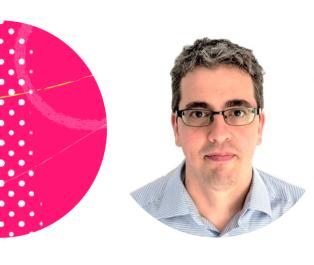
Using Asynchronous Code



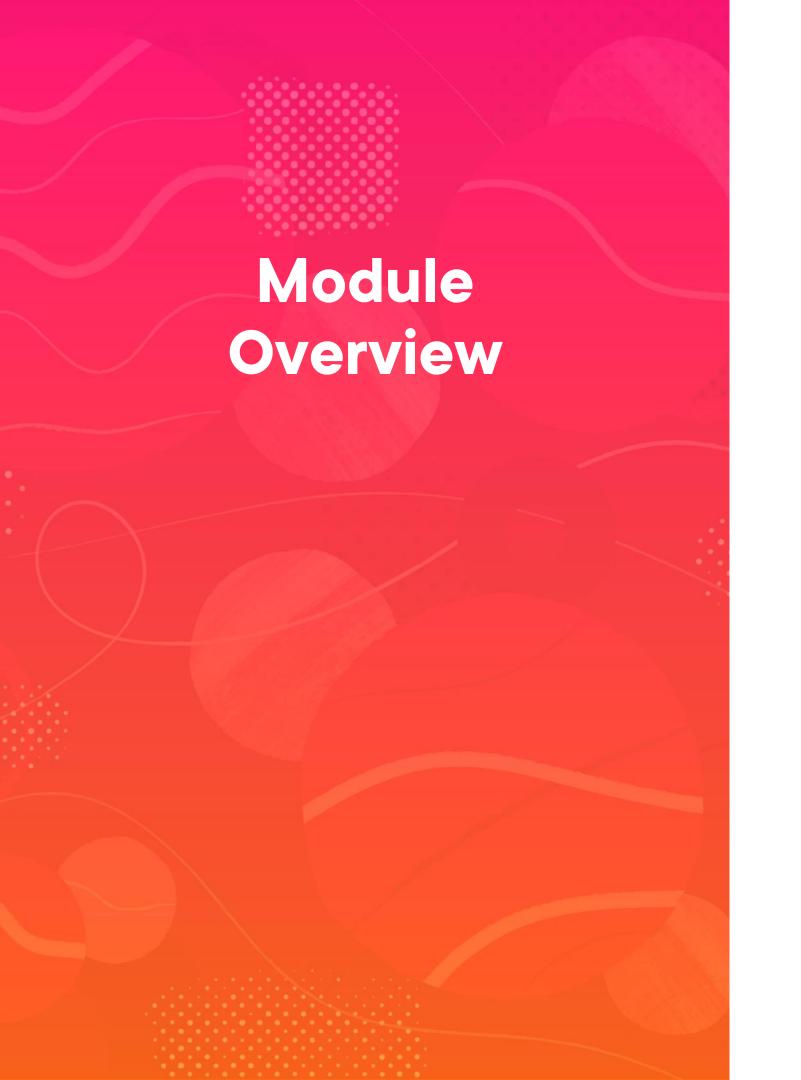
Dan Tofan

Software Engineer, PhD

@dan_tofan | www.programmingwithdan.com







Asynchronous code
Challenges of working with asyncio
When to use asyncio



Asynchronous Code

Inspired from other languages

Reduce potential for bugs

Maximize core utilization

New syntax, concepts, tools



Async vs Threads

Async VS Threads

Low overhead

Low potential for bugs

Learning curve

Compatibility constraint

High overhead

High potential for bugs

Simple syntax

High compatibility

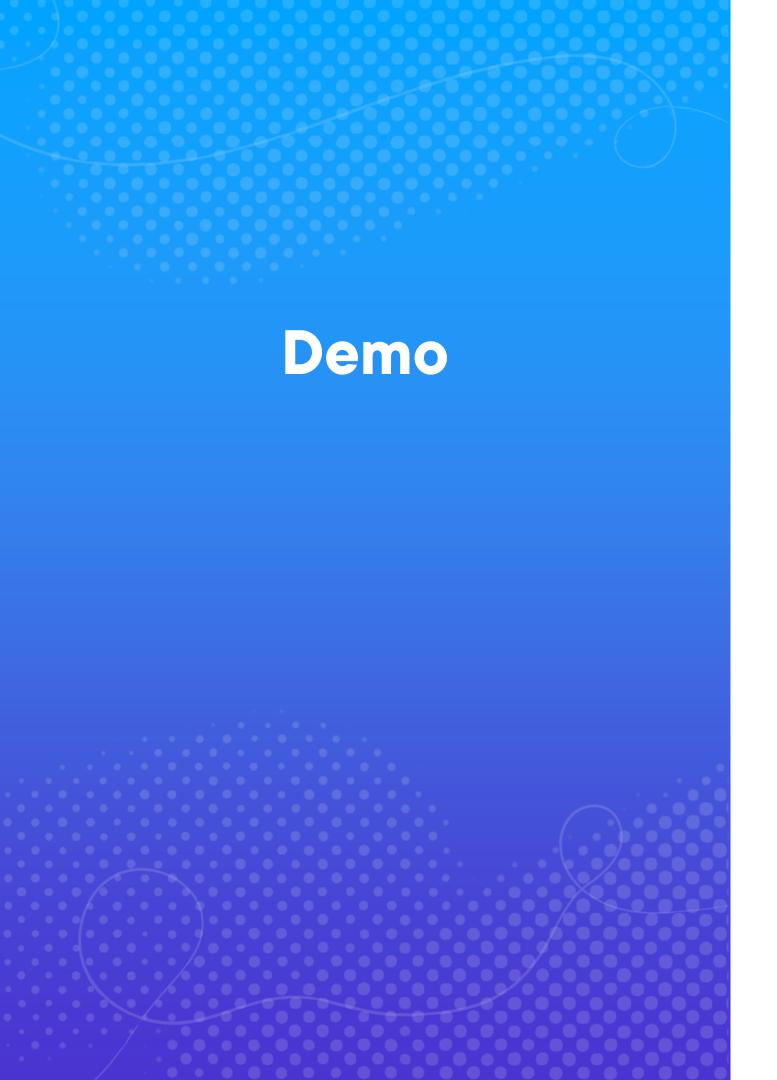
```
import asyncio
async def process_order():
    await asyncio.sleep(1)
    print("Order complete")
async def main():
    await process_order()
    print("Finished processing")
asyncio.run(main())
```

