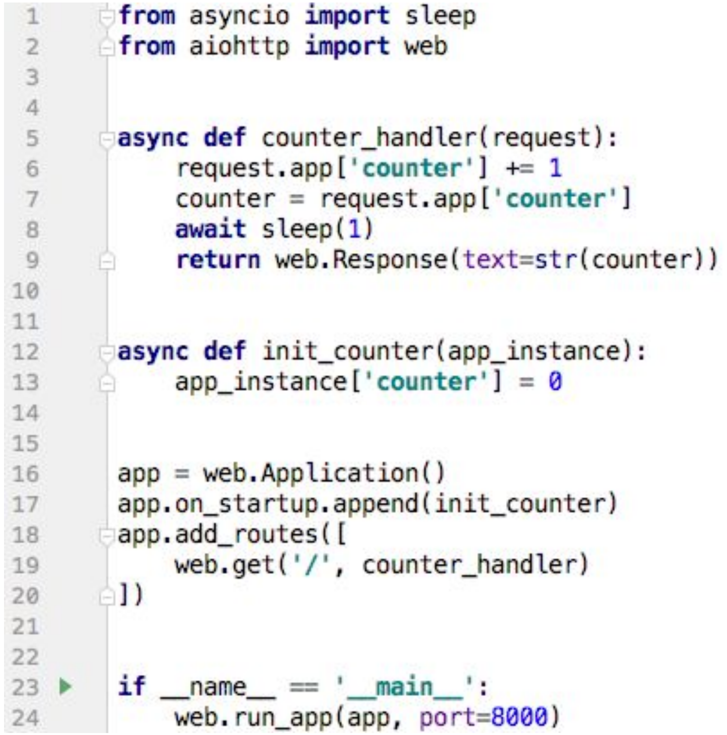
# Concurrency in Python

Prepared for Budapest GP meeting talks
August 2019

GP GovPredict

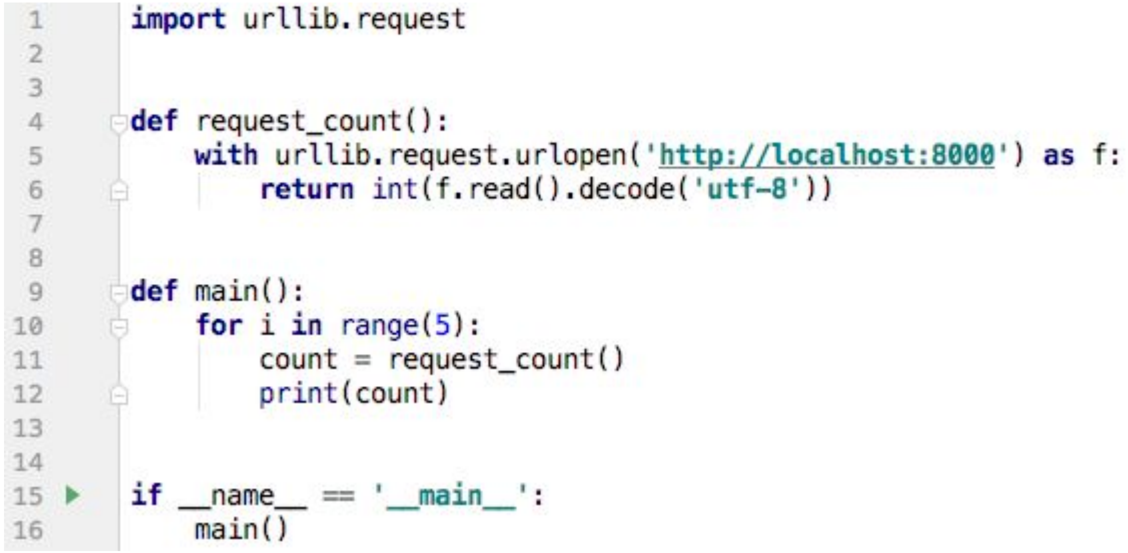
## Server



```python
from asyncio import sleep
from aiohttp import web


async def counter_handler(request):
    request.app['counter'] += 1
    counter = request.app['counter']
    await sleep(1)
    return web.Response(text=str(counter))


async def init_counter(app_instance):
    app_instance['counter'] = 0


app = web.Application()
app.on_startup.append(init_counter)
app.add_routes([
    web.get('/', counter_handler)
])


if __name__ == '__main__':
    web.run_app(app, port=8000)
```

