Unittest for `task`

August 15, 2025

Contents

1 Test Information

1.1 Test Candidate 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.

Library Information	
Name	task
State	Released
Supported Interpreters	python3
Version	dfaed91376075c069c9e784b77342f24
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	0de92de1eb874ac24955dd6f67631bee
Testruns with	python 3.13.5 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	Debian GNU/Linux 13 trixie
Hostname	ahorn
Kernel	6.12.38+deb13-amd64 (#1 SMP PREEMPT_DYNAMIC Debian 6.12.38-1 (2025-07-16))
Machine	x86_64
Path	/home/dirk/work/unittest_collection/task
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun with python 3.13.5 (final)

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	217.022s

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 with python 3.13.5 (final)

3.1.1 pylibs.task.crontab: Test cronjob

Testresult

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:32,306
Finished-Time:	2025-08-15 21:03:32,321
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'>).

3.1.2 pylibs.task.crontab: Test crontab

Testresult

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

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:32,322
Finished-Time:	2025-08-15 21:07:02,327
Time-Consumption	210.005s

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.

3.1.3 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:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:25,003
Finished-Time:	2025-08-15 21:03:25,516
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 0.2503364086151123 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 0.010375022888183594 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 0.005130767822265625 in [0.00395 ... 0.00705] and Type is <class 'float'>).

3.1.4 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:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:25,516
Finished-Time:	2025-08-15 21:03:28,061
Time-Consumption	2.545s

Testsummary:

Info	Running a periodic task for 10 cycles with a cycletime of 0.25s
Success	Minimum cycle time is correct (Content 0.2503952980041504 in [0.2465 ... 0.2545] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.25099709298875594 in [0.2465 ... 0.2545] and Type is <class 'float'>).
Success	Maximum cycle time is correct (Content 0.2532165050506592 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 0.010394811630249023 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.010824150509304471 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).
Success	Maximum cycle time is correct (Content 0.012276411056518555 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 0.005440473556518555 in [0.00395 ... 0.00705] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.005537403954399956 in [0.00395 ... 0.00705] and Type is <class 'float'>).
Success	Maximum cycle time is correct (Content 0.0055887699127197266 in [0.00395 ... 0.009049999999999999] and Type is <class 'float'>).

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:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:28,276
Finished-Time:	2025-08-15 21:03:28,282
Time-Consumption	0.006s

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.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:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:28,061
Finished-Time:	2025-08-15 21:03:28,168
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.

3.1.7 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:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:28,168
Finished-Time:	2025-08-15 21:03:28,275
Time-Consumption	0.107s

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.

3.1.8 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:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:31,401
Finished-Time:	2025-08-15 21:03:32,009
Time-Consumption	0.608s

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.

3.1.9 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:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/task/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:03:28,283
Finished-Time:	2025-08-15 21:03:31,400
Time-Consumption	3.118s

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.

A Trace for testrun with python 3.13.5 (final)

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 0.2503364086151123 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Time consumption): 0.2503364086151123 (<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 0.010375022888183594 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Time consumption): 0.010375022888183594 (<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 0.005130767822265625 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Time consumption): 0.005130767822265625 (<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 1755284605.517322
Task execution number 2 at 1755284605.768132
Task execution number 3 at 1755284606.018527
Task execution number 4 at 1755284606.269294
Task execution number 5 at 1755284606.519991
Task execution number 6 at 1755284606.770860
Task execution number 7 at 1755284607.021444
```

```
Task execution number 8 at 1755284607.272279
Task execution number 9 at 1755284607.523079
Task execution number 10 at 1755284607.776295
```

Success Minimum cycle time is correct (Content 0.2503952980041504 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.2503952980041504 (<class 'float'>)
Expectation (Minimum cycle time): 0.2465 <= result <= 0.2545
```

Success Mean cycle time is correct (Content 0.25099709298875594 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.25099709298875594 (<class 'float'>)
Expectation (Mean cycle time): 0.2465 <= result <= 0.2545
```

Success Maximum cycle time is correct (Content 0.2532165050506592 in [0.2465 ... 0.2565] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.2532165050506592 (<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 1755284607.826881
Task execution number 2 at 1755284607.837669
Task execution number 3 at 1755284607.848182
Task execution number 4 at 1755284607.860459
Task execution number 5 at 1755284607.871138
Task execution number 6 at 1755284607.882188
Task execution number 7 at 1755284607.892791
Task execution number 8 at 1755284607.903185
Task execution number 9 at 1755284607.913738
Task execution number 10 at 1755284607.924298
```

