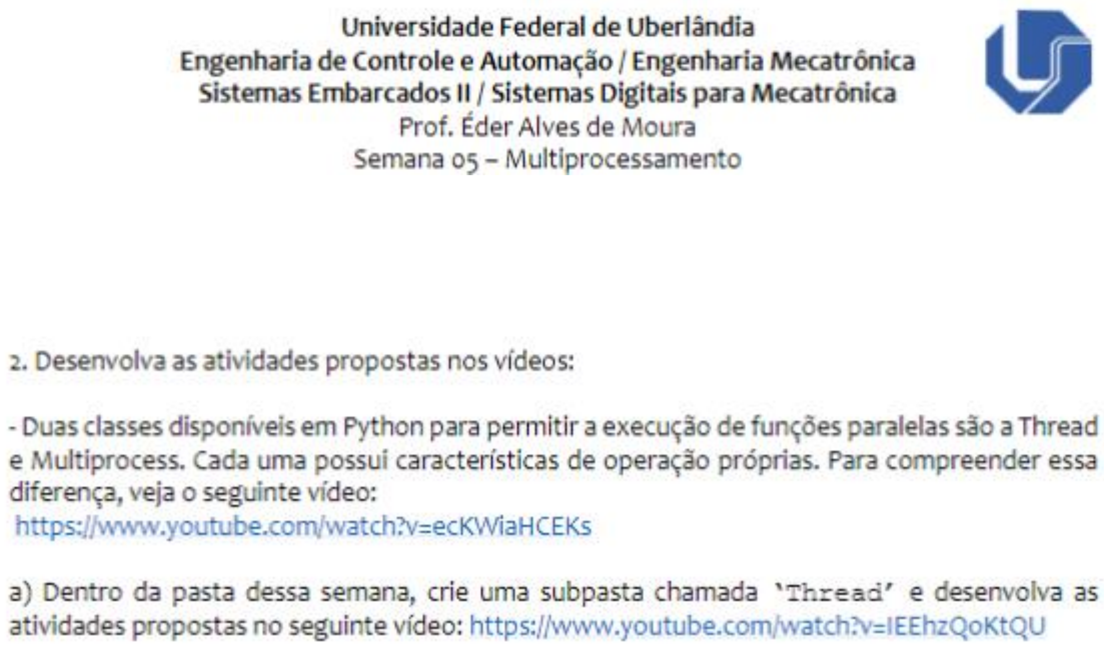


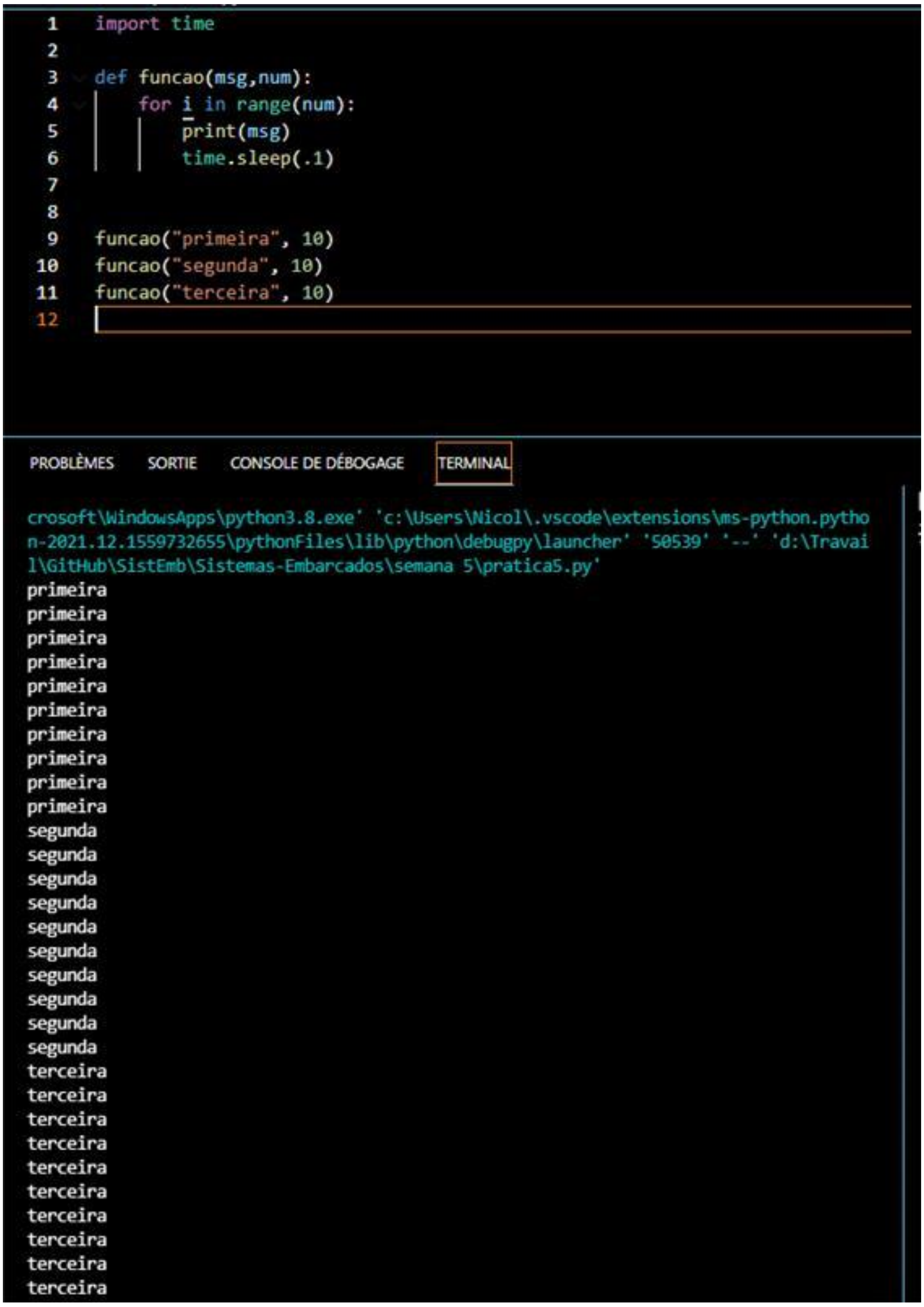
Universidade Federal de Uberlândia
Engenharia de Controle e Automação / Engenharia Mecatrônica
Sistemas Embarcados II / Sistemas Digitais para Mecatrônica
Prof. Éder Alves de Moura
Semana 05 – Multiprocessamento

2. Desenvolva as atividades propostas nos vídeos:

- Duas classes disponíveis em Python para permitir a execução de funções paralelas são a Thread e Multiprocess. Cada uma possui características de operação próprias. Para compreender essa diferença, veja o seguinte vídeo:
https://www.youtube.com/watch?v=ecKWiaHCEKs

