

sched — Timed Event Scheduler

Purpose: Generic event scheduler.

The sched module implements a generic event scheduler for running tasks at specific times. The scheduler class uses a `time` function to learn the current time, and a delay function to wait for a specific period of time. The actual units of time are not important, which makes the interface flexible enough to be used for many purposes.

The time function is called without any arguments, and should return a number representing the current time. The delay function is called with a single integer argument, using the same scale as the time function, and should wait that many time units before returning. By default `monotonic()` and `sleep()` from [time](#) are used, but the examples in this section use `time.time()`, which also meets the requirements, because it makes the output easier to understand.

To support multi-threaded applications, the delay function is called with argument 0 after each event is generated, to ensure that other threads also have a chance to run.

Running Events With a Delay

Events can be scheduled to run after a delay, or at a specific time. To schedule them with a delay, use the `enter()` method, which takes four arguments.

- A number representing the delay
- A priority value
- The function to call
- A tuple of arguments for the function

This example schedules two different events to run after two and three seconds respectively. When the event's time comes up, `print_event()` is called and prints the current time and the name argument passed to the event.

```
# sched_basic.py

import sched
import time

scheduler = sched.scheduler(time.time, time.sleep)

def print_event(name, start):
    now = time.time()
    elapsed = int(now - start)
    print('EVENT: {} elapsed={} name={}'.format(
        time.ctime(now), elapsed, name))

start = time.time()
print('START:', time.ctime(start))
scheduler.enter(2, 1, print_event, ('first', start))
scheduler.enter(3, 1, print_event, ('second', start))

scheduler.run()
```

Running the program produces:

```
$ python3 sched_basic.py

START: Sun Sep  4 16:21:01 2016
EVENT: Sun Sep  4 16:21:03 2016 elapsed=2 name=first
EVENT: Sun Sep  4 16:21:04 2016 elapsed=3 name=second
```

The time printed for the first event is two seconds after start, and the time for the second event is three seconds after start.

Overlapping Events

The call to `run()` blocks until all of the events have been processed. Each event is run in the same thread, so if an event takes longer to run than the delay between events, there will be overlap. The overlap is resolved by postponing the later event. No events are lost, but some events may be called later than they were scheduled. In the next example, `long_event()` sleeps but it could just as easily delay by performing a long calculation or by blocking on I/O.

```
# sched_overlap.py

import sched
import time

scheduler = sched.scheduler(time.time, time.sleep)

def long_event(name):
    print('BEGIN EVENT :', time.ctime(time.time()), name)
    time.sleep(2)
    print('FINISH EVENT:', time.ctime(time.time()), name)

print('START:', time.ctime(time.time()))
scheduler.enter(2, 1, long_event, ('first',))
scheduler.enter(3, 1, long_event, ('second',))

scheduler.run()
```

The result is the second event is run immediately after the first finishes, since the first event took long enough to push the clock past the desired start time of the second event.

```
$ python3 sched_overlap.py

START: Sun Sep  4 16:21:04 2016
BEGIN EVENT : Sun Sep  4 16:21:06 2016 first
FINISH EVENT: Sun Sep  4 16:21:08 2016 first
BEGIN EVENT : Sun Sep  4 16:21:08 2016 second
FINISH EVENT: Sun Sep  4 16:21:10 2016 second
```

Event Priorities

If more than one event is scheduled for the same time their priority values are used to determine the order they are run.

```
# sched_priority.py

import sched
import time

scheduler = sched.scheduler(time.time, time.sleep)

def print_event(name):
    print('EVENT:', time.ctime(time.time()), name)

now = time.time()
print('START:', time.ctime(now))
scheduler.enterabs(now + 2, 2, print_event, ('first',))
scheduler.enterabs(now + 2, 1, print_event, ('second',))

scheduler.run()
```

This example needs to ensure that they are scheduled for the exact same time, so the `enterabs()` method is used instead of `enter()`. The first argument to `enterabs()` is the time to run the event, instead of the amount of time to delay.

```
$ python3 sched_priority.py

START: Sun Sep  4 16:21:10 2016
EVENT: Sun Sep  4 16:21:12 2016 second
EVENT: Sun Sep  4 16:21:12 2016 first
```

Canceling Events

Both `enter()` and `enterabs()` return a reference to the event that can be used to cancel it later. Because `run()` blocks, the event has to be canceled in a different thread. For this example, a thread is started to run the scheduler and the main processing thread is used to cancel the event.

```
# sched_cancel.py

import sched
import threading
import time

scheduler = sched.scheduler(time.time, time.sleep)

# Set up a global to be modified by the threads
counter = 0

def increment_counter(name):
    global counter
    print('EVENT:', time.ctime(time.time()), name)
    counter += 1
    print('NOW:', counter)

print('START:', time.ctime(time.time()))
e1 = scheduler.enter(2, 1, increment_counter, ('E1',))
e2 = scheduler.enter(3, 1, increment_counter, ('E2',))

# Start a thread to run the events
t = threading.Thread(target=scheduler.run)
t.start()

# Back in the main thread, cancel the first scheduled event.
scheduler.cancel(e1)

# Wait for the scheduler to finish running in the thread
t.join()

print('FINAL:', counter)
```

Two events were scheduled, but the first was later canceled. Only the second event runs, so the counter variable is only incremented one time.

```
$ python3 sched_cancel.py

START: Sun Sep  4 16:21:13 2016
EVENT: Sun Sep  4 16:21:16 2016 E2
NOW: 1
FINAL: 1
```

See also

- [Standard library documentation for sched](#)
- [time](#) - The time module.

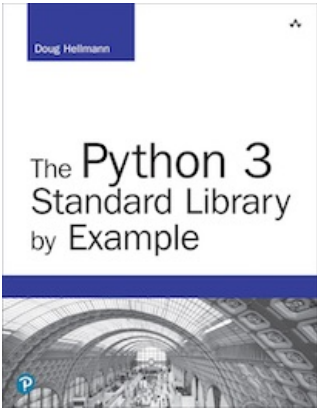
Quick Links

- [Running Events With a Delay](#)
- [Overlapping Events](#)
- [Event Priorities](#)
- [Canceling Events](#)

This page was last updated 2016-12-30.

Navigation

- [atexit — Program Shutdown Callbacks](#)
- [Internationalization and Localization](#)





[Get the book](#)

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for [examples for Python 2?](#)

This Site

-  [Module Index](#)
-  [Index](#)



© Copyright 2019, Doug Hellmann



Other Writing

-  [Blog](#)
-  [The Python Standard Library By Example](#)