Unittest for `task`

February 28, 2021

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	python2, python3
Version	d03c7bd7995b3c967ec523eaddf376d5
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	0de92de1eb874ac24955dd6f67631bee
Testruns with	python 2.7.18 (final), python 3.8.5 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	Linux Mint 20.1 ulyssa
Hostname	erle
Kernel	5.8.0-44-generic (#50 20.04.1-Ubuntu SMP Wed Feb 10 21:07:30 UTC 2021)
Machine	x86_64
Path	/usr/data/dirk/prj/unittest/task/unittest
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun with python 2.7.18 (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	217.150s

2.2 Test-Statistic for testrun with python 3.8.5 (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	217.137s

2.3 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
`task`	98.9%	98.1%
`task.__init__.py`	98.9%	

3 Testcases with no corresponding Requirement

3.1 Summary for testrun with python 2.7.18 (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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (28)
Start-Time:	2021-02-28 18:49:22,995
Finished-Time:	2021-02-28 18:49:23,014
Time-Consumption	0.018s

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 <type 'bool'>).
Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content True and Type is <type 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <type '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 <type 'bool'>).
Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content True and Type is <type 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 3 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <type '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 <type 'bool'>).
Success	Return value for minute: 45; hour: 12; day: 25; month: 03, day_of_week: 5 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 22; hour: 17; day: 25; month: 02, day_of_week: 1 is correct (Content True and Type is <type 'bool'>).

Success	Return value for minute: 22; hour: 17; day: 25; month: 05, day_of_week: 3 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 45; hour: 14; day: 25; month: 02, day_of_week: 1 is correct (Content False and Type is <type 'bool'>).
Success	Return value for minute: 23; hour: 17; day: 24; month: 02, day_of_week: 1 is correct (Content False and Type is <type 'bool'>).
Info	Resetting trigger condition (again).
Success	1st run - execution not needed is correct (Content False and Type is <type 'bool'>).
Success	2nd run - execution not needed is correct (Content False and Type is <type 'bool'>).
Success	3rd run - execution needed is correct (Content True and Type is <type 'bool'>).
Success	4th run - execution needed is correct (Content True and Type is <type 'bool'>).
Success	5th run - execution not needed is correct (Content False and Type is <type 'bool'>).
Success	6th run - execution not needed is correct (Content False and Type is <type '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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (29)
Start-Time:	2021-02-28 18:49:23,015
Finished-Time:	2021-02-28 18:52:53,120
Time-Consumption	210.106s
Testsummary:	
Info	Creating Crontab with callback execution in +1 and +3 minutes.
Success	Number of submitted values is correct (Content 2 and Type is <type '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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (21)
Start-Time:	2021-02-28 18:49:15,668
Finished-Time:	2021-02-28 18:49:16,184
Time-Consumption	0.516s
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 0.25023794174194336 in [0.2465 ... 0.2545] and Type is <type '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 0.010259866714477539 in [0.008900000000000002 ... 0.0121] and Type is <type '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 0.005254983901977539 in [0.00395 ... 0.00705] and Type is <type '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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (22)
Start-Time:	2021-02-28 18:49:16,184
Finished-Time:	2021-02-28 18:49:18,736
Time-Consumption	2.551s

Testsummary:

Info	Running a periodic task for 10 cycles with a cycletime of 0.25s
Success	Minimum cycle time is correct (Content 0.251039981842041 in [0.2465 ... 0.2545] and Type is <type 'float'>).
Success	Mean cycle time is correct (Content 0.2511719862620036 in [0.2465 ... 0.2545] and Type is <type 'float'>).
Success	Maximum cycle time is correct (Content 0.2513270378112793 in [0.2465 ... 0.2565] and Type is <type 'float'>).
Info	Running a periodic task for 10 cycles with a cycletime of 0.01s
Success	Minimum cycle time is correct (Content 0.010766983032226562 in [0.008900000000000002 ... 0.0121] and Type is <type 'float'>).
Success	Mean cycle time is correct (Content 0.011101775699191622 in [0.008900000000000002 ... 0.0121] and Type is <type 'float'>).
Success	Maximum cycle time is correct (Content 0.011808156967163086 in [0.008900000000000002 ... 0.0141] and Type is <type 'float'>).
Info	Running a periodic task for 10 cycles with a cycletime of 0.005s
Success	Minimum cycle time is correct (Content 0.0056459903717041016 in [0.00395 ... 0.00705] and Type is <type 'float'>).
Success	Mean cycle time is correct (Content 0.005896435843573676 in [0.00395 ... 0.00705] and Type is <type 'float'>).
Success	Maximum cycle time is correct (Content 0.006392955780029297 in [0.00395 ... 0.009049999999999999] and Type is <type '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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (25)
Start-Time:	2021-02-28 18:49:18,953
Finished-Time:	2021-02-28 18:49:18,959
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 <type 'int'>).
Success	Size of Queue after execution is correct (Content 3 and Type is <type '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 <type '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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (23)
Start-Time:	2021-02-28 18:49:18,736
Finished-Time:	2021-02-28 18:49:18,844
Time-Consumption	0.107s

