Unittest for `task`

May 15, 2026

Contents

1 Test Information

1.1 Testobject Information

The Module task is designed to help with task issues like periodic tasks, delayed tasks, queues, threaded queues and crontabs. For more Information read the documentation.

Information	
Name	task
State	Released
Version	0.6.8
Dependencies	

1.2 Testdata Information

Information	
Version	0.6.8
Testruns	Library test

1.3 Testsystem Information

System Information	
Architecture	64bit
Distribution	Debian GNU/Linux 13 trixie
Hostname	ahorn
Kernel	6.12.86+deb13-amd64 (#1 SMP PREEMPT_DYNAMIC Debian 6.12.86-1 (2026-05-08))
Machine	x86_64
Path	/home/dirk/work/unittest_collection/task/unittest/src
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun: Library test

Number of tests	**9**
Number of successfull tests	**9**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	217.025s

2.2 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
task	98.9%	98.0%
task.__init__.py	98.9%	

3 Testcases with no corresponding Requirement

3.1 Summary for testrun: Library test

3.1.1 pylibs.task.delayed: Test parallel processing and timing for a delayed execution

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.1!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:41,368
Finished-Time:	2026-05-15 18:43:41,881
Time-Consumption	0.512s

Testsummary:

Info	Added a delayed task for execution in 0.250s.
Success	Execution of task and delayed task (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.
Success	Time consumption is correct (Content 0.2504415512084961 in [0.2465 ... 0.2545] and Type is <class 'float'>).
Info	Added a delayed task for execution in 0.010s.
Success	Execution of task and delayed task (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.
Success	Time consumption is correct (Content 0.01038503646850586 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).
Info	Added a delayed task for execution in 0.005s.
Success	Execution of task and delayed task (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.
Success	Time consumption is correct (Content 0.0051746368408203125 in [0.00395 ... 0.00705] and Type is <class 'float'>).

3.1.2 pylibs.task.periodic: Test periodic execution

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.2!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:41,881
Finished-Time:	2026-05-15 18:43:44,427
Time-Consumption	2.546s

Testsummary:

Info	Running a periodic task for 10 cycles with a cycletime of 0.25s
Success	Minimum cycle time is correct (Content 0.25066661834716797 in [0.2465 ... 0.2545] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.250814782248603 in [0.2465 ... 0.2545] and Type is <class 'float'>).

Success	Maximum cycle time is correct (Content 0.25113558769226074 in [0.2465 ... 0.2565] and Type is <class 'float'>).
Info	Running a periodic task for 10 cycles with a cycletime of 0.01s
Success	Minimum cycle time is correct (Content 0.010708093643188477 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.010889053344726562 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).
Success	Maximum cycle time is correct (Content 0.011335611343383789 in [0.008900000000000002 ... 0.0141] and Type is <class 'float'>).
Info	Running a periodic task for 10 cycles with a cycletime of 0.005s
Success	Minimum cycle time is correct (Content 0.005619049072265625 in [0.00395 ... 0.00705] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.006003247367011176 in [0.00395 ... 0.00705] and Type is <class 'float'>).
Success	Maximum cycle time is correct (Content 0.007524013519287109 in [0.00395 ... 0.009049999999999999] and Type is <class 'float'>).

3.1.3 pylibs.task.queue: Test qsize and queue execution order by priority

Testresult

This test was passed with the state: Success. See also full trace in section A.1.3!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:44,428
Finished-Time:	2026-05-15 18:43:44,534
Time-Consumption	0.106s

Testsummary:

Info	Enqueued 6 unordered tasks.
Success	Size of Queue before execution is correct (Content 6 and Type is <class 'int'>).
Success	Size of Queue after execution is correct (Content 0 and Type is <class 'int'>).
Success	Queue execution (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

3.1.4 pylibs.task.queue: Test stop method

Testresult

This test was passed with the state: Success. See also full trace in section A.1.4!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:44,535
Finished-Time:	2026-05-15 18:43:44,643
Time-Consumption	0.108s

Testsummary:

Info	Enqueued 6 tasks (stop request within 4th task).

Success	Size of Queue before 1st execution is correct (Content 6 and Type is <class 'int'>).
Success	Size of Queue after 1st execution is correct (Content 2 and Type is <class 'int'>).
Success	Queue execution (1st part; identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.
Success	Size of Queue after 2nd execution is correct (Content 0 and Type is <class 'int'>).
Success	Queue execution (2nd part; identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

3.1.5 pylibs.task.queue: Test clean_queue method

Testresult

This test was passed with the state: Success. See also full trace in section A.1.5!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:44,643
Finished-Time:	2026-05-15 18:43:44,649
Time-Consumption	0.005s

Testsummary:

Info	Enqueued 6 tasks (stop request within 3rd task).
Success	Size of Queue before execution is correct (Content 6 and Type is <class 'int'>).
Success	Size of Queue after execution is correct (Content 3 and Type is <class 'int'>).
Success	Queue execution (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.
Info	Cleaning Queue.
Success	Size of Queue after cleaning queue is correct (Content 0 and Type is <class 'int'>).

3.1.6 pylibs.task.threaded_queue: Test qsize and queue execution order by priority

Testresult

This test was passed with the state: Success. See also full trace in section A.1.6!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:44,649
Finished-Time:	2026-05-15 18:43:47,771
Time-Consumption	3.122s

Testsummary:

Info	Enqueued 6 unordered tasks.
Success	Size of Queue before execution is correct (Content 7 and Type is <class 'int'>).
Info	Executing Queue, till Queue is empty..
Success	Size of Queue after execution is correct (Content 0 and Type is <class 'int'>).
Success	Queue execution (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.
Info	Setting expire flag and enqueued again 2 tasks.
Success	Size of Queue before restarting queue is correct (Content 2 and Type is <class 'int'>).