GP GovPredict

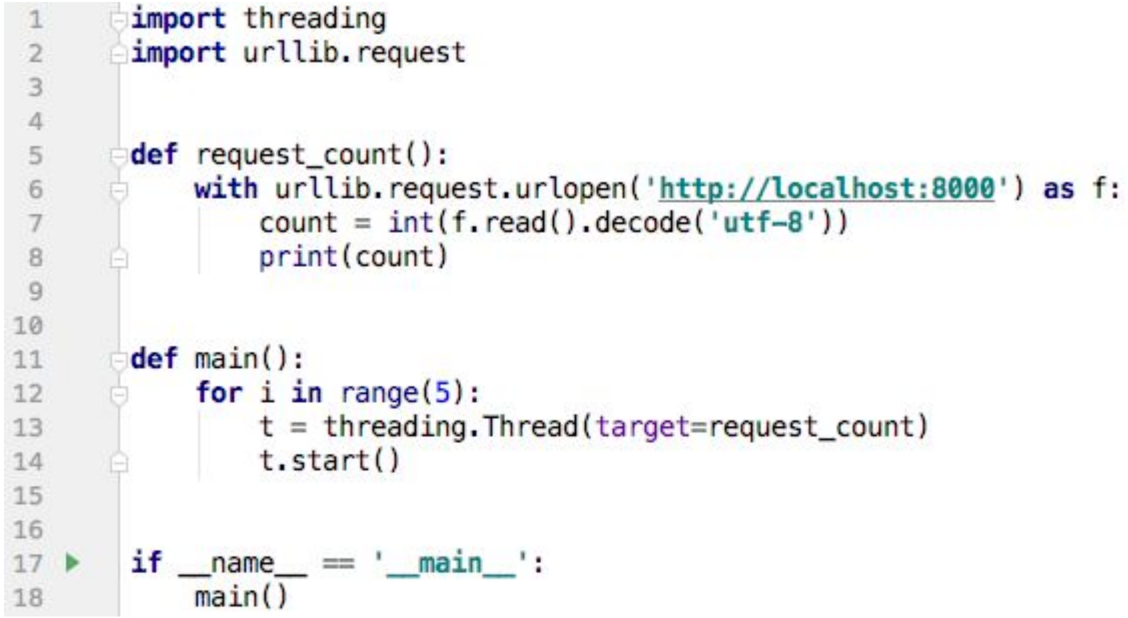
# Naive client



```python
import urllib.request


def request_count():
    with urllib.request.urlopen('http://localhost:8000') as f:
        return int(f.read().decode('utf-8'))


def main():
    for i in range(5):
        count = request_count()
        print(count)


if __name__ == '__main__':
    main()
```

GovPredict

# Concurrency with Python threads



```python
import threading
import urllib.request


def request_count():
    with urllib.request.urlopen('http://localhost:8000') as f:
        count = int(f.read().decode('utf-8'))
        print(count)


def main():
    for i in range(5):
        t = threading.Thread(target=request_count)
        t.start()


if __name__ == '__main__':
    main()
```

