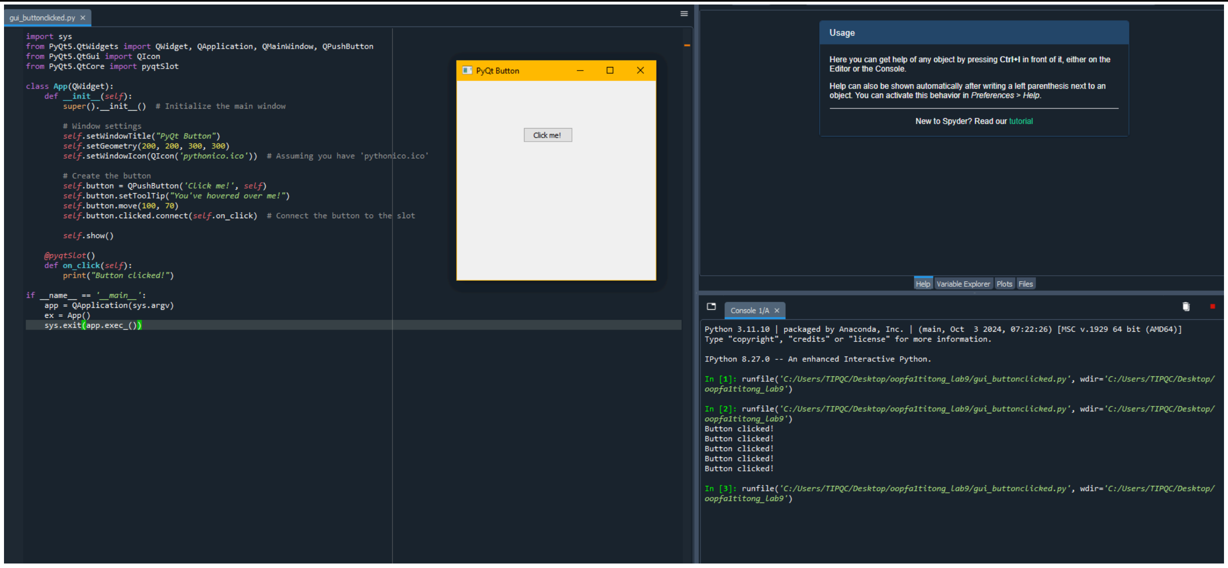
CPE 009 Lab Activity 5 - Introduction to Event Handling in GUI Development.pdf	
Titong, Lee Ivan B.	10/21/2024
CPE21S4	Prof. Maria Rizette Sayo

PROCEDURE
Event Handling

Python

```python
#gui_buttonclicked.py
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_())
```



```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_())
```

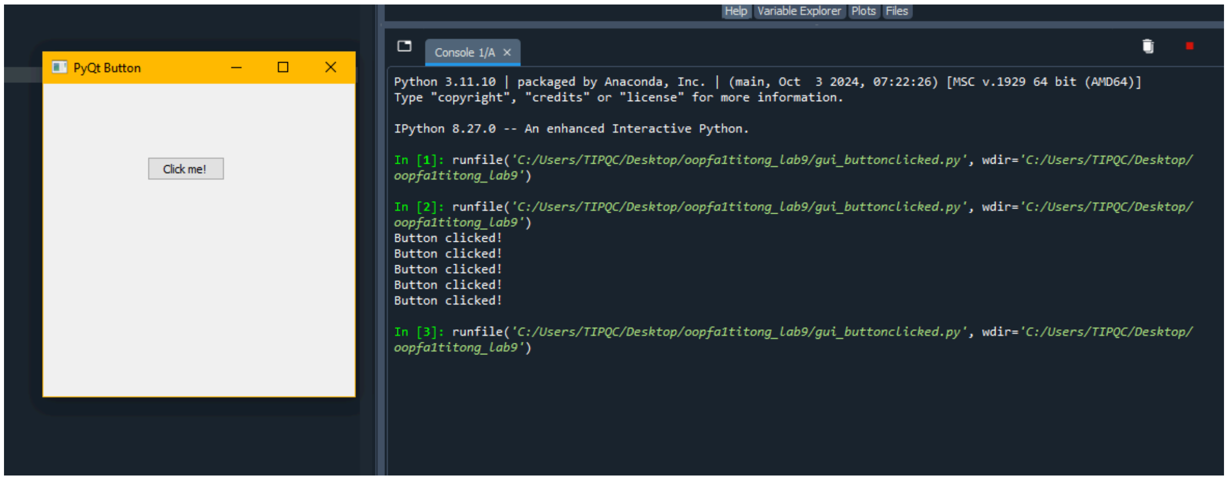
Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

