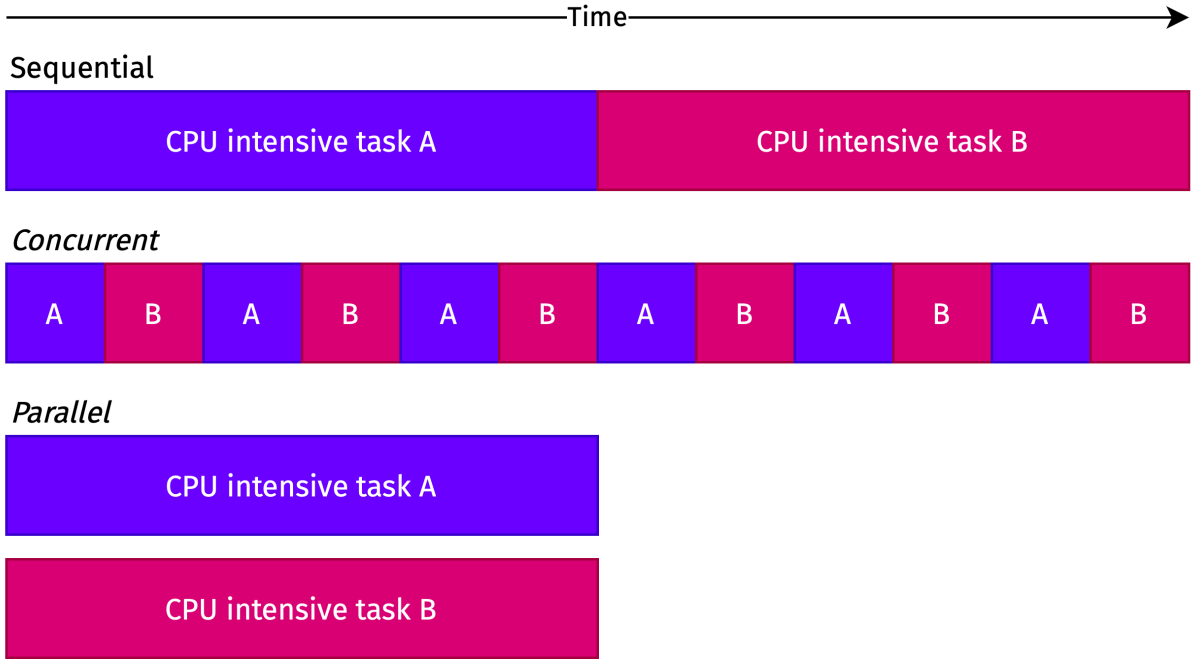
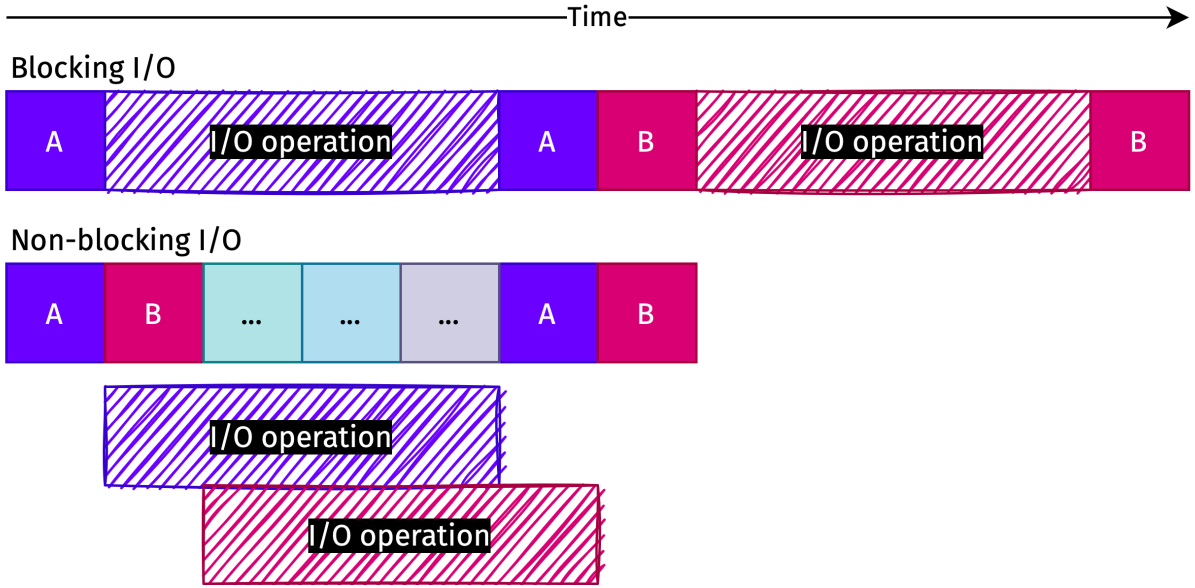
Asynchronous programming in Python