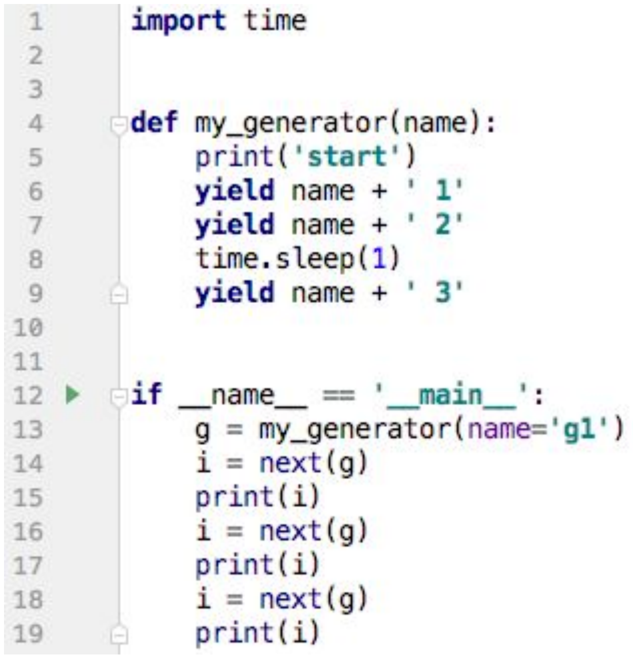
GovPredict

# Timeline

- **1991**: First Python release
- **May 2001**: PEP 255 was created, **Simple Generators** <- start
- **October 2002**: Twisted (Python network programming framework uses ioloop and futures) released
- **May 2005**: PEP 342 was created, generator functions are **coroutines** 👍
- **September 2012**: Python 3.3 with yield from statement (PEP 380)
- **December 2012**: asyncio (formerly tulip) was proposed as an enhancement of Python in order to add asynchronous I/O support
- **October 2013**: asyncio 0.1.1 released
- **October 2013**: aiohttp 0.1 released
- **March 2014**: Python 3.4 with **asyncio** in the standard library
- **September 2015**: Python 3.5 with **async/await** statements (PEP 492) 😍
- **December 2016**: Python 3.6 <- we use now
- **June 2018**: Python 3.7

GP GovPredict

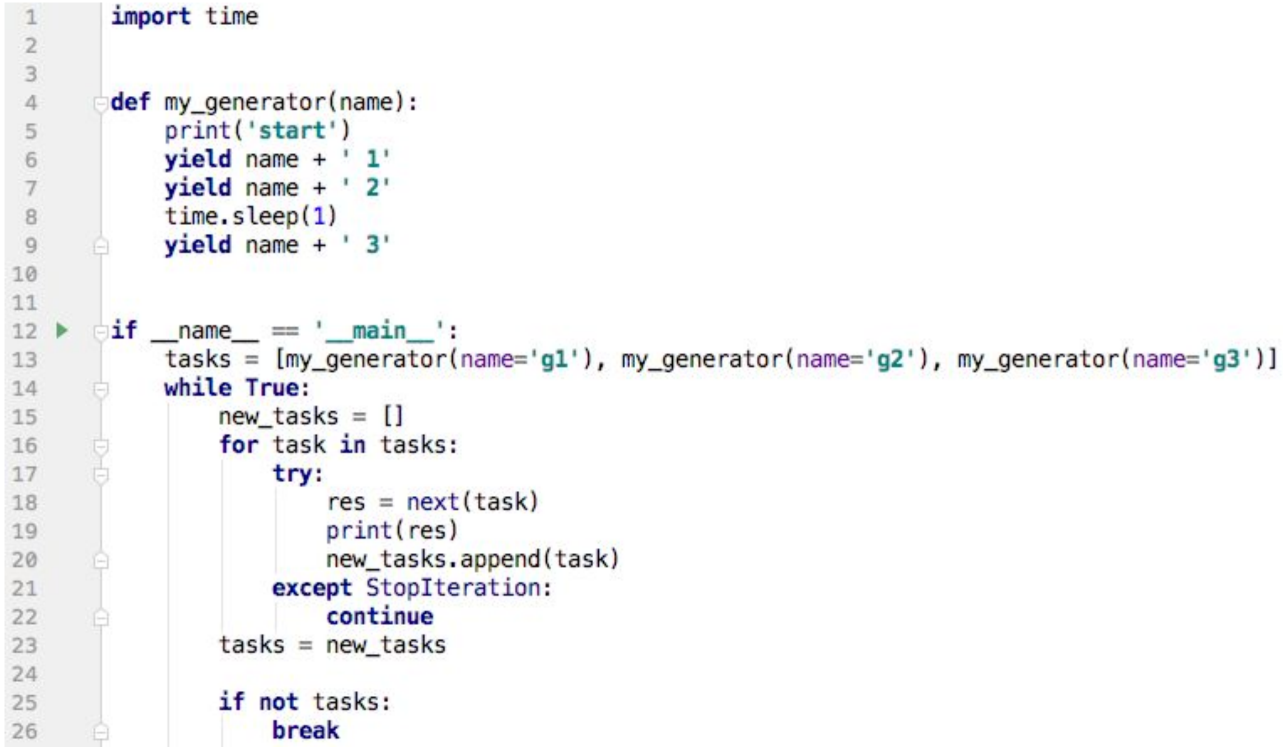
## Generators



```python
import time


def my_generator(name):
    print('start')
    yield name + ' 1'
    yield name + ' 2'
    time.sleep(1)
    yield name + ' 3'


if __name__ == '__main__':
    g = my_generator(name='g1')
    i = next(g)
    print(i)
    i = next(g)
    print(i)
    i = next(g)
    print(i)
```