Info	Executing Queue, till Queue is empty..
Success	Queue execution (rerun; identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

3.1.7 pylibs.task.threaded_queue: Test enqueue while queue is running

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.7!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:47,772
Finished-Time:	2026-05-15 18:43:48,378
Time-Consumption	0.606s

Testsummary:

Success	Size of Queue before execution is correct (Content 0 and Type is <class 'int'>).
Info	Enqueued 2 tasks.
Success	Size of Queue after execution is correct (Content 0 and Type is <class 'int'>).
Success	Queue execution (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

3.1.8 pylibs.task.crontab: Test cronjob

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.8!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:48,676
Finished-Time:	2026-05-15 18:43:48,691
Time-Consumption	0.015s

Testsummary:

Info	Initialising cronjob with minute: [23, 45]; hour: [12, 17]; day: 25; month: any; day_of_week: any.
Success	Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content True and Type is <class 'bool'>).
Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content True and Type is <class 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Info	Storing reminder for execution (minute: 23, hour: 17, day: 25, month: 2, day_of_week: 1).

Success	Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content True and Type is <class 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Info	Resetting trigger condition with minute: 22; hour: any; day: [12, 17, 25], month: 2.
Success	Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content True and Type is <class 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 05, day_of_week: 3 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).
Info	Resetting trigger condition (again).
Success	1st run - execution not needed is correct (Content False and Type is <class 'bool'>).
Success	2nd run - execution not needed is correct (Content False and Type is <class 'bool'>).
Success	3rd run - execution needed is correct (Content True and Type is <class 'bool'>).
Success	4th run - execution needed is correct (Content True and Type is <class 'bool'>).
Success	5th run - execution not needed is correct (Content False and Type is <class 'bool'>).
Success	6th run - execution not needed is correct (Content False and Type is <class 'bool'>).

3.1.9 pylibs.task.crontab: Test crontab

Testresult

This test was passed with the state: Success. See also full trace in section A.1.9!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (329)
Start-Time:	2026-05-15 18:43:48,692
Finished-Time:	2026-05-15 18:47:18,696
Time-Consumption	210.004s

Testsummary:

Info	Creating Crontab with callback execution in +1 and +3 minutes.
Success	Number of submitted values is correct (Content 2 and Type is <class 'int'>).
Success	Timing of crontasks: Valueaccuracy and number of submitted values is correct. See detailed log for more information.

A Trace for testrun: Library test

A.1 Tests with status Info (9)

A.1.1 pylibs.task.delayed: Test parallel processing and timing for a delayed execution

Testresult
This test was passed with the state: **Success**.

Info	Added a delayed task for execution in 0.250s.

Success	Execution of task and delayed task (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Execution of task and delayed task (identified by a submitted sequence number)): [ 1,
↪  2 ] (<class 'list'>)
Expectation (Execution of task and delayed task (identified by a submitted sequence number)):
↪  result = [ 1, 2 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 2 (<class 'int'>)
Expectation (Submitted value number 2): result = 2 (<class 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <class 'int'>).
```

Success	Time consumption is correct (Content 0.2504415512084961 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Time consumption): 0.2504415512084961 (<class 'float'>)
Expectation (Time consumption): 0.2465 <= result <= 0.2545
```

Info	Added a delayed task for execution in 0.010s.

Success	Execution of task and delayed task (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Execution of task and delayed task (identified by a submitted sequence number)): [ 1,
↪  2 ] (<class 'list'>)
Expectation (Execution of task and delayed task (identified by a submitted sequence number)):
↪  result = [ 1, 2 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 2 (<class 'int'>)
```

```
Expectation (Submitted value number 2): result = 2 (<class 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <class 'int'>).
```

Success	Time consumption is correct (Content 0.01038503646850586 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Time consumption): 0.01038503646850586 (<class 'float'>)
Expectation (Time consumption): 0.008900000000000002 <= result <= 0.0121
```

Info	Added a delayed task for execution in 0.005s.

Success	Execution of task and delayed task (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Execution of task and delayed task (identified by a submitted sequence number)): [ 1,
↪  2 ] (<class 'list'>)
Expectation (Execution of task and delayed task (identified by a submitted sequence number)):
↪  result = [ 1, 2 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 2 (<class 'int'>)
Expectation (Submitted value number 2): result = 2 (<class 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <class 'int'>).
```

Success	Time consumption is correct (Content 0.0051746368408203125 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Time consumption): 0.0051746368408203125 (<class 'float'>)
Expectation (Time consumption): 0.00395 <= result <= 0.00705
```

A.1.2 pylibs.task.periodic: Test periodic execution

Testresult

This test was passed with the state: **Success**.

Info	Running a periodic task for 10 cycles with a cycletime of 0.25s

```
Task execution number 1 at 1778863421.883826
Task execution number 2 at 1778863422.134492
Task execution number 3 at 1778863422.385178
Task execution number 4 at 1778863422.636012
Task execution number 5 at 1778863422.886886
Task execution number 6 at 1778863423.137625
Task execution number 7 at 1778863423.388301
```

```
Task execution number 8 at 1778863423.639189
Task execution number 9 at 1778863423.890325
Task execution number 10 at 1778863424.141159
```

Success	Minimum cycle time is correct (Content 0.25066661834716797 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.25066661834716797 (<class 'float'>)
Expectation (Minimum cycle time): 0.2465 <= result <= 0.2545
```

Success	Mean cycle time is correct (Content 0.250814782248603 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.250814782248603 (<class 'float'>)
Expectation (Mean cycle time): 0.2465 <= result <= 0.2545
```

Success	Maximum cycle time is correct (Content 0.25113558769226074 in [0.2465 ... 0.2565] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.25113558769226074 (<class 'float'>)
Expectation (Maximum cycle time): 0.2465 <= result <= 0.2565
```

Info	Running a periodic task for 10 cycles with a cycletime of 0.01s