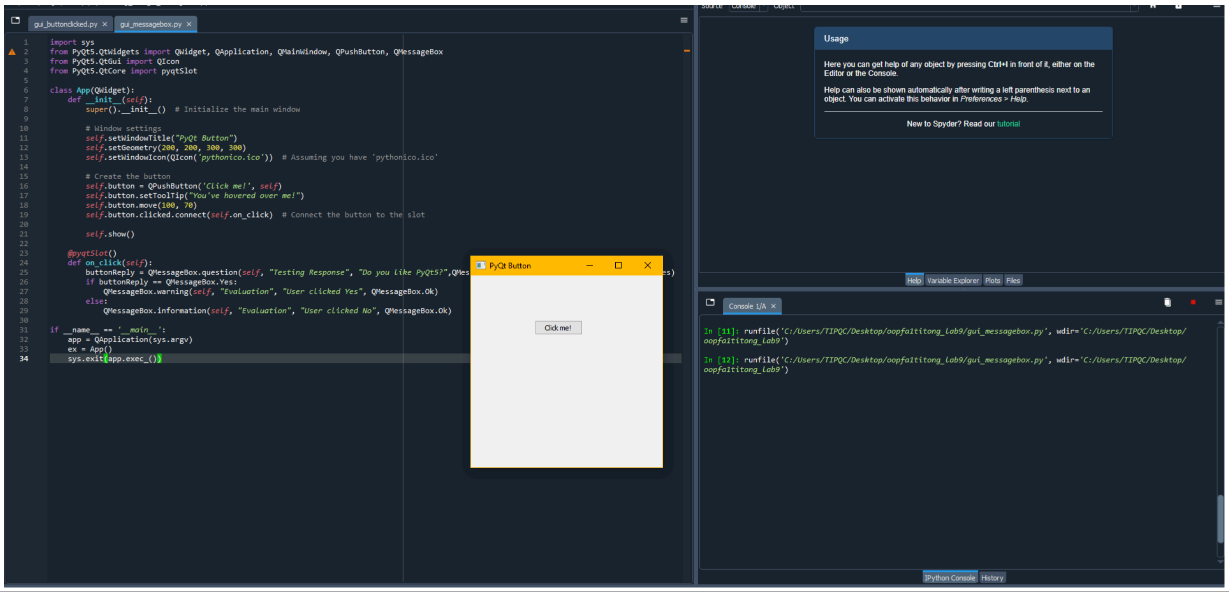
```
Python 3.11.10 | packaged by Anaconda, Inc. | (main, Oct  3 2024, 07:22:26) [MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.27.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Desktop/
oopfa1titong_lab9')

In [2]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Desktop/
oopfa1titong_lab9')
Button clicked!
Button clicked!
Button clicked!
Button clicked!
Button clicked!

In [3]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Desktop/
oopfa1titong_lab9')
```



```
Python 3.11.10 | packaged by Anaconda, Inc. | (main, Oct  3 2024, 07:22:26) [MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.27.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Desktop/
oopfa1titong_lab9')

In [2]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Desktop/
oopfa1titong_lab9')
Button clicked!
Button clicked!
Button clicked!
Button clicked!
Button clicked!

In [3]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/gui_buttonclicked.py', wdir='C:/Users/TIPQC/Desktop/
oopfa1titong_lab9')
```

Adding a Message Box

Python
```python
#gui_messagebox.py
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, QMessageBox.Yes)
        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_())
```



```
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?",QMes                    es)
        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_())
```