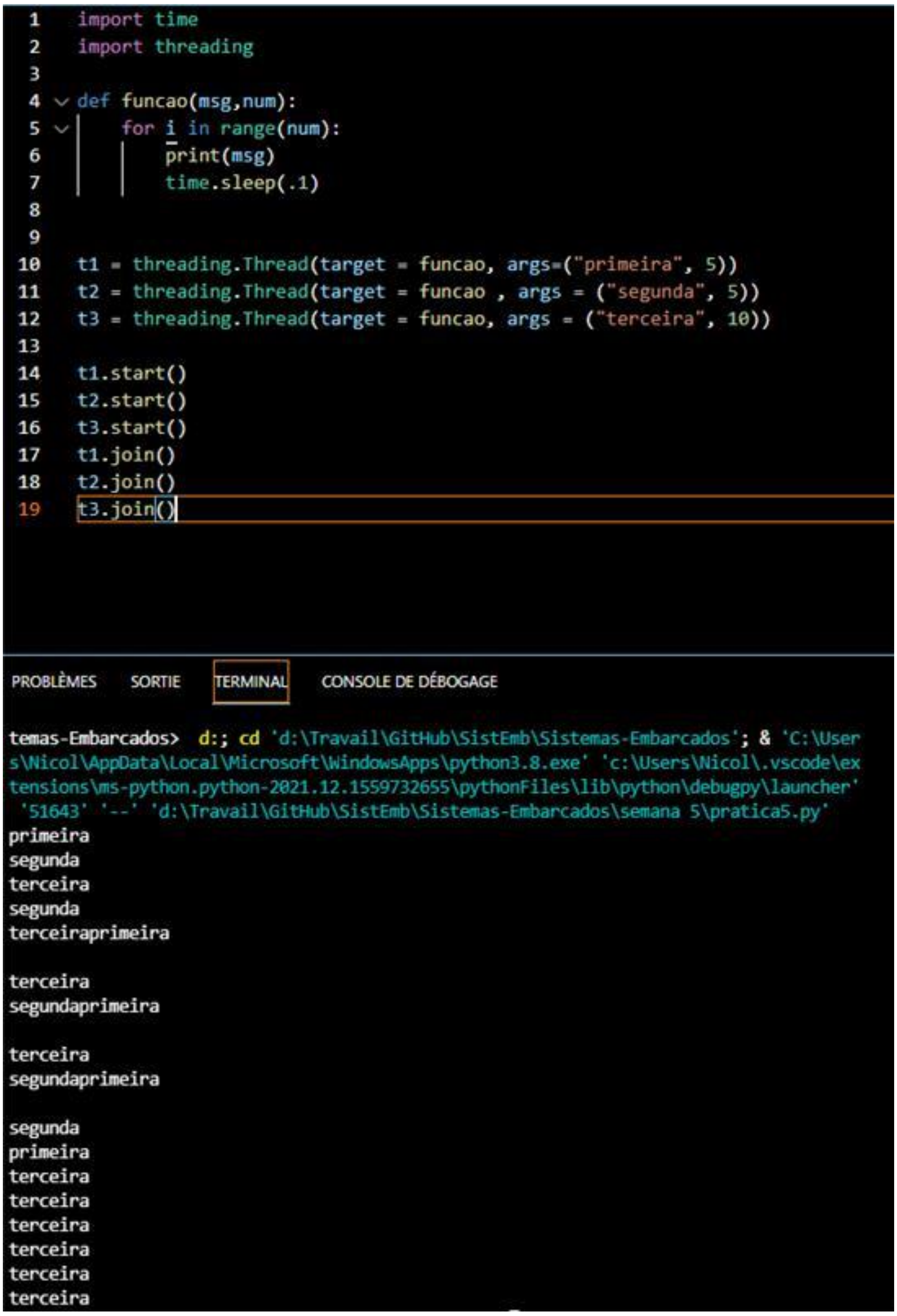
a) Dentro da pasta dessa semana, crie uma subpasta chamada `'Thread'` e desenvolva as atividades propostas no seguinte vídeo: https://www.youtube.com/watch?v=IEEhzQoKtQU

A atividade a) é parecida à atividade desenvolvida na aula pratica.



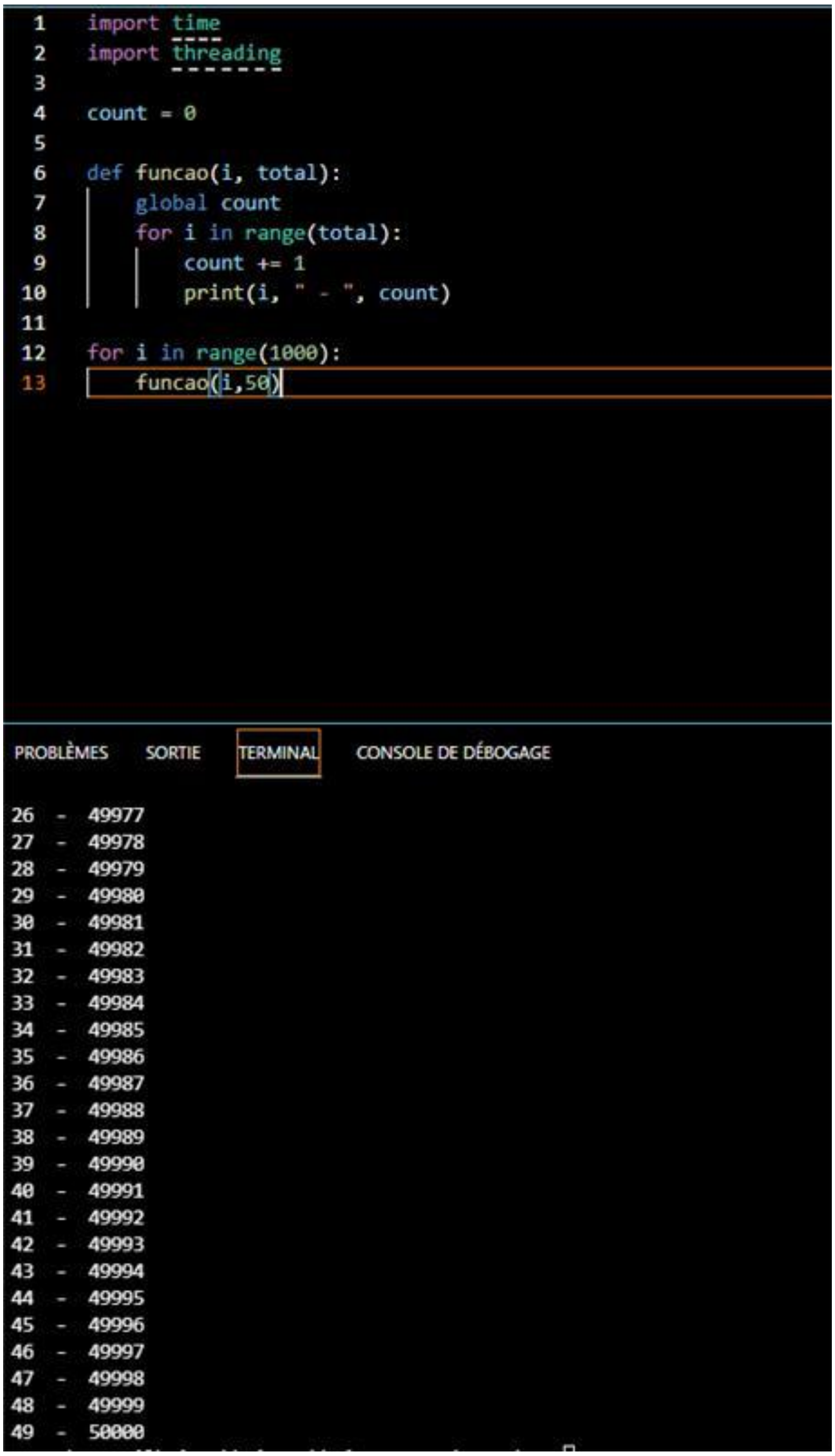
```python
import time

def funcao(msg,num):
    for i in range(num):
        print(msg)
        time.sleep(.1)


funcao("primeira", 10)
funcao("segunda", 10)
funcao("terceira", 10)

```

```
crosoft\WindowsApps\python3.8.exe' 'c:\Users\Nicol\.vscode\extensions\ms-python.pytho
n-2021.12.1559732655\pythonFiles\lib\python\debugpy\launcher' '50539' '--' 'd:\Travai
l\GitHub\SistEmb\Sistemas-Embarcados\semana 5\pratica5.py'
primeira
primeira
primeira
primeira
primeira
primeira
primeira
primeira
primeira
primeira
segunda
segunda
segunda
segunda
segunda
segunda
segunda
segunda
segunda
segunda
terceira
terceira
terceira
terceira
terceira
terceira
terceira
terceira
terceira
terceira
```



