Hands-on Lab: Using Factories and Fakes

Estimated time needed: 30 minutes

Welcome to the **Using Factories and Fakes** lab. You often need fake data to test against. Of course, you can use some hard-coded sample data in your tests. But what if you need hundreds, or even thousands, of records of test data? That can get tedious to create and maintain.

In this lab, you're going to see how to use a popular Python package called **FactoryBoy** to provide fake data for testing.

Learning Objectives

After completing this lab, you will be able to:

- Summarize how to create a Factory class
- Use the Faker class and Fuzzy attributes to provide realistic test data
- Write test cases that use Factory classes to provide test data

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

You have a little preparation to do before you 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.

```
cd /home/project
```

Clone the Code Repo

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

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

Change into the Lab Folder

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

```
cd duwjx-tdd_bdd_PracticeCode/labs/05_factories_and_fakes
```

Install Python Dependencies

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

```
python3.8 -m pip install -r requirements.txt
```

You are now ready to start the lab.

Optional

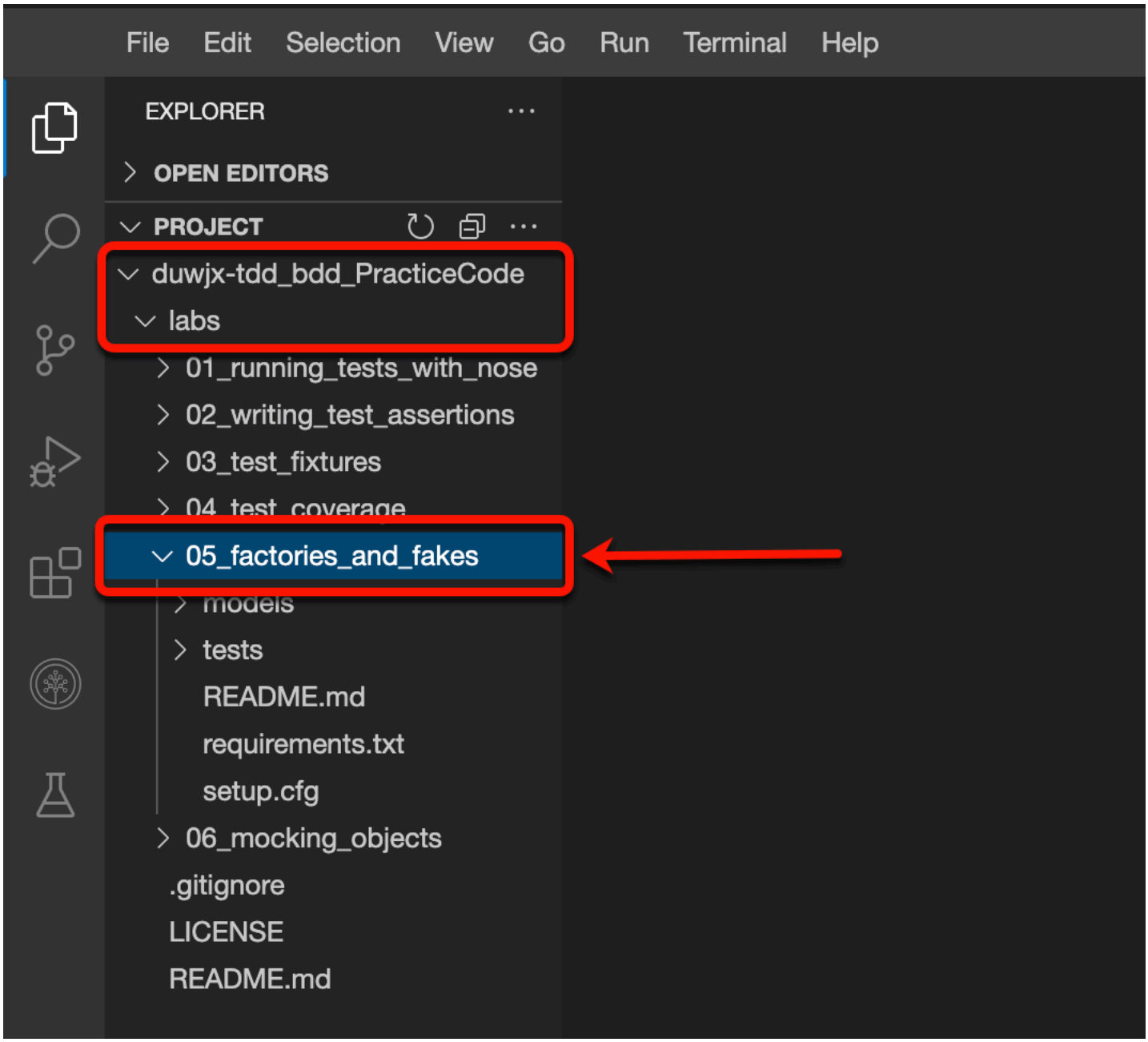
If working in the terminal becomes difficult because the command prompt is very long, you can shorten the prompt using the following command:

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

Navigate to the Code

In the IDE, navigate to the `duwjx-tdd_bdd_PracticeCode/labs/05_factories_and_fakes` folder. This folder contains all of the source code that you will use for this lab.

```
duwjx-tdd_bdd_PracticeCode/labs/05_factories_and_fakes
```



Step 1: Run nosetests

Before you make any changes to your code, you should check that all of the test cases are passing. Otherwise, if you encounter failing test cases later, you won't know if you caused them to fail or if they were failing before you changed anything.

