Multi Threading

BY MOHAMMAD HASAN KHODDAMI

What is Multi Threading

- Each process has some threads (sometimes a process just has one thread) the threads make the process.
- What is process?
- When you run a program and it`s working we call it a process
- Each program that we work with it, is a process

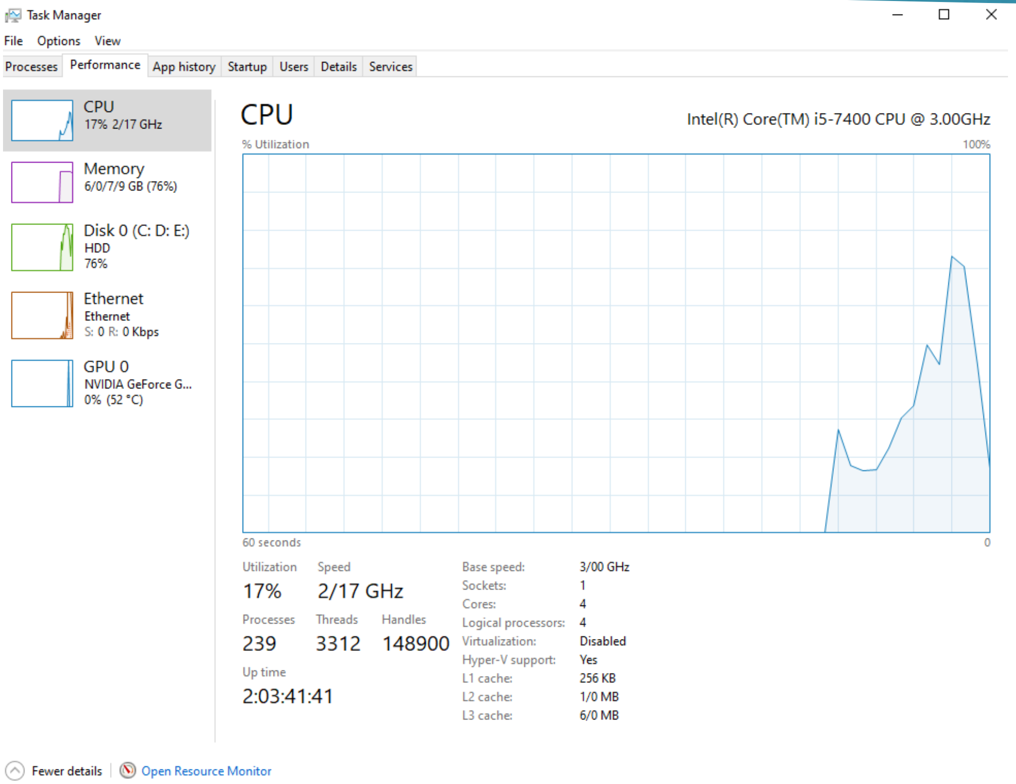
Multi Threading real world example

Imagine this car
The car is a process

The Components of this car are threads like:
Break , engine, windows , tires and etc.

Because each components do their own jobs
Engine for power , break for stop and etc.

Multi treading CPU Example



CPU
17% 2/17 GHz

Memory
6/0/7/9 GB (76%)

Disk 0 (C: D: E:)
HDD
76%

Ethernet
Ethernet
S: 0 R: 0 Kbps

GPU 0
NVIDIA GeForce G...
0% (52 °C)

CPU
Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz

Utilization
17%

Speed
2/17 GHz

Processes
239

Threads
3312

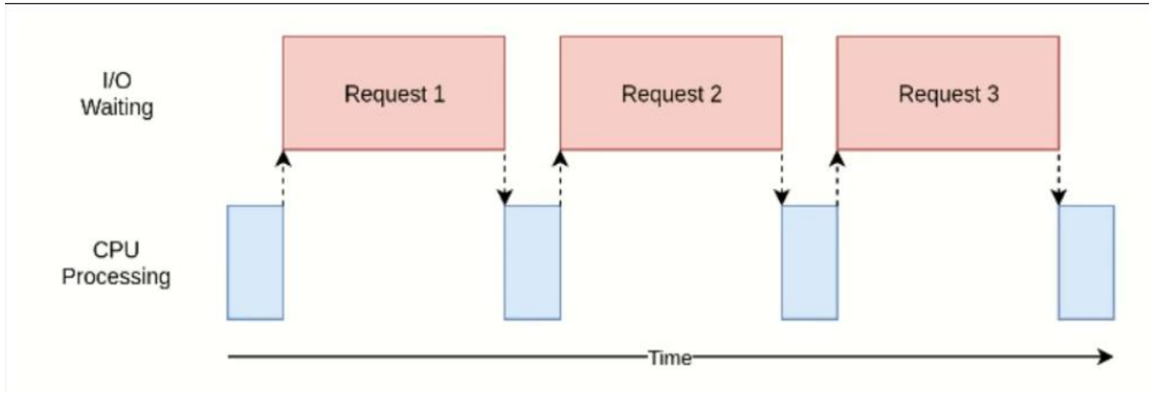
Handles
148900

Up time
2:03:41:41

Base speed: 3/00 GHz
Sockets: 1
Cores: 4
Logical processors: 4
Virtualization: Disabled
Hyper-V support: Yes
L1 cache: 256 KB
L2 cache: 1/0 MB
L3 cache: 6/0 MB

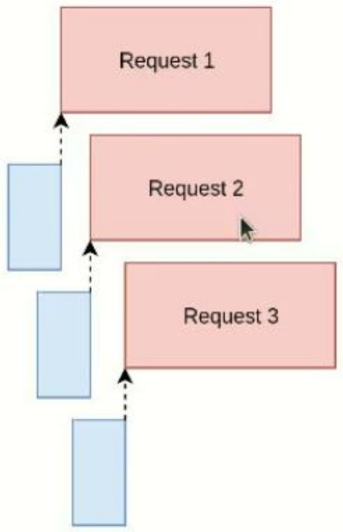
My CPU has 239 process right now
And 3312 Threads.
it means each process avg has 13
threads (it`s could be less or more)

I/O bound



1- we send a request (sending an input)(CPU working)
2- request processing (CPU not working)
2.5 – because CPU is waiting for the output(answer of the request)

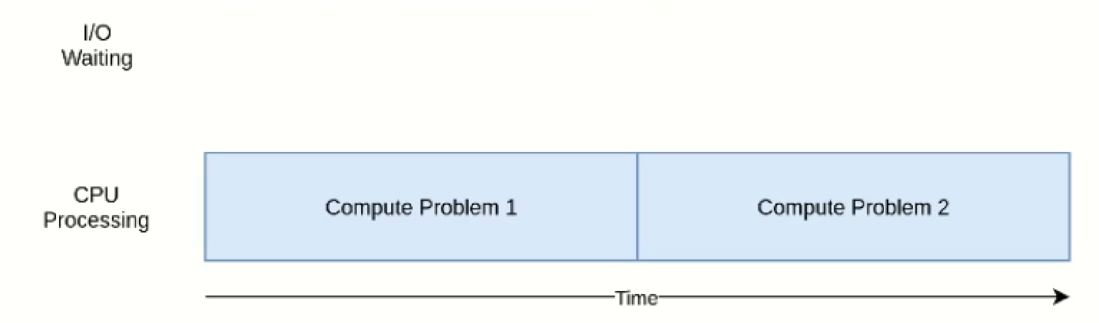
I/O Bound Multi Threading



In multi threading I/O Bound

A. CPU sends a request (input)
B. CPU doesn`t wait for answer(output)
C. CPU sends the second request (input)
D. CPU sends request as many as possible (input)
E. Then CPU waiting for answers(outputs)

CPU processing (Multi Processing)



In this method there is no inputs/outputs and there is no request/answer.
CPU process all by itself
It`s multi processing.

When we should use Multi Threading?

- When we have issue with I/O Bound
- And we want to increase our program runtime speed
- Notice :
- In python there is no advance Multi Threading if we want to

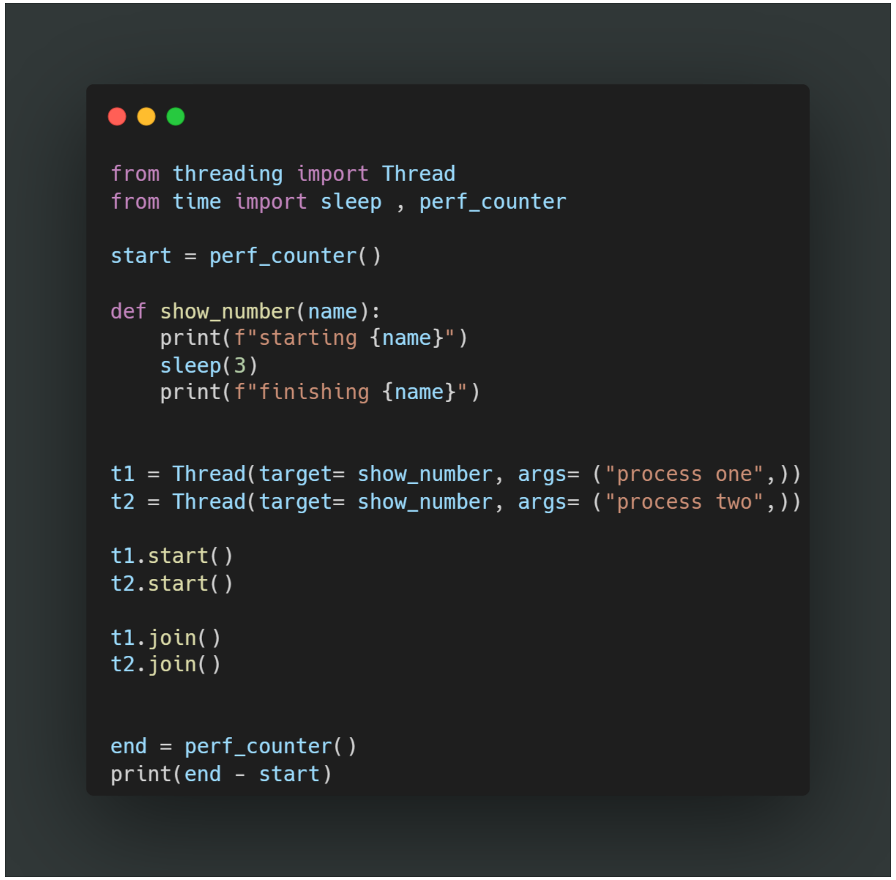
  Do some advance multi threading we should use some Programing Language like GO

Create a Thread

- There is 2 ways for create a thread in python
- A. by passing a callable object to constructor
- B. by overriding the run() method in a subclass
- Notice:

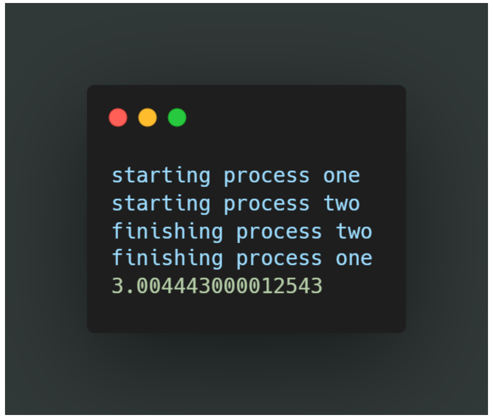
If you decide to create a subclass you have to only override the __init__ and run() method of this class.

