Unit tests

Ref: youtube.com/watch?v=6tNS--WetLI

(Corey Schafer - Python Tutorial)

say you have a calc.py (calculator)

with 4 functions

add(x,y)

subtract(x,y)

multiply(x,y)

divide(x,y)

test_calc.py ← where you will write your tests

```
import unittest
import calc                      # your module you want to test

class TestCalc(unittest.TestCase):          # inheriting from TestCase module

    def test_add(self):                     # what you are testing
        result = calc.add(10, 5)
        self.assertEqual(result, 15)
```

all methods need to start with test_ otherwise won't run

can add more assert statements

still (test.

cannot just run

```
python test_calc.py
```

won't return anything

```
python -m unittest test_calc.py
```

```
.
----------------------
Ran 1 test

OK
```

To directly use test_calc.py, --

can add

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

then can just do

```
python test_calc.py
```

If OK

```
.
-----------
Ran 1 test

OK
```

If not

```
F
=============
FAIL: test_add
```

OK

AssertionError ...

Say your calc.py

→ raises a ValueError for input (0, 0)

To check in your unit test that your function raises the correct Error

function

```
self.assertRaises(ValueError, calc.divide, (0, 0))
```

error you should be raising

input

or

context manager

```
with self.assertRaises(ValueError):
    calc.divide(10, 0)
```

what if you want to create some set up for your test cases that you want to reuse while testing a lot of methods? (to avoid creating the set up for the test case in every method individually)

will run its code before every single test

```
def setUp(self):
    self.example = ----
```

def tearDown(self):

will run its code after every single test

create attributes for the class

tests may not run in order !!

If you want to run some code before starting tests and run some code after all tests ...

@classmethod

def setUpClass(cls):

def tearDownClass(cls):

If your code depends on pulling some data from an external website and that website is down, then you don't want to record that as failure of your code ....

"mocking"

from unittest.mock import patch

(allows you to "mock" an object during a test and after test is run, object is automatically recreated)

with patch(`obj_you_want_to_mock`) as mocked_var:

Set mocked_var's value

mocked_var = ...