GP GovPredict

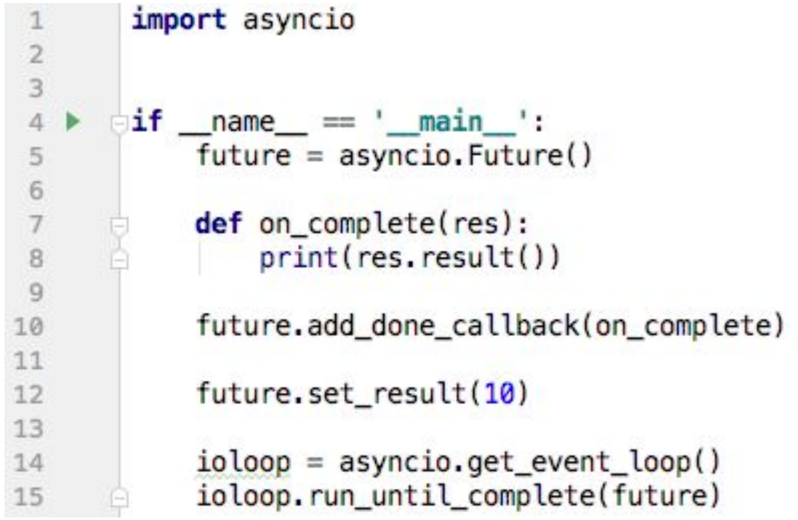
# IOLoop



```python
import time


def my_generator(name):
    print('start')
    yield name + ' 1'
    yield name + ' 2'
    time.sleep(1)
    yield name + ' 3'


if __name__ == '__main__':
    tasks = [my_generator(name='g1'), my_generator(name='g2'), my_generator(name='g3')]
    while True:
        new_tasks = []
        for task in tasks:
            try:
                res = next(task)
                print(res)
                new_tasks.append(task)
            except StopIteration:
                continue
        tasks = new_tasks

        if not tasks:
            break
```

GovPredict

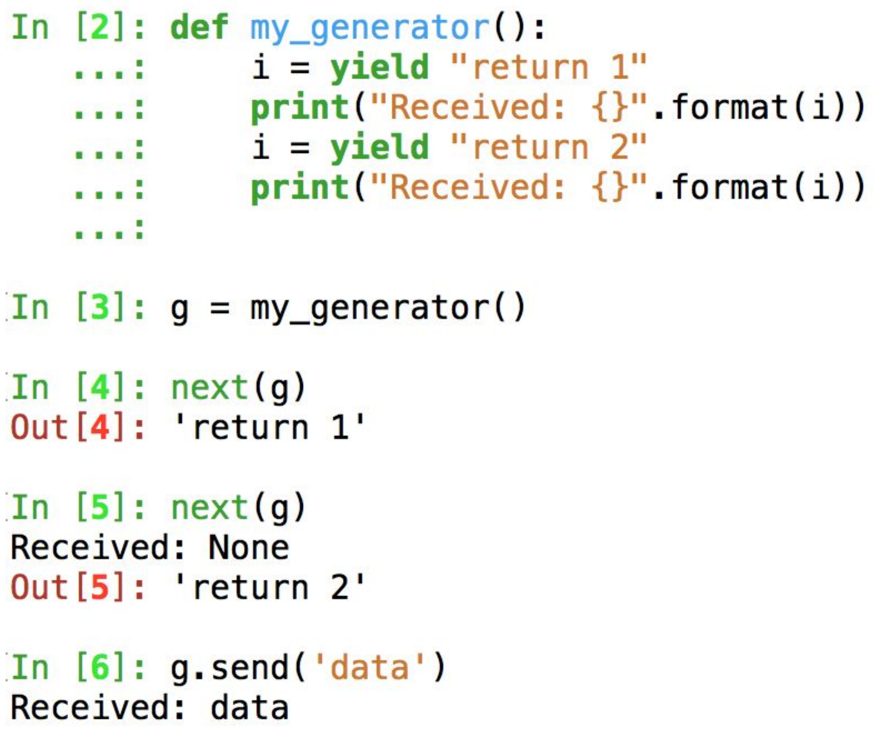
## Future



```python
import asyncio


if __name__ == '__main__':
    future = asyncio.Future()

    def on_complete(res):
        print(res.result())

    future.add_done_callback(on_complete)

    future.set_result(10)

    ioloop = asyncio.get_event_loop()
    ioloop.run_until_complete(future)
```