Create Thread by method A



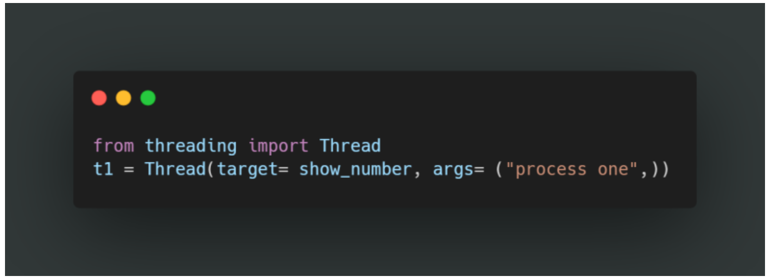
```python
from threading import Thread
from time import sleep , perf_counter

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    sleep(3)
    print(f"finishing {name}")


t1 = Thread(target= show_number, args= ("process one",))
t2 = Thread(target= show_number, args= ("process two",))

t1.start()
t2.start()

t1.join()
t2.join()


end = perf_counter()
print(end - start)
```

Output

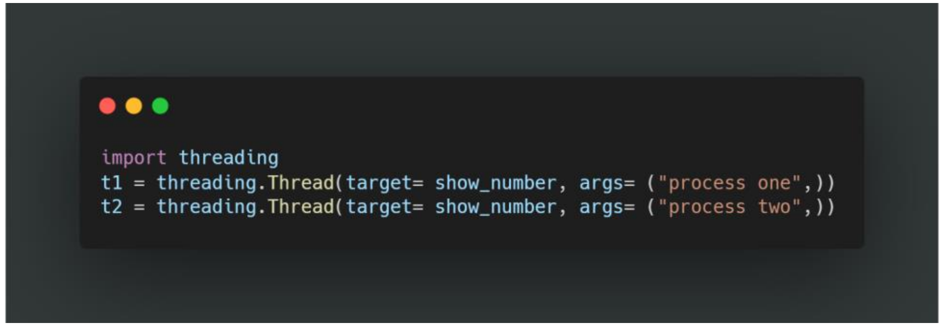


```
starting process one
starting process two
finishing process two
finishing process one
3.004443000012543
```

Frequently Asked Questions

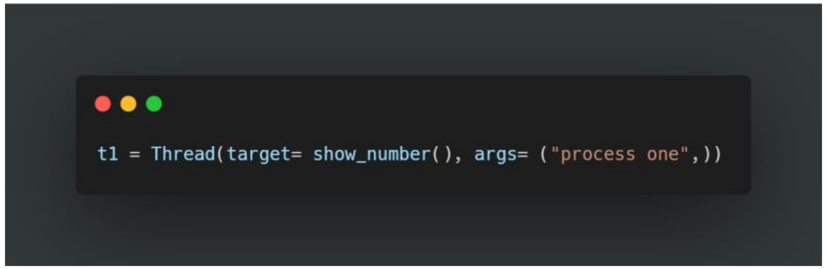


```
from threading import Thread
t1 = Thread(target= show_number, args= ("process one",))
```



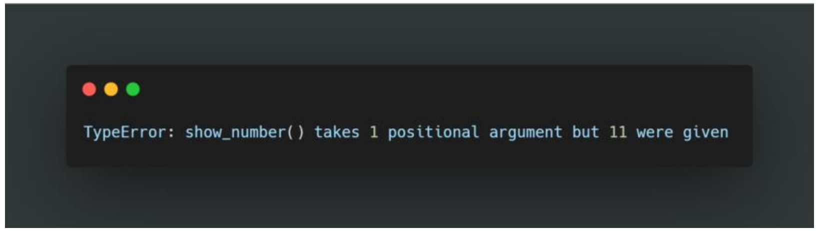
```
import threading
t1 = threading.Thread(target= show_number, args= ("process one",))
t2 = threading.Thread(target= show_number, args= ("process two",))
```

We can also coding like this



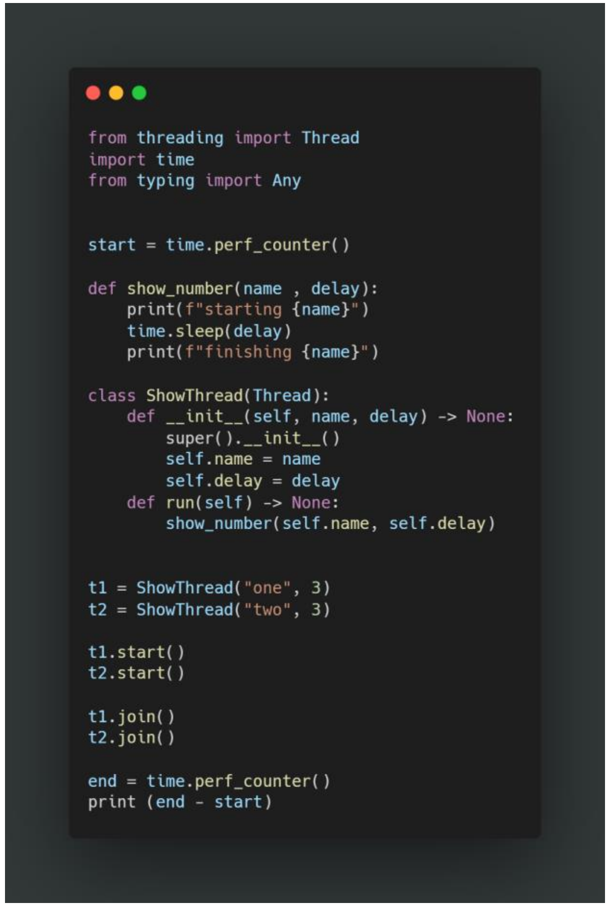
```
t1 = Thread(target= show_number(), args= ("process one",))
```

The comma at the end of arg is important if we don`t use comma we get an error



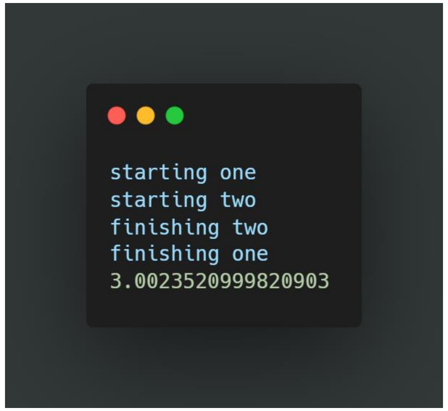
```
TypeError: show_number() takes 1 positional argument but 11 were given
```

Create Thread by method B



```python
from threading import Thread
import time
from typing import Any


start = time.perf_counter()

def show_number(name , delay):
    print(f"starting {name}")
    time.sleep(delay)
    print(f"finishing {name}")

class ShowThread(Thread):
    def __init__(self, name, delay) -> None:
        super().__init__()
        self.name = name
        self.delay = delay
    def run(self) -> None:
        show_number(self.name, self.delay)


t1 = ShowThread("one", 3)
t2 = ShowThread("two", 3)

t1.start()
t2.start()

t1.join()
t2.join()

end = time.perf_counter()
print (end - start)
```

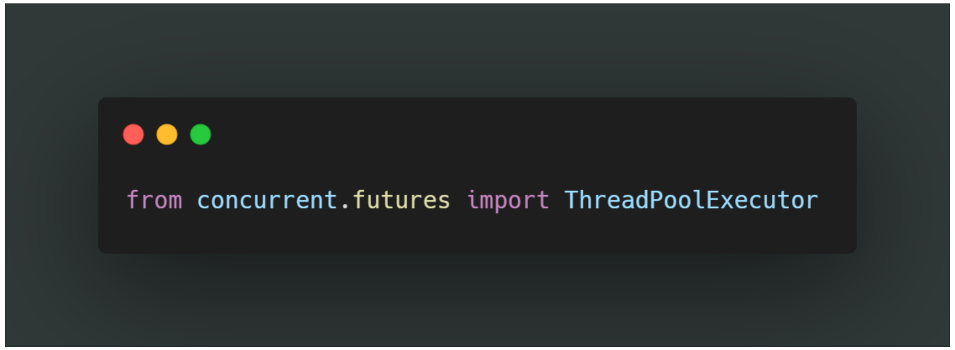
Output



```
starting one
starting two
finishing two
finishing one
3.0023520999820903
```

