Hands-On Lab: Creating an Initial State Using Test Fixtures

Estimated time needed: 30 minutes

Welcome to **Creating an Initial State Using Test Fixtures**. Test fixtures are used to establish an initial known state before and after running tests. With test fixtures, you can describe what the test environment looks like before a test, or suite of tests, is run, and then again after the test runs.

Learning Objectives

After completing this lab, you will be able to:

- Create test fixtures for setting up and tearing down initial state
- Load test data from external files for testing your code
- Use loaded test data in your test cases and assertions

About Theia

Theia is an open-source IDE (Integrated Development Environment) that can be run on desktop or on cloud. You will be using the Theia IDE to do this lab. When you log into the Theia environment, you are presented with a 'dedicated computer on the cloud' exclusively for you. This is available to you as long as you work on the labs. Once you log off, this 'dedicated computer on the cloud' is deleted along with any files you may have created. So, it is a good idea to finish your labs in a single session. If you finish part of the lab and return to the Theia lab later, you may have to start from the beginning. Plan to work out all your Theia labs when you have the time to finish the complete lab in a single session.

Set Up the Lab Environment

We have a little preparation to do before we can start the lab.

Open a Terminal

Open a terminal window by using the menu in the editor: Terminal > New Terminal.

In the terminal, if you are not already in the `/home/projects` folder, change to your project folder now.

```
1. 1
```

```
1. cd /home/project
```

Copied! Executed!

Clone the Code Repo

Now let's get the code that we need to test. To do this, you will use the `git clone` command to clone the git repository:

```
1. 1
```

```
1. git clone https://github.com/ibm-developer-skills-network/duwjx-tdd_bdd_PracticeCode.git
```

Copied! Executed!

Change into the Lab Folder

Once you have cloned the repository, change to the lab directory,

```
1. 1
```

```
1. cd duwjx-tdd_bdd_PracticeCode/labs/03_test_fixtures
```

Copied! Executed!

Install Python Dependencies

The final preparation step is to use `pip` to install the Python packages needed for the lab:

```
1. 1
```

```
1. pip install -r requirements.txt
```

Copied! Executed!

You are now ready to start the lab.

Optional

If the command prompt becomes too long you can shorten it to something more manageable like this:

```
1. 1
```

```
1. export PS1="[\[\033[01;32m\]\u\[\033[00m\]: \[\033[01;34m\]\W\[\033[00m\]]\$ "
```

Copied! Executed!

Review of Test Fixtures

In this lab, you are going to use the various test fixtures that are available in the PyUnit package. You should review the code below that shows you what test fixtures are available and when they will be invoked.

During the lab, you will select the proper place to add code. For example, you may need to add code that will run once before all of the tests in a class. You should know that the `setUpClass()` method will run once before all of the tests in the test case.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
def setUpModule():             # runs once before any tests
def tearDownModule():          # runs once after all tests
#
class MyTestCases(TestCase):   # the start of a test case

    @classmethod
    def setUpClass(cls):       # runs once before test case
#
    @classmethod
    def tearDownClass(cls):    # runs once after test case
#
    def setUp(self):           # runs before each test

    def tearDown(self):        # runs after each test
```

Copied!

Open the Code Editor

In this lab, you will see the different ways in which test fixtures can be used to set up and tear down the initial state before and after testing.

In the IDE, navigate to the `duwjx-tdd_bdd_PracticeCode/labs/03_test_fixtures` folder. This is where all of the source code that you will be using for this lab is located.

```
1
2
3
```

```
duwjx-tdd_bdd_PracticeCode
└── labs
    └── 03_test_fixtures
```

Copied!

You will do all of your editing work in the file `tests/test_account.py`. Open that up in the editor to get started.

Open **test_account.py** in IDE

Step 1: Initialize the Database

In this step, you are going to set up a test fixture to connect and disconnect from the database. You only need to do this once before and after all of the tests.

Your Task

Think about which of the text fixtures are best used for connecting to a database before all tests, and disconnecting from the database after all tests.

The following **SQLAlchemy** commands will help you do this:

Command	Definition
db.create_all()	Make the sqlalchemy tables
db.session.close()	Close the database connection

Use the class level fixtures to invoke `db.create_all()` before all tests and `db.session.close()` after all tests.

Solution

▶ Click here for the solution.

Run the Tests

Run `nosetests` to make sure that your test case executes without errors.

