Timer

This lesson discusses the Timer class with examples.

Timer

The Timer object allows execution of a callable object after a certain amount of time has elapsed. Consider the snippet below where a **`Timer`** object executes the method **`say_hi()`** in a different thread than the main thread. The thread names can be observed from the output.

Creating a timer

```
from threading import Timer
from threading import current_thread
import time


def say_hi():
    print("{0} says Hi!".format(current_thread().getName()))


timer = Timer(1, say_hi)
timer.start()

time.sleep(2)

print("{0} exiting".format(current_thread().getName()))
```

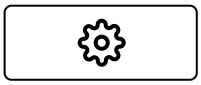
The timer constructor takes in a floating point number representing the seconds that need to elapse before the task is executed. Though when the task is executed may not be exactly after the passed in number of seconds. It may be slightly more than the interval passed in.

Time is a subclass of the **`Thread`** class and similarly accepts arguments as a list or a keyword dictionary.

A timer object can also be cancelled using the **`cancel()`** method before the task has been executed.

```
from threading import Timer
from threading import current_thread
import time


def say_hi(name):
    print("{0} says Hi {1}!".format(current_thread().getName(), name))


timer = Timer(1, say_hi, args=["reader"])
timer.start()

time.sleep(2)

print("{0} exiting".format(current_thread().getName()))
```

As an exercise, comment out **line#13** in the snippet above and re-run the program. The order of print statements will be reversed. Realize that if the timer object instantiated a daemon thread, it would have been killed when the main thread exited.



Back

Next

Events

Barrier

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/timer__threading-module__python-concurrency-for-senior-engineering-interviews)