0x01. Python - Async

Python Back-end

Weight: 1

Project over - took place from Jul 8, 2024 6:00 AM to Jul 9, 2024 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 27.0/27 mandatory
- **Altogether: 100.0%**
  - Mandatory: 100.0%
  - Optional: no optional tasks

Resources

Read or watch:

- Async IO in Python: A Complete Walkthrough (/rltoken/zYkXScziW1D5rNdNEvObjQ)
- asyncio - Asynchronous I/O (/rltoken/aZUO4GiWHbPIrVBIwptFAw)
- random.uniform (/rltoken/72mVf1s8rx2ih_U2WjBmaA)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/RzzuxS2J7-SysSxP0Hu3cA), **without the help of Google**:

- `async` and `await` syntax
- How to execute an async program with `asyncio`
- How to run concurrent coroutines
- How to create `asyncio` tasks
- How to use the `random` module

Requirements

General

- A `README.md` file, at the root of the folder of the project, is mandatory
- 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
- All your files must be executable
- The length of your files will be tested using `wc`
- The first line of all your files should be exactly `#!/usr/bin/env python3`
- Your code should use the `pycodestyle` style (version 2.5.x)
- All your functions and coroutines must be type-annotated.
- 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)

Tasks

0. The basics of async

mandatory

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

Write an asynchronous coroutine that takes in an integer argument (`max_delay`, with a default value of 10) named `wait_random` that waits for a random delay between 0 and `max_delay` (included and float value) seconds and eventually returns it.

Use the `random` module.

(/)
Home(/)
My Planning(/planning/me)
Projects(/projects/current)
QA Reviews I can make(/corrections/to_review)
Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)
Curriculums(/dashboards/my_curriculums)
Concepts(/concepts)
Conference rooms(/dashboards/video_rooms)
Servers(/servers)
Sandboxes(/user_containers/current)
Tools(/dashboards/my_tools)
Video on demand(/dashboards/videos)
Peers(/users/peers)
Discord(https://discord.com/app)
My Profile(/users/my_profile)

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

import asyncio

wait_random = __import__('0-basic_async_syntax').wait_random

print(asyncio.run(wait_random()))
print(asyncio.run(wait_random(5)))
print(asyncio.run(wait_random(15)))

bob@dylan:~$ ./0-main.py
9.034261504534394
1.6216525464615306
10.634589756751769
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x01-python_async_function`
- File: `0-basic_async_syntax.py`

Check submission | Get a sandbox | View results

1. Let's execute multiple coroutines at the same time with async

mandatory

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

Import `wait_random` from the previous python file that you've written and write an async routine called `wait_n` that takes in 2 int arguments (in this order): `n` and `max_delay`. You will spawn `wait_random` `n` times with the specified `max_delay`.

`wait_n` should return the list of all the delays (float values). The list of the delays should be in ascending order without using `sort()` because of concurrency.

(/)

Home(/)

My Planning(/planning/me)

Projects(/projects/current)

QA Reviews I can make(/corrections/to_review)

Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)

Curriculums(/dashboards/my_curriculums)

Concepts(/concepts)

Conference rooms(/dashboards/video_rooms)

Servers(/servers)

Sandboxes(/user_containers/current)

Tools(/dashboards/my_tools)

Video on demand(/dashboards/videos)

Peers(/users/peers)

Discord(https://discord.com/app)

My Profile(/users/my_profile)

```
bob@dylan:~$ cat 1-main.py
#!/usr/bin/env python3
'''
Test file for printing the correct output of the wait_n coroutine
'''
import asyncio

wait_n = __import__('1-concurrent_coroutines').wait_n

print(asyncio.run(wait_n(5, 5)))
print(asyncio.run(wait_n(10, 7)))
print(asyncio.run(wait_n(10, 0)))

bob@dylan:~$ ./1-main.py
[0.9693881173832269, 1.0264573845731002, 1.7992690129519855, 3.641373003434587,
4.500011569340617]
[0.07256214141415429, 1.518551245602588, 3.355762808432721, 3.7032593997182923,
3.7796178143655546, 4.744537840582318, 5.50781365463315, 5.758942587637626, 6.109
707751654879, 6.831351588271327]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

The output for your answers might look a little different and that's okay.

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x01-python_async_function`
- File: `1-concurrent_coroutines.py`

Check submission | Get a sandbox | View results

2. Measure the runtime

mandatory

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

From the previous file, import `wait_n` into `2-measure_runtime.py`.

Create a `measure_time` function with integers `n` and `max_delay` as arguments that measures the total execution time for `wait_n(n, max_delay)`, and returns `total_time / n`. Your function should return a float.

Use the `time` module to measure an approximate elapsed time.

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

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

n = 5
max_delay = 9

print(measure_time(n, max_delay))

bob@dylan:~$ ./2-main.py
1.759705400466919
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x01-python_async_function`
- File: `2-measure_runtime.py`

Check submission Get a sandbox View results

3. Tasks

mandatory

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

Import `wait_random` from `0-basic_async_syntax`.

Write a function (do not create an async function, use the regular function syntax to do this) `task_wait_random` that takes an integer `max_delay` and returns a `asyncio.Task`.

```
bob@dylan:~$ cat 3-main.py
#!/usr/bin/env python3

import asyncio

task_wait_random = __import__('3-tasks').task_wait_random


async def test(max_delay: int) -> float:
    task = task_wait_random(max_delay)
    await task
    print(task.__class__)

asyncio.run(test(5))

bob@dylan:~$ ./3-main.py
<class '_asyncio.Task'>
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x01-python_async_function`
- File: `3-tasks.py`

Check submission | Get a sandbox | View results

4. Tasks

mandatory

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

Take the code from `wait_n` and alter it into a new function `task_wait_n`. The code is nearly identical to `wait_n` except `task_wait_random` is being called.

```
bob@dylan:~$ cat 4-main.py
#!/usr/bin/env python3

import asyncio

task_wait_n = __import__('4-tasks').task_wait_n

n = 5
max_delay = 6
print(asyncio.run(task_wait_n(n, max_delay)))

bob@dylan:~$ ./4-main.py
[0.2261658205652346, 1.1942770588220557, 1.8410422186086628, 2.1457353803430523, 4.002505454641153]
```

Repo:

- GitHub repository: `alx-backend-python`
- Directory: `0x01-python_async_function`
- File: `4-tasks.py`

Check submission | Get a sandbox | View results