pyats-04 - Implementing data-driven testing

Syllabus

- sections and looping
- loop the parameters
- loop the modules with tests

Presentation

Sections and looping

The mapping between sections and loops

```
# sections.py
from pyats.aetest import (
    Testcase, CommonSetup, CommonCleanup,
    subsection, setup, test, cleanup, loop,
    main
)

class Setup(CommonSetup):
    @loop(uids=['up_one', 'up_two'])
    @subsection
    def up(self):
        pass


@loop(uids=['t1', 't2'])
class Test(Testcase):

    @setup
    def test_up(self):
        pass

    @loop(uids=['check1', 'check2'])
    @test
    def check(self):
        pass

    @cleanup
    def test_cleanup(self):
        pass


class Cleanup(CommonCleanup):
    @loop(uids=['clean_one', 'clean_two'])
    @subsection
    def cleanup(self):
        pass


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

Loop parameters

Demonstration of `uids`, parameters groups and per-parameters mapping.

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


@loop(a=[1, 2, 3])
class Test1(Testcase):
    @test.loop(b=['a', 'b', 'c'])
    def only_parameters(self, a, b):
        pass


class Uids(Testcase):
    @test.loop(uids=['l1', 'l2'],
               a=[1, 2, 3])
    def less_uids(self, a):
        pass

    @test.loop(uids=['e1', 'e2', 'e3'],
               a=[1, 2, 3])
    def equal_uids(self, a):
        pass

    @test.loop(uids=['m1', 'm2', 'm3', 'm4'],
               a=[1, 2, 3])
    def more_uids(self, a):
        pass


class Definitions(Testcase):
    @test.loop(a=[1, 2, 3],
               b=['a', 'b', 'c'])
    def per_parameter(self, a, b):
        pass

    @test.loop(args=['a', 'b'],
               argvs=[(1, 'a'), (2, 'b'), (3, 'c')])
    def groups(self, a, b):
        pass


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

Looping with functions (`callable` , `iterable` , or a `generator`)

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


class Test(Testcase):
    @test.loop(a=range(10))
    def check(self, a):
        pass


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

Loop the modules with tests

Testbed file parametrization

```yaml
# test-env.yaml
devices:
    main:
        alias: master
        os: 'linux'
        type: 'linux'
        connections:
          linux:
            protocol: ssh
            ip: 192.168.242.44
        custom:
          size: 10
          tags: [a,b,smoke]

# dev-env.yaml
devices:
    main:
        alias: master
        os: 'linux'
        type: 'linux'
        connections:
          linux:
            protocol: ssh
            ip: 192.168.242.55
        custom:
          size: 100
          tags: [a,b]
```

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


class Smoke(Testcase):
    @test
    def test_one(self, testbed):
        print('alias: ', testbed.devices['main'].alias)
        print('size: ', testbed.devices['main'].custom.size)


if __name__ == '__main__':
    main()

# tb_job.py
from pyats.easypy import run


def main():
    run(testscript='tb.py')
```

```
easypy tb_job.py -testbed_file test-env.yaml
easypy tb_job.py -testbed_file dev-env.yaml
```

Tasks parametrization

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


class Smoke(Testcase):
    @test
    def both(self, foo, bar):
        print(f'foo: {foo}; bar: {bar}')

    @test
    def foo(self, foo):
        print(f'foo: {foo}')

    @test
    def bar(self, bar):
        print(f'bar: {bar}')


if __name__ == '__main__':
    main()

# tasks_job.py
from pyats.easypy import Task


def main():
    tasks = (
        Task(taskid='single letter',
                    testscript='tasks.py', foo='f', bar='b'),
        Task(taskid='single number',
                    testscript='tasks.py', foo='2', bar='5'),
        Task(taskid='long string',
                    testscript='tasks.py', foo='dfgdf', bar='mnfgbgfnbghn'),
        Task(taskid='long numbers',
                    testscript='tasks.py', foo='555', bar='34'),
        Task(taskid='empty data',
                    testscript='tasks.py', foo='', bar=''),
    )
    for task in tasks:
        task.start()
        task.join()
```

```
easypy tasks_job.py
```

Additional materials

- https://pubhub.devnetcloud.com/media/pyats/docs/aetest/loop.html
- https://pubhub.devnetcloud.com/media/pyats/docs/aetest/examples.html#script-arguments
- https://pubhub.devnetcloud.com/media/pyats/docs/aetest/examples.html#mega-looping
- https://pubhub.devnetcloud.com/media/pyats/docs/easypy/jobfile.html#tasks

Control

Task description

There is the following pseudo-module:

```
# tests.py
mapping = {
    'a': (1,3,4,5,6,7,8),
    'b': (0,2,3,4),
    'c': (7,9,0,6,5,4,3,1),
}

class Test(Testcase):

    @setup
    def setup(self, letter):
        pass

    @test
    def check(self, number):
        print(f'number: {number}')
```

You need to use this module as a base for implementing the following logic. A user with run `python tests.py --letter a` command and expect to see looping of `check` test as many times as many numbers are present in the mapping dictionary for the given letter.

Review items

Please send the `tests.py` module. If you create other files, please send them also.