Correction: Advanced PyQt5 Concepts Quiz

Question 1: Signals and Slots (4 Marks)

Answer:

- **‘pyqtSignal‘**: Used to define custom signals in a PyQt5 class. Signals are emitted when a specific event occurs. - **‘pyqtSlot‘**: Used to define slots (methods) that can be connected to signals. Slots are called when a signal is emitted.

Example:

```
from PyQt5.QtCore import pyqtSignal, pyqtSlot, QObject

class Communicator(QObject):
    # Define a custom signal
    message_signal = pyqtSignal(str)

    def __init__(self):
        super().__init__()

    @pyqtSlot(str)
    def receive_message(self, message):
        print(f"Received: {message}")

# Usage
communicator = Communicator()
communicator.message_signal.connect(communicator.receive_message)
communicator.message_signal.emit("Hello, PyQt5!")
```

—

Question 2: Multithreading with QThread (4 Marks)

Answer:

- **How to Use ‘QThread‘**: - Create a subclass of ‘QThread‘ and override the ‘run‘ method to perform the background task. - Use signals to communicate between the thread and the main UI.

- **Potential Pitfalls**: - **Blocking the Main Thread**: Avoid calling ‘QThread.wait()‘ in the main thread, as it can freeze the UI. - **Thread Safety**: Ensure that shared resources are accessed safely using locks or signals.

Example:

```
from PyQt5.QtCore import QThread, pyqtSignal

class WorkerThread(QThread):
    result_ready = pyqtSignal(int)

    def run(self):
        for i in range(10):
```

```
            self.result_ready.emit(i)
            self.msleep(500)  # Simulate work
```

—

Question 3: Custom Widgets (4 Marks)

Answer:

- **Custom Widget Code**:

```
from PyQt5.QtWidgets import QWidget, QSlider, QLabel, QVBoxLayout
from PyQt5.QtCore import Qt

class SliderWithLabel(QWidget):
    def __init__(self):
        super().__init__()
        self.slider = QSlider(Qt.Horizontal)
        self.label = QLabel("0")
        self.slider.valueChanged.connect(self.update_label)
        layout = QVBoxLayout()
        layout.addWidget(self.slider)
        layout.addWidget(self.label)
        self.setLayout(layout)

    def update_label(self, value):
        self.label.setText(str(value))
```

- **Usage**:

```
from PyQt5.QtWidgets import QApplication

app = QApplication([])
widget = SliderWithLabel()
widget.show()
app.exec_()
```

—

Question 4: Event Handling (4 Marks)

Answer:

- **Override 'mousePressEvent'**:

```
from PyQt5.QtWidgets import QWidget, QMessageBox

class ClickableWidget(QWidget):
    def mousePressEvent(self, event):
        QMessageBox.information(self, "Clicked", "You clicked the widget!")
```

- **Usage**:

```
from PyQt5.QtWidgets import QApplication

app = QApplication([])
widget = ClickableWidget()
widget.show()
app.exec_()
```

—

Question 5: Advanced UI Design (4 Marks)

Answer:

- **Dynamic UI Example**:

```
from PyQt5.QtWidgets import QComboBox, QTableWidget, QVBoxLayout, QWidget, QApplication

class DynamicUI(QWidget):
    def __init__(self):
        super().__init__()
        self.combo = QComboBox()
        self.combo.addItems(["Option 1", "Option 2"])
        self.table = QTableWidget()
        self.combo.currentTextChanged.connect(self.update_table)
        layout = QVBoxLayout()
        layout.addWidget(self.combo)
        layout.addWidget(self.table)
        self.setLayout(layout)

    def update_table(self, text):
        self.table.clear()
        if text == "Option 1":
            self.table.setRowCount(2)
            self.table.setColumnCount(2)
            self.table.setItem(0, 0, QTableWidgetItem("A"))
            self.table.setItem(0, 1, QTableWidgetItem("B"))
        elif text == "Option 2":
            self.table.setRowCount(3)
            self.table.setColumnCount(3)
            self.table.setItem(0, 0, QTableWidgetItem("X"))
            self.table.setItem(1, 1, QTableWidgetItem("Y"))
```

- **Usage**:

```
app = QApplication([])
ui = DynamicUI()
ui.show()
app.exec_()
```