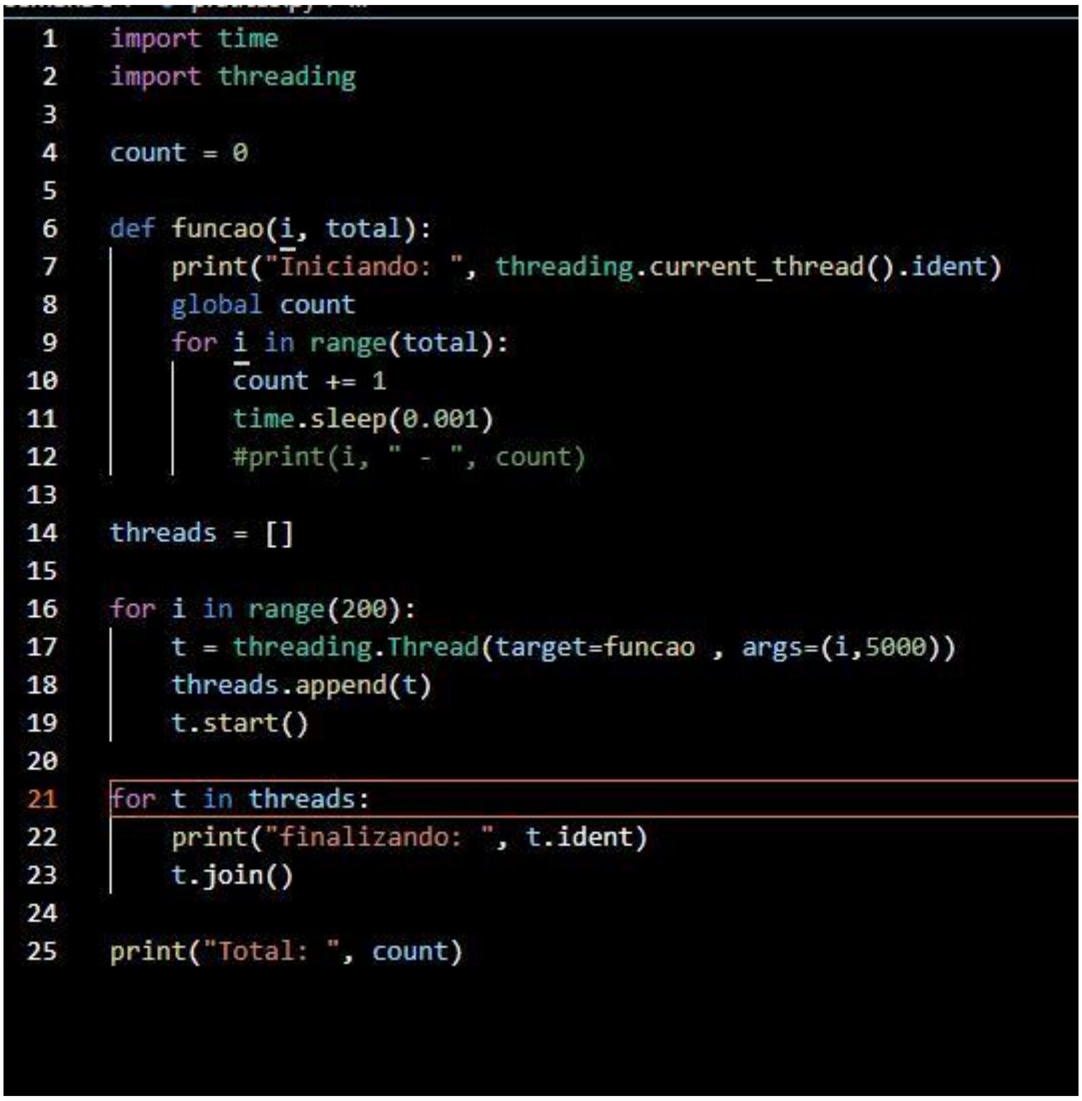
```python
import time
import threading

def funcao(msg,num):
    for i in range(num):
        print(msg)
        time.sleep(.1)


t1 = threading.Thread(target = funcao, args=("primeira", 5))
t2 = threading.Thread(target = funcao , args = ("segunda", 5))
t3 = threading.Thread(target = funcao, args = ("terceira", 10))

t1.start()
t2.start()
t3.start()
t1.join()
t2.join()
t3.join()
```

```
temas-Embarcados> d:; cd 'd:\Travail\GitHub\SistEmb\Sistemas-Embarcados'; & 'C:\User
s\Nicol\AppData\Local\Microsoft\WindowsApps\python3.8.exe' 'c:\Users\Nicol\.vscode\ex
tensions\ms-python.python-2021.12.1559732655\pythonFiles\lib\python\debugpy\launcher'
 '51643' '--' 'd:\Travail\GitHub\SistEmb\Sistemas-Embarcados\semana 5\pratica5.py'
primeira
segunda
terceira
segunda
terceiraprimeira

terceira
segundaprimeira

terceira
segundaprimeira

segunda
primeira
terceira
terceira
terceira
terceira
terceira
terceira
```

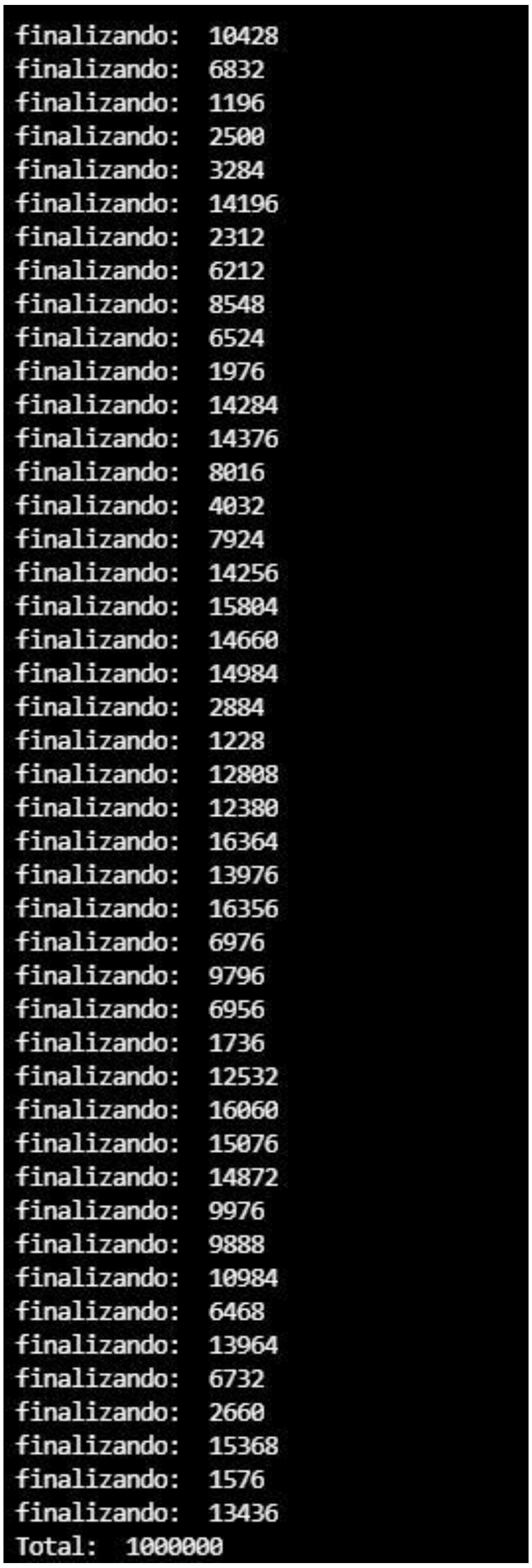


```python
import time
import threading

count = 0

def funcao(i, total):
    global count
    for i in range(total):
        count += 1
        print(i, " - ", count)

for i in range(1000):
    funcao(i,50)
```

