Abstracting Concurrency

Tim Ojo
@tim_ojo www.timojo.com

Abstracting Concurrency Mechanisms

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

t = threading.Thread(target=sayhello)
t.start()
t.join()
```

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

t = threading.Thread(target=sayhello)
t.start()
t.join()
```

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

t = threading.Thread(target=sayhello)
t.start()
t.join()
```

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

t = threading.Thread(target=sayhello)
t.start()
t.join()
```

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

t = threading.Thread(target=sayhello)
t.start()
t.join()
```

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

num_threads = 4
threads = []
for _ in range(num_threads):
    t = threading.Thread(target=sayhello)
    t.start()
    threads.append(t)

[t.join() for t in threads]
```

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

num_threads = 4
threads = []
for _ in range(num_threads):
    t = threading.Thread(target=sayhello)
    t.start()
    threads.append(t)

[t.join() for t in threads]
```

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

num_threads = 4
threads = []
for _ in range(num_threads):
    t = threading.Thread(target=sayhello)
    t.start()
    threads.append(t)

[t.join() for t in threads]
```

```
import multiprocessing, os

def sayhello():
    print("hello from {}".format(os.getpid()))
    return

if __name__ == '__main__':
    p = multiprocessing.Process(target=sayhello)
    p.start()
    p.join()
```

```
import multiprocessing, os

def sayhello():
    print("hello from {}".format(os.getpid()))
    return

if __name__ == '__main__':
    p = multiprocessing.Process(target=sayhello)
    p.start()
    p.join()
```

```
import multiprocessing, os

def sayhello():
    print("hello from {}".format(os.getpid()))
    return

num_processes = 4
processes = []
if __name__ == '__main__':
    for _ in range(num_processes):
        p = multiprocessing.Process(target=sayhello)
        p.start()
        processes.append(p)

    [p.join() for p in processes]
```

```
import multiprocessing, os

def sayhello():
    print("hello from {}".format(os.getpid()))
    return

num_processes = 4
processes = []
if __name__ == '__main__':
    for _ in range(num_processes):
        p = multiprocessing.Process(target=sayhello)
        p.start()
        processes.append(p)

    [p.join() for p in processes]
```

```
import multiprocessing, os

def sayhello():
    print("hello from {}".format(os.getpid()))
    return

num_processes = 4
processes = []
if __name__ == '__main__':
    for _ in range(num_processes):
        p = multiprocessing.Process(target=sayhello)
        p.start()
        processes.append(p)

    [p.join() for p in processes]
```

- Define Task

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

t = threading.Thread(target=sayhello)
t.start()
t.join()
```

```
import multiprocessing, os

def sayhello():
    print("hello from {}".format(os.getpid()))
    return

if __name__ == '__main__':
    p = multiprocessing.Process(target=sayhello)
    p.start()
    p.join()
```

- Define Task
- **Pass to Executor**

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

t = threading.Thread(target=sayhello)
t.start()
t.join()
```

```
import multiprocessing, os

def sayhello():
    print("hello from {}".format(os.getpid()))
    return

if __name__ == '__main__':
    p = multiprocessing.Process(target=sayhello)
    p.start()
    p.join()
```

- Define Task
- Pass to Executor
- **Get Results**

```
import threading

def sayhello():
    print("hello from {}".format(threading.current_thread().name))
    return

t = threading.Thread(target=sayhello)
t.start()
t.join()
```

```
import multiprocessing, os

def sayhello():
    print("hello from {}".format(os.getpid()))
    return

if __name__ == '__main__':
    p = multiprocessing.Process(target=sayhello)
    p.start()
    p.join()
```

