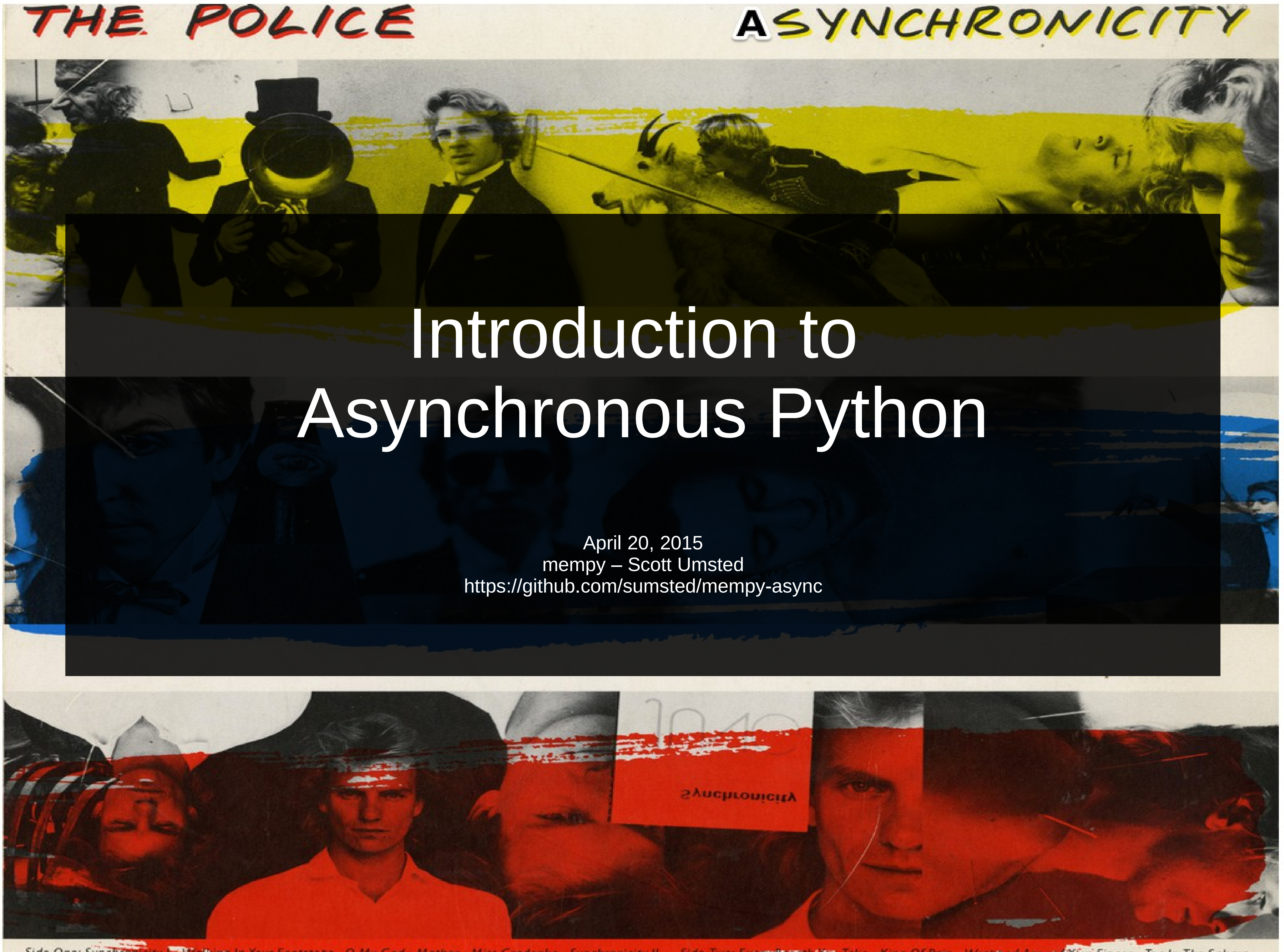


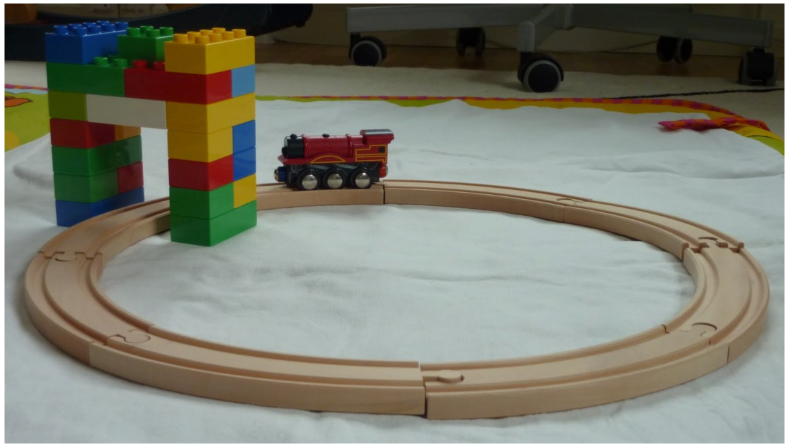
Introduction to Asynchronous Python

April 20, 2015
mempy – Scott Umsted
https://github.com/sumsted/mempy-async

Asynchronous Python

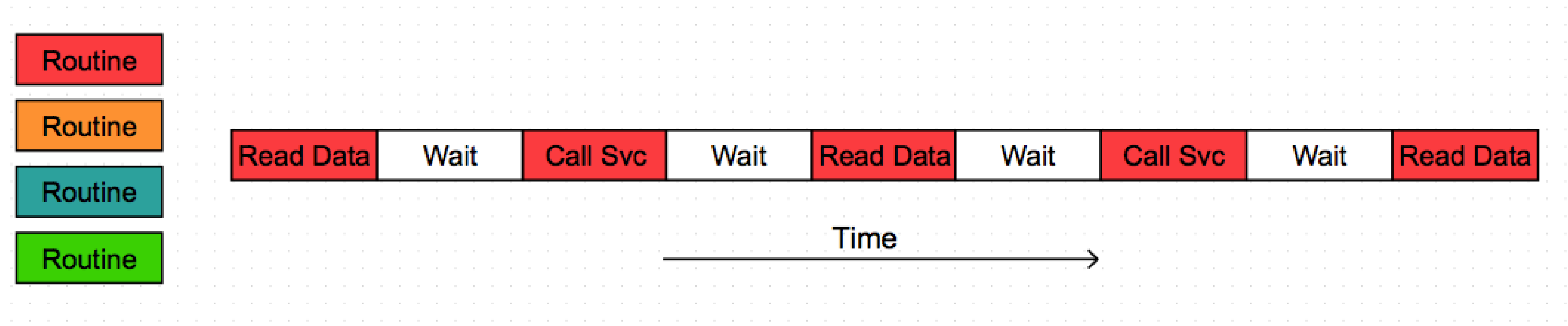
$$\begin{tikzcd}
\text{Thread(s) vs async code} \arrow[dr, bend right] & & \text{Async in Python} \arrow[r] & \text{asyncio and aiohttp} \arrow[r] & \text{Blocking, threads, async} \arrow[r] & \text{Other async patterns} \arrow[r] & \text{Other async topics} \\
& \text{Generators} \arrow[ur, bend right] & & & & &
\end{tikzcd}$$

A thread



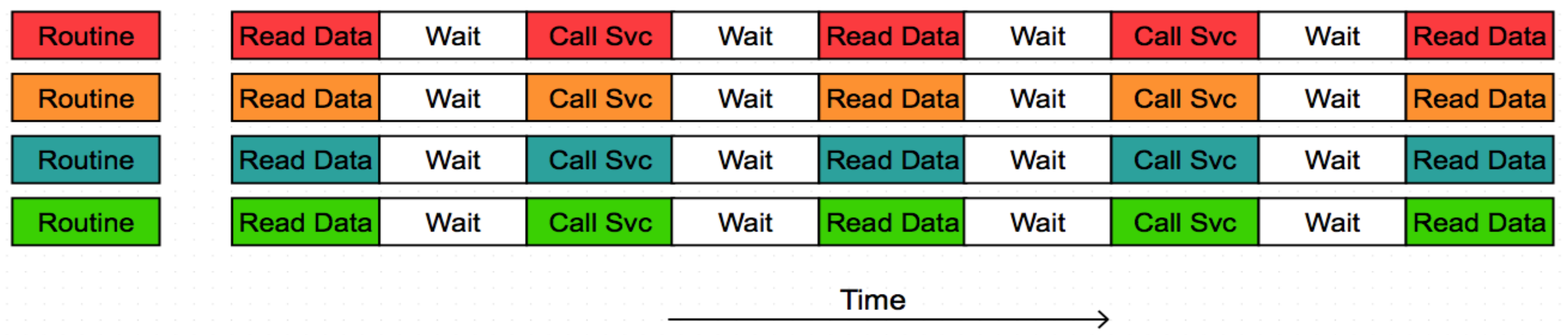
- A thread is like a train
- Length of train is time
- Each car represents an opportunity to execute an instruction
- Empty cars are waste