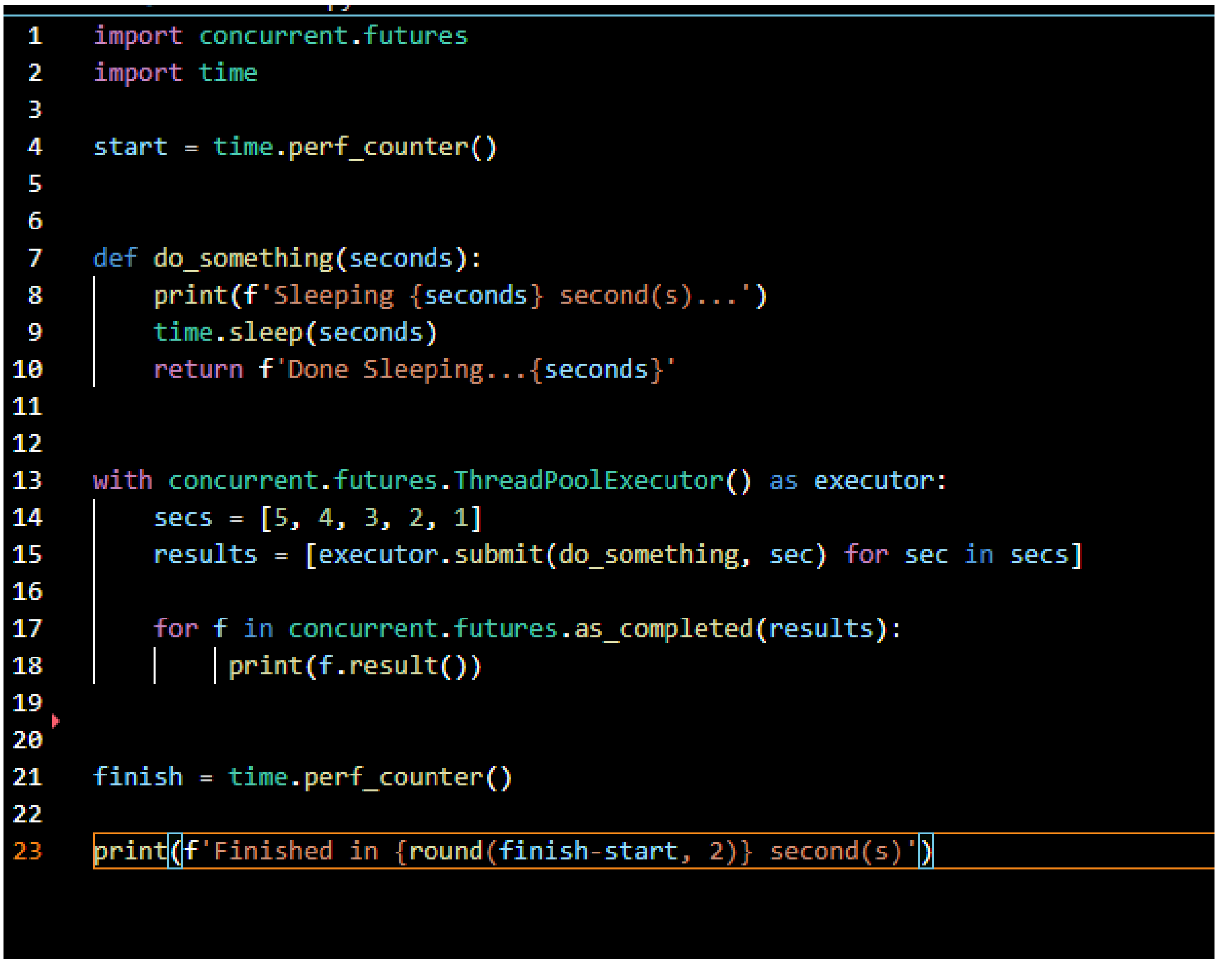
```
26  -  49977
27  -  49978
28  -  49979
29  -  49980
30  -  49981
31  -  49982
32  -  49983
33  -  49984
34  -  49985
35  -  49986
36  -  49987
37  -  49988
38  -  49989
39  -  49990
40  -  49991
41  -  49992
42  -  49993
43  -  49994
44  -  49995
45  -  49996
46  -  49997
47  -  49998
48  -  49999
49  -  50000
```



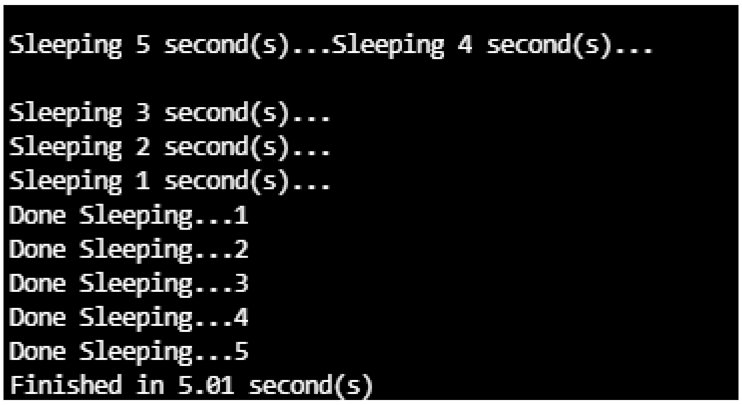
```python
import time
import threading

count = 0

def funcao(i, total):
    print("Iniciando: ", threading.current_thread().ident)
    global count
    for i in range(total):
        count += 1
        time.sleep(0.001)
        #print(i, " - ", count)

threads = []

for i in range(200):
    t = threading.Thread(target=funcao , args=(i,5000))
    threads.append(t)
    t.start()

for t in threads:
    print("finalizando: ", t.ident)
    t.join()

print("Total: ", count)
```



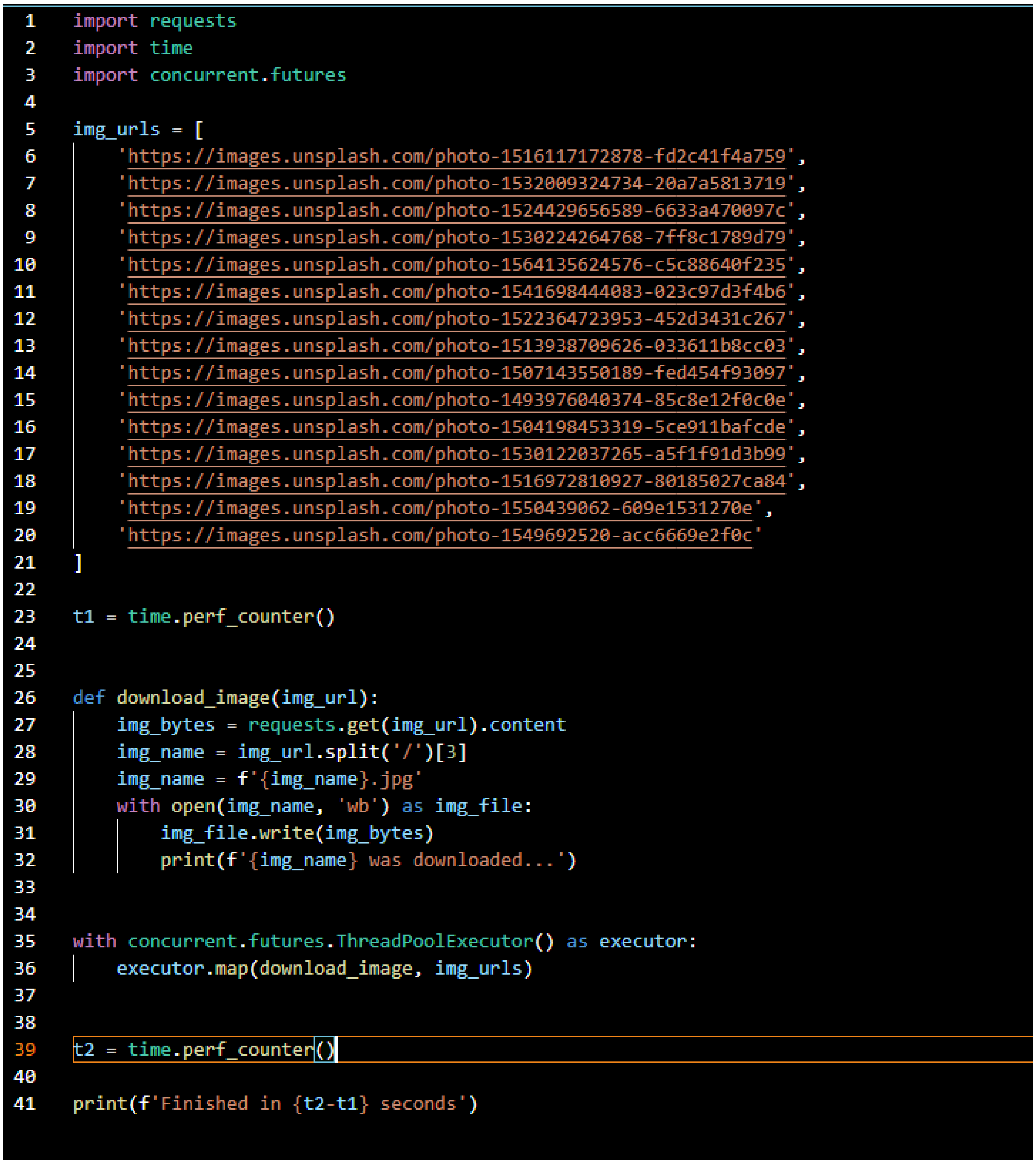
```
finalizando:  10428
finalizando:  6832
finalizando:  1196
finalizando:  2500
finalizando:  3284
finalizando:  14196
finalizando:  2312
finalizando:  6212
finalizando:  8548
finalizando:  6524
finalizando:  1976
finalizando:  14284
finalizando:  14376
finalizando:  8016
finalizando:  4032
finalizando:  7924
finalizando:  14256
finalizando:  15804
finalizando:  14660
finalizando:  14984
finalizando:  2884
finalizando:  1228
finalizando:  12808
finalizando:  12380
finalizando:  16364
finalizando:  13976
finalizando:  16356
finalizando:  6976
finalizando:  9796
finalizando:  6956
finalizando:  1736
finalizando:  12532
finalizando:  16060
finalizando:  15076
finalizando:  14872
finalizando:  9976
finalizando:  9888
finalizando:  10984
finalizando:  6468
finalizando:  13964
finalizando:  6732
finalizando:  2660
finalizando:  15368
finalizando:  1576
finalizando:  13436
Total:  1000000
```



