Python Test Fixtures

Summary: in this tutorial, you'll learn about Python test fixtures including `setUp()` and `tearDown()` methods.

Introduction to the Python Test fixtures

By definition, a test fixture is a function (https://www.pythontutorial.net/python-basics/python-functions/) or method (https://www.pythontutorial.net/python-oop/python-methods/) that runs before and after a block of test code executes. In other words, it is a step carried out before or after a test.

Module-level fixtures

Suppose you have a test module called `test_my_module.py`. In the `test_my_module.py`, the `setUpModule()` and `tearDownModule()` functions are the module-level fixtures.

- The `setUpModule()` function runs before all test methods in the test module.
- The `tearDownModule()` function runs after all methods in the test module.

See the following example:

```
import unittest
```

```
def setUpModule():
    print('Running setUpModule')


def tearDownModule():
    print('Running tearDownModule')


class TestMyModule(unittest.TestCase):
    def test_case_1(self):
        self.assertEqual(5+5, 10)

    def test_case_2(self):
        self.assertEqual(1+1, 2)
```

If you run the test:

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

Output:

```
Running setUpModule
test_case_1 (test_my_module.TestMyModule) ... ok
test_case_2 (test_my_module.TestMyModule) ... ok
Running tearDownModule

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

In this example, the `setUpModule()` function runs before all the test methods and the `tearDownModule()` function runs after all the test methods.

Class-level fixtures

The `setUpClass()` and `tearDownClass()` are class-level fixtures:

- The `setUpClass()` runs before all test methods of a class
- The `tearDownClass()` runs after all test methods of a class.

For example:

```
import unittest


def setUpModule():
    print('Running setUpModule')


def tearDownModule():
    print('Running tearDownModule')


class TestMyModule(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        print('Running setUpClass')

    @classmethod
    def tearDownClass(cls):
        print('Running tearDownClass')

    def test_case_1(self):
        self.assertEqual(5+5, 10)

    def test_case_2(self):
        self.assertEqual(1+1, 2)
```

In this example, we added the class methods: `setUpClass()` and `tearDownClass()` to the `TestMyModule` class.

If you run the test, you'll see the following output:

```
Running setUpModule
Running setUpClass
test_case_1 (test_my_module.TestMyModule) ... ok
test_case_2 (test_my_module.TestMyModule) ... ok
Running tearDownClass
Running tearDownModule

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Method-level fixtures

The `setUp()` and `tearDown()` are method-level fixtures:

- The `setUp()` runs before every test method in the test class.
- The `tearDown()` runs after every test method in the test class.

For example:

```
import unittest


def setUpModule():
    print('Running setUpModule')


def tearDownModule():
    print('Running tearDownModule')
```

```
class TestMyModule(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        print('Running setUpClass')

    @classmethod
    def tearDownClass(cls):
        print('Running tearDownClass')

    def setUp(self):
        print('')
        print('Running setUp')

    def tearDown(self):
        print('Running tearDown')

    def test_case_1(self):
        print('Running test_case_1')
        self.assertEqual(5+5, 10)

    def test_case_2(self):
        print('Running test_case_2')
        self.assertEqual(1+1, 2)
```

The following shows the test result:

```
Running setUpModule
Running setUpClass
test_case_1 (test_my_module.TestMyModule) ...
Running setUp
Running test_case_1
Running tearDown
ok
test_case_2 (test_my_module.TestMyModule) ...
Running setUp
Running test_case_2
```

```
Running tearDown
ok
Running tearDownClass
Running tearDownModule

----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
```

In this example, the `setUp()` and `tearDown()` executes before and after each test method including `test_case_1()` and `test_case_2()`.

Python test fixtures example

First, define classes called `BankAccount` and `InsufficientFund` classes in the `bank_account.py` module:

```
class InsufficientFund(Exception):
    pass


class BankAccount:
    def __init__(self, balance: float) -> None:
        if balance < 0:
            raise ValueError('balance cannot be negative')
        self._balance = balance

    @property
    def balance(self) -> float:
        return self._balance

    def deposit(self, amount: float) -> None:
        if amount <= 0:
```

```
            raise ValueError('The amount must be positive')

        self._balance += amount

    def withdraw(self, amount: float) -> None:
        if amount <= 0:
            raise ValueError('The withdrawal amount must be more than 0')

        if amount > self._balance:
            raise InsufficientFund('Insufficient ammount for withdrawal')

        self._balance -= amount
```

Second, define the `TestBankAccount` class in the `test_bank_account.py` module:

```
import unittest

from bank_account import BankAccount


class TestBankAccount(unittest.TestCase):
    def test_deposit(self):
        self.bank_account = BankAccount(100)
        self.bank_account.deposit(100)
        self.assertEqual(self.bank_account.balance, 200)

    def test_withdraw(self):
        self.bank_account = BankAccount(100)
        self.bank_account.withdraw(50)
        self.assertEqual(self.bank_account.balance, 50)
```

The `TestBankAccount` class has two test methods:

- `test_ deposit()` – test the `deposit()` method of the bank account.
- `test_ withdraw()` – test the `withdraw()` method of the bank account.

Both methods create a new instance of the `BankAccount`. It's redundant.

To avoid redundancy, you can create an instance of the BankAccount class in `setUp()` method and use it in all the test methods:

```
import unittest

from bank_account import BankAccount


class TestBankAccount(unittest.TestCase):
    def setUp(self) -> None:
        self.bank_account = BankAccount(100)

    def test_deposit(self):
        self.bank_account.deposit(100)
        self.assertEqual(self.bank_account.balance, 200)

    def test_withdraw(self):
        self.bank_account.withdraw(50)
        self.assertEqual(self.bank_account.balance, 50)
```

In the `setUp()` method:

- First, create an instance of the `BankAccount` class and assign it to the instance variable `self.bank_account`.
- Then, use `self.bank_account` instance in both `test_deposit()` and `test_withdraw()` methods.

When running test methods `test_deposit()` and `test_withdraw()`, the `setUp()` runs before each test method.

For `test_deposit()` method:

```
setUp()
test_deposit()
```

For `test_withdraw()` method:

```
setUp()
test_withdraw()
```

If you run the test:

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

It'll output the following:

```
test_deposit (test_bank_account.TestBankAccount) ... ok
test_withdraw (test_bank_account.TestBankAccount) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

The following adds the `tearDown()` method to the `TestBankAccount`:

```
import unittest

from bank_account import BankAccount, InsufficientFund


class TestBankAccount(unittest.TestCase):
    def setUp(self) -> None:
        self.bank_account = BankAccount(100)

    def test_deposit(self):
        self.bank_account.deposit(100)
        self.assertEqual(self.bank_account.balance, 200)

    def test_withdraw(self):
```

```
        self.bank_account.withdraw(50)
        self.assertEqual(self.bank_account.balance, 50)

    def tearDown(self) -> None:
        self.bank_account = None
```

The `tearDown()` method assigns `None` to the `self.bank_account` instance.

Summary

- Fixtures are functions and methods that execute before and after test code blocks execute.
- The `setUpModule()` and `tearDownModule()` run before and after all test methods in the module.
- The `setUpclass()` and `tearDownClass()` run before and after all test methods in a test class.
- The `setUp()` and `tearDown()` run before and after each test method of a test class.