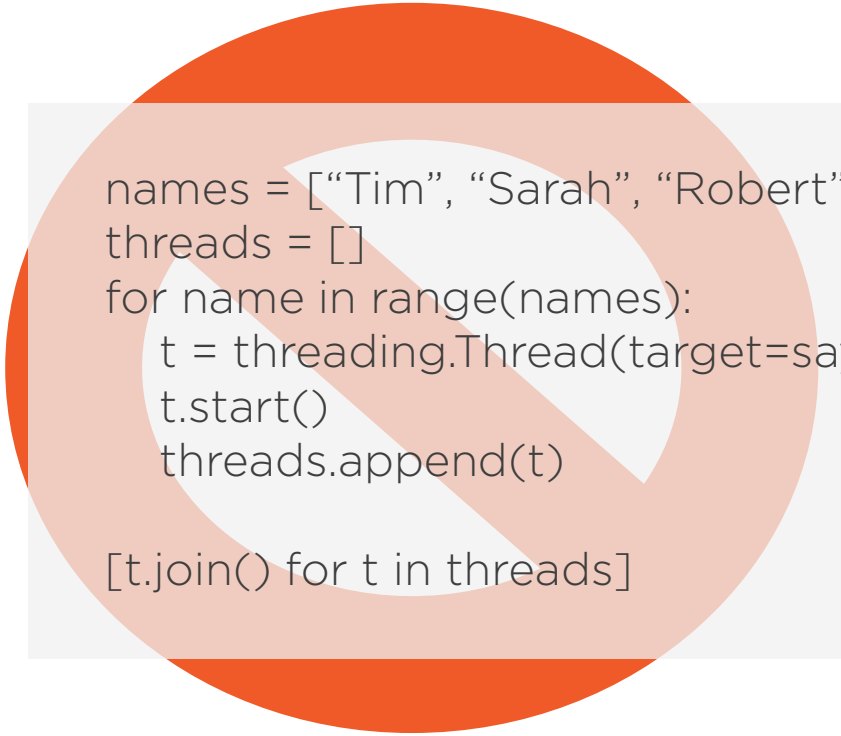
```
def sayhello(name):
    print(”hello from {}“.format(name))
    return

executor = # new threadpool or processpool instance
future_result = executor.submit(sayhello, “Tim”)
```

```
def sayhello(name):
    print(”hello from {}“.format(name))
    return

executor = # new threadpool or processpool instance
names = [“Tim”, “Sarah”, “Robert”]
results = executor.map(sayhello, names)
```

```
executor = # new threadpool or processpool instance
names = [“Tim”, “Sarah”, “Robert”]
results = executor.map(sayhello, names)
```



```
names = [“Tim”, “Sarah”, “Robert”]
threads = []
for name in range(names):
    t = threading.Thread(target=sayhello, arg=(name,))
    t.start()
    threads.append(t)

[t.join() for t in threads]
```

```
def sayhello(name):
    print(”hello from {}“.format(name))
    return

executor = #
names = [“Tim”, “Sarah”, “Robert”]
results = executor.map(sayhello, names)
```

```
def sayhello(name):
    print(”hello from {}“.format(name))
    return

executor = # new threadpool instance
names = [“Tim”, “Sarah”, “Robert”]
results = executor.map(sayhello, names)
```

```
def sayhello(name):
    print("hello from {}".format(name))
    return

executor = # new processpool instance
names = ["Tim", "Sarah", "Robert"]
results = executor.map(sayhello, names)
```

Reasons to Switch Executors?

Reasons to Switch Executors?

- Tasks mix IO and CPU

Reasons to Switch Executors?

- Tasks mix IO and CPU
- Change of platforms

The Executor API

Executor Methods

```
submit(fn, *args, **kwargs) # schedules a function to run
```

Executor Methods

```
map(func, *iterables, timeout=None, chunksize=1)
```

Executor Methods

```
map(func,
        *iterables,
        timeout=None, # how long to wait for a task to complete
        chunksize=1)
```

Executor Methods

```
map(func,
        *iterables,
        timeout=None,
        chunksize=1) # how chop up the iterable per worker
```

Executor Methods

```
shutdown(wait=True) # stop accepting tasks and shutdown
```

Executor Methods

```
shutdown(wait=True) # stop accepting tasks and shutdown
```

Executor Methods

```
with executor:
    ...use executor...
    # no need to call shutdown
```

Executor Subclass

Executor Subclass

- ThreadPoolExecutor

Executor Subclass

- ThreadPoolExecutor
- ProcessPoolExecutor

Executor Subclass

```
ThreadPoolExecutor(max_workers=None, thread_name_prefix=”)
```

