

Laboratory Activity 5 - Introduction to Event Handling in GUI Development

Bonifacio, Redj Guillian F.

11/07/2024

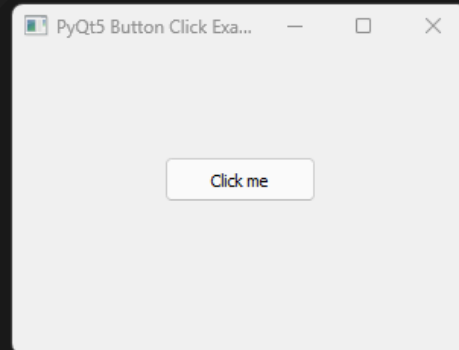
CPE 009B / CPE21S1

Sayo, Maria Rizette

5. Procedure:

gui_buttonclicked.py

```
1 import sys
2 from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton
3 from PyQt5.QtCore import pyqtSlot
4
5 class App(QMainWindow):
6     def __init__(self):
7         super().__init__()
8         self.title = 'PyQt5 Button Click Example'
9         self.left = 100
10        self.top = 100
11        self.width = 300
12        self.height = 200
13        self.initUI()
14
15    def initUI(self):
16        self.setWindowTitle(self.title)
17        self.setGeometry(self.left, self.top, self.width, self.height)
18
19        button = QPushButton('Click me', self)
20        button.setToolTip('Example Button')
21        button.move(100,70)
22        button.clicked.connect(self.on_click)
23
24        self.show()
25
26    @pyqtSlot()
27    def on_click(self):
28        print('Button clicked!')
29
30 if __name__ == '__main__':
31     app = QApplication(sys.argv)
32     ex = App()
33     sys.exit(app.exec_())
34
```

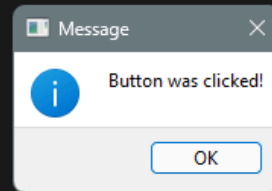
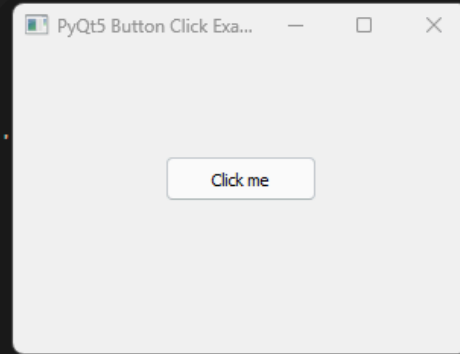


PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton
ModuleNotFoundError: No module named 'PyQt5'
Button clicked!
Button clicked!
```

gui_messagebox.py

```
1 import sys
2 from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QMessageBox
3 from PyQt5.QtCore import pyqtSlot
4
5 class App(QMainWindow):
6     def __init__(self):
7         super().__init__()
8         self.title = 'PyQt5 Button Click Example'
9         self.left = 100
10        self.top = 100
11        self.width = 300
12        self.height = 200
13        self.initUI()
14
15    def initUI(self):
16        self.setWindowTitle(self.title)
17        self.setGeometry(self.left, self.top, self.width, self.height)
18
19        button = QPushButton('Click me', self)
20        button.setToolTip('Example Button')
21        button.move(100,70)
22        button.clicked.connect(self.on_click)
23
24        self.show()
25
26    @pyqtSlot()
27    def on_click(self):
28        QMessageBox.information(self, 'Message', 'Button was clicked!', QMessageBox.Ok)
29
30 if __name__ == '__main__':
31     app = QApplication(sys.argv)
32     ex = App()
33     sys.exit(app.exec_())
34
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

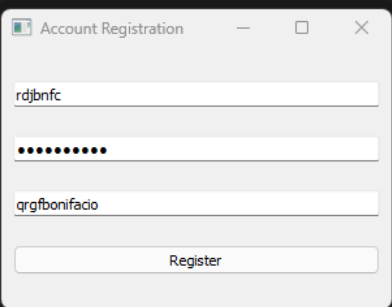
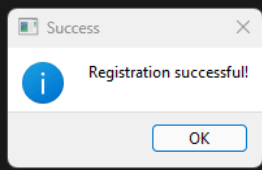
Button clicked!

6. Supplementary Activity:

```

1 import sys
2 from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QLineEdit, QPushButton, QVBoxLayout, QMessageBox
3 import csv
4
5 class AccountRegistration(QMainWindow):
6     def __init__(self):
7         super().__init__()
8         self.setWindowTitle('Account Registration')
9         self.setGeometry(100, 100, 300, 200)
10
11         layout = QVBoxLayout()
12         central_widget = QWidget()
13         central_widget.setLayout(layout)
14         self.setCentralWidget(central_widget)
15
16         self.username = QLineEdit(placeholderText="Username")
17         self.password = QLineEdit(placeholderText="Password", echoMode=QLineEdit.Password)
18         self.email = QLineEdit(placeholderText="Email")
19
20         register_button = QPushButton('Register')
21         register_button.clicked.connect(self.register)
22
23         layout.addWidget(self.username)
24         layout.addWidget(self.password)
25         layout.addWidget(self.email)
26         layout.addWidget(register_button)
27
28     def register(self):
29         username = self.username.text().strip()
30         password = self.password.text().strip()
31         email = self.email.text().strip()
32
33         if not all([username, password, email]):
34             QMessageBox.warning(self, 'Error', 'All fields must be filled!')
35             return
36
37         try:
38             with open('accounts.csv', 'a', newline='') as file:
39                 writer = csv.writer(file)
40                 writer.writerow([username, password, email])
41             QMessageBox.information(self, 'Success', 'Registration successful!')
42             self.username.clear()
43             self.password.clear()
44             self.email.clear()
45         except Exception as e:
46             QMessageBox.critical(self, 'Error', f'Registration failed: {str(e)}')
47
48 if __name__ == '__main__':
49     app = QApplication(sys.argv)
50     ex = AccountRegistration()
51     ex.show()
52     sys.exit(app.exec_())
53

```

Questions

- What are the other signals available in PyQt5? (give at least 3 and describe each)
PyQt5 has many signals like clicked() for buttons, textChanged() for typing, and valueChanged() for sliders. These signals help detect different user actions on various parts of the app.
- Why do you think that event handling in Python is divided into signals and slots?
Signals and slots in Python make it easier to organize code. They separate what happens (signals) from how to respond (slots). This makes the code more flexible and easier to change or fix later.

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 make apps more user-friendly. They tell users what's happening, warn about errors, ask for confirmation, and provide help. This keeps users informed and prevents mistakes.

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

Error-handling means preparing for and dealing with problems in a program. In this task, it was used to check for empty fields, handle file-saving issues, show error messages, and keep the app running smoothly even when things go wrong.

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

Error handling is important because it stops the app from crashing, helps users understand problems, makes fixing bugs easier, protects data, keeps the app working even if part of it fails, improves security, and meets user expectations for good software.

7. Conclusion:

In conclusion, learning about event handling, signals, and slots in PyQt5 is key to making interactive and easy-to-use apps. By using message boxes and handling errors well, developers can improve the user experience, keep the program running smoothly, and protect data, resulting in better and more reliable software.