A thread



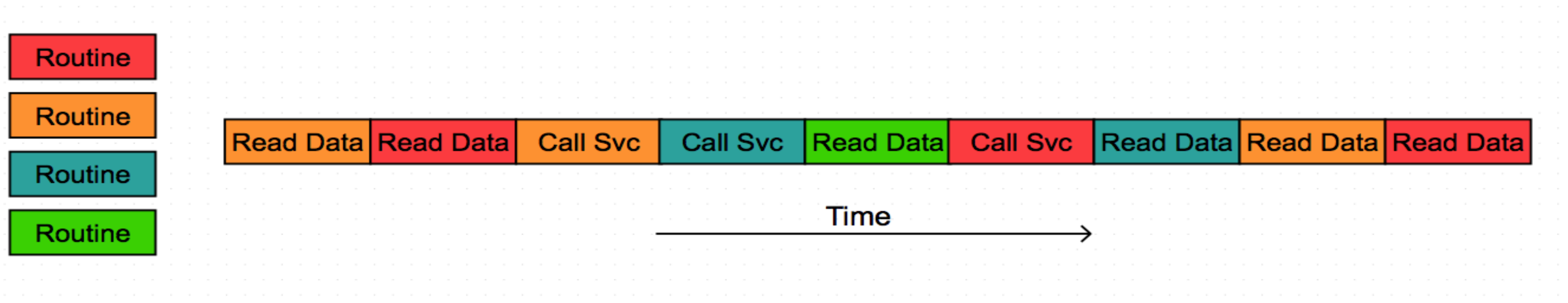
- Single line of execution that processes one instruction at a time
- Blocking of thread occurs on I/O reads
- Waiting for file system, DNS lookup, http server connections and reads

Multiple threads or processes



- Parallel execution of tasks
- More efficient use of time
- But still waste in each thread
- Each thread or process blocks and has a cost
- Python threads and the GIL

Asynchronous



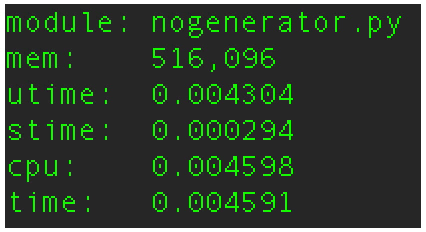
- More efficient use of single thread
- Shelve waiting code so other code may execu
- Fill in the gaps where waits occur

Python generators

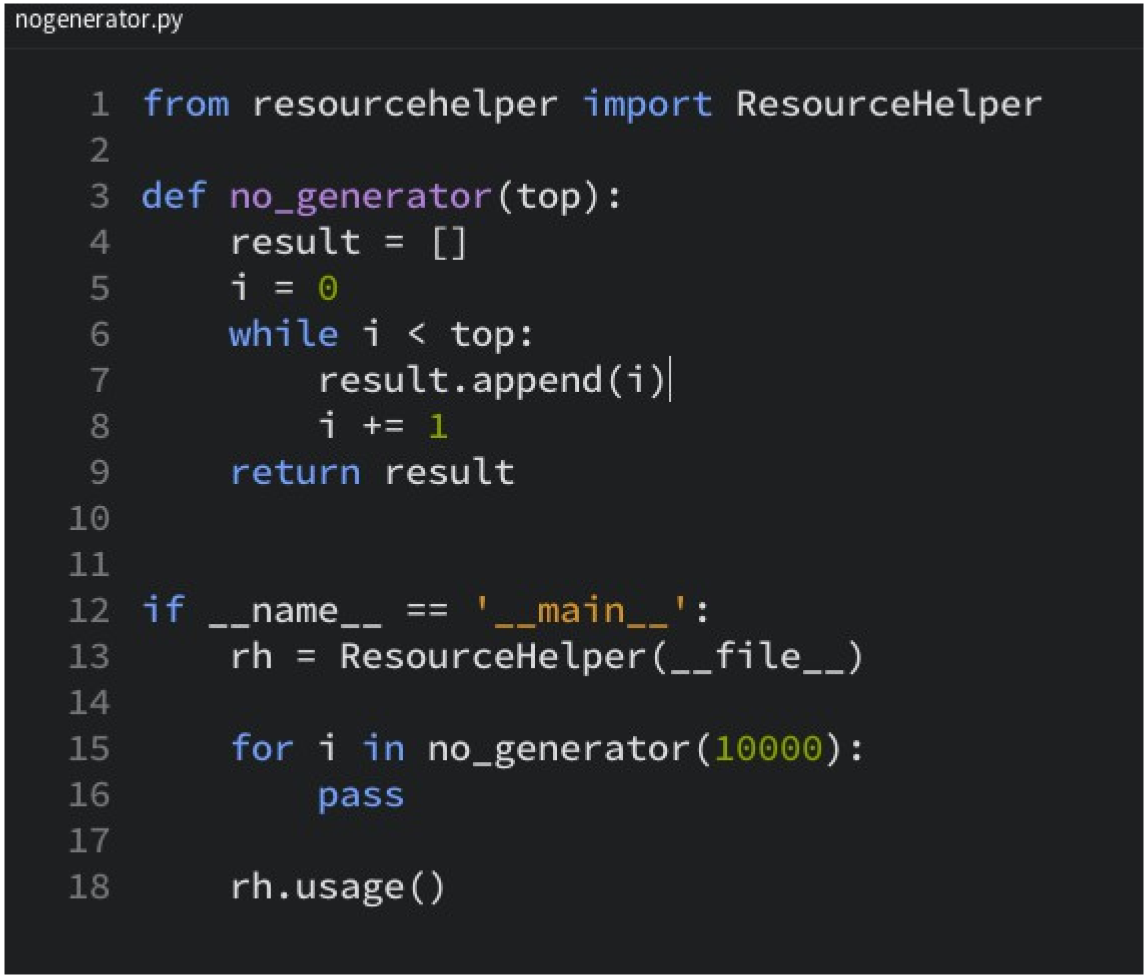
- I need a list of numbers

  0 to 9999

- I could create a list of numbers in memory
- But its not very efficient



```
module: nogenerator.py
mem:    516,096
utime:  0.004304
stime:  0.000294
cpu:    0.004598
time:   0.004591
```