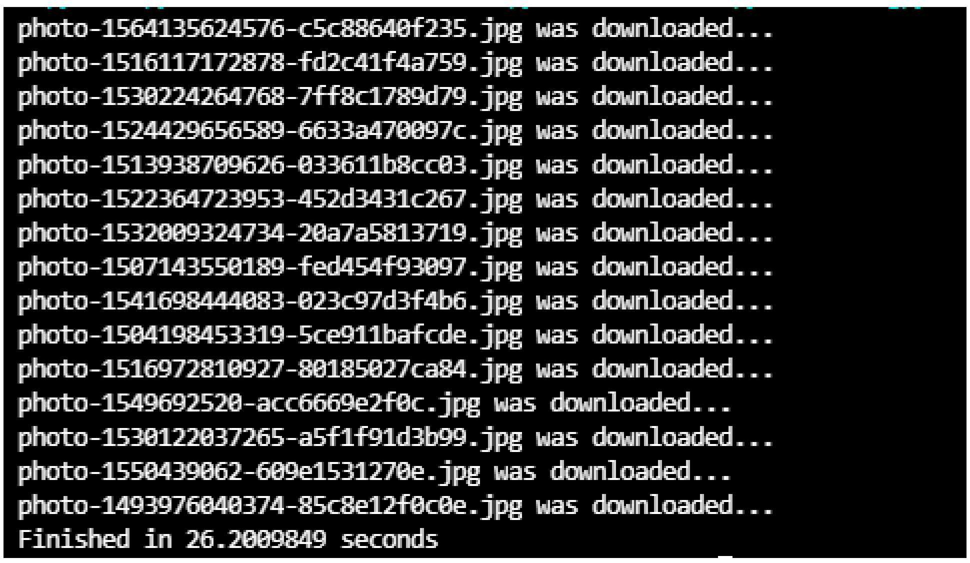
```python
import concurrent.futures
import time

start = time.perf_counter()


def do_something(seconds):
    print(f'Sleeping {seconds} second(s)...')
    time.sleep(seconds)
    return f'Done Sleeping...{seconds}'


with concurrent.futures.ThreadPoolExecutor() as executor:
    secs = [5, 4, 3, 2, 1]
    results = [executor.submit(do_something, sec) for sec in secs]

    for f in concurrent.futures.as_completed(results):
        print(f.result())


finish = time.perf_counter()

print(f'Finished in {round(finish-start, 2)} second(s)')
```



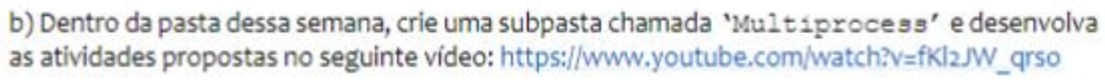
```
Sleeping 5 second(s)...Sleeping 4 second(s)...

Sleeping 3 second(s)...
Sleeping 2 second(s)...
Sleeping 1 second(s)...
Done Sleeping...1
Done Sleeping...2
Done Sleeping...3
Done Sleeping...4
Done Sleeping...5
Finished in 5.01 second(s)
```



```python
import requests
import time
import concurrent.futures

img_urls = [
    'https://images.unsplash.com/photo-1516117172878-fd2c41f4a759',
    'https://images.unsplash.com/photo-1532009324734-20a7a5813719',
    'https://images.unsplash.com/photo-1524429656589-6633a470097c',
    'https://images.unsplash.com/photo-1530224264768-7ff8c1789d79',
    'https://images.unsplash.com/photo-1564135624576-c5c88640f235',
    'https://images.unsplash.com/photo-1541698444083-023c97d3f4b6',
    'https://images.unsplash.com/photo-1522364723953-452d3431c267',
    'https://images.unsplash.com/photo-1513938709626-033611b8cc03',
    'https://images.unsplash.com/photo-1507143550189-fed454f93097',
    'https://images.unsplash.com/photo-1493976040374-85c8e12f0c0e',
    'https://images.unsplash.com/photo-1504198453319-5ce911bafcde',
    'https://images.unsplash.com/photo-1530122037265-a5f1f91d3b99',
    'https://images.unsplash.com/photo-1516972810927-80185027ca84',
    'https://images.unsplash.com/photo-1550439062-609e1531270e',
    'https://images.unsplash.com/photo-1549692520-acc6669e2f0c'
]

t1 = time.perf_counter()


def download_image(img_url):
    img_bytes = requests.get(img_url).content
    img_name = img_url.split('/')[3]
    img_name = f'{img_name}.jpg'
    with open(img_name, 'wb') as img_file:
        img_file.write(img_bytes)
        print(f'{img_name} was downloaded...')


with concurrent.futures.ThreadPoolExecutor() as executor:
    executor.map(download_image, img_urls)


t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

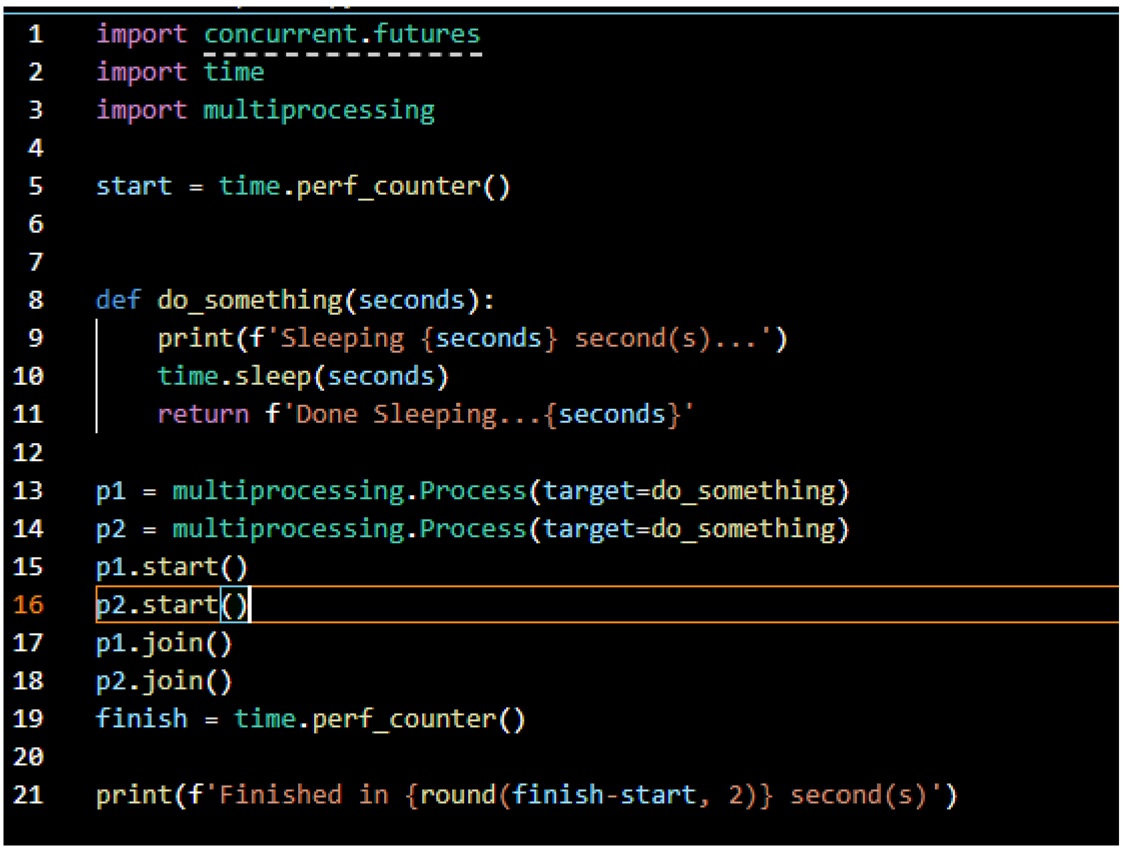