LINCS Python Workshop — May 2021


Time
Sequential
CPU intensive task A
CPU intensive task B
Concurrent
A
B
A
B
A
B
A
B
A
B
A
B
Parallel
CPU intensive task A
CPU intensive task B


Time
Blocking I/O
A
I/O operation
A
B
I/O operation
B
Non-blocking I/O
A
B
...
...
...
A
B
I/O operation
I/O operation

Hello World

In [2]:
```
def hello():
    logging.info("Hello")
    time.sleep(1) # Blocking "I/O" call
    logging.info("World")
```

In [3]:
```
hello()
hello()
```

```
MainProcess  MainThread  2021-06-10 11:18:01,568  Hello
MainProcess  MainThread  2021-06-10 11:18:02,574  World
MainProcess  MainThread  2021-06-10 11:18:02,575  Hello
MainProcess  MainThread  2021-06-10 11:18:03,576  World
```

In [4]:
```
async def hello():
    logging.info("Hello")
    await asyncio.sleep(1) # Non-blocking "I/O" call
    logging.info("World")
```

In [5]:
```
loop = asyncio.get_running_loop()
loop.create_task(hello())
loop.create_task(hello());
```

```
MainProcess  MainThread  2021-06-10 11:18:04,440  Hello
MainProcess  MainThread  2021-06-10 11:18:04,441  Hello
MainProcess  MainThread  2021-06-10 11:18:05,446  World
MainProcess  MainThread  2021-06-10 11:18:05,448  World
```

Coroutines

In [6]:
```
async def hello():
    logging.info("Hello")
    await asyncio.sleep(1) # Non-blocking "I/O" call
    logging.info("World")
```

In [7]:
```
hello()
```

Out[7]:
```
<coroutine object hello at 0x105099f40>
```

Coroutines are computer program components that generalize subroutines for ***non-preemptive multitasking****, by allowing execution to be* ***suspended and resumed*** *—* Wikipedia

In [8]:
```
coro = hello()           # Instantiate the coroutine, nothing will run yet
future = coro.send(None) # Run the coroutine until it "yields" back control
```

```
MainProcess  MainThread  2021-06-10 11:18:09,134  Hello
```

In [10]:
```
# At this point we have the control back and we get a future
# This will be True after ~1s
future.done()
```

Out[10]:
```
True
```

In [11]:
```
# Resume the coroutine
coro.send(None)
```

```
MainProcess  MainThread  2021-06-10 11:18:12,503  World
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-11-535ffde4991f> in <module>
      1 # Resume the coroutine
----> 2 coro.send(None)

StopIteration:
```

Event loop (1/2)

Runs asynchronous tasks and callbacks, perform network IO operations, and run subprocesses.
One loop per *context*: by default one loop per thread.
Can use an alternative event loop (e.g. QT event loop).
See https://docs.python.org/3/library/asyncio-policy.html.