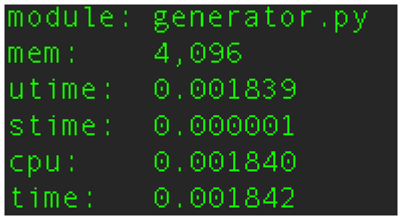


nogenerator.py

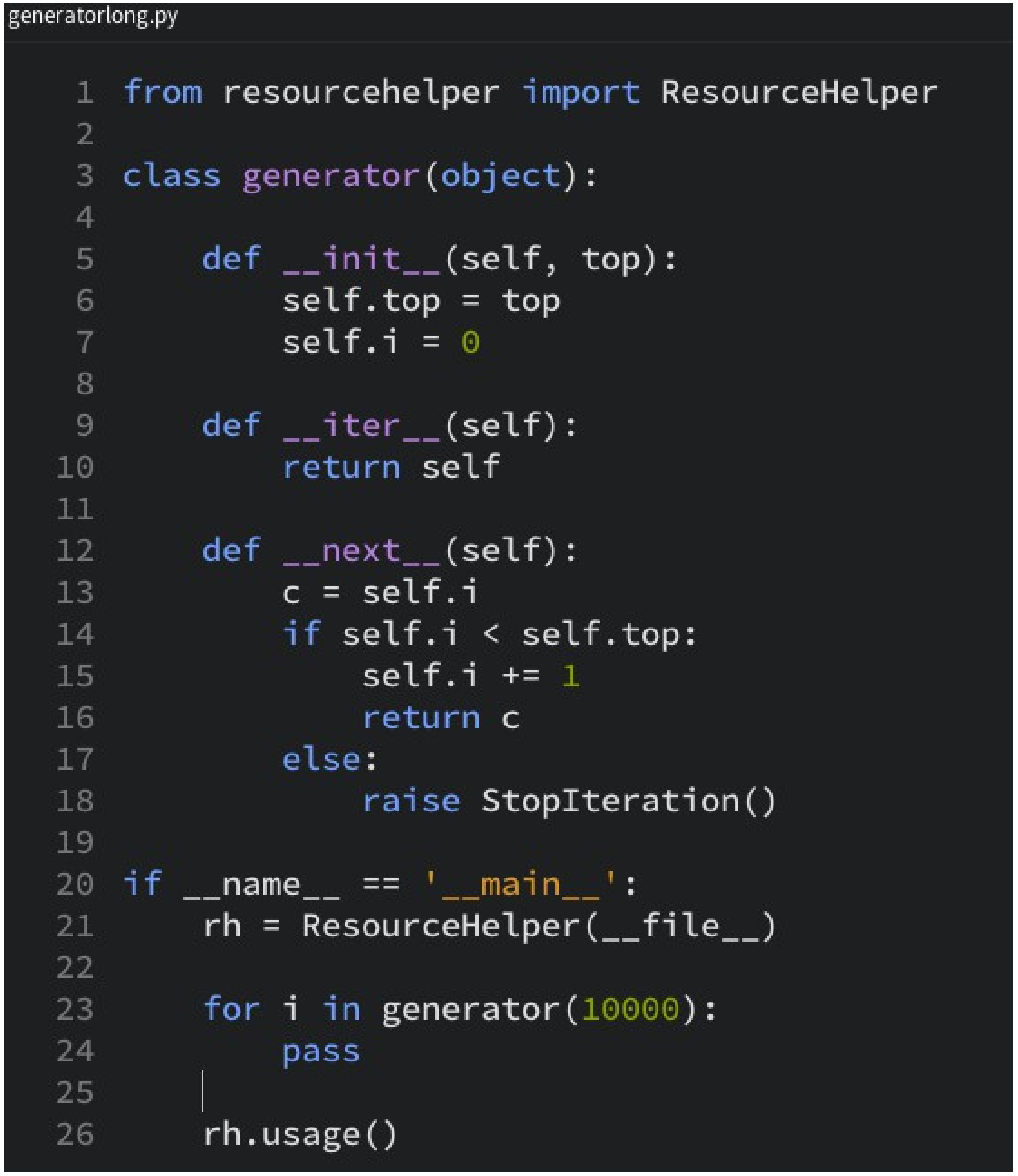
```python
from resourcehelper import ResourceHelper

def no_generator(top):
    result = []
    i = 0
    while i < top:
        result.append(i)
        i += 1
    return result


if __name__ == '__main__':
    rh = ResourceHelper(__file__)

    for i in no_generator(10000):
        pass

    rh.usage()
```

Python generators

- I could also use a generator pattern
- Much more efficient, but lots of code
- Create an iterable object
- Creates data as needed
- Think range(n)



```
module: generator.py
mem:    4,096
utime:  0.001839
stime:  0.000001
cpu:    0.001840
time:   0.001842
```

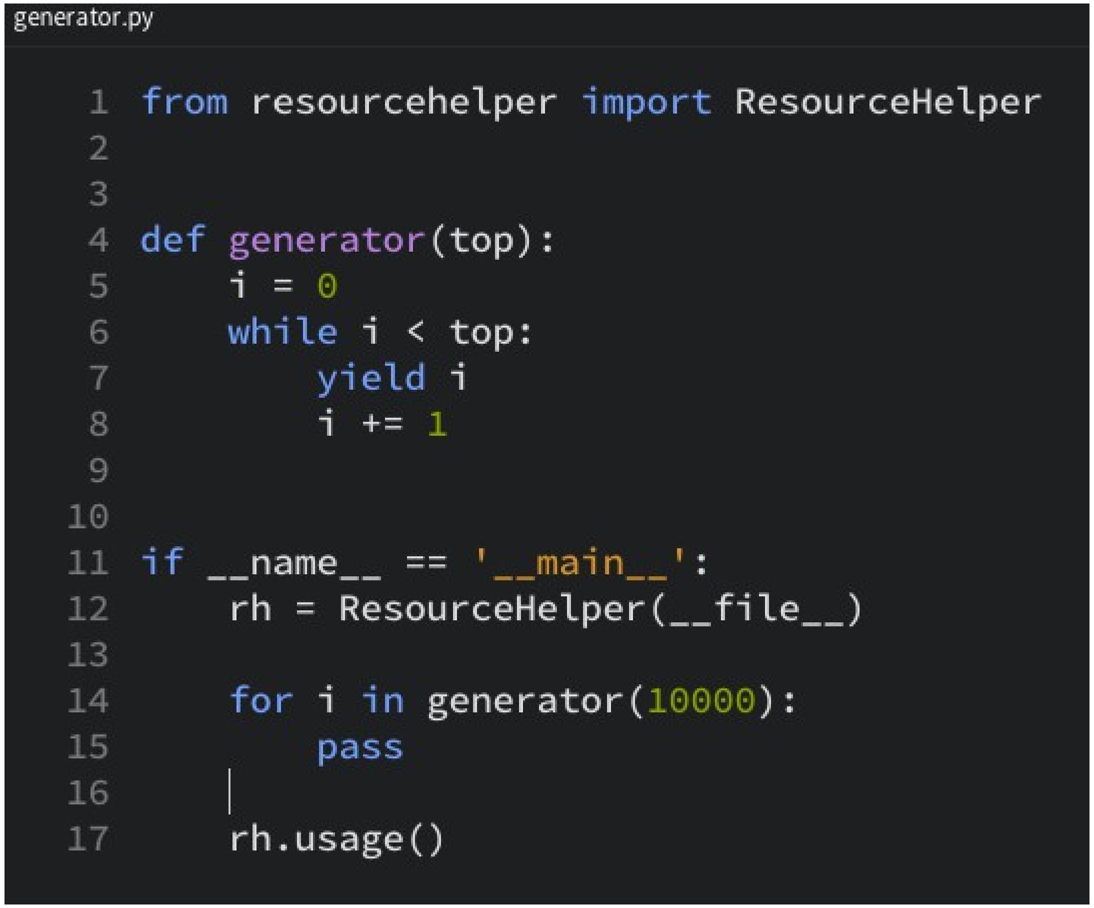


generatorlong.py

```python
from resourcehelper import ResourceHelper

class generator(object):

    def __init__(self, top):
        self.top = top
        self.i = 0

    def __iter__(self):
        return self

    def __next__(self):
        c = self.i
        if self.i < self.top:
            self.i += 1
            return c
        else:
            raise StopIteration()

if __name__ == '__main__':
    rh = ResourceHelper(__file__)

    for i in generator(10000):
        pass

    rh.usage()
```

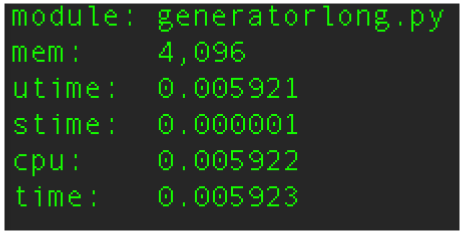
Python generators

- Python provides a nice shortcut to create generators
- yield tells thread to return value and shelve code until next call of generator

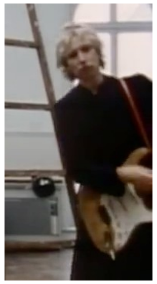


generator.py

```python
from resourcehelper import ResourceHelper


def generator(top):
    i = 0
    while i < top:
        yield i
        i += 1


if __name__ == '__main__':
    rh = ResourceHelper(__file__)

    for i in generator(10000):
        pass

    rh.usage()
```



```
module: generatorlong.py
mem:    4,096
utime:  0.005921
stime:  0.000001
cpu:    0.005922
time:   0.005923
```



asyncio and aiohttp

asyncio

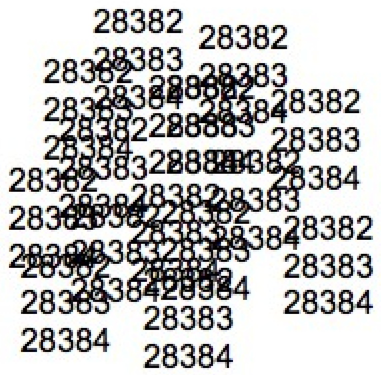
- Introduced in 3.4, derived from tulip project
- Deferred pattern using coroutines, futures and tasks as opposed to callbacks
- Thought is that callbacks can get complicated real quick and hard to read
- Single event loop per thread
- Provides transport library to asynchronous manage connections

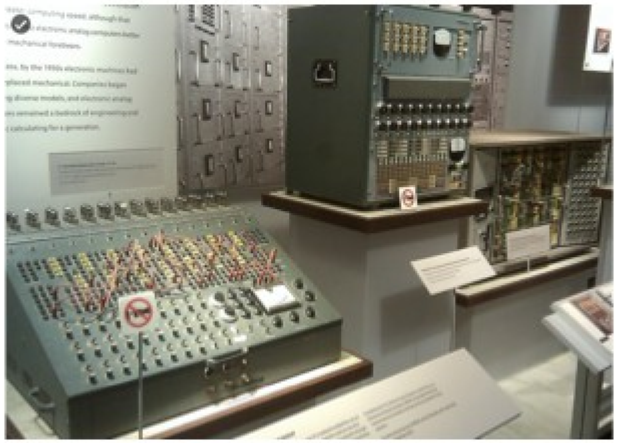
aiohttp

- Third party library that provides http protocol support to asyncio
- Sits on to of asyncio transport library
- Think of it as a requests for asyncio

