Unittest for `state_machine`

December 26, 2019

Contents

1 Test Information

1.1 Test Candidate Information

This Module helps implementing state machines.

Library Information	
Name	state_machine
State	Released
Supported Interpreters	python2, python3
Version	62acd0029b6217cb4a2151caafb560a7
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	769ddbf886b3c54506bf1f74ea6e1878
Testruns with	python 2.7.17 (final), python 3.6.9 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	LinuxMint 19.3 tricia
Hostname	ahorn
Kernel	5.0.0-37-generic (#40 18.04.1-Ubuntu SMP Thu Nov 14 12:06:39 UTC 2019)
Machine	x86_64
Path	/user_data/data/dirk/prj/modules/state_machine/unittest
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun with python 2.7.17 (final)

Number of tests	**19**
Number of successfull tests	**19**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	1.651s

2.2 Test-Statistic for testrun with python 3.6.9 (final)

Number of tests	**19**
Number of successfull tests	**19**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	1.649s

2.3 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
state_machine	100.0%	100.0%
state_machine.__init__.py	100.0%	

3 Tested Requirements

3.1 Module Initialisation

3.1.1 Default State

Description
The state machine shall start in the state, given while module initialisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
State machine is in the initial state after initialisation.

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (22)
Start-Time:	2019-12-26 13:33:59,995
Finished-Time:	2019-12-26 13:33:59,996
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state_c
Success	State after initialisation is correct (Content 'state_c' and Type is <type 'str'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (22)
Start-Time:	2019-12-26 13:34:01,995
Finished-Time:	2019-12-26 13:34:01,995
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state_c
Success	State after initialisation is correct (Content 'state_c' and Type is <class 'str'>).

3.1.2 Default Last Transition Condtion

Description
The state machine shall return the string `__init__` for last transition condition after initalisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
The last transition condition is __init__ after initialisation.

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (23)
Start-Time:	2019-12-26 13:33:59,996
Finished-Time:	2019-12-26 13:33:59,996
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state_c
Success	Last transition condition after initialisation is correct (Content '__init__' and Type is <type 'str'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (23)
Start-Time:	2019-12-26 13:34:01,995
Finished-Time:	2019-12-26 13:34:01,995
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state_c
Success	Last transition condition after initialisation is correct (Content '__init__' and Type is <class 'str'>).

3.1.3 Default Previous State

Description
The state machine shall return None for previous state after initalisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
The previous state is None after initialisation.

Testresult

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (24)
Start-Time:	2019-12-26 13:33:59,996
Finished-Time:	2019-12-26 13:33:59,996
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state_c
Success	Last state after initialisation is correct (Content None and Type is <type 'NoneType'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (24)
Start-Time:	2019-12-26 13:34:01,995
Finished-Time:	2019-12-26 13:34:01,996
Time-Consumption	0.000s

Testsummary:

Info	Initialising the state machine with state_c
Success	Last state after initialisation is correct (Content None and Type is <class 'NoneType'>).

3.1.4 Additional Keyword Arguments

Description

The state machine shall store all given keyword arguments as variables of the classes instance.

Reason for the implementation

Store further information (e.g. for calculation of the transition conditions).

Fitcriterion

At least two given keyword arguments with different types are available after initialisation.

Testresult

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (25)
Start-Time:	2019-12-26 13:33:59,997
Finished-Time:	2019-12-26 13:33:59,997
Time-Consumption	0.001s

Testsummary:	
Info	Initialising the state machine with state_c
Success	Keyword argument kw_arg_no_4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <type 'dict'>).
Success	Keyword argument kw_arg_no_1 stored in state_machine is correct (Content 1 and Type is <type 'int'>).
Success	Keyword argument kw_arg_no_3 stored in state_machine is correct (Content True and Type is <type 'bool'>).
Success	Keyword argument kw_arg_no_2 stored in state_machine is correct (Content '2' and Type is <type 'str'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (25)
Start-Time:	2019-12-26 13:34:01,996
Finished-Time:	2019-12-26 13:34:01,996
Time-Consumption	0.001s

Testsummary:	
Info	Initialising the state machine with state_c
Success	Keyword argument kw_arg_no_1 stored in state_machine is correct (Content 1 and Type is <class 'int'>).
Success	Keyword argument kw_arg_no_2 stored in state_machine is correct (Content '2' and Type is <class 'str'>).
Success	Keyword argument kw_arg_no_3 stored in state_machine is correct (Content True and Type is <class 'bool'>).
Success	Keyword argument kw_arg_no_4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <class 'dict'>).

3.2 Transition Changes

3.2.1 Transitiondefinition and -flow

Description

The user shall be able to define multiple states and transitions for the state machine. A transition shall have a start state, a target state and a transition condition. The transition condition shall be a method, where the user is able to calculate the condition on demand.

Reason for the implementation

Definition of the transitions for a state machine.

Fitcriterion

The order of at least three state changes is correct.

Testresult

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (28)
Start-Time:	2019-12-26 13:33:59,997
Finished-Time:	2019-12-26 13:33:59,998
Time-Consumption	0.001s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
Info	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)
Success	State after 1st execution of work method is correct (Content 'state_b' and Type is <type 'str'>).
Info	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)
Success	State after 2nd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).
Info	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)
Success	State after 3rd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (28)
Start-Time:	2019-12-26 13:34:01,996
Finished-Time:	2019-12-26 13:34:01,997
Time-Consumption	0.001s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)
Success	State after 1st execution of work method is correct (Content 'state_b' and Type is <class 'str'>).
Info	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)
Success	State after 2nd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).
Info	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)
Success	State after 3rd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).

3.2.2 Transitiontiming

Description

The user shall be able to define for each transition a transition time. On change of the transition condition to `True`, the transition timer starts counting the time from 0.0s. After reaching the transition time, the transition gets active.

Reason for the implementation

Robustness of the state changes (e.g. Oscillating conditions shall be ignored).

Fitcriterion

The transition time and the restart of the transion timer by setting the transition condition to `False` and to `True` again results in the expected transition timing (±0.05s).

Testresult

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (29)
Start-Time:	2019-12-26 13:33:59,999
Finished-Time:	2019-12-26 13:34:00,378
Time-Consumption	0.380s

Testsummary:

Info	Initialising state machine with state_a
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
Info	Waiting for 0.160s or state change
Success	State after 1st cycle is correct (Content 'state_b' and Type is <type 'str'>).
Success	Transition time after 1st cycle is correct (Content 0.15059208869934082 in [0.145 ... 0.155] and Type is <type 'float'>).
Info	Waiting for 0.235s or state change
Success	State after 2nd cycle is correct (Content 'state_c' and Type is <type 'str'>).
Success	Transition time after 2nd cycle is correct (Content 0.1503589153289795 in [0.145 ... 0.155] and Type is <type 'float'>).
Success	Previous state duration is correct (Content 0.22557401657104492 in [0.21999999999999997 ... 0.22999999999999998] and Type is <type 'float'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (29)
Start-Time:	2019-12-26 13:34:01,997
Finished-Time:	2019-12-26 13:34:02,377
Time-Consumption	0.380s

Testsummary:

Info	Initialising state machine with state_a

Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info	Waiting for 0.160s or state change
Success	State after 1st cycle is correct (Content 'state_b' and Type is <class 'str'>).
Success	Transition time after 1st cycle is correct (Content 0.15062165260314941 in [0.145 ... 0.155] and Type is <class 'float'>).
Info	Waiting for 0.235s or state change
Success	State after 2nd cycle is correct (Content 'state_c' and Type is <class 'str'>).
Success	Transition time after 2nd cycle is correct (Content 0.15032720565795898 in [0.145 ... 0.155] and Type is <class 'float'>).
Success	Previous state duration is correct (Content 0.2256786823272705 in [0.21999999999999997 ... 0.22999999999999998] and Type is <class 'float'>).