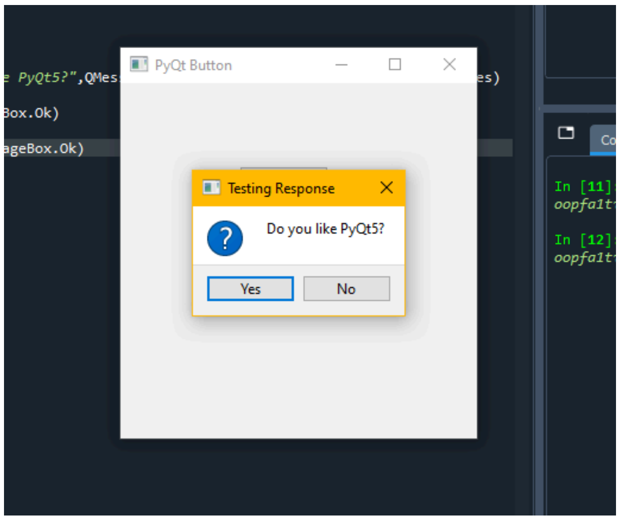
Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
In [11]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/gui_messagebox.py', wdir='C:/Users/TIPQC/Desktop/
oopfa1titong_lab9')

In [12]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/gui_messagebox.py', wdir='C:/Users/TIPQC/Desktop/
oopfa1titong_lab9')
```



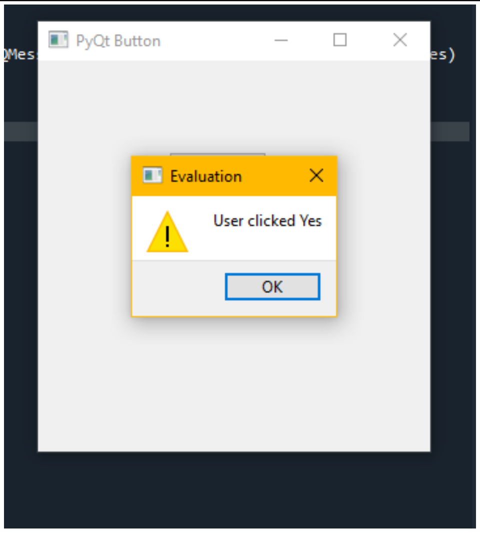
```
e PyQt5?",QMes                         es)
Box.Ok)
ageBox.Ok)
```