```
1
```

```
nosetests
```

Copied! Executed!

Step 2: Load Test Data

In this step, you are going to load some test data so that it can be used during testing. This should only need to be done once before all tests so you will do this in a class method.

Your Task

There is test data in a file under the `tests/fixtures` folder called `account_data.json`.

Load the data from `tests/fixtures/account_data.json` into a global variable called `ACCOUNT_DATA` that has already been declared.

The Python code to load the data is:

```
1
2
```

```
with open('tests/fixtures/account_data.json') as json_data:
    ACCOUNT_DATA = json.load(json_data)
```

Copied!

Solution

▶ Click here for the solution.

Run the Tests

Run `nosetests` to make sure that your test case executes without errors.

```
1
```

```
nosetests
```

Copied! Executed!

Step 3: Write a Test Case to Create an Account

Now you are ready to write your first test. You will create a single account using the `ACCOUNT_DATA` dictionary that has test data for five accounts.

Your Task

The `Account` class has a `create()` method that can be used to add an account to the database. It also has an `all()` method that performs a query that returns all accounts.

Your test cases should create an account and then call the `Account.all()` method and assert that one account was returned.

Solution

▶ Click here for the solution.

Run the Tests

Run `nosetests` to make sure that your test case passes.

```
1
```

```
nosetests
```

Copied! Executed!

You should see:

```
Test Account Model
- Test create a single Account

Name                 Stmts   Miss  Cover   Missing
--------------------------------------------------
models/__init__.py       6      0   100%
models/account.py       40     13    68%   26, 30, 34-35, 45-48, 52-54, 74-75
--------------------------------------------------
TOTAL                   46     13    72%
----------------------------------------------------------------------
Ran 1 test in 0.344s

OK
```

Step 4: Write a Test Case to Create All Accounts

Now that you know that one account can be successfully created, let's write a test case that creates all five of the accounts in the `ACCOUNT_DATA` dictionary.

Your Task

Use a `for` loop to load all of the data from the `ACCOUNT_DATA` dictionary and then use the `Account.all()` method to retrieve them and assert that the number of accounts returned is equal to the number of accounts in the test data dictionary.

Solution

▶ Click here for the solution.

Run the Tests

Run `nosetests` to make sure that your test case passes.

```
1
```

```
nosetests
```

Copied! Executed!

ERROR: This time the tests did not pass! You should have received two errors about **AssertionError: 6 != 5** and **AssertionError: 7 != 1**:

```
1
2
3
4
5
6
7
```

```
======================================================================
FAIL: Test Account creation using known data
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/rofrano/Code/duwjx-tdd_bdd_PracticeCode/labs/03_test_fixtures/tests/test_account.py", line 52, in test_create_an_account
    self.assertEqual(len(Account.all()), 1)
AssertionError: 7 != 1
```

Copied!

Why were seven accounts returned when we only expect one?

Let's see how we can fix this in the next step.

Step 5: Clear out the tables before each test

The reason that your test case failed is because data from a previous test has affected the outcome of the next test. To avoid this, you need to add more test fixtures that will run before and after each test.

Your Task

One way of removing the data from a table is with the `db.session.query(<Table>).delete()` command where `<Table>` is the class name of the tables. This will delete all of the records in the table. You must also use `db.session.commit()` to commit this change. Add that to the fixture that runs before each test.

The correct syntax for your `Account` class is:

```
1
```

```
db.session.query(Account).delete()
```

Copied!

It's also a good idea to use the `db.session.remove()` command after each test. Add that to the fixture that runs after each test.

Solution

▶ Click here for the solution.

Run the Tests

Run `nosetests` to make sure that your test case passes.

```
1
```

```
nosetests
```

Copied! Executed!

You should see the following report:

```
Test Account Model
- Test creating multiple Accounts
- Test create a single Account

Name                  Stmts   Miss  Cover   Missing
-----------------------------------------------------
models/__init__.py        6      0   100%
models/account.py        40     13    68%   26, 30, 34-35, 45-48, 52-54, 74-75
-----------------------------------------------------
TOTAL                    46     13    72%
----------------------------------------------------------------------
Ran 2 tests in 0.395s

OK
```

Congratulations! All of your test cases have passed this time.

Conclusion

Congratulations on Completing the Test Fixtures Lab

Hopefully you now have a good understanding of how to use test fixtures in your testing. You have seen how using test fixtures allows you to control the state of the system before and after each test so that tests run in isolation and get repeatable results every time. Now try writing test cases for the code on some of your personal projects.

Author(s)

John Rofrano

Changelog

Date	Version	Changed by	Change Description
2022-04-14	1.0	Rofrano	Create new Lab