**Lab Exercise 29– Observer Design Pattern in PyQT**

here's a lab exercise that demonstrates the implementation of the Observer design pattern in PyQt:

from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton

from PyQt5.QtCore import QObject, pyqtSignal, pyqtSlot

class Subject(QObject):

data\_changed = pyqtSignal(str)

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.\_data = ""

@property

def data(self):

return self.\_data

@data.setter

def data(self, value):

self.\_data = value

self.data\_changed.emit(value)

class Observer(QMainWindow):

def \_\_init\_\_(self, subject):

super().\_\_init\_\_()

self.subject = subject

self.subject.data\_changed.connect(self.on\_data\_changed)

self.setWindowTitle("Observer Design Pattern Example")

self.setGeometry(100, 100, 300, 200)

self.button = QPushButton("Update Data", self)

self.button.clicked.connect(self.update\_data)

self.setCentralWidget(self.button)

@pyqtSlot(str)

def on\_data\_changed(self, data):

print(f"Data changed to: {data}")

@pyqtSlot()

def update\_data(self):

new\_data = "New data from observer"

self.subject.data = new\_data

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication([])

subject = Subject()

observer = Observer(subject)

observer.show()

app.exec\_()

In this example, we have a Subject class that emits a data\_changed signal when its data property is updated. The Observer class is a simple QMainWindow that listens for changes from the Subject and prints the updated data. When the button in the Observer window is clicked, it updates the data in the Subject, which triggers the signal and notifies the observer.