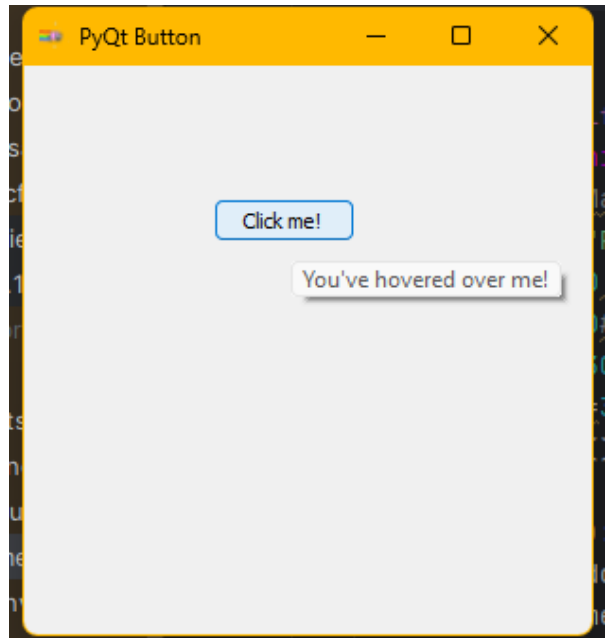
Activity Name #5 - Introduction to Event Handling in GUI Development	
Valleser, Kenn Jie L.	21/10/2024
CPE009B/CPE21S4	Engr. Ma. Rizette Sayo

6. Output	[redacted]
Code	

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

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.wi
dth,self.height)

self.setWindowIcon(QIcon('pythonico.ic
o'))#sets an icon

       #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_cl
ick)
       self.show()
```

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

Screenshot	
Observation	The Code created a window that has button named Click me! and when u hover the mouse is said You've Hovered over me!

Code	```
	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 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.wi
dth,self.height)

self.setWindowIcon(QIcon('pythonico.ic
o'))#sets an icon

        #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_cl
ick)
        self.show()

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


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

Screenshot

```
"C:\Users\TIPQC\PycharmProjects\Laboratory Activity
You clicked me!
You clicked me!
You clicked me!
You clicked me!
You clicked me!

Process finished with exit code -1073740791 (0xC0000
```

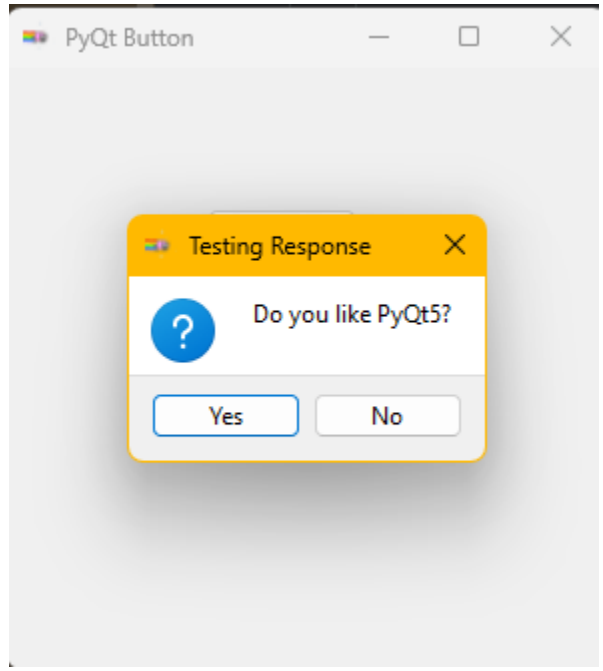
Observation

This time the button does something to the terminal which displays You clicked me!

Code

```
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__()
       # initializes the main window
like in the previous one
       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.ic
o'))  # sets an icon
       # 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_cl
ick)
       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, 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	
Observation	Now, When you click a new window pops up and said Do you like PyQt5? and it has different response according of what choice you make (Yes/No)