Testsummary:

Info	Enqueued 6 unordered tasks.
Success	Size of Queue before execution is correct (Content 6 and Type is <type 'int'>).
Success	Size of Queue after execution is correct (Content 0 and Type is <type '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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (24)
Start-Time:	2021-02-28 18:49:18,844
Finished-Time:	2021-02-28 18:49:18,952
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 <type 'int'>).

Success	Size of Queue after 1st execution is correct (Content 2 and Type is <type '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 <type '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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (27)
Start-Time:	2021-02-28 18:49:22,088
Finished-Time:	2021-02-28 18:49:22,698
Time-Consumption	0.610s
Testsummary:	
Success	Size of Queue before execution is correct (Content 0 and Type is <type 'int'>).
Info	Enqueued 2 tasks.
Success	Size of Queue after execution is correct (Content 0 and Type is <type '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 2.7.18 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (26)
Start-Time:	2021-02-28 18:49:18,960
Finished-Time:	2021-02-28 18:49:22,087
Time-Consumption	3.128s
Testsummary:	
Info	Enqueued 6 unordered tasks.
Success	Size of Queue before execution is correct (Content 7 and Type is <type 'int'>).
Info	Executing Queue, till Queue is empty..
Success	Size of Queue after execution is correct (Content 0 and Type is <type '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 <type '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.

3.2 Summary for testrun with python 3.8.5 (final)

3.2.1 pylibs.task.crontab: Test cronjob

Testresult

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

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (28)
Start-Time:	2021-02-28 18:53:00,963
Finished-Time:	2021-02-28 18:53:00,977
Time-Consumption	0.014s

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'>).

3.2.2 pylibs.task.crontab: Test crontab

Testresult

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

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (29)
Start-Time:	2021-02-28 18:53:00,978
Finished-Time:	2021-02-28 18:56:31,082
Time-Consumption	210.104s
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.

3.2.3 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 B.1.1!

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (21)
Start-Time:	2021-02-28 18:52:53,644
Finished-Time:	2021-02-28 18:52:54,157
Time-Consumption	0.513s
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 0.2501850128173828 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 0.010187149047851562 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 0.005234718322753906 in [0.00395 ... 0.00705] and Type is <class 'float'>).

3.2.4 pylibs.task.periodic: Test periodic execution

Testresult

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

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (22)
Start-Time:	2021-02-28 18:52:54,158
Finished-Time:	2021-02-28 18:52:56,708
Time-Consumption	2.550s
Testsummary:	
Info	Running a periodic task for 10 cycles with a cycletime of 0.25s
Success	Minimum cycle time is correct (Content 0.25077366828918457 in [0.2465 ... 0.2545] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.2510095172458225 in [0.2465 ... 0.2545] and Type is <class 'float'>).
Success	Maximum cycle time is correct (Content 0.25113654136657715 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 0.010764360427856445 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.010903808805677626 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).
Success	Maximum cycle time is correct (Content 0.011025190353393555 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 0.0057222843170166016 in [0.00395 ... 0.00705] and Type is <class 'float'>).
Success	Mean cycle time is correct (Content 0.005827241473727756 in [0.00395 ... 0.00705] and Type is <class 'float'>).
Success	Maximum cycle time is correct (Content 0.005957841873168945 in [0.00395 ... 0.009049999999999999] and Type is <class 'float'>).

3.2.5 pylibs.task.queue: Test clean_queue method

Testresult

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

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (25)

Start-Time:	2021-02-28 18:52:56,925
Finished-Time:	2021-02-28 18:52:56,930
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'>).

3.2.6 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 B.1.3!

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (23)
Start-Time:	2021-02-28 18:52:56,709
Finished-Time:	2021-02-28 18:52:56,816
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.

3.2.7 pylibs.task.queue: Test stop method

Testresult

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

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (24)
Start-Time:	2021-02-28 18:52:56,816
Finished-Time:	2021-02-28 18:52:56,924
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.

3.2.8 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 B.1.7!

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (27)
Start-Time:	2021-02-28 18:53:00,057
Finished-Time:	2021-02-28 18:53:00,666
Time-Consumption	0.610s
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.

3.2.9 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 B.1.6!