In [12]:
```python
async def sleep_and_return():
    await asyncio.sleep(1)
    return "Hello World"
```

In [13]:
```python
loop = asyncio.get_running_loop()
task = loop.create_task(sleep_and_return())
# <=> asyncio.create_task(sleep_and_return())
```

In [14]:
```python
task.done()
# task.cancel()
```

Out[14]:
```
True
```

In [15]:
```python
task.result()
```

Out[15]:
```
'Hello World'
```

Event loop (2/2)

In [16]:
```
# See also: loop.call_at
# See also: setTimeout(function, delay) in JavaScript
loop.call_later(1, lambda: print("Hello World"))
```

Out[16]:
```
<TimerHandle when=22.926138333 <lambda>() at <ipython-input-16-91b2edc10bc
5>:3>

Hello World
```

In [17]:
```
# JavaScript-style callbacks
task = asyncio.create_task(sleep_and_return())
task.add_done_callback(lambda x: print("Done: ", x.result()))
```

In [18]:
```
await asyncio.wait_for(sleep_and_return(), 1.5)
```

```
Done:  Hello World
```

Out[18]:
```
'Hello World'
```

Event loop starving

In [19]:

```
# No await, will block the loop
# async def cpu_intensive():
#     logging.info("Computing...")
#     s =  sum(range(10**8))
#     logging.info("Done")
#     return s

# Periodically yields back control to the loop
async def cpu_intensive():
    logging.info("Computing...")
    s = 0
    for _ in range(10):
        s += sum(range(10**7))
        await asyncio.sleep(0)
    logging.info("Done")
    return s
```

In [20]:

```
asyncio.create_task(cpu_intensive())
asyncio.create_task(hello())
asyncio.create_task(hello());
```

```
MainProcess  MainThread  2021-06-10 11:18:26,192  Computing...
MainProcess  MainThread  2021-06-10 11:18:26,310  Hello
MainProcess  MainThread  2021-06-10 11:18:26,311  Hello
MainProcess  MainThread  2021-06-10 11:18:27,170  Done
MainProcess  MainThread  2021-06-10 11:18:27,316  World
MainProcess  MainThread  2021-06-10 11:18:27,316  World
```

Running blocking tasks in thread/process pools

See `process_pool.py`.

Async overhead

In [21]:

```
def add(a, b):
    return a + b

def do():
    for i in range(10**7):
        add(i, i)

with Timer():
    do()
```

```
Total time: 622.796 ms
```

In [22]:

```
async def add(a, b):
    return a + b

async def do():
    for i in range(10**7):
        await add(i, i)

with Timer():
    await do()
```

```
Total time: 1114.945 ms
```

HTTP requests (sync)

In [23]:
```python
def fetch_quote():
    res = httpx.get("https://kaamelott.chaudie.re/api/random")
    return res.json().get("citation", {}).get("citation")
```

In [25]:
```python
with Timer():
    for _ in range(10):
        print(fetch_quote())
```

```
 C’est dingue, cette histoire ! C’est pas parce que vous faites des tartes
pour des petits-enfants qui existent pas que je dois les emmener à la pêch
e, si ?!
(''À Léodagan'')
Votre gendre. Eh ben, il a peur du noir !…
Ils ont rien ramené parce que c'est des débiles. Arrêtez d'envoyer Perceva
l et Karadoc en mission, c'est ridicule !
 Je te mettrai à genoux, Arthur de Bretagne !
 LEODAGAN CONTRE-ATTAQUE !!! ''(il passe la balle à Léodagan)''
 Ooooooooh, non mais faire du labyrinthe avec un trou-de-balle pareil, fau
t drôlement de la vaillance, hein !
 ''(Beurré)'' Sire, vous êtes quand même un sacré souverain. Accueillir de
s péquenots qui sentent la bouse, comme ça, dans votre chapeau, ben je dis
château !
 (''À Bohort'')
Vous êtes marié, comme moi ; vous savez que la monstruosité peut prendre d
es formes très diverses.
 La fleur en bouquet fane, et jamais ne renaît !
 Je préviens monsieur et madame que s'ils ont dans l'idée de remplacer leu
r hypothétique progéniture par des groupes d'amis dans le style de celui-c
i, en ce qui me concerne, y a d'la démission dans l'air.

Total time: 763.456 ms
```

HTTP requests (async)

In [29]:
```python
async def fetch_quote_async():
    async with httpx.AsyncClient() as client:
        res = await client.get("https://kaamelott.chaudie.re/api/random")
    return res.json().get("citation", {}).get("citation")
```