7. Supplementary Activity

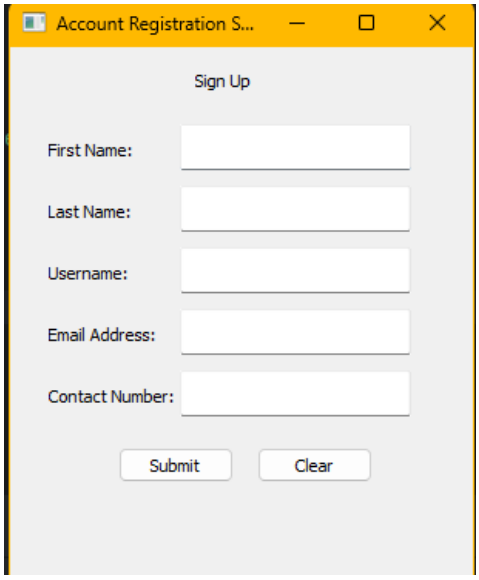
Code	```
	from PyQt5.QtWidgets import QWidget, QApplication, QPushButton, QLineEdit, QLabel, QMessageBox
	from PyQt5.QtGui import QIcon
	import sys
	
	class App(QWidget):
	  def __init__(self):
	  super().__init__()
	  self.title = "Account Registration System"
	  self.x = 200
	  self.y = 200
	  self.width = 300
	  self.height = 350 # Adjusted for better fit
	``

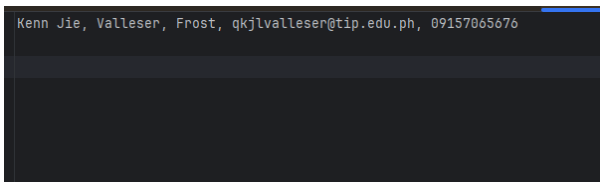
```
        self.initUI()

    def initUI(self):

self.setWindowTitle(self.title)
        self.setGeometry(self.x,
self.y, self.width, self.height)

self.setWindowIcon(QIcon('pythonico.i
co'))

        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(sel
f.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().availableGeome
try().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





Questions:

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

QLCDNumber: This signal goes off when the QLCDNumber widget tries to show a number that's too big for its display. It's handy for catching errors or changing how the display works on the fly.

QProgressBar: This signal is triggered every time the value of the progress bar changes. It's great for updating other parts of the user interface or starting actions based on how far along the progress is.

QDial: This signal is emitted whenever the dial's value changes. It's useful for updating other UI elements or triggering actions depending on where the dial is set.

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

In PyQt5, signals and slots are like a messaging system for your app. When something happens, like clicking a button, a signal is sent out. Slots are functions that catch these signals and do something in response. This keeps your code neat and flexible because you can easily connect different parts of your app without them needing to know about each other.

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 a GUI application can really improve user experience by providing clear and immediate feedback. For example, they can show error messages when something goes wrong, so users know what happened and what to do next. Message boxes make the app more interactive and user-friendly by keeping users informed and guiding them through their tasks.

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

Error-handling is like a safety net for your code such that it catches mistakes and helps your program deal with them without crashing. In the Procedure we did, error-handling could be applied by using message boxes to show error messages when something goes wrong, like if a user enters invalid data. The app can inform the user about the issue and suggest how to fix it, making the whole experience smoother and less frustrating.

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

First, it helps improve user experience by giving clear feedback when something goes wrong. It also keeps the application stable, so it doesn't crash unexpectedly. Plus, good error handling makes it easier for programmers to find and fix bugs, and it helps protect data from getting corrupted.

8. Conclusion

In summary, effective use of signals and slots in PyQt5 enhances the interactivity and responsiveness of applications, making them more user-friendly. Implementing message boxes improves user experience by providing clear feedback and guidance during interactions. Additionally, error handling is essential for maintaining application stability, protecting data integrity, and ensuring a smooth experience.