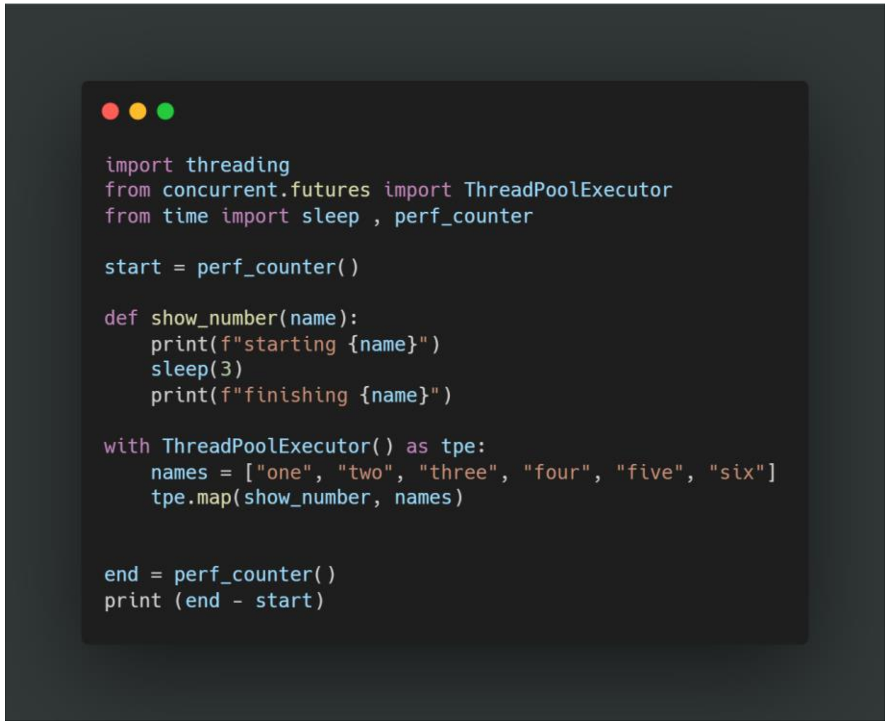
What if we want to create many threads?

- Obviously we can`t define a hundred variables !
- We have to use ThreadPoolExecutor
- For using this class you have to import concurrent.futures



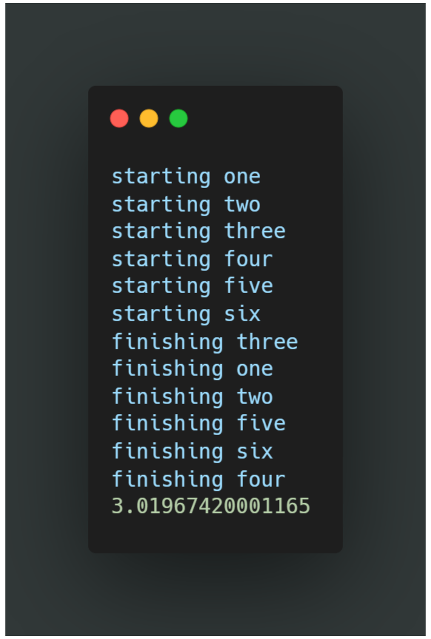
```
from concurrent.futures import ThreadPoolExecutor
```

ThreadPoolExecutor



```python
import threading
from concurrent.futures import ThreadPoolExecutor
from time import sleep , perf_counter

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    sleep(3)
    print(f"finishing {name}")

with ThreadPoolExecutor() as tpe:
    names = ["one", "two", "three", "four", "five", "six"]
    tpe.map(show_number, names)


end = perf_counter()
print (end - start)
```

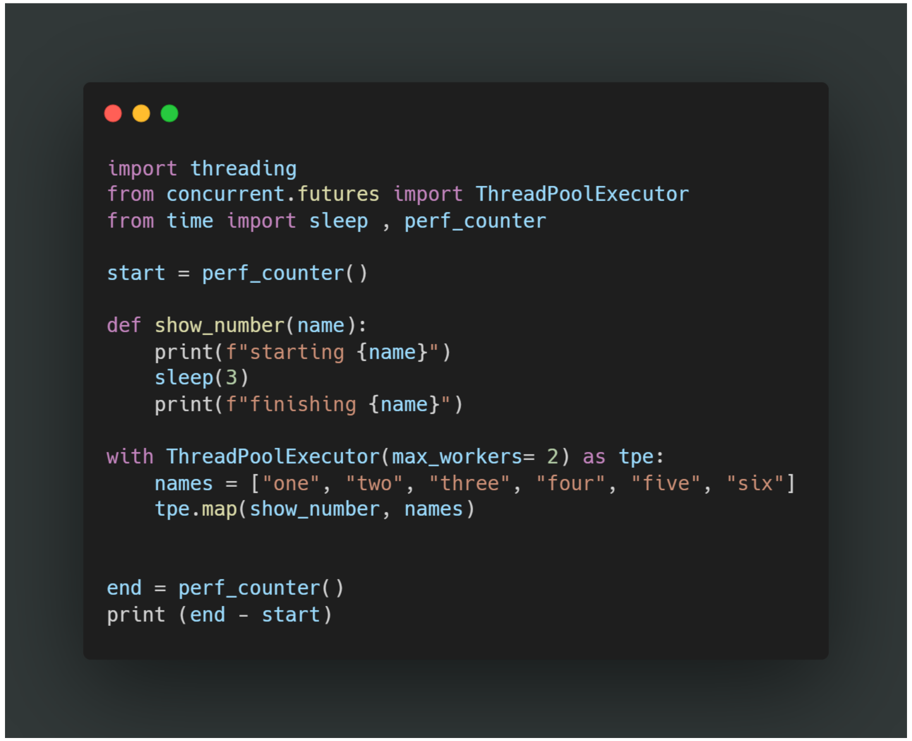
Output



```
starting one
starting two
starting three
starting four
starting five
starting six
finishing three
finishing one
finishing two
finishing five
finishing six
finishing four
3.01967420001165
```

Max_workers?

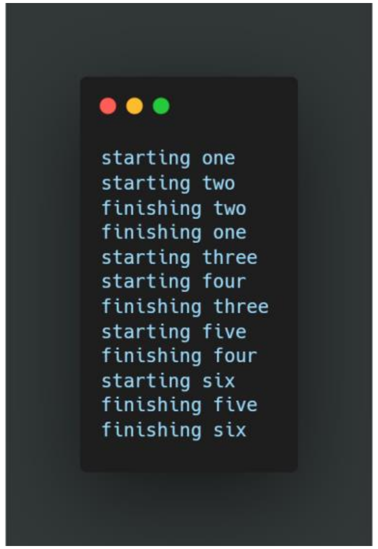
- Using max_workers parameter helps us to thread by defined workers!



```python
import threading
from concurrent.futures import ThreadPoolExecutor
from time import sleep , perf_counter

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    sleep(3)
    print(f"finishing {name}")

with ThreadPoolExecutor(max_workers= 2) as tpe:
    names = ["one", "two", "three", "four", "five", "six"]
    tpe.map(show_number, names)


end = perf_counter()
print (end - start)
```

In this code we thread objects 2 by 2

Output



```
starting one
starting two
finishing two
finishing one
starting three
starting four
finishing three
starting five
finishing four
starting six
finishing five
finishing six
```

Daemon

- What is Daemon in general ? The programs that running in the background are Daemons.
- What is Daemon in python? The Threads which program can ignore them and finish the process
- By default Daemon = False and program have to wait till process finish the Thread
- Notice:

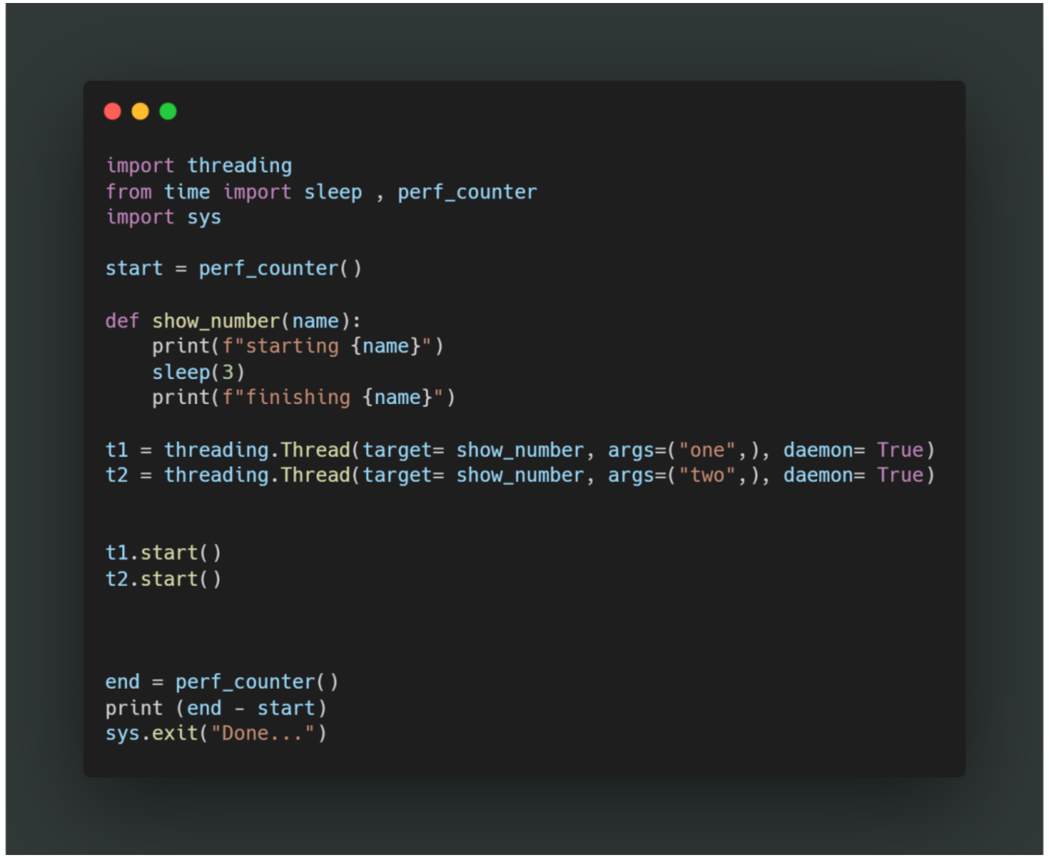
If we want use Daemon parameter , we have to code that before start()

Daemon has 2 method : isDaemon , setDaemon

Daemon = True and join()