Executor Subclass

```
ThreadPoolExecutor(max_workers=None, thread_name_prefix=”)
```

Executor Subclass

```
ThreadPoolExecutor(max_workers=None, thread_name_prefix=”)
```

```
from concurrent.futures import ThreadPoolExecutor
from urllib.request import urlopen

def load_url(url, timeout):
    with urlopen(url, timeout=timeout) as conn:
        return conn.read()

with ThreadPoolExecutor(max_workers=2) as executor:
    url1 = "http://www.cnn.com/"
    url2 = "http://www.some-made-up-domain.com/"
    f1 = executor.submit(load_url, url1, 60)
    f2 = executor.submit(load_url, url2, 60)

    try:
        data1 = f1.result()
        print("{} page is {} bytes".format(url1, len(data1)))
        data2 = f2.result()
        print("{} page is {} bytes".format(url2, len(data2)))
    except Exception as ex:
        print("Exception downloading page " + str(ex))
```

```
from concurrent.futures import ThreadPoolExecutor
from urllib.request import urlopen

def load_url(url, timeout):
    with urlopen(url, timeout=timeout) as conn:
        return conn.read()

with ThreadPoolExecutor(max_workers=2) as executor:
    url1 = "http://www.cnn.com/"
    url2 = "http://www.some-made-up-domain.com/"
    f1 = executor.submit(load_url, url1, 60)
    f2 = executor.submit(load_url, url2, 60)

    try:
        data1 = f1.result()
        print("{} page is {} bytes".format(url1, len(data1)))
        data2 = f2.result()
        print("{} page is {} bytes".format(url2, len(data2)))
    except Exception as ex:
        print("Exception downloading page " + str(ex))
```

```
from concurrent.futures import ThreadPoolExecutor
from urllib.request import urlopen

def load_url(url, timeout):
    with urlopen(url, timeout=timeout) as conn:
        return conn.read()

with ThreadPoolExecutor(max_workers=2) as executor:
    url1 = "http://www.cnn.com/"
    url2 = "http://www.some-made-up-domain.com/"
    f1 = executor.submit(load_url, url1, 60)
    f2 = executor.submit(load_url, url2, 60)

    try:
        data1 = f1.result()
        print("{} page is {} bytes".format(url1, len(data1)))
        data2 = f2.result()
        print("{} page is {} bytes".format(url2, len(data2)))
    except Exception as ex:
        print("Exception downloading page " + str(ex))
```

```
from concurrent.futures import ThreadPoolExecutor
from urllib.request import urlopen

def load_url(url, timeout):
    with urlopen(url, timeout=timeout) as conn:
        return conn.read()

with ThreadPoolExecutor(max_workers=2) as executor:
    url1 = "http://www.cnn.com/"
    url2 = "http://www.some-made-up-domain.com/"
    f1 = executor.submit(load_url, url1, 60)
    f2 = executor.submit(load_url, url2, 60)

    try:
        data1 = f1.result()
        print("{} page is {} bytes".format(url1, len(data1)))
        data2 = f2.result()
        print("{} page is {} bytes".format(url2, len(data2)))
    except Exception as ex:
        print("Exception downloading page " + str(ex))
```

```
from concurrent.futures import ThreadPoolExecutor
from urllib.request import urlopen

def load_url(url, timeout):
    with urlopen(url, timeout=timeout) as conn:
        return conn.read()

with ThreadPoolExecutor(max_workers=2) as executor:
    url1 = "http://www.cnn.com/"
    url2 = "http://www.some-made-up-domain.com/"
    f1 = executor.submit(load_url, url1, 60)
    f2 = executor.submit(load_url, url2, 60)

    try:
        data1 = f1.result()
        print("{} page is {} bytes".format(url1, len(data1)))
        data2 = f2.result()
        print("{} page is {} bytes".format(url2, len(data2)))
    except Exception as ex:
        print("Exception downloading page " + str(ex))
```

```
from concurrent.futures import ThreadPoolExecutor
from urllib.request import urlopen

def load_url(url, timeout):
    with urlopen(url, timeout=timeout) as conn:
        return conn.read()

with ThreadPoolExecutor(max_workers=2) as executor:
    url1 = "http://www.cnn.com/"
    url2 = "http://www.some-made-up-domain.com/"
    f1 = executor.submit(load_url, url1, 60)
    f2 = executor.submit(load_url, url2, 60)

    try:
        data1 = f1.result()
        print("{} page is {} bytes".format(url1, len(data1)))
        data2 = f2.result()
        print("{} page is {} bytes".format(url2, len(data2)))
    except Exception as ex:
        print("Exception downloading page " + str(ex))
```

Executor Subclass

```
ProcessPoolExecutor(max_workers=None)
```

Executor Subclass

```
ProcessPoolExecutor(max_workers=None)
```

```
from concurrent.futures import ProcessPoolExecutor
import hashlib

texts = [b"the quick brown fox jumped over the lazy dog",
         b"the big fat panda sat on the hungry snake",
         b"the slick mountain lion ran up the tall tree"]

def generate_hash(text):
   return hashlib.sha384(text).hexdigest()

if __name__ == '__main__':
   with ProcessPoolExecutor() as executor:
      for text, hash_t in zip(texts, executor.map(generate_hash, texts)):
         print('%s hash is: %s' % (text, hash_t))
```

```
from concurrent.futures import ProcessPoolExecutor
import hashlib

texts = [b"the quick brown fox jumped over the lazy dog",
         b"the big fat panda sat on the hungry snake",
         b"the slick mountain lion ran up the tall tree"]

def generate_hash(text):
   return hashlib.sha384(text).hexdigest()

if __name__ == '__main__':
   with ProcessPoolExecutor() as executor:
      for text, hash_t in zip(texts, executor.map(generate_hash, texts)):
         print('%s hash is: %s' % (text, hash_t))
```

```
from concurrent.futures import ProcessPoolExecutor
import hashlib

texts = [b"the quick brown fox jumped over the lazy dog",
         b"the big fat panda sat on the hungry snake",
         b"the slick mountain lion ran up the tall tree"]

def generate_hash(text):
   return hashlib.sha384(text).hexdigest()

if __name__ == '__main__':
   with ProcessPoolExecutor() as executor:
      for text, hash_t in zip(texts, executor.map(generate_hash, texts)):
         print('%s hash is: %s' % (text, hash_t))
```

```
from concurrent.futures import ProcessPoolExecutor
import hashlib

texts = [b"the quick brown fox jumped over the lazy dog",
         b"the big fat panda sat on the hungry snake",
         b"the slick mountain lion ran up the tall tree"]

def generate_hash(text):
    return hashlib.sha384(text).hexdigest()

if __name__ == '__main__':
    with ProcessPoolExecutor() as executor:
        for text, hash_t in zip(texts, executor.map(generate_hash, texts)):
            print('%s hash is: %s' % (text, hash_t))
```

```
from concurrent.futures import ProcessPoolExecutor
import hashlib

texts = [b"the quick brown fox jumped over the lazy dog",
         b"the big fat panda sat on the hungry snake",
         b"the slick mountain lion ran up the tall tree"]

def generate_hash(text):
    return hashlib.sha384(text).hexdigest()

if __name__ == '__main__':
    with ProcessPoolExecutor() as executor:
        for text, hash_t in zip(texts, executor.map(generate_hash, texts)):
            print('%s hash is: %s' % (text, hash_t))
```

```
from concurrent.futures import ProcessPoolExecutor
import hashlib

texts = [b"the quick brown fox jumped over the lazy dog",
         b"the big fat panda sat on the hungry snake",
         b"the slick mountain lion ran up the tall tree"]

def generate_hash(text):
    return hashlib.sha384(text).hexdigest()

if __name__ == '__main__':
    with ProcessPoolExecutor() as executor:
        for text, hash_t in zip(texts, executor.map(generate_hash, texts)):
            print('%s hash is: %s' % (text, hash_t))
```

Future Object

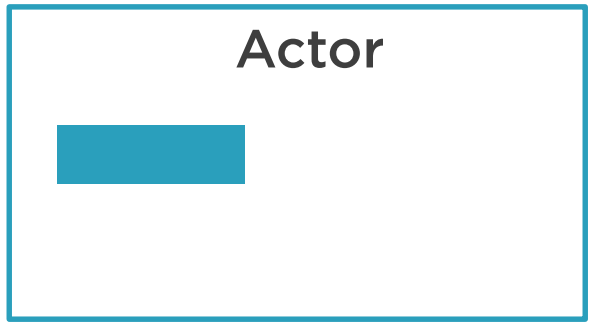
Future

An object that acts as a proxy for a result that is yet to be computed

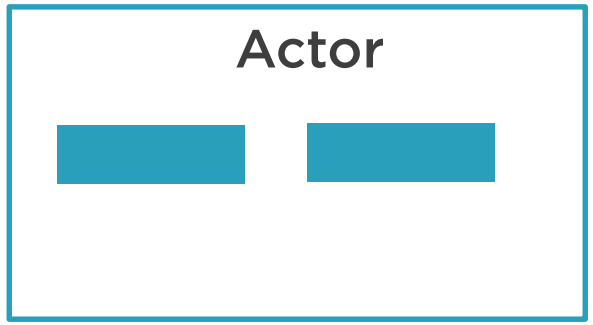
Asynchronous Programming

Asynchronous Programming

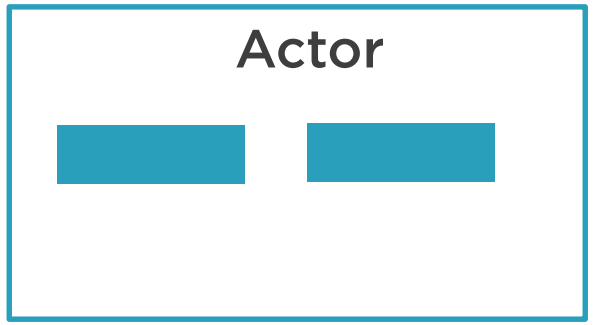
Asynchronous Programming



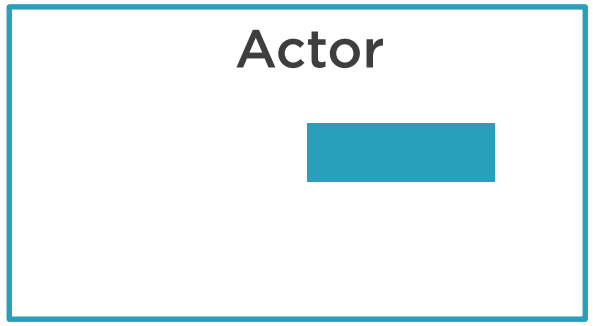
Asynchronous Programming



Asynchronous Programming



Asynchronous Programming



Asynchronous Programming

Asynchronous Programming

```
future = executor.submit(func, args*)
... do other things ...
result = future.result()
```

```
future = executor.submit(func, args*)
... do other things ...
result = future.result()
```

```
future = executor.submit(func, args*)
... do other things ...
result = future.result()
```

```
future = executor.submit(func, args*)
... do other things ...
result = future.result() # can throw exception
```

Future Methods

```
cancel() # attempt to cancel execution. Return True if successful
```

Future Methods

```
done() # returns True if completed or canceled
```

Future Methods

```
exception(timeout=None) # returns the exception raised, if any
```

Future Methods

```
add_done_callback(fn) # attaches function to be called on
completion or cancellation
```

Module Functions

```
concurrent.futures.wait(fs, timeout=None, return_when=ALL_COMPLETED)
```

Module Functions

```
concurrent.futures.wait(fs,
                        timeout=None,
                        return_when=ALL_COMPLETED)
```

Module Functions

```
concurrent.futures.wait(fs,
                        timeout=None,
                        return_when=ALL_COMPLETED)
```

Module Functions

```
concurrent.futures.as_completed(fs, timeout=None)
```

Summary

Abstracting Concurrency Mechanisms

The Executor API

The Future Object