```
Task execution number 1 at 1778863424.192312
Task execution number 2 at 1778863424.203155
Task execution number 3 at 1778863424.214052
Task execution number 4 at 1778863424.224918
Task execution number 5 at 1778863424.235652
Task execution number 6 at 1778863424.246360
Task execution number 7 at 1778863424.257293
Task execution number 8 at 1778863424.268220
Task execution number 9 at 1778863424.278978
Task execution number 10 at 1778863424.290313
```

Success	Minimum cycle time is correct (Content 0.010708093643188477 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.010708093643188477 (<class 'float'>)
Expectation (Minimum cycle time): 0.008900000000000002 <= result <= 0.0121
```

Success	Mean cycle time is correct (Content 0.010889053344726562 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.010889053344726562 (<class 'float'>)
Expectation (Mean cycle time): 0.008900000000000002 <= result <= 0.0121
```

Success	Maximum cycle time is correct (Content 0.011335611343383789 in [0.008900000000000002 ... 0.0141] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.011335611343383789 (<class 'float'>)
Expectation (Maximum cycle time): 0.008900000000000002 <= result <= 0.0141
```

Info	Running a periodic task for 10 cycles with a cycletime of 0.005s

```
Task execution number 1 at 1778863424.316036
Task execution number 2 at 1778863424.321714
Task execution number 3 at 1778863424.327686
Task execution number 4 at 1778863424.333362
Task execution number 5 at 1778863424.339052
Task execution number 6 at 1778863424.345016
Task execution number 7 at 1778863424.350828
Task execution number 8 at 1778863424.356922
Task execution number 9 at 1778863424.362541
Task execution number 10 at 1778863424.370065
```

Success	Minimum cycle time is correct (Content 0.005619049072265625 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.005619049072265625 (<class 'float'>)
Expectation (Minimum cycle time): 0.00395 <= result <= 0.00705
```

Success	Mean cycle time is correct (Content 0.006003247367011176 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.006003247367011176 (<class 'float'>)
Expectation (Mean cycle time): 0.00395 <= result <= 0.00705
```

Success	Maximum cycle time is correct (Content 0.007524013519287109 in [0.00395 ... 0.009049999999999999] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.007524013519287109 (<class 'float'>)
Expectation (Maximum cycle time): 0.00395 <= result <= 0.009049999999999999
```

A.1.3 pylibs.task.queue: Test qsize and queue execution order by priority

Testresult

This test was passed with the state: **Success**.

Info	Enqueued 6 unordered tasks.

Success	Size of Queue before execution is correct (Content 6 and Type is <class 'int'>).

```
Result (Size of Queue before execution): 6 (<class 'int'>)
```

```
Expectation (Size of Queue before execution): result = 6 (<class 'int'>)
```

Success	Size of Queue after execution is correct (Content 0 and Type is <class 'int'>).

```
Result (Size of Queue after execution): 0 (<class 'int'>)
Expectation (Size of Queue after execution): result = 0 (<class 'int'>)
```

Success	Queue execution (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Queue execution (identified by a submitted sequence number)): [ 1, 2, 3, 5, 6, 7 ]
↪  (<class 'list'>)
Expectation (Queue execution (identified by a submitted sequence number)): result = [ 1, 2, 3,
↪  5, 6, 7 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 2 (<class 'int'>)
Expectation (Submitted value number 2): result = 2 (<class 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <class 'int'>).
Result (Submitted value number 3): 3 (<class 'int'>)
Expectation (Submitted value number 3): result = 3 (<class 'int'>)
Submitted value number 3 is correct (Content 3 and Type is <class 'int'>).
Result (Submitted value number 4): 5 (<class 'int'>)
Expectation (Submitted value number 4): result = 5 (<class 'int'>)
Submitted value number 4 is correct (Content 5 and Type is <class 'int'>).
Result (Submitted value number 5): 6 (<class 'int'>)
Expectation (Submitted value number 5): result = 6 (<class 'int'>)
Submitted value number 5 is correct (Content 6 and Type is <class 'int'>).
Result (Submitted value number 6): 7 (<class 'int'>)
Expectation (Submitted value number 6): result = 7 (<class 'int'>)
Submitted value number 6 is correct (Content 7 and Type is <class 'int'>).
```

A.1.4 pylibs.task.queue: Test stop method

Testresult

This test was passed with the state: **Success**.

Info	Enqueued 6 tasks (stop request within 4th task).

Success	Size of Queue before 1st execution is correct (Content 6 and Type is <class 'int'>).

```
Result (Size of Queue before 1st execution): 6 (<class 'int'>)
Expectation (Size of Queue before 1st execution): result = 6 (<class 'int'>)
```

Success Size of Queue after 1st execution is correct (Content 2 and Type is <class 'int'>).

```
Result (Size of Queue after 1st execution): 2 (<class 'int'>)
Expectation (Size of Queue after 1st execution): result = 2 (<class 'int'>)
```

Success Queue execution (1st part; identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Queue execution (1st part; identified by a submitted sequence number)): [ 1, 2, 3, 5 ]
↪  (<class 'list'>)
Expectation (Queue execution (1st part; identified by a submitted sequence number)): result =
↪  [ 1, 2, 3, 5 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 2 (<class 'int'>)
Expectation (Submitted value number 2): result = 2 (<class 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <class 'int'>).
Result (Submitted value number 3): 3 (<class 'int'>)
Expectation (Submitted value number 3): result = 3 (<class 'int'>)
Submitted value number 3 is correct (Content 3 and Type is <class 'int'>).
Result (Submitted value number 4): 5 (<class 'int'>)
Expectation (Submitted value number 4): result = 5 (<class 'int'>)
Submitted value number 4 is correct (Content 5 and Type is <class 'int'>).
```

Success Size of Queue after 2nd execution is correct (Content 0 and Type is <class 'int'>).