3.2.3 Transitionpriorisation

Description

The state machine shall use the first active transition. If multiple transition are active, the transition with the highest overlap time will be used.

Reason for the implementation

Compensate the weakness of the execution quantisation.

Fitcriterion

At least one transition with at least two active conditions results in the expected state change.

Testresult

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (30)
Start-Time:	2019-12-26 13:34:00,379
Finished-Time:	2019-12-26 13:34:00,623
Time-Consumption	0.244s

Testsummary:

Info	Initialising state machine with state_a, a transition to state_b after 0.151s and a transition to state_c after 0.150s
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).
Info	Waiting for 0.300s or state change
Success	State after 1st cycle is correct (Content 'state_c' and Type is <type 'str'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (30)
Start-Time:	2019-12-26 13:34:02,377

Finished-Time:	2019-12-26 13:34:02,621
Time-Consumption	0.244s

Testsummary:

Info	Initialising state machine with state_a, a transition to state_b after 0.151s and a transition to state_c after 0.150s
Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).
Info	Waiting for 0.300s or state change
Success	State after 1st cycle is correct (Content 'state_c' and Type is <class 'str'>).

3.3 Module Interface

3.3.1 This State

Description

The Module shall have a method for getting the current state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returend state fits to the expecation.

Testresult

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (33)
Start-Time:	2019-12-26 13:34:00,623
Finished-Time:	2019-12-26 13:34:00,624
Time-Consumption	0.001s

Testsummary:

Info	Initialising the state machine with state_c
Success	Returnvalue of this_state() is correct (Content 'state_c' and Type is <type 'str'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (33)
Start-Time:	2019-12-26 13:34:02,621
Finished-Time:	2019-12-26 13:34:02,622
Time-Consumption	0.001s

Testsummary:

Info	Initialising the state machine with state_c
Success	Returnvalue of this_state() is correct (Content 'state_c' and Type is <class 'str'>).

3.3.2 This State is

Description

The Module shall have a method for checking if the given state is currently active.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (34)
Start-Time:	2019-12-26 13:34:00,625
Finished-Time:	2019-12-26 13:34:00,626
Time-Consumption	0.001s

Testsummary:

Info	Initialising the state machine with state_c
Success	Returnvalue of this_state_is(state_c) is correct (Content True and Type is <type 'bool'>).
Success	Returnvalue of this_state_is(state_b) is correct (Content False and Type is <type 'bool'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (34)
Start-Time:	2019-12-26 13:34:02,623
Finished-Time:	2019-12-26 13:34:02,624
Time-Consumption	0.001s

Testsummary:

Info	Initialising the state machine with state_c
Success	Returnvalue of this_state_is(state_c) is correct (Content True and Type is <class 'bool'>).
Success	Returnvalue of this_state_is(state_b) is correct (Content False and Type is <class 'bool'>).

3.3.3 This State Duration

Description
The Module shall have a method for getting the time since the last state change appears.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returned duration fits to the current state duration (± 0.05s).

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.10!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (35)
Start-Time:	2019-12-26 13:34:00,626
Finished-Time:	2019-12-26 13:34:00,879
Time-Consumption	0.252s
Testsummary:	
Info	Running state machine test sequence.
Success	Return Value of this_state_duration() is correct (Content 0.2511169910430908 in [0.2 ... 0.3] and Type is <type 'float'>).

Testresult
This test was passed with the state: **Success**. See also full trace in section B.1.10!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (35)
Start-Time:	2019-12-26 13:34:02,624
Finished-Time:	2019-12-26 13:34:02,876
Time-Consumption	0.252s
Testsummary:	
Info	Running state machine test sequence.
Success	Return Value of this_state_duration() is correct (Content 0.2508230209350586 in [0.2 ... 0.3] and Type is <class 'float'>).

3.3.4 Last Transition Condition

Description
The Module shall have a method for getting the last transition condition.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returned transition condition fits to the expectation.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.11!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (36)
Start-Time:	2019-12-26 13:34:00,879
Finished-Time:	2019-12-26 13:34:00,880
Time-Consumption	0.001s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <type 'str'>).

Testresult
This test was passed with the state: **Success**. See also full trace in section B.1.11!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (36)
Start-Time:	2019-12-26 13:34:02,876
Finished-Time:	2019-12-26 13:34:02,877
Time-Consumption	0.001s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <class 'str'>).

3.3.5 Last Transition Condition was

Description
The Module shall have a method for checking if the given condition was the last transition condition.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.12!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (37)
Start-Time:	2019-12-26 13:34:00,881
Finished-Time:	2019-12-26 13:34:00,883
Time-Consumption	0.002s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <type 'bool'>).
Success	Returnvalue of last_transition_condition(condition_c) is correct (Content False and Type is <type 'bool'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.12!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (37)
Start-Time:	2019-12-26 13:34:02,878
Finished-Time:	2019-12-26 13:34:02,879
Time-Consumption	0.001s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <class 'bool'>).
Success	Returnvalue of last_transition_condition(condition_c) is correct (Content False and Type is <class 'bool'>).

3.3.6 Previous State

Description

The Module shall have a method for getting the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returend state fits to the expecation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.13!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (38)
Start-Time:	2019-12-26 13:34:00,883
Finished-Time:	2019-12-26 13:34:00,884
Time-Consumption	0.001s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of previous_state() is correct (Content 'state_a' and Type is <type 'str'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.13!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (38)
Start-Time:	2019-12-26 13:34:02,879
Finished-Time:	2019-12-26 13:34:02,880
Time-Consumption	0.001s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of previous_state() is correct (Content 'state_a' and Type is <class 'str'>).

3.3.7 Previous State was

Description

The Module shall have a method for checking if the given state was the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.14!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (39)
Start-Time:	2019-12-26 13:34:00,885
Finished-Time:	2019-12-26 13:34:00,887
Time-Consumption	0.002s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of previous_state_was(state_a) is correct (Content True and Type is <type 'bool'>).

Success	Returnvalue of previous_state_was(state_b) is correct (Content False and Type is <type 'bool'>).

Testresult

This test was passed with the state: Success. See also full trace in section B.1.14!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (39)
Start-Time:	2019-12-26 13:34:02,881
Finished-Time:	2019-12-26 13:34:02,882
Time-Consumption	0.001s

Testsummary:

Info	Running state machine test sequence.
Success	Returnvalue of previous_state_was(state_a) is correct (Content True and Type is <class 'bool'>).
Success	Returnvalue of previous_state_was(state_b) is correct (Content False and Type is <class 'bool'>).