Problem: total .gov jobs in zips

a bunch of zip codes + usajobs.gov = number of jobs



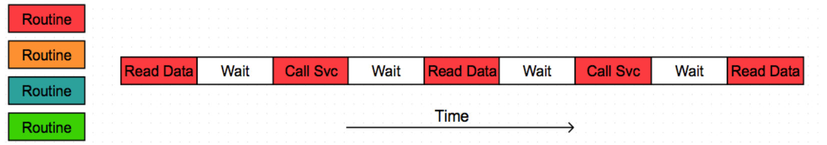


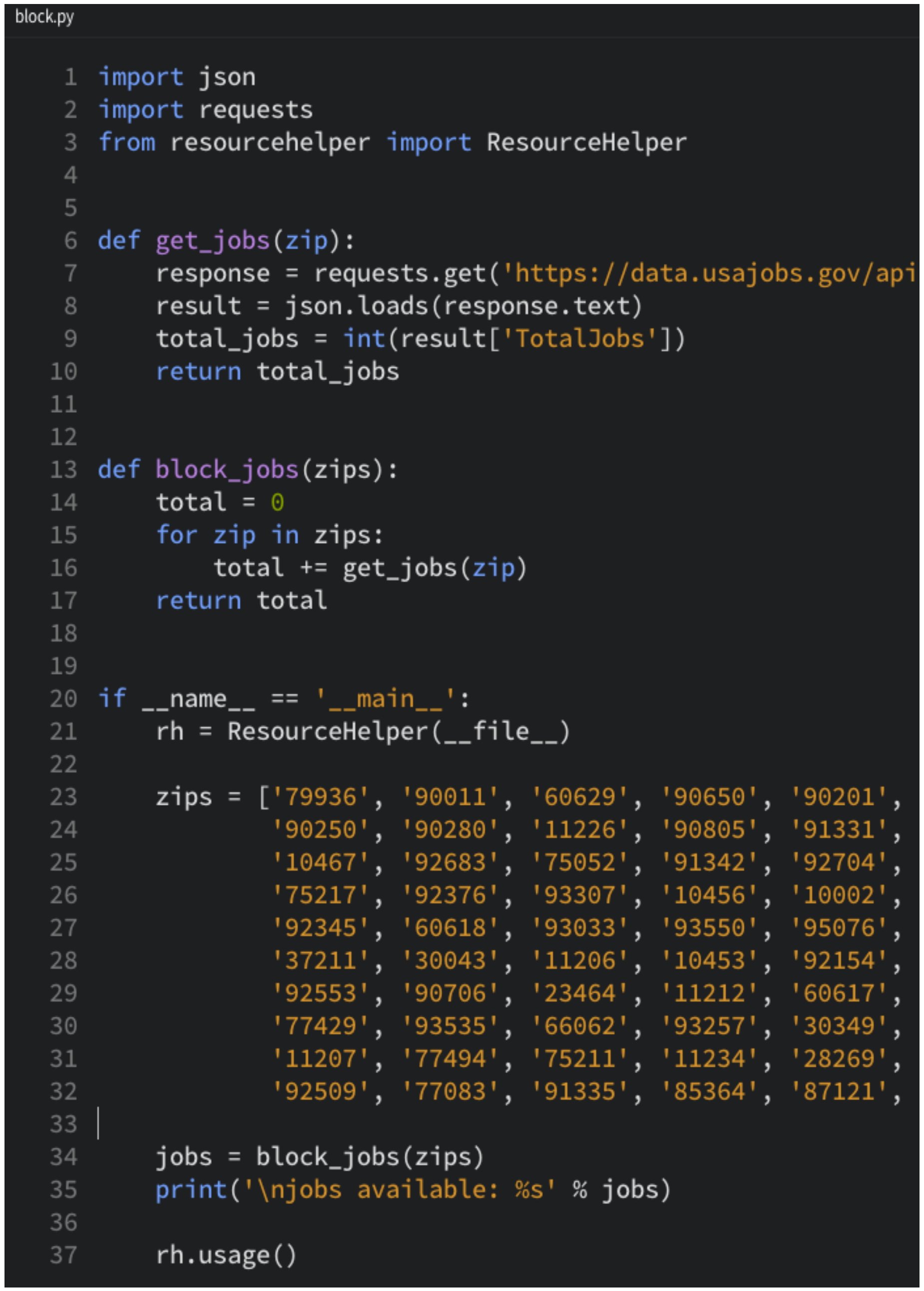
1,706

- total_jobs = 0
- loop through all zips
  - call https://data.usajobs.gov/api/jobs?LocationId=' + zip
  - total_jobs = total_jobs + response.number_of_jobs
- print total_jobs

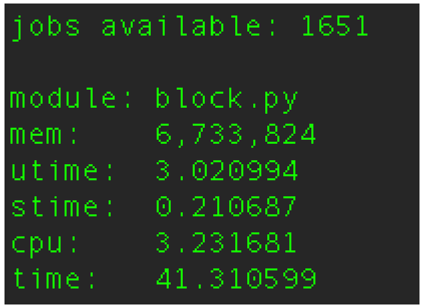
Blocking solution

- Single thread, synchronous execution
- Lots of wasted time waiting for usajobs.gov
- Wait for DNS lookup and for a response from server



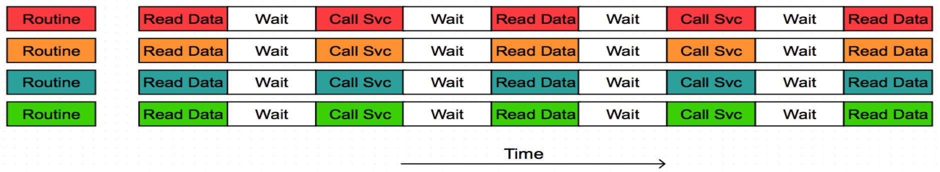

block.py

```python
import json
import requests
from resourcehelper import ResourceHelper


def get_jobs(zip):
    response = requests.get('https://data.usajobs.gov/api
    result = json.loads(response.text)
    total_jobs = int(result['TotalJobs'])
    return total_jobs


def block_jobs(zips):
    total = 0
    for zip in zips:
        total += get_jobs(zip)
    return total


if __name__ == '__main__':
    rh = ResourceHelper(__file__)

    zips = ['79936', '90011', '60629', '90650', '90201',
            '90250', '90280', '11226', '90805', '91331',
            '10467', '92683', '75052', '91342', '92704',
            '75217', '92376', '93307', '10456', '10002',
            '92345', '60618', '93033', '93550', '95076',
            '37211', '30043', '11206', '10453', '92154',
            '92553', '90706', '23464', '11212', '60617',
            '77429', '93535', '66062', '93257', '30349',
            '11207', '77494', '75211', '11234', '28269',
            '92509', '77083', '91335', '85364', '87121',

    jobs = block_jobs(zips)
    print('\njobs available: %s' % jobs)

    rh.usage()
```

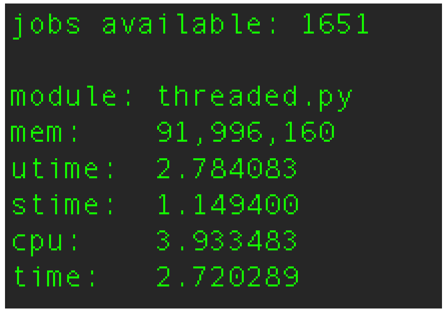


```
jobs available: 1651

module: block.py
mem:    6,733,824
utime:  3.020994
stime:  0.210687
cpu:    3.231681
time:   41.310599
```

