**Lab Exercise 14 – Integrating QML Components into Python Code**

**Objective:**

Learn how to **connect QML UI components with Python logic** using PySide6. This lab focuses on:

* Loading QML in Python using QQmlApplicationEngine.
* Exposing Python functions to QML using setContextProperty().
* Using signals and slots to handle user interactions.
* Manipulating QML UI elements from Python.

**1. Learning Outcomes**

By completing this lab, you will:  
✅ Load a QML UI file from Python.  
✅ Connect QML buttons and labels to Python logic.  
✅ Modify QML elements dynamically from Python.  
✅ Use signals and slots for event-driven programming.

**2. Project Setup & Requirements**

**Project Structure**

qml\_python\_integration/

│── main.py # Python backend

│── interface.qml # QML UI file

│── requirements.txt # Dependencies (PySide6)

**Install Dependencies**

Ensure you have **PySide6** installed:

pip install PySide6

**3. Step 1: Create the QML UI (interface.qml)**

This QML file defines:

* A **Label** displaying text.
* A **Button** that triggers a Python function to update the text.

import QtQuick 6.2  
import QtQuick.Controls 6.2  
  
ApplicationWindow {  
 visible: true  
 width: 400  
 height: 200  
 title: "Update Label from Python"  
  
 Column {  
 anchors.centerIn: parent  
 spacing: 10  
  
 Label {  
 id: displayText  
 objectName: "displayText" // Important: Assign objectName for findChild()  
 text: "Original Text"  
 font.pixelSize: 16  
 }  
  
 Button {  
 text: "Update Text"  
 onClicked: backend.updateText() // Call Python function  
 }  
 }  
}

**4. Step 2: Implement Python Backend (main.py)**

This Python script:

* Loads the **QML UI** using QQmlApplicationEngine.
* Exposes a **Python function** (updateText()) to QML.
* Dynamically updates the **Label text** in QML from Python

import sys  
import os  
from PySide6.QtWidgets import QApplication  
from PySide6.QtQml import QQmlApplicationEngine  
from PySide6.QtCore import QObject, Slot, Signal  
  
  
class Backend(QObject):  
 """Handles UI interactions from QML."""  
  
 textUpdated = Signal(str) # Define a signal for updating the label  
  
 def \_\_init\_\_(self, engine):  
 super().\_\_init\_\_()  
 self.engine = engine # Store the engine reference  
  
 @Slot()  
 def updateText(self):  
 """Updates the Label in QML using findChild()."""  
 root\_objects = self.engine.rootObjects()  
 if not root\_objects:  
 print("Error: No root objects found!")  
 return  
  
 root = root\_objects[0] # Get the main QML object  
 label = root.findChild(QObject, "displayText") # Find Label by objectName  
  
 if label:  
 label.setProperty("text", "Text Updated from Python!") # Update Label  
 else:  
 print("Error: displayText object not found!")  
  
  
def main():  
 app = QApplication(sys.argv)  
 engine = QQmlApplicationEngine()  
  
 backend = Backend(engine)  
 engine.rootContext().setContextProperty("backend", backend) # Expose backend to QML  
  
 qml\_file = os.path.join(os.path.dirname(\_\_file\_\_), "interface.qml")  
 engine.load(qml\_file) # Ensure the QML file loads correctly  
  
 if not engine.rootObjects():  
 sys.exit(-1)  
  
 sys.exit(app.exec())  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**5. Running the Application**

Run the Python script:

python main.py

✅ **Click "Change Text"** → The label text updates dynamically from Python.

**6. Key Concepts in This Exercise**

✅ **QQmlApplicationEngine** loads QML into Python.  
✅ **setContextProperty("backend", backend)** connects Python to QML.  
✅ **QML signals (onClicked) trigger Python slots (updateText()).**  
✅ **Python modifies QML UI elements dynamically.**

**7. Next Steps & Enhancements**

* ✅ Add an **input field** in QML that sends user input to Python.
* ✅ Implement a **QML ListView that updates with Python data**.
* ✅ Use **QML animations for smooth UI transitions**.