```
In [11]
oopfa1t
In [12]
oopfa1t
```

PyQt Button

Testing Response

Do you like PyQt5?



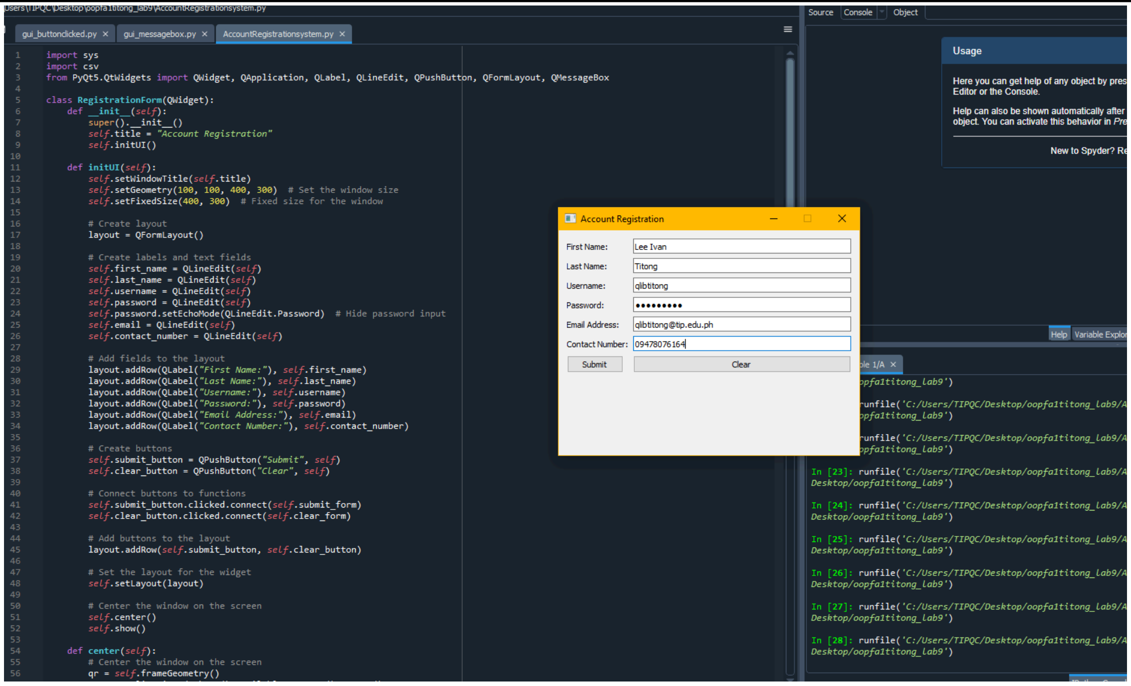
```
QMes                         es)
```

PyQt Button

Evaluation

User clicked Yes

Supplementary Task


Users\TIPQC\Desktop\oopfa1titong_lab9\AccountRegistrationsystem.py
gui_buttonclicked.py
gui_messagebox.py
AccountRegistrationsystem.py
Source
Console
Object
Help
Variable Explor

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

class RegistrationForm(QWidget):
    def __init__(self):
        super().__init__()
        self.title = "Account Registration"
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(100, 100, 400, 300)  # Set the window size
        self.setFixedSize(400, 300)  # Fixed size for the window

        # Create layout
        layout = QFormLayout()

        # Create labels and text fields
        self.first_name = QLineEdit(self)
        self.last_name = QLineEdit(self)
        self.username = QLineEdit(self)
        self.password = QLineEdit(self)
        self.password.setEchoMode(QLineEdit.Password)  # Hide password input
        self.email = QLineEdit(self)
        self.contact_number = QLineEdit(self)

        # Add fields to the layout
        layout.addRow(QLabel("First Name:"), self.first_name)
        layout.addRow(QLabel("Last Name:"), self.last_name)
        layout.addRow(QLabel("Username:"), self.username)
        layout.addRow(QLabel("Password:"), self.password)
        layout.addRow(QLabel("Email Address:"), self.email)
        layout.addRow(QLabel("Contact Number:"), self.contact_number)

        # Create buttons
        self.submit_button = QPushButton("Submit", self)
        self.clear_button = QPushButton("Clear", self)

        # Connect buttons to functions
        self.submit_button.clicked.connect(self.submit_form)
        self.clear_button.clicked.connect(self.clear_form)

        # Add buttons to the layout
        layout.addRow(self.submit_button, self.clear_button)

        # Set the layout for the widget
        self.setLayout(layout)

        # Center the window on the screen
        self.center()
        self.show()

    def center(self):
        # Center the window on the screen
        qr = self.frameGeometry()
```

Usage

Here you can get help of any object by pres
Editor or the Console.

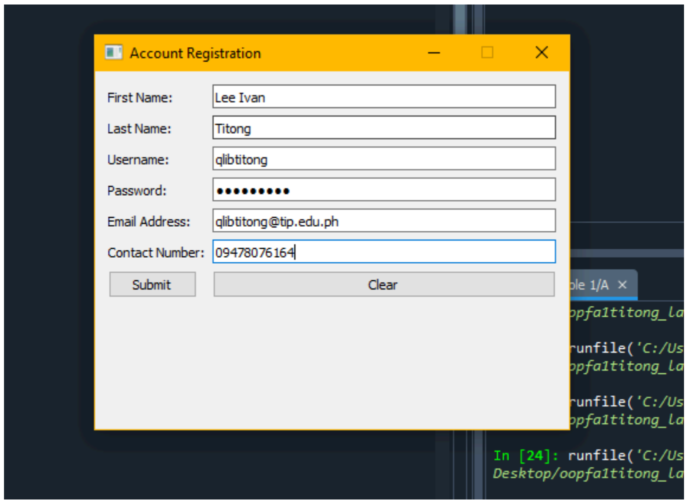
Help can also be shown automatically after
object. You can activate this behavior in *Pre*

