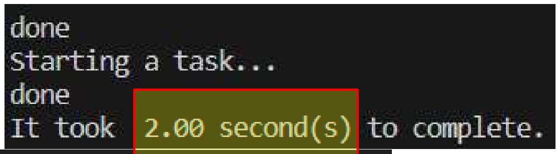
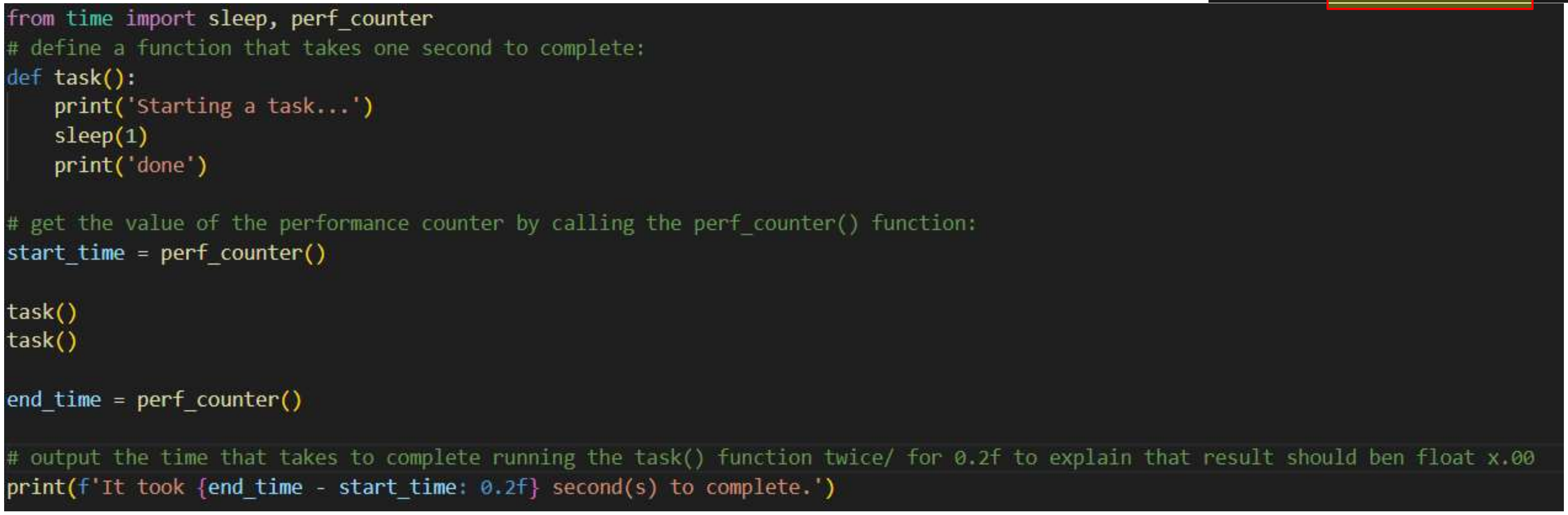
PYTHON MULTITHREADING