3.3.8 Previous State Duration

Description

The Module shall have a method for getting active time for the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least one returned duration fits to the previous state duration (± 0.05s).

Testresult

This test was passed with the state: Success. See also full trace in section A.1.15!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (40)
Start-Time:	2019-12-26 13:34:00,887
Finished-Time:	2019-12-26 13:34:01,640
Time-Consumption	0.753s

Testsummary:

Info	Running state machine test sequence.
Success	Return Value of previous_state_duration() is correct (Content 0.7514150142669678 in [0.7 ... 0.8] and Type is <type 'float'>).

Testresult

This test was passed with the state: Success. See also full trace in section B.1.15!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (40)
Start-Time:	2019-12-26 13:34:02,882
Finished-Time:	2019-12-26 13:34:03,635
Time-Consumption	0.753s

Testsummary:

Info	Running state machine test sequence.
Success	Return Value of previous_state_duration() is correct (Content 0.7513992786407471 in [0.7 ... 0.8] and Type is <class 'float'>).

3.4 Transition Callbacks

3.4.1 State change callback for a defined transition and targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined set of *transition_condition* and *target_state*.

Reason for the implementation

Triggering state change actions for a specific transition condition and targetstate.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and targetstate and at least for one other condition not.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.16!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (43)
Start-Time:	2019-12-26 13:34:01,640
Finished-Time:	2019-12-26 13:34:01,644
Time-Consumption	0.004s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	List of the submitted values for Execution of state machine callback (1) (state_b, condition_a) identified by a sequence number is correct (Content [1] and Type is <type 'list'>).
Success	List of the submitted values for Execution of state machine callback (2) (state_b, condition_a) identified by a sequence number is correct (Content [2] and Type is <type 'list'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.16!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (43)
Start-Time:	2019-12-26 13:34:03,635
Finished-Time:	2019-12-26 13:34:03,639
Time-Consumption	0.004s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	List of the submitted values for Execution of state machine callback (1) (state_b, condition_a) identified by a sequence number is correct (Content [1] and Type is <class 'list'>).
Success	List of the submitted values for Execution of state machine callback (2) (state_b, condition_a) identified by a sequence number is correct (Content [2] and Type is <class 'list'>).

3.4.2 State change callback for a defined transition

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined *transition_condition* and all *target_states*.

Reason for the implementation

Triggering state change actions for a specific transition condition.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and at least for one other transition condition not.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.17!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (44)
Start-Time:	2019-12-26 13:34:01,645
Finished-Time:	2019-12-26 13:34:01,649
Time-Consumption	0.004s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	List of the submitted values for Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number is correct (Content [2, 5] and Type is <type 'list'>).
Success	List of the submitted values for Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number is correct (Content [3, 6] and Type is <type 'list'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.17!

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (44)
Start-Time:	2019-12-26 13:34:03,639
Finished-Time:	2019-12-26 13:34:03,643
Time-Consumption	0.004s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	List of the submitted values for Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number is correct (Content [2, 5] and Type is <class 'list'>).
Success	List of the submitted values for Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number is correct (Content [3, 6] and Type is <class 'list'>).

3.4.3 State change callback for a defined targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all *transition_conditions* and a defined *target_state*.

Reason for the implementation

Triggering state change actions for a specific targetstate.

Fitcriterion

Methods are called in the registration order after state change with the defined targetstate and at least for one other targetstate not.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.18!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (45)
Start-Time:	2019-12-26 13:34:01,649
Finished-Time:	2019-12-26 13:34:01,651
Time-Consumption	0.002s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	List of the submitted values for Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number is correct (Content [1, 5] and Type is <type 'list'>).
Success	List of the submitted values for Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number is correct (Content [2, 6] and Type is <type 'list'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.18!

Testrun:	python 3.6.9 (final)

Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (45)
Start-Time:	2019-12-26 13:34:03,643
Finished-Time:	2019-12-26 13:34:03,646
Time-Consumption	0.003s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	List of the submitted values for Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number is correct (Content [1, 5] and Type is <class 'list'>).
Success	List of the submitted values for Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number is correct (Content [2, 6] and Type is <class 'list'>).

3.4.4 State change callback for all kind of state changes

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all transitions.

Reason for the implementation

Triggering state change actions for all transition conditions and targetstates.

Fitcriterion

Methods are called in the registration order after state change.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.19!

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (46)
Start-Time:	2019-12-26 13:34:01,651
Finished-Time:	2019-12-26 13:34:01,652
Time-Consumption	0.001s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	List of the submitted values for Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number is correct (Content [1, 4, 7, 10] and Type is <type 'list'>).
Success	List of the submitted values for Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number is correct (Content [2, 5, 8, 11] and Type is <type 'list'>).

Testresult

This test was passed with the state: **Success**. See also full trace in section B.1.19!

Testrun:	python 3.6.9 (final)

Caller:	/user_data/data/dirk/prj/modules/state_machine/unittest/src/tests/__init__.py (46)
Start-Time:	2019-12-26 13:34:03,646
Finished-Time:	2019-12-26 13:34:03,648
Time-Consumption	0.001s

Testsummary:

Info	Running state machine sequence and storing sequence number for each callback
Success	List of the submitted values for Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number is correct (Content [1, 4, 7, 10] and Type is <class 'list'>).
Success	List of the submitted values for Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number is correct (Content [2, 5, 8, 11] and Type is <class 'list'>).

A Trace for testrun with python 2.7.17 (final)

A.1 Tests with status Info (19)

A.1.1 Default State

Description
The state machine shall start in the state, given while module initialisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
State machine is in the initial state after initialisation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	State after initialisation is correct (Content 'state_c' and Type is <type 'str'>).

```
Result: 'state_c' (<type 'str'>)
Expectation: result = 'state_c' (<type 'str'>)
```

A.1.2 Default Last Transition Condtion

Description
The state machine shall return the string __init__ for last transition condition after initalisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
The last transition condition is __init__ after initialisation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Last transition condition after initialisation is correct (Content '__init__' and Type is <type 'str'>).

```
Result: '__init__' (<type 'str'>)
Expectation: result = '__init__' (<type 'str'>)
```

A.1.3 Default Previous State

Description
The state machine shall return None for previous state after initalisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
The previous state is None after initialisation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Last state after initialisation is correct (Content None and Type is <type 'NoneType'>).

```
Result: None (<type 'NoneType'>)
Expectation: result = None (<type 'NoneType'>)
```

A.1.4 Additional Keyword Arguments

Description
The state machine shall store all given keyword arguments as variables of the classes instance.

Reason for the implementation
Store further information (e.g. for calculation of the transition conditions).