New to Spyder? Re

```
ole 1/A
opfa1titong_lab9')

runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/A
opfa1titong_lab9')

runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/A
opfa1titong_lab9')

In [23]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/A
Desktop/oopfa1titong_lab9')

In [24]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/A
Desktop/oopfa1titong_lab9')

In [25]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/A
Desktop/oopfa1titong_lab9')

In [26]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/A
Desktop/oopfa1titong_lab9')

In [27]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/A
Desktop/oopfa1titong_lab9')

In [28]: runfile('C:/Users/TIPQC/Desktop/oopfa1titong_lab9/A
Desktop/oopfa1titong_lab9')
```

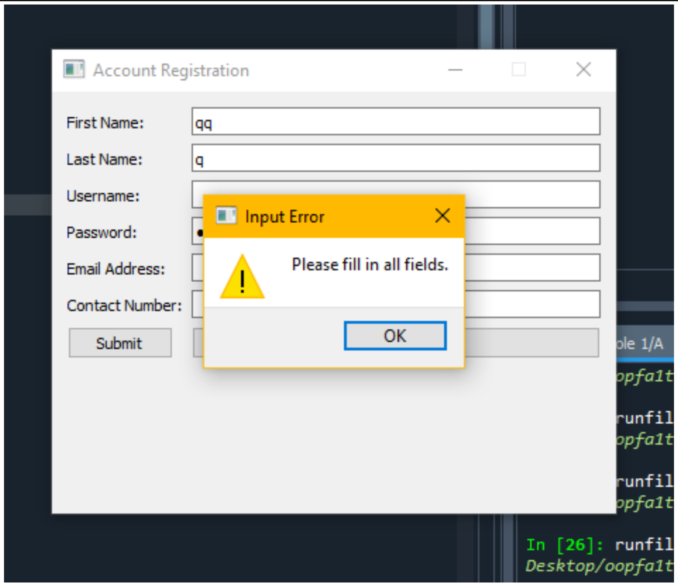
Account Registration

First Name:	Lee Ivan
Last Name:	Titong
Username:	qlibtitong
Password:	•••••••••
Email Address:	qlibtitong@tip.edu.ph
Contact Number:	09478076164


Account Registration

First Name:	Lee Ivan
Last Name:	Titong
Username:	qlibtitong
Password:	•••••••••
Email Address:	qlibtitong@tip.edu.ph
Contact Number:	09478076164

```
ole 1/A
opfa1titong_la
runfile('C:/Us
opfa1titong_la
runfile('C:/Us
opfa1titong_la
In [24]: runfile('C:/Us
Desktop/oopfa1titong_la
```


Account Registration

First Name: qq

Last Name: q

Username:

Password:

Email Address:

Contact Number:

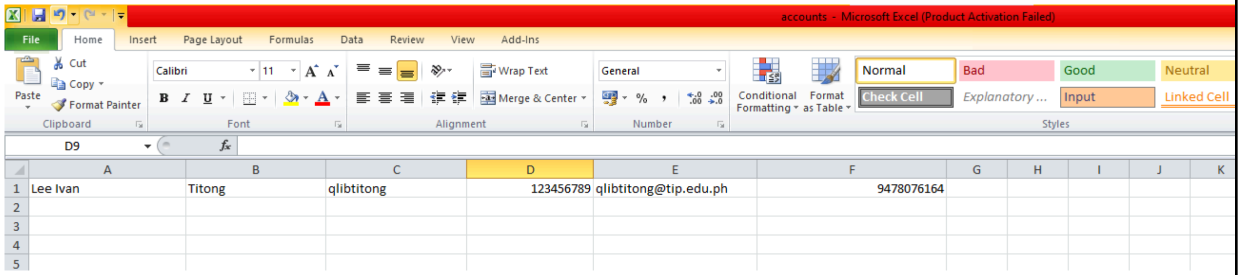
Submit

Input Error

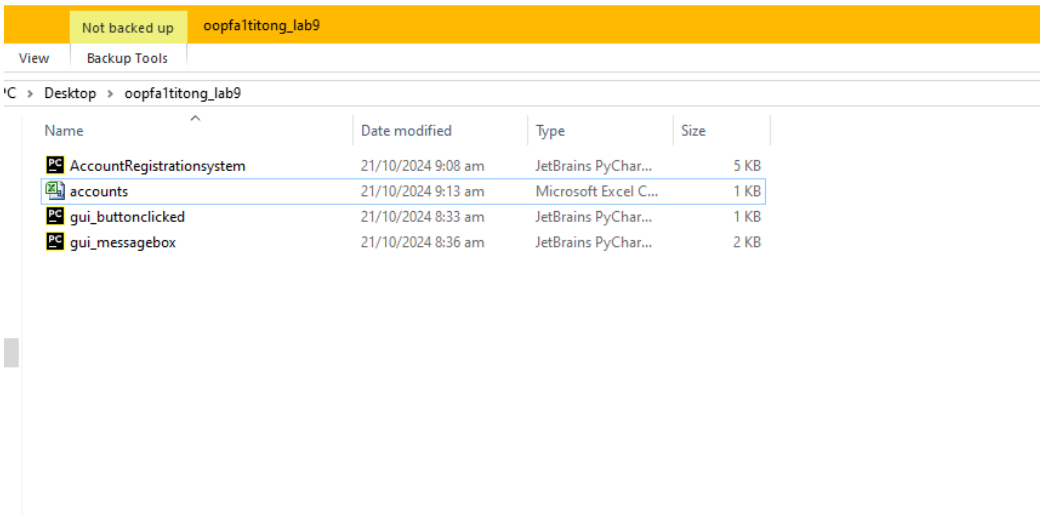
Please fill in all fields.

OK

```
ole 1/A
opfa1t
runfil
opfa1t
runfil
opfa1t
In [26]: runfil
Desktop/oopfa1t
```


accounts - Microsoft Excel (Product Activation Failed)

	A	B	C	D	E	F	G	H	I	J	K
1	Lee Ivan	Titong	qlibtitong	123456789	qlibtitong@tip.edu.ph	9478076164					
2											
3											
4											
5											



Name	Date modified	Type	Size
AccountRegistrationsystem	21/10/2024 9:08 am	JetBrains PyChar...	5 KB
accounts	21/10/2024 9:13 am	Microsoft Excel C...	1 KB
gui_buttonclicked	21/10/2024 8:33 am	JetBrains PyChar...	1 KB
gui_messagebox	21/10/2024 8:36 am	JetBrains PyChar...	2 KB

Questions:

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

- clicked: Triggered when a button is clicked. Used to perform actions like submitting a form.

- textChanged: Triggered when the text in a text box changes. Useful for validating input in real-time.
- currentIndexChanged: Triggered when the selected item in a dropdown changes. Can be used to update other parts of the UI based on the selection.

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

- Loose Coupling: Objects can communicate without knowing about each other, making the code easier to manage.
- Asynchronous Handling: Signals can be emitted without stopping the program, keeping the user interface responsive.
- Flexibility: Multiple actions can be connected to one signal, allowing for complex 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?

- Error Notifications: Inform users about mistakes, helping them correct them easily.
- Confirmation Dialogs: Ask users to confirm actions (like deleting something) to prevent mistakes.
- Informational Messages: Let users know when actions are successful (like successful registration).

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

Error Handling is how programs deal with problems that occur during execution. In the code, it uses try-except blocks to catch errors (like file issues) and show friendly messages instead of crashing.

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

- User Experience: Makes the app easier to use by providing helpful feedback.
- Application Stability: Prevents crashes and keeps the app running smoothly.
- Debugging and Maintenance: Helps find and fix issues more easily.
- Data Integrity: Protects data from being lost or corrupted

Conclusion

In this laboratory activity, I developed a foundational understanding of event handling in Graphical User Interface (GUI) applications using PyQt5. I learned to identify key GUI components and how these elements interact with user actions through signals and slots. By creating a simple account registration system, I practiced validating user input and enhancing user experience with message boxes, which provided essential feedback and error notifications. The activity emphasized the importance of error handling, teaching me how to catch exceptions and deliver meaningful messages to users, thereby ensuring application stability and usability. Overall, this hands-on experience not only reinforced theoretical concepts but also prepared me for more advanced programming tasks and highlighted the significance of user-friendly design in software development.