

0x03. Unittests and Integration Tests

Unit tests My Profile Planning Integration tests



Weight: 1

Projects(/projects/current)



Project over - took place from Jul 25, 2024 6:00 AM to Jul 30, 2024 6:00 AM



An auto QA review will automatically make corrections(to_review)



Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)

In a nutshell...

- **Auto QA review:** 17.0/26 mandatory & 2.0/4 optional

- **Altogether: 98.07%**



Curriculums(/dashboards/my_curriculums)

◦ Mandatory: 65.38%

◦ Optional: 50.0%

◦ Calculation: $65.38\% + (65.38\% * 50.0\%) == 98.07\%$ 

Concepts(/concepts)



Conference rooms(/dashboards/video_rooms)



Servers(/servers)



Sandboxes(/user_containers/current)



Tools(/dashboards/my_tools)



Video on demand(/dashboards/videos)

Unit test vs. Integration test



Pair programming

Unit testing is the process of testing that a particular function returns expected results for different set of inputs. A unit test is proposed to test standard inputs and corner cases. A unit test should only test the logic defined inside the tested function. Most calls to additional functions should be mocked, especially if they make network or database calls.

Discord(<https://discord.com/app>)

The goal of a unit test is to answer the question: if everything defined outside this function works as expected, does this function work as expected?

Integration tests aim to test a code path end-to-end. In general, only low level functions that make external calls such as HTTP requests, file i/o, database i/o, etc. are mocked.

Integration tests are placed between every part of your code.



Execute your tests with



```
$ python -m unittest path/to/test_file.py
```

Resources

Read or watch:



- My Planning(/planning/me)
- unittest — Unit testing framework (/rltoken/a_AEObGK8jeqPtTPmm-gIA)
- unittest.mock — mock object library (/rltoken/PKetnACd7FfRiU8_kpe5EA)



- How to mock a readonly property with mock? (/rltoken/2ueVPK1kWZuz525FvZ1v2Q)
- Projects(/projects/current)
- parameterized (/rltoken/ml7qc3Y42aZ7GTILXDxgEg)
- Memoization (/rltoken/x83Hdr54q4Vax5xQ2Z3HSA)



QA Reviews I can make(/corrections/to_review)

Learning Objectives



At the end of this project, you are expected to be able to explain to anyone (/rltoken/NfT-nNKrNHGrDMY-Qm-1Dg), **without the help of Google**:

- The difference between unit and integration tests.
- Common testing patterns such as mocking, parametrizations and fixtures



Curriculums(/dashboards/my_curriculums)

Requirements



Concepts(/concepts)

- All your files will be interpreted/compiled on Ubuntu 18.04 LTS using python3 (version 3.7)
- All your files should end with a new line



The first line of all your files should be exactly:#!/usr/bin/env python3

- A README.md file, at the root of the folder of the project, is mandatory
- Your code should use the pycodestyle style (version 2.5)



Servers(/servers)

- All your files must be executable
- All your modules should have a documentation (python3 -c 'print(__import__("my_module").__doc__)')



Sandboxes(/user_containers/current)

- All your classes should have a documentation (python3 -c 'print(__import__("my_module").MyClass.__doc__)')



All your functions (inside and outside a class) should have a documentation (python3 -c

'print(__import__("my_module").my_function.__doc__)' and python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)')



Video on demand(/dashboards/videos)

- A documentation is not a simple word, it's a real sentence explaining what's the purpose of the module, class or method (the length of it will be verified)
- All your functions and coroutines must be type-annotated.



Peers(/users/peers)

Required Files



utils.py (or download (<https://intranet-projects-files.s3.amazonaws.com/webstack/utils.py>))





Click to show/hide file contents

My Profile(/users/my_profile)

 `client.py` (or download (<https://intranet-projects-files.s3.amazonaws.com/webstack/client.py>))

Click to show/hide file contents

 `fixtures.py` (or download (<https://intranet-projects-files.s3.amazonaws.com/webstack/fixtures.py>))

 My Planning(/planning/me)

Click to show/hide file contents

 Projects(/projects/current)

Tasks


5 A Reviews I can make(/corrections/to_review)


0. Parameterize a unit test

mandatory


 Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)


Score: 100.0% (Checks completed: 100.0%)


 Familiarize yourself with the `utils.access_nested_map` function and understand its purpose. Play with it in the Python console to make sure you understand.

 In this task you will write the first unit test for `utils.access_nested_map`.
Concepts(/concepts)

Create a `TestAccessNestedMap` class that inherits from `unittest.TestCase`.


 Implement the `TestAccessNestedMap.test_access_nested_map` method to test that the method returns what it is supposed to.
Conference rooms(/dashboards/video_rooms)

 Decorate the method with `@parameterized.expand` to test the function for following inputs:
Servers(/servers)



```
nested_map={"a": 1}, path=("a",)
nested_map={"a": {"b": 2}}, path=("a",)
nested_map={"a": {"b": 2}}, path=("a", "b")
```


Sandboxes(/user_containers/current)

 For each of these inputs test with `assertEqual` that the function returns the expected result.
Tools(/dashboards/my_tools)

The body of the test method should not be longer than 2 lines.


