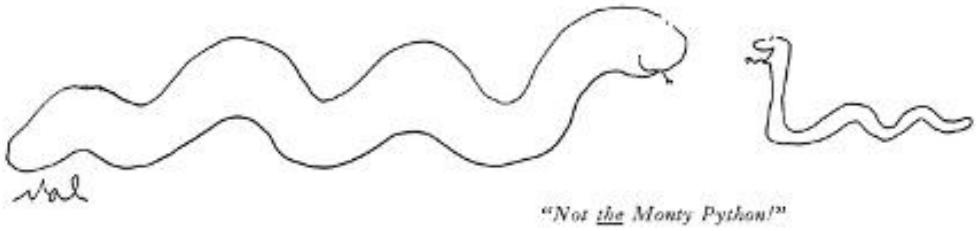
Python asyncio & Friends

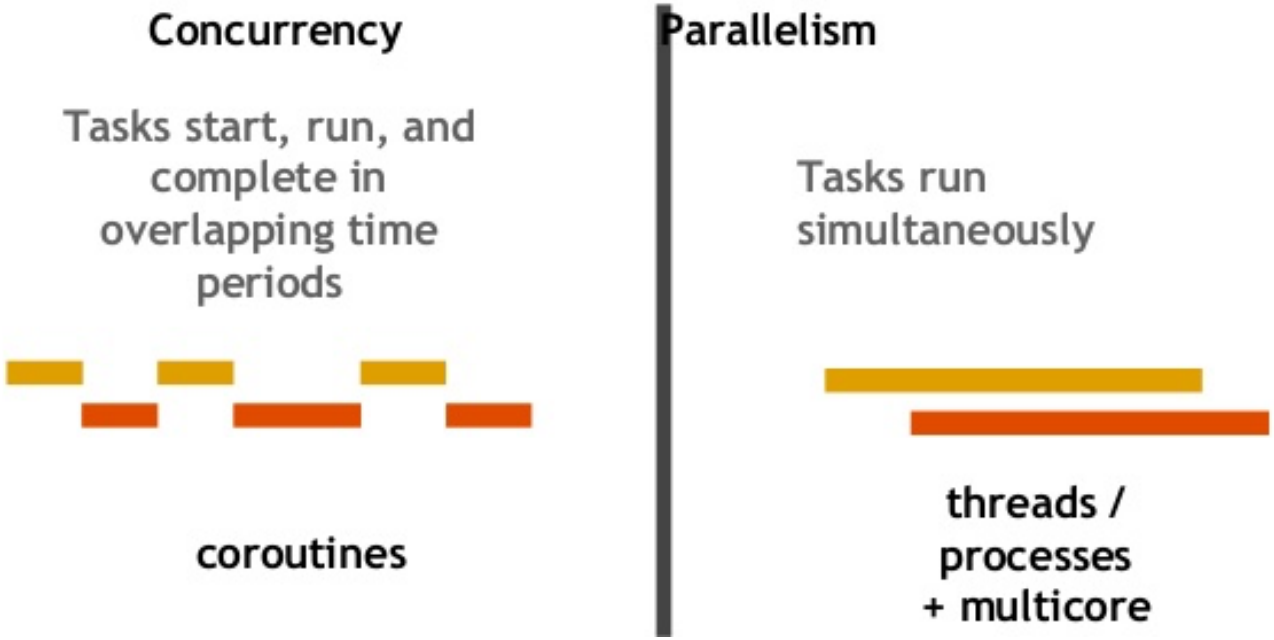


"Not the Monty Python!"

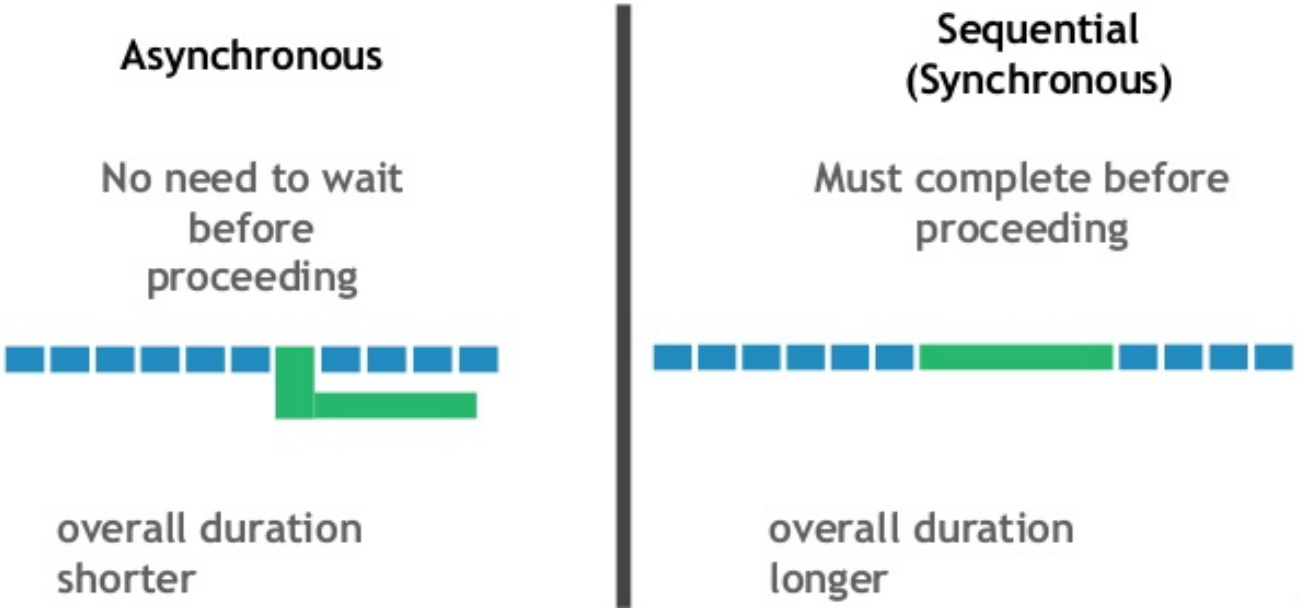
universidade
de aveiro

O que é Async ?

- É um padrão de programação concorrente



O que é Async ?


Asynchronous
No need to wait before proceeding
overall duration shorter
Sequential (Synchronous)
Must complete before proceeding
overall duration longer

Como funciona o Python?

- O SO gere todo o trabalho de multi-tasking
- Em CPython, a GIL (global interpreter lock) impede concorrencia usando multiplos cores – o interpretador está locked a um core.

- Asyncio:
  - Sem intervenção do SO
  - Um processo, uma thread

Xadrez – Mestre a jogar contra 50 oponentes

- Mestre leva 30 segundos a fazer uma jogada
- Jogador Normal leva 5 minutos

- Quanto tempo levariam os 50 jogos sincronamente (media de 30 movimentos)?
  - 30*0,5 + 30*5 = 165 min por jogo
  - 165min * 50 = 5,7 dias

- Se considerarmos que o Mestre leva 1 min a mudar de contexto
  - 30*0,5 + 30*1 = 45 min por ronda
  - 45min * 30 = 22h30

Como suportar Async?

- Funções Async precisam de poder suspender e resumir.
- Uma função que entre num periodo de espera deve ser suspensa, e só resumir quando a esperar tiver terminado
- Como implementar suspensão/resumir em Python:
  - Callback's
  - Geradores
  - Async/Await (Python 3.5+)

Programação de tarefas asincronas

- Os frameworks async precisam de um scheduler, normalmente chamado de “event loop”
- O loop toma conta de todas as tarefas em execução
- Quando uma função suspende, devolve o control ao loop, que procura por outra tarefa que lhe suceda
- A isto chama-se “cooperative multi-tasking”

```python
import time

def hello():
    print('Hello')
    time.sleep(3)
    print('World')

if __name__ == '__main__':
    for _ in range(3):
        hello()
```

```python
import asyncio
loop = asyncio.get_event_loop()

async def hello():
    print('Hello')
    await asyncio.sleep(3)
    print('World')

async def mainloop():
    t = []
    for _ in range(3):
        t.append(hello())
    await asyncio.gather(*t)

if __name__ == '__main__':
    loop.run_until_complete(mainloop())
```

Sync vs Async

Problemas…

- Tarefas intensivas (CPU) que demorem muito tempo precisam libertar periodicamente o CPU para evitar que as outras tarefas bloqueiem.
  - Solução: Pode-se fazer um `await asyncio.sleep(0)` periodicamente.
- Várias bibliotecas nativas não são compativeis!
  - `Socket, select, subprocess, os.waitpid, threading, multiprocessing, time.sleep`
  - Existem bibliotecas alternativas que suportam async

Como assim não posso usar `Socket` ?

- Stream based

```python
import asyncio

async def echo_server(reader, writer):
    while True:
        data = await reader.read(100)
        if not data:
            break
        writer.write(data)
        await writer.drain()  # Flow control
    writer.close()


async def main(host, port):
    server = await asyncio.start_server(echo_server, host, port)
    await server.serve_forever()

loop = asyncio.get_event_loop()
loop.run_until_complete(main('127.0.0.1', 5000))
loop.close()
```

Como assim não posso usar `Socket`?

- Callback based

```python
import asyncio

class EchoProtocol(asyncio.Protocol):
    def connection_made(self, transport):
        self.transport = transport
    def data_received(self, data):
        self.transport.write(data)

async def main(host, port):
    loop = asyncio.get_running_loop()
    server = await loop.create_server(EchoProtocol, host, port)
    await server.serve_forever()

loop = asyncio.get_event_loop()
loop.run_until_complete(main('127.0.0.1', 5000))
loop.close()
```

All code:

http://tiny.cc/pvcb7y

Original:

https://gist.github.com/dgomes/cd8a7e46967b5cfbccc474b748662ad8

one more thing…

Ninguem desenvolve um sistema distribuido do inicio (a menos que exista uma MUITO BOA razão para isso)

É usual criar-se uma pilha (stack) apartir de várias peças/projectos existentes.

Vamos analisar alguns….

NGINX

- Nginx is a web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache.

Flask

- Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

RabbitMQ

- RabbitMQ is an open-source message-broker software that originally implemented the Advanced Message Queuing Protocol and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol, Message Queuing Telemetry Transport, and other protocols.

redis

- Redis is an in-memory data structure project implementing a **distributed, in-memory key-value database** with optional durability. Redis supports different kinds of abstract data structures, such as strings, lists, maps, sets, sorted sets, HyperLog Logs, bitmaps, streams, and spatial indexes.

HAProxy

- HAProxy is free, open source software that provides a high availability load balancer and proxy server for TCP and HTTP-based applications that spreads requests across multiple servers. It is written in C and has a reputation for being fast and efficient.

É tudo Python !?

- https://stackshare.io/stacks