Testrun:	python 3.8.5 (final)
Caller:	/usr/data/dirk/prj/unittest/task/unittest/src/tests/__init__.py (26)
Start-Time:	2021-02-28 18:52:56,930
Finished-Time:	2021-02-28 18:53:00,056
Time-Consumption	3.126s
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 2.7.18 (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 ] (<type 'list'>)
Expectation (Execution of task and delayed task (identified by a submitted sequence number)):
↪ result = [ 1, 2 ] (<type 'list'>)
Result (Submitted value number 1): 1 (<type 'int'>)
Expectation (Submitted value number 1): result = 1 (<type 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <type 'int'>).
Result (Submitted value number 2): 2 (<type 'int'>)
Expectation (Submitted value number 2): result = 2 (<type 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <type 'int'>).
```

Success Time consumption is correct (Content 0.25023794174194336 in [0.2465 ... 0.2545] and Type is <type 'float'>).

```
Result (Time consumption): 0.25023794174194336 (<type '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 ] (<type 'list'>)
Expectation (Execution of task and delayed task (identified by a submitted sequence number)):
↪ result = [ 1, 2 ] (<type 'list'>)
Result (Submitted value number 1): 1 (<type 'int'>)
Expectation (Submitted value number 1): result = 1 (<type 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <type 'int'>).
Result (Submitted value number 2): 2 (<type 'int'>)
```

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

Success Time consumption is correct (Content 0.010259866714477539 in [0.008900000000000002 ... 0.0121] and Type is <type 'float'>).

```
Result (Time consumption): 0.010259866714477539 (<type '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 ] (<type 'list'>)
Expectation (Execution of task and delayed task (identified by a submitted sequence number)):
↪  result = [ 1, 2 ] (<type 'list'>)
Result (Submitted value number 1): 1 (<type 'int'>)
Expectation (Submitted value number 1): result = 1 (<type 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <type 'int'>).
Result (Submitted value number 2): 2 (<type 'int'>)
Expectation (Submitted value number 2): result = 2 (<type 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <type 'int'>).
```

Success Time consumption is correct (Content 0.005254983901977539 in [0.00395 ... 0.00705] and Type is <type 'float'>).

```
Result (Time consumption): 0.005254983901977539 (<type '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 1614534556.186958
Task execution number 2 at 1614534556.438144
Task execution number 3 at 1614534556.689184
Task execution number 4 at 1614534556.940225
Task execution number 5 at 1614534557.191529
Task execution number 6 at 1614534557.442606
Task execution number 7 at 1614534557.693933
Task execution number 8 at 1614534557.945113
Task execution number 9 at 1614534558.196180
Task execution number 10 at 1614534558.447506
```

Success Minimum cycle time is correct (Content 0.251039981842041 in [0.2465 ... 0.2545] and Type is <type 'float'>).

```
Result (Minimum cycle time): 0.251039981842041 (<type 'float'>)
Expectation (Minimum cycle time): 0.2465 <= result <= 0.2545
```

Success Mean cycle time is correct (Content 0.2511719862620036 in [0.2465 ... 0.2545] and Type is <type 'float'>).

```
Result (Mean cycle time): 0.2511719862620036 (<type 'float'>)
Expectation (Mean cycle time): 0.2465 <= result <= 0.2545
```

Success Maximum cycle time is correct (Content 0.2513270378112793 in [0.2465 ... 0.2565] and Type is <type 'float'>).

```
Result (Maximum cycle time): 0.2513270378112793 (<type '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 1614534558.498764
Task execution number 2 at 1614534558.510531
Task execution number 3 at 1614534558.521298
Task execution number 4 at 1614534558.532262
Task execution number 5 at 1614534558.543040
Task execution number 6 at 1614534558.554051
Task execution number 7 at 1614534558.565859
Task execution number 8 at 1614534558.576854
Task execution number 9 at 1614534558.587765
Task execution number 10 at 1614534558.598680
```

Success Minimum cycle time is correct (Content 0.010766983032226562 in [0.008900000000000002 ... 0.0121] and Type is <type 'float'>).

```
Result (Minimum cycle time): 0.010766983032226562 (<type 'float'>)
Expectation (Minimum cycle time): 0.008900000000000002 <= result <= 0.0121
```

Success Mean cycle time is correct (Content 0.011101775699191622 in [0.008900000000000002 ... 0.0121] and Type is <type 'float'>).

```
Result (Mean cycle time): 0.011101775699191622 (<type 'float'>)
Expectation (Mean cycle time): 0.008900000000000002 <= result <= 0.0121
```

Success Maximum cycle time is correct (Content 0.011808156967163086 in [0.008900000000000002 ... 0.0141] and Type is <type 'float'>).