Success Minimum cycle time is correct (Content 0.010394811630249023 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.010394811630249023 (<class 'float'>)
Expectation (Minimum cycle time): 0.008900000000000002 <= result <= 0.0121
```

Success Mean cycle time is correct (Content 0.010824150509304471 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.010824150509304471 (<class 'float'>)
Expectation (Mean cycle time): 0.008900000000000002 <= result <= 0.0121
```

Success	Maximum cycle time is correct (Content 0.012276411056518555 in [0.008900000000000002 ... 0.0141] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.012276411056518555 (<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 1755284607.949154
Task execution number 2 at 1755284607.954714
Task execution number 3 at 1755284607.960250
Task execution number 4 at 1755284607.965772
Task execution number 5 at 1755284607.971321
Task execution number 6 at 1755284607.976910
Task execution number 7 at 1755284607.982350
Task execution number 8 at 1755284607.987925
Task execution number 9 at 1755284607.993436
Task execution number 10 at 1755284607.998991
```

Success	Minimum cycle time is correct (Content 0.005440473556518555 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.005440473556518555 (<class 'float'>)
Expectation (Minimum cycle time): 0.00395 <= result <= 0.00705
```

Success	Mean cycle time is correct (Content 0.005537403954399956 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.005537403954399956 (<class 'float'>)
Expectation (Mean cycle time): 0.00395 <= result <= 0.00705
```

Success	Maximum cycle time is correct (Content 0.0055887699127197266 in [0.00395 ... 0.009049999999999999] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.0055887699127197266 (<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 1755284642s, requested for 1755284640s
Crontab execution number 2 at 1755284762s, requested for 1755284760s
Result (Timing of crontasks): [ 1755284642, 1755284762 ] (<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): 1755284642 (<class 'int'>)
Expectation (Submitted value number 1): 1755284640 <= result <= 1755284671
Submitted value number 1 is correct (Content 1755284642 in [1755284640 ... 1755284671] and
↪  Type is <class 'int'>).
Result (Submitted value number 2): 1755284762 (<class 'int'>)
Expectation (Submitted value number 2): 1755284760 <= result <= 1755284791
Submitted value number 2 is correct (Content 1755284762 in [1755284760 ... 1755284791] 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 #!/usr/bin/env python
2 # -*- coding: UTF-8 -*-
3
4 """
5 task (Task Module)
6 ==================
```

```
7
8  **Author:**
9
10 * Dirk Alders <sudo-dirk@mount-mockery.de>
11
12 **Description:**
13
14     This Module supports helpfull classes for queues, tasks, ...
15
16 **Submodules:**
17
18 * :class:`task.crontab`
19 * :class:`task.delayed`
20 * :class:`task.periodic`
21 * :class:`task.queue`
22 * :class:`task.threaded_queue`
23
24 **Unittest:**
25
26         See also the :download:`unittest <task/_testresults_/unittest.pdf>` documentation.
27
28 **Module Documentation:**
29
30 """
31 __DEPENDENCIES__ = []
32
33 import logging
34 import threading
35 import time
36 from queue import PriorityQueue
37 from queue import Empty
38
39 try:
40     from config import APP_NAME as ROOT_LOGGER_NAME
41 except ImportError:
42     ROOT_LOGGER_NAME = 'root'
43 logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)
44
45 __DESCRIPTION__ = """The Module {\\tt %s} is designed to help with task issues like periodic
    tasks, delayed tasks, queues, threaded queues and crontabs.
46 For more Information read the documentation.""" % __name__.replace('_', '\\_')
47 """The Module Description"""
48 __INTERPRETER__ = (3, )
49 """The Tested Interpreter-Versions"""
50
51
52 class queue(object):
53     """
54     Class to execute queued callbacks.
55
56     :param bool expire: The default value for expire. See also :py:func:`expire`.
57
58     **Example:**
59
60     .. literalinclude:: task/_examples_/tqueue.py
61
62     Will result to the following output:
63
64     .. literalinclude:: task/_examples_/tqueue.log
65     """
```

```
66      class job(object):
67          def __init__(self, priority, callback, *args, **kwargs):
68              self.time = time.time()
69              self.priority = priority
70              self.callback = callback
71              self.args = args
72              self.kwargs = kwargs
73
74          def run(self, queue):
75              self.callback(queue, *self.args, **self.kwargs)
76
77          def __lt__(self, other):
78              if self.priority != other.priority:
79                  return self.priority < other.priority
80              else:
81                  return self.time < other.time
82
83      def __init__(self, expire=True):
84          self.__expire = expire
85          self.__stop = False
86          self.queue = PriorityQueue()
87
88      def clean_queue(self):
89          """
90          This Methods removes all jobs from the queue.
91
92          .. note:: Be aware that already running jobs will not be terminated.
93          """
94          while not self.queue.empty():
95              try:
96                  self.queue.get(False)
