



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

EN.540.635 Software Carpentry

Mini Lecture Testing

- “Tests” that are written in order to check the smallest part of your code: a single function
- Benefits
 - Helps find bugs early!
 - Especially useful with multiple developers — ensure a change doesn’t break someone else’s code
 - Can be automated
- Disadvantages
 - Takes time to write
 - False sense of confidence — only as good as the tests themselves!
- Many frameworks to help keep unit tests organized

Unit Testing - Example

```
def is_prime(p):  
    """  
    Test if a given integer p is prime or not.  
  
    **Parameters**  
    p: *int*  
        Some number to test.  
  
    **Returns**  
    chk: *bool*  
        Whether p is prime (True) or not (False).  
    """  
    small_primes = [1, 2, 3, 5, 7]  
    if p in small_primes: return True  
  
    for i in range(2, int((p**0.5))+1):  
        if p%i==0: return False  
  
    return True
```

```
def Test1():  
    assert is_prime(2) == True  
  
def Test2():  
    assert is_prime(1) == False  
  
def Test3():  
    assert is_prime(22341352) == False  
  
if __name__ == "__main__":  
    Test1()  
    Test2()  
    Test3()
```

```
Tasia@Anastasias-MacBook-Air 8A_Unit_Testing % python3 UnitTesting.py  
Tasia@Anastasias-MacBook-Air 8A_Unit_Testing %
```

- **unittest** (built-in)
 - Requires tests to be stored as class
 - Uses special assertion statements available in `unittest.TestCase` class
- nose2
- pytest

```
'''
Created October 25, 2022
'''
import unittest

def is_prime(p): ...

class TestIsPrime(unittest.TestCase):
    '''
    Testing suite for is_prime function.
    '''

    def test_1(self):
        self.assertFalse(is_prime(1))

    def test_2(self):
        self.assertTrue(is_prime(2))

    def test_3(self):
        self.assertEqual(is_prime(22341352), False)

    def test_bad_type(self):
        data = "oops"
        with self.assertRaises(TypeError):
            result = is_prime(data)

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

must begin with "test_"

```
Tasia@Anastasias-MacBook-Air 8A_Unit_Testing % python -m unittest UnitTesting_2
....
-----
Ran 4 tests in 0.001s

OK
Tasia@Anastasias-MacBook-Air 8A_Unit_Testing %
```

```
Tasia@Anastasias-MacBook-Air 8A_Unit_Testing % python -m unittest UnitTesting_2
..F.
=====
FAIL: test_3 (UnitTesting_2.TestIsPrime)
-----
Traceback (most recent call last):
  File "UnitTesting_2.py", line 45, in test_3
    self.assertEqual(is_prime(22341352), True)
AssertionError: False != True
-----
Ran 4 tests in 0.001s

FAILED (failures=1)
```

- Functions that perform multiple tasks (such as potentially changing a state) may be hard to unit test!
- Single Responsibility Principle: Each unit test “should be responsible for one, and only one, actor”
 - Single call to testing function
 - Single **logical** assertion (checking multiple aspects of a single object is okay)
 - Unit test does not depend on state
 - Don't duplicate tests!

- “Tests” to see if different components are working with each other
- Databases, other files, jpg images can be stored out in folder
- Most unit tests & integration tests are separated

Integration Testing - Example

```
project/
├── my_app/
│   └── __init__.py
└── tests/
    ├── unit/
    │   ├── __init__.py
    │   └── test_sum.py
    └── integration/
        ├── fixtures/
        │   ├── test_basic.json
        │   └── test_complex.json
        ├── __init__.py
        └── test_integration.py
```

```
class TestBasic(unittest.TestCase):
    def setUp(self):
        # Load test data
        self.app = App(database='fixtures/test_basic.json')

    def test_customer_count(self):
        self.assertEqual(len(self.app.customers), 100)

    def test_existence_of_customer(self):
        customer = self.app.get_customer(id=10)
        self.assertEqual(customer.name, "Org XYZ")
        self.assertEqual(customer.address, "10 Red Road, Reading")

class TestComplexData(unittest.TestCase):
    def setUp(self):
        # load test data
        self.app = App(database='fixtures/test_complex.json')

    def test_customer_count(self):
        self.assertEqual(len(self.app.customers), 10000)

    def test_existence_of_customer(self):
        customer = self.app.get_customer(id=9999)
        self.assertEqual(customer.name, u"バナナ")
        self.assertEqual(customer.address, "10 Red Road, Akihabara, Tokyo")

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

- <https://realpython.com/python-testing/>
- <https://docs.python.org/3/library/unittest.html#>