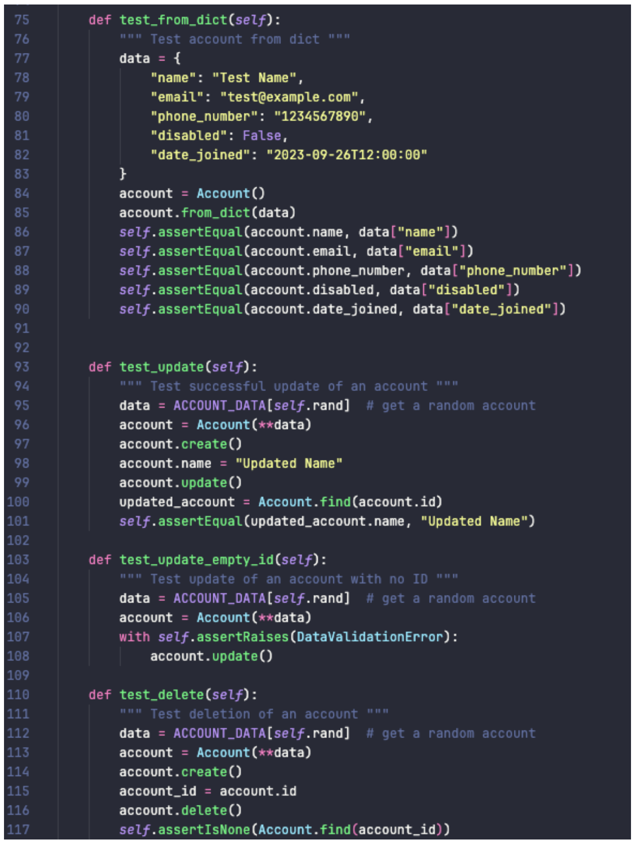
Angad Bhatti
CS 472 Section 1001
26 September 2023
Testing Lab

Tasks 1-3:
In this lab, we were given 5 tasks, JPacman Test Coverage, Increasing Coverage on JPacman, JaCoCo Report on JPacman, Working with Python Test Coverage, and Test Driven Development (TDD). For the first task, the test coverage on the code was not good enough because only 16% of the Classes, 9% methods, and 8% lines were covered. However, adding the PlayerTest class increased the coverage to 41.8% for classes, 24.5% for methods, and 23.3% for lines. For Task 2, the coverage results from JaCoCo are similar to the ones I got from IntelliJ. The JaCoCo results give more detail on the missed instructions, branches, lines, and highlight the lines that are covered and not covered. The source code visualization was really helpful and is something I prefer when finding missed lines over the basic report from IntelliJ coverage window.

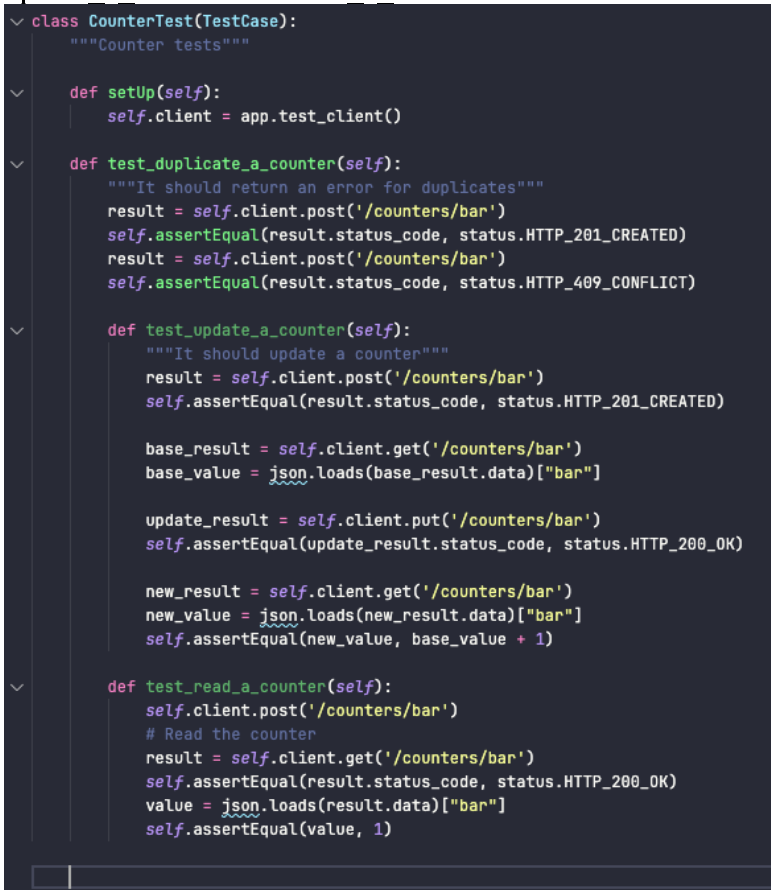
Tasks 4:
In Task 4, I had to implement test for from_dict, create, update, and delete (Figure 1). To run nosetest I had to install nose2 and run 'nose2 --with-coverage' due to my Python version.



```python
    def test_from_dict(self):
        """ Test account from dict """
        data = {
            "name": "Test Name",
            "email": "test@example.com",
            "phone_number": "1234567890",
            "disabled": False,
            "date_joined": "2023-09-26T12:00:00"
        }
        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"])
        self.assertEqual(account.date_joined, data["date_joined"])


    def test_update(self):
        """ Test successful update of an account """
        data = ACCOUNT_DATA[self.rand]  # get a random account
        account = Account(**data)
        account.create()
        account.name = "Updated Name"
        account.update()
        updated_account = Account.find(account.id)
        self.assertEqual(updated_account.name, "Updated Name")

    def test_update_empty_id(self):
        """ Test update of an account with no ID """
        data = ACCOUNT_DATA[self.rand]  # get a random account
        account = Account(**data)
        with self.assertRaises(DataValidationError):
            account.update()

    def test_delete(self):
        """ Test deletion of an account """
        data = ACCOUNT_DATA[self.rand]  # get a random account
        account = Account(**data)
        account.create()
        account_id = account.id
        account.delete()
        self.assertIsNone(Account.find(account_id))
```

Figure 1: Task 4 Code

Task 5:
In Task 5, we had to write test for creating, reading, and updating counter. In the first phase of this task, the test failed (Red Phase). Second Phase we had to add functionality to pass (Green Phase). Lastly we had to reduce redundancy in the code (Blue Phase). I also added test for update_a_counter and read_a_counter.



```
class CounterTest(TestCase):
    """Counter tests"""

    def setUp(self):
        self.client = app.test_client()

    def test_duplicate_a_counter(self):
        """It should return an error for duplicates"""
        result = self.client.post('/counters/bar')
        self.assertEqual(result.status_code, status.HTTP_201_CREATED)
        result = self.client.post('/counters/bar')
        self.assertEqual(result.status_code, status.HTTP_409_CONFLICT)

        def test_update_a_counter(self):
            """It should update a counter"""
            result = self.client.post('/counters/bar')
            self.assertEqual(result.status_code, status.HTTP_201_CREATED)

            base_result = self.client.get('/counters/bar')
            base_value = json.loads(base_result.data)["bar"]

            update_result = self.client.put('/counters/bar')
            self.assertEqual(update_result.status_code, status.HTTP_200_OK)

            new_result = self.client.get('/counters/bar')