```
Result (Size of Queue after 2nd execution): 0 (<class 'int'>)
Expectation (Size of Queue after 2nd execution): result = 0 (<class 'int'>)
```

Success Queue execution (2nd part; identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Queue execution (2nd part; identified by a submitted sequence number)): [ 6, 7 ]
↪  (<class 'list'>)
Expectation (Queue execution (2nd part; identified by a submitted sequence number)): result =
↪  [ 6, 7 ] (<class 'list'>)
Result (Submitted value number 1): 6 (<class 'int'>)
Expectation (Submitted value number 1): result = 6 (<class 'int'>)
Submitted value number 1 is correct (Content 6 and Type is <class 'int'>).
Result (Submitted value number 2): 7 (<class 'int'>)
Expectation (Submitted value number 2): result = 7 (<class 'int'>)
Submitted value number 2 is correct (Content 7 and Type is <class 'int'>).
```

A.1.5 pylibs.task.queue: Test clean_queue method

Testresult

This test was passed with the state: **Success**.

Info	Enqueued 6 tasks (stop request within 3rd task).

Success	Size of Queue before execution is correct (Content 6 and Type is <class 'int'>).

```
Result (Size of Queue before execution): 6 (<class 'int'>)
Expectation (Size of Queue before execution): result = 6 (<class 'int'>)
```

Success	Size of Queue after execution is correct (Content 3 and Type is <class 'int'>).

```
Result (Size of Queue after execution): 3 (<class 'int'>)
Expectation (Size of Queue after execution): result = 3 (<class 'int'>)
```

Success	Queue execution (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Queue execution (identified by a submitted sequence number)): [ 1, 2, 3 ] (<class
↪  'list'>)
Expectation (Queue execution (identified by a submitted sequence number)): result = [ 1, 2, 3
↪  ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 2 (<class 'int'>)
Expectation (Submitted value number 2): result = 2 (<class 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <class 'int'>).
Result (Submitted value number 3): 3 (<class 'int'>)
Expectation (Submitted value number 3): result = 3 (<class 'int'>)
Submitted value number 3 is correct (Content 3 and Type is <class 'int'>).
```

Info	Cleaning Queue.

Success	Size of Queue after cleaning queue is correct (Content 0 and Type is <class 'int'>).

```
Result (Size of Queue after cleaning queue): 0 (<class 'int'>)
Expectation (Size of Queue after cleaning queue): result = 0 (<class 'int'>)
```

A.1.6 pylibs.task.threaded_queue: Test qsize and queue execution order by priority

Testresult
This test was passed with the state: **Success**.

Info	Enqueued 6 unordered tasks.

```
Adding Task 5.1 with Priority 5
Adding Task 3.0 with Priority 3
Adding Task 7.0 with Priority 7
Adding Task 5.2 with Priority 5
Adding Task 2.0 with Priority 2
Adding Task 6.0 with Priority 6
Adding Task 1.0 with Priority 1
```

Success	Size of Queue before execution is correct (Content 7 and Type is <class 'int'>).

```
Result (Size of Queue before execution): 7 (<class 'int'>)
Expectation (Size of Queue before execution): result = 7 (<class 'int'>)
```

Info	Executing Queue, till Queue is empty..

```
Starting Queue execution (run)
Queue is empty.
```

Success	Size of Queue after execution is correct (Content 0 and Type is <class 'int'>).

```
Result (Size of Queue after execution): 0 (<class 'int'>)
Expectation (Size of Queue after execution): result = 0 (<class 'int'>)
```

Success	Queue execution (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Queue execution (identified by a submitted sequence number)): [ 1, 2, 3, 5.1, 5.2, 6,
↪  7 ] (<class 'list'>)
Expectation (Queue execution (identified by a submitted sequence number)): result = [ 1, 2, 3,
↪  5.1, 5.2, 6, 7 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 2 (<class 'int'>)
Expectation (Submitted value number 2): result = 2 (<class 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <class 'int'>).
Result (Submitted value number 3): 3 (<class 'int'>)
Expectation (Submitted value number 3): result = 3 (<class 'int'>)
```

```
Submitted value number 3 is correct (Content 3 and Type is <class 'int'>).
Result (Submitted value number 4): 5.1 (<class 'float'>)
Expectation (Submitted value number 4): result = 5.1 (<class 'float'>)
Submitted value number 4 is correct (Content 5.1 and Type is <class 'float'>).
Result (Submitted value number 5): 5.2 (<class 'float'>)
Expectation (Submitted value number 5): result = 5.2 (<class 'float'>)
Submitted value number 5 is correct (Content 5.2 and Type is <class 'float'>).
Result (Submitted value number 6): 6 (<class 'int'>)
Expectation (Submitted value number 6): result = 6 (<class 'int'>)
Submitted value number 6 is correct (Content 6 and Type is <class 'int'>).
Result (Submitted value number 7): 7 (<class 'int'>)
Expectation (Submitted value number 7): result = 7 (<class 'int'>)
Submitted value number 7 is correct (Content 7 and Type is <class 'int'>).
```

Info Setting expire flag and enqueued again 2 tasks.

```
Expire executed
Adding Task 6 with Priority 6
Adding Task 1 with Priority 1
```

Success Size of Queue before restarting queue is correct (Content 2 and Type is <class 'int'>).

```
Result (Size of Queue before restarting queue): 2 (<class 'int'>)
Expectation (Size of Queue before restarting queue): result = 2 (<class 'int'>)
```

Info Executing Queue, till Queue is empty..

```
Starting Queue execution (run)
Queue joined and stopped.
```

Success Queue execution (rerun; identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Queue execution (rerun; identified by a submitted sequence number)): [ 1, 6 ] (<class
↪  'list'>)
Expectation (Queue execution (rerun; identified by a submitted sequence number)): result = [
↪  1, 6 ] (<class 'list'>)
Result (Submitted value number 1): 1 (<class 'int'>)
Expectation (Submitted value number 1): result = 1 (<class 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 2): 6 (<class 'int'>)
Expectation (Submitted value number 2): result = 6 (<class 'int'>)
Submitted value number 2 is correct (Content 6 and Type is <class 'int'>).
```

A.1.7 pylibs.task.threaded_queue: Test enqueue while queue is running

Testresult

This test was passed with the state: **Success**.

Success	Size of Queue before execution is correct (Content 0 and Type is <class 'int'>).