Run `nosetests` and make sure that all of the tests pass with **100%** test coverage.

```
nosetests
```

You should see the following output:

```
[theia: 05_factories_and_fakes]$ nosetests

Test Account Model
- Test creating multiple Accounts
- Test Account creation using known data
- Test Account update using known data
- Test account from dict
- Test invalid ID update
- Test the representation of an account
- Test account to dict
- Test Account update using known data

Name                    Stmts   Miss  Cover   Missing
-----------------------------------------------------
models/__init__.py          6      0   100%
models/account.py          40      0   100%
-----------------------------------------------------
TOTAL                      46      0   100%
----------------------------------------------------------------------
Ran 8 tests in 0.485s

OK
```

All tests are colored green! This means they all pass, so you can now move on to modifying the code.

Step 2: Create an AccountFactory class

In this step, you will create an `AccountFactory` class.

Open the `models/account.py` file to familiarize yourself with the attributes of the `Account` class. These are the same attributes that you will need to add to the `AccountFactory` class.

Open **account.py** in IDE

Open the `tests/factories.py` file in the IDE editor. This is the file in which you will add the attributes of the `Account` class to the `AccountFactory` class.

Open **factories.py** in IDE

You want to take advantage of the fact that **FactoryBoy** comes with the **Faker** class. This class has Fake providers and a number of Fuzzy attributes.

Here are some useful providers for the Faker class:

```
Faker("name")
Faker("email")
Faker("phone_number")
```

Here are some Fuzzy attributes you might find useful:

```
FuzzyChoice(choices=[True, False])
FuzzyDate(date(2008, 1, 1))
```

Your Task

Use the **Faker** providers and **Fuzzy** attributes to create fake data for the `id`, `name`, `email`, `phone_number`, `disabled`, and `date_joined` fields by adding them to the `AccountFactory` class.

Solution

▼ Click here for the solution.

In `factories.py`, overwrite the code from line 16 onward by pasting in the following code in its place. Be sure to indent properly.

```
class AccountFactory(factory.Factory):
    """ Creates fake Accounts """
    class Meta:
        model = Account
    id = factory.Sequence(lambda n: n)
    name = factory.Faker("name")
    email = factory.Faker("email")
    phone_number = factory.Faker("phone_number")
    disabled = FuzzyChoice(choices=[True, False])
    date_joined = FuzzyDate(date(2008, 1, 1))
```

Step 3: Update the Test Cases

In this step, you will update the test cases to use the new `AccountFactory` that you created in the previous step.

Open the `tests/test_account.py` file. Then add the following `import` near the top of the file, after the other `imports`. This will import your new `AccountFactory` class from the `factories` module:

Open **test_account.py** in IDE

```
from factories import AccountFactory
```

In the remaining steps, your goal is to change all references to `Account` so that they now use `AccountFactory`. You will do this one test at a time.

Your Task

Start with the `test_create_all_accounts()` test:

- Remove the references to `ACCOUNT_DATA` and `Account` and replace them with `AccountFactory`.
- Change the code to create ten Accounts.

Solution

▼ Click here for the solution.

```
def test_create_all_accounts(self):
    """ Test creating multiple Accounts """
    for _ in range(10):
        account = AccountFactory()
        account.create()
    self.assertEqual(len(Account.all()), 10)
```

Run the Tests

Run `nosetests` to make sure the test cases still pass.

```
nosetests
```

Step 4: Update test_create_an_account()

In this step, you will update the `test_create_an_account()` test.

Your Task

In test_account.py, modify the code in the `test_create_an_account()` test to remove the references to `ACCOUNT_DATA` and `Account` and replace them with `AccountFactory`.

Solution

▼ Click here for the solution.

```
        def test_create_an_account(self):
            """ Test Account creation using known data """
            account = AccountFactory()
            account.create()
            self.assertEqual(len(Account.all()), 1)
```

Run the Tests

Run `nosetests` to make sure the test cases still pass.

```
    nosetests
```

Step 5: Update test_to_dict()

In this step, you will update the `test_to_dict()` test.

Your Task

In test_account.py, modify the code in the `test_to_dict()` test to remove the references to `ACCOUNT_DATA` and `Account` and replace them with `AccountFactory`.