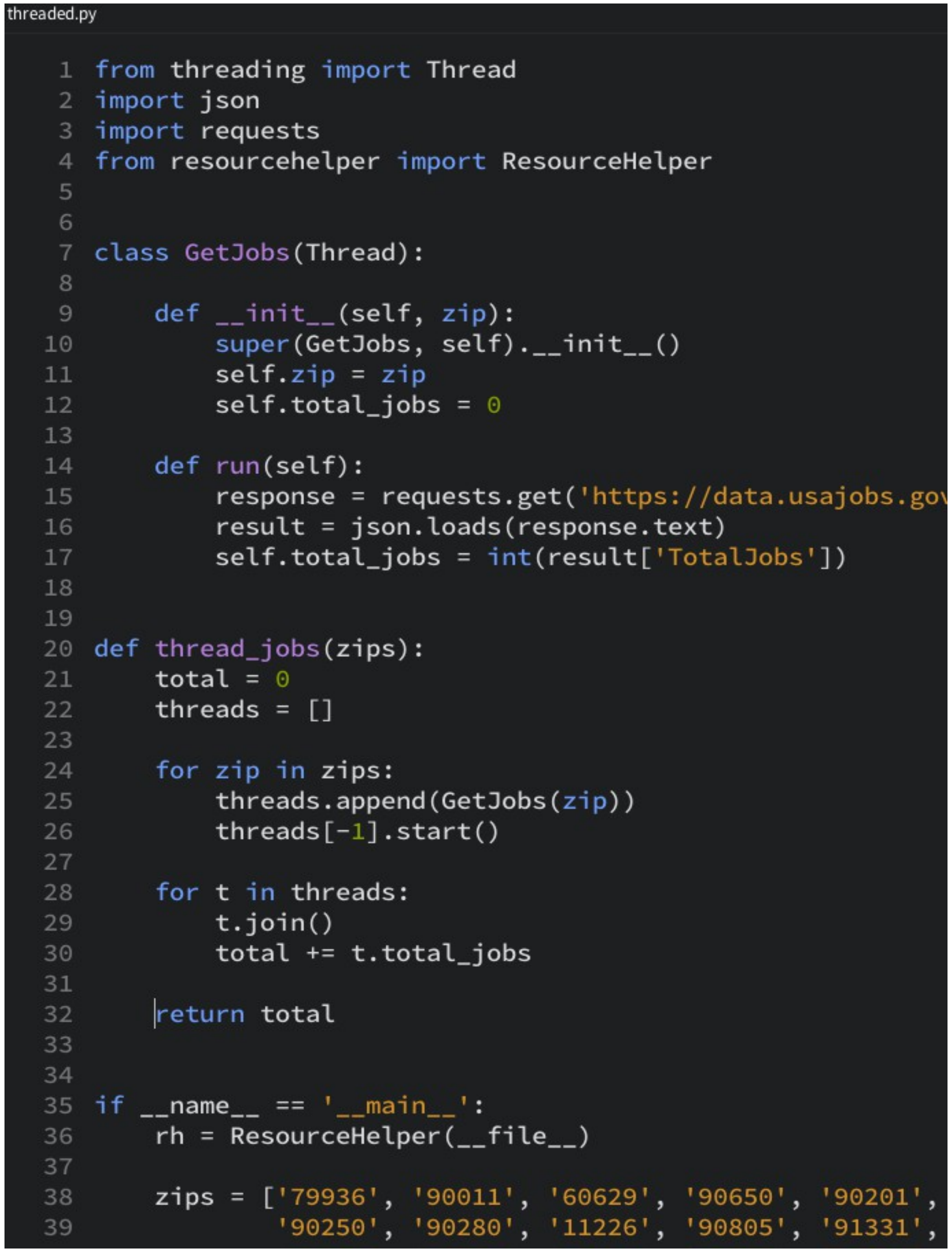
Threaded solution



- Spin a thread for each lookup
- As threads end capture result
- For I/O operations very fast but can use a lot of memory
- For CPU operations GIL will block



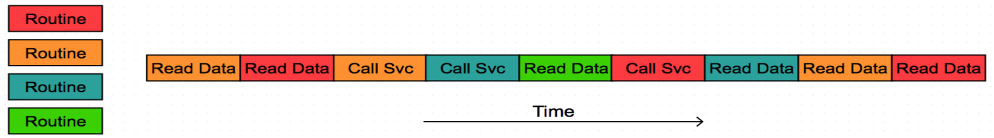
```
jobs available: 1651

module: threaded.py
mem:    91,996,160
utime:  2.784083
stime:  1.149400
cpu:    3.933483
time:   2.720289
```



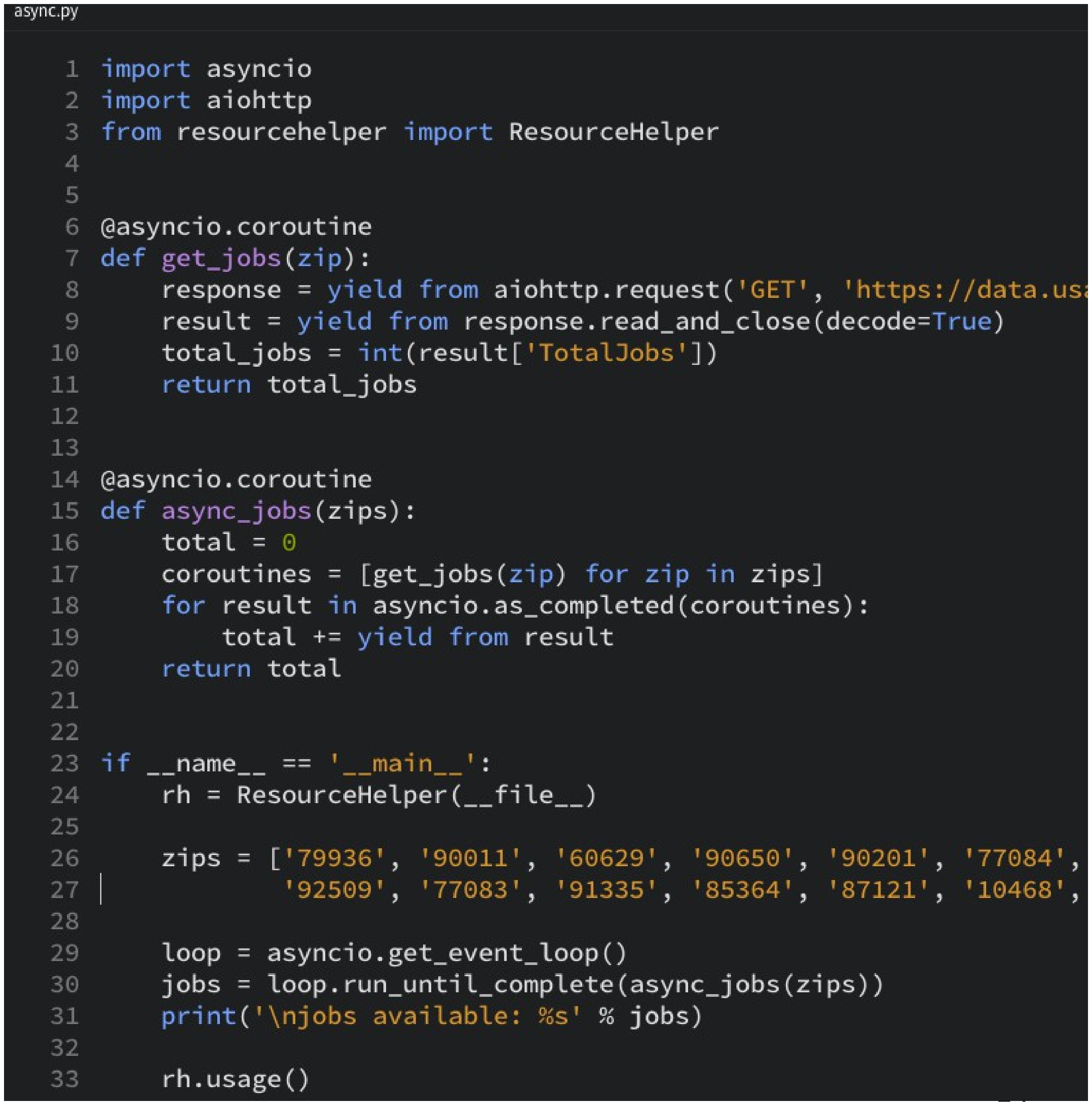
threaded.py

```python
from threading import Thread
import json
import requests
from resourcehelper import ResourceHelper


class GetJobs(Thread):

    def __init__(self, zip):
        super(GetJobs, self).__init__()
        self.zip = zip
        self.total_jobs = 0

    def run(self):
        response = requests.get('https://data.usajobs.gov
        result = json.loads(response.text)
        self.total_jobs = int(result['TotalJobs'])


def thread_jobs(zips):
    total = 0
    threads = []

    for zip in zips:
        threads.append(GetJobs(zip))
        threads[-1].start()

    for t in threads:
        t.join()
        total += t.total_jobs

    return total


if __name__ == '__main__':
    rh = ResourceHelper(__file__)

    zips = ['79936', '90011', '60629', '90650', '90201',
            '90250', '90280', '11226', '90805', '91331',
```