GP GovPredict

## Coroutine

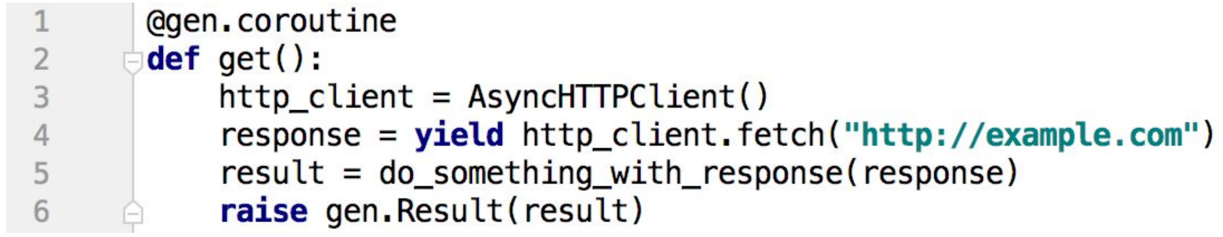


```
In [2]: def my_generator():
   ...:     i = yield "return 1"
   ...:     print("Received: {}".format(i))
   ...:     i = yield "return 2"
   ...:     print("Received: {}".format(i))
   ...:

In [3]: g = my_generator()

In [4]: next(g)
Out[4]: 'return 1'

In [5]: next(g)
Received: None
Out[5]: 'return 2'

In [6]: g.send('data')
Received: data
```

GovPredict

# Async function

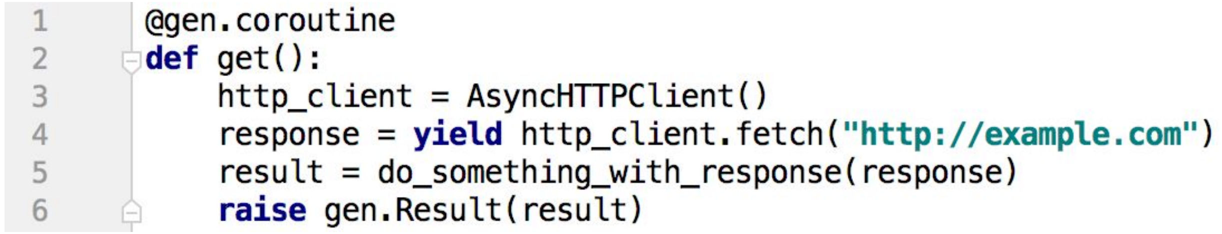
Tornado code example using Python 3.3



```python
@gen.coroutine
def get():
    http_client = AsyncHTTPClient()
    response = yield http_client.fetch("http://example.com")
    result = do_something_with_response(response)
    raise gen.Result(result)
```