 Video on demand(/dashboards/videos)

Repo:

- GitHub repository: `alx-backend-python`
 - Directory: `0x03-Unittests_and_integration_tests`
 - File: `test_utils.py`
-
- Peers(/users/peers)

 Discord(<https://discord.com/app>)

Check submission

 Get a sandbox

View results



My Profile(/users/my_profile)

1. Parameterize a unit test

mandatory

(/)

Score: 100.0% (Checks completed: 100.0%)



Home(/)



My Planning(planning/me)



Projects(/projects/current)

```
nested_map={}, path=("a",)
```

```
nested_map={"a": 1}, path=("a", "b")
```

Also make sure that the exception message is as expected.



QA Reviews I can make(/corrections/to_review)

Repo:



Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)

- GitHub repository: alx-backend-python
- Directory: 0x03-Unittests_and_integration_tests
- File: test_utils.py



Curriculums(/dashboards/my_curriculums)

Check submission

> Get a sandbox

View results



Concepts(/concepts)

2. Mock HTTP calls

mandatory



Conference rooms(/dashboards/video_rooms)

Score: 100.0% (Checks completed: 100.0%)



Familiarize yourself with the `utils.get_json` function.

Servers(servers)

Define the `TestGetJson(unittest.TestCase)` class and implement the



`TestGetJson.test_get_json` method to test that `utils.get_json` returns the expected result.

Sandboxes(/user_containers/current)

We don't want to make any actual external HTTP calls. Use `unittest.mock.patch` to patch



requests.get. Make sure it returns a `Mock` object with a `json` method that returns `test_payload`

Tools(/dashboards/my_tools)

which you parametrize alongside the `test_url` that you will pass to `get_json` with the following inputs:



Video on demand(/dashboards/videos)

```
test_url="http://example.com", test_payload={"payload": True}
```

```
test_url="http://holberton.io", test_payload={"payload": False}
```



Test that the mocked `get` method was called exactly once (per input) with `test_url` as argument.

Peers(/users/peers)

Test that the output of `get_json` is equal to `test_payload`.



Discord(https://discord.com/app)

Repo:

- GitHub repository: alx-backend-python
 - Directory: 0x03-Unittests_and_integration_tests
 - File: test_utils.py
- My Profile(/users/my_profile)



[Check submission](#)[Get a sandbox](#)[View results](#)[\(/\)](#)

3. Parameterize and patch

mandatory[Home\(/\)](#)

Score: 100.0% (Checks completed: 100.0%)

Read about [My Planning\(planning.html\)](#) and parameterize yourself with the `utils.memoize` decorator.Implement the `TestMemoize(unittest.TestCase)` class with a `test_memoize` method.[Projects\(/projects/current\)](#)Inside `test_memoize`, define following classclass `TestClass`:
`QA Reviews I can make(/corrections/to_review)``def a_method(self):``return 42``Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)``@memoize``def a_property(self):``return self.a_method()`[Curriculums\(/dashboards/my_curriculums\)](#)Use `unittest.mock.patch` to mock `a_method`. Test that when calling `a_property` twice, the correct result is returned but `a_method` is only called once using `assert_called_once`.[Concepts\(/concepts\)](#)**Repo:**[Conference rooms\(/dashboards/video_rooms\)](#)

- GitHub repository: `alx-backend-python`
- Directory: `0x03-Unittests_and_integration_tests`
- File: `test_utils.py`

[Servers\(/servers\)](#)[Sandboxes\(/user_containers/current\)](#)[Check submission](#)[Get a sandbox](#)[View results](#)[Tools\(/dashboards/my_tools\)](#)

4. Parameterize and patch as decorators

mandatory[Video on demand\(/dashboards/videos\)](#)

Score: 100.0% (Checks completed: 100.0%)

Familiarize yourself with the `client.GithubOrgClient` class.[Peers\(/users/peers\)](#)In a new `test_client.py` file, declare the `TestGithubOrgClient(unittest.TestCase)` class and[Discord\(https://discord.com/app\)](#)This method should test that `GithubOrgClient.org` returns the correct value.Use `@patch` as a decorator to make sure `get_json` is called once with the expected argument but make sure it is not executed.Use `@parameterized.expand` decorator to parametrize the test with a couple of `org` examples to pass to `GithubOrgClient`, in this order:

- google



- abc

(/)
Of course, no external HTTP calls should be made.



Home(/)

Repo:



- GitHub repository: alx-backend-python
- My Planning(/planning/me)
• Directory: 0x03-Unittests_and_integration_tests
- File: test_client.py



Projects(/projects/current)

Check submission

>_ Get a sandbox

View results



QA Reviews I can make(/corrections/to_review)

5. Mocking a property

mandatory



Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)

Score: 50.0% (Checks completed: 50.0%)

memoize turns methods into properties. Read up on how to mock a property (see resource).



Curriculums(/dashboards/my_curriculums)

Implement the `test_public_repos_url` method to unit-test `GithubOrgClient._public_repos_url`.

Use `patch` as a context manager to patch `GithubOrgClient.org` and make it return a known payload.



Concepts(/concepts)

Test that the result of `_public_repos_url` is the expected one based on the mocked payload.



Conference rooms(/dashboards/video_rooms)

Repo:



- GitHub repository: alx-backend-python
- Servers(/servers)
• Directory: 0x03-Unittests_and_integration_tests
- File: test_client.py



Sandboxes(/user_containers/current)

Check submission

Mark submission

>_ Get a sandbox

View results



Tools(/dashboards/my_tools)



Video on demand(/dashboards/videos)

6. More patching

mandatory

Score: 40.0% (Checks completed: 40.0%)



Peers(/users/peers)

Implement `TestGithubOrgClient.test_public_repos` to unit-test `GithubOrgClient.public_repos`.



Use `@patch` as a decorator to mock `get_json` and make it return a payload of your choice.

Discord(<https://discord.com/api>)

Use `patch` as a context manager to mock `GithubOrgClient._public_repos_url` and return a value of your choice.

Test that the list of repos is what you expect from the chosen payload.

Test that the mocked `get_json` was called once.

My Profile(/user_profile)



Repo:



- GitHub repository: alx-backend-python
- Directory: 0x03-Unittests_and_integration_tests
- File: test_client.py



Check submission

My Planning(/planning/me)

Mark submission

> Get a sandbox

View results



7. Parameterize

Projects(/projects/current)

mandatory



Score: 0.0% (Checks completed: 0.0%)

QA Reviews I can make(/corrections/to_review)

Implement `TestGithubOrgClient.test_has_license` to unit-test `GithubOrgClient.has_license`.



Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)

Parameterize the test with the following inputs

```
repo={"license": {"key": "my_license"}}, license_key="my_license"
repo={"license": {"key": "other_license"}}, license_key="my_license"
```



Curriculums(/dashboards/my_curriculums)

You should also parameterize the expected returned value.



Concepts(/concepts)

Repo:



- GitHub repository: alx-backend-python
- Directory: 0x03-Unittests_and_integration_tests
- File: test_client.py



Servers(/servers)

Check submission

Mark submission

> Get a sandbox

View results



Sandboxes(/user_containers/current)



Tools(/dashboards/my_tools)

8. Integration test: fixtures

mandatory



Video on demand(/dashboards/videos)

Score: 0.0% (Checks completed: 0.0%)

We want to test the `GithubOrgClient.public_repos` method in an integration test. That means that we will only mock code that sends external requests.



Peers(/users/peers)

Create the `TestIntegrationGithubOrgClient(unittest.TestCase)` class and implement the `setUpClass` and `tearDownClass` which are part of the `unittest.TestCase` API.



Discord(<https://discord.com/app>)

Use `@parameterized_class` to decorate the class and parameterize it with fixtures found in `fixtures.py`. The file contains the following fixtures:

```
org_payload, repos_payload, expected_repos, apache2_repos
```

My Profile(/users/my_profile)

The `setUpClass` should mock `requests.get` to return example payloads found in the fixtures.



Use `patch` to start a patcher named `get_patcher`, and use `side_effect` to make sure the mock of `requests.get(url).json()` returns the correct fixtures for the various values of `url` that you anticipate to receive.

Implement the `tearDownClass` class method to stop the patcher.

Home(/)

Repo:

My Planning(/planning/me)

- GitHub repository: `alx-backend-python`
- Directory: `0x03-Unittests_and_integration_tests`
- File: `test_client.py`

Check submission

Mark submission

Get a sandbox

View results

9. Integration tests

Evaluation quizzes(/dashboards/my_current_evaluation_quizzes)

Score: 50.0% (Checks completed: 50.0%)

Implement the `test_public_repos` method to test `GithubOrgClient.public_repos`.
Make sure that the method returns the expected results based on the fixtures.

Implement `test_public_repos_with_license` to test the `public_repos` with the argument `license="apache-2.0"` and make sure the result matches the expected value from the fixtures.

Conference rooms(/dashboards/video_rooms)

Repo:

Servers(/servers)

- GitHub repository: `alx-backend-python`
- Directory: `0x03-Unittests_and_integration_tests`
- File: `test_client.py`

Check submission

Mark submission

Get a sandbox

View results

Tools(/dashboards/my_tools)

Video on demand(/dashboards/videos)

Peers(/users/peers)

Discord(<https://discord.com/app>)

My Profile(/users/my_profile)

Copyright © 2024 ALX, All rights reserved.