```
photo-1564135624576-c5c88640f235.jpg was downloaded...
photo-1516117172878-fd2c41f4a759.jpg was downloaded...
photo-1530224264768-7ff8c1789d79.jpg was downloaded...
photo-1524429656589-6633a470097c.jpg was downloaded...
photo-1513938709626-033611b8cc03.jpg was downloaded...
photo-1522364723953-452d3431c267.jpg was downloaded...
photo-1532009324734-20a7a5813719.jpg was downloaded...
photo-1507143550189-fed454f93097.jpg was downloaded...
photo-1541698444083-023c97d3f4b6.jpg was downloaded...
photo-1504198453319-5ce911bafcde.jpg was downloaded...
photo-1516972810927-80185027ca84.jpg was downloaded...
photo-1549692520-acc6669e2f0c.jpg was downloaded...
photo-1530122037265-a5f1f91d3b99.jpg was downloaded...
photo-1550439062-609e1531270e.jpg was downloaded...
photo-1493976040374-85c8e12f0c0e.jpg was downloaded...
Finished in 26.2009849 seconds
```

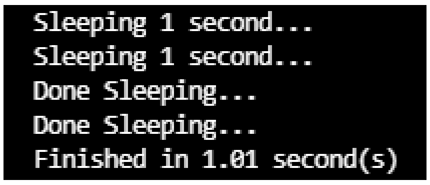


b) Dentro da pasta dessa semana, crie uma subpasta chamada 'Multiprocess' e desenvolva as atividades propostas no seguinte vídeo: https://www.youtube.com/watch?v=fKl2JW_qrso

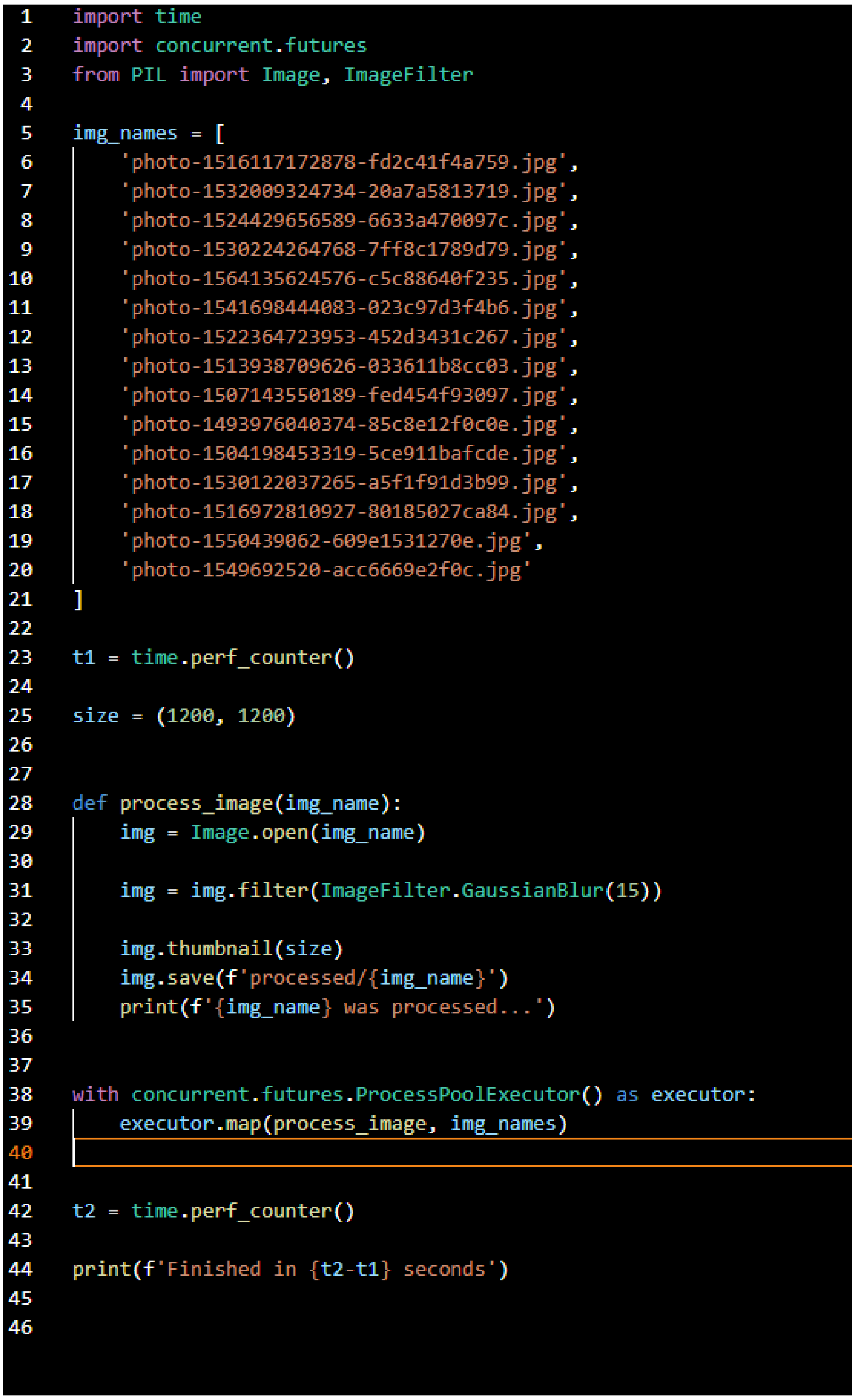
Multi-processing é parecido com Threading na aplicação, mas pode ser mais o menos rápido em função das aplicações e do computador.



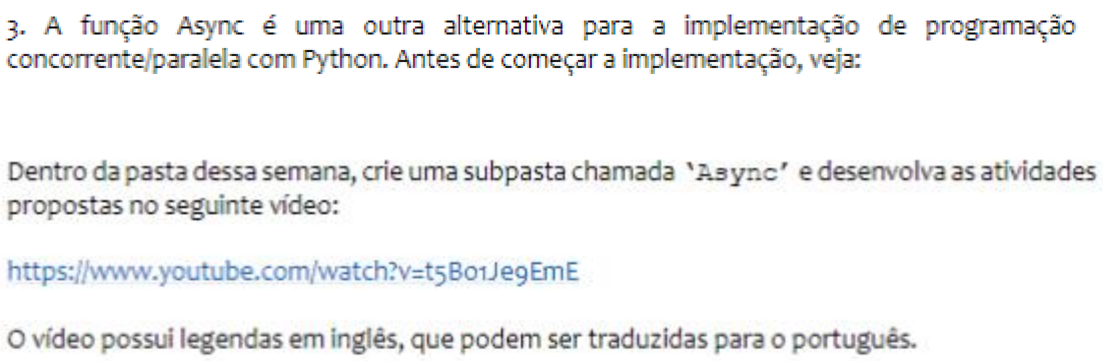
```python
import concurrent.futures
import time
import multiprocessing