In [30]:
```python
with Timer():
    futures = []
    # 1) Schedule the coroutines
    for _ in range(10):
        futures.append(asyncio.create_task(fetch_quote_async()))
    # 2) Iterate on the coroutines result, as they finish
    for future in asyncio.as_completed(futures):
        print(await future)
    # Alternative to wait on the completion of *all* coroutines:
    # quotes = await asyncio.gather(futures)
```

```
 Désolé mais vous l'avez chié votre mariage. Ce s'rait rien si c'était pas
juste la deuxième fois.
 Allez, vous devriez mettre les bouts, les demi-sels ! C'est gentil d'être
passés ! On va vous faire un p'tit sac avec des restes pour manger chez vo
us.
 Ah non, avec les jolies seulement. C’est de là que j’ai conclu que, comme
y m’touchait pas, je faisais moi-même partie des grosses mochetés.
 La religion c'est le bordel, admettez-le ! Alors laissez-moi prier c'que
j'veux tranquille. 'M'empêche pas d'la chercher, votre saloperie de Graal.
 Noblesse bien remisée ne trouve jamais l'hiver à sa porte... Non, porte c
lose...
 Mon père, il n'était pas ébouriffé, déjà, hein, il avait une coupe à la c
on mais c'était plutôt aplati et puis il était pas vaporeux, voilà ! Alle
z, au lit !
 VIVIANNE : Vous voulez vraiment pas me dire qui vous êtes?
MELEAGANT : Non, mon nom ne vous dira rien... Mais chez vous on m'appell
e... La réponse [...] La réponse à votre pathétique désastre!
 Parce que mon couteau pour le pâté, euuh, y'a rien à faire, jm'en tape.
 Si on cueille pas les cerises tant qu'elles sont sur l'arbre, on fera tin
tin pour le clafoutis.
 Mon père, y dit toujours qu'on arrive jamais en prison par hasard.

Total time: 196.126 ms
```

HTTP requests (async + semaphore)

In [31]:
```python
async def fetch_quote_async(semaphore):
    async with semaphore:
        async with httpx.AsyncClient() as client:
            res = await client.get("https://kaamelott.chaudie.re/api/random")
    return res.json().get("citation", {}).get("citation")
```

In [33]:
```python
semaphore = asyncio.Semaphore(5)

with Timer():
    futures = []
    # 1) Schedule the coroutines
    for _ in range(10):
        futures.append(asyncio.create_task(fetch_quote_async(semaphore)))
    # 2) Iterate on the coroutines result, as they finish
    for future in asyncio.as_completed(futures):
        print(await future)
```

```
Bah ça va, je picole pas souvent !
 Non, moi j'crois qu'il faut qu'vous arrêtiez d'essayer d'dire des trucs. Ça vous fatigue, déjà, et pour les autres, vous vous rendez pas compte de c'que c'est. Moi quand vous faites ça, ça me fout une angoisse... j'pourrais vous tuer, j'crois. De chagrin, hein ! J'vous jure c'est pas bien. Il faut plus que vous parliez avec des gens.
 Ah, mais des tanches pareilles, on devrait les mettre sous verre, hein !
 Pourquoi pas ?
 J'étais tellement en colère, je leur ai lancé un de ces regards… Ils ne sont pas venus chercher la monnaie de leur pièce.
 J'espère que tu fabules, bourgeois de bon aloi ! As-tu la fabulette bien preste ?
 Vous n'êtes pas le plus fort, Môssieur Élias ! Quand on confond un clafoutis et une part de clafoutis, on vient pas la ramener !
 (''À Karadoc'') Vous vous prenez peut-être pour une statue grecque ?
 Du temps de Pendragon, on avait le sens du dramatique : le Lac, Stoneheng e, Avalon… Maintenant, dès qu'ils croisent un dragon, ils font un meeting.
 Ah ouais non mais attends, c'est du joli boulot là, les p'tits sacripants... Oui, sacripants, oui. C'est un terme un peu craignos. D'ailleurs, ben voilà, même craignos, c'est craignos. Mais c'est parce que je suis choqué ! Qu'est-ce que j'entends ? Tu demandes en mariage une personne âgée ? Et la p'tite Julia alors, dans tout ça ? Ah il faut que j'm'en occupe tout seul, c'est ça ? Ah très bien... Merci les p'tits fripons... Tiens, ça aussi, c'est un peu craignos, tu vois ? Mais c'est parce que là ouuuh ! Et bravo général, beau boulot ! Ah les pots cassés, c'est Verinus qui répare les pots cassés tout simplement... Ok d'accord, très bien...

Total time: 201.505 ms
```