Solution

▼ Click here for the solution.

```
        def test_to_dict(self):
            """ Test account to dict """
            account = AccountFactory()
            result = account.to_dict()
            self.assertEqual(account.name, result["name"])
            self.assertEqual(account.email, result["email"])
            self.assertEqual(account.phone_number, result["phone_number"])
            self.assertEqual(account.disabled, result["disabled"])
            self.assertEqual(account.date_joined, result["date_joined"])
```

Run the Tests

Run `nosetests` to make sure the test cases still pass.

```
nosetests
```

Step 6: Update test_from_dict()

In this step, you will update the `test_from_dict()` test.

Your Task

In test_account.py, modify the code in the `test_from_dict()` test to remove the references to `ACCOUNT_DATA` and `Account` and replace them with `AccountFactory`.

Solution

▼ Click here for the solution.

```
    def test_from_dict(self):
        """ Test account from dict """
        data = AccountFactory().to_dict()
        account = Account()
        account.from_dict(data)
        self.assertEqual(account.name, data["name"])
        self.assertEqual(account.email, data["email"])
        self.assertEqual(account.phone_number, data["phone_number"])
        self.assertEqual(account.disabled, data["disabled"])
```

Run the Tests

Run `nosetests` to make sure the test cases still pass.

```
nosetests
```

Step 7: Update test_update_an_account()

In this step, you will update the `test_update_an_account()` test.

Your Task

In test_account.py, modify the code to in the `test_update_an_account()` test to remove the references to `ACCOUNT_DATA` and `Account` and replace with `AccountFactory`.

Solution

▼ Click here for the solution.

```
    def test_update_an_account(self):
        """ Test Account update using known data """
        account = AccountFactory()
        account.create()
        self.assertIsNotNone(account.id)
        account.name = "Rumpelstiltskin"
        account.update()
        found = Account.find(account.id)
        self.assertEqual(found.name, account.name)
```

Run the Tests

Run `nosetests` to make sure the test cases still pass.

```
nosetests
```

Step 8: Update test_invalid_id_on_update()

In this step, you will update the `test_invalid_id_on_update()` test.

Your Task

In test_account.py, modify the code in the `test_invalid_id_on_update()` test to remove the references to `ACCOUNT_DATA` and `Account` and replace them with `AccountFactory`.

Solution

▼ Click here for the solution.

```
    def test_invalid_id_on_update(self):
        """ Test invalid ID update """
        account = AccountFactory()
        account.id = None
        self.assertRaises(DataValidationError, account.update)
```

Nosetest

Run `nosetests` to make sure the test cases still pass.

```
nosetests
```

Step 9: Update test_delete_an_account()

In this step, you will update the `test_delete_an_account()` test.

Your Task

In test_account.py, modify the code in the `test_delete_an_account()` test to remove the references to `ACCOUNT_DATA` and `Account` and replace them with `AccountFactory`.

Solution

▼ Click here for the solution.

```
        def test_delete_an_account(self):
            """ Test Account update using known data """
            account = AccountFactory()
            account.create()
            self.assertEqual(len(Account.all()), 1)
            account.delete()
            self.assertEqual(len(Account.all()), 0)
```

Nosetest

Run `nosetests` to make sure the test cases still pass.

```
nosetests
```

Step 10: Remove ACCOUNT_DATA references

Since you have replaced all instances of `ACCOUNT_DATA` with `AccountFactory`, you can clean up the code. In test_account.py, you will remove all remaining references to `ACCOUNT_DATA` and remove the lines that load it from the JSON data file.

Task A

Remove line 31 from `setUp()`:

```
self.rand = randrange(0, len(ACCOUNT_DATA))
```

▼ Click here for the solution.

```
        def setUp(self):
            """Truncate the tables"""
            db.session.query(Account).delete()
            db.session.commit()
```

Task B

Remove lines 20-22 from `setUpClass()`:

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

▼ Click here for the solution.

```
        @classmethod
        def setUpClass(cls):
            """ Load data needed by tests """
            db.create_all()  # make our sqlalchemy tables
```

You can also delete line `11` that declares `ACCOUNT_DATA`:

```
    ACCOUNT_DATA = {}   # delete this line
```

Task C

Finally, delete lines 4-5, which import `json` and `randrange`:

```
    import json
    from random import randrange
```

Run the Tests

Save your changes and run `nosetests` one last time to make sure that the test cases still pass.

```
    nosetests
```

You should see the following results:

```
    Name                 Stmts   Miss  Cover   Missing
    ----------------------------------------------------
    models/__init__.py       6      0   100%
    models/account.py       40      0   100%
    ----------------------------------------------------
    TOTAL                   46      0   100%
    ----------------------------------------------------------------------
    Ran 8 tests in 0.548s
```

Conclusion

Congratulations on Completing the Factories and Fakes Lab

Hopefully you can now build a factory for your classes using **Faker** and **Fuzzy** attributes. You can also use a factory class in your test cases to provide unlimited test data.

Try using a factory in your personal projects. Anywhere you have created static data to test your code, you can substitute dynamic factories to make testing more robust.

Author(s)

John J. Rofrano