97              except Empty:           # This block is hard to reach for a testcase, but is
98                  continue            # needed, if the thread runs dry while cleaning the queue.
99              self.queue.task_done()
100
101     def enqueue(self, priority, callback, *args, **kwargs):
102         """
103         This enqueues a given callback.
104
105         :param number priority: The priority indication number of this task. The lowest value
    will be queued first.
106         :param callback callback: Callback to be executed
107         :param args args: Arguments to be given to callback
108         :param kwargs kwargs: Keword Arguments to be given to callback
109
110         .. note:: Callback will get this instance as first argument, followed by :py:data:`args`
    und :py:data:`kwargs`.
111         """
112         self.queue.put(self.job(priority, callback, *args, **kwargs))
113
114     def qsize(self):
115         return self.queue.qsize()
116
117     def run(self):
118         """
119         This starts the execution of the queued callbacks.
120         """
121         self.__stop = False
122         while not self.__stop:
123             try:
124                 self.queue.get(timeout=0.1).run(self)
125             except Empty:
```

```
126                 if self.__expire:
127                     break
128         if type(self) is threaded_queue:
129             self.thread = None
130 
131     def expire(self):
132         """
133         This sets the expire flag. That means that the process will stop after queue gets empty.
134         """
135         self.__expire = True
136 
137     def stop(self):
138         """
139         This sets the stop flag. That means that the process will stop after finishing the active
     task.
140         """
141         self.__stop = True
142 
143 
144 class threaded_queue(queue):
145     """Class to execute queued callbacks in a background thread (See also parent :py:class:`queue
    `).
146 
147     :param bool expire: The default value for expire. See also :py:func:`queue.expire`.
148 
149     **Example:**
150 
151     .. literalinclude:: task/_examples_/threaded_queue.py
152 
153     Will result to the following output:
154 
155     .. literalinclude:: task/_examples_/threaded_queue.log
156     """
157 
158     def __init__(self, expire=False):
159         queue.__init__(self, expire=expire)
160         self.thread = None
161 
162     def run(self):
163         if self.thread is None:
164             self.thread = threading.Thread(target=self._start, args=(), daemon=True)
165             self.thread.daemon = True    # Daemonize thread
166             self.thread.start()          # Start the execution
167 
168     def join(self):
169         """
170         This blocks till the queue is empty.
171 
172         .. note:: If the queue does not run dry, join will block till the end of the days.
173         """
174         self.expire()
175         if self.thread is not None:
176             self.thread.join()
177 
178     def stop(self):
179         queue.stop(self)
180         self.join()
181 
182     def _start(self):
183         queue.run(self)
184 
185 
186 class periodic(object):
```

```
187     """
188     Class to execute a callback cyclicly.
189
190     :param float cycle_time: Cycle time in seconds -- callback will be executed every *cycle_time
        * seconds
191     :param callback callback: Callback to be executed
192     :param args args: Arguments to be given to the callback
193     :param kwargs kwargs: Keword Arguments to be given to callback
194
195     .. note:: The Callback will get this instance as first argument, followed by :py:data:`args`
        und :py:data:`kwargs`.
196
197     **Example:**
198
199     .. literalinclude:: task/_examples_/periodic.py
200
201     Will result to the following output:
202
203     .. literalinclude:: task/_examples_/periodic.log
204     """
205
206     def __init__(self, cycle_time, callback, *args, **kwargs):
207         self._lock = threading.Lock()
208         self._timer = None
209         self.callback = callback
210         self.cycle_time = cycle_time
211         self.args = args
212         self.kwargs = kwargs
213         self._stopped = True
214         self._last_tm = None
215         self.dt = None
216
217     def join(self):
218         """
219         This blocks till the cyclic task is terminated.
220
221         .. note:: Using join means that somewhere has to be a condition calling :py:func:`stop`
        to terminate. Otherwise :func:`task.join` will never return.
222         """
223         while not self._stopped:
224             time.sleep(.1)
225
226     def run(self):
227         """
228         This starts the cyclic execution of the given callback.
229         """
230         if self._stopped:
231             self._set_timer(force_now=True)
232
233     def stop(self):
234         """
235         This stops the execution of any further task.
236         """
237         self._lock.acquire()
238         self._stopped = True
239         if self._timer is not None:
240             self._timer.cancel()
241         self._lock.release()
242
243     def _set_timer(self, force_now=False):
```

```
244         """
245         This sets the timer for the execution of the next task.
