pyats-02 - Managing suites of the tests

Syllabus

- installation
- structure of a job file
- `easypy` CLI options
- extend `easypy` options
- advanced suites with `Task`s

Presentation

Installation

```
pip install pyats.easypy
```

Structure of a job file

A test file

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


class Smoke(Testcase):
    @test
    def test_one(self):
        print('Alisa is fine!')


class Health(Testcase):
    @test
    def status(self):
        print('Alisa\'s health is fine!')


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

A job file

```
# alisa_job.py
from pyats.easypy import run


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

Job's execution

```
easypy alisa_job.py
```

`easypy` CLI options

```
easypy -help
```

Extend `easypy` options

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


class Smoke(Testcase):
    @test
    def test(self, duration, interval):
        print(f'Duration: {duration}; interval: {interval}')
        if duration > 50:
            self.failed('Too big duration!')
        time.sleep(interval)


if __name__ == '__main__':
    main(duration=23, interval=1)

# options.py
import sys
import argparse
from pyats.easypy import run

# according to pyATS documentation, define arguments here
parser = argparse.ArgumentParser(description="Possible configuration options")
parser.add_argument('--duration', required=True, help='How long ...')
parser.add_argument('--interval', required=True, help='How often ...')

def main():
    # according to pyATS documentation, parse arguments in the `main` method
    args, sys.argv[1:] = parser.parse_known_args(sys.argv[1:])
    print(args)
    run(testscript='arguments.py', duration=args.duration)
```

```
easypy options.py --help
easypy options.py --duration 100 --interval 2
```

Advanced suites with `Task`s

`easypy.run` API

```
from pyats.easypy import run
help(run)
```

```
# runs.py
from pyats.easypy import run


def main(runtime):

    # using run() api to run a task, save the result to variable
    # (max_runtime = 60*5 seconds = 5 minutes)
    for duration in range(0,100,20):
        result = run(testscript='arguments.py',
                     runtime=runtime,
                     taskid=f'duration [{duration}] | interval [2]',
                     max_runtime=60*5,
                     duration=duration, interval=2)

        # check whether the next script should continue
        # based on previous task's results.
        if not result:
            break
```

`easypy.Task` API

Parallel tests execution

```
# tasks.py
import time
from pyats.easypy import Task


def new_task(runtime, duration):
    return Task(testscript='arguments.py', runtime=runtime, duration=duration, interval=10)

def main(runtime):
    long_task = new_task(runtime, 30)
    tasks = [new_task(runtime, duration) for duration in range(0,100,10)]

    long_task.start()
    long_task.join()

    # start tasks in parallel
    for task in tasks:
        task.start()

    # wait for completion
    for task in tasks:
        if task.is_alive():
            task.wait()
```

Additional materials

- https://pubhub.devnetcloud.com/media/pyats/docs/easypy/introduction.html#installation-updates
- https://pubhub.devnetcloud.com/media/pyats/docs/easypy/usages.html
- https://pubhub.devnetcloud.com/media/pyats/docs/easypy/usages.html#argument-propagation
- https://pubhub.devnetcloud.com/media/pyats/docs/easypy/jobfile.html
- https://pubhub.devnetcloud.com/media/pyats/docs/easypy/jobfile.html#custom-arguments
- CiscoTestAutomation/pyats#4

Control

Task description

There are two numbers. Each of them can be negative, floating etc. You need to create a job which will accept two numbers as the arguments. This job has to run in parallel 4 different tests: multiplication, division, addition, subtraction. Each of these tests will perform an appropriate math operation on given numbers. A test has to fail if the calculation result is less then 0, otherwise, the test is passed.

Review items

Please send Python's modules (tests and job file) as well as a command to run the job.