```
Result (Size of Queue before execution): 0 (<class 'int'>)
Expectation (Size of Queue before execution): result = 0 (<class 'int'>)
```

Info	Enqueued 2 tasks.

```
Starting Queue execution (run)
Adding Task 6 with Priority 6 and waiting for 0.1s (half of the queue task delay time)
Adding Task 3 with Priority 3
Adding Task 2 with Priority 2
Adding Task 1 with Priority 1
```

Success	Size of Queue after execution is correct (Content 0 and Type is <class 'int'>).

```
Result (Size of Queue after execution): 0 (<class 'int'>)
Expectation (Size of Queue after execution): result = 0 (<class 'int'>)
```

Success	Queue execution (identified by a submitted sequence number): Values and number of submitted values is correct. See detailed log for more information.

```
Result (Queue execution (identified by a submitted sequence number)): [ 6, 1, 2, 3 ] (<class
↪  'list'>)
Expectation (Queue execution (identified by a submitted sequence number)): result = [ 6, 1, 2,
↪  3 ] (<class 'list'>)
Result (Submitted value number 1): 6 (<class 'int'>)
Expectation (Submitted value number 1): result = 6 (<class 'int'>)
Submitted value number 1 is correct (Content 6 and Type is <class 'int'>).
Result (Submitted value number 2): 1 (<class 'int'>)
Expectation (Submitted value number 2): result = 1 (<class 'int'>)
Submitted value number 2 is correct (Content 1 and Type is <class 'int'>).
Result (Submitted value number 3): 2 (<class 'int'>)
Expectation (Submitted value number 3): result = 2 (<class 'int'>)
Submitted value number 3 is correct (Content 2 and Type is <class 'int'>).
Result (Submitted value number 4): 3 (<class 'int'>)
Expectation (Submitted value number 4): result = 3 (<class 'int'>)
Submitted value number 4 is correct (Content 3 and Type is <class 'int'>).
```

A.1.8 pylibs.task.crontab: Test cronjob

Testresult

This test was passed with the state: **Success**.

Info	Initialising cronjob with minute: [23, 45]; hour: [12, 17]; day: 25; month: any; day_of_week: any.

Success	Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content True and Type is <class 'bool'>).

```
Result (Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1): True
↪  (<class 'bool'>)
```

```
Expectation (Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1):
↪  result = True (<class 'bool'>)
```

Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content True and Type is <class 'bool'>).

```
Result (Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5): True
↪  (<class 'bool'>)
```

```
Expectation (Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5):
↪  result = True (<class 'bool'>)
```

Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
```

```
Expectation (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3): False
↪  (<class 'bool'>)
```

```
Expectation (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
```

```
Expectation (Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Info	Storing reminder for execution (minute: 23, hour: 17, day: 25, month: 2, day_of_week: 1).

Success	Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content True and Type is <class 'bool'>).

```
Result (Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5): True
↪  (<class 'bool'>)
Expectation (Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5):
↪  result = True (<class 'bool'>)
```

Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
```

```
Expectation (Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Info	Resetting trigger condition with minute: 22; hour: any; day: [12, 17, 25], month: 2.

Success	Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 23; hour: 17; day: 25; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content True and Type is <class 'bool'>).

```
Result (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1): True
↪  (<class 'bool'>)
Expectation (Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1):
↪  result = True (<class 'bool'>)
```

Success	Return value for minute: 22; hour: 17; day: 25; month: 05, day_of_week: 3 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 22; hour: 17; day: 25; month: 05, day_of_week: 3): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 22; hour: 17; day: 25; month: 05, day_of_week: 3):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <class 'bool'>).

```
Result (Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1): False
↪  (<class 'bool'>)
Expectation (Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1):
↪  result = False (<class 'bool'>)
```

Info	Resetting trigger condition (again).

Success	1st run - execution not needed is correct (Content False and Type is <class 'bool'>).

```
Result (1st run - execution not needed): False (<class 'bool'>)
Expectation (1st run - execution not needed): result = False (<class 'bool'>)
```

Success	2nd run - execution not needed is correct (Content False and Type is <class 'bool'>).

```
Result (2nd run - execution not needed): False (<class 'bool'>)
Expectation (2nd run - execution not needed): result = False (<class 'bool'>)
```

Success	3rd run - execution needed is correct (Content True and Type is <class 'bool'>).

```
Result (3rd run - execution needed): True (<class 'bool'>)
Expectation (3rd run - execution needed): result = True (<class 'bool'>)
```

Success	4th run - execution needed is correct (Content True and Type is <class 'bool'>).

```
Result (4th run - execution needed): True (<class 'bool'>)
Expectation (4th run - execution needed): result = True (<class 'bool'>)
```

Success	5th run - execution not needed is correct (Content False and Type is <class 'bool'>).

```
Result (5th run - execution not needed): False (<class 'bool'>)
Expectation (5th run - execution not needed): result = False (<class 'bool'>)
```

Success	6th run - execution not needed is correct (Content False and Type is <class 'bool'>).