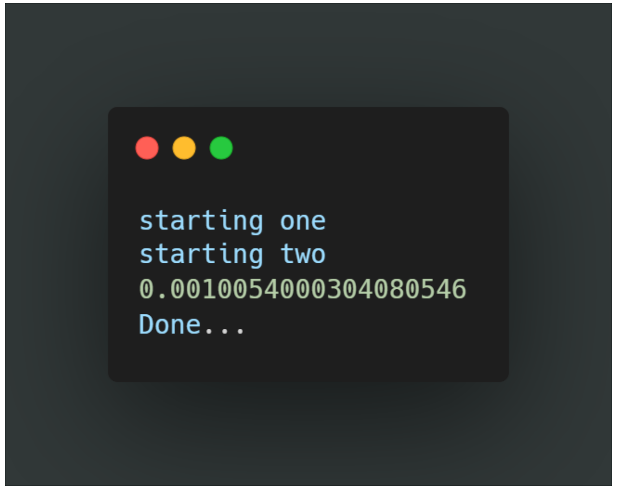
- Join() has priority to Daemon which means if we using join() Daemon won`t work any more
- What does join() do ? Wait until the thread terminates
- So if we use join() doesn`t matter Daemon is False or True , Daemons won`t work

Example



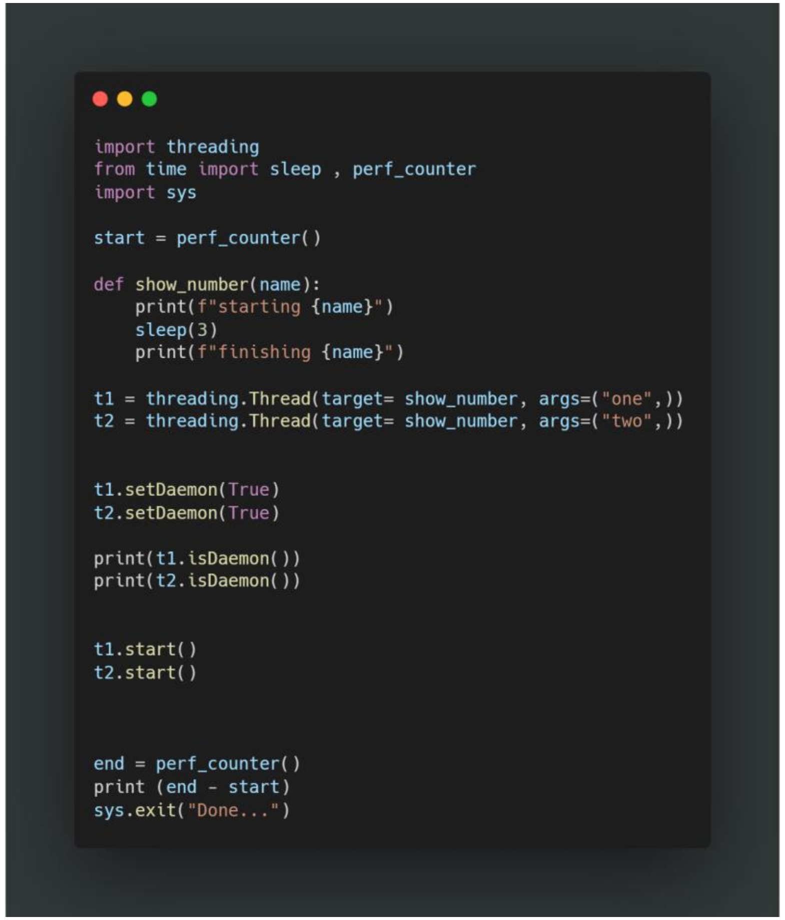
```python
import threading
from time import sleep , perf_counter
import sys

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    sleep(3)
    print(f"finishing {name}")

t1 = threading.Thread(target= show_number, args=("one",), daemon= True)
t2 = threading.Thread(target= show_number, args=("two",), daemon= True)


t1.start()
t2.start()



end = perf_counter()
print (end - start)
sys.exit("Done...")
```

Output



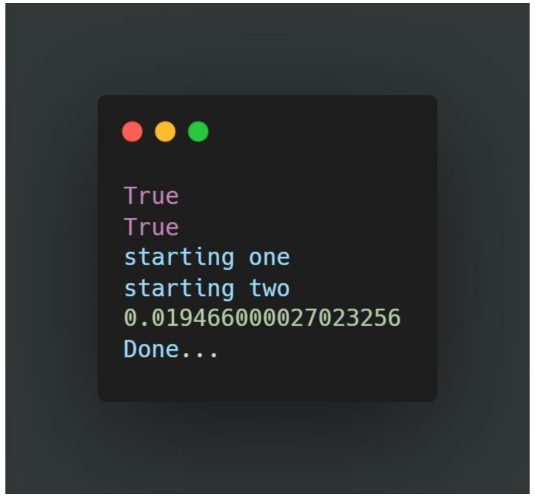
```
starting one
starting two
0.0010054000304080546
Done...
```

Example by setDaemon()



```python
import threading
from time import sleep , perf_counter
import sys

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    sleep(3)
    print(f"finishing {name}")

t1 = threading.Thread(target= show_number, args=("one",))
t2 = threading.Thread(target= show_number, args=("two",))


t1.setDaemon(True)
t2.setDaemon(True)

print(t1.isDaemon())
print(t2.isDaemon())


t1.start()
t2.start()



end = perf_counter()
print (end - start)
sys.exit("Done...")
```

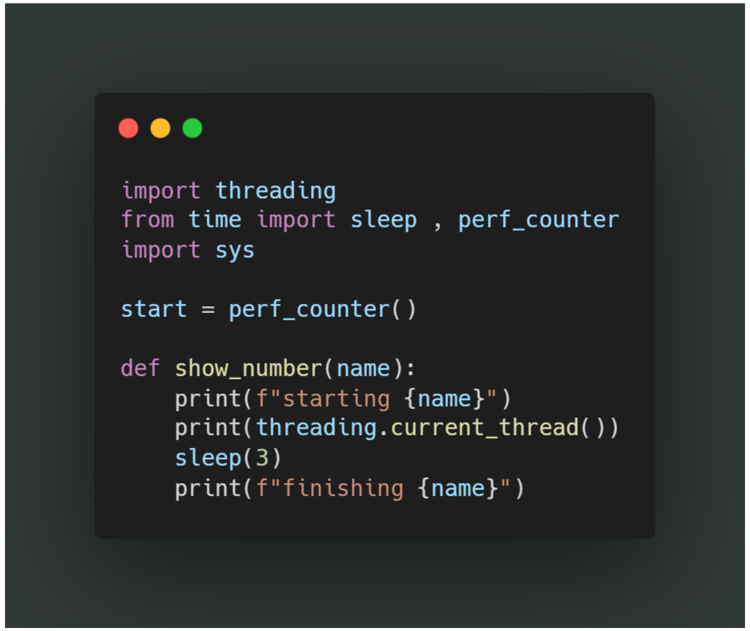
Output



```
True
True
starting one
starting two
0.019466000027023256
Done...
```

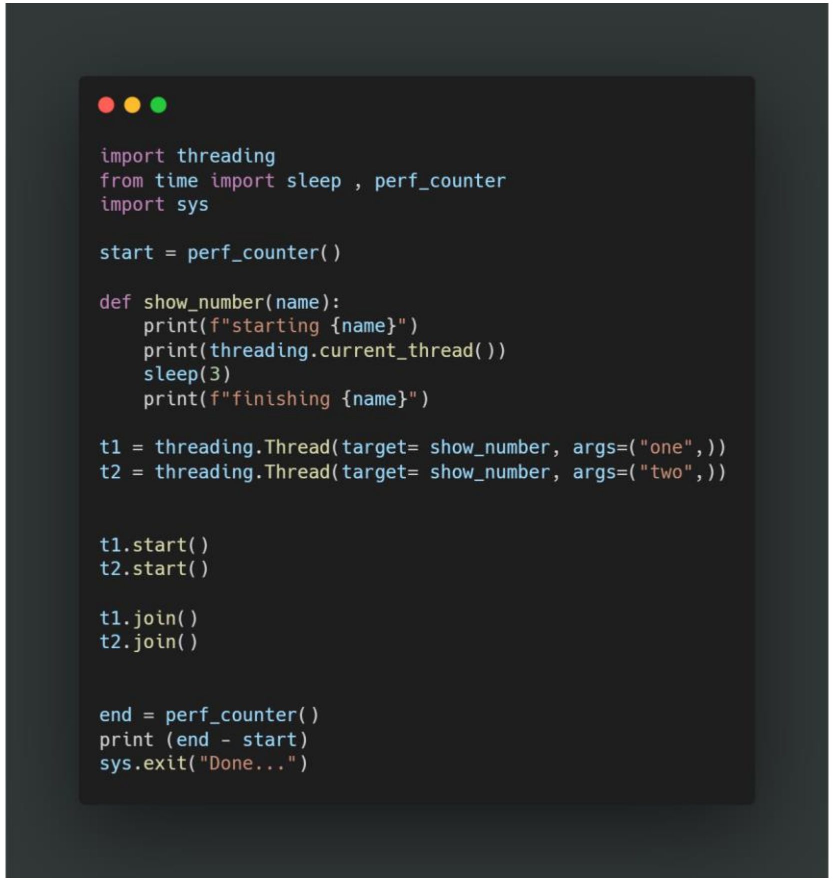
Current Thread

- Return the active Thread for us.
- If we want to use this method we need to call it like this:



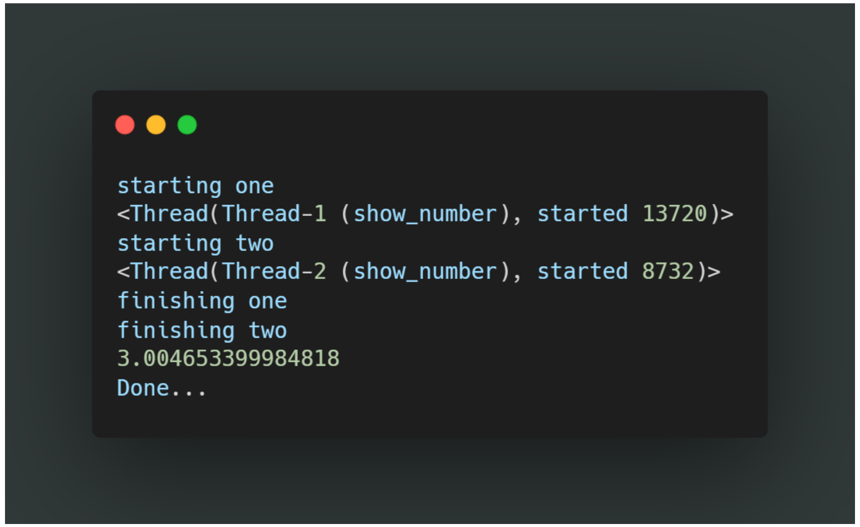
```
import threading
from time import sleep , perf_counter
import sys

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    print(threading.current_thread())
    sleep(3)
    print(f"finishing {name}")
```

Full example



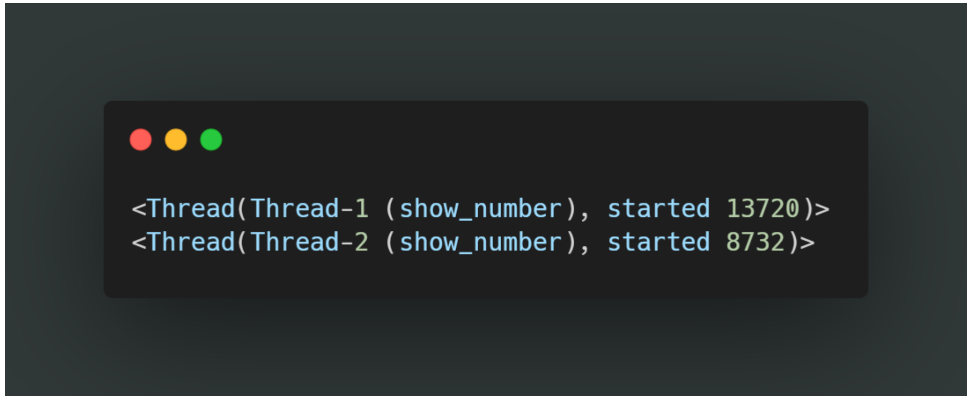
```
import threading
from time import sleep , perf_counter
import sys

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    print(threading.current_thread())
    sleep(3)
    print(f"finishing {name}")

t1 = threading.Thread(target= show_number, args=("one",))
t2 = threading.Thread(target= show_number, args=("two",))


t1.start()
t2.start()

t1.join()
t2.join()


end = perf_counter()
print (end - start)
sys.exit("Done...")
```

Output



```
starting one
<Thread(Thread-1 (show_number), started 13720)>
starting two
<Thread(Thread-2 (show_number), started 8732)>
finishing one
finishing two
3.004653399984818
Done...
```

Example explanation

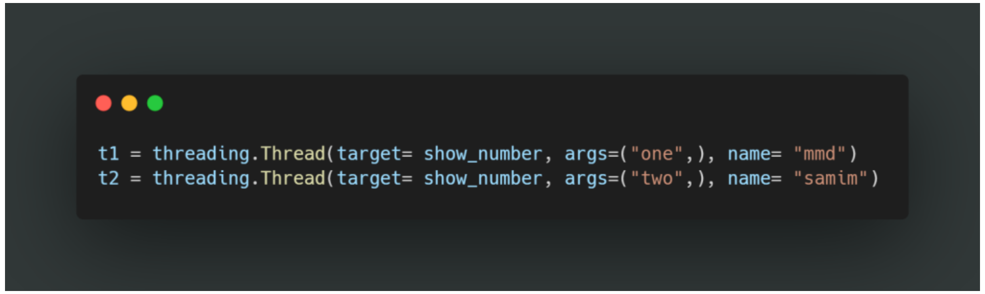


```
<Thread(Thread-1 (show_number), started 13720)>
<Thread(Thread-2 (show_number), started 8732)>
```

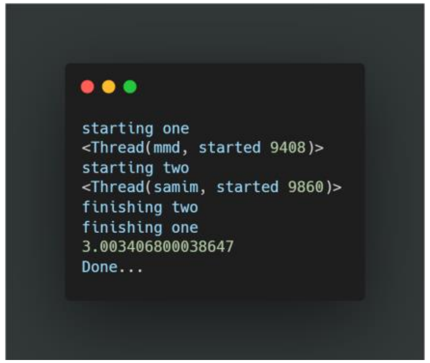
Current Thread has 2 important argument
1- name
2- ident

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

How to change name :



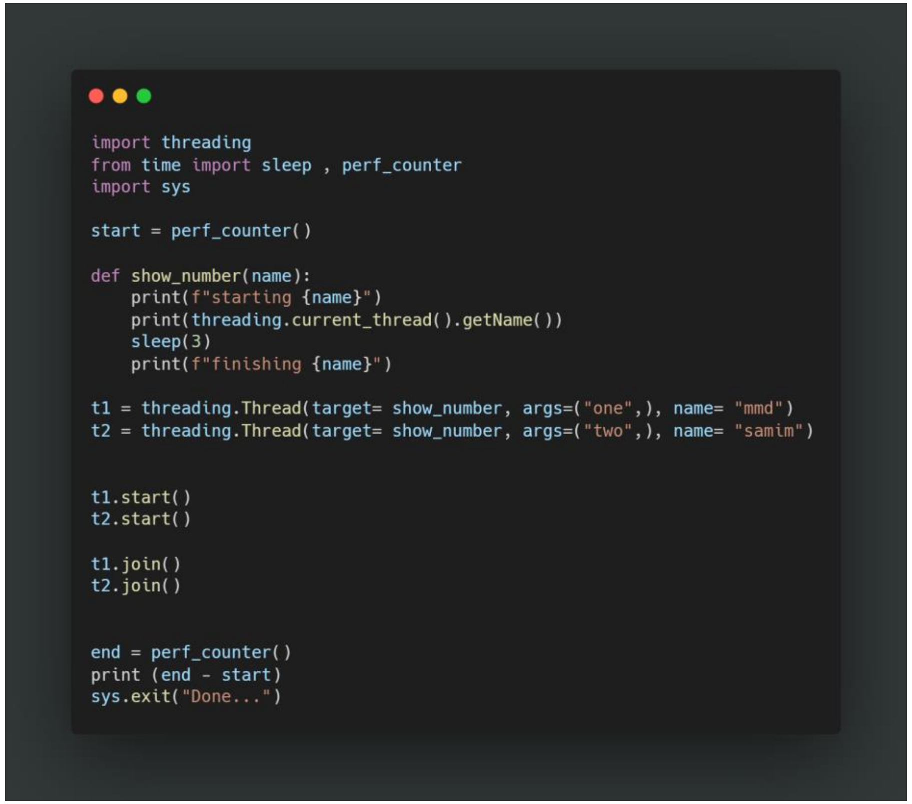
```
t1 = threading.Thread(target= show_number, args=("one",), name= "mmd")
t2 = threading.Thread(target= show_number, args=("two",), name= "samim")
```



```
starting one
<Thread(mmd, started 9408)>
starting two
<Thread(samim, started 9860)>
finishing two
finishing one
3.003406800038647
Done...
```

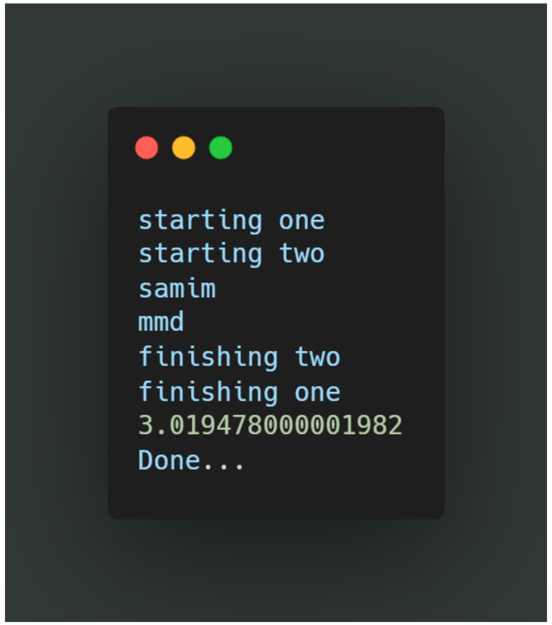
getName()

- If we don`t want to see ident we using getName()



```python
import threading
from time import sleep , perf_counter
import sys

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    print(threading.current_thread().getName())
    sleep(3)
    print(f"finishing {name}")

t1 = threading.Thread(target= show_number, args=("one",), name= "mmd")
t2 = threading.Thread(target= show_number, args=("two",), name= "samim")


t1.start()
t2.start()

t1.join()
t2.join()


end = perf_counter()
print (end - start)
sys.exit("Done...")
```

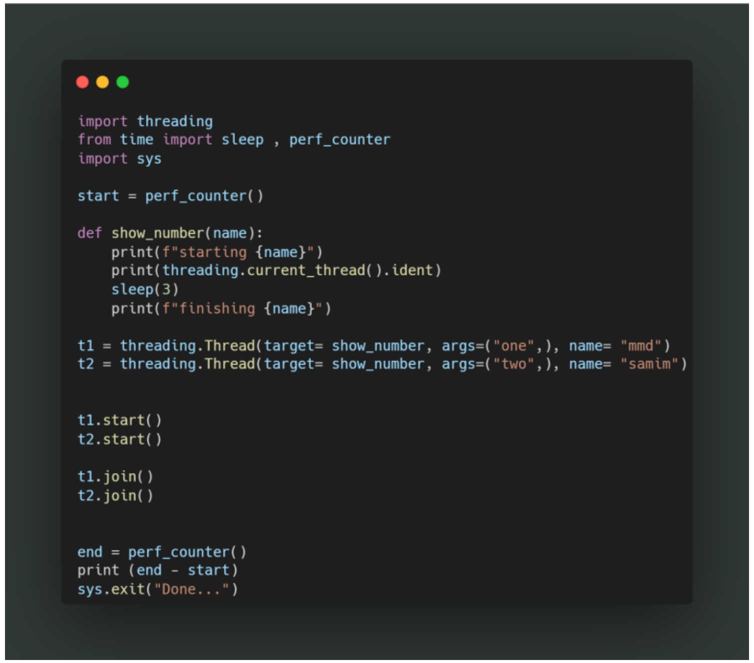
Output



```
starting one
starting two
samim
mmd
finishing two
finishing one
3.019478000001982
Done...
```