Guizani Ahmed

Single-threaded applications



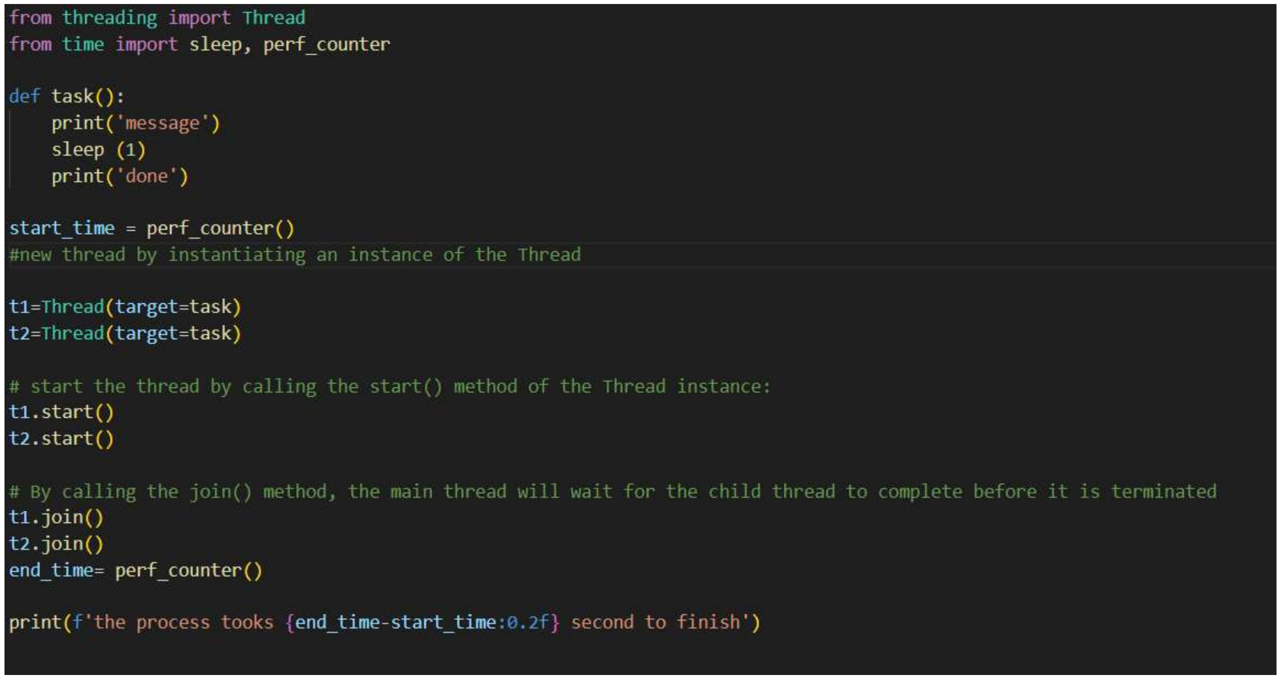
```
done
Starting a task...
done
It took  2.00 second(s) to complete.
```



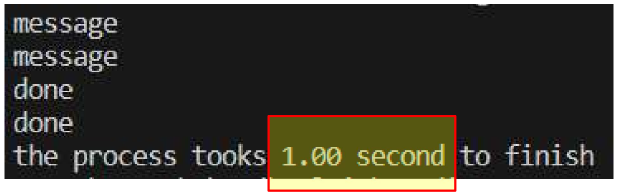
```python
from time import sleep, perf_counter
# define a function that takes one second to complete:
def task():
    print('Starting a task...')
    sleep(1)
    print('done')

# get the value of the performance counter by calling the perf_counter() function:
start_time = perf_counter()

task()
task()

end_time = perf_counter()

# output the time that takes to complete running the task() function twice/ for 0.2f to explain that result should ben float x.00
print(f'It took {end_time - start_time: 0.2f} second(s) to complete.')
```

- First, the task() function executes and sleeps for one second. Then it executes the second time and also sleeps for another second. Finally, the program completes.
- When the task() function **calls** the **sleep() function**, the **CPU doesn't do anything**, which is **not efficient** in terms of resource utilization.
- This program has **one process with a single thread**, which is called the **main thread**. Because the program has only one thread, it's called a **single-threaded program.**

Using Python threading to develop a multi-threaded program example



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

def task():
    print('message')
    sleep (1)
    print('done')

start_time = perf_counter()
#new thread by instantiating an instance of the Thread

t1=Thread(target=task)
t2=Thread(target=task)

# start the thread by calling the start() method of the Thread instance:
t1.start()
t2.start()

# By calling the join() method, the main thread will wait for the child thread to complete before it is terminated
t1.join()
t2.join()
end_time= perf_counter()

print(f'the process tooks {end_time-start_time:0.2f} second to finish')
```



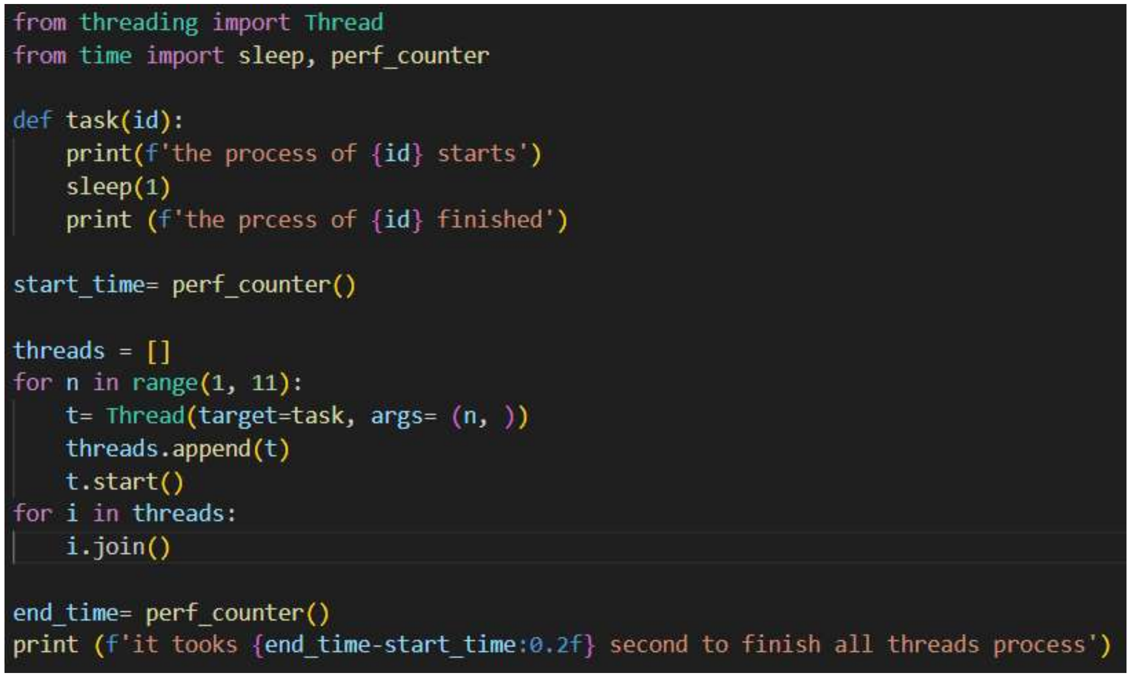
```
message
message
done
done
the process tooks 1.00 second to finish
```

•new_thread = Thread(target=fn,args=args_tuple)

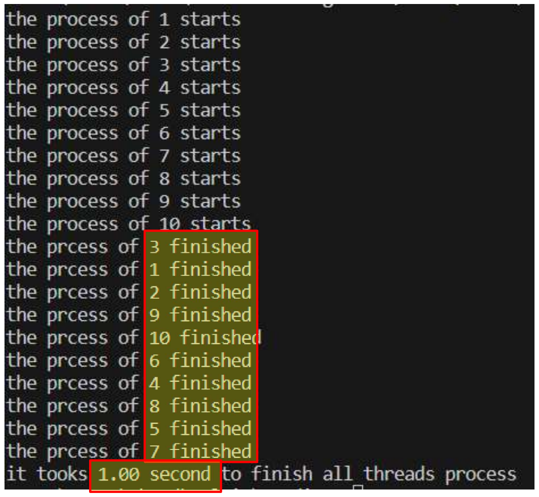
The Thread() accepts many parameters. The main ones are:

•target: specifies a function (fn) to run in the new thread.
•args: specifies the arguments of the function (fn). The args argument is a tuple.

Create 10 new threads and pass an id to each



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

def task(id):
    print(f'the process of {id} starts')
    sleep(1)
    print (f'the prcess of {id} finished')

start_time= perf_counter()

threads = []
for n in range(1, 11):
    t= Thread(target=task, args= (n, ))
    threads.append(t)
    t.start()
for i in threads:
    i.join()

end_time= perf_counter()
print (f'it tooks {end_time-start_time:0.2f} second to finish all threads process')
```



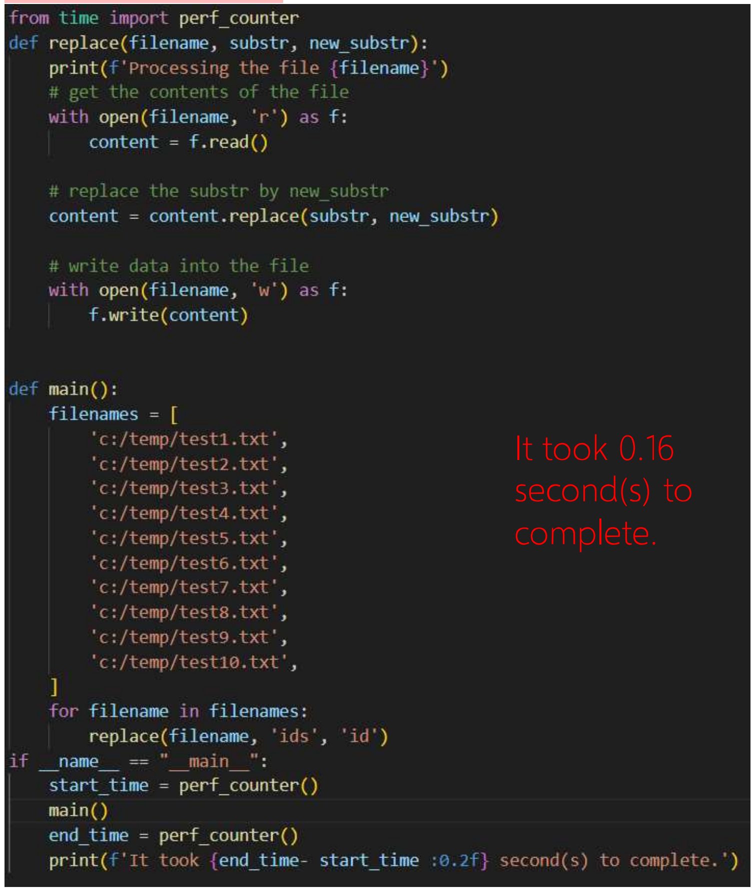
```
the process of 1 starts
the process of 2 starts
the process of 3 starts
the process of 4 starts
the process of 5 starts
the process of 6 starts
the process of 7 starts
the process of 8 starts
the process of 9 starts
the process of 10 starts
the prcess of 3 finished
the prcess of 1 finished
the prcess of 2 finished
the prcess of 9 finished
the prcess of 10 finished
the prcess of 6 finished
the prcess of 4 finished
the prcess of 8 finished
the prcess of 5 finished
the prcess of 7 finished
it tooks 1.00 second to finish all threads process
```

- Not in order
- Take 1 second with multithreading

Notice:
- If the join() method called inside the loop of start thread, the program will wait for the first thread to complete before starting the next one.

