Multithreading: For a given list of numbers print square and cube of every numbers

For example:

Input: [2,3,8,9]

Output: square list – [4,9,64,81]

Cube list: [8,27,512,729]

import time

def calc\_squares(numbers):

print("calculate square of numbers")

for n in numbers:

time.sleep(0.2)

print("squares:", n\*n)

def calc\_cubes(numbers):

print("calculate cube of numbers")

for n in numbers:

time.sleep(0.2)

print("cubes:", n\*n\*n)

arr= [2,3,8,9]

t=time.time()

calc\_squares(arr)

calc\_cubes(arr)

print("done in:",time.time()-t)

print("Hah... I am done with all my work now!")

Result: calculate square of numbers

squares: 4

squares: 9

squares: 64

squares: 81

calculate cube of numbers

cubes: 8

cubes: 27

cubes: 512

cubes: 729

done in: 1.7035505771636963

Hah... I am done with all my work now!

Multiprocess: Create two processes,

1. First is to calculate square of all the numbers
2. Second one is to calculate cube of numbers

import time

import multiprocessing

def calc\_square(numbers):

for n in numbers:

print('square', n\*n)

def calc\_cube(numbers):

for n in numbers:

print('cube' ,str(n\*n\*n))

if \_\_name\_\_ == "\_\_main\_\_":

arr = [2,3,8,9]

p1 = multiprocessing.Process(target=calc\_square, args=(arr,))

p2 = multiprocessing.Process(target=calc\_cube, args=(arr,))

p1.start()

p2.start()

p1.join()

p2.join()

print("Done!")

The Threading Module:

The threading module exposes all the modules of the thread module and provides some addition information

1. Threading.activeCount() : Returns the number of thread objects that are active.
2. Threading.currentThread(): Returns the number of thread objects in the caller’s thread control
3. Threading.enumerate(): Returns a list of all the thread objects that are currently active.

In addition to the methods the threading module has the Thread class that implements threading. The methods provided by the thread class are as follows.

1. Run(): The run() method is the entry point for a thread.
2. Start(): The start() method starts a thread by calling the run method.
3. Join([time]): The join() waits for threads to terminate.
4. IsAlive(): The isAlive() method checks whether a thread is still executing.
5. getName(): The getName() method returns the name of a thread.
6. setName(): The setName() method sets the name of a thread.

Creating Thread Using Threading Module:

To implement a new thread using the threading module, you have to do the following

* Define a new subclass of the Thread class.
* Override the \_\_int\_\_(self[,args]) method to add additional arguments.
* Then, override the run(self[,args]) method to implement what the thread should do when started.

Once you have created the new Thread subclass, you can create an instance of it and then start a new thread by invoking the start(), which in turn calls the run() method.

import threading

import time

exitFlag = 0

class myThread(threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print("Starting" + self.name)

print\_time(self.name, self.counter, 5)

print("Exiting" + self.name)

def print\_time(threadName,delay,counter):

while counter:

if exitFlag:

threadName.exit()

time.sleep(delay)

print("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

thread1.start()

thread2.start()

thread1.join()

thread2.join()

print("Existing Main Thread")

Synchronizing Threads : The threading module provided with Python includes a simple to implement locking mechanism that allows you to synchronize threads A new lock is created by calling the Lock() method, which returns the new lock.

The acquire(blocking) method of the new lock object is used to force the threads to run synchronously The optional blocking parameters enables you to control whether the thread waits to acquire the lock.

If blocking is set t 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock is acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released

The release() method of the new lock object is used to release the lock when it is no longer required.

import threading

import time

class myThread(threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print("Starting" + self.name)

threadLock.acquire()

print\_time(self.name, self.counter, 3)

threadLock.release()

def print\_time(threadName, delay, counter):

while counter:

time.sleep(delay)

print("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

threadLock = threading.Lock()

threads = []

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

thread1.start()

thread2.start()

thread1.appened(thread1)

thread2.appened(thread2)

for t in threads:

t.join()

print("Exiting Main Thread")

Multithreaded Priority Queue: The Queue module allows you to create a new queue object that can hold a specific number of items. There are following methods to control the Queue.

* Get(): The get() removes and returns an item from the queue
* Put(): The put adds item to a queue.
* Qsize(): The qsize() returns the number of items that are currently in the queue
* Empty(): The empty() returns True if queue is empty; otherwise, False.
* Full(): the full() returns True if queue is full; otherwise, False.

"""import time

def calc\_squares(numbers):

print("calculate square of numbers")

for n in numbers:

time.sleep(0.2)

print("squares:", n\*n)

def calc\_cubes(numbers):

print("calculate cube of numbers")

for n in numbers:

time.sleep(0.2)

print("cubes:", n\*n\*n)

arr= [2,3,8,9]

t=time.time()

calc\_squares(arr)

calc\_cubes(arr)

print("done in:",time.time()-t)

print("Hah... I am done with all my work now!")"""

"""import threading

import time

exitFlag = 0

class myThread(threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print("Starting" + self.name)

print\_time(self.name, self.counter, 5)

print("Exiting" + self.name)

def print\_time(threadName,delay,counter):

while counter:

if exitFlag:

threadName.exit()

time.sleep(delay)

print("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

thread1.start()

thread2.start()

thread1.join()

thread2.join()

print("Existing Main Thread")"""

"""import threading

import time

class myThread(threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print("Starting" + self.name)

threadLock.acquire()

print\_time(self.name, self.counter, 3)

threadLock.release()

def print\_time(threadName, delay, counter):

while counter:

time.sleep(delay)

print("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

threadLock = threading.Lock()

threads = []

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

thread1.start()

thread2.start()

thread1.appened(thread1)

thread2.appened(thread2)

for t in threads:

t.join()

print("Exiting Main Thread")"""

class myThread(threading.Thread):

def \_\_init\_\_(self, threadID, name,q):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.q = q

def run(self):

print("Starting" + self.name)

process\_data(self.name, self.q)

print("Exiting" + self.name)

def process\_data(threadName,q):

while not exitFlag:

queueLock.acquire()

if not workQueue.empty():

data = q.get()

queueLock.release()

print("%s processing %s" % (threadName,ArithmeticError data))

else:

queueLock.release()

time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]

namelist = ["One", "Two", "Three", "Four", "Five"]

queueLock = threading.Lock()

workQueue = queue.Queue(10)

threads = []

threadID = 1

for tName in threadList:

thread = myThread(threadID, tName, workQueue)

thread.start()

threads.append(thread)

threadID += 1

queueLock.acquire()

for word in nameList:

workQueue.put(word)

queueLock.release()

while not workQueue.empty():

pass

exitFlag = 1

for t in threads:

t.join()

print ("Exiting Main Thread")