Async solution

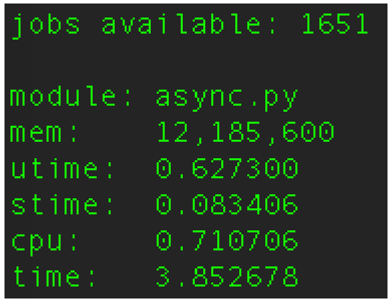


- Event loop is created and passed a coroutine
- coroutine is a generator that wraps a function, it makes it event loop aware
- coroutines return futures
- A future represents a future result, think deferred
- yield from used to handle future of coroutine
- yield from tells the loop to shelve a coroutine until its ready
- coroutines may be stacked on top of one another, results can bubble up

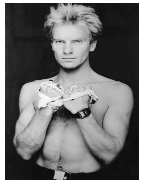


async.py

```python
import asyncio
import aiohttp
from resourcehelper import ResourceHelper


@asyncio.coroutine
def get_jobs(zip):
    response = yield from aiohttp.request('GET', 'https://data.usa
    result = yield from response.read_and_close(decode=True)
    total_jobs = int(result['TotalJobs'])
    return total_jobs


@asyncio.coroutine
def async_jobs(zips):
    total = 0
    coroutines = [get_jobs(zip) for zip in zips]
    for result in asyncio.as_completed(coroutines):
        total += yield from result
    return total


if __name__ == '__main__':
    rh = ResourceHelper(__file__)

    zips = ['79936', '90011', '60629', '90650', '90201', '77084',
            '92509', '77083', '91335', '85364', '87121', '10468',

    loop = asyncio.get_event_loop()
    jobs = loop.run_until_complete(async_jobs(zips))
    print('\njobs available: %s' % jobs)

    rh.usage()
```

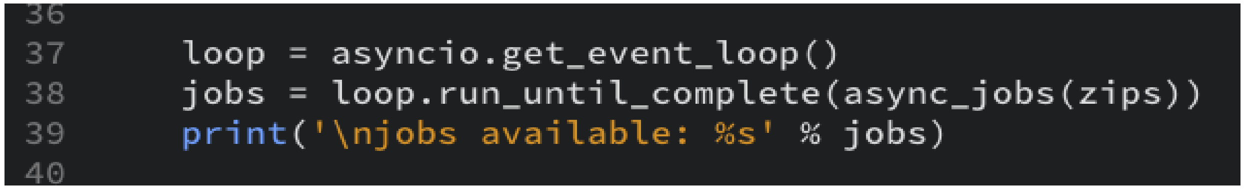


```
jobs available: 1651

module: async.py
mem:    12,185,600
utime:  0.627300
stime:  0.083406
cpu:    0.710706
time:   3.852678
```

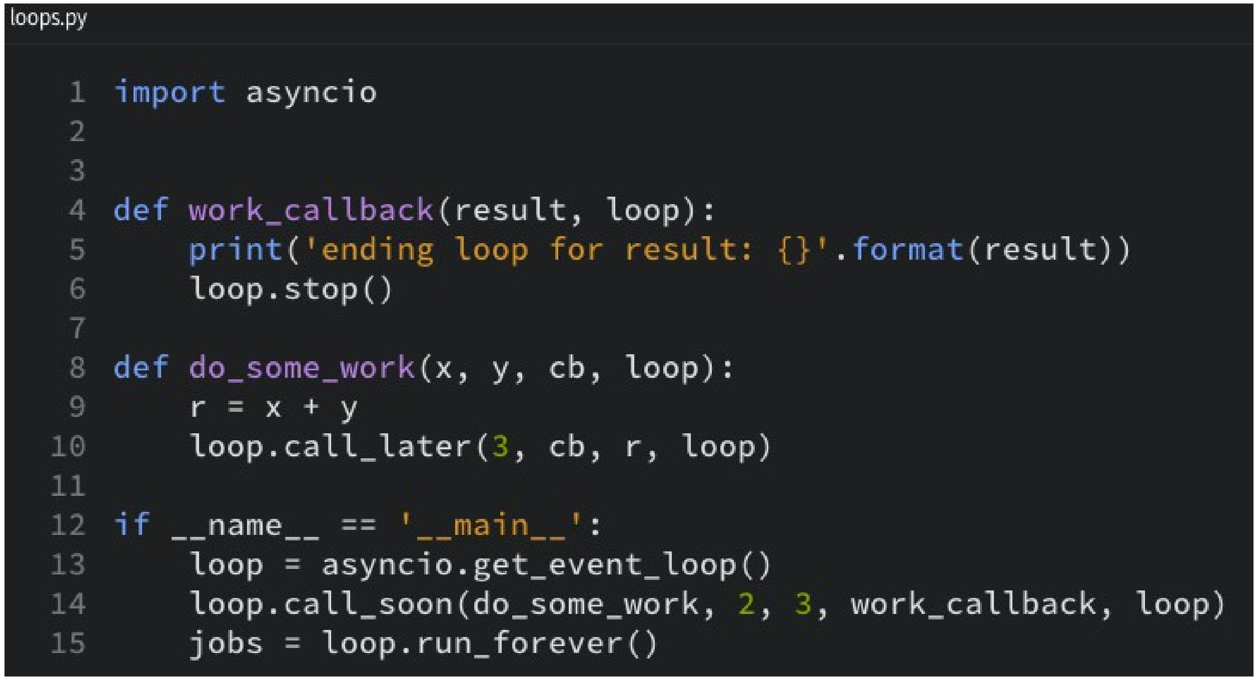


Event Loop

- Is a loop that facilitates the sharing of computing resources by coroutines
- One loop per thread, so typically a single loop
- May be created and retrieved by .get_event_loop(), typically in main()
- Only the main thread may do this, other threads must create their own and use .set_event_loop() before the get
- Ex. .run_until_complete(future) to start our loop OR .run_forever() + .stop()
- Callbacks within loop using .call_soon(cb, args), .call_later(), .call_at()



```python

    loop = asyncio.get_event_loop()
    jobs = loop.run_until_complete(async_jobs(zips))
    print('\njobs available: %s' % jobs)

```

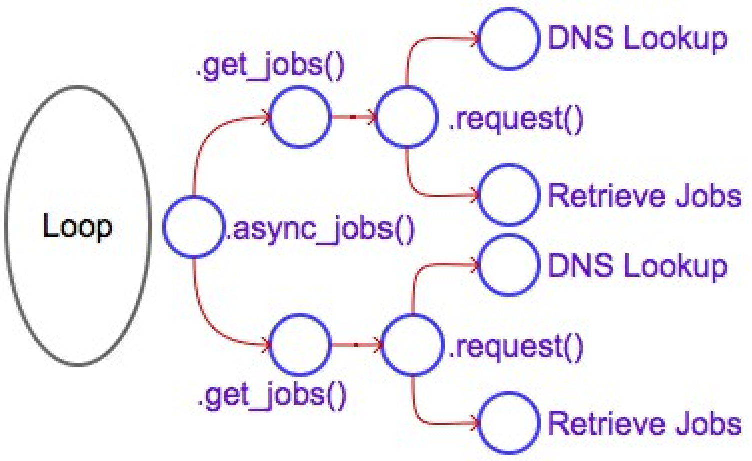


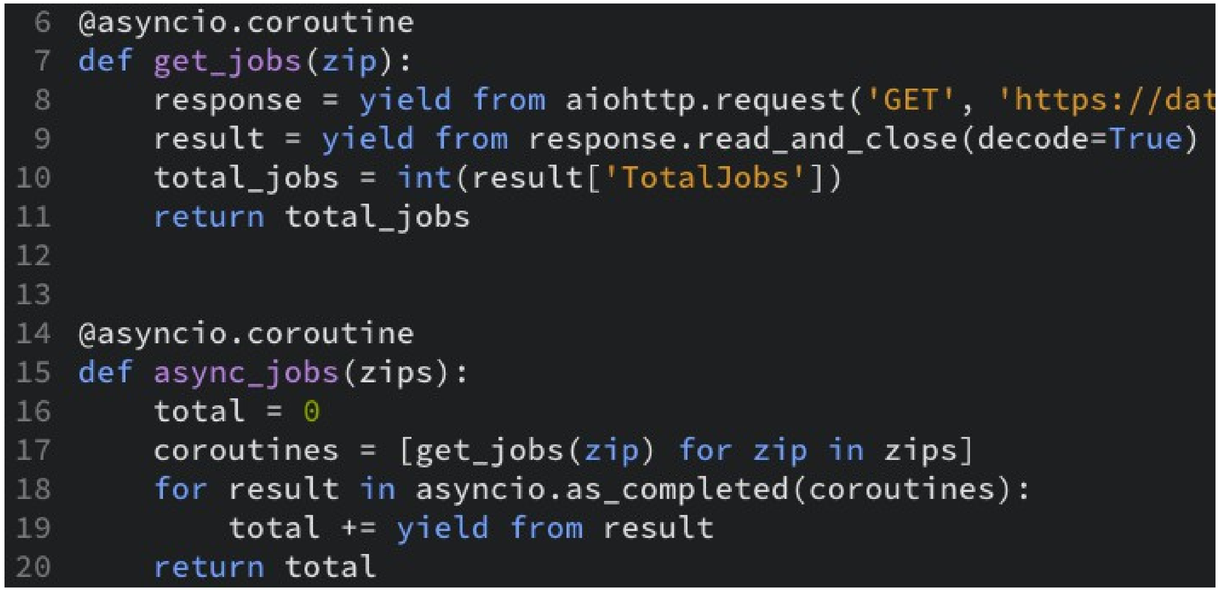
loops.py

```python
import asyncio


def work_callback(result, loop):
    print('ending loop for result: {}'.format(result))
    loop.stop()

def do_some_work(x, y, cb, loop):
    r = x + y
    loop.call_later(3, cb, r, loop)

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.call_soon(do_some_work, 2, 3, work_callback, loop)
    jobs = loop.run_forever()
```