Changing a name in a file

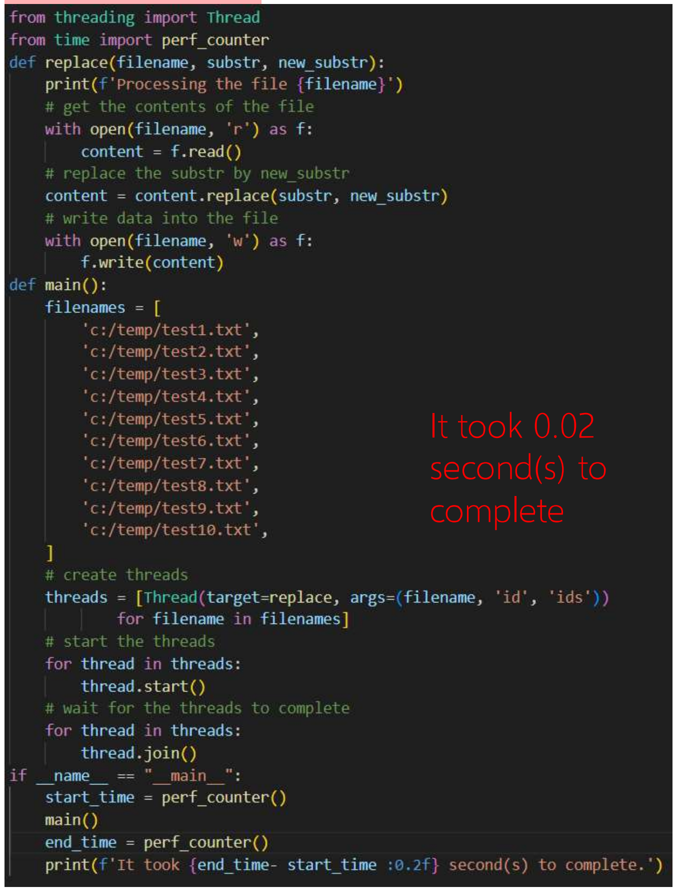
- Without threading



```python
from time import perf_counter
def replace(filename, substr, new_substr):
    print(f'Processing the file {filename}')
    # get the contents of the file
    with open(filename, 'r') as f:
        content = f.read()

    # replace the substr by new_substr
    content = content.replace(substr, new_substr)

    # write data into the file
    with open(filename, 'w') as f:
        f.write(content)


def main():
    filenames = [
        'c:/temp/test1.txt',
        'c:/temp/test2.txt',
        'c:/temp/test3.txt',
        'c:/temp/test4.txt',
        'c:/temp/test5.txt',
        'c:/temp/test6.txt',
        'c:/temp/test7.txt',
        'c:/temp/test8.txt',
        'c:/temp/test9.txt',
        'c:/temp/test10.txt',
    ]
    for filename in filenames:
        replace(filename, 'ids', 'id')
if __name__ == "__main__":
    start_time = perf_counter()
    main()
    end_time = perf_counter()
    print(f'It took {end_time- start_time :0.2f} second(s) to complete.')
```

It took 0.16 second(s) to complete.

- With threading



```python
from threading import Thread
from time import perf_counter
def replace(filename, substr, new_substr):
    print(f'Processing the file {filename}')
    # get the contents of the file
    with open(filename, 'r') as f:
        content = f.read()
    # replace the substr by new_substr
    content = content.replace(substr, new_substr)
    # write data into the file
    with open(filename, 'w') as f:
        f.write(content)
def main():
    filenames = [
        'c:/temp/test1.txt',
        'c:/temp/test2.txt',
        'c:/temp/test3.txt',
        'c:/temp/test4.txt',
        'c:/temp/test5.txt',
        'c:/temp/test6.txt',
        'c:/temp/test7.txt',
        'c:/temp/test8.txt',
        'c:/temp/test9.txt',
        'c:/temp/test10.txt',
    ]
    # create threads
    threads = [Thread(target=replace, args=(filename, 'id', 'ids'))
            for filename in filenames]
    # start the threads
    for thread in threads:
        thread.start()
    # wait for the threads to complete
    for thread in threads:
        thread.join()
if __name__ == "__main__":
    start_time = perf_counter()
    main()
    end_time = perf_counter()
    print(f'It took {end_time- start_time :0.2f} second(s) to complete.')
```

It took 0.02 second(s) to complete

WWW.PYTHONTUTORIAL.NET/PYTHON-CONCURRENCY/PYTHON-THREADING/