Residência em Software Automotivo

Trabalho de Noções de Automação de Testes
Framework PyUnit / Unittest

PyUnit / Unittest

- Biblioteca padrão do Python.
- Testes para funções e classes.
- Suporte automação de testes.
- Não possui métodos parametrizáveis,pode ser implementado de maneira customizada.
- Exige syntax padrões para chamar sua execução.
- Não possui suporte de cobertura, necessário o uso de outra ferramenta.

```python
class Calculator():

    def addition(self, a, b):
        return a + b

    def addition2(self, a, b, c):
        return a + b + c

    def addition3(self, a, b, c):
        return a + b + c

    def subtraction(self, a, b):
        return a - b

    def multiplication(self, a, b):
        return a * b

    def division(self, a, b):
        return a / b

class Aquarium():
    def add_fish_to_aquarium(self, fish_list):
        if len(fish_list) > 10:
            raise ValueError("A maximum of 10 fish can be added to the aquarium")
        return {"tank_a": fish_list}
```

PyUnit / Unittest

- Biblioteca padrão do Python.
- Testes para funções e classes.
- Suporte automação de testes.

```python
import unittest
from mainClass import Calculator, Aquarium
from pyunitreport import HTMLTestRunner

class TestCalculator(unittest.TestCase):


    def setUp(self): # Preparation code that run before each Test.
        self.calc = Calculator()

    def tearDown(self): # Clean-up all the registry and modification after each test completes
        self.calc = Calculator()

    def setUpModule(self):
        self.calc = Calculator()
        pass

    def tearDownModule():
        pass

    def test_addition(self):
        actual =  self.calc.addition(2,2)
        expected = 4
        self.assertEqual(actual, expected)  # Assert is the method of class that do the TesCase
```

```python
class TestAddFishToAquarium(unittest.TestCase):

    def setUp(self): # Preparation code that run before each Test
        self.aqua = Aquarium()

    def tearDown(self): # Clean-up all the registry and modification after each test completes
        self.aqua = Aquarium()

    def test_add_fish_to_aquarium_success(self):
        actual = self.aqua.add_fish_to_aquarium(fish_list=["shark", "tuna"])
        expected = {"tank_a": ["shark", "tuna"]}
        self.assertEqual(actual, expected)

    def test_add_fish_to_aquarium_success2(self):
        actual = self.aqua.add_fish_to_aquarium(fish_list=["shark", "tuna"])
        expected = {"tank_a": ["error", "tuna"]}
        self.assertEqual(actual, expected)

    def test_add_fish_to_aquarium_exception(self): # This test intend to cause exception at the fu
        too_many_fish = ["shark"] * 25
        with self.assertRaises(ValueError) as exception_context:
            self.aqua.add_fish_to_aquarium(fish_list=too_many_fish)
        self.assertEqual(str(exception_context.exception),
                         "A maximum of 10 fish can be added to the aquarium")

if __name__ == '__main__':
    unittest.main (verbosity=2)
    #unittest.main (verbosity=2,testRunner=HTMLTestRunner(output='C:/Git/Unittest'))
```

PyUnit / Unittest

SETUP:

Terminal Git Bash / Visual Code:
cd c:
mkdir Git (Se não houver pasta)
cd Git
git clone https://github.com/chlo1997/Unittest.git
cd Unittest

PyUnit / Unittest

COMMAND LINE:

python -m unittest -h

```
Usage: python -m unittest [options] [tests]

Options:
  -h, --help       Show this message
  -v, --verbose    Verbose output
  -q, --quiet      Minimal output
  -f, --failfast   Stop on first failure
  -c, --catch      Catch control-C and display results
  -b, --buffer     Buffer stdout and stderr during test runs

Examples:
  python -m unittest test_module               - run tests from test_module
  python -m unittest module.TestClass          - run tests from module.TestClass
  python -m unittest module.Class.test_method  - run specified test method

[tests] can be a list of any number of test modules, classes and test
methods.

Alternative Usage: python -m unittest discover [options]

Options:
  -v, --verbose    Verbose output
  -f, --failfast   Stop on first failure
  -c, --catch      Catch control-C and display results
  -b, --buffer     Buffer stdout and stderr during test runs
  -s directory     Directory to start discovery ('.' default)
  -p pattern       Pattern to match test files ('test*.py' default)
  -t directory     Top level directory of project (default to
                   start directory)

For test discovery all test modules must be importable from the top
level directory of the project.
```

PyUnit / Unittest

SETUP:

Run todo o teste dentro do diretório:

python -m unittest test_mainClass

```
$ python -m unittest test_mainClass
..F..s.
======================================================================
FAIL: test_add_fish_to_aquarium_success2 (test_mainClass.TestAddFishToAquarium)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_mainClass.py", line 58, in test_add_fish_to_aquarium_success2
    self.assertEqual(actual, expected)
AssertionError: {'tank_a': ['shark', 'tuna']} != {'tank_a': ['error', 'tuna']}
- {'tank_a': ['shark', 'tuna']}
?              ^^^ ^

+ {'tank_a': ['error', 'tuna']}
?              ^ ^^^


----------------------------------------------------------------------
Ran 7 tests in 0.003s

FAILED (failures=1, skipped=1)
```

PyUnit / Unittest

SETUP:

Run modo descoberta, se houver em pastas distinta, necessita __init__.py:
python -m unittest discover

```
$ python -m unittest discover
..F..s.
======================================================================
FAIL: test_add_fish_to_aquarium_success2 (test_mainClass.TestAddFishToAquarium)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Git\Unittest\test_mainClass.py", line 58, in test_add_fish_to_aquarium_succe
ss2
    self.assertEqual(actual, expected)
AssertionError: {'tank_a': ['shark', 'tuna']} != {'tank_a': ['error', 'tuna']}
- {'tank_a': ['shark', 'tuna']}
?              ^^^ ^

+ {'tank_a': ['error', 'tuna']}
?              ^ ^^^


----------------------------------------------------------------------
Ran 7 tests in 0.002s

FAILED (failures=1, skipped=1)
```

PyUnit / Unittest

SETUP:
Run o teste dentro do diretório com verbosidade:
python -m unittest -v test_mainClass

```
$ python -m unittest -v  test_mainClass
test_add_fish_to_aquarium_exception (test_mainClass.TestAddFishToAquarium) ... o
k
test_add_fish_to_aquarium_success (test_mainClass.TestAddFishToAquarium) ... ok
test_add_fish_to_aquarium_success2 (test_mainClass.TestAddFishToAquarium) ... FA
IL
test_addition (test_mainClass.TestCalculator) ... ok
test_addition2 (test_mainClass.TestCalculator) ... ok
test_addition3 (test_mainClass.TestCalculator) ... skipped 'demonstrating skippi
ng'
test_subtraction (test_mainClass.TestCalculator) ... ok

======================================================================
FAIL: test_add_fish_to_aquarium_success2 (test_mainClass.TestAddFishToAquarium)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_mainClass.py", line 58, in test_add_fish_to_aquarium_success2
    self.assertEqual(actual, expected)
AssertionError: {'tank_a': ['shark', 'tuna']} != {'tank_a': ['error', 'tuna']}
- {'tank_a': ['shark', 'tuna']}
?             ^^^ ^

+ {'tank_a': ['error', 'tuna']}
?             ^ ^^^


----------------------------------------------------------------------
Ran 7 tests in 0.010s

FAILED (failures=1, skipped=1)
```

PyUnit / Unittest

SETUP:
Run Classe ou Teste específico
python -m unittest -v test_mainClass.TestCalculator

python -m unittest -v test_mainClass.TestCalculator.test_addition

```
$ python -m unittest  -v test_mainClass.TestCalculator
test_addition (test_mainClass.TestCalculator) ... ok
test_addition2 (test_mainClass.TestCalculator) ... ok
test_addition3 (test_mainClass.TestCalculator) ... skipped 'demonstrating skipping'
test_subtraction (test_mainClass.TestCalculator) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.006s

OK (skipped=1)
```

```
$ python -m unittest -v  test_mainClass.TestCalculator.test_addition
test_addition (test_mainClass.TestCalculator) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

PyUnit / Unittest

Assertions: Métodos que podem ser utilizado nos teste de casos.

Method	Checks that
`assertEqual(a, b)`	`a == b`
`assertNotEqual(a, b)`	`a != b`
`assertTrue(x)`	`bool(x) is True`
`assertFalse(x)`	`bool(x) is False`
`assertIs(a, b)`	`a is b`
`assertIsNot(a, b)`	`a is not b`
`assertIsNone(x)`	`x is None`
`assertIsNotNone(x)`	`x is not None`
`assertIn(a, b)`	`a in b`
`assertNotIn(a, b)`	`a not in b`
`assertIsInstance(a, b)`	`isinstance(a, b)`
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`