```
Result (6th run - execution not needed): False (<class 'bool'>)
Expectation (6th run - execution not needed): result = False (<class 'bool'>)
```

A.1.9 pylibs.task.crontab: Test crontab

Testresult
This test was passed with the state: **Success**.

Info	Creating Crontab with callback execution in +1 and +3 minutes.

Success	Number of submitted values is correct (Content 2 and Type is <class 'int'>).

```
Crontab accuracy is 30s
Crontab execution number 1 at 1778863458s, requested for 1778863440s
Crontab execution number 2 at 1778863578s, requested for 1778863560s
Result (Timing of crontasks): [ 1778863458, 1778863578 ] (<class 'list'>)
Result (Number of submitted values): 2 (<class 'int'>)
Expectation (Number of submitted values): result = 2 (<class 'int'>)
```

Success	Timing of crontasks: Valueaccuracy and number of submitted values is correct. See detailed log for more information.

```
Result (Submitted value number 1): 1778863458 (<class 'int'>)
Expectation (Submitted value number 1): 1778863440 <= result <= 1778863471
Submitted value number 1 is correct (Content 1778863458 in [1778863440 ... 1778863471] and
↪  Type is <class 'int'>).
Result (Submitted value number 2): 1778863578 (<class 'int'>)
Expectation (Submitted value number 2): 1778863560 <= result <= 1778863591
Submitted value number 2 is correct (Content 1778863578 in [1778863560 ... 1778863591] and
↪  Type is <class 'int'>).
```

B Test-Coverage

B.1 task

The line coverage for task was 98.9%
The branch coverage for task was 98.0%

B.1.1 task.__init__.py

The line coverage for task.__init__.py was 98.9%
The branch coverage for task.__init__.py was 98.0%

```
1 """
2 task (Task Module)
3 ===================
4
5 **Author:**
6
```

```
* Dirk Alders <sudo-dirk@mount-mockery.de>

**Description:**

    This Module supports helpfull classes for queues, tasks, ...

**Submodules:**

* :class:`task.crontab`
* :class:`task.delayed`
* :class:`task.periodic`
* :class:`task.queue`
* :class:`task.threaded_queue`

**Unittest:**

        See also the :download:`unittest <task/_testresults_/unittest.pdf>` documentation.

**Module Documentation:**

"""

__VERSION__ = "0.6.8"
  DEPENDENCIES    = []

import logging
import threading
import time
from queue import Empty, PriorityQueue

try:
    from config import APP_NAME as ROOT_LOGGER_NAME
except ImportError:
    ROOT_LOGGER_NAME = "root"
logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)

__DESCRIPTION__ = """The Module {\\tt %s} is designed to help with task issues like periodic
    tasks, delayed tasks, queues, threaded queues and crontabs.
For more Information read the documentation.""" % __name__.replace("_", "\\_")
"""The Module Description"""


class queue(object):
    """
    Class to execute queued callbacks.

    :param bool expire: The default value for expire. See also :py:func:`expire`.

    **Example:**

    .. literalinclude:: task/_examples_/tqueue.py

    Will result to the following output:

    .. literalinclude:: task/_examples_/tqueue.log
    """

    class job(object):
        def __init__(self, priority, callback, *args, **kwargs):
            self.time = time.time()
            self.priority = priority
            self.callback = callback
            self.args = args
            self.kwargs = kwargs
```

```
70
71          def run(self, queue):
72              self.callback(queue, *self.args, **self.kwargs)
73
74          def __lt__(self, other):
75              if self.priority != other.priority:
76                  return self.priority < other.priority
77              else:
78                  return self.time < other.time
79
80      def __init__(self, expire=True):
81          self.__expire = expire
82          self.__stop = False
83          self.queue = PriorityQueue()
84
85      def clean_queue(self):
86          """
87          This Methods removes all jobs from the queue.
88
89          .. note:: Be aware that already running jobs will not be terminated.
90          """
91          while not self.queue.empty():
92              try:
93                  self.queue.get(False)
94              except Empty:  # This block is hard to reach for a testcase, but is
95                  continue  # needed, if the thread runs dry while cleaning the queue.
96              self.queue.task_done()
97
98      def enqueue(self, priority, callback, *args, **kwargs):
99          """
100         This enqueues a given callback.
101
102         :param number priority: The priority indication number of this task. The lowest value
    will be queued first.
103         :param callback callback: Callback to be executed
104         :param args args: Arguments to be given to callback
105         :param kwargs kwargs: Keword Arguments to be given to callback
106
107         .. note:: Callback will get this instance as first argument, followed by :py:data:`args`
    und :py:data:`kwargs`.
108         """
109         self.queue.put(self.job(priority, callback, *args, **kwargs))
110
111     def qsize(self):
112         return self.queue.qsize()
113
114     def run(self):
115         """
116         This starts the execution of the queued callbacks.
117         """
118         self.__stop = False
119         while not self.__stop:
120             try:
121                 self.queue.get(timeout=0.1).run(self)
122             except Empty:
123                 if self.__expire:
124                     break
125         if type(self) is threaded_queue:
126             self.thread = None
127
128     def expire(self):
```

```
129         """
130         This sets the expire flag. That means that the process will stop after queue gets empty.
131         """
132         self.__expire = True
133 
134     def stop(self):
135         """
136         This sets the stop flag. That means that the process will stop after finishing the active
     task.
137         """
138         self.__stop = True
139 
140 
141 class threaded_queue(queue):
142     """Class to execute queued callbacks in a background thread (See also parent :py:class:`queue
    `).
143 
144     :param bool expire: The default value for expire. See also :py:func:`queue.expire`.
145 
146     **Example:**
147 
148     .. literalinclude:: task/_examples_/threaded_queue.py
149 
150     Will result to the following output:
151 
152     .. literalinclude:: task/_examples_/threaded_queue.log
153     """
154 
155     def __init__(self, expire=False):
156         queue.__init__(self, expire=expire)
157         self.thread = None
158 
159     def run(self):
160         if self.thread is None:
161             self.thread = threading.Thread(target=self._start, args=(), daemon=True)
162             self.thread.daemon = True  # Daemonize thread
163             self.thread.start()  # Start the execution
164 
165     def join(self):
166         """
167         This blocks till the queue is empty.
168 
169         .. note:: If the queue does not run dry, join will block till the end of the days.
170         """
171         self.expire()
172         if self.thread is not None:
173             self.thread.join()
174 
175     def stop(self):
176         queue.stop(self)
177         self.join()
178 
179     def _start(self):
180         queue.run(self)
181 
182 
183 class periodic(object):
```