246         """
247         self._lock.acquire()
248         self._stopped = False
249         if force_now:
250             self._timer = threading.Timer(0, self._start)
251         else:
252             self._timer = threading.Timer(self.cycle_time, self._start)
253         self._timer.daemon = True
254         self._timer.start()
255         self._lock.release()
256 
257     def _start(self):
258         tm = time.time()
259         if self._last_tm is not None:
260             self.dt = tm - self._last_tm
261         self._set_timer(force_now=False)
262         self.callback(self, *self.args, **self.kwargs)
263         self._last_tm = tm
264 
265 
266 class delayed(periodic):
267     """Class to execute a callback a given time in the future. See also parent :py:class:`
    periodic`.
268 
269     :param float time: Delay time for execution of the given callback
270     :param callback callback: Callback to be executed
271     :param args args: Arguments to be given to callback
272     :param kwargs kwargs: Keword Arguments to be given to callback
273 
274     **Example:**
275 
276     .. literalinclude:: task/_examples_/delayed.py
277 
278     Will result to the following output:
279 
280     .. literalinclude:: task/_examples_/delayed.log
281     """
282 
283     def run(self):
284         """
285         This starts the timer for the delayed execution.
286         """
287         self._set_timer(force_now=False)
288 
289     def _start(self):
290         self.callback(*self.args, **self.kwargs)
291         self.stop()
292 
293 
294 class crontab(periodic):
295     """Class to execute a callback at the specified time conditions. See also parent :py:class:`
    periodic`.
296 
297     :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.
298     :type accuracy: float
299 
300     **Example:**
301 
302     .. literalinclude:: task/_examples_/crontab.py
```

```
303
304     Will result to the following output:
305
306     .. literalinclude:: task/_examples_/crontab.log
307     """
308     ANY = '*'
309     """Constant for matching every condition."""
310
311     class cronjob(object):
312         """Class to handle cronjob parameters and cronjob changes.
313
314         :param minute: Minute for execution. Either 0...59, [0...59, 0...59, ...] or :py:const:`crontab.ANY` for every Minute.
315         :type minute: int, list, str
316         :param hour: Hour for execution. Either 0...23, [0...23, 0...23, ...] or :py:const:`crontab.ANY` for every Hour.
317         :type hour: int, list, str
318         :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.
319         :type day_of_month: int, list, str
320         :param month: Month for execution. Either 0...12, [0...12, 0...12, ...] or :py:const:`crontab.ANY` for every Month.
321         :type month: int, list, str
322         :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.
323         :type day_of_week: int, list, str
324         :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.
325         :type callback: func
326
327         .. 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()`.
328         """
329         class all_match(set):
330             """Universal set - match everything"""
331
332             def __contains__(self, item):
333                 (item)
334                 return True
335
336         def __init__(self, minute, hour, day_of_month, month, day_of_week, callback, *args, **kwargs):
337             self.set_trigger_conditions(minute or crontab.ANY, hour or crontab.ANY,
338                                         day_of_month or crontab.ANY, month or crontab.ANY, day_of_week or crontab.ANY)
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 minute in self.minute and hour in self.hour and day_of_month in self.
    day_of_month and month in self.month and day_of_week in self.day_of_week:
381                     return True
382             return False
383 
384         def __store_execution_reminder__(self, minute, hour, day_of_month, month, day_of_week):
385             self.__last_execution__ = [minute, hour, day_of_month, month, day_of_week]
386 
387         def cron_execution(self, tm):
388             """This Methods executes the Cron-Callback, if a execution is needed for the given
    time (depending on the parameters on initialisation)
389 
390             :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())).
391             :type tm: int
392             """
393             if self.__last_cron_check_time__ is None:
394                 self.__last_cron_check_time__ = tm - 1
395             #
396             for t in range(self.__last_cron_check_time__ + 1, tm + 1):
397                 lt = time.localtime(t)
398                 if self.__execution_needed_for__(lt[4], lt[3], lt[2], lt[1], lt[6]):
399                     self.callback(self, *self.args, **self.kwargs)
400                     self.__store_execution_reminder__(lt[4], lt[3], lt[2], lt[1], lt[6])
401                     break
402             self.__last_cron_check_time__ = tm
403 
404     def __init__(self, accuracy=30):
405         periodic.__init__(self, accuracy, self.__periodic__)
406         self.__crontab__ = []
407 
```

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