**Basics of Python Threading and QThread in PyQt/PySide**

Python provides **threading** for running tasks concurrently, but **PyQt/PySide** offers QThread for GUI applications to prevent UI freezing while performing background tasks.

**1. Python Threading (Basic Multithreading)**

Python’s threading module allows executing code in separate threads but has limitations due to the **Global Interpreter Lock (GIL)**.

**Example: Running a Task in a Background Thread**

import threading

import time

def background\_task():

"""A function that runs in a separate thread."""

for i in range(5):

print(f"Task running... {i}")

time.sleep(1)

# Create and start the thread

thread = threading.Thread(target=background\_task)

thread.start()

print("Main thread is free!")

**Key Features:**

✅ Runs a task in a background thread.  
✅ The **main thread remains responsive**.  
✅ **Limitation**: Cannot directly update the **GUI** in PyQt/PySide using threading.Thread.

**2. Using QThread for GUI Applications**

Since threading.Thread cannot update the UI, **QThread** is preferred for PyQt/PySide applications.

**Example: Running a Background Task with QThread**

from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QLabel

from PySide6.QtCore import QThread, Signal

import time

import sys

class WorkerThread(QThread):

progress = Signal(int) # Custom signal to update UI

def run(self):

"""Long-running task (runs in background)."""

for i in range(5):

time.sleep(1) # Simulate work

self.progress.emit(i + 1) # Send progress update

class MyWindow(QWidget):

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

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

self.setWindowTitle("QThread Example")

self.resize(300, 200)

layout = QVBoxLayout()

self.label = QLabel("Click the button to start task")

layout.addWidget(self.label)

self.button = QPushButton("Start Task")

self.button.clicked.connect(self.start\_thread)

layout.addWidget(self.button)

self.setLayout(layout)

def start\_thread(self):

"""Start the worker thread."""

self.worker = WorkerThread()

self.worker.progress.connect(self.update\_label) # Connect signal to UI update

self.worker.start()

def update\_label(self, value):

"""Update the UI from the worker thread."""

self.label.setText(f"Progress: {value}/5")

app = QApplication(sys.argv)

window = MyWindow()

window.show()

sys.exit(app.exec())

**Key Features of QThread**

✅ **Runs a task in the background** without freezing the UI.  
✅ Uses **signals (Signal)** to safely update the UI.  
✅ The run() method contains the long-running task.  
✅ The **UI remains responsive** while the task runs.

**Conclusion**

* **Python threading.Thread** works for background tasks but **cannot update the UI**.
* **QThread (subclassing)** is useful for simple tasks that need **UI updates via signals**.