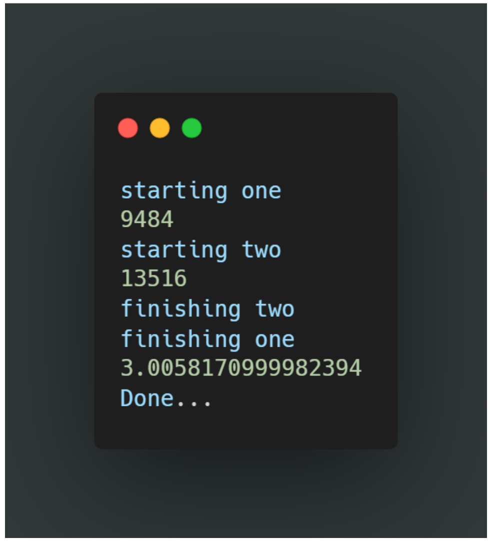
ident

- Return the ‘thread identifier’ of the current thread. This is a nonzero integer. Its value has no direct meaning; it is intended as a magic cookie to be used e.g. to index a dictionary of thread-specific data. Thread identifiers may be recycled when a thread exits and another thread is created.



```python
import threading
from time import sleep , perf_counter
import sys

start = perf_counter()

def show_number(name):
    print(f"starting {name}")
    print(threading.current_thread().ident)
    sleep(3)
    print(f"finishing {name}")

t1 = threading.Thread(target= show_number, args=("one",), name= "mmd")
t2 = threading.Thread(target= show_number, args=("two",), name= "samim")


t1.start()
t2.start()

t1.join()
t2.join()


end = perf_counter()
print (end - start)
sys.exit("Done...")
```

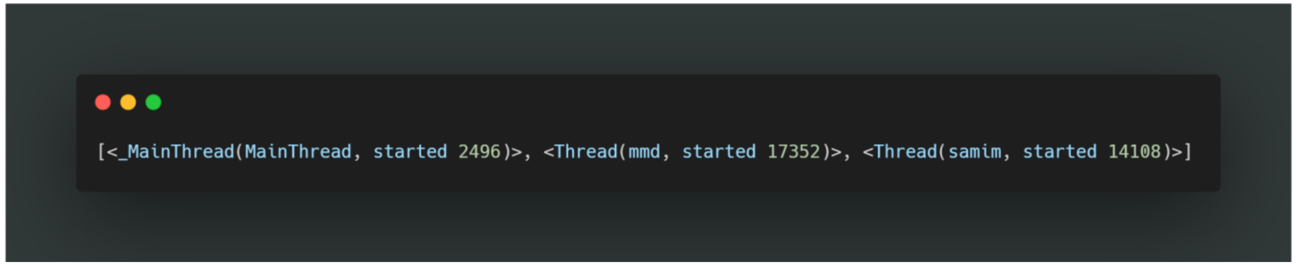
Output



```
starting one
9484
starting two
13516
finishing two
finishing one
3.0058170999982394
Done...
```

Enumerate()

- Return a list of all Thread objects currently active. The list includes daemonic threads and dummy thread objects created by current_thread(). It excludes terminated threads and threads that have not yet been started. However, the main thread is always part of the result, even when terminated



```
[<_MainThread(MainThread, started 2496)>, <Thread(mmd, started 17352)>, <Thread(samim, started 14108)>]
```

All the process is a thread too , the main thread is our process

Race Condition

- Race condition is like a bug it happens when 2 or more threads are working on one shared resource , it happened 1% but the Output could not be reliable
- How to fix?

By **Thread safe**, means that the thread have to respect the other threads. Ex. When Thread-1 is working, Thread-2 have to wait then start processing.

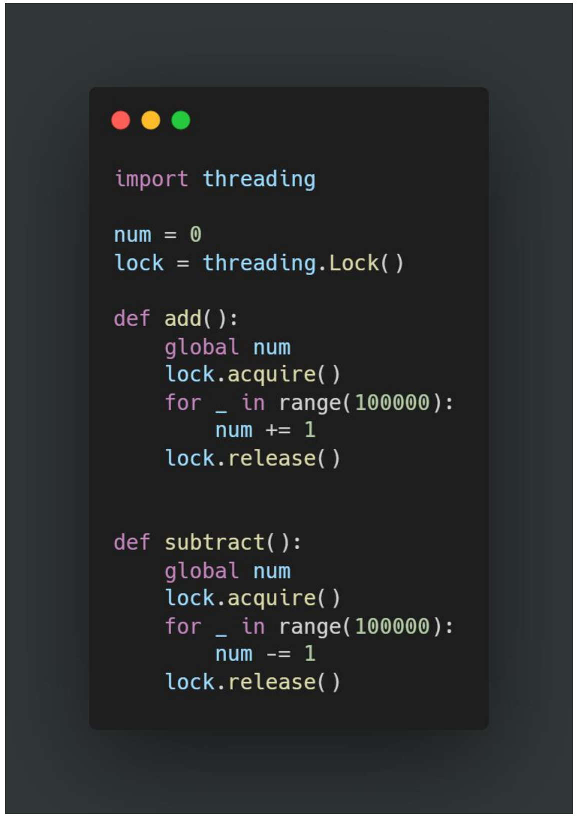
When we use Thread safe the program will be atomic

Atomic means : a process should have done 100% or shouldn`t run at the first

The Class that use for Thread safe is lock but we have issue that called dead lock , in this situation we forget to release() the shared resource and program won`t working

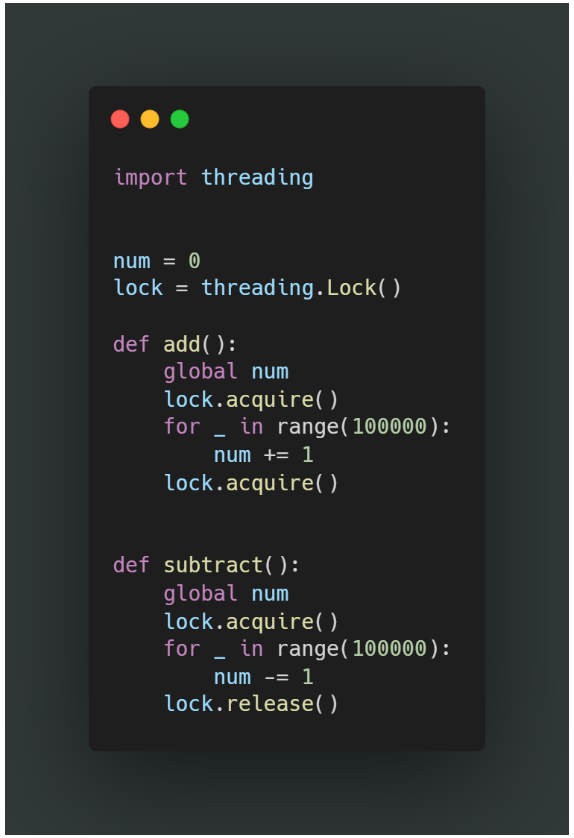
Thread safe , Deadlock

Thread safe



```python
import threading

num = 0
lock = threading.Lock()

def add():
    global num
    lock.acquire()
    for _ in range(100000):
        num += 1
    lock.release()


def subtract():
    global num
    lock.acquire()
    for _ in range(100000):
        num -= 1
    lock.release()
```

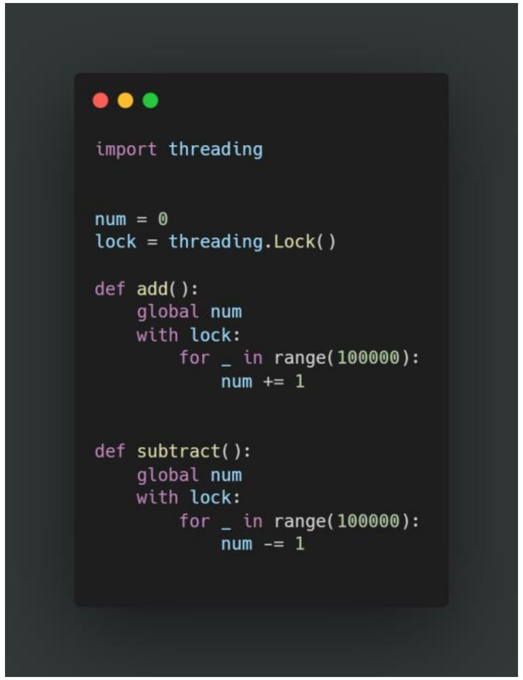
Deadlock



```python
import threading


num = 0
lock = threading.Lock()

def add():
    global num
    lock.acquire()
    for _ in range(100000):
        num += 1
    lock.acquire()


def subtract():
    global num
    lock.acquire()
    for _ in range(100000):
        num -= 1
    lock.release()
```

How to fix DeadLock?

- We can do Thread safe by another method, in this method we don`t have deadlock issue , Using Lock as a context manager.