```
Result (Maximum cycle time): 0.011808156967163086 (<type '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 1614534558.623197
Task execution number 2 at 1614534558.629145
Task execution number 3 at 1614534558.634885
Task execution number 4 at 1614534558.640802
Task execution number 5 at 1614534558.646739
Task execution number 6 at 1614534558.652385
Task execution number 7 at 1614534558.658778
Task execution number 8 at 1614534558.664658
Task execution number 9 at 1614534558.670497
Task execution number 10 at 1614534558.676265
```

Success Minimum cycle time is correct (Content 0.0056459903717041016 in [0.00395 ... 0.00705] and Type is <type 'float'>).

```
Result (Minimum cycle time): 0.0056459903717041016 (<type 'float'>)
Expectation (Minimum cycle time): 0.00395 <= result <= 0.00705
```

Success Mean cycle time is correct (Content 0.005896435843573676 in [0.00395 ... 0.00705] and Type is <type 'float'>).

```
Result (Mean cycle time): 0.005896435843573676 (<type 'float'>)
Expectation (Mean cycle time): 0.00395 <= result <= 0.00705
```

Success Maximum cycle time is correct (Content 0.006392955780029297 in [0.00395 ... 0.009049999999999999] and Type is <type 'float'>).

```
Result (Maximum cycle time): 0.006392955780029297 (<type '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 <type 'int'>).

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

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

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

```
Result (Size of Queue after execution): 0 (<type 'int'>)
Expectation (Size of Queue after execution): result = 0 (<type '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 ]
↪  (<type 'list'>)
Expectation (Queue execution (identified by a submitted sequence number)): result = [ 1, 2,
↪  3, 5, 6, 7 ] (<type 'list'>)
Result (Submitted value number 1): 1 (<type 'int'>)
Expectation (Submitted value number 1): result = 1 (<type 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <type 'int'>).
Result (Submitted value number 2): 2 (<type 'int'>)
Expectation (Submitted value number 2): result = 2 (<type 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <type 'int'>).
Result (Submitted value number 3): 3 (<type 'int'>)
Expectation (Submitted value number 3): result = 3 (<type 'int'>)
Submitted value number 3 is correct (Content 3 and Type is <type 'int'>).
Result (Submitted value number 4): 5 (<type 'int'>)
Expectation (Submitted value number 4): result = 5 (<type 'int'>)
Submitted value number 4 is correct (Content 5 and Type is <type 'int'>).
Result (Submitted value number 5): 6 (<type 'int'>)
Expectation (Submitted value number 5): result = 6 (<type 'int'>)
Submitted value number 5 is correct (Content 6 and Type is <type 'int'>).
Result (Submitted value number 6): 7 (<type 'int'>)
Expectation (Submitted value number 6): result = 7 (<type 'int'>)
Submitted value number 6 is correct (Content 7 and Type is <type '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 <type 'int'>).

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

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

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

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

```
Result (Size of Queue after 2nd execution): 0 (<type 'int'>)
Expectation (Size of Queue after 2nd execution): result = 0 (<type '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 ]
↪  (<type 'list'>)
Expectation (Queue execution (2nd part; identified by a submitted sequence number)): result =
↪  [ 6, 7 ] (<type 'list'>)
Result (Submitted value number 1): 6 (<type 'int'>)
Expectation (Submitted value number 1): result = 6 (<type 'int'>)
Submitted value number 1 is correct (Content 6 and Type is <type 'int'>).
Result (Submitted value number 2): 7 (<type 'int'>)
Expectation (Submitted value number 2): result = 7 (<type 'int'>)
Submitted value number 2 is correct (Content 7 and Type is <type '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 <type 'int'>).

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

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

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

Info Cleaning Queue.

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

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

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

```
Result (Size of Queue after execution): 0 (<type 'int'>)
Expectation (Size of Queue after execution): result = 0 (<type '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 ] (<type 'list'>)
Expectation (Queue execution (identified by a submitted sequence number)): result = [ 1, 2,
↪  3, 5.1, 5.2, 6, 7 ] (<type 'list'>)
Result (Submitted value number 1): 1 (<type 'int'>)
Expectation (Submitted value number 1): result = 1 (<type 'int'>)
Submitted value number 1 is correct (Content 1 and Type is <type 'int'>).
Result (Submitted value number 2): 2 (<type 'int'>)
Expectation (Submitted value number 2): result = 2 (<type 'int'>)
Submitted value number 2 is correct (Content 2 and Type is <type 'int'>).
Result (Submitted value number 3): 3 (<type 'int'>)
Expectation (Submitted value number 3): result = 3 (<type 'int'>)
```