```
184     """
185     Class to execute a callback cyclicly.
186
187     :param float cycle_time: Cycle time in seconds -- callback will be executed every *cycle_time
        * seconds
188     :param callback callback: Callback to be executed
189     :param args args: Arguments to be given to the callback
190     :param kwargs kwargs: Keword Arguments to be given to callback
191
192     .. note:: The Callback will get this instance as first argument, followed by :py:data:`args`
        und :py:data:`kwargs`.
193
194     **Example:**
195
196     .. literalinclude:: task/_examples_/periodic.py
197
198     Will result to the following output:
199
200     .. literalinclude:: task/_examples_/periodic.log
201     """
202
203     def __init__(self, cycle_time, callback, *args, **kwargs):
204         self._lock = threading.Lock()
205         self._timer = None
206         self.callback = callback
207         self.cycle_time = cycle_time
208         self.args = args
209         self.kwargs = kwargs
210         self._stopped = True
211         self._last_tm = None
212         self.dt = None
213
214     def join(self):
215         """
216         This blocks till the cyclic task is terminated.
217
218         .. note:: Using join means that somewhere has to be a condition calling :py:func:`stop`
        to terminate. Otherwise :func:`task.join` will never return.
219         """
220         while not self._stopped:
221             time.sleep(0.1)
222
223     def run(self):
224         """
225         This starts the cyclic execution of the given callback.
226         """
227         if self._stopped:
228             self._set_timer(force_now=True)
229
230     def stop(self):
231         """
232         This stops the execution of any further task.
233         """
234         self._lock.acquire()
235         self._stopped = True
236         if self._timer is not None:
237             self._timer.cancel()
238         self._lock.release()
239
240     def _set_timer(self, force_now=False):
```

```
241         """
242         This sets the timer for the execution of the next task.
243         """
244         self._lock.acquire()
245         self._stopped = False
246         if force_now:
247             self._timer = threading.Timer(0, self._start)
248         else:
249             self._timer = threading.Timer(self.cycle_time, self._start)
250         self._timer.daemon = True
251         self._timer.start()
252         self._lock.release()
253 
254     def _start(self):
255         tm = time.time()
256         if self._last_tm is not None:
257             self.dt = tm - self._last_tm
258         self._set_timer(force_now=False)
259         self.callback(self, *self.args, **self.kwargs)
260         self._last_tm = tm
261 
262 
263 class delayed(periodic):
264     """Class to execute a callback a given time in the future. See also parent :py:class:`
    periodic`.
265 
266     :param float time: Delay time for execution of the given callback
267     :param callback callback: Callback to be executed
268     :param args args: Arguments to be given to callback
269     :param kwargs kwargs: Keword Arguments to be given to callback
270 
271     **Example:**
272 
273     .. literalinclude:: task/_examples_/delayed.py
274 
275     Will result to the following output:
276 
277     .. literalinclude:: task/_examples_/delayed.log
278     """
279 
280     def run(self):
281         """
282         This starts the timer for the delayed execution.
283         """
284         self._set_timer(force_now=False)
285 
286     def _start(self):
287         self.callback(*self.args, **self.kwargs)
288         self.stop()
289 
290 
291 class crontab(periodic):
292     """Class to execute a callback at the specified time conditions. See also parent :py:class:`
    periodic`.
293 
294     :param accuracy: Repeat time in seconds for background task checking event triggering. This
    time is the maximum delay between specified time condition and the execution.
295     :type accuracy: float
296 
297     **Example:**
298 
299     .. literalinclude:: task/_examples_/crontab.py
```

```
300
301     Will result to the following output:
302
303     .. literalinclude:: task/_examples_/crontab.log
304     """
305
306     ANY = "*"
307     """Constant for matching every condition."""
308
309     class cronjob(object):
310         """Class to handle cronjob parameters and cronjob changes.
311
312         :param minute: Minute for execution. Either 0...59, [0...59, 0...59, ...] or :py:const:`crontab.ANY` for every Minute.
313         :type minute: int, list, str
314         :param hour: Hour for execution. Either 0...23, [0...23, 0...23, ...] or :py:const:`crontab.ANY` for every Hour.
315         :type hour: int, list, str
316         :param day_of_month: Day of Month for execution. Either 0...31, [0...31, 0...31, ...] or :py:const:`crontab.ANY` for every Day of Month.
317         :type day_of_month: int, list, str
318         :param month: Month for execution. Either 0...12, [0...12, 0...12, ...] or :py:const:`crontab.ANY` for every Month.
319         :type month: int, list, str
320         :param day_of_week: Day of Week for execution. Either 0...6, [0...6, 0...6, ...] or :py:const:`crontab.ANY` for every Day of Week.
321         :type day_of_week: int, list, str
322         :param callback: The callback to be executed. The instance of :py:class:`cronjob` will be given as the first, args and kwargs as the following parameters.
323         :type callback: func
324
325         .. note:: This class should not be used stand alone. An instance will be created by adding a cronjob by using :py:func:`crontab.add_cronjob()`.
326         """
327
328         class all_match(set):
329             """Universal set - match everything"""
330
331             def __contains__(self, item):
332                 (item)
333                 return True
334
335         def __init__(self, minute, hour, day_of_month, month, day_of_week, callback, *args, **kwargs):
336             self.set_trigger_conditions(
337                 minute or crontab.ANY, hour or crontab.ANY, day_of_month or crontab.ANY, month or crontab.ANY, day_of_week or crontab.ANY
338             )
339             self.callback = callback
340             self.args = args
341             self.kwargs = kwargs
342             self.__last_cron_check_time__ = None
343             self.__last_execution__ = None
344
345         def set_trigger_conditions(self, minute=None, hour=None, day_of_month=None, month=None, day_of_week=None):
346             """This Method changes the execution parameters.
347
348             :param minute: Minute for execution. Either 0...59, [0...59, 0...59, ...] or :py:const:`crontab.ANY` for every Minute.
349             :type minute: int, list, str
```