◄ Import the asyncio module

- **◄** Define new coroutine
- **■** Wait with await

- **◄** Define another coroutine
- **◄** Call and wait for coroutine

■ Start event loop



Process orders with the asyncio module

Challenges of Working with Asyncio

Learning curve

Debugging

Compatibility



Learning Curve

New syntax: async await

New concepts: coroutine event loop

New libraries: aiohttp aiomysql



Debugging

Understand order of execution

Understand state of application



Compatibility

Third-party libraries

Blocking code



Demo

Process orders with asyncio

Check performance

- Process one vs two orders
- Blocking vs unblocking code

When to Use Asyncio

I/O operations

Many small tasks

Avoid synchronizing threads

Asynchronous dependencies

Data processing pipelines

Networking applications



When to Avoid Using Asyncio

CPU intensive tasks

Blocking code

Blocking dependencies

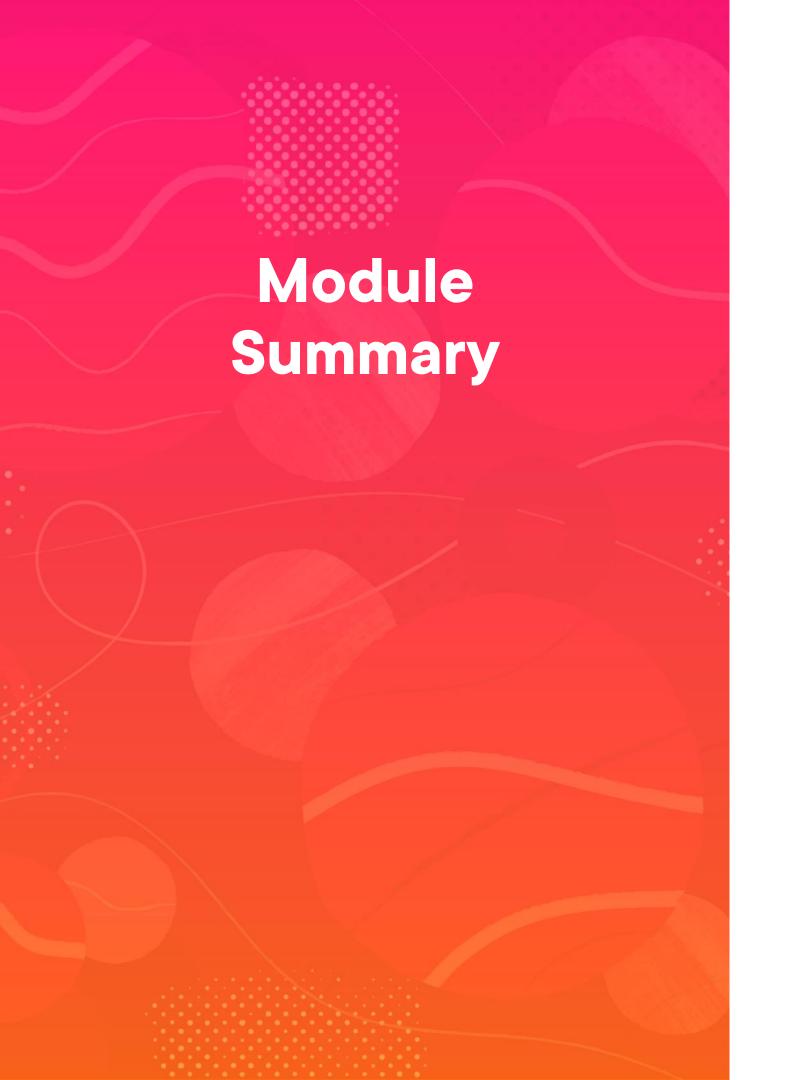


Demo

Download from URL

Compare performance

- Synchronous code
- Asynchronous code



Asynchronous code
Challenges of working with asyncio
When to use asyncio



More Information

Getting Started with Python 3 Concurrency Tim Ojo

Up Next:

Using More Processes

