Asyncio

Martin Natano, Ivana Kellyerova

May 4, 2019

What is asyncio?

Coroutines

```
import asyncio

async def alice():
    print('Hi!')
    print('My name is Alice')
    await asyncio.sleep(1)
    print('Bye')

async def bob():
    print('Nice to meet you')
```

Concurrency vs. Parallelism

- **parallelism**: multiple tasks being processed at the same time, e.g. on a multicore machine

- **concurrency**: tasks can overlap, but don't necessarily run at the same time

Concurrency vs. Parallelism

- **parallelism**: multiple tasks being processed at the same time, e.g. on a multicore machine
  $\rightarrow$ CPU intensive computations

- **concurrency**: tasks can overlap, but don't necessarily run at the same time

Concurrency vs. Parallelism

- **parallelism**: multiple tasks being processed at the same time, e.g. on a multicore machine
  $\rightarrow$ CPU intensive computations
  $\rightarrow$ threads, processes

- **concurrency**: tasks can overlap, but don't necessarily run at the same time

Concurrency vs. Parallelism

- **parallelism**: multiple tasks being processed at the same time, e.g. on a multicore machine
  $\rightarrow$ CPU intensive computations
  $\rightarrow$ threads, processes

- **concurrency**: tasks can overlap, but don't necessarily run at the same time

Concurrency vs. Parallelism

- **parallelism**: multiple tasks being processed at the same time, e.g. on a multicore machine
  $\rightarrow$ CPU intensive computations
  $\rightarrow$ threads, processes

- **concurrency**: tasks can overlap, but don't necessarily run at the same time
  $\rightarrow$ waiting for IO, e.g. networking

Concurrency vs. Parallelism

- **parallelism**: multiple tasks being processed at the same time, e.g. on a multicore machine
  $\rightarrow$ CPU intensive computations
  $\rightarrow$ threads, processes

- **concurrency**: tasks can overlap, but don't necessarily run at the same time
  $\rightarrow$ waiting for IO, e.g. networking
  $\rightarrow$ coroutines :)

Libraries using asyncio

- clients and servers for HTTP, websockets

Libraries using asyncio

- clients and servers for HTTP, websockets
- DB connectors: InfluxDB, MySQL, Postgres, ...

Libraries using asyncio

- clients and servers for HTTP, websockets
- DB connectors: InfluxDB, MySQL, Postgres, ...
- message queue connectors: Kafka, ZeroMQ, ...

Libraries using asyncio

- clients and servers for HTTP, websockets
- DB connectors: InfluxDB, MySQL, Postgres, ...
- message queue connectors: Kafka, ZeroMQ, ...
- implementations of networking protocols: DNS, SSH, ...

Libraries using asyncio

- clients and servers for HTTP, websockets
- DB connectors: InfluxDB, MySQL, Postgres, ...
- message queue connectors: Kafka, ZeroMQ, ...
- implementations of networking protocols: DNS, SSH, ...
- ...and so much more!

Running coroutines

asyncio.run()

```
asyncio.run(coro())
```

await

```
result = await coro()
```

asyncio.wait_for()

```
try:
    result = await asyncio.wait_for(
        coro(),
        timeout=1,
    )
except asyncio.TimeoutError:
    print('Timeout!')
```

asyncio.gather()

```
results = await asyncio.gather(
    coro_1(),
    coro_2(),
)
```

asyncio.wait()

```
await asyncio.wait([coro_1(), coro_2()])

done, pending = await asyncio.wait(
    [coro_1(), coro_2()],
    timeout=1,
)

done, pending = await asyncio.wait(
    [coro_1(), coro_2()],
    return_when=asyncio.FIRST_COMPLETED,
)
```

asyncio.create_task()

```
task = asyncio.create_task(coro())
...
result = await task
task.cancel()
```

Synchronization

Lock

```
sound_lock = asyncio.Lock()

async def play_sound():
    async with sound_lock:
        ...          # play sound
```

Event

```
bob_is_done = asyncio.Event()

async def alice():
    await bob_is_done.wait()
    print('finally')

async def bob():
    await asyncio.sleep(60)          # chill
    bob_is_done.set()
```

Semaphore

```
max_three = asyncio.Semaphore(3)

async def download_large_file():
    async with max_three:
        ...             # download large file
```

Condition

```
cond = asyncio.Condition()
```

Queue

```
queue = asyncio.Queue()

async def compute_squares():
    for i in range(1000):
        await queue.put(i ** 2)

async def print_squares():
    while True:
        print(await queue.get())
```

Hands-on

Which Python version

Python 3.7

(or newer)

virtualenv

```
$ pyenv install 3.7.3
$ pyenv local 3.7.3
$ python --version
Python 3.7.3
$ python -m venv env
$ . env/bin/activate
(env) $ pip install -U pip setuptools
(env) $ pip install jupyter
(env) $ jupyter notebook
```

Exercises:

https://www.natano.net/data/workshops/pydays2019/

The End

Should you use asyncio for ...

Should you use asyncio for ...

a HTTP microservice?

Should you use asyncio for ...

a HTTP microservice? yes

Should you use asyncio for ...

a HTTP microservice? yes
calculating prime numbers?

Should you use asyncio for ...

a HTTP microservice? yes
calculating prime numbers? no

Should you use asyncio for ...

a HTTP microservice? yes
calculating prime numbers? no
AI, machine learning?

Should you use asyncio for ...

a HTTP microservice? yes
calculating prime numbers? no
AI, machine learning? probably not

Should you use asyncio for ...

a HTTP microservice? yes
calculating prime numbers? no
AI, machine learning? probably not
a server for an online multiplayer game?

Should you use asyncio for ...

a HTTP microservice? yes
calculating prime numbers? no
AI, machine learning? probably not
a server for an online multiplayer game? yes, absolutely!

Questions

Any questions?

Thanks

Thx!

https://www.natano.net/data/workshops/pydays2019/