# Python Async/Await Guide

==========

## INTRODUCTION TO ASYNC/AWAIT

Async/await is Python's way of writing asynchronous code. It allows you to write concurrent code that can handle multiple tasks without blocking.

## BASIC ASYNC FUNCTION

Here's a simple async function:

```
import asyncio

async def greet(name):
    print(f"Hello, {name}!")
    await asyncio.sleep(1)
    print(f"Goodbye, {name}!")

# Run the async function
asyncio.run(greet("Alice"))
```

The 'async' keyword defines an asynchronous function. The 'await' keyword pauses execution until the awaited task completes.

## RUNNING MULTIPLE TASKS

You can run multiple async tasks concurrently using asyncio.gather():

```
import asyncio

async def fetch_data(url):
    print(f"Fetching {url}...")
    await asyncio.sleep(2)  # Simulate network delay
    return f"Data from {url}"

async def main():
    urls = ["api.example.com/users", "api.example.com/posts"]
    results = await asyncio.gather(*[fetch_data(url) for url in urls])
    for result in results:
        print(result)

asyncio.run(main())
```

This code fetches from two URLs concurrently, completing in ~2 seconds instead of ~4 seconds if done sequentially.

## ERROR HANDLING IN ASYNC CODE

Use try/except blocks just like synchronous code:

```
async def safe_fetch(url):
    try:
        result = await fetch_data(url)
        return result
    except Exception as e:
        print(f"Error fetching {url}: {e}")
        return None
```

## ASYNC CONTEXT MANAGERS

For resources that need async setup/teardown:

```
class AsyncDatabase:
    async def __aenter__(self):
        await self.connect()
```