Unit Testing in Python

COMP3613: Software Engineering II

Unit Testing

- Frameworks exists for many programming languages
- Will use Python
  - But principles generalise
  - Why? Easy to setup, use, and understand
- Most unit testing guided by the use of *assert* statements of some form

Python Unit Testing

- Many different unit testing frameworks in Python
- Most popular:
  - unittest
  - pytest
  - nose
- Others exist, but principles generalise
- Will focus on unittest
  - Comes as standard in Python
- If using pytest in practice, I recommend looking into QuickCheck
  - https://pypi.org/project/pytest-quickcheck/

unittest

- Test runner for unittest runs all files named using the format test_*.py
  - where * is any combination of non-whitespace characters
    - Regular Expressions! :D
  - Examples: test_complex.py, test_rational.py, test_quaternion.py, etc…
- Test are run using python -m unittest <filename> on command line
  - Example python -m unittest test_rational.py

unittest

- unittest contains a class named TestCase
- Unit Tests are encapsulated in objects that inherit from TestCase
- Each method in such classes contain tests to be executed
- Each method should use assert methods inherited from TestCase; not the in-built assert statement!

```python
from unittest import TestCase

class ArithTests(TestCase):

    def test_eq_1(self):
        x = 1
        y = 1
        self.assertEqual(x, y)
```

unittest

```
[Inzamams-MBP:basic inzamamrahaman$ python -m unittest test_arith.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

unittest

- Let’s make a test that will fail!

unittest

```
Inzamams-MBP:basic inzamamrahaman$ python -m unittest test_arith.py
.F
======================================================================
FAIL: test_eq_2 (test_arith.ArithTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/inzamamrahaman/Desktop/Teaching2019/sites/COMP3613/tutorials and auxiilary
material/unit_testing/basic/test_arith.py", line 14, in test_eq_2
    self.assertEqual(x, y)
AssertionError: 1 != 2

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

unittest

test that failed

```
[Inzamams-MBP:basic inzamamrahaman$ python -m unittest test_arith.py
.F
======================================================================
FAIL: test_eq_2 (test_arith.ArithTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/inzamamrahaman/Desktop/Teaching2019/sites/COMP3613/tutorials and auxiilary
material/unit_testing/basic/test_arith.py", line 14, in test_eq_2
    self.assertEqual(x, y)
AssertionError: 1 != 2

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

Number of tests that failed

unittest

- Checking whether or not two things are equal is fine and dandy but we want to check other things
- unittest also provides:
  - assertIn(x, y) - checks if x is in y
  - assertNotIn(x, y) - checks if x is not in y
  - assertTrue(x) - checks if bool(x) is True
  - assertFalse(x) - checks if bool(x) is False
  - assertNotEqual(x, y) - checks if x != y
  - assertIsNone(x) - checks if x is None
  - etc … (documentation: https://docs.python.org/2/library/unittest.html#assert-methods)

Excercises

1. Consider the Complex class provided, write a test suite using unittest to test the methods provided in said class. Use a white-box approach

2. Notice that when running your tests, some of the previously written tests fail. Fix the code in complex.py to remedy this