Python TUTORIAL

Organizing Code & Running unittest

Summary: in this tutorial, you'll learn how to organize the test code and how to use the various commands to run unit tests.

Organizing code

If you have a few modules (https://www.pythontutorial.net/python-basics/python-module/) , you can create test modules and place them within the same directory.

In practice, you may have many modules organized into packages (https://www.pythontutorial.net/python-basics/python-packages/) . Therefore, it's important to keep the development code and test code more organized.

It's a good practice to keep the development code and the test code in separate directories. And you should place the test code in a directory called `test` to make it obvious.

For demonstration purposes, we'll create a sample project with the following structure:

```
D:\python-unit-testing
├── shapes
|   ├── circle.py
|   ├── shape.py
|   └── square.py
└── test
   ├── test_circle.py
   ├── test_square.py
   └── __init__.py
```

First, create the `shapes` and `test` directories in the project folder (`python-unit-testing`).

Second, create three modules `shape.py` , `circle.py` , and `square.py` modules and place them in the `shapes` directory.

shape.py

```
from abc import ABC, abstractmethod


class Shape(ABC):
    @abstractmethod
    def area() -> float:
        pass
```

The `Shape` class is an abstract class (https://www.pythontutorial.net/python-oop/python-abstract-class/) that has the `area()` method. It is the base class of the `Circle` and `Square` class.

circle.py

```
import math
from .shape import Shape


class Circle(Shape):
    def __init__(self, radius: float) -> None:
        if radius < 0:
            raise ValueError('The radius cannot be negative')

        self._radius = radius
    def area(self) -> float:
        return math.pi * math.pow(self._radius, 2)
```

The `Circle` class also inherits from the `Shape` class. It implements the `area()` method that returns the area of the circle.

square.py

```
import math
from .shape import Shape


class Square(Shape):
    def __init__(self, length: float) -> None:
        if length < 0:
            raise ValueError('The length cannot be negative')
        self._length = length

    def area(self) -> float:
        return math.pow(self._length, 2)
```

Like the `Circle` class, the `Square` class has the `area()` method that returns the area of the square.

Third, create the `test_circle.py` and `test_square.py` test modules and place them in the `test` folder:

test_circle.py

```
import unittest
import math

from shapes.circle import Circle
from shapes.shape import Shape
```

```
class TestCircle(unittest.TestCase):
    def test_circle_instance_of_shape(self):
        circle = Circle(10)
        self.assertIsInstance(circle, Shape)

    def test_create_circle_negative_radius(self):
        with self.assertRaises(ValueError):
            circle = Circle(-1)

    def test_area(self):
        circle = Circle(2.5)
        self.assertAlmostEqual(circle.area(), math.pi * 2.5*2.5)
```

The `test_circle` module uses the `Circle` and `Shape` from the `circle` and `shape` test modules in the `shapes` package.

test_square.py

```
import unittest

from shapes.square import Square
from shapes.shape import Shape


class TestSquare(unittest.TestCase):

    def test_create_square_negative_length(self):
        with self.assertRaises(ValueError):
            square = Square(-1)

    def test_square_instance_of_shape(self):
        square = Square(10)
        self.assertIsInstance(square, Shape)

    def test_area(self):
        square = Square(10)
        area = square.area()
        self.assertEqual(area, 100)
```

The `test_square` module uses the `Square` and `Shape` classes from the `square` and `shape` modules in the `shapes` package.

It's important to create an `__init__.py` file and place it in the `test` folder. Otherwise, commands in the following section won't work as expected.

Running unit tests

The unittest (https://www.pythontutorial.net/python-unit-testing/python-unittest/) module provides you with many ways to run unit tests.

1) Running all tests

To run all tests in the `test` directory, you execute the following command from the project directory folder (`python-unit-testing`):

```
python -m unittest discover -v
```

The `discover` is a subcommand that finds all the tests in the project.

Output:

```
test_area (test_circle.TestCircle) ... ok
test_circle_instance_of_shape (test_circle.TestCircle) ... ok
test_create_circle_negative_radius (test_circle.TestCircle) ... ok
test_area (test_square.TestSquare) ... ok
test_create_square_negative_length (test_square.TestSquare) ... ok
test_square_instance_of_shape (test_square.TestSquare) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.002s
```

```
OK
```

2) Running a single test module

To run a single test module, you use the following command:

```
python -m unittest test_package.test_module -v
```

For example, the following execute all tests in the `test_circle` module of the `test` package:

```
python -m unittest test.test_circle -v
```

Output:

```
test_area (test.test_circle.TestCircle) ... ok
test_circle_instance_of_shape (test.test_circle.TestCircle) ... ok
test_create_circle_negative_radius (test.test_circle.TestCircle) ... ok

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

OK
```

3) Running a single test class

A test module may have multiple classes. To run a single test class in a test module, you use the following command:

```
python -m unittest test_package.test_module.TestClass -v
```

For example, the following command tests the `TestSquare` class from the `test_square` module of the test package:

```
python -m unittest test.test_circle.TestCircle -v
```

Output:

```
test_area (test.test_square.TestSquare) ... ok
test_create_square_negative_length (test.test_square.TestSquare) ... ok
test_square_instance_of_shape (test.test_square.TestSquare) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

4) Running a single test method

To run a single test method of a test class, you use the following command:

```
python -m unittest test_package.test_module.TestClass.test_method -v
```

For example, the following command tests the `test_area()` method of the `TestCircle` class:

```
python -m unittest test.test_circle.TestCircle.test_area -v
```

Output:

```
test_area (test.test_circle.TestCircle) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Summary

- Place the development code and test code in separate directories. It’s a good practice to store the test code in the test directory.
- Use the command `python -m unittest discover -v` to discover and execute all the tests.
- Use the command `python -m unittest test_package.test_module -v` to run a single test module.
- Use the command `python -m unittest test_package.test_module.TestClass -v` to run a single test class.
- Use the command `python -m unittest test_package.test_module.TestClass.test_method` -v to run a single test method.