Testing Code with Unit Tests

Obriously, this is somehow what Zybooks does to see if your program is correct!!

But, it also has tremendous uses beyond auto-grading systems.

Big Thing → Continuous Integration (CI)

You have a huge project that is being managed on GitHub, with many people contributing, all the time.

How to check that someone did not "break something" with their latest update?

→ Design a set of tests to make sure!!

Unit tests are super important in OO Programming ... we want to make sure that changing something in some class does not break things elsewhere (for some user that is inheriting our class).

Simple Example:

```
import unittest

class Circle:

    def __init__(self, radius):
        self.radius = radius

    def compute_area(self):
        return 3.14159265 *
               self.radius **2

    def compute_circ(self):
        return 2.0*3.14159265 *
               self.radius
```

```
class TestCircle(unittest.TestCase):
    def test_area(self):
        c = Circle(0)
        self.assertEqual(c.compute_area(),
                         0.0)
        c = Circle(5)
        self.assertEqual(c.compute_area(),
                         78.5398)

    def test_stupid(self):
        c = Circle(5)
        self.assertLess(c.comput_area(),
                        0.0)

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

Test → Area = 0.0 (Expect Pass)

Test → Area = 78.5398 (Expect Pass)

Test → Area < 0.0 (Expect Fail → that's why it's stupid ☺)

Notes :

① Test class inherits from unittest.TestCase !!

② Tests are defined as functions of the Test class.

→ must begin with "test_" !!

③ unittest.main() in the main program invokes the test procedure.

See circle_unittest.py for more details !!

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`
assertAlmostEqual(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