НАПИСАННЯ ТЕСТОВИХ НАБОРІВ ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ UNITTEST

Питання 8.2

Doctest має обмежену популярність через відсутність API та конфігурування тестів

- unittest (сторонній модуль PyUnit) – портований JUnit для Python.
  - Починаючи з Python, модуль PyUnit став частиною Python Standard library.
  - Пізніше перейменовано в unittest.
  - Unittest is the batteries-included test automation library of Python, which means you do not have to install an additional library or tool in order to start using it.
- Багато мов програмування мають більше одного xUnit-подібного фреймворку.
  - У Java є TestNG, крім JUnit.
  - Python також має nose, pytest, Nose2.

Загальна архітектура

- Основні компоненти:
  - ***Клас тестового випадку (Test case class)***: базовий клас усіх тестових класів у тестовому модулі. Всі тестові класи походять від нього.
  - ***Тестові фікстури (Test fixtures)***: функції чи методи, які запускаються до чи після блоків тестового коду.
  - ***Твердження (Assertions)***: функції чи методи, які використовуються для перевірки поведінки тестованого компоненту. Більшість xUnit-подібних фреймворків постачаються з потужними assertion-методами.
  - ***Тестовий набір (Test suite)***: колекція чи група пов'язаних тестів, яку можна виконати чи запланувати на запуск разом.
  - ***Запускалка тесту (Test runner)***: програма чи блок коду, який запускає тестовий набір.
  - ***Форматувальник тестового результату (Test result formatter)***: форматує результати тестів, щоб вивести інформацію в читабельному форматі, зокрема тексті, HTML, and XML.

Використання Unittest

```python
import unittest
class TestClass01(unittest.TestCase):
    def test_case01(self):
        my_str = "Stanisav"
        my_int = 999
        self.assertTrue(isinstance(my_str, str))
        self.assertTrue(isinstance(my_int, int))

    def test_case02(self):
        my_pi = 3.14
        self.assertFalse(isinstance(my_pi, int))

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

- unittest.main() – це запускалка тестів.

```
(C:\Program Files\Anaconda3) C:\Users\User>python D:\pythonTest\test_module01.py -v
test_case01 (__main__.TestClass01) ... ok
test_case02 (__main__.TestClass01) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK

(C:\Program Files\Anaconda3) C:\Users\User>python D:\pythonTest\test_module01.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK

(C:\Program Files\Anaconda3) C:\Users\User>
```

Порядок виконання тестових методів

```python
import unittest
import inspect
class TestClass02(unittest.TestCase):
    def test_case02(self):
        print("\nRunning Test Method : " + inspect.stack()[0][3])
    def test_case01(self):
        print("\nRunning Test Method : " + inspect.stack()[0][3])

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

```
..
Running Test Method : test_case01

Running Test Method : test_case02

----------------------------------------------------------------------
Ran 2 tests in 0.146s

OK
```

- Метод inspect.stack()[0][3] виводить назву поточного тестового методу.
  - Корисно, якщо важливий порядок, у якому виконуються методи з тестового класу.
  - Тестові методи запускаються в алфавітномзу порядку!

Контроль Verbosity

```python
import unittest
import inspect

def add(x, y):
    print("We're in custom made function : " + inspect.stack()[0][3])
    return(x + y)

class TestClass03(unittest.TestCase):
    def test_case01(self):
        print("\nRunning Test Method : " + inspect.stack()[0][3])
        self.assertEqual(add(2, 3), 5)

    def test_case02(self):
        print("\nRunning Test Method : " + inspect.stack()[0][3])
        my_var = 3.14
        self.assertTrue(isinstance(my_var, float))

    def test_case03(self):
        print("\nRunning Test Method : " + inspect.stack()[0][3])
        self.assertEqual(add(2, 2), 5)

    def test_case04(self):
        print("\nRunning Test Method : " + inspect.stack()[0][3])
        my_var = 3.14
        self.assertTrue(isinstance(my_var, int))

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

```
....FF
Running Test Method : test_case01

Running Test Method : test_case02

Running Test Method : test_case01
We're in custom made function : add

Running Test Method : test_case02

Running Test Method : test_case03
We're in custom made function : add

Running Test Method : test_case04

======================================================================
FAIL: test_case03 (__main__.TestClass03)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/spuasson/Desktop/test_module03.py", line 20, in test_case03
    self.assertEqual(add(2, 2), 5)
AssertionError: 4 != 5

======================================================================
FAIL: test_case04 (__main__.TestClass03)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:/Users/spuasson/Desktop/test_module03.py", line 25, in test_case04
    self.assertTrue(isinstance(my_var, int))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 6 tests in 0.100s

FAILED (failures=2)
```

Кілька тестових класів в одному Test файлі/модулі

- .py-файл, який містить тестовий клас, також називають тестовим модулем.

```
..
Classname : TestClass04
Running Test Method : test_case01

Classname : TestClass05
Running Test Method : test_case01

----------------------------------------------------------------------
Ran 2 tests in 0.146s

OK
```

```
1  import unittest
2  import inspect
3
4
5  class TestClass04(unittest.TestCase):
6
7          def test_case01(self):
8                  print("\nClassname : " + self.__class__.__name__)
9                  print("Running Test Method : " + inspect.stack()[0][3])
10
11
12 class TestClass05(unittest.TestCase):
13
14         def test_case01(self):
15                 print("\nClassname : " + self.__class__.__name__)
16                 print("Running Test Method : " + inspect.stack()[0][3])
17
18 if __name__ == '__main__':
19         unittest.main(verbosity=2)
```

```python
import unittest

def setUpModule():
    """called once, before anything else in this module"""
    print("In setUpModule()...")


def tearDownModule():
    """called once, after everything else in this module"""
    print("In tearDownModule()...")


class TestClass06(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        """called once, before any test"""
        print("In setUpClass()...")

    @classmethod
    def tearDownClass(cls):
        """called once, after all tests, if setUpClass successful"""
        print("In tearDownClass()...")

    def setUp(self):
        """called multiple times, before every test method"""
        print("\nIn setUp()...")

    def tearDown(self):
        """called multiple times, after every test method"""
        print("In tearDown()...")

    def test_case01(self):
        self.assertTrue("PYTHON".isupper())
        print("In test_case01()")

    def test_case02(self):
        self.assertFalse("python".isupper())
        print("In test_case02()")

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

Тестові фікстури (Test Fixtures)

- Набір дій до та після виконання тестів.
  - У unittest вони реалізовані як методи класу TestCase з можливістю переозначення для власних цілей.
- Методи setUpModule() та tearDownModule() є фікстурами рівня модуля.
  - setUpModule() виконується до будь-якого методу в тестовому модулі.
  - tearDownModule() виконується після всіх методів тестового модуля.
- setUpClass() та tearDownClass() є фікстурами рівня класу.

```
....In setUpModule()...

Classname : TestClass04
Running Test Method : test_case01

Classname : TestClass05
Running Test Method : test_case01
In setUpClass()...

In setUp()...
In test_case01()
In tearDown()...

In setUp()...
In test_case02()
In tearDown()...
In tearDownClass()...
In tearDownModule()...

----------------------------------------
Ran 4 tests in 0.032s

OK
```

- Декоратор @classmethod повинен мати посилання на об’єкт класу (перший параметр).
  - setUp() та tearDown() – фікстури рівня методу.
  - setUp() та tearDown() – методи, які виконуються до та після кожного тестового методу в тестовому класі.

Контроль гранульованості виконання тестів

- Можна запустити один тестовий клас таким чином:
- python3 -m unittest -v test_module04.TestClass04

```
test_case01 (test_module04.TestClass04) ...
Classname : TestClass04
Running Test Method : test_case01
ok

----------------------------------------------------
Ran 1 test in 0.077s

OK
```

- python3 -m unittest -v test_module04.TestClass05.test_case01

```
test_case01 (test_module04.TestClass05) ...
Classname : TestClass05
Running Test Method : test_case01
ok

----------------------------------------------------
Ran 1 test in 0.077s

OK
```

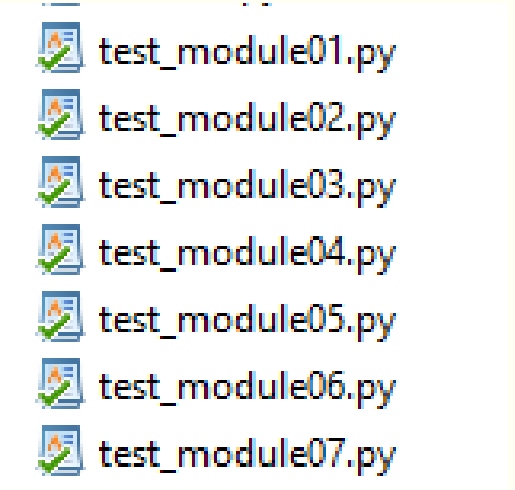
```
(C:\ProgramData\Anaconda3) C:\Users\spuasson>python -m unittest -h
usage: python.exe -m unittest [-h] [-v] [-q] [--locals] [-f] [-c] [-b]
                              [tests [tests ...]]

positional arguments:
  tests           a list of any number of test modules, classes and test
                  methods.

optional arguments:
  -h, --help      show this help message and exit
  -v, --verbose   Verbose output
  -q, --quiet     Quiet output
  --locals        Show local variables in tracebacks
  -f, --failfast  Stop on first fail or error
  -c, --catch     Catch Ctrl-C and display results so far
  -b, --buffer    Buffer stdout and stderr during tests

Examples:
  python.exe -m unittest test_module               - run tests from test_module
  python.exe -m unittest module.TestClass          - run tests from module.TestClass
  python.exe -m unittest module.Class.test_method  - run specified test method
  python.exe -m unittest path/to/test_file.py      - run tests from test_file.py

usage: python.exe -m unittest discover [-h] [-v] [-q] [--locals] [-f] [-c]
                                       [-b] [-s START] [-p PATTERN] [-t TOP]

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Verbose output
  -q, --quiet           Quiet output
  --locals              Show local variables in tracebacks
  -f, --failfast        Stop on first fail or error
  -c, --catch           Catch Ctrl-C and display results so far
  -b, --buffer          Buffer stdout and stderr during tests
  -s START, --start-directory START
                        Directory to start discovery ('.' default)
  -p PATTERN, --pattern PATTERN
                        Pattern to match tests ('test*.py' default)
  -t TOP, --top-level-directory TOP
                        Top level directory of project (defaults to start
                        directory)

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

Опції командного рядка та довідка

- Ключ -q передбачає тихий (quiet) режим.
- Ключ -f означає відмовобезпечність (failsafe).
  - Змушує зупинити виконання, як тільки тест провалюється.

Створення тестового пакету

- Можна використовувати вбудовану функціональність Python з формування пакетів, щоб створити тестовий пакет.
- У папці з тестами створіть файл __init__.py.
  - За потреби додати новий тестовий модуль у пакет потрібно дописати в нього назву цього модуля.




```
all = ["test_module01", "test_module02", "test_module03", "test_module04",
"test_module05", "test_module06", "test_module07"]
```

  - test (назва папки) – назва тестового пакету та всіх модулів, згаданих у __init__.py belong to this package.

```
python3 -m unittest -v test.test_module04
```

```
test_case01 (test.test_module04.TestClass04) ...
Classname : TestClass04
Running Test Method : test_case01
ok
test_case01 (test.test_module04.TestClass05) ...
Classname : TestClass05
Running Test Method : test_case01
ok

----------------------------------------------------------------------
Ran 2 tests in 0.090s

OK
```

```
python3 -m unittest -v
test.test_module04.TestClass04.test_case01
```

The output is as follows:

```
test_case01 (test.test_module04.TestClass04) ...
Classname : TestClass04
Running Test Method : test_case01
ok

----------------------------------------------------------------------
Ran 1 test in 0.079s

OK
```

```
python3 -m unittest -v test.test_module04.TestClass04
```

```
test_case01 (test.test_module04.TestClass04) ...
Classname : TestClass04
Running Test Method : test_case01
ok

----------------------------------------------------------------------
Ran 1 test in 0.078s

OK
```

```python
def add(x, y):
    return(x + y)


def mul(x, y):
    return(x * y)


def sub(x, y):
    return(x - y)


def div(x, y):
    return(x / y)
```

Організація коду

- Створимо папки dev та test в одній директорії.
  - У папці test створимо модуль test_me.py
- Цей модуль можна імпортувати в інші модулі з цієї папки, використовуючи інструкцію
  - import test_me

```python
import unittest
import test_me


class TestClass09(unittest.TestCase):

    def test_case01(self):
        self.assertEqual(test_me.add(2, 3), 5)
        print("\nIn test_case01()")

    def test_case02(self):
        self.assertEqual(test_me.mul(2, 3), 6)
        print("\nIn test_case02()")
```

```
python3 -m unittest -v test_module08
 test_case01 (test_module08.TestClass09) ...
In test_case01()
ok
test_case02 (test_module08.TestClass09) ...
In test_case02()
ok

----------------------------------------------------------------------
Ran 2 tests in 0.004s

OK
```

Розміщення Development and Test коду в окремих папках

```
pi@raspberrypi:~/book/code/chapter03 $ tree
.
├── mypackage
│   ├── __init__.py
│   ├── mymathlib.py
│   └── mymathsimple.py
└── test
    ├── __init__.py
    ├── test_me.py
    ├── test_module01.py
    ├── test_module02.py
    ├── test_module03.py
    ├── test_module04.py
    ├── test_module05.py
    ├── test_module06.py
    ├── test_module07.py
    └── test_module08.py
```

```python
class mymathlib:
    def __init__(self):
        """Constructor for this class..."""
        print("Creating object : " + self.__class__.__name__)

    def add(self, x, y):
        return(x + y)

    def mul(self, x, y):
        return(x * y)

    def mul(self, x, y):
        return(x - y)

    def __del__(self):
        """Destructor for this class..."""
        print("Destroying object : " + self.__class__.__name__)
```

```python
def add(x, y):
    return(x + y)


def mul(x, y):
    return(x * y)


def sub(x, y):
    return(x - y)


def div(x, y):
    return(x / y)
```

mymathsimple.py

- Створені модулі відносяться до development-модулів.
- Для створення пакету додаємо новий файл __init__.py до development modules:
  - all = ["mymathlib", "mymathsimple"]

test_module09.py

```python
from mypackage.mymathlib import *
import unittest

math_obj = 0

def setUpModule():
        """called once, before anything else in the module"""
        print("In setUpModule()...")
        global math_obj
        math_obj = mymathlib()


def tearDownModule():
        """called once, after everything else in the module"""
        print("In tearDownModule()...")
        global math_obj
        del math_obj

```

```
19 class TestClass10(unittest.TestCase):
20
21         @classmethod
22         def setUpClass(cls):
23                 """called only once, before any test in the class"""
24                 print("In setUpClass()...")
25
26         def setUp(self):
27                 """called once before every test method"""
28                 print("\nIn setUp()...")
29
30         def test_case01(self):
31                 print("In test_case01()")
32                 self.assertEqual(math_obj.add(2, 5), 7)
33
34         def test_case02(self):
35                 print("In test_case02()")
36
37         def tearDown(self):
38                 """called once after every test method"""
39                 print("In tearDown()...")
40
41         @classmethod
42         def tearDownClass(cls):
43                 """called once, after all the tests in the class"""
44                 print("In tearDownClass()...")
```

Запустимо код

- python3 -m unittest -v test_module09
  - З'явиться помилка: ImportError: No module named 'mypackage'

```
In [1]: runfile('C:/Users/spuasson/Google Диск/[College]/[Основи програмування та алгоритмічні мови]/
[Презентації]/code_Pajankar/chapter03/test/test_module09.py', wdir='C:/Users/spuasson/Google Диск/[College]/
[Основи програмування та алгоритмічні мови]/[Презентації]/code_Pajankar/chapter03/test')
Traceback (most recent call last):

  File "<ipython-input-1-048562fd6e62>", line 1, in <module>
    runfile('C:/Users/spuasson/Google Диск/[College]/[Основи програмування та алгоритмічні мови]/[Презентації]/
code_Pajankar/chapter03/test/test_module09.py', wdir='C:/Users/spuasson/Google Диск/[College]/[Основи
програмування та алгоритмічні мови]/[Презентації]/code_Pajankar/chapter03/test')

  File "C:\ProgramData\Anaconda3\lib\site-packages\spyder\utils\site\sitecustomize.py", line 710, in runfile
    execfile(filename, namespace)

  File "C:\ProgramData\Anaconda3\lib\site-packages\spyder\utils\site\sitecustomize.py", line 101, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "C:/Users/spuasson/Google Диск/[College]/[Основи програмування та алгоритмічні мови]/[Презентації]/
code_Pajankar/chapter03/test/test_module09.py", line 1, in <module>
    from mypackage.mymathlib import *

ModuleNotFoundError: No module named 'mypackage'
```

- Модуль mypackage не видно з тестової директорії.
  - У цій папці його видно як mypackage, що знаходиться в chapter03.
  - python3 -m unittest -v test.test_module09

Результати запуску

```
In setUpModule()...
Creating object : mymathlib
In setUpClass()...
test_case01 (test.test_module09.TestClass10) ...
In setUp()...
In test_case01()
In tearDown()...
ok
test_case02 (test.test_module09.TestClass10) ...
In setUp()...
In test_case02()
In tearDown()...
ok
In tearDownClass()...
In tearDownModule()...
Destroying object : mymathlib

----------------------------------------------------------
Ran 2 tests in 0.004s

OK
```

- Відділяти тестовий код від тестованого коду – стандартна практика.

Відкриття тестів (Test Discovery)

- Процес розкриття та виконання всіх тестів у папці проекту та всіх субдиректоріях.
  - У unittest він автоматизований та може викликатись командою discover.
  - python3 -m unittest discover
- Частковий вивід після виконання команди в папці chapter02.

```
..
Running Test Method : test_case01
.
Running Test Method : test_case02
.
Running Test Method : test_case01
We're in custom made function : add
.
Running Test Method : test_case02
.
Running Test Method : test_case03
We're in custom made function : add
F
Running Test Method : test_case04
F
Classname : TestClass04
Running Test Method : test_case01
```

Частковий вивід команди python3 -m unittest discover -v

```
test_case01 (test.test_module01.TestClass01) ... ok
test_case02 (test.test_module01.TestClass01) ... ok
test_case01 (test.test_module02.TestClass02) ...
Running Test Method : test_case01
ok
test_case02 (test.test_module02.TestClass02) ...
Running Test Method : test_case02
ok
test_case01 (test.test_module03.TestClass03) ...
Running Test Method : test_case01
We're in custom made function : add
ok
test_case02 (test.test_module03.TestClass03) ...
Running Test Method : test_case02
ok
test_case03 (test.test_module03.TestClass03) ...
Running Test Method : test_case03
We're in custom made function : add
```

Угоди з написання коду для unittest

- Розкриття тестів автоматично знаходить та запускає всі тести в директорії проекту.
  - Для сумісності з розкриттям тестів всі файли з тестами повинні бути доступними для імпорту з top-level directory проекту модулями чи пакетами.
  - By default, the test discovery always starts from the current directory.
  - By default, test discovery always searches for test*.py patterns in the filenames.

Твердження в unittest

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)`

Твердження в unittest

- Методи id() та shortDescription() корисні для налагодження.
  - id() повертає назву методу
  - shortDescription() повертає опис методу.

```
import unittest

class TestClass11(unittest.TestCase):

        def test_case01(self):
                """This is a test method..."""
                print("\nIn test_case01()")
                print(self.id())
                print(self.shortDescription())
```

```
test_case01 (test_module10.TestClass11)
This is a test method... ...
In test_case01()
test_module10.TestClass11.test_case01
This is a test method...
ok

----------------------------------------------------------
Ran 1 test in 0.002s

OK
```

Method	Checks That
assertAlmostEqual(a, b)	round(a-b, 7) == 0
assertNotAlmostEqual(a, b)	round(a-b, 7) != 0
assertGreater(a, b)	a > b
assertGreaterEqual(a, b)	a >= b
assertLess(a, b)	a < b
assertLessEqual(a, b)	a <= b
assertRegexpMatches(s, r)	r.search(s)
assertNotRegexpMatches(s, r)	not r.search(s)
assertItemsEqual(a, b)	sorted(a) == sorted(b)
assertDictContainsSubset(a, b)	all the key/value pairs in a exist in b

Method	Used to Compare
assertMultiLineEqual(a, b)	strings
assertSequenceEqual(a, b)	sequences
assertListEqual(a, b)	lists
assertTupleEqual(a, b)	tuples
assertSetEqual(a, b)	sets or frozensets
assertDictEqual(a, b)	dicts

Провалення тесту

- Можливо, потрібно явно вказати, що тест повинен провалитись при виклику.
  - У unittest використовується метод fail()

```
test_case01 (test_module11.TestClass12)
This is a test method... ...
test_module11.TestClass12.test_case01
FAIL

======================================================================
FAIL: test_case01 (test_module11.TestClass12)
This is a test method...
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/pi/book/code/chapter03/test/test_module11.py", line 9, in
test_case01
    self.fail()
AssertionError: None

----------------------------------------------------------------------
Ran 1 test in 0.004s

FAILED (failures=1)

Skipping tests
```

```
import unittest

class TestClass12(unittest.TestCase):

        def test_case01(self):
                """This is a test method..."""
                print(self.id())
                self.fail()
```

```
import sys
import unittest

class TestClass13(unittest.TestCase):

    @unittest.skip("demonstrating unconditional skipping")
    def test_case01(self):
        self.fail("FATAL")

    @unittest.skipUnless(sys.platform.startswith("win"), "requires Windows")
    def test_case02(self):
        # Windows specific testing code
        pass

    @unittest.skipUnless(sys.platform.startswith("linux"), "requires Linux")
    def test_case03(self):
        # Linux specific testing code
        pass
```

unittest постачає механізм пропуску тестів

Використовуються декоратори:

- ***unittest.skip(reason)***: безумовно пропускає декорований тест. Параметр reason повинен описувати, чому тест пропущено.
- ***unittest.skipIf(condition, reason)***: пропускає декорований тест, якщо значення condition = True.
- ***unittest.skipUnless(condition, reason)***: пропускає декорований тест, поки condition = True.
- ***unittest.expectedFailure()***: позначає тест, як очікуваний для падіння. Якщо тест провалюється, він не враховується в провалених тестах.

- Windows

```
test_case01 (test_module12.TestClass13) ... skipped 'demonstrating
unconditional skipping'
test_case02 (test_module12.TestClass13) ... ok
test_case03 (test_module12.TestClass13) ... skipped
'requires Linux'

----------------------------------------------------------------------
Ran 3 tests in 0.003s

OK (skipped=2)
```

- Linux

```
test_case01 (test_module12.TestClass13) ... skipped 'demonstrating
unconditional skipping'
test_case02 (test_module12.TestClass13) ... skipped 'requires Windows'
test_case03 (test_module12.TestClass13) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.003s

OK (skipped=2)
```

Винятки в тестовому випадку

- Коли викидається виняток у тестовому випадку, цей випадок падає.

```
import unittest

class TestClass14(unittest.TestCase):
    def test_case01(self):
        raise Exception
```

```
test_case01 (test_module13.TestClass14) ... ERROR

======================================================================
ERROR: test_case01 (test_module13.TestClass14)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/pi/book/code/chapter03/test/test_module13.py", line 6, in
  test_case01
    raise Exception
Exception

----------------------------------------------------------------------
Ran 1 test in 0.004s

FAILED (errors=1)
```

Метод assertRaises()

- Використовується для перевірки, чи викидає блок коду виняток, згаданий у методі.
  - Якщо код викидає виняток, тест пройдено; інакше - провалено.

```
import unittest

class Calculator:

        def add1(self, x, y):
                return x + y

        def add2(self, x, y):
                number_types = (int, float, complex)
                if isinstance(x, number_types) and isinstance(y, number_
                types):
                        return x + y
                else:
                        raise ValueError

calc = 0
```

```
class TestClass16(unittest.TestCase):

        @classmethod
        def setUpClass(cls):
                global calc
                calc = Calculator()

        def setUp(self):
                print("\nIn setUp()...")

        def test_case01(self):
                self.assertEqual(calc.add1(2, 2), 4)

        def test_case02(self):
                self.assertEqual(calc.add2(2, 2), 4)

        def test_case03(self):
                self.assertRaises(ValueError, calc.add1, 2, 'two')

        def test_case04(self):
                self.assertRaises(ValueError, calc.add2, 2, 'two')

        def tearDown(self):
                print("\nIn tearDown()...")

        @classmethod
        def tearDownClass(cls):
                global calc
                del calc
```

```
test_case01 (test_module14.TestClass16) ...
In setUp()...

In tearDown()...
ok
test_case02 (test_module14.TestClass16) ...
In setUp()...

In tearDown()...
ok
test_case03 (test_module14.TestClass16) ...
In setUp()...

In tearDown()...
ERROR
test_case04 (test_module14.TestClass16) ...
In setUp()...

In tearDown()...
ok
```

Вивід

```
======================================================================
ERROR: test_case03 (test_module14.TestClass16)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/pi/book/code/chapter03/test/test_module14.py", line 37, in
  test_case03
    self.assertRaises(ValueError, calc.add1, 2, 'two')
  File "/usr/lib/python3.4/unittest/case.py", line 704, in assertRaises
    return context.handle('assertRaises', callableObj, args, kwargs)
  File "/usr/lib/python3.4/unittest/case.py", line 162, in handle
    callable_obj(*args, **kwargs)
  File "/home/pi/book/code/chapter03/test/test_module14.py", line 7, in add1
    return x + y

TypeError: unsupported operand type(s) for +: 'int' and 'str'

----------------------------------------------------------------------
Ran 4 tests in 0.030s

FAILED (errors=1)
```

ДЯКУЮ ЗА УВАГУ!

Наступне питання: