(/)

Curriculum
Short Specializations
Average: **97.3%**

0x02. Python - Async Comprehension

Python **Back-end**

By: Emmanuel Turlay, Staff Software Engineer at Cruise

Weight: 1

Project over - took place from Dec 11, 2023 6:00 AM to Dec 14, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 7.0/18 mandatory
- **Altogether: 38.89%**
  - Mandatory: 38.89%
  - Optional: no optional tasks

(/)

I Am Devloper
@iamdevloper

Roses are red
And so are you
Violets are blue
Asynchronous operations
are great

15:53 · 14 Feb 19 · Twitter Web App

Resources

Read or watch:

- PEP 530 – Asynchronous Comprehensions (/rltoken/hlwtED-iLsdORSgly8DsyQ)
- What’s New in Python: Asynchronous Comprehensions / Generators (/rltoken/0OkbObYzCKtO7ZUAxfKvkw)
- Type-hints for generators (/rltoken/l4Fnno568VbVIn9GvrFVtQ)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/_jK22HqiCeh5NjKJ4ZHBww), **without the help of Google**:

- How to write an asynchronous generator
- How to use async comprehensions
- How to type-annotate generators

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`

(/)
- All your files will be interpreted/compiled on Ubuntu 18.04 LTS using `python3` (version 3.7)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/usr/bin/env python3`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the `pycodestyle` style (version 2.5.x)
- The length of your files will be tested using `wc`
- All your modules should have a documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)
- All your functions should have a documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'`
- A documentation is not a simple word, it's a real sentence explaining what's the purpose of the module, class or method (the length of it will be verified)
- All your functions and coroutines must be type-annotated.

Tasks

0. Async Generator

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a coroutine called `async_generator` that takes no arguments.

The coroutine will loop 10 times, each time asynchronously wait 1 second, then yield a random number between 0 and 10. Use the `random` module.

```
bob@dylan:~$ cat 0-main.py
#!/usr/bin/env python3

import asyncio

async_generator = __import__('0-async_generator').async_generator

async def print_yielded_values():
    result = []
    async for i in async_generator():
        result.append(i)
    print(result)

asyncio.run(print_yielded_values())

bob@dylan:~$ ./0-main.py
[4.403136952967102, 6.9092712604587465, 6.293445466782645, 4.549663490048418, 4.1326
571686139015, 9.99058525304903, 6.726734105473811, 9.84331704602206, 1.0067279479988
345, 1.3783306401737838]
```

(/)

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x02-python_async_comprehension`
- File: `0-async_generator.py`

Done! Help Check your code Get a sandbox QA Review

1. Async Comprehensions

mandatory

Score: 16.67% (*Checks completed: 16.67%*)

Import `async_generator` from the previous task and then write a coroutine called `async_comprehension` that takes no arguments.

The coroutine will collect 10 random numbers using an async comprehensing over `async_generator`, then return the 10 random numbers.

```
bob@dylan:~$ cat 1-main.py
#!/usr/bin/env python3

import asyncio

async_comprehension = __import__('1-async_comprehension').async_comprehension


async def main():
    print(await async_comprehension())

asyncio.run(main())

bob@dylan:~$ ./1-main.py
[9.861842105071727, 8.572355293354995, 1.7467182056248265, 4.0724372912858575, 0.552
4750922145316, 8.084266576021555, 8.387128918690468, 1.5486451376520916, 7.713335177
885325, 7.673533267041574]
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x02-python_async_comprehension`
- File: `1-async_comprehension.py`

Done? Help Check your code Ask for a new correction Get a sandbox QA Review

2. Run time for four parallel comprehensions

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Import `async_comprehension` from the previous file and write a `measure_runtime` coroutine that will execute `async_comprehension` four times in parallel using `asyncio.gather`.

`measure_runtime` should measure the total runtime and return it.

Notice that the total runtime is roughly 10 seconds, explain it to yourself.

```
bob@dylan:~$ cat 2-main.py
#!/usr/bin/env python3

import asyncio


measure_runtime = __import__('2-measure_runtime').measure_runtime


async def main():
    return await(measure_runtime())

print(
    asyncio.run(main())
)

bob@dylan:~$ ./2-main.py
10.021936893463135
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x02-python_async_comprehension`
- File: `2-measure_runtime.py`

☐ Done? Help Check your code Ask for a new correction Get a sandbox QA Review