AsyncIO in Python – Colorful Notes

1 What is asyncio?

`asyncio` is a Python module that allows your program to **run multiple tasks concurrently** without using multiple threads or processes.

Normally, Python runs **line by line** (synchronously). With `asyncio`, Python can **pause a task while waiting** (like for network or file) and **switch to another task**, making your program more efficient.

Analogy: Like a chef in a kitchen: instead of waiting for water to boil, the chef chops veggies while waiting.

2 Key Concepts

Keyword	Meaning
`async`	Marks a function as asynchronous (can be paused and resumed)
`await`	Pauses until result is ready, then continues
`asyncio.run()`	Starts the async program
`asyncio.gather()`	Runs multiple async functions concurrently

3 How it works (Q&A style)

a) Normal execution vs async

Normal (synchronous) code:

```
print("Line 1")
print("Line 2")
time.sleep(2)  # wait 2 sec
print("Line 3")
print("Line 4")
```

Line 3 blocks everything for 2 seconds.

Async code:

```
import asyncio

async def func():
    print("Line 1")
    await asyncio.sleep(2)  # pause here
    print("Line 2")

asyncio.run(func())
```

While waiting, Python can run other async tasks.

---

b) Between async functions

```
import asyncio

async def func1():
    print("func1 start")
    await asyncio.sleep(2)
    print("func1 end")

async def func2():
    print("func2 start")
    await asyncio.sleep(1)
    print("func2 end")

async def main():
    await asyncio.gather(func1(), func2())

asyncio.run(main())
```

Execution order: 1 `func1` starts → pauses for 2 sec. 2 `func2` starts → pauses for 1 sec. 3 `func2` finishes first → prints `func2 end`. 4 `func1` resumes → prints `func1 end`.

While one waits, the other continues.

---

c) Inside the same function

```
async def func():
    print("A")
    await asyncio.sleep(2)
    print("B")
```

After `await`, execution **pauses here**. Other tasks can run, but this function continues at **print("B")** only after 2 seconds.

---

d) Key takeaway

If **func1 is waiting**, **func2 can run**. Within a function, code **after** "** waits** until resume. Python remembers the paused point and resumes correctly.

---

4 Why use asyncio?

Best for **I/O-bound tasks** (network requests, file reads). Runs tasks **concurrently without threads**. Saves time and CPU cycles .

---

5 Real-world analogy

Two friends texting:

- **Friend1** sends a message → waits for reply (`await`).
- While waiting, **Friend2** sends a message.
- When reply comes, **Friend1** continues exactly where they left off.

AsyncIO lets Python **multitask efficiently** without real parallel threads.

Excellent observation You are right — let's break it down carefully.

**1 Async function without **"

```
async def func():
    print("Hello")

func()   #  just calling
```

This does **not run the function** immediately.
It only creates a **coroutine object** (like a "promise to run later").

Without `asyncio.run(func())` or `await func()`, nothing actually executes.

**2 Async function with **"

```
async def func():
    print("Hello")

asyncio.run(func())  #  runs the async function
```

Now it executes, because the **event loop** is started .

---

3 Using “** (sequential)**

If you just `await` one function, it behaves like a normal function call (but non-blocking inside):

```
async def func1():
    print("Start 1")
    await asyncio.sleep(2)
    print("End 1")

async def main():
    await func1()   # runs like normal, step by step

asyncio.run(main())
```

Only `func1` runs, just like a normal function — **no concurrency yet** .

---

4 With “** (concurrent)**

To run multiple async functions **concurrently**, you use `asyncio.gather()`:

```
async def func1():
    print("Start 1")
    await asyncio.sleep(2)
    print("End 1")

async def func2():
    print("Start 2")
    await asyncio.sleep(1)
    print("End 2")

async def main():
    await asyncio.gather(func1(), func2())

asyncio.run(main())
```

Both functions run **at the same time**. While one is waiting, the other continues.

Colorful Summary:

- `async def` → defines an async function.

- Calling it → returns a **coroutine object** (not executed yet).
- `asyncio.run()` → starts the event loop and actually runs it.
- `await` → runs one coroutine, step by step.
- `gather` → runs multiple coroutines concurrently.

Extra Visualization Idea:

- `await func1(); await func2();` → `func1` fully runs, then `func2` runs.
- `await asyncio.gather(func1(), func2());` → both run together, saving time.