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

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

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

```
Result (Size of Queue after execution): 0 (<type 'int'>)
Expectation (Size of Queue after execution): result = 0 (<type '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 ] (<type
↪  'list'>)
Expectation (Queue execution (identified by a submitted sequence number)): result = [ 6, 1,
↪  2, 3 ] (<type 'list'>)
Result (Submitted value number 1): 6 (<type 'int'>)
Expectation (Submitted value number 1): result = 6 (<type 'int'>)
Submitted value number 1 is correct (Content 6 and Type is <type 'int'>).
Result (Submitted value number 2): 1 (<type 'int'>)
Expectation (Submitted value number 2): result = 1 (<type 'int'>)
Submitted value number 2 is correct (Content 1 and Type is <type 'int'>).
Result (Submitted value number 3): 2 (<type 'int'>)
Expectation (Submitted value number 3): result = 2 (<type 'int'>)
Submitted value number 3 is correct (Content 2 and Type is <type 'int'>).
Result (Submitted value number 4): 3 (<type 'int'>)
Expectation (Submitted value number 4): result = 3 (<type 'int'>)
Submitted value number 4 is correct (Content 3 and Type is <type '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 <type 'bool'>).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Info Resetting trigger condition (again).

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

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

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

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

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

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

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

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

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

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

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

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

```
Crontab accuracy is 30s
Crontab execution number 1 at 1614534623s, requested for 1614534600s
Crontab execution number 2 at 1614534743s, requested for 1614534720s
Result (Timing of crontasks): [ 1614534623, 1614534743 ] (<type 'list'>)
Result (Number of submitted values): 2 (<type 'int'>)
Expectation (Number of submitted values): result = 2 (<type 'int'>)
```

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

```
Result (Submitted value number 1): 1614534623 (<type 'int'>)
Expectation (Submitted value number 1): 1614534600 <= result <= 1614534631
Submitted value number 1 is correct (Content 1614534623 in [1614534600 ... 1614534631] and
↪ Type is <type 'int'>).
Result (Submitted value number 2): 1614534743 (<type 'int'>)
Expectation (Submitted value number 2): 1614534720 <= result <= 1614534751
Submitted value number 2 is correct (Content 1614534743 in [1614534720 ... 1614534751] and
↪ Type is <type 'int'>).
```

B Trace for testrun with python 3.8.5 (final)

B.1 Tests with status Info (9)

B.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 0.2501850128173828 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Time consumption): 0.2501850128173828 (<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 0.010187149047851562 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Time consumption): 0.010187149047851562 (<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 0.005234718322753906 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Time consumption): 0.005234718322753906 (<class 'float'>)
Expectation (Time consumption): 0.00395 <= result <= 0.00705
```

B.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 1614534774.160674
Task execution number 2 at 1614534774.411448
Task execution number 3 at 1614534774.662584
Task execution number 4 at 1614534774.913616
Task execution number 5 at 1614534775.164559
Task execution number 6 at 1614534775.415640
Task execution number 7 at 1614534775.666654
Task execution number 8 at 1614534775.917708
Task execution number 9 at 1614534776.168775
Task execution number 10 at 1614534776.419760
```

Success	Minimum cycle time is correct (Content 0.25077366828918457 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.25077366828918457 (<class 'float'>)
Expectation (Minimum cycle time): 0.2465 <= result <= 0.2545
```

Success	Mean cycle time is correct (Content 0.2510095172458225 in [0.2465 ... 0.2545] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.2510095172458225 (<class 'float'>)
Expectation (Mean cycle time): 0.2465 <= result <= 0.2545
```

Success Maximum cycle time is correct (Content 0.25113654136657715 in [0.2465 ... 0.2565] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.25113654136657715 (<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 1614534776.472581
Task execution number 2 at 1614534776.483465
Task execution number 3 at 1614534776.494432
Task execution number 4 at 1614534776.505340
Task execution number 5 at 1614534776.516291
Task execution number 6 at 1614534776.527113
Task execution number 7 at 1614534776.538138
Task execution number 8 at 1614534776.549007
Task execution number 9 at 1614534776.559951
Task execution number 10 at 1614534776.570715
```

Success Minimum cycle time is correct (Content 0.010764360427856445 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.010764360427856445 (<class 'float'>)
Expectation (Minimum cycle time): 0.008900000000000002 <= result <= 0.0121
```

Success Mean cycle time is correct (Content 0.010903808805677626 in [0.008900000000000002 ... 0.0121] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.010903808805677626 (<class 'float'>)
Expectation (Mean cycle time): 0.008900000000000002 <= result <= 0.0121
```

Success Maximum cycle time is correct (Content 0.011025190353393555 in [0.008900000000000002 ... 0.0141] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.011025190353393555 (<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 1614534776.596651
Task execution number 2 at 1614534776.602505
Task execution number 3 at 1614534776.608362
Task execution number 4 at 1614534776.614205
Task execution number 5 at 1614534776.619941
Task execution number 6 at 1614534776.625721
Task execution number 7 at 1614534776.631444
Task execution number 8 at 1614534776.637402
Task execution number 9 at 1614534776.643209
Task execution number 10 at 1614534776.649096
```

Success	Minimum cycle time is correct (Content 0.0057222843170166016 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Minimum cycle time): 0.0057222843170166016 (<class 'float'>)
Expectation (Minimum cycle time): 0.00395 <= result <= 0.00705
```

Success	Mean cycle time is correct (Content 0.005827241473727756 in [0.00395 ... 0.00705] and Type is <class 'float'>).

```
Result (Mean cycle time): 0.005827241473727756 (<class 'float'>)
Expectation (Mean cycle time): 0.00395 <= result <= 0.00705
```

Success	Maximum cycle time is correct (Content 0.005957841873168945 in [0.00395 ... 0.009049999999999999] and Type is <class 'float'>).

```
Result (Maximum cycle time): 0.005957841873168945 (<class 'float'>)
Expectation (Maximum cycle time): 0.00395 <= result <= 0.009049999999999999
```

B.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'>).
```

B.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'>).
```

B.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'>)
```

B.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'>).
```

B.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'>).
```

B.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'>)
```

B.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 1614534840s, requested for 1614534840s
Crontab execution number 2 at 1614534960s, requested for 1614534960s
Result (Timing of crontasks): [ 1614534840, 1614534960 ] (<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): 1614534840 (<class 'int'>)
Expectation (Submitted value number 1): 1614534840 <= result <= 1614534871
Submitted value number 1 is correct (Content 1614534840 in [1614534840 ... 1614534871] and
↪ Type is <class 'int'>).
Result (Submitted value number 2): 1614534960 (<class 'int'>)
Expectation (Submitted value number 2): 1614534960 <= result <= 1614534991
Submitted value number 2 is correct (Content 1614534960 in [1614534960 ... 1614534991] and
↪ Type is <class 'int'>).
```

C Test-Coverage

C.1 task

The line coverage for task was 98.9%
The branch coverage for task was 98.1%

C.1.1 task.__init__.py

The line coverage for task.__init__.py was 98.9%
The branch coverage for task.__init__.py was 98.1%

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

"""
task (Task Module)
==================

**Author:**

* 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:**

"""
__DEPENDENCIES__ = []

import logging
import sys
import threading
import time
if sys.version_info >= (3, 0):
    from queue import PriorityQueue
    from queue import Empty
else:
    from Queue import PriorityQueue
    from Queue import Empty

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"""
__INTERPRETER__ = (2, 3)
"""The Tested Interpreter-Versions"""


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

        def run(self, queue):
            self.callback(queue, *self.args, **self.kwargs)

        def __lt__(self, other):
            if self.priority != other.priority:
                return self.priority < other.priority
            else:
                return self.time < other.time

    def __init__(self, expire=True):
        self.__expire = expire
        self.__stop = False
        self.queue = PriorityQueue()

    def clean_queue(self):
        """
        This Methods removes all jobs from the queue.

        .. note:: Be aware that already running jobs will not be terminated.
        """
        while not self.queue.empty():
            try:
                self.queue.get(False)
            except Empty:           # This block is hard to reach for a testcase, but is
                continue            # needed, if the thread runs dry while cleaning the queue.
            self.queue.task_done()

    def enqueue(self, priority, callback, *args, **kwargs):
        """
        This enqueues a given callback.

        :param number priority: The priority indication number of this task. The lowest value
    will be queued first.
        :param callback callback: Callback to be executed
        :param args args: Arguments to be given to callback
        :param kwargs kwargs: Keword Arguments to be given to callback

        .. note:: Callback will get this instance as first argument, followed by :py:data:`args`
    und :py:data:`kwargs`.
        """
        self.queue.put(self.job(priority, callback, *args, **kwargs))

    def qsize(self):
        return self.queue.qsize()

    def run(self):
        """
        This starts the execution of the queued callbacks.
        """
        self.__stop = False
        while not self.__stop:
            try:
                self.queue.get(timeout=0.1).run(self)
            except Empty:
                if self.__expire:
                    break
        if type(self) is threaded_queue:
            self.thread = None

    def expire(self):
        """
        This sets the expire flag. That means that the process will stop after queue gets empty.
        """
        self.__expire = True

    def stop(self):
        """
        This sets the stop flag. That means that the process will stop after finishing the active
     task.
        """
        self.__stop = True
```

```python


class threaded_queue(queue):
    """Class to execute queued callbacks in a background thread (See also parent :py:class:`queue`).

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

    **Example:**

    .. literalinclude:: task/_examples_/threaded_queue.py

    Will result to the following output:

    .. literalinclude:: task/_examples_/threaded_queue.log
    """
    def __init__(self, expire=False):
        queue.__init__(self, expire=expire)
        self.thread = None

    def run(self):
        if self.thread is None:
            self.thread = threading.Thread(target=self._start, args=())
            self.thread.daemon = True     # Daemonize thread
            self.thread.start()           # Start the execution

    def join(self):
        """
        This blocks till the queue is empty.

        .. note:: If the queue does not run dry, join will block till the end of the days.
        """
        self.expire()
        if self.thread is not None:
            self.thread.join()

    def stop(self):
        queue.stop(self)
        self.join()

    def _start(self):
        queue.run(self)


class periodic(object):
    """
    Class to execute a callback cyclicly.

    :param float cycle_time: Cycle time in seconds -- callback will be executed every *cycle_time* seconds
    :param callback callback: Callback to be executed
    :param args args: Arguments to be given to the callback
    :param kwargs kwargs: Keword Arguments to be given to callback

    .. note:: The Callback will get this instance as first argument, followed by :py:data:`args` und :py:data:`kwargs`.

    **Example:**

    .. literalinclude:: task/_examples_/periodic.py

    Will result to the following output:

    .. literalinclude:: task/_examples_/periodic.log
    """
    def __init__(self, cycle_time, callback, *args, **kwargs):
        self._lock = threading.Lock()
        self._timer = None
        self.callback = callback
        self.cycle_time = cycle_time
        self.args = args
        self.kwargs = kwargs
        self._stopped = True
        self._last_tm = None
        self.dt = None

    def join(self):
        """
        This blocks till the cyclic task is terminated.

        .. note:: Using join means that somewhere has to be a condition calling :py:func:`stop` to terminate. Otherwise :func:`task.join` will never return.
        """
        while not self._stopped:
            time.sleep(.1)

    def run(self):
        """
        This starts the cyclic execution of the given callback.
        """
        if self._stopped:
            self._set_timer(force_now=True)

    def stop(self):
        """
        This stops the execution of any further task.
        """
        self._lock.acquire()
        self._stopped = True
        if self._timer is not None:
            self._timer.cancel()
        self._lock.release()

    def _set_timer(self, force_now=False):
        """
        This sets the timer for the execution of the next task.
        """
        self._lock.acquire()
        self._stopped = False
        if force_now:
            self._timer = threading.Timer(0, self._start)
        else:
            self._timer = threading.Timer(self.cycle_time, self._start)
        self._timer.start()
        self._lock.release()

    def _start(self):
        tm = time.time()
        if self._last_tm is not None:
            self.dt = tm - self._last_tm
        self._set_timer(force_now=False)
        self.callback(self, *self.args, **self.kwargs)
        self._last_tm = tm


class delayed(periodic):
    """Class to execute a callback a given time in the future. See also parent :py:class:`periodic`.

    :param float time: Delay time for execution of the given callback
    :param callback callback: Callback to be executed
    :param args args: Arguments to be given to callback
    :param kwargs kwargs: Keword Arguments to be given to callback

    **Example:**

    .. literalinclude:: task/_examples_/delayed.py

    Will result to the following output:

    .. literalinclude:: task/_examples_/delayed.log
    """
    def run(self):
        """
        This starts the timer for the delayed execution.
        """
        self._set_timer(force_now=False)

    def _start(self):
        self.callback(*self.args, **self.kwargs)
        self.stop()


class crontab(periodic):
    """Class to execute a callback at the specified time conditions. See also parent :py:class:`periodic`.

    :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.
    :type accuracy: float

    **Example:**

    .. literalinclude:: task/_examples_/crontab.py

    Will result to the following output:

    .. literalinclude:: task/_examples_/crontab.log
    """
    ANY = '*'
    """Constant for matching every condition."""

    class cronjob(object):
        """Class to handle cronjob parameters and cronjob changes.

        :param minute: Minute for execution. Either 0...59, [0...59, 0...59, ...] or :py:const:`crontab.ANY` for every Minute.
        :type minute: int, list, str
        :param hour: Hour for execution. Either 0...23, [0...23, 0...23, ...] or :py:const:`crontab.ANY` for every Hour.
        :type hour: int, list, str
        :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.
        :type day_of_month: int, list, str
        :param month: Month for execution. Either 0...12, [0...12, 0...12, ...] or :py:const:`crontab.ANY` for every Month.
        :type month: int, list, str
        :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.
        :type day_of_week: int, list, str
        :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.
        :type callback: func

        .. 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()`.
        """
    class all_match(set):
        """Universal set - match everything"""
        def __contains__(self, item):
            (item)
            return True

    def __init__(self, minute, hour, day_of_month, month, day_of_week, callback, *args, **kwargs):
        self.set_trigger_conditions(minute or crontab.ANY, hour or crontab.ANY, day_of_month or crontab.ANY, month or crontab.ANY, day_of_week or crontab.ANY)
        self.callback = callback
        self.args = args
        self.kwargs = kwargs
        self.__last_cron_check_time__ = None
        self.__last_execution__ = None

    def set_trigger_conditions(self, minute=None, hour=None, day_of_month=None, month=None, day_of_week=None):
        """This Method changes the execution parameters.

        :param minute: Minute for execution. Either 0...59, [0...59, 0...59, ...] or :py:const:`crontab.ANY` for every Minute.
        :type minute: int, list, str
        :param hour: Hour for execution. Either 0...23, [0...23, 0...23, ...] or :py:const:`crontab.ANY` for every Hour.
        :type hour: int, list, str
        :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.
        :type day_of_month: int, list, str
        :param month: Month for execution. Either 0...12, [0...12, 0...12, ...] or :py:const:`crontab.ANY` for every Month.
        :type month: int, list, str
        :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.
        :type day_of_week: int, list, str
        """
        if minute is not None:
            self.minute = self.__conv_to_set__(minute)
        if hour is not None:
            self.hour = self.__conv_to_set__(hour)
        if day_of_month is not None:
            self.day_of_month = self.__conv_to_set__(day_of_month)
        if month is not None:
            self.month = self.__conv_to_set__(month)
        if day_of_week is not None:
            self.day_of_week = self.__conv_to_set__(day_of_week)

    def __conv_to_set__(self, obj):
        if obj is crontab.ANY:
            return self.all_match()
        elif isinstance(obj, (int, long) if sys.version_info < (3,0) else (int)):
            return set([obj])
```

```python
            else:
                return set(obj)

        def __execution_needed_for__(self, minute, hour, day_of_month, month, day_of_week):
            if self.__last_execution__ != [minute, hour, day_of_month, month, day_of_week]:
                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:
                    return True
            return False

        def __store_execution_reminder__(self, minute, hour, day_of_month, month, day_of_week):
            self.__last_execution__ = [minute, hour, day_of_month, month, day_of_week]

        def cron_execution(self, tm):
            """This Methods executes the Cron-Callback, if a execution is needed for the given time (depending on the parameters on initialisation)

            :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())).
            :type tm: int
            """
            if self.__last_cron_check_time__ is None:
                self.__last_cron_check_time__ = tm - 1
            #
            for t in range(self.__last_cron_check_time__ + 1, tm + 1):
                lt = time.localtime(t)
                if self.__execution_needed_for__(lt[4], lt[3], lt[2], lt[1], lt[6]):
                    self.callback(self, *self.args, **self.kwargs)
                    self.__store_execution_reminder__(lt[4], lt[3], lt[2], lt[1], lt[6])
                    break
            self.__last_cron_check_time__ = tm

    def __init__(self, accuracy=30):
        periodic.__init__(self, accuracy, self.__periodic__)
        self.__crontab__ = []

    def __periodic__(self, rt):
        (rt)
        tm = int(time.time())
        for cronjob in self.__crontab__:
            cronjob.cron_execution(tm)

    def add_cronjob(self, minute, hour, day_of_month, month, day_of_week, callback, *args, **kwargs):
        """This Method adds a cronjob to be executed.

        :param minute: Minute for execution. Either 0...59, [0...59, 0...59, ...] or :py:const:`crontab.ANY` for every Minute.
        :type minute: int, list, str
        :param hour: Hour for execution. Either 0...23, [0...23, 0...23, ...] or :py:const:`crontab.ANY` for every Hour.
        :type hour: int, list, str
        :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.
        :type day_of_month: int, list, str
        :param month: Month for execution. Either 0...12, [0...12, 0...12, ...] or :py:const:`crontab.ANY` for every Month.
        :type month: int, list, str
        :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.
        :type day_of_week: int, list, str
        :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.
        :type callback: func

        .. note:: The ``callback`` will be executed with it's instance of :py:class:`cronjob` as the first parameter.
            The given Arguments (:data:`args`) and keyword Arguments (:data:`kwargs`) will be stored in that object.
        """
        self.__crontab__.append(self.cronjob(minute, hour, day_of_month, month, day_of_week, callback, *args, **kwargs))
```