Subprocesses

In [34]:
```
async def log_stream(stream):
    while line := await stream.readline():
        logging.info(line.decode('utf-8'))
```

In [35]:
```
proc = await asyncio.create_subprocess_shell(
    "tee --",
    stdin=asyncio.subprocess.PIPE,
    stdout=asyncio.subprocess.PIPE,
)
logger = loop.create_task(log_stream(proc.stdout))
```

In [36]:
```
proc.stdin.write(b"Hello\n")
proc.stdin.write(b"World\n")
```

```
MainProcess  MainThread  2021-06-10 11:19:19,271  Hello

MainProcess  MainThread  2021-06-10 11:19:19,272  World
```

In [37]:
```
logger.cancel()
proc.terminate()
```

Queues

Similar to Go channels.

In [38]:
```python
# From https://docs.python.org/3/library/asyncio-queue.html
async def worker(name, queue):
    while True:
        sleep_for = await queue.get()
        await asyncio.sleep(sleep_for)
        queue.task_done()
        print(f"{name} has slept for {sleep_for:.2f} seconds")
```

In [39]:
```python
queue = asyncio.Queue()
tasks = [
    asyncio.create_task(worker("Worker 1", queue)),
    asyncio.create_task(worker("Worker 2", queue))
]
```

In [40]:
```python
# put_nowait: doesn't block if there is no free slot.
queue.put_nowait(1)
queue.put_nowait(0.5)
queue.put_nowait(0.75)
await queue.join()
```

```
Worker 2 has slept for 0.50 seconds
Worker 1 has slept for 1.00 seconds
Worker 2 has slept for 0.75 seconds
```

In [41]:
```python
[task.cancel() for task in tasks]
```

Out[41]:
```
[True, True]
```

See also

Asynchronous context managers and iterators

`async with`, `async for`
https://www.python.org/dev/peps/pep-0492/#asynchronous-context-managers-and-async-with

Debugging

```
export  PYTHONASYNCIODEBUG=1
```
https://docs.python.org/3/library/asyncio-dev.html

Sockets

`asyncio.open_connection`, `asyncio.start_server`
https://docs.python.org/3/library/asyncio-stream.html

Synchronization primitives

Event, Lock, Semaphore, ...
https://docs.python.org/3/library/asyncio-sync.html

Some asyncio friendly libraries

- aiofiles
- aioch, aiopg, aiomysql, ...
- FastAPI
- Flask 2.0
- httpx
- pytest-asyncio
- trio

Summary

- **Awaitable:** an object that can be used in an `await` expression: coroutines, futures and tasks.
- **Coroutine:** a function that can be paused and resumed at specific point in time.
- **Event loop:** schedule and run coroutines concurrently.
- **Executor:** run synchronous functions asynchronously (e.g. in a thread pool).
- **Future:** an object that represents an on-going computation and its eventual, future, result.
- **Task:** a future that represents a scheduled coroutine.

Summary

	Processes	Threads	Async
Optimize waiting periods	Yes (OS does it)	Yes (OS does it)	Yes
Use all CPU cores	Yes	No	No
Scalability	Low	Medium	High
Use blocking std library functions	Yes	Yes	No
GIL interference	No	Some	No

Miguel Grinberg — Asynchronous Python for the Complete Beginner — PyCon 2017

References

- asyncio — Asynchronous I/O
- A tale of event loops
- Fluent Python
- Miguel Grinberg — Asynchronous Python for the Complete Beginner — PyCon 2017
- PEP 492 -- Coroutines with async and await syntax