start = time.perf_counter()


def do_something(seconds):
    print(f'Sleeping {seconds} second(s)...')
    time.sleep(seconds)
    return f'Done Sleeping...{seconds}'

p1 = multiprocessing.Process(target=do_something)
p2 = multiprocessing.Process(target=do_something)
p1.start()
p2.start()
p1.join()
p2.join()
finish = time.perf_counter()

print(f'Finished in {round(finish-start, 2)} second(s)')
```



```
Sleeping 1 second...
Sleeping 1 second...
Done Sleeping...
Done Sleeping...
Finished in 1.01 second(s)
```



```python
import time
import concurrent.futures
from PIL import Image, ImageFilter

img_names = [
    'photo-1516117172878-fd2c41f4a759.jpg',
    'photo-1532009324734-20a7a5813719.jpg',
    'photo-1524429656589-6633a470097c.jpg',
    'photo-1530224264768-7ff8c1789d79.jpg',
    'photo-1564135624576-c5c88640f235.jpg',
    'photo-1541698444083-023c97d3f4b6.jpg',
    'photo-1522364723953-452d3431c267.jpg',
    'photo-1513938709626-033611b8cc03.jpg',
    'photo-1507143550189-fed454f93097.jpg',
    'photo-1493976040374-85c8e12f0c0e.jpg',
    'photo-1504198453319-5ce911bafcde.jpg',
    'photo-1530122037265-a5f1f91d3b99.jpg',
    'photo-1516972810927-80185027ca84.jpg',
    'photo-1550439062-609e1531270e.jpg',
    'photo-1549692520-acc6669e2f0c.jpg'
]

t1 = time.perf_counter()

size = (1200, 1200)


def process_image(img_name):
    img = Image.open(img_name)

    img = img.filter(ImageFilter.GaussianBlur(15))

    img.thumbnail(size)
    img.save(f'processed/{img_name}')
    print(f'{img_name} was processed...')


with concurrent.futures.ProcessPoolExecutor() as executor:
    executor.map(process_image, img_names)


t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```



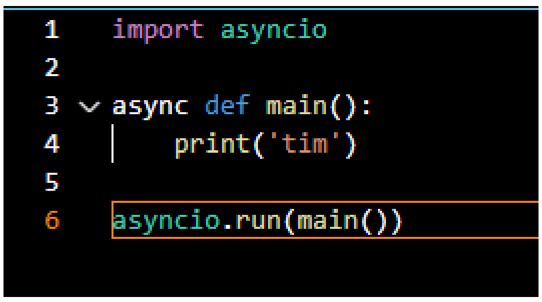
3. A função Async é uma outra alternativa para a implementação de programação concorrente/paralela com Python. Antes de começar a implementação, veja:

Dentro da pasta dessa semana, crie uma subpasta chamada `'Async'` e desenvolva as atividades propostas no seguinte vídeo:

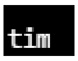
https://www.youtube.com/watch?v=t5Bo1Je9EmE

O vídeo possui legendas em inglês, que podem ser traduzidas para o português.

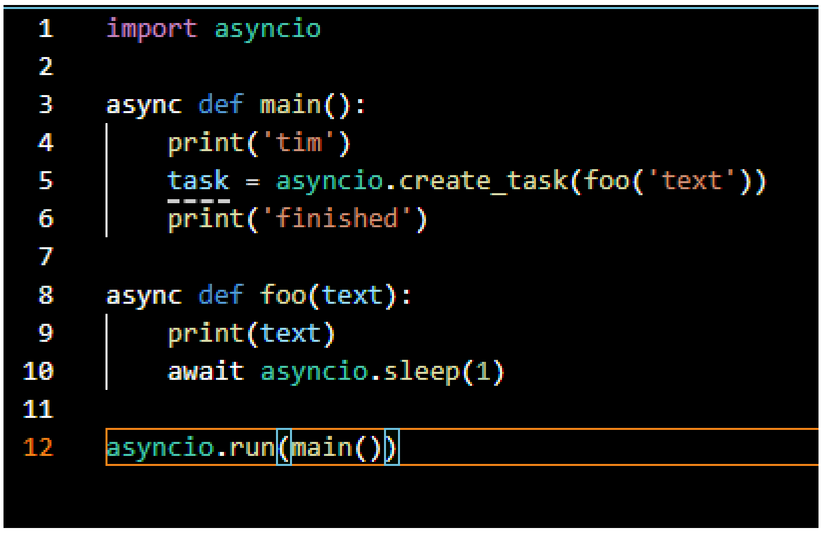
A biblioteca asyncio permite programar de forma assíncrona. Ela gerencia a os 'futuros' e os threads.



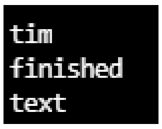
```python
import asyncio

async def main():
    print('tim')

asyncio.run(main())
```



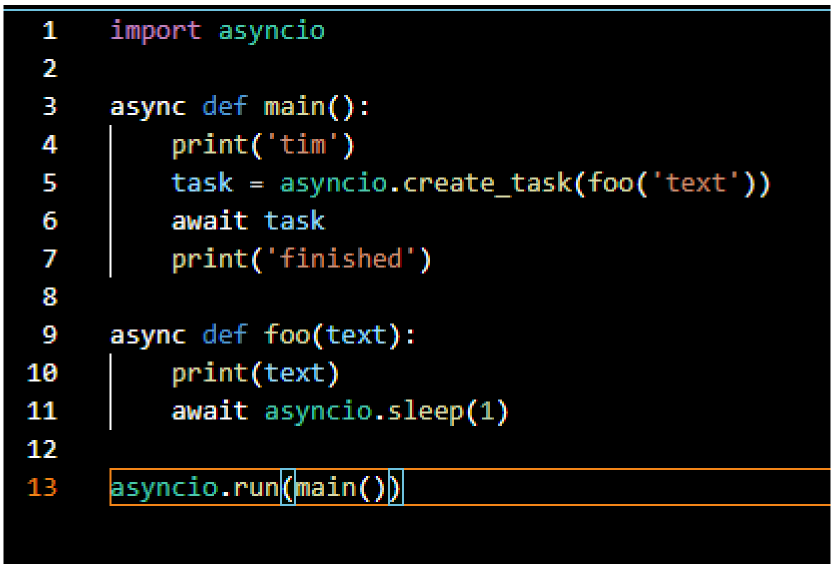
```
tim
```



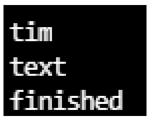
```python
import asyncio

async def main():
    print('tim')
    task = asyncio.create_task(foo('text'))
    print('finished')

async def foo(text):
    print(text)
    await asyncio.sleep(1)

asyncio.run(main())
```



