# pyats-01 - Writing tests using aetest

# Syllabus

- installation
- structure of a test
- tests execution
- test's parameters
- execution results of a section
- test's steps

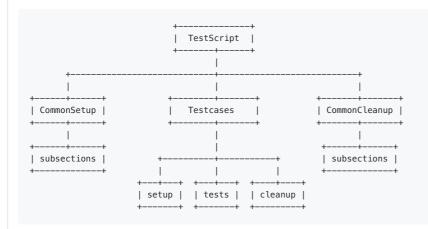
# Presentation

## Installation

pip install pyats.aetest

## Structure of a test

Logical scheme of aetest Python's module:



## Tests execution

```
# rabbit.py
from pyats.aetest import (
  Testcase, CommonSetup, CommonCleanup,
   subsection, setup, test, cleanup,
   main
class InstallRabbit(CommonSetup):
   @subsection
   def step_1(self):
      print('Download a Rabbit')
   @subsection
   def step_2(self):
      print('Rabbit is installed')
class SmokeTest(Testcase):
   def setup(self):
      print("A setup of smoke test")
   def test_1(self):
     print('Test #1')
   @test
   def test_2(self):
      print('Test #2')
   @cleanup
   def cleanup(self):
     print("A cleanup of smoke test")
class ConfigurationCheck(Testcase):
   def something(self):
      print('Rabbit is fine!')
class SmoothClean(CommonCleanup):
  @subsection
   def cleanup(self):
     print('Do nothing')
if __name__ == '__main__':
  main()
```

CLI execution:

```
python rabbit.py
```

# Test's parameters

Default parameters sample

```
# parameters.py
from pyats.aetest import Testcase, test, main
parameters = {
     'a': 10,
     'b': 2
class SmokeTest(Testcase):
    parameters = {'c': 3}
    @test
    def test_1(self, a, b):
       print('c: ', self.parameters['c'])
print('all: ', self.parameters)
assert a > b, '"a" has to be greater then "b"'
class Redefine(Testcase):
   parameters = {'a': 1}
    @test
    def test_1(self, a, b):
       print(self.parameters)
        assert a > b, '"a" has to be greater then "b"'
if __name__ == '__main__':
    main()
```

Required input parameters sample

```
# configs.py
import sys
import argparse
from pyats.aetest import Testcase, test, main
class SmokeTest(Testcase):
    @test
    def test_1(self, a, b):
       print(self.parameters)
        assert a > b, '"a" has to be greater then "b"'
if __name__ == '__main__':
   parser = argparse.ArgumentParser(description="standalone parser")
    parser.add_argument('-a', dest='bigger', type=int, required=True)
parser.add_argument('-b', dest='smaller', type=int, required=True)
    # do the parsing
    # always use parse_known_args, as aetest needs to parse any
    # remainder arguments that this parser does not understand
    args, sys.argv[1:] = parser.parse_known_args(sys.argv[1:])
    # and pass all arguments to aetest.main() as kwargs
    main(a=args.bigger, b=args.smaller)
```

CLI execution:

```
python configs.py python configs.py -a 1 -b 23
```

#### Execution results of a section

## Result APIs

- TestItem.passed(reason, goto, from\_exception)
- TestItem.failed(reason, goto, from\_exception)
- TestItem.errored(reason, goto, from\_exception)
- TestItem.skipped(reason, goto, from\_exception)
- TestItem.blocked(reason, goto, from\_exception)
- TestItem.aborted(reason, goto, from\_exception)

• TestItem.passx(reason, goto, from\_exception)

All results APIs accept the following optional arguments:

- reason, describing the conditions & reasons of why this result is provided
- goto , list of sections to "go to" after this section
- from\_exception, accepts an exception object and will add the traceback of this exception to the result's reason

```
# api.py
from pyats.aetest import Testcase, test, main
class SmokeTest(Testcase):
   def test_passed(self):
       self.passed('passed')
   def test_failed(self):
       self.failed('failed because of ...')
   def test_errored(self):
       self.errored('Hmmmm....', from_exception=ValueError('Nothing! But needs something!'))
   def test_skipped(self):
       self.skipped('Someone other is guilty!')
   def test_blocked(self):
       self.blocked('I need a pen....')
   @test
   def test_aborted(self):
       self.aborted('I do not want to work!')
   def test_passx(self):
       self.passx('passx!!!')
if __name__ == '__main__':
   main()
```

## Test's steps

```
# steps.py
from pyats.aetest import Testcase, test, main

class SmokeTest(Testcase):
    @test
    def test(self, steps):
        with steps.start('the passed step') as step:
            step.passed('passed')

    with steps.start('Allow next steps', continue_=True) as step:
        step.failed('reason is ...')

    with steps.start('Sub-steps') as long_step:
        with long_step.start('the failed sub-step') as step1:
        step1.failed('reason is ...')

    with long_step.start("Won't be executed") as step2:
        step2.passx('reason is ...')

if __name__ == '__main__':
    main()
```

Steps' results APIs contain all methods from section's API, but only reason can be specified.

## Additional materials

• https://pubhub.devnetcloud.com/media/pyats/docs/aetest/introduction.html

- https://pubhub.devnetcloud.com/media/pyats/docs/aetest/structure.html
- https://pubhub.devnetcloud.com/media/pyats/docs/aetest/parameters.html
- https://pubhub.devnetcloud.com/media/pyats/docs/aetest/results.html
- https://pubhub.devnetcloud.com/media/pyats/docs/aetest/run.html

# Control

## Task description

Write one test for each function located in calculation.py module. The tests have to be a part of one class. Also, a user needs to pass num1 and num2 while running the tests. By default, num1 = 3, num2 = 0. If there are math errors (like ZeroDivisionError), make a test as passx. If the calculated value is less than 0, please skip the test. All other cases have to be considered as passed.

```
# calculation.py
def add(num1, num2):
    return num1 + num2

def divide(num1, num2):
    return num1 / num2
```

## Review items

Please send a Python's module with tests and a command to run them.