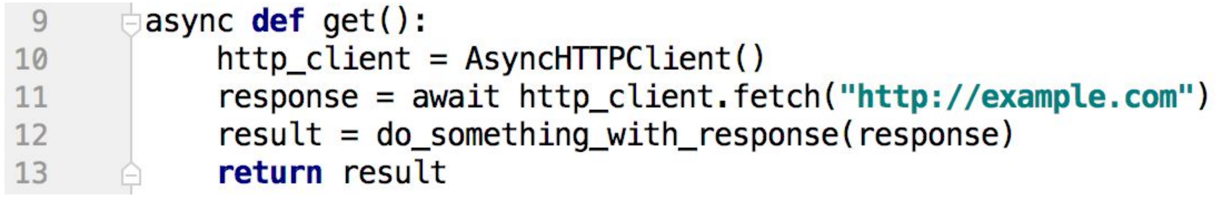
## Async/await syntax

Tornado code example using Python 3.3



```python
@gen.coroutine
def get():
    http_client = AsyncHTTPClient()
    response = yield http_client.fetch("http://example.com")
    result = do_something_with_response(response)
    raise gen.Result(result)
```

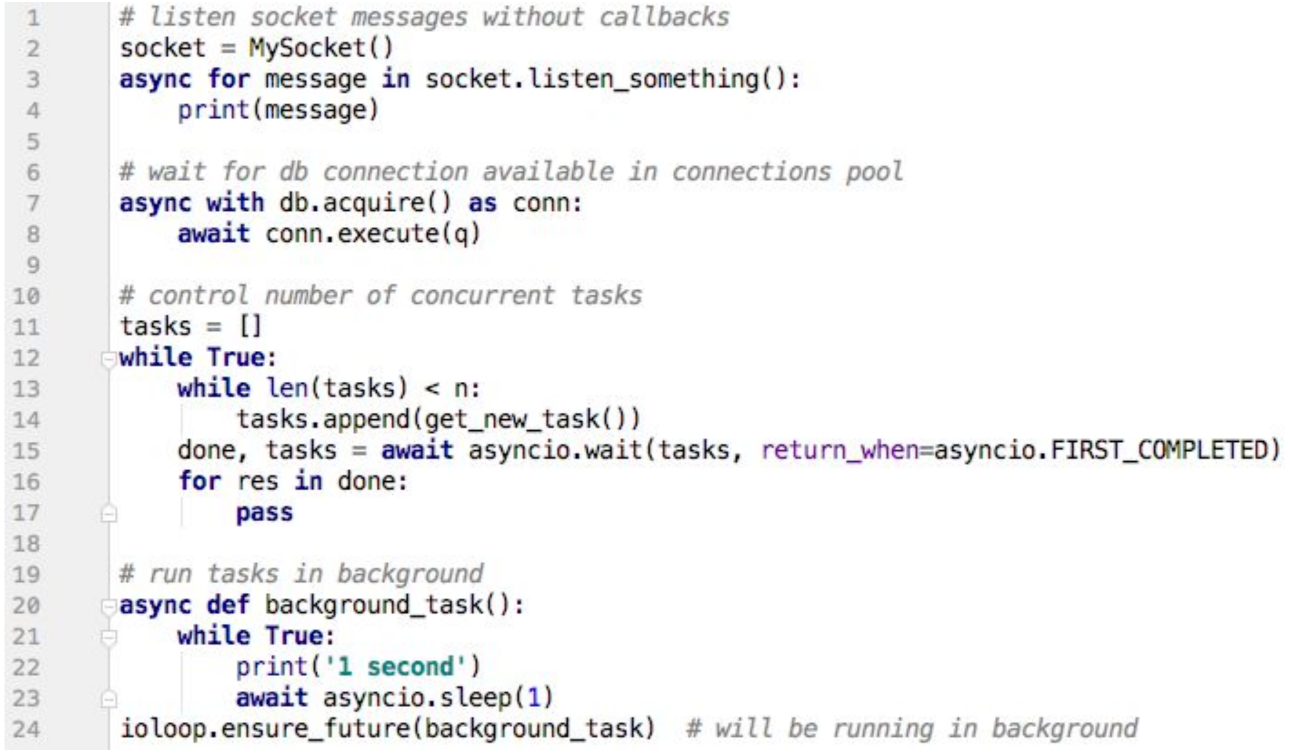
Python 3.5



```python
async def get():
    http_client = AsyncHTTPClient()
    response = await http_client.fetch("http://example.com")
    result = do_something_with_response(response)
    return result
```

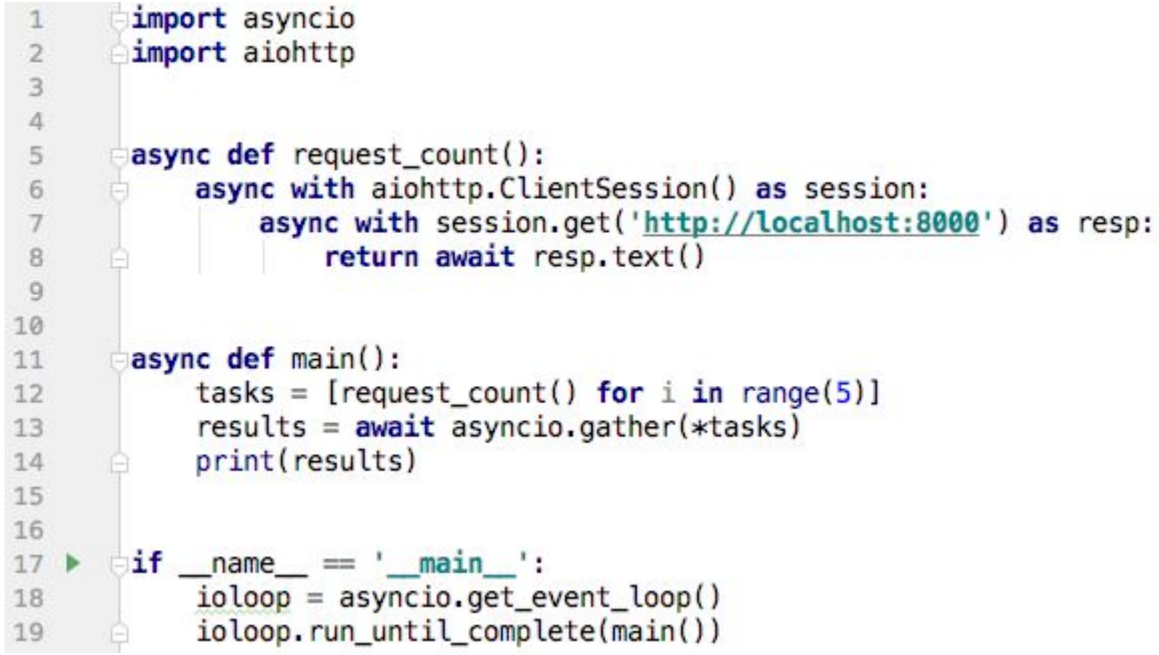
GP GovPredict

## Examples



```python
# listen socket messages without callbacks
socket = MySocket()
async for message in socket.listen_something():
    print(message)

# wait for db connection available in connections pool
async with db.acquire() as conn:
    await conn.execute(q)

# control number of concurrent tasks
tasks = []
while True:
    while len(tasks) < n:
        tasks.append(get_new_task())
    done, tasks = await asyncio.wait(tasks, return_when=asyncio.FIRST_COMPLETED)
    for res in done:
        pass

# run tasks in background
async def background_task():
    while True:
        print('1 second')
        await asyncio.sleep(1)
ioloop.ensure_future(background_task)  # will be running in background
```

# aiohttp client



```python
import asyncio
import aiohttp


async def request_count():
    async with aiohttp.ClientSession() as session:
        async with session.get('http://localhost:8000') as resp:
            return await resp.text()


async def main():
    tasks = [request_count() for i in range(5)]
    results = await asyncio.gather(*tasks)
    print(results)


if __name__ == '__main__':
    ioloop = asyncio.get_event_loop()
    ioloop.run_until_complete(main())
```

GovPredict

# Python concurrency for scraping

- Scrapy (Twisted) - the most of web scrapers we have
- Google News scraper (asyncio)
- Textract (asyncio) - text extraction
- State Data, Townint Seeds and Backend, Watson (asyncio)

- blocking http client
- threads
- generators
- ioloop
- coroutines
- asyncio syntax and examples
- how we currently use Python concurrency

## Thank you!

GP GovPredict