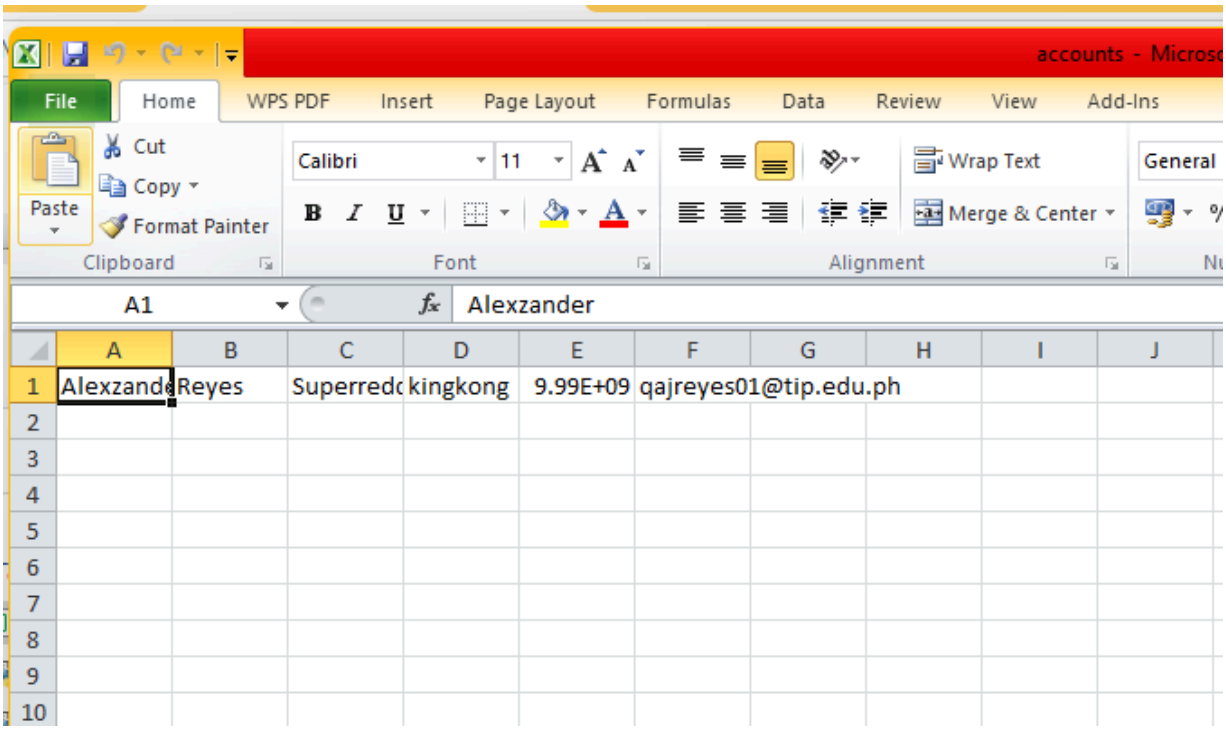
```
                QApplication.quit()
        except Exception as e:
            QMessageBox.critical(self, "Error", f"An error occurred while saving the account: {e}",
QMessageBox.Ok)


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

        # Window settings
        self.setWindowTitle("Registration")
        self.setGeometry(200, 200, 300, 400)

        # Create registration page
        self.registration_page = RegistrationPage(self)

        self.setLayout(QVBoxLayout())
        self.layout().addWidget(self.registration_page)


if __name__ == "__main__":
    app = QApplication(sys.argv)
    main_window = App()
    main_window.show()
    sys.exit(app.exec_()
```

Output:


gui_messagebox.py
gui.registration.py
Registration
First Name:
Alexzander
Last Name:
Reyes
Phone Number:
09991555145
Email Address:
qajreyes01@tip.edu.ph
Username:
Superredd
Password:
Register


gui_messagebox.py
gui.registration.py
Registration
First Name:
Alexzander
Last Name:
Reyes
Registration Successful
Account registered successfully!
OK
Username:
Superredd
Password:
Register
Downloads
Videos


Downloads
lab5
Sort
View
Name
Date modified
Type
Size
Today
accounts
21/10/2024 9:29 am
Microsoft Excel C...
0 KB
gui.registration
21/10/2024 9:26 am
Python File
5 KB
gui_messagebox
21/10/2024 8:57 am
Python File
2 KB
gui_buttonclicked
21/10/2024 8:40 am
Python File
1 KB


accounts - Microsc
File
Home
WPS PDF
Insert
Page Layout
Formulas
Data
Review
View
Add-Ins
Cut
Copy
Format Painter
Paste
Clipboard
Calibri
11
Font
Wrap Text
Merge & Center
Alignment
General
A1
Alexzander
Alexzande
Reyes
Superredc
kingkong
9.99E+09
qajreyes01@tip.edu.ph

Questions:

1. What are the other signals available in PyQt5? (give at least 3 and describe each)
   clicked():
   Description: This signal is emitted by a QPushButton (or any clickable widget) when the button is clicked.
   textChanged():
   Description: This signal is emitted by input fields like QLineEdit or QTextEdit when the text inside them is modified.
   stateChanged(int):
   Description: This signal is emitted by a QCheckBox when its state changes between checked, unchecked, or partially checked (in the case of tristate checkboxes).

2. Why do you think that event handling in Python is divided into signals and slots?
   The division into **signals and slots** provides a structured, modular, and maintainable way of handling events in an application. It decouples the event source (signal) from the event handler (slot), enabling reusable, flexible, and scalable applications, especially in GUI programming, where various asynchronous events need to be managed efficiently.

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 allow the application to provide users with instant feedback on their actions. For example, after submitting a form or saving a file, a message box can confirm whether the action was successful or if there was an error. This gives users confidence that the action was completed and reduces uncertainty about whether the application functioned correctly.

4. What is Error-handling and how was it applied in the task performed?
   Error-handling is the process of managing and responding to unexpected conditions or problems that arise during the execution of a program. It involves detecting, handling, and recovering from errors to ensure that a program behaves in a controlled and predictable manner, even when something goes wrong.

5. What maybe the reasons behind the need to implement error handling?
   By handling errors appropriately and providing meaningful feedback (e.g., via message boxes), users are informed about the issue in a user-friendly manner, making the software easier to use and less prone to confusion. Also, proper error-handling ensures that the system can recover from issues without losing data or leaving it in an inconsistent state. It may involve rolling back transactions, closing files safely, or prompting users to retry an operation.

7. Conclusion:

Event handling and error management in GUI programming with PyQt5 were the main topics of the activity, which showed how important they are in producing dynamic and intuitive programs. A user registration interface was used to explore important ideas like slots, signals, and input validation. This helped developers construct responsive apps that successfully engage with users by giving them an accurate view of how events are saved and handled.

Additionally, it was emphasized how important error management is to improving program stability by handling unexpected situations, preventing crashes, and giving useful feedback. In addition to enhancing user experience, the program preserves data confidentiality and integrity by utilizing error-handling techniques.

8. Assessment Rubic: