

SENG 265

unittest introduction

the idea

```
def assertTrue(condition):  
    if not condition:  
        raise AssertionError
```

```
first = None  
"""operations on first which are supposed to assign  
    non-None values to first..."""  
assertTrue( first != None )
```

the idea can be generalized

```
def assertEquals (a, b):  
    assertTrue(a == b)
```

```
def assertFalse (condition):  
    if condition:  
        raise AssertionError
```

motivating example (file: largest.py)

```
import sys

class Largest:

    def largest(list):
        max = sys.maxint
        for i in range(0 .. len(list)-1):
            if (list[i] > max):
                max = list[i]

        return max
```

some simple things should be true...

```
Largest.largest([7, 8, 9]) should be 9
```

```
Largest.largest([8, 9, 7]) should be 9
```

```
Largest.largest([9, 7, 8]) should be 9
```

```
Largest.largest([7, 9, 8, 9]) should be 9
```

```
Largest.largest([1]) should be 1
```

```
Largest.largest([-9, -8, -7]) should be
```

insight: can write simple tests as asserts

```
import unittest
import Largest from largest

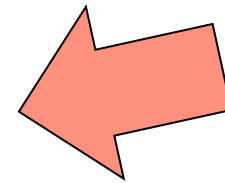
class TestLargest(unittest.TestCase):

    def setUp(self):
        """ set up data used in the tests. set up is
        called before each test function execution. """
        self.l = Largest()

    def testOrder(self):
        title = "Testing order"
        self.assertEqual(9, self.l.largest([8, 9, 7]))

def suite():
    suite = unittest.TestSuite()
    suite.addTest(unittest.makeSuite(TestLargest))
    return suite

if __name__ == "__main__":
    unittest.TextTestRunner(verbosity=2).run(suite())
```



what would have been output?

```
$ python unittest_largest.py
```

```
testOrder (__main__.TestLargest) ... FAIL
=====
FAIL: testOrder (__main__.TestLargest)
-----
Traceback (most recent call last):
  File "./unittest_largest.py", line 17, in testOrder
    self.assertEqual(9, self.l.largest([8, 9, 7]))
AssertionError: 9 != 9223372036854775807
-----
Ran 1 test in 0.000s

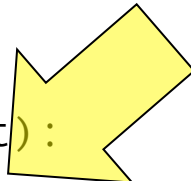
FAILED (failures=1)
```

oops!

```
import sys

class Largest:

    def largest(self, list):
        max = sys.maxint
        for i in range(0, len(list)-1):
            if (list[i] > max):
                max = list[i]
        return max
```



fix it

```
import sys

class Largest:

    def largest(self, list):
        max = 0
        for i in range(0, len(list)-1):
            if (list[i] > max):
                max = list[i]
        return max
```

for completeness...

```
import unittest
from largest import Largest

class TestLargest(unittest.TestCase):

    def setUp(self):
        """ set up data used in the tests. set up is
        called before each test function execution. """
        self.l = Largest()

    def testOrder(self):
        title = "Testing order"
        self.assertEqual(9, self.l.largest([9, 8, 7]))
        self.assertEqual(9, self.l.largest([8, 9, 7]))
        self.assertEqual(9, self.l.largest([7, 8, 9]))

# Etc. etc.
```

oops! (ver 2)

```
$ ./unittest_largest.py
testOrder (__main__.TestLargest) ... FAIL

=====
FAIL: testOrder (__main__.TestLargest)
-----
Traceback (most recent call last):
  File "./unittest_smaller.py", line 17, in testOrder
    self.assertEqual(9, self.l.largest([7, 8, 9]))
AssertionError: 9 != 8

-----
Ran 1 test in 0.000s

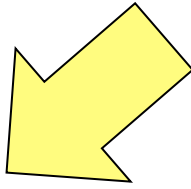
FAILED (failures=1)
```

oops! (ver 2)

```
import sys

class Largest:

    def largest(self, list):
        max = 0
        for i in range(0, len(list)-1):
            if (list[i] > max):
                max = list[i]
        return max
```



fix (ver 2)

```
import sys

class Largest:

    def largest(self, list):
        max = 0
        for i in range(0, len(list)):
            if (list[i] > max):
                max = list[i]
        return max
```

clean!

```
$ ./unittest_largest.py  
testOrder (__main__.TestLargest) ... ok
```

```
-----  
Ran 1 test in 0.000s
```

```
OK
```

add more of the tests

```
import unittest
from largest import Largest

class TestLargest(unittest.TestCase):

    def setUp(self):
        """ set up data used in the tests. set up is
        called before each test function execution. """
        self.l = Largest()

    def testOrder(self):
        title = "Testing order"
        self.assertEqual(9, self.l.largest([9, 8, 7]))
        self.assertEqual(9, self.l.largest([8, 9, 7]))
        self.assertEqual(9, self.l.largest([7, 8, 9]))

    def testDups(self):
        self.assertEqual(9, self.l.largest([9, 7, 9, 8]))

    def testSingleton(self):
        self.assertEqual(1, self.l.largest([1]))

# etc.
```

clean again!

```
$ ./unittest_largest.py  
testDups (__main__.TestLargest) ... ok  
testOrder (__main__.TestLargest) ... ok  
testSingleton (__main__.TestLargest) ... ok
```

```
-----  
Ran 3 tests in 0.000s
```

```
OK
```


and more...

```
def testNegative(self):  
    self.assertEqual(-7, self.l.largest([-9, -8, -7]))
```

```
$ ./unittest_largest.py  
testDups (__main__.TestLargest) ... ok  
testNegative (__main__.TestLargest) ... FAIL  
testOrder (__main__.TestLargest) ... ok  
testSingleton (__main__.TestLargest) ... ok
```

```
=====
```

```
FAIL: testNegative (__main__.TestLargest)
```

```
-----
```

```
Traceback (most recent call last):
```

```
  File "./unittest_largest.py", line 26, in testNegative  
    self.assertEqual(-7, self.l.largest([-9, -8, -7]))
```

```
AssertionError: -7 != 0
```

```
-----
```

```
Ran 4 tests in 0.000s
```

```
FAILED (failures=1)
```

testing empty

```
def testEmpty(self):
    try:
        self.l.largest([])
    except ValueError:
        pass
    else:
        self.fail("Should have thrown a ValueError exception")
```

```
=====
FAIL: testEmpty (__main__.TestLargest)
```

```
-----
Traceback (most recent call last):
```

```
  File "./unittest_largest.py", line 34, in testEmpty
```

```
    self.fail("Should have thrown a ValueError exception")
```

```
AssertionError: Should have thrown a ValueError exception
```

asserts

```
assertEquals(first, second [,msg])  
assertGreater(first, second [,msg])  
assertLess(first, second [,msg])  
failIfEqual(first, second [,msg])  
assertIsNone(expr [,msg])  
assertIsNotNone(expr [,msg])  
<and many, many others>
```

test suites

- Without "suite", using Python reflection
- May wish to bundle tests together into distinct sets
 - Each set is a "suite"
 - Can combine long running tests into one suite
 - Leave other tests in a second suite
- We will see this construct a little later, but for now be aware it exists

per-test setup and tear-down

- Permits each test to run independently of others
- May also involve database connection setup, network setup, (and teardown, etc.)
- Again, we will see these later.
- Important thing is to learn how to come up with the right tests