HANDS-ON AYSNCIO/AIOHTTP

Bruno Renié

Django Meetup Suisse Romande, March 16, 2017

PYTHON'S `ASYNCIO` PACKAGE

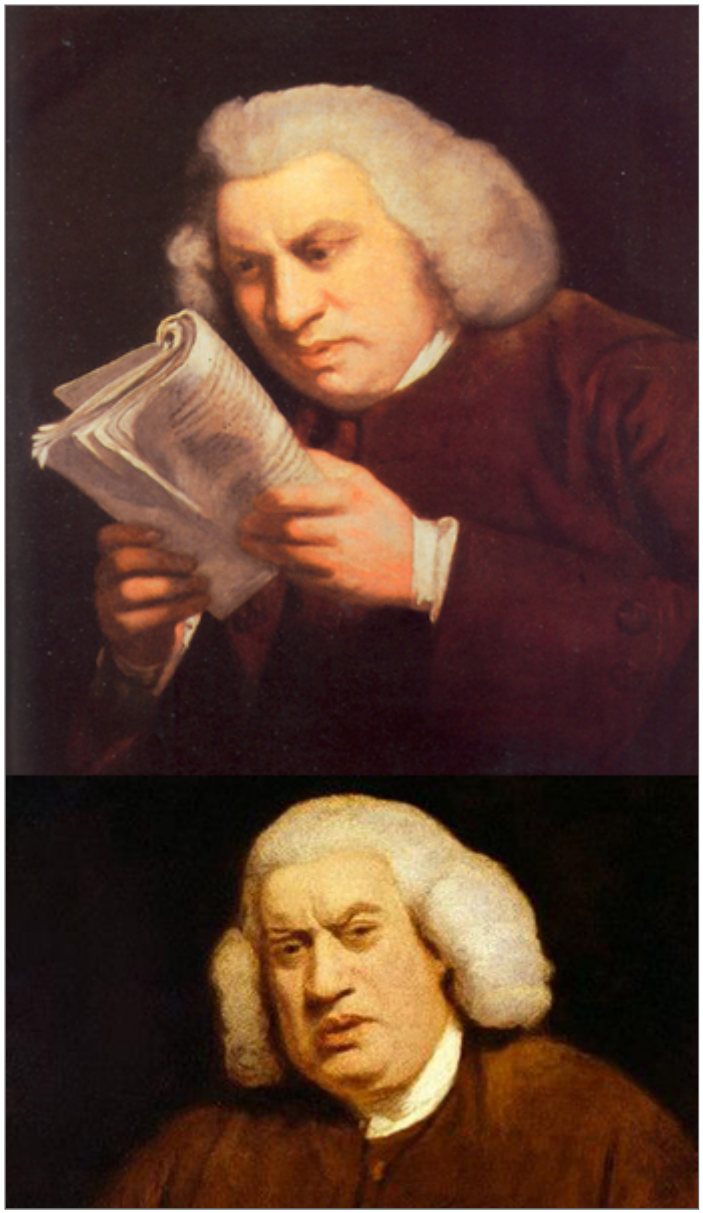
Asynchronous networking

Python 3.4+

Coroutines / generator-based syntax (`yield`)

```
import asyncio

@asyncio.coroutine
def async_task():
    yield from asyncio.sleep(1)
    return "I slept"
```



Let's not bother and use Python 3.5+

```
import asyncio

async def async_task():
    await asyncio.sleep(1)
    return "I slept"
```

`async def` marks an asynchronous function

Async functions **must** be `await`-ed (syntax error otherwise)

More syntax extension: `async for`, `async with`

```
async def print_rows():
    async with database.connect() as session:
        async for row in session.execute('SELECT * from users'):
            print(row)
```

👍 COOL STORY 👍

REAL-WORLD ADVANTAGES

No more worker model

Non-blocking requests

Async / websockets / long-polling 😍

LET'S SEE AN EXAMPLE

Hello world for async web apps?

👉 Chat app 👈

Enter `aiohttp`

Like Flask, but for asyncio

```
from aiohttp import web

def woot(request):
    return web.Response(text="yay")

app = web.Application()
app.router.add_route('GET', '/', woot)
web.run_app(app)
```

OUR SPECS

User gets auto-generated usernames

Messages are POST'ed to `/events`

App exposes a live endpoint with SSE

SSE?

Like WebSockets, but:

- One-way: server to client only
- No special nginx config to setup

Websocket-like workflow with AJAX POSTS & SSE stream

```
const events = new EventSource('/events')
events.onmessage = msg => {
    payload = JSON.parse(msg.data);

    // do something useful with payload
}
```

SSE WITH AIOHTTP

```
from aiohttp_sse import EventSourceResponse

async def events(request):
    response = EventSourceResponse()
    response.start(request)

    while True:
        event = await wait_for_event()
        response.send(json.dumps(event))

    response.stop_streaming()
    return response

app.router.add_route('GET', '/events', events)
```

CORE CHAT COMPONENTS

- Registry of logged-in users
- Queues for sending messages

```
user_events = defaultdict(asyncio.Queue)   # Combo!
```

queue.get(), queue.put(…) are awaitable

```
async def broadcast_message(request):
    data = await.request.json()
    for queue in user_events.values():
        await queue.put(data)
    return web.json_response({'success': True})

app.router.add_route('POST', '/events', broadcast_message)
```

Chat member registration: when they subscribe to the SSE endpoint

```
async def events(request):
    response = EventSourceResponse()
    response.start(request)

    username = random_username()
    # Tell the user which username they got
    await user_events[username].put({ 'type': 'username',
                                      'username': username})
    # Send the list of connected users
    await user_events[username].put(
        {'type': 'users', 'users': list(user_events.keys())})
    # Wait for messages in queue and send them
    while True:
        event = await user_events[username].get()
        response.send(json.dumps(event))
```

DEMO