```
350             :param hour: Hour for execution. Either 0...23, [0...23, 0...23, ...] or :py:const:`crontab.ANY` for every Hour.
351             :type hour: int, list, str
352             :param day_of_month: Day of Month for execution. Either 0...31, [0...31, 0...31, ...] or :py:const:`crontab.ANY` for every Day of Month.
353             :type day_of_month: int, list, str
354             :param month: Month for execution. Either 0...12, [0...12, 0...12, ...] or :py:const:`crontab.ANY` for every Month.
355             :type month: int, list, str
356             :param day_of_week: Day of Week for execution. Either 0...6, [0...6, 0...6, ...] or :py:const:`crontab.ANY` for every Day of Week.
357             :type day_of_week: int, list, str
358             """
359             if minute is not None:
360                 self.minute = self.__conv_to_set__(minute)
361             if hour is not None:
362                 self.hour = self.__conv_to_set__(hour)
363             if day_of_month is not None:
364                 self.day_of_month = self.__conv_to_set__(day_of_month)
365             if month is not None:
366                 self.month = self.__conv_to_set__(month)
367             if day_of_week is not None:
368                 self.day_of_week = self.__conv_to_set__(day_of_week)
369 
370         def __conv_to_set__(self, obj):
371             if obj is crontab.ANY:
372                 return self.all_match()
373             elif isinstance(obj, (int)):
374                 return set([obj])
375             else:
376                 return set(obj)
377 
378         def __execution_needed_for__(self, minute, hour, day_of_month, month, day_of_week):
379             if self.__last_execution__ != [minute, hour, day_of_month, month, day_of_week]:
380                 if (
381                     minute in self.minute
382                     and hour in self.hour
383                     and day_of_month in self.day_of_month
384                     and month in self.month
385                     and day_of_week in self.day_of_week
386                 ):
387                     return True
388             return False
389 
390         def __store_execution_reminder__(self, minute, hour, day_of_month, month, day_of_week):
391             self.__last_execution__ = [minute, hour, day_of_month, month, day_of_week]
392 
393         def cron_execution(self, tm):
394             """This Methods executes the Cron-Callback, if a execution is needed for the given time (depending on the parameters on initialisation)
395 
396             :param tm: (Current) Time Value to be checked. The time needs to be given in seconds since 1970 (e.g. generated by int(time.time())).
397             :type tm: int
398             """
399             if self.__last_cron_check_time__ is None:
400                 self.__last_cron_check_time__ = tm - 1
401             #
402             for t in range(self.__last_cron_check_time__ + 1, tm + 1):
403                 lt = time.localtime(t)
404                 if self.__execution_needed_for__(lt[4], lt[3], lt[2], lt[1], lt[6]):
405                     self.callback(self, *self.args, **self.kwargs)
406                     self.__store_execution_reminder__(lt[4], lt[3], lt[2], lt[1], lt[6])
407                     break
408             self.__last_cron_check_time__ = tm
```

```
409
410     def __init__(self, accuracy=30):
411         periodic.__init__(self, accuracy, self.__periodic__)
412         self.__crontab__ = []
413
414     def __periodic__(self, rt):
415         (rt)
416         tm = int(time.time())
417         for cronjob in self.__crontab__:
418             cronjob.cron_execution(tm)
419
420     def add_cronjob(self, minute, hour, day_of_month, month, day_of_week, callback, *args, **
    kwargs):
421         """This Method adds a cronjob to be executed.
422
423         :param minute: Minute for execution. Either 0...59, [0...59, 0...59, ...] or :py:const:`
    crontab.ANY` for every Minute.
424         :type minute: int, list, str
425         :param hour: Hour for execution. Either 0...23, [0...23, 0...23, ...] or :py:const:`
    crontab.ANY` for every Hour.
426         :type hour: int, list, str
427         :param day_of_month: Day of Month for execution. Either 0...31, [0...31, 0...31, ...] or
    :py:const:`crontab.ANY` for every Day of Month.
428         :type day_of_month: int, list, str
429         :param month: Month for execution. Either 0...12, [0...12, 0...12, ...] or :py:const:`
    crontab.ANY` for every Month.
430         :type month: int, list, str
431         :param day_of_week: Day of Week for execution. Either 0...6, [0...6, 0...6, ...] or :py:
    const:`crontab.ANY` for every Day of Week.
432         :type day_of_week: int, list, str
433         :param callback: The callback to be executed. The instance of :py:class:`cronjob` will be
     given as the first, args and kwargs as the following parameters.
434         :type callback: func
435
436         .. note:: The ``callback`` will be executed with it's instance of :py:class:`cronjob` as
    the first parameter.
437             The given Arguments (:data:`args`) and keyword Arguments (:data:`kwargs`) will be
    stored in that object.
438         """
439         self.__crontab__.append(self.cronjob(minute, hour, day_of_month, month, day_of_week,
    callback, *args, **kwargs))
```