1. What is Unit Testing?

Unit testing is a method for testing software that looks at the smallest testable pieces of code, called units, which are tested for correct operation. By doing unit testing, we can verify that each part of the code, including helper functions that may not be exposed to the user, works correctly and as intended.

```python
# calc.py
def add(x, y):
    """Add Function"""
    return x + y

def subtract(x, y):
    """Subtract Function"""
    return x - y

def multiply(x, y):
    """Multiply Function"""
    return x * y

def divide(x, y):
    """Divide Funciton"""
    if y == 0:
        raise ValueError("Cannot divide by Zero")
    return x / y

# Check the test result of add function
print(add(2, 3))
# result : 5
```

But, testing our code this way isn't easy to automate and it's also hard to maintain.

The **unittest** unit testing framework supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

2. How to use unittest module?

Step 1 : Importing library and module we want to test

```python
# test_calc.py
import unittest
import calc
```

2.2 Delcare Test Class inheritance of unittest

The basic building block of unit testing are test cases. The test cases are represented by unittest.TestCase instances. In order to test something, we use one of the assert*() method provided by the TestCase base class.

If the test fails, an exception will be reaised with an explanatory message, and unittest will identify the test case as a failure.

assertEqual(first, second)

Test that first and second are equal. If the value do not compare equal, the test will fail.

```python
class TestCalc(unittest.TestCase):

    # The name of method to test must start with test_
    def test_add(self):
        result = calc.add(10, 5)
        self.assertEqual(result, 15)

    def test_subtract(self):
        self.assertEqual(calc.subtract(10, 5), 5)

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

2.3 Testing Exceptions

asertRaises(exception, callable, *args, **kwds)

Test that an exception is raised when callable is called with any positional or keyword arguments that are also passed to assertRaises(). The test passes if exception is raised, is an error if another exception is raised, or fails if no exception is raised.

```python
class TestCalc(unittest.TestCase):

    # The name of method to test must start with test_
    def test_add(self):
        result = calc.add(10, 5)
        self.assertEqual(result, 15)

    def test_subtract(self):
        self.assertEqual(calc.subtract(10, 5), 5)

    def test_divide(self):
        self.assertRaises(ValueError, calc.divide, 10, 0)

        # Same code as above
        with self.assertRaise(ValueError):
            calc.divide(10, 0)

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

2.4 Execute unittest in Command Line

If we want to do unittest on command line, we command python -m unittest file.py on command line.

It is recommended that we use TestCase implementations to group tests together according to the features they test. calling unittest.main() will do the right thing and collect all the module's test cases for us and execute them.

The way of using __name__ == '__main__' saying that if we run this module directly then run the code within conditional and that code within conditional is unittest.main().

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

.
---------------------------------------------
Ran 2 test in 0.000s

OK

$ python test_calc.py

.
---------------------------------------------
Ran 2 test in 0.000s

OK
```

3. Organizing test code

Tests can be numerous, and their set-up can be repetitive. Luckily, we can factor out set-up code by implementing a method called setUp(), which the testing framework will automatically call for every single test we run:

We can provide a tearDown() method that tides up after the test method has been run. If setUp() succeeded, tearDown() will be run whether the test method suceeded or not.

```python
from employee import Employee
import unittest

class TestEmplyee(unittest.TestCase):

    @classmethod
    def setUp(cls):
        print('setUp')
        self.emp_1 = Employee('Corey', 'Schafer', 50000)
        self.emp_2 = Employee('Sue', 'Smith', 60000)

    @classmethod
    def tearDown(cls):
        print('tearDown')

    def test_email(self):
        print('test_email')
        self.assertEqual(self.emp_1.email, 'Corey.Schafer@email.com')
        self.assertEqual(self.emp_2.email, 'Sue.Smith@email.com')

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

```
CLI
$ python test_calc.py

setUp
test_email
tearDown

.
---------------------------------------------
Ran 1 test in 0.000s

OK
```