Using More Threads

Dan Tofan

Software Engineer, PhD

@dan_tofan | www.programmingwithdan.com

Module Overview

What are threads?

Challenges of working with threads

When to use multithreading

Threads

Separate execution flows

Part of a process

Concurrency

Improve performance

Threads vs Processes

Threads	VS	Processes
Lightweight		Heavyweight
Shared memory		Separate memory
High potential for bugs		Low potential for bugs
GIL constraint		No GIL constraint

```
import threading
```

◀ **Import the threading module**

```
class OrderProcessing(threading.Thread):
    def run(self):
        print(f"Processing")

thread = OrderProcessing()

thread.start()
```

◀ **Create a subclass of Thread**

◀ **Create new instance**

◀ **Start the new thread**

```
def process():
    print(f"Processing...")

t = threading.Thread(target=process)

t.start()
```

◀ **Python function**

◀ **Create new thread**

◀ **Start the new thread**

Demo

Process orders

Use the threading module

Create two threads

Challenges of Working with Threads

Synchronizing threads

Troubleshooting

GIL

Synchronizing Threads

Race conditions

Deadlocks

Starvation

Livelocks

Troubleshooting

Reproducing

Debugging

Finding the root cause of issues

Global Interpreter Lock

Only one thread is running

The sleep() function releases the GIL

Most impact on CPU-intensive tasks

Demo

Process orders

Check performance impact of GIL

- One thread
- Two threads

How to Synchronize Threads

Locks

Semaphores

Condition variables

When to Use Threads

Use threads	VS	Avoid threads
Tasks that wait for external events		No waiting for external events
Blocking I/O		CPU-intensive tasks
Simple logic		Complex logic

Demo

Download from URL

Compare performance

- Single thread
- Multiple threads

Module Summary

What are threads?

Challenges of working with threads

When to use multithreading

Up Next:

Using Asynchronous Code