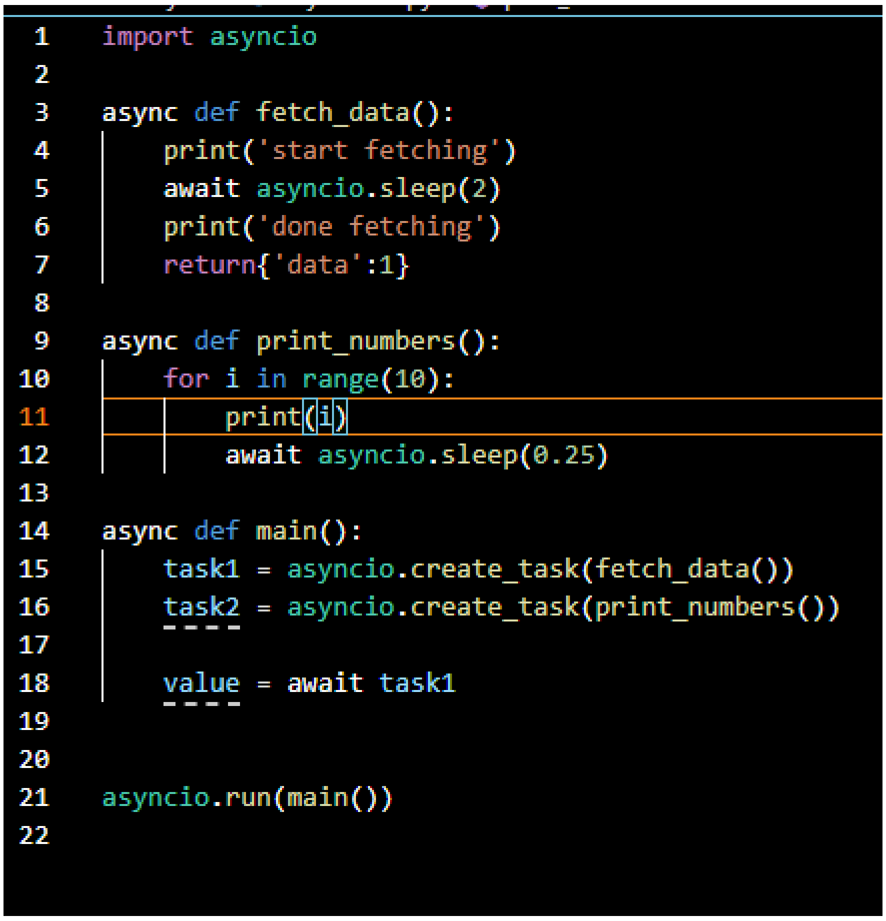
```
tim
finished
text
```



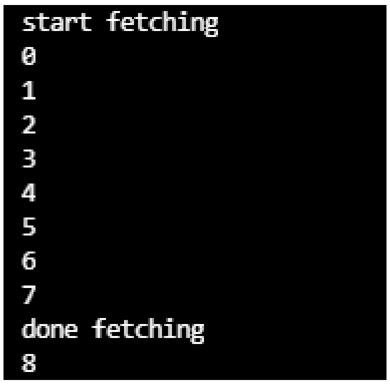
```python
import asyncio

async def main():
    print('tim')
    task = asyncio.create_task(foo('text'))
    await task
    print('finished')

async def foo(text):
    print(text)
    await asyncio.sleep(1)

asyncio.run(main())
```



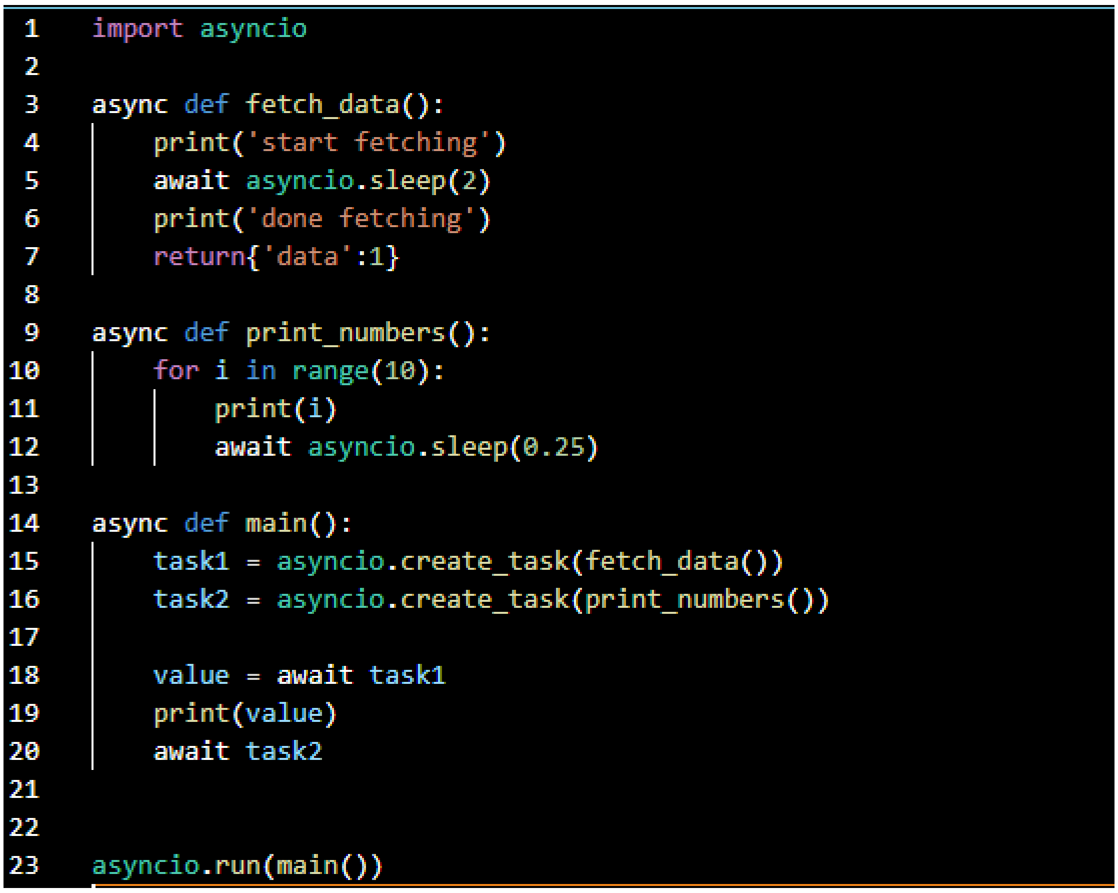
```
tim
text
finished
```



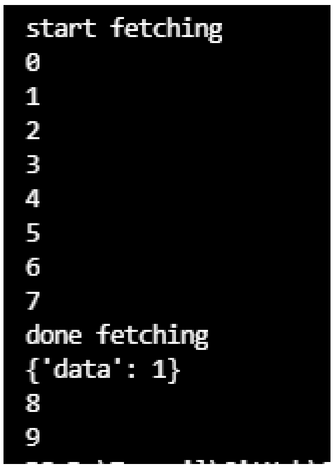
```python
import asyncio

async def fetch_data():
    print('start fetching')
    await asyncio.sleep(2)
    print('done fetching')
    return{'data':1}

async def print_numbers():
    for i in range(10):
        print(i)
        await asyncio.sleep(0.25)

async def main():
    task1 = asyncio.create_task(fetch_data())
    task2 = asyncio.create_task(print_numbers())

    value = await task1


asyncio.run(main())
```



```
start fetching
0
1
2
3
4
5
6
7
done fetching
8
```



```python
import asyncio

async def fetch_data():
    print('start fetching')
    await asyncio.sleep(2)
    print('done fetching')
    return{'data':1}

async def print_numbers():
    for i in range(10):
        print(i)
        await asyncio.sleep(0.25)

async def main():
    task1 = asyncio.create_task(fetch_data())
    task2 = asyncio.create_task(print_numbers())

    value = await task1
    print(value)
    await task2


asyncio.run(main())
```



```
start fetching
0
1
2
3
4
5
6
7
done fetching
{'data': 1}
8
9
```