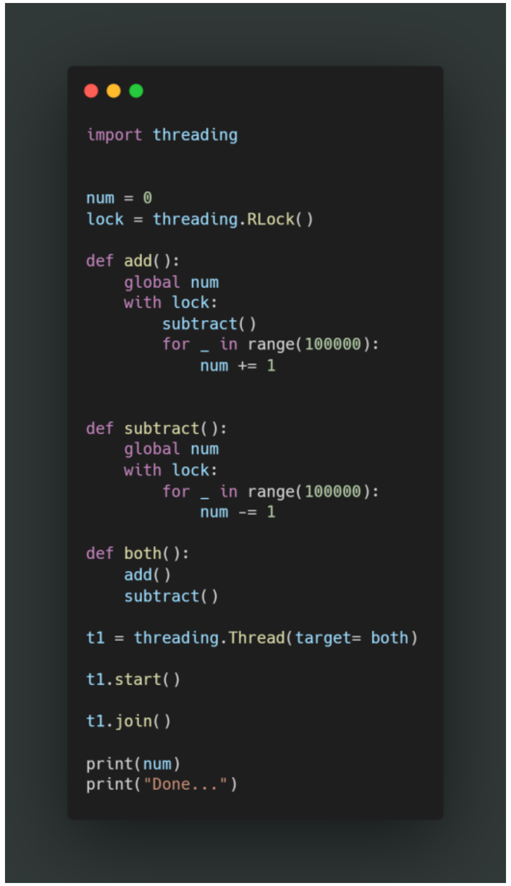


```
import threading


num = 0
lock = threading.Lock()

def add():
    global num
    with lock:
        for _ in range(100000):
            num += 1


def subtract():
    global num
    with lock:
        for _ in range(100000):
            num -= 1
```

Rlock

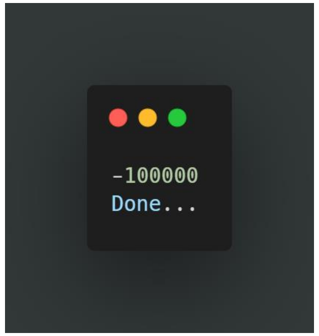
- When he have a returned program, have to use Rlock
- A reentrant lock is a synchronization primitive that may be acquired multiple times by the same thread. Internally, it uses the concepts of “owning thread” and “recursion level” in addition to the locked/unlocked state used by primitive locks. In the locked state, some thread owns the lock; in the unlocked state, no thread owns it.

Example



```python
import threading


num = 0
lock = threading.RLock()

def add():
    global num
    with lock:
        subtract()
        for _ in range(100000):
            num += 1


def subtract():
    global num
    with lock:
        for _ in range(100000):
            num -= 1

def both():
    add()
    subtract()

t1 = threading.Thread(target= both)

t1.start()

t1.join()

print(num)
print("Done...")
```

Output



```
-100000
Done...
```

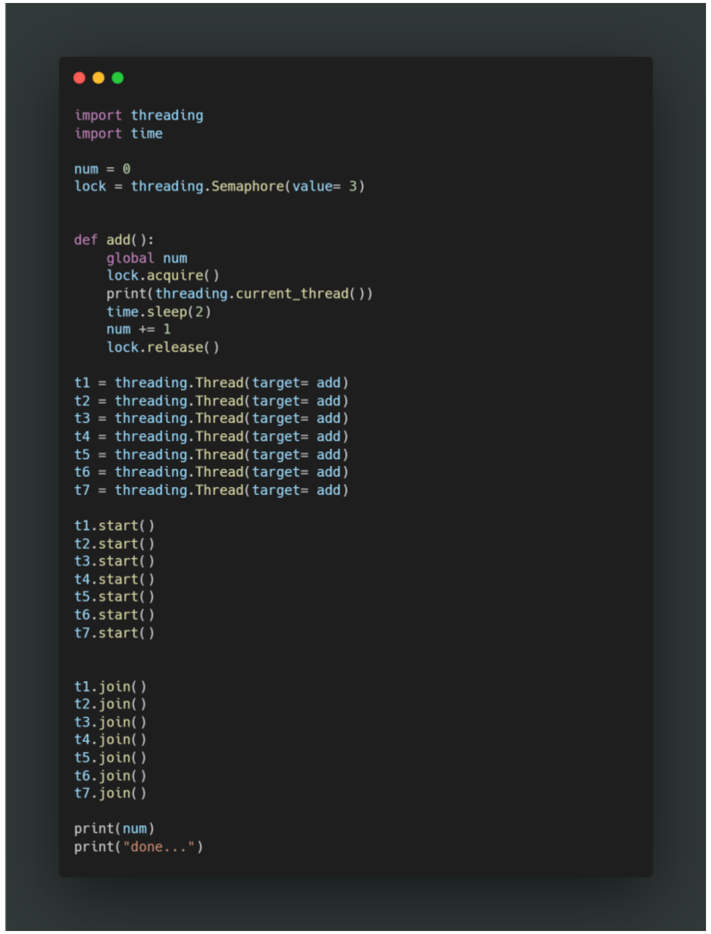
Semaphore

- It`s like look but if we want to connect into shared resource and we have limit for threads we use Semaphore
- **Methods:**
- acquire() , Release()
- General knowledge:

This is one of the oldest synchronization primitives in the history of computer science, invented by the early Dutch computer scientist Edsger W. Dijkstra (he used the names P() and V() instead of acquire() and release()).

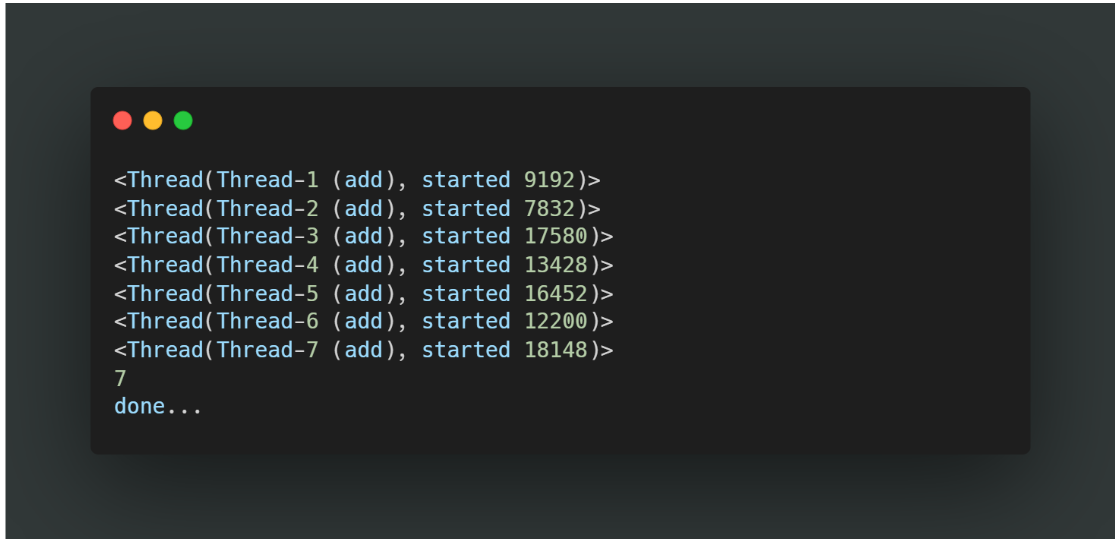
By value parameter we can identify that how many threads start toghether

Example



```python
import threading
import time

num = 0
lock = threading.Semaphore(value= 3)


def add():
    global num
    lock.acquire()
    print(threading.current_thread())
    time.sleep(2)
    num += 1
    lock.release()

t1 = threading.Thread(target= add)
t2 = threading.Thread(target= add)
t3 = threading.Thread(target= add)
t4 = threading.Thread(target= add)
t5 = threading.Thread(target= add)
t6 = threading.Thread(target= add)
t7 = threading.Thread(target= add)

t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
t6.start()
t7.start()


t1.join()
t2.join()
t3.join()
t4.join()
t5.join()
t6.join()
t7.join()

print(num)
print("done...")
```

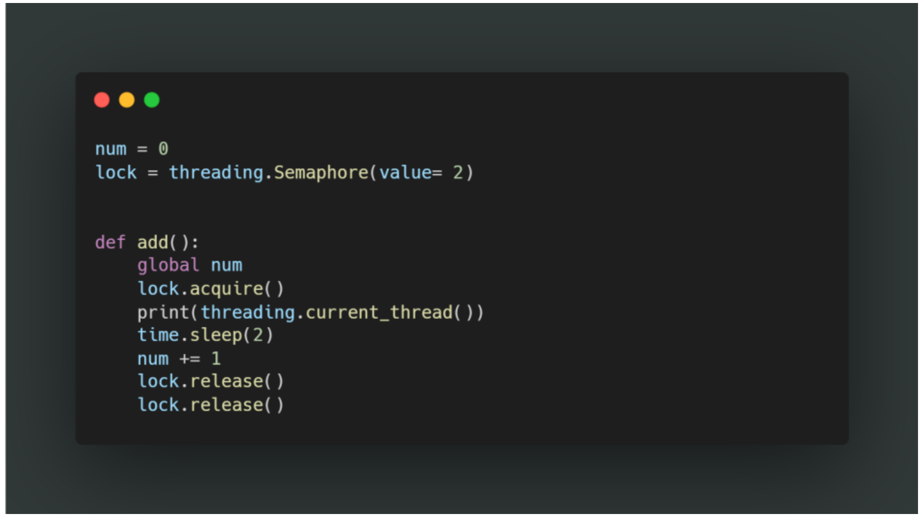
Output



```
<Thread(Thread-1 (add), started 9192)>
<Thread(Thread-2 (add), started 7832)>
<Thread(Thread-3 (add), started 17580)>
<Thread(Thread-4 (add), started 13428)>
<Thread(Thread-5 (add), started 16452)>
<Thread(Thread-6 (add), started 12200)>
<Thread(Thread-7 (add), started 18148)>
7
done...
```

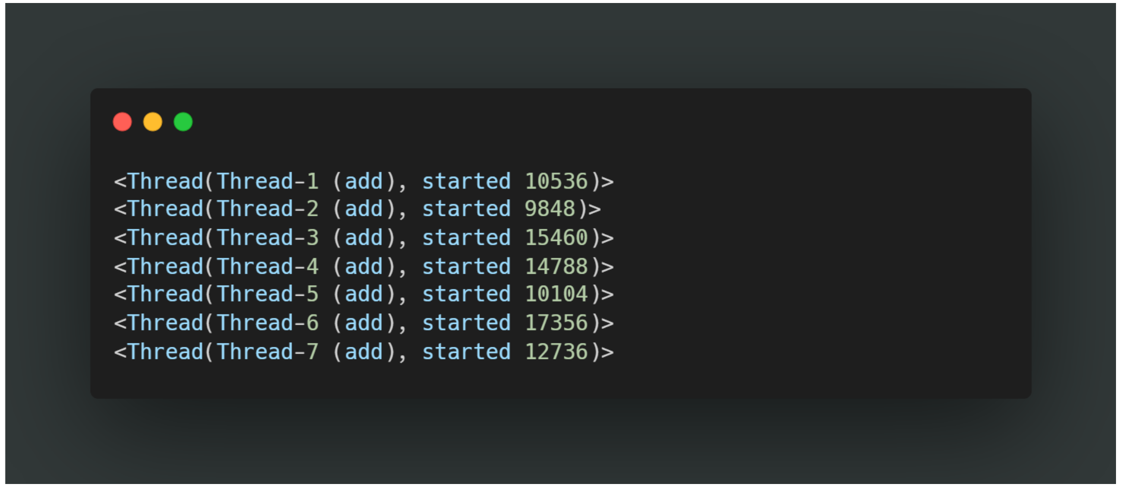
Semaphore vs Bounded semaphore

- In semaphore there is a issue with release() , if we call release 2 times , semaphore doing 2 value process then , value equals to -2 and semaphore open 4 values spots



```
num = 0
lock = threading.Semaphore(value= 2)


def add():
    global num
    lock.acquire()
    print(threading.current_thread())
    time.sleep(2)
    num += 1
    lock.release()
    lock.release()
```

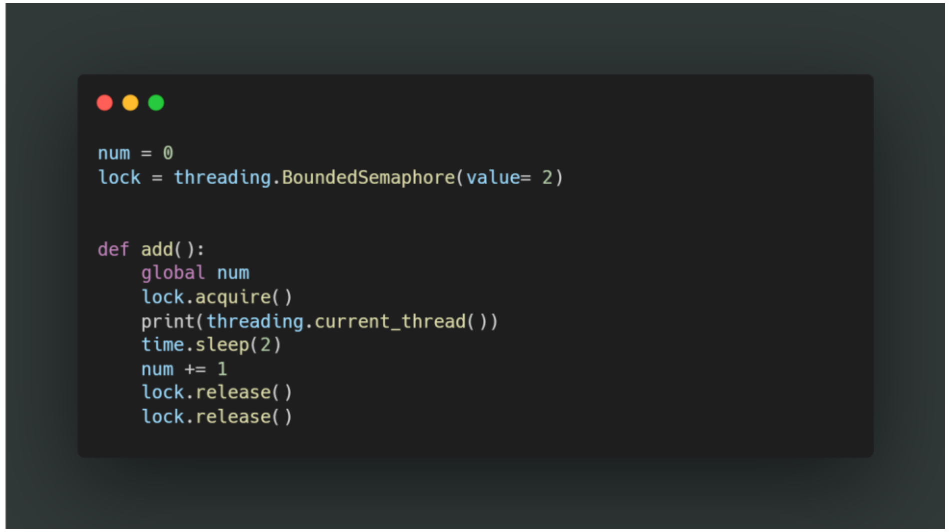
Output



```
<Thread(Thread-1 (add), started 10536)>
<Thread(Thread-2 (add), started 9848)>
<Thread(Thread-3 (add), started 15460)>
<Thread(Thread-4 (add), started 14788)>
<Thread(Thread-5 (add), started 10104)>
<Thread(Thread-6 (add), started 17356)>
<Thread(Thread-7 (add), started 12736)>
```

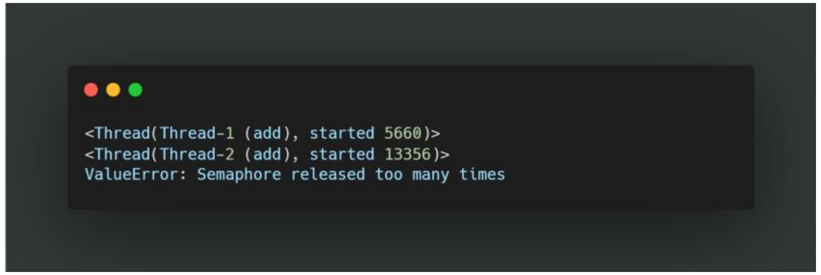
Semaphore vs BoundedSemaphore

- BoundedSemaphore fix this issue by ValueError for release times.



```
num = 0
lock = threading.BoundedSemaphore(value= 2)


def add():
    global num
    lock.acquire()
    print(threading.current_thread())
    time.sleep(2)
    num += 1
    lock.release()
    lock.release()
```

Output



```
<Thread(Thread-1 (add), started 5660)>
<Thread(Thread-2 (add), started 13356)>
ValueError: Semaphore released too many times
```

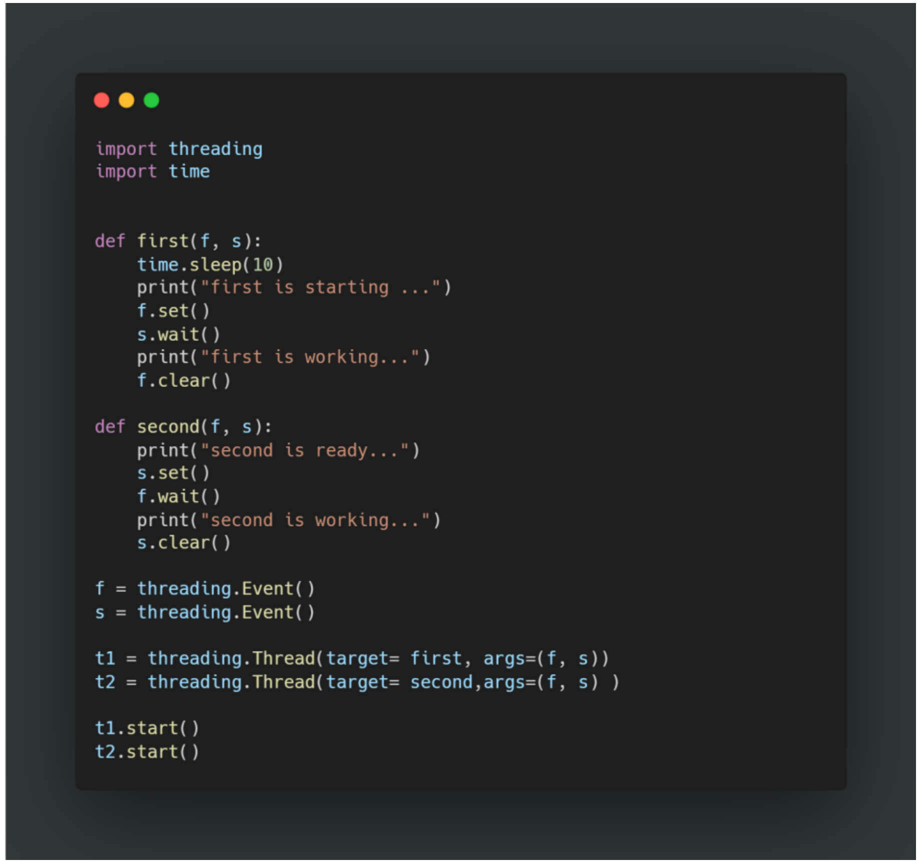
Also you can use semaphore and boundedsemaphore as context manager

Event

- This is one of the simplest mechanisms for communication between threads: one thread signals an event and other threads wait for it.

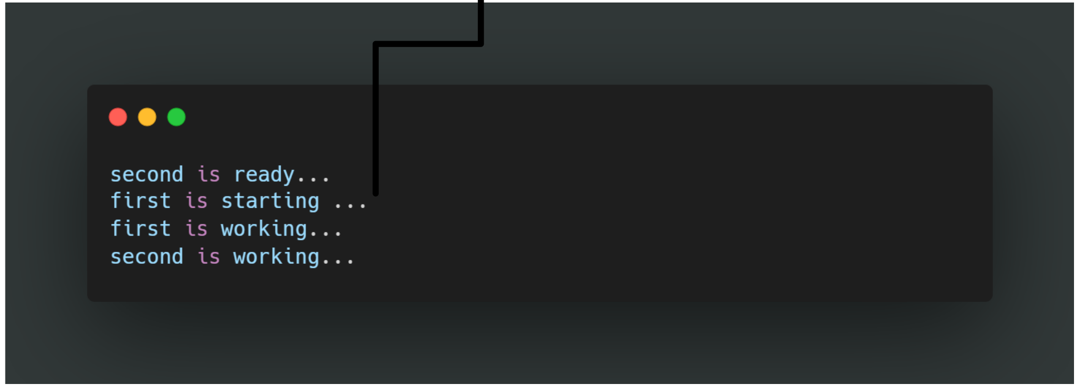
- An event object manages an internal flag that can be set to true with the set() method and reset to false with the clear() method. The wait() method blocks until the flag is true.

Example



```
import threading
import time


def first(f, s):
    time.sleep(10)
    print("first is starting ...")
    f.set()
    s.wait()
    print("first is working...")
    f.clear()

def second(f, s):
    print("second is ready...")
    s.set()
    f.wait()
    print("second is working...")
    s.clear()

f = threading.Event()
s = threading.Event()

t1 = threading.Thread(target= first, args=(f, s))
t2 = threading.Thread(target= second,args=(f, s) )

t1.start()
t2.start()
```

Output



```
second is ready...
first is starting ...
first is working...
second is working...
```

Wait 10 sec to first join

Frequently asked questions

- Why we code set() and wait() on 2 methods ?

Because we don`t know which one will start first

What does set and wait means?

We set() that the thread is ready and these 2 threads have to wait() for each other

When we use Event() Class don`t need to use join() method

For learning more

- If you want to know more about Multi threading and methods you can study about : timer() , object condition() , barrier object()

These topics are 10% of multi threading and the introduced topics in this lecture are 90% of multi threading

Thanks for you attention and your Time.

Good luck