Fitcriterion
At least two given keyword arguments with different types are available after initialisation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Keyword argument kw_arg_no_4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <type 'dict'>).

```
Result: { '1': 1, '2': 'two' } (<type 'dict'>)
Expectation: result = { '1': 1, '2': 'two' } (<type 'dict'>)
```

Success	Keyword argument kw_arg_no_1 stored in state_machine is correct (Content 1 and Type is <type 'int'>).

```
Result: 1 (<type 'int'>)
Expectation: result = 1 (<type 'int'>)
```

Success	Keyword argument kw_arg_no_3 stored in state_machine is correct (Content True and Type is <type 'bool'>).

```
Result: True (<type 'bool'>)
Expectation: result = True (<type 'bool'>)
```

Success	Keyword argument kw_arg_no_2 stored in state_machine is correct (Content '2' and Type is <type 'str'>).

```
Result: '2' (<type 'str'>)
Expectation: result = '2' (<type 'str'>)
```

A.1.5 Transitiondefinition and -flow

Description
The user shall be able to define multiple states and transitions for the state machine. A transition shall have a start state, a target state and a transition condition. The transition condition shall be a method, where the user is able to calculate the condition on demand.

Reason for the implementation
Definition of the transitions for a state machine.

Fitcriterion
The order of at least three state changes is correct.

Testresult

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

Info	Initialising state machine with state_a

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).

```
Result: 'state_a' (<type 'str'>)
Expectation: result = 'state_a' (<type 'str'>)
```

Info	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)

```
StateMachine: State change ('condition_true'): 'state_a' -> 'state_b'
```

Success	State after 1st execution of work method is correct (Content 'state_b' and Type is <type 'str'>).

```
Result: 'state_b' (<type 'str'>)
Expectation: result = 'state_b' (<type 'str'>)
```

Info	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)

```
StateMachine: State change ('condition_true'): 'state_b' -> 'state_c'
```

Success	State after 2nd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).

```
Result: 'state_c' (<type 'str'>)
Expectation: result = 'state_c' (<type 'str'>)
```

Info	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)

Success	State after 3rd execution of work method is correct (Content 'state_c' and Type is <type 'str'>).

```
Result: 'state_c' (<type 'str'>)
Expectation: result = 'state_c' (<type 'str'>)
```

A.1.6 Transitiontiming

Description

The user shall be able to define for each transition a transition time. On change of the transition condition to `True`, the transition timer starts counting the time from 0.0s. After reaching the transition time, the transition gets active.

Reason for the implementation

Robustness of the state changes (e.g. Oscillating conditions shall be ignored).

Fitcriterion

The transition time and the restart of the transion timer by setting the transition condition to `False` and to `True` again results in the expected transition timing (±0.05s).

Testresult

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

Info	Initialising state machine with state_a

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).

```
Result: 'state_a' (<type 'str'>)
Expectation: result = 'state_a' (<type 'str'>)
```

Info	Waiting for 0.160s or state change

```
StateMachine: State change ('condition_true'): 'state_a' -> 'state_b'
```

Success	State after 1st cycle is correct (Content 'state_b' and Type is <type 'str'>).

```
Result: 'state_b' (<type 'str'>)
Expectation: result = 'state_b' (<type 'str'>)
```

Success	Transition time after 1st cycle is correct (Content 0.15059208869934082 in [0.145 ... 0.155] and Type is <type 'float'>).

```
Result: 0.15059208869934082 (<type 'float'>)
Expectation: 0.145 <= result <= 0.155
```

Info	Waiting for 0.235s or state change

```
StateMachine: State change ('condition_true'): 'state_b' -> 'state_c'
```

Success	State after 2nd cycle is correct (Content 'state_c' and Type is <type 'str'>).

```
Result: 'state_c' (<type 'str'>)
Expectation: result = 'state_c' (<type 'str'>)
```

Success	Transition time after 2nd cycle is correct (Content 0.1503589153289795 in [0.145 ... 0.155] and Type is <type 'float'>).

```
Result: 0.1503589153289795 (<type 'float'>)
Expectation: 0.145 <= result <= 0.155
```

Success	Previous state duration is correct (Content 0.22557401657104492 in [0.21999999999999997 ... 0.22999999999999998] and Type is <type 'float'>).

```
Result: 0.22557401657104492 (<type 'float'>)
Expectation: 0.21999999999999997 <= result <= 0.22999999999999998
```

A.1.7 Transitionpriorisation

Description
The state machine shall use the first active transition. If multiple transition are active, the transition with the highest overlap time will be used.

Reason for the implementation
Compensate the weakness of the execution quantisation.

Fitcriterion
At least one transition with at least two active conditions results in the expected state change.

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

Info	Initialising state machine with state_a, a transition to state_b after 0.151s and a transition to state_c after 0.150s

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <type 'str'>).

```
Result: 'state_a' (<type 'str'>)
Expectation: result = 'state_a' (<type 'str'>)
```

Info	Waiting for 0.300s or state change

```
Executing method work after 0.000s
Executing method work after 0.060s
Executing method work after 0.121s
Executing method work after 0.181s
StateMachine: State change ('condition_true'): 'state_a' -> 'state_c'
```

Success	State after 1st cycle is correct (Content 'state_c' and Type is <type 'str'>).

```
Result: 'state_c' (<type 'str'>)
Expectation: result = 'state_c' (<type 'str'>)
```

A.1.8 This State

Description
The Module shall have a method for getting the current state.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returend state fits to the expecation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Returnvalue of this_state() is correct (Content 'state_c' and Type is <type 'str'>).

```
Result: 'state_c' (<type 'str'>)
Expectation: result = 'state_c' (<type 'str'>)
```

A.1.9 This State is

Description
The Module shall have a method for checking if the given state is currently active.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least two calls with different return values fit to the expectation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Returnvalue of this_state_is(state_c) is correct (Content True and Type is <type 'bool'>).

```
Result: True (<type 'bool'>)
Expectation: result = True (<type 'bool'>)
```

Success	Returnvalue of this_state_is(state_b) is correct (Content False and Type is <type 'bool'>).

```
Result: False (<type 'bool'>)
Expectation: result = False (<type 'bool'>)
```

A.1.10 This State Duration

Description
The Module shall have a method for getting the time since the last state change appears.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returned duration fits to the current state duration (± 0.05s).

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Waiting for 0.25s
```

Success	Return Value of this_state_duration() is correct (Content 0.2511169910430908 in [0.2 ... 0.3] and Type is <type 'float'>).

```
Result: 0.2511169910430908 (<type 'float'>)
Expectation: 0.2 <= result <= 0.3
```

A.1.11 Last Transition Condition

Description
The Module shall have a method for getting the last transition condition.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returned transition condition fits to the expectation.

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success	Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <type 'str'>).

```
Result: 'condition_a' (<type 'str'>)
Expectation: result = 'condition_a' (<type 'str'>)
```

A.1.12 Last Transition Condition was

Description

The Module shall have a method for checking if the given condition was the last transition condition.

Reason for the implementation

Comfortable user interface.

Fitcriterion

At least two calls with different return values fit to the expectation.

Testresult

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success	Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <type 'bool'>).

```
Result: True (<type 'bool'>)
Expectation: result = True (<type 'bool'>)
```

Success	Returnvalue of last_transition_condition(condition_c) is correct (Content False and Type is <type 'bool'>).

```
Result: False (<type 'bool'>)
Expectation: result = False (<type 'bool'>)
```

A.1.13 Previous State

Description

The Module shall have a method for getting the previous state.

Reason for the implementation

Comfortable user interface.

Fitcriterion
At least one returend state fits to the expecation.

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success	Returnvalue of previous_state() is correct (Content 'state_a' and Type is <type 'str'>).

```
Result: 'state_a' (<type 'str'>)
Expectation: result = 'state_a' (<type 'str'>)
```

A.1.14 Previous State was

Description
The Module shall have a method for checking if the given state was the previous state.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least two calls with different return values fit to the expectation.

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success	Returnvalue of previous_state_was(state_a) is correct (Content True and Type is <type 'bool'>).

```
Result: True (<type 'bool'>)
Expectation: result = True (<type 'bool'>)
```

Success	Returnvalue of previous_state_was(state_b) is correct (Content False and Type is <type 'bool'>).

```
Result: False (<type 'bool'>)
Expectation: result = False (<type 'bool'>)
```

A.1.15 Previous State Duration

Description
The Module shall have a method for getting active time for the previous state.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returned duration fits to the previous state duration (± 0.05s).

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Waiting for 0.75s
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
```

Success	Return Value of previous_state_duration() is correct (Content 0.7514150142669678 in [0.7 ... 0.8] and Type is <type 'float'>).

```
Result: 0.7514150142669678 (<type 'float'>)
Expectation: 0.7 <= result <= 0.8
```

A.1.16 State change callback for a defined transition and targetstate

Description
The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined set of *transition_condition* and *target_state*.

Reason for the implementation
Triggering state change actions for a specific transition condition and targetstate.

Fitcriterion
Methods are called in the registration order after state change with all user given arguments for the defined transition condition and targetstate and at least for one other condition not.

Testresult

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

Info	Running state machine sequence and storing sequence number for each callback

```
StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by callback_execution
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 6 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
```

Success	List of the submitted values for Execution of state machine callback (1) (state_b, condition_a) identified by a sequence number is correct (Content [1] and Type is <type 'list'>).

```
Result: [ 1 ] (<type 'list'>)
Expectation: result = [ 1 ] (<type 'list'>)
```

Success	List of the submitted values for Execution of state machine callback (2) (state_b, condition_a) identified by a sequence number is correct (Content [2] and Type is <type 'list'>).

```
Result: [ 2 ] (<type 'list'>)
Expectation: result = [ 2 ] (<type 'list'>)
```

A.1.17 State change callback for a defined transition

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined *transition_condition* and all *target_states*.

Reason for the implementation

Triggering state change actions for a specific transition condition.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and at least for one other transition condition not.

Testresult

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

Info	Running state machine sequence and storing sequence number for each callback

```
StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by callback_execution
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 6 caused by callback_execution
Increasing sequence number to 7 caused by callback_execution
Increasing sequence number to 8 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
```

Success	List of the submitted values for Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number is correct (Content [2, 5] and Type is <type 'list'>).

```
Result: [ 2, 5 ] (<type 'list'>)
Expectation: result = [ 2, 5 ] (<type 'list'>)
```

Success	List of the submitted values for Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number is correct (Content [3, 6] and Type is <type 'list'>).

```
Result: [ 3, 6 ] (<type 'list'>)
Expectation: result = [ 3, 6 ] (<type 'list'>)
```

A.1.18 State change callback for a defined targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all *transition_conditions* and a defined *target_state*.

Reason for the implementation

Triggering state change actions for a specific targetstate.

Fitcriterion

Methods are called in the registration order after state change with the defined targetstate and at least for one other targetstate not.

Testresult

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

Info	Running state machine sequence and storing sequence number for each callback

```
StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by callback_execution
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 6 caused by callback_execution
Increasing sequence number to 7 caused by callback_execution
Increasing sequence number to 8 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
```

Success	List of the submitted values for Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number is correct (Content [1, 5] and Type is <type 'list'>).

```
Result: [ 1, 5 ] (<type 'list'>)
Expectation: result = [ 1, 5 ] (<type 'list'>)
```

Success	List of the submitted values for Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number is correct (Content [2, 6] and Type is <type 'list'>).

```
Result: [ 2, 6 ] (<type 'list'>)
Expectation: result = [ 2, 6 ] (<type 'list'>)
```

A.1.19 State change callback for all kind of state changes

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all transitions.

Reason for the implementation

Triggering state change actions for all transition conditions and targetstates.

Fitcriterion

Methods are called in the registration order after state change.

Testresult

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

Info	Running state machine sequence and storing sequence number for each callback

```
StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by callback_execution
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by callback_execution
Increasing sequence number to 6 caused by callback_execution
Increasing sequence number to 7 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 8 caused by callback_execution
Increasing sequence number to 9 caused by callback_execution
Increasing sequence number to 10 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
Increasing sequence number to 11 caused by callback_execution
Increasing sequence number to 12 caused by callback_execution
```

Success	List of the submitted values for Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number is correct (Content [1, 4, 7, 10] and Type is <type 'list'>).

```
Result: [ 1, 4, 7, 10 ] (<type 'list'>)
Expectation: result = [ 1, 4, 7, 10 ] (<type 'list'>)
```

Success	List of the submitted values for Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number is correct (Content [2, 5, 8, 11] and Type is <type 'list'>).

```
Result: [ 2, 5, 8, 11 ] (<type 'list'>)
Expectation: result = [ 2, 5, 8, 11 ] (<type 'list'>)
```

B Trace for testrun with python 3.6.9 (final)

B.1 Tests with status Info (19)

B.1.1 Default State

Description

The state machine shall start in the state, given while module initialisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
State machine is in the initial state after initialisation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	State after initialisation is correct (Content 'state_c' and Type is <class 'str'>).

```
Result: 'state_c' (<class 'str'>)
Expectation: result = 'state_c' (<class 'str'>)
```

B.1.2 Default Last Transition Condtion

Description
The state machine shall return the string __init__ for last transition condition after initalisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
The last transition condition is __init__ after initialisation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Last transition condition after initialisation is correct (Content '__init__' and Type is <class 'str'>).

```
Result: '__init__' (<class 'str'>)
Expectation: result = '__init__' (<class 'str'>)
```

B.1.3 Default Previous State

Description
The state machine shall return None for previous state after initalisation.

Reason for the implementation
Creation of a defined state after initialisation.

Fitcriterion
The previous state is None after initialisation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Last state after initialisation is correct (Content None and Type is <class 'NoneType'>).

```
Result: None (<class 'NoneType'>)
Expectation: result = None (<class 'NoneType'>)
```

B.1.4 Additional Keyword Arguments

Description
The state machine shall store all given keyword arguments as variables of the classes instance.

Reason for the implementation
Store further information (e.g. for calculation of the transition conditions).

Fitcriterion
At least two given keyword arguments with different types are available after initialisation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Keyword argument kw_arg_no_1 stored in state_machine is correct (Content 1 and Type is <class 'int'>).

```
Result: 1 (<class 'int'>)
Expectation: result = 1 (<class 'int'>)
```

Success	Keyword argument kw_arg_no_2 stored in state_machine is correct (Content '2' and Type is <class 'str'>).

```
Result: '2' (<class 'str'>)
Expectation: result = '2' (<class 'str'>)
```

Success	Keyword argument kw_arg_no_3 stored in state_machine is correct (Content True and Type is <class 'bool'>).

```
Result: True (<class 'bool'>)
Expectation: result = True (<class 'bool'>)
```

Success	Keyword argument kw_arg_no_4 stored in state_machine is correct (Content {'1': 1, '2': 'two'} and Type is <class 'dict'>).

```
Result: { '1': 1, '2': 'two' } (<class 'dict'>)
Expectation: result = { '1': 1, '2': 'two' } (<class 'dict'>)
```

B.1.5 Transitiondefinition and -flow

Description

The user shall be able to define multiple states and transitions for the state machine. A transition shall have a start state, a target state and a transition condition. The transition condition shall be a method, where the user is able to calculate the condition on demand.

Reason for the implementation

Definition of the transitions for a state machine.

Fitcriterion

The order of at least three state changes is correct.

Testresult

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

Info	Initialising state machine with state_a

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).

```
Result: 'state_a' (<class 'str'>)
Expectation: result = 'state_a' (<class 'str'>)
```

Info	Work routine executed the 1st time to do the state change. Defined Transitions are: True→state_b (0.0s); False→state_c (0.0s)

```
StateMachine: State change ('condition_true'): 'state_a' -> 'state_b'
```

Success	State after 1st execution of work method is correct (Content 'state_b' and Type is <class 'str'>).

```
Result: 'state_b' (<class 'str'>)
Expectation: result = 'state_b' (<class 'str'>)
```

Info	Work routine executed the 2nd time to do the state change. Defined Transitions are: False→state_a (0.0s); True→state_c (0.0s)

```
StateMachine: State change ('condition_true'): 'state_b' -> 'state_c'
```

Success	State after 2nd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).

```
Result: 'state_c' (<class 'str'>)
Expectation: result = 'state_c' (<class 'str'>)
```

Info	Work routine executed the 3rd time with no effect. No Transitions starting from state_c (dead end)

Success	State after 3rd execution of work method is correct (Content 'state_c' and Type is <class 'str'>).

```
Result: 'state_c' (<class 'str'>)
Expectation: result = 'state_c' (<class 'str'>)
```

B.1.6 Transitiontiming

Description

The user shall be able to define for each transition a transition time. On change of the transition condition to `True`, the transition timer starts counting the time from 0.0s. After reaching the transition time, the transition gets active.

Reason for the implementation

Robustness of the state changes (e.g. Oscillating conditions shall be ignored).

Fitcriterion

The transition time and the restart of the transion timer by setting the transition condition to `False` and to `True` again results in the expected transition timing (±0.05s).

Testresult

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

Info	Initialising state machine with state_a

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).

```
Result: 'state_a' (<class 'str'>)
Expectation: result = 'state_a' (<class 'str'>)
```

Info	Waiting for 0.160s or state change

```
StateMachine: State change ('condition_true'): 'state_a' -> 'state_b'
```

Success	State after 1st cycle is correct (Content 'state_b' and Type is <class 'str'>).

```
Result: 'state_b' (<class 'str'>)
Expectation: result = 'state_b' (<class 'str'>)
```

Success	Transition time after 1st cycle is correct (Content 0.15062165260314941 in [0.145 ... 0.155] and Type is <class 'float'>).

```
Result: 0.15062165260314941 (<class 'float'>)
Expectation: 0.145 <= result <= 0.155
```

Info	Waiting for 0.235s or state change

```
StateMachine: State change ('condition_true'): 'state_b' -> 'state_c'
```

Success	State after 2nd cycle is correct (Content 'state_c' and Type is <class 'str'>).

```
Result: 'state_c' (<class 'str'>)
Expectation: result = 'state_c' (<class 'str'>)
```

Success	Transition time after 2nd cycle is correct (Content 0.15032720565795898 in [0.145 ... 0.155] and Type is <class 'float'>).

```
Result: 0.15032720565795898 (<class 'float'>)
Expectation: 0.145 <= result <= 0.155
```

Success	Previous state duration is correct (Content 0.2256786823272705 in [0.21999999999999997 ... 0.22999999999999998] and Type is <class 'float'>).

```
Result: 0.2256786823272705 (<class 'float'>)
Expectation: 0.21999999999999997 <= result <= 0.22999999999999998
```

B.1.7 Transitionpriorisation

Description

The state machine shall use the first active transition. If multiple transition are active, the transition with the highest overlap time will be used.

Reason for the implementation

Compensate the weakness of the execution quantisation.

Fitcriterion

At least one transition with at least two active conditions results in the expected state change.

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

Info	Initialising state machine with state_a, a transition to state_b after 0.151s and a transition to state_c after 0.150s

```
StateMachine: State change ('__init__'): None -> 'state_a'
```

Success	Initial state after Initialisation is correct (Content 'state_a' and Type is <class 'str'>).

```
Result: 'state_a' (<class 'str'>)
Expectation: result = 'state_a' (<class 'str'>)
```

Info	Waiting for 0.300s or state change

```
Executing method work after 0.000s
Executing method work after 0.060s
Executing method work after 0.121s
Executing method work after 0.181s
StateMachine: State change ('condition_true'): 'state_a' -> 'state_c'
```

Success	State after 1st cycle is correct (Content 'state_c' and Type is <class 'str'>).

```
Result: 'state_c' (<class 'str'>)
Expectation: result = 'state_c' (<class 'str'>)
```

B.1.8 This State

Description
The Module shall have a method for getting the current state.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returend state fits to the expecation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Returnvalue of this_state() is correct (Content 'state_c' and Type is <class 'str'>).

```
Result: 'state_c' (<class 'str'>)
Expectation: result = 'state_c' (<class 'str'>)
```

B.1.9 This State is

Description
The Module shall have a method for checking if the given state is currently active.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least two calls with different return values fit to the expectation.

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

Info	Initialising the state machine with state_c

```
StateMachine: State change ('__init__'): None -> 'state_c'
```

Success	Returnvalue of this_state_is(state_c) is correct (Content True and Type is <class 'bool'>).

```
Result: True (<class 'bool'>)
Expectation: result = True (<class 'bool'>)
```

Success	Returnvalue of this_state_is(state_b) is correct (Content False and Type is <class 'bool'>).

```
Result: False (<class 'bool'>)
Expectation: result = False (<class 'bool'>)
```

B.1.10 This State Duration

Description
The Module shall have a method for getting the time since the last state change appears.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returned duration fits to the current state duration ($\pm$ 0.05s).

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Waiting for 0.25s
```

Success	Return Value of this_state_duration() is correct (Content 0.2508230209350586 in [0.2 ... 0.3] and Type is <class 'float'>).

```
Result: 0.2508230209350586 (<class 'float'>)
Expectation: 0.2 <= result <= 0.3
```

B.1.11 Last Transition Condition

Description
The Module shall have a method for getting the last transition condition.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returned transition condition fits to the expectation.

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success	Returnvalue of last_transition_condition() is correct (Content 'condition_a' and Type is <class 'str'>).

```
Result: 'condition_a' (<class 'str'>)
Expectation: result = 'condition_a' (<class 'str'>)
```

B.1.12 Last Transition Condition was

Description
The Module shall have a method for checking if the given condition was the last transition condition.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least two calls with different return values fit to the expectation.

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success	Returnvalue of last_transition_condition(condition_a) is correct (Content True and Type is <class 'bool'>).

```
Result: True (<class 'bool'>)
Expectation: result = True (<class 'bool'>)
```

Success	Returnvalue of last_transition_condition(condition_c) is correct (Content False and Type is <class 'bool'>).

```
Result: False (<class 'bool'>)
Expectation: result = False (<class 'bool'>)
```

B.1.13 Previous State

Description
The Module shall have a method for getting the previous state.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returend state fits to the expecation.

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success	Returnvalue of previous_state() is correct (Content 'state_a' and Type is <class 'str'>).

```
Result: 'state_a' (<class 'str'>)
Expectation: result = 'state_a' (<class 'str'>)
```

B.1.14 Previous State was

Description
The Module shall have a method for checking if the given state was the previous state.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least two calls with different return values fit to the expectation.

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
```

Success	Returnvalue of previous_state_was(state_a) is correct (Content True and Type is <class 'bool'>).

```
Result: True (<class 'bool'>)
Expectation: result = True (<class 'bool'>)
```

Success	Returnvalue of previous_state_was(state_b) is correct (Content False and Type is <class 'bool'>).

```
Result: False (<class 'bool'>)
Expectation: result = False (<class 'bool'>)
```

B.1.15 Previous State Duration

Description
The Module shall have a method for getting active time for the previous state.

Reason for the implementation
Comfortable user interface.

Fitcriterion
At least one returned duration fits to the previous state duration ($\pm$ 0.05s).

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

Info	Running state machine test sequence.

```
StateMachine: State change ('__init__'): None -> 'state_a'
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Waiting for 0.75s
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
```

Success	Return Value of previous_state_duration() is correct (Content 0.7513992786407471 in [0.7 ... 0.8] and Type is <class 'float'>).

```
Result: 0.7513992786407471 (<class 'float'>)
Expectation: 0.7 <= result <= 0.8
```

B.1.16 State change callback for a defined transition and targetstate

Description
The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined set of *transition_condition* and *target_state*.

Reason for the implementation
Triggering state change actions for a specific transition condition and targetstate.

Fitcriterion
Methods are called in the registration order after state change with all user given arguments for the defined transition condition and targetstate and at least for one other condition not.

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

Info	Running state machine sequence and storing sequence number for each callback

```
StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by callback_execution
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 6 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
```

Success	List of the submitted values for Execution of state machine callback (1) (state_b, condition_a) identified by a sequence number is correct (Content [1] and Type is <class 'list'>).

```
Result: [ 1 ] (<class 'list'>)
Expectation: result = [ 1 ] (<class 'list'>)
```

Success	List of the submitted values for Execution of state machine callback (2) (state_b, condition_a) identified by a sequence number is correct (Content [2] and Type is <class 'list'>).

```
Result: [ 2 ] (<class 'list'>)
Expectation: result = [ 2 ] (<class 'list'>)
```

B.1.17 State change callback for a defined transition

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for a defined *transition_condition* and all *target_states*.

Reason for the implementation

Triggering state change actions for a specific transition condition.

Fitcriterion

Methods are called in the registration order after state change with all user given arguments for the defined transition condition and at least for one other transition condition not.

Testresult

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

Info	Running state machine sequence and storing sequence number for each callback

```
StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by callback_execution
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 6 caused by callback_execution
Increasing sequence number to 7 caused by callback_execution
Increasing sequence number to 8 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
```

Success	List of the submitted values for Execution of state machine callback (1) (all_transitions, condition_b) identified by a sequence number is correct (Content [2, 5] and Type is <class 'list'>).

```
Result: [ 2, 5 ] (<class 'list'>)
Expectation: result = [ 2, 5 ] (<class 'list'>)
```

Success	List of the submitted values for Execution of state machine callback (2) (all_transitions, condition_b) identified by a sequence number is correct (Content [3, 6] and Type is <class 'list'>).

```
Result: [ 3, 6 ] (<class 'list'>)
Expectation: result = [ 3, 6 ] (<class 'list'>)
```

B.1.18 State change callback for a defined targetstate

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all *transition_conditions* and a defined *target_state*.

Reason for the implementation

Triggering state change actions for a specific targetstate.

Fitcriterion

Methods are called in the registration order after state change with the defined targetstate and at least for one other targetstate not.

Testresult

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

Info	Running state machine sequence and storing sequence number for each callback

```
StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by callback_execution
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 6 caused by callback_execution
Increasing sequence number to 7 caused by callback_execution
Increasing sequence number to 8 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
```

Success	List of the submitted values for Execution of state machine callback (1) (state_b, all_conditions) identified by a sequence number is correct (Content [1, 5] and Type is <class 'list'>).

```
Result: [ 1, 5 ] (<class 'list'>)
Expectation: result = [ 1, 5 ] (<class 'list'>)
```

Success	List of the submitted values for Execution of state machine callback (2) (state_b, all_conditions) identified by a sequence number is correct (Content [2, 6] and Type is <class 'list'>).

```
Result: [ 2, 6 ] (<class 'list'>)
Expectation: result = [ 2, 6 ] (<class 'list'>)
```

B.1.19 State change callback for all kind of state changes

Description

The state machine shall call all registered methods in the same order like the registration with all user given arguments for all transitions.

Reason for the implementation

Triggering state change actions for all transition conditions and targetstates.

Fitcriterion

Methods are called in the registration order after state change.

Testresult

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

Info	Running state machine sequence and storing sequence number for each callback

```
StateMachine: State change ('__init__'): None -> 'state_a'
Increasing sequence number to 1 caused by sequence progress
StateMachine: State change ('condition_a'): 'state_a' -> 'state_b'
Increasing sequence number to 2 caused by callback_execution
Increasing sequence number to 3 caused by callback_execution
Increasing sequence number to 4 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_b' -> 'state_a'
Increasing sequence number to 5 caused by callback_execution
Increasing sequence number to 6 caused by callback_execution
Increasing sequence number to 7 caused by sequence progress
StateMachine: State change ('condition_b'): 'state_a' -> 'state_b'
Increasing sequence number to 8 caused by callback_execution
Increasing sequence number to 9 caused by callback_execution
Increasing sequence number to 10 caused by sequence progress
StateMachine: State change ('condition_c'): 'state_b' -> 'state_c'
Increasing sequence number to 11 caused by callback_execution
Increasing sequence number to 12 caused by callback_execution
```

Success	List of the submitted values for Execution of state machine callback (1) (all_transitions, all_conditions) identified by a sequence number is correct (Content [1, 4, 7, 10] and Type is <class 'list'>).

```
Result: [ 1, 4, 7, 10 ] (<class 'list'>)
```

```
Expectation: result = [ 1, 4, 7, 10 ] (<class 'list'>)
```

Success	List of the submitted values for Execution of state machine callback (2) (all_transitions, all_conditions) identified by a sequence number is correct (Content [2, 5, 8, 11] and Type is <class 'list'>).

```
Result: [ 2, 5, 8, 11 ] (<class 'list'>)
```

```
Expectation: result = [ 2, 5, 8, 11 ] (<class 'list'>)
```

C Test-Coverage

C.1 state_machine

The line coverage for state_machine was 100.0%
The branch coverage for state_machine was 100.0%

C.1.1 state_machine.__init__.py

The line coverage for state_machine.__init__.py was 100.0%
The branch coverage for state_machine.__init__.py was 100.0%

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
"""
state_machine (State Machine)
=============================

**Author:**

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

**Description:**

    This Module helps implementing state machines.

**Submodules:**

* :class:`state_machine.state_machine`

**Unittest:**

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

**Module Documentation:**

"""
__DEPENDENCIES__ = []

import logging
import time


```

```
logger_name = 'STATE_MACHINE'
logger = logging.getLogger(logger_name)


__INTERPRETER__ = (2, 3)
"""The supported Interpreter-Versions"""
__DESCRIPTION__ = """This Module helps implementing state machines."""
"""The Module description"""


class state_machine(object):
    """
    :param default_state: The default state which is set on initialisation.
    :param log_lvl: The log level, this Module logs to (see Loging-Levels of Module :mod:`logging`)

    .. note:: Additional keyword parameters well be stored as varibles of the instance (e.g. to give variables or methods for transition condition calculation).

    A state machine class can be created by deriving it from this class. The transitions are defined by overriding the variable `TRANSITIONS`.
    This Variable is a dictionary, where the key is the start-state and the content is a tuple or list of transitions. Each transition is a tuple or list
    including the following information: (condition-method (str), transition-time (number), target_state (str)).

    .. note:: The condition-method needs to be implemented as part of the new class.

    .. note:: It is usefull to define the states as variables of this class.


    **Example:**

    .. literalinclude:: ../examples/example.py

    .. literalinclude:: ../examples/example.log
    """
    TRANSITIONS = {}
    LOG_PREFIX = 'StateMachine:'

    def __init__(self, default_state, log_lvl, **kwargs):
        self.__state__ = None
        self.__last_transition_condition__ = None
        self.__conditions_start_time__ = {}
        self.__state_change_callbacks__ = {}
        self.__log_lvl__ = log_lvl
        self.__set_state__(default_state, '__init__')
        for key in kwargs:
            setattr(self, key, kwargs.get(key))

    def register_state_change_callback(self, state, condition, callback, *args, **kwargs):
        """
        :param state: The target state. The callback will be executed, if the state machine changes to this state. None means all states.
        :type state: str
        :param condition: The transition condition. The callback will be executed, if this condition is responsible for the state change. None means all conditions.
        :type condition: str
        :param callback: The callback to be executed.

        .. note:: Additional arguments and keyword parameters are supported. These arguments and parameters will be used as arguments and parameters for the callback execution.

        This methods allows to register callbacks which will be executed on state changes.
        """
```

```
90          if state not in self.__state_change_callbacks__:
91              self.__state_change_callbacks__[state] = {}
92          if condition not in self.__state_change_callbacks__[state]:
93              self.__state_change_callbacks__[state][condition] = []
94          self.__state_change_callbacks__[state][condition].append((callback, args, kwargs))
95
96      def this_state(self):
97          """
98          :return: The current state.
99
100         This method returns the current state of the state machine.
101         """
102         return self.__state__
103
104     def this_state_is(self, state):
105         """
106         :param state: The state to be checked
107         :type state: str
108         :return: True if the given state is currently active, else False.
109         :rtype: bool
110
111         This methods returns the boolean information if the state machine is currently in the
    given state.
112         """
113         return self.__state__ == state
114
115     def this_state_duration(self):
116         """
117         :return: The time how long the current state is active.
118         :rtype: float
119
120         This method returns the time how long the current state is active.
121         """
122         return time.time() - self.__time_stamp_state_change__
123
124     def last_transition_condition(self):
125         """
126         :return: The last transition condition.
127         :rtype: str
128
129         This method returns the last transition condition.
130         """
131         return self.__last_transition_condition__
132
133     def last_transition_condition_was(self, condition):
134         """
135         :param condition: The condition to be checked
136         :type condition: str
137         :return: True if the given condition was the last transition condition, else False.
138         :rtype: bool
139
140         This methods returns the boolean information if the last transition condition is
    equivalent to the given condition.
141         """
142         return self.__last_transition_condition__ == condition
143
144     def previous_state(self):
145         """
146         :return: The previous state.
147         :rtype: str
148
149         This method returns the previous state of the state machine.
150         """
```

```
151         return self.__prev_state__
152
153     def previous_state_was(self, state):
154         """
155         :param state: The state to be checked
156         :type state: str
157         :return: True if the given state was previously active, else False.
158         :rtype: bool
159
160         This methods returns the boolean information if the state machine was previously in the
    given state.
161         """
162         return self.__prev_state__ == state
163
164     def previous_state_duration(self):
165         """
166         :return: The time how long the previous state was active.
167         :rtype: float
168
169         This method returns the time how long the previous state was active.
170         """
171         return self.__prev_state_dt__
172
173     def __set_state__(self, target_state, condition):
174         logger.log(self.__log_lvl__, "%s State change (%s): %s -> %s", self.LOG_PREFIX, repr(
    condition), repr(self.__state__), repr(target_state))
175         timestamp = time.time()
176         self.__prev_state__ = self.__state__
177         if self.__prev_state__ is None:
178             self.__prev_state_dt__ = 0.
179         else:
180             self.__prev_state_dt__ = timestamp - self.__time_stamp_state_change__
181         self.__state__ = target_state
182         self.__last_transition_condition__ = condition
183         self.__time_stamp_state_change__ = timestamp
184         self.__conditions_start_time__ = {}
185         for callback, args, kwargs in self.__state_change_callbacks__.get(None, {}).get(None, [])
    :
186             callback(*args, **kwargs)
187         for callback, args, kwargs in self.__state_change_callbacks__.get(target_state, {}).get(
    None, []):
188             callback(*args, **kwargs)
189         for callback, args, kwargs in self.__state_change_callbacks__.get(None, {}).get(condition
    , []):
190             callback(*args, **kwargs)
191         for callback, args, kwargs in self.__state_change_callbacks__.get(target_state, {}).get(
    condition, []):
192             callback(*args, **kwargs)
193
194     def work(self):
195         """
196         This Method needs to be executed cyclicly to enable the state machine.
197         """
198         tm = time.time()
199         transitions = self.TRANSITIONS.get(self.this_state())
200         if transitions is not None:
201             active_transitions = []
202             cnt = 0
203             for method_name, transition_delay, target_state in transitions:
204                 method = getattr(self, method_name)
205                 if method():
206                     if method_name not in self.__conditions_start_time__:
207                         self.__conditions_start_time__[method_name] = tm
208                     if tm - self.__conditions_start_time__[method_name] >= transition_delay:
209                         active_transitions.append((transition_delay - tm + self.
    __conditions_start_time__[method_name], cnt, target_state, method_name))
```

```
210                 else:
211                     self.__conditions_start_time__[method_name] = tm
212                 cnt += 1
213             if len(active_transitions) > 0:
214                 active_transitions.sort()
215                 self.__set_state__(active_transitions[0][2], active_transitions[0][3])
```