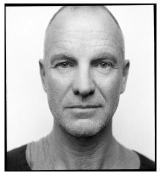
Coroutine

- Is a generator that wraps a function, made to be event loop aware
- As a generator they return a computed value
- A future represents a future result, think deferred
- Coroutines return futures, either not done or done with result
- If done code block continues, if not done continue to shelve coroutine
- yield from identifies a coroutine
- yield from used to handle future of called coroutine
- yield from tells the loop to shelve a coroutine until its ready
- “pretend that yield from isn't there”

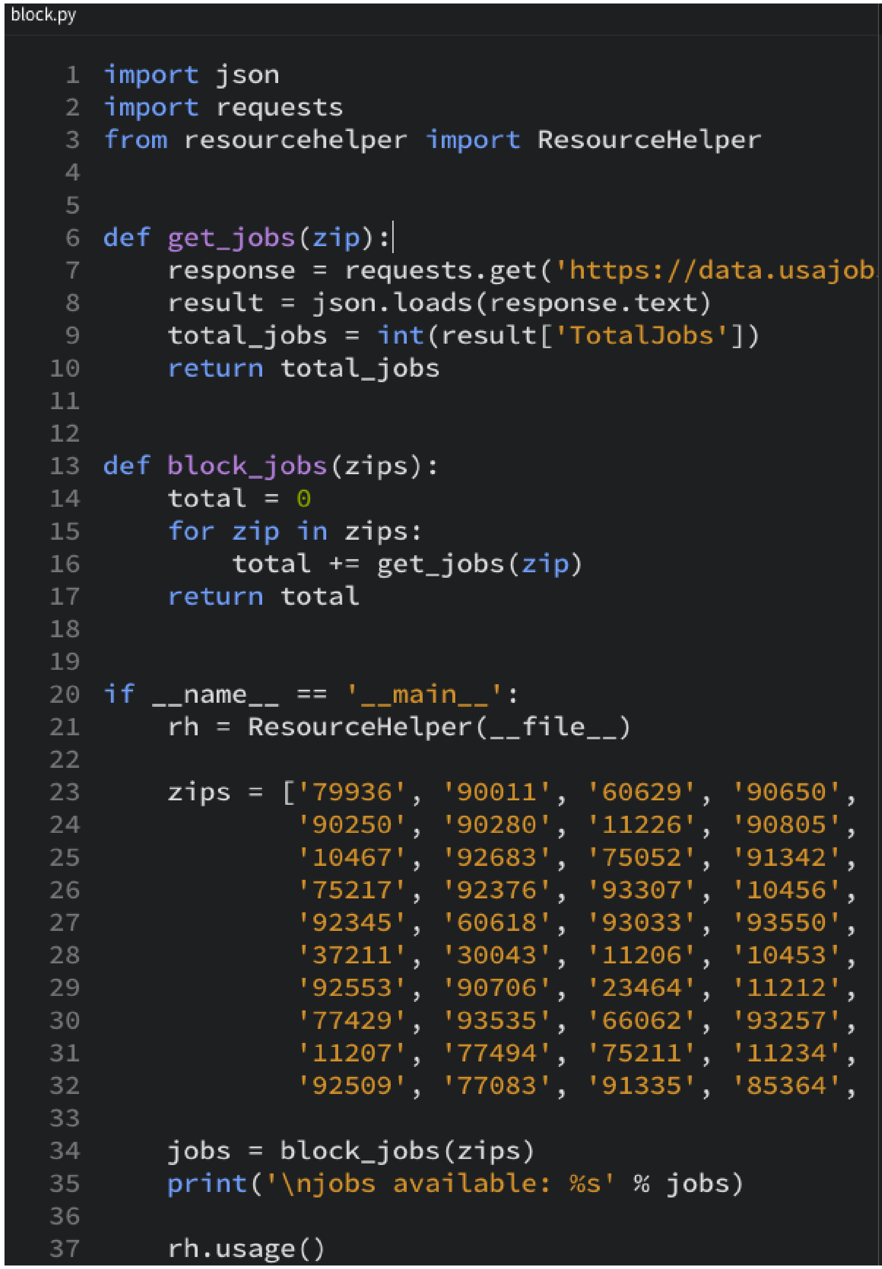




```
@asyncio.coroutine
def get_jobs(zip):
    response = yield from aiohttp.request('GET', 'https://dat
    result = yield from response.read_and_close(decode=True)
    total_jobs = int(result['TotalJobs'])
    return total_jobs


@asyncio.coroutine
def async_jobs(zips):
    total = 0
    coroutines = [get_jobs(zip) for zip in zips]
    for result in asyncio.as_completed(coroutines):
        total += yield from result
    return total
```

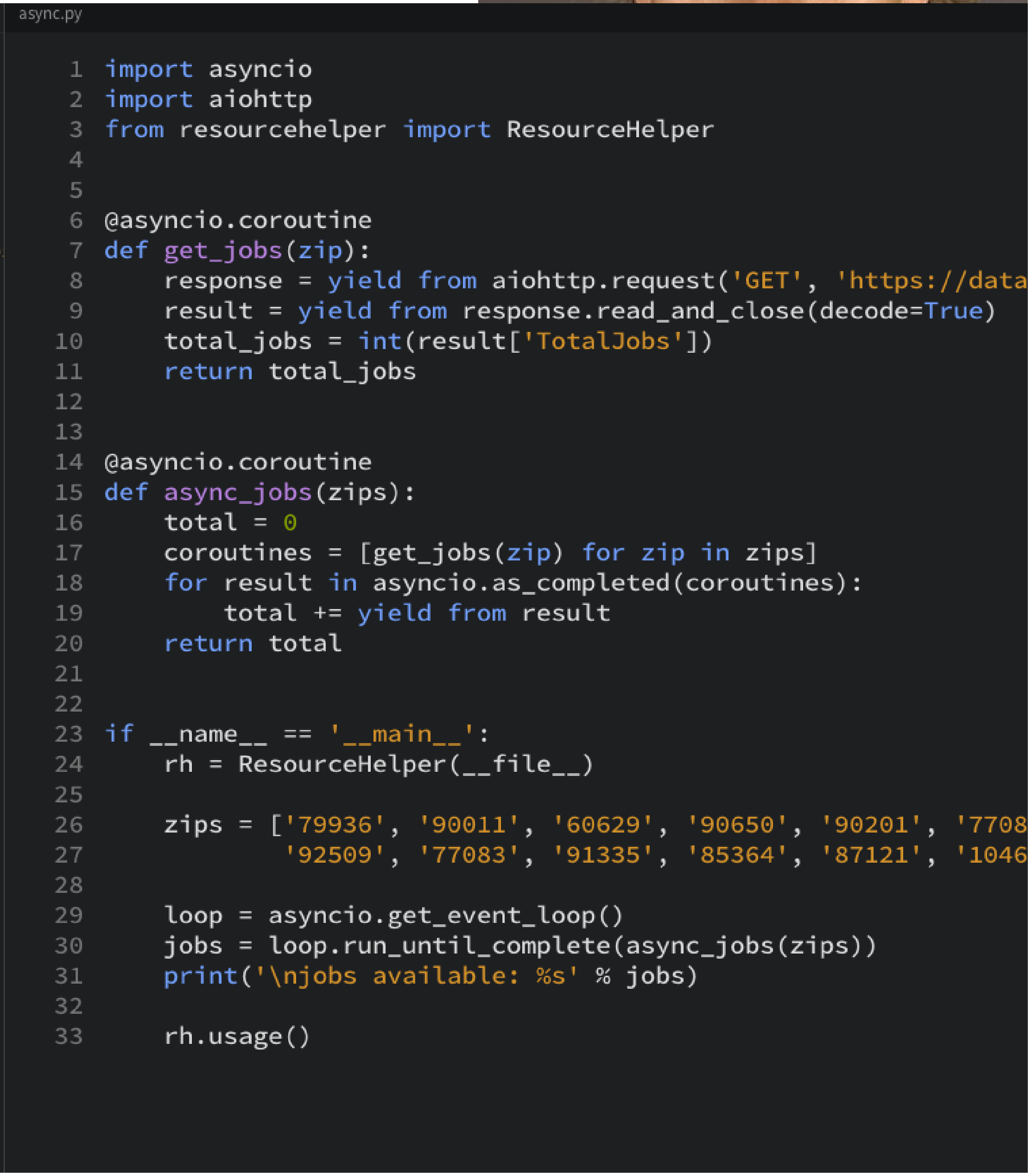


Blocking vs Async



block.py

```
import json
import requests
from resourcehelper import ResourceHelper


def get_jobs(zip):
    response = requests.get('https://data.usajob
    result = json.loads(response.text)
    total_jobs = int(result['TotalJobs'])
    return total_jobs


def block_jobs(zips):
    total = 0
    for zip in zips:
        total += get_jobs(zip)
    return total


if __name__ == '__main__':
    rh = ResourceHelper(__file__)

    zips = ['79936', '90011', '60629', '90650',
            '90250', '90280', '11226', '90805',
            '10467', '92683', '75052', '91342',
            '75217', '92376', '93307', '10456',
            '92345', '60618', '93033', '93550',
            '37211', '30043', '11206', '10453',
            '92553', '90706', '23464', '11212',
            '77429', '93535', '66062', '93257',
            '11207', '77494', '75211', '11234',
            '92509', '77083', '91335', '85364',

    jobs = block_jobs(zips)
    print('\njobs available: %s' % jobs)

    rh.usage()
```



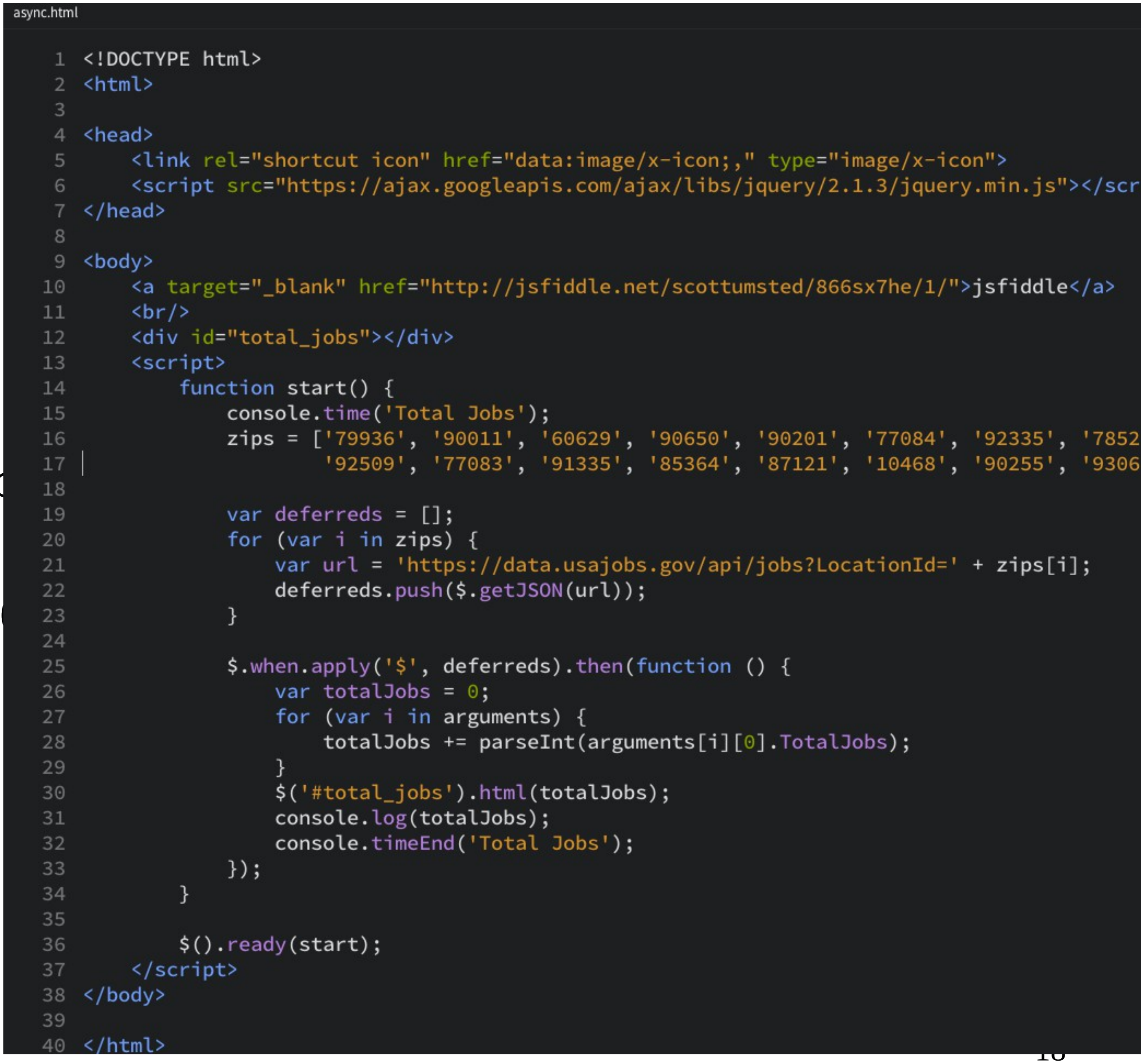
async.py

```
import asyncio
import aiohttp
from resourcehelper import ResourceHelper


@asyncio.coroutine
def get_jobs(zip):
    response = yield from aiohttp.request('GET', 'https://data
    result = yield from response.read_and_close(decode=True)
    total_jobs = int(result['TotalJobs'])
    return total_jobs


@asyncio.coroutine
def async_jobs(zips):
    total = 0
    coroutines = [get_jobs(zip) for zip in zips]
    for result in asyncio.as_completed(coroutines):
        total += yield from result
    return total


if __name__ == '__main__':
    rh = ResourceHelper(__file__)

    zips = ['79936', '90011', '60629', '90650', '90201', '7708
            '92509', '77083', '91335', '85364', '87121', '1046

    loop = asyncio.get_event_loop()
    jobs = loop.run_until_complete(async_jobs(zips))
    print('\njobs available: %s' % jobs)

    rh.usage()
```

JavaScript

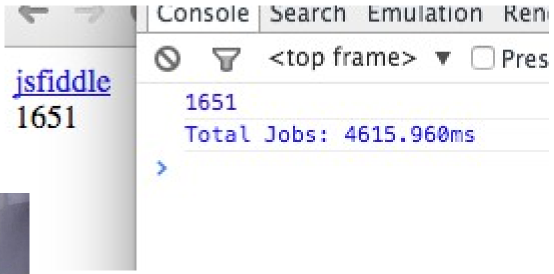
this.element.addEventListener('click',function(){alert('false');});

- JavaScript is single threaded
- I/O operations are asynchronous
- Callbacks used to identify follow up code execution
- Easy to end up with multip nested callbacks
- JQuery deferreds and ES promises



async.html

```
<!DOCTYPE html>
<html>

<head>
    <link rel="shortcut icon" href="data:image/x-icon;," type="image/x-icon">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></scr
</head>

<body>
    <a target="_blank" href="http://jsfiddle.net/scottumsted/866sx7he/1/">jsfiddle</a>
    <br/>
    <div id="total_jobs"></div>
    <script>
        function start() {
            console.time('Total Jobs');
            zips = ['79936', '90011', '60629', '90650', '90201', '77084', '92335', '7852
                '92509', '77083', '91335', '85364', '87121', '10468', '90255', '9306

            var deferreds = [];
            for (var i in zips) {
                var url = 'https://data.usajobs.gov/api/jobs?LocationId=' + zips[i];
                deferreds.push($.getJSON(url));
            }

            $.when.apply('$', deferreds).then(function () {
                var totalJobs = 0;
                for (var i in arguments) {
                    totalJobs += parseInt(arguments[i][0].TotalJobs);
                }
                $('#total_jobs').html(totalJobs);
                console.log(totalJobs);
                console.timeEnd('Total Jobs');
            });
        }

        $().ready(start);
    </script>
</body>

</html>
```



jsfiddle
1651

```
1651
Total Jobs: 4615.960ms
>
```

Other Resources

- Guido –https://youtu.be/1coLC-MUCJc
- https://docs.python.org/3.4/library/asyncio.html
- https://www.python.org/dev/peps/pep-3156/
- http://www.drdobbs.com/open-source/the-new-asyncio-module-in-p
  thon-34-even/240168401
- https://wiki.python.org/moin/Generators