Demystifying async & await In Python and JavaScript

Kurian Benoy(He/Him)

Outline

- Concurrency/Parallelism
- What is async/await?
- async/await in context of Python
- async/await in context of JavaScript
- Where is it used?
- Common mistakes while using async/await
- Conclusion

About Me

- SE - Data Scientist @ AOT Technologies
- Loves Python
- Open source enthusiast

formsflow.ai

If you want to follow along with slides: bit.ly/async-await-pycon

async/await

```
async def get_profile(request: HttpRequest):
   profile = load_profile(request.GET['user_id'])
   ...

 async def run_query(query: str):
   connection = create_sql_connection()
   result = await connection.execute(query)
```

Code courtesy: https://github.com/facebook/pyre-check/

courtesy: Vipul Gupta

Chess Simul

Assumptions

- Playing 24 opponents
- Each opponent is less than 1500 ELO(Chess rating system)
- Each game averages 30 moves by both players
- Anand moves in 5 seconds
- Opponents move in 55 seconds

Photo courtesy: Amruta Mokal(ChessbaseIndia)

Synchronous Chess Simul

- Each game runs for 30 minutes
- 24 sequential games would take

24*30 minutes = 12 hours

Photo courtesy: Keith Rust

Asynchronous Chess Simul

- Anand would make first move, and move onto to 2nd player, 3rd, and so on.
- Anand completes first round playing 24 opponents in = 5 sec * 24 = 2 minutes
- Now the first opponent is ready for their next move!
- So if all games conclude in 30 moves pairs: 30*2 minutes = 1 hour

Photo courtesy: Keith Rust

BottleNeck in Chess Simul

I/O Bound Problem - *refers to a condition in which the time it takes to complete a computation is determined principally by* ***the period spent waiting for input/output operations*** *to be completed.This circumstance arises when the rate at which data is requested is slower than the rate it is consumed or, in other words,* ***more time is spent requesting data than processing it****.*

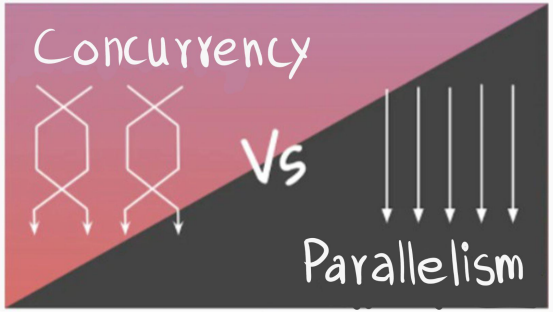
Concurrency is about dealing with lots of things at once.

– Rob Pike

Photo courtesy: Los Muertos Crew

Parallel Chess Simul with grandmasters(GM)

Edited Photo courtesy: Keith Rust, Albi Ani



Concurrency != Parallelism

Concurrency is about dealing with lots of things at once.
Parallelism is about doing lots of things at once.
Not the same, but related.
One is about structure, one is about execution.

Concurrency provides a way to structure a solution to solve a problem that may (but not necessarily) be parallelizable.

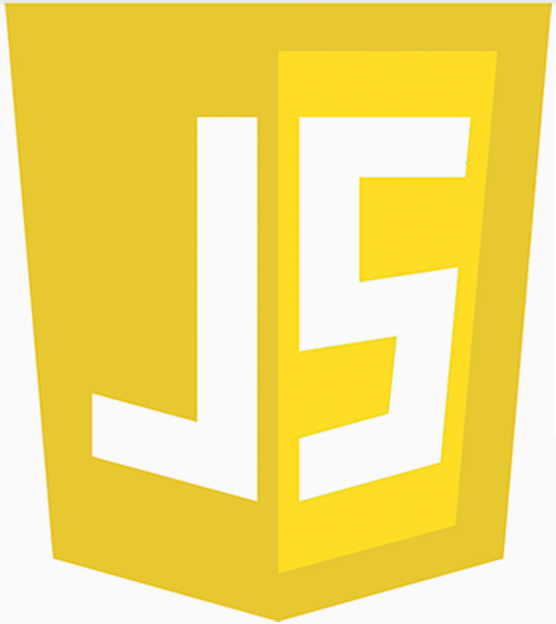
-Rob Pike

What is async & await?

<u>async</u> - a way to run code concurrently

<u>await</u> - to wait and handle a concurrent result

Python Usage with Other languages



Jet Brains - Python Developer Survey 2020 - https://www.jetbrains.com/lp/python-developers-survey-2020/

PYTHON

```python
import time


def greet_after(delay: int, person: str):
    time.sleep(delay)
    print(f"Hello {person} \U0001F44B")


if __name__ == "__main__":
    print(f"Started at {time.strftime('%X')}")
    greet_after(3, "Guido van Rossum")
    greet_after(2, "Sebastián Ramírez")
    greet_after(1, "Luciano Ramaldho")
    print(f"Finished at {time.strftime('%X')}")
```

```
Started at 14:06:06
Hello Guido van Rossum 👋
Hello Sebastián Ramírez 👋
Hello Luciano Ramaldho 👋
Finished at 14:06:12
>
```

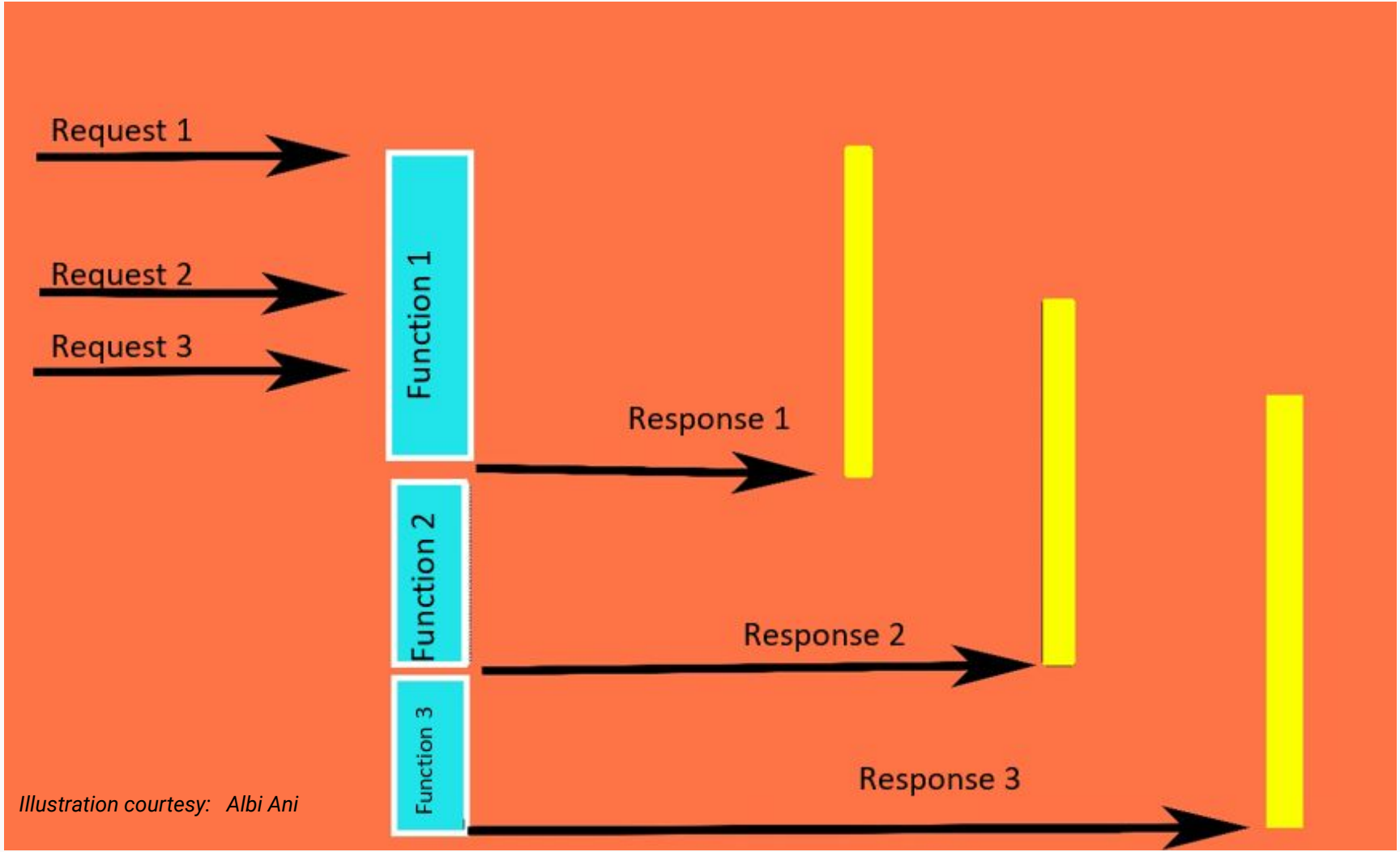


Illustration courtesy: Albi Ani

Concurrency in CPython

- **Threading**
- **Asyncio module(async/await)**

Photo courtesy : Keith Rust

Threading library

- Separate flow of execution
- Mapping internally to operating system threads
- Suitable for I/O concurrency

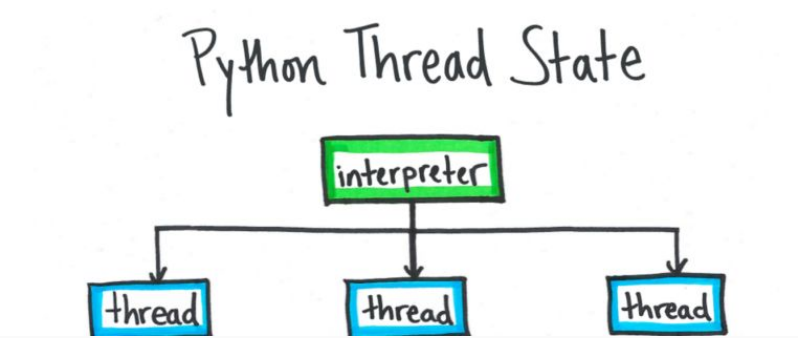


Photo from Luminousgen

```python
import threading


def main():
    thread1 = threading.Thread(target=greet_after, args=(3, "Guido van Rossum"))
    thread2 = threading.Thread(target=greet_after, args=(2, "Sebastián Ramírez"))
    thread3 = threading.Thread(target=greet_after, args=(1, "Luciano Ramaldho"))
    thread1.start()
    thread2.start()
    thread3.start()


if __name__ == "__main__":
    main()
```

```
> Hello Luciano Ramaldho 👋
Hello Sebastián Ramírez 👋
Hello Guido van Rossum 👋
```

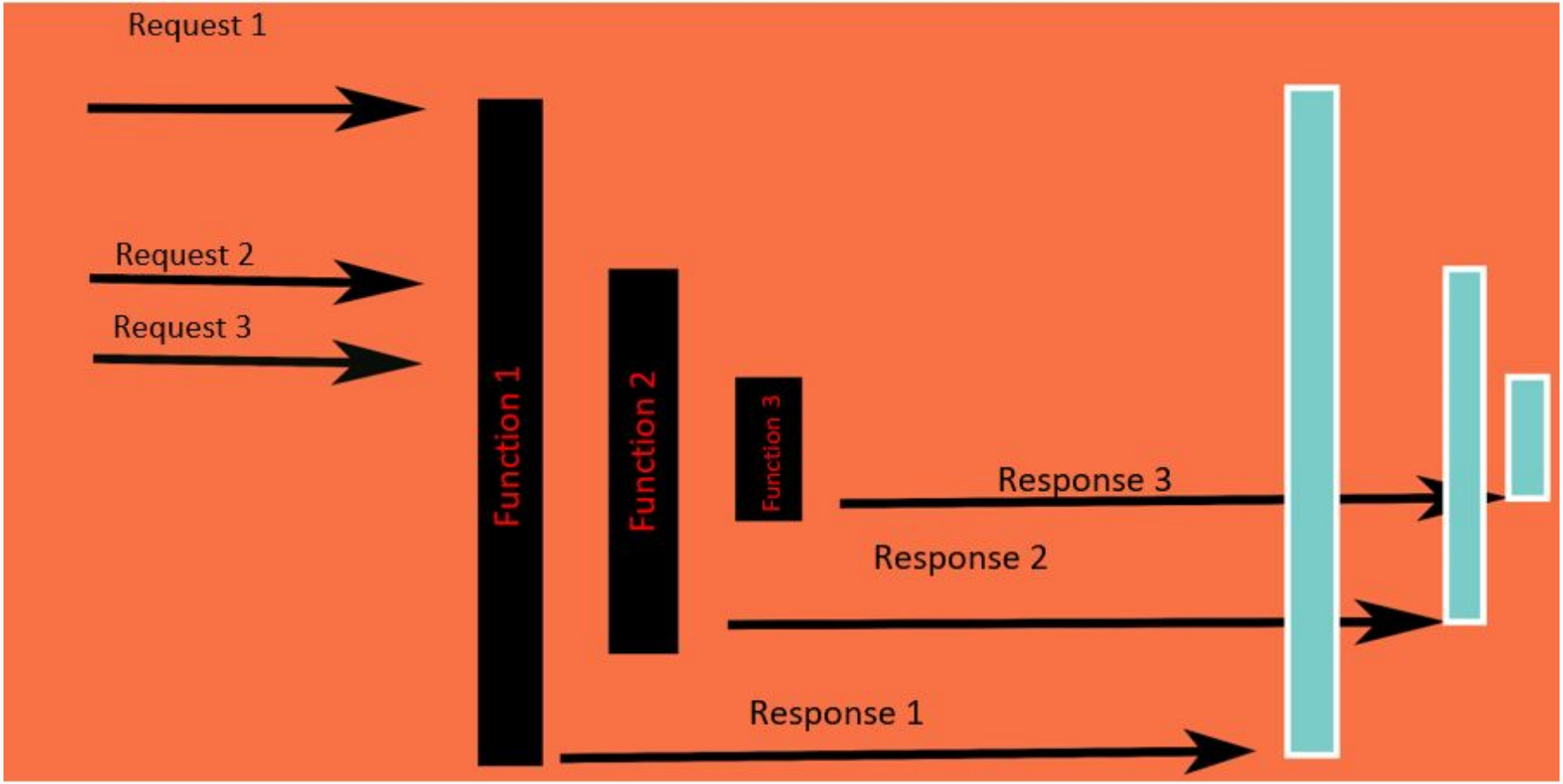


Illustration courtesy: Albi Ani

Limitations of threading

- Race Condition
- Causes Deadlock
- Certains tasks may never be run

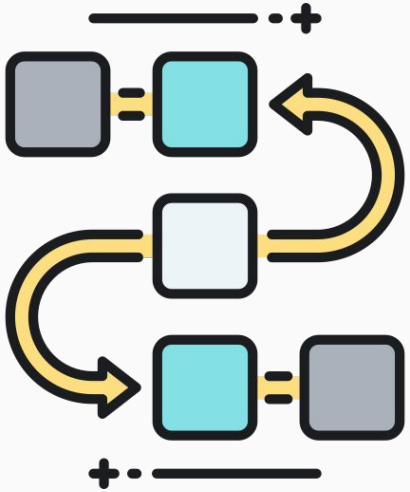
Asyncio module

- Using cooperative multitasking
- async/await syntax

- Coroutines
- Eventloops
- Tasks
- Futures

Coroutines

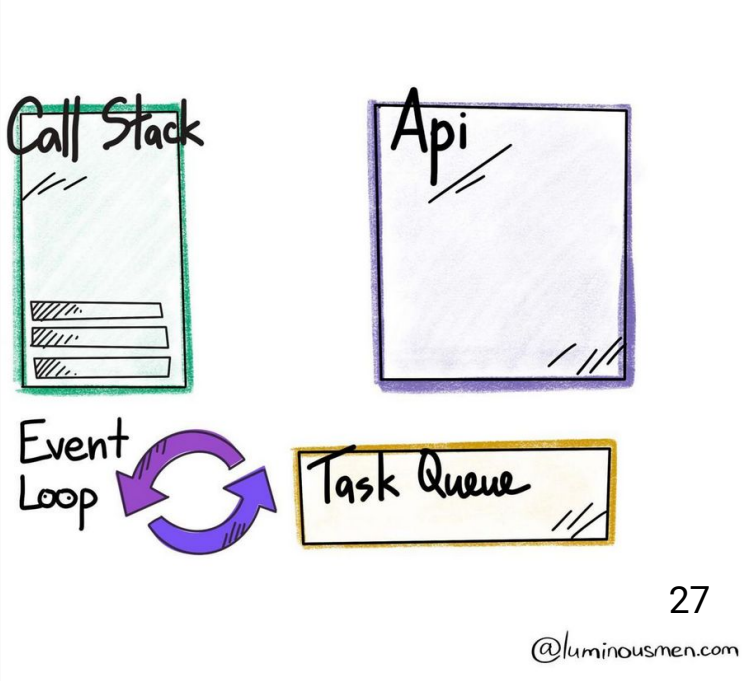
- Functions that can suspend/resume
- **async def** syntax
- Returns coroutine object
- Must be **awaited**



Picture from flaticon.com

EventLoops

- Executes Coroutines
- Picks next *coroutine* from the queue
- **asyncio.run()**
- Plugabble event loop, **uvloop**



Tasks

- Schedules Coroutines
- Wraps coroutine with **asyncio.create_task()**
- Returns tasks object
- Multiple tasks run concurrently by **asyncio.gather()**

Picture from flaticon.com

Futures

- Used to bridge low level callback-based code to high level async await code
- Property to ensure that code is being run after some time

Picture from flaticon.com

```python
import asyncio

async def main():
    task1 = asyncio.create_task(greet_after(3, "Guido van Rossum"))
    task2 = asyncio.create_task(greet_after(2, "Sebastián Ramírez"))
    task3 = asyncio.create_task(greet_after(1, "Luciano Ramaldho"))
    final_task = asyncio.gather(task1, task2, task3)
    await final_task

if __name__ == "__main__":
    print(f"Started at {time.strftime('%X')}")
    asyncio.run(main())
    print(f"Finished at {time.strftime('%X')}")
```

```
Started at 14:16:36
Hello Luciano Ramaldho 👋
Hello Sebastián Ramírez 👋
Hello Guido van Rossum 👋
Finished at 14:16:39
>
```

JAVASCRIPT

```
function find_primes(num) {
  let flag = 0;
  for (var index= 2; index<num; index ++) {
    for(var i=2; i<num/2; i++) {
      if(index %i === 0) {
        flag = 1;
       }
      }
      if (flag===0) {
        console.log(index);
        }
      flag = 0;
      }
}


console.log(find_primes(1000000))
console.log(find_primes(1000))
console.log(find_primes(10))
```

Callbacks

Promises

async/await in context of JavaScript

The **<u>async</u>** keyword before a function has two effects:

- Make it return a promise
- Allow await to be used in it

The **<u>await</u>** keyword before a promise to wait until the promise is settled,

1. Else returns the result when promise is settled
2. If it's an error, throws an exception

```javascript
function wait() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve(console.log("Thanks for waiting"))
    }, 4000)
  })
}

async function say_hello() {

  console.log("Hello");
  console.log("Hello");

  await wait();
  console.log("World");
}

function print_warmup() {
  console.log("Pythonistas");
}

say_hello()
print_warmup();
```

```
Hello
Hello
Pythonistas
Hint: hit control+c anytime to enter REPL.
Thanks for waiting
World
```

```
console.log('Hi');

setTimeout(function cb() {
    console.log('there');
}, 5000);

console.log('JSConfEU');
```

stack

setTimeout cb

main()

webapis

Console

Hi

event loop

task queue

```
console.log('Hi');

setTimeout(function cb() {
    console.log('there');
}, 5000);

console.log('JSConfEU');
```

stack

webapis

Console

```
Hi

JSConfEU
```

event loop

task queue

cb

```
console.log('Hi');

setTimeout(function cb() {
    console.log('there');
}, 5000);

console.log('JSConfEU');
```

stack

log('there')

cb

webapis

Console

Hi

JSConfEU

there

event loop

task queue

What the heck is Eventloop? - Philip Roberts

	Concurrency	**Parallelism**
Python	● Threading ● asyncio, ..	● multiprocessing, ..
JavaScript	● Callbacks ● Promises ● async/await, ..	● Web workers, ..

Where is it useful?

- Useful in massive scaling
  - Extremely busy network servers of any kind
  - Websocket servers

Where is it useful?

- To build high performance web frameworks dealing with lot of I/0 operations like:

1. FastAPI
2. Tornado
3. Aiohttp
4. Quart

Common mistakes when using async/await

Await without async function

- When a await is scheduled to resolve a promise without async function

```
1   function f() {
2     let promise = Promise.resolve(1);
3     let result = await promise; // Syntax error
4   }
```

```
debian@DESKTOP-6FG3BCV:~/KurianBenoy/async-await/badPractises$ node notusingasync.js
/home/debian/KurianBenoy/async-await/badPractises/notusingasync.js:3
  let result = await promise; // Syntax error
               ^^^^^

SyntaxError: await is only valid in async function
    at wrapSafe (internal/modules/cjs/loader.js:1001:16)
    at Module._compile (internal/modules/cjs/loader.js:1049:27)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47
```

Never-awaited coroutines(async)

- The async function is not scheduled with await
- The usual fix to use an await or create a task

```python
import asyncio

async def test():
    print("never scheduled")

async def main():
    test()

asyncio.run(main())
```

Writing blocking code

```python
async def greet_after(delay: int, person: str):
    await asyncio.sleep(delay)
    print(f"Hello {person} \U0001F44B")

async def main():
    #blocking input/output code
    task1 = greet_after(10, "Guido van Rossum")
    task2 = greet_after(2, "Sebastián Ramírez")
    task3 = greet_after(3, "Luciano Ramaldho")
    await task1
    await task2
    await task3

asyncio.run(main())
```

```
Started at 02:36:26
Hello Guido van Rossum 👋
Hello Sebastián Ramírez 👋
Hello Luciano Ramaldho 👋
Finished at 02:36:41
Program finished in 15.019392490386963 seconds
```

Writing Blocking Code

```python
import asyncio
import time

async def greet_after(delay: int, person: str):
    await asyncio.sleep(delay)
    print(f"Hello {person} \U0001F44B")

async def main():
    #blocking input/output code
    task1 = greet_after(10, "Guido van Rossum")
    task2 = greet_after(2, "Sebastián Ramírez")
    task3 = greet_after(3, "Luciano Ramaldho")
    await asyncio.gather(task1, task2, task3)
```

```
Started at 02:44:34
Hello Sebastián Ramírez 👋
Hello Luciano Ramaldho 👋
Hello Guido van Rossum 👋
Finished at 02:44:44
Program finished in 10.009770631790161 seconds
```

Conclusion

- Use the right tool for the right purpose
- Perfect for database calls, API calls or any I/O bound task
- Similar syntax in both python and javascript

Picture from Solatire consulting

References

1. **MDN Docs**
2. Asynchronous Python for the complete beginner, Miguel Grinberg
3. [Simple English] What the and why the problem with JavaScript Asynchronicity, Vipul Gupta
4. Python docs
5. Python's tale of concurrency, Pradhvan
6. Talking concurrency Part1, 2 , Pradhvan
7. **import asyncio series, EdgeDB - Lukasz Langa**
8. FastAPI docs
9. Real Python - Intro to threading
10. LuminiousGen Python asynchronous programming
11. Get Started with async & await - Arun Ravindran
12. **async/await - javascript.info**
13. High Performance Python - Micha Gorelick & Ian Ozsvald
14. Python docs

AREEB JAMAL

15.10.1995

28.04.2021

Forever in our hearts
and our memory

Thank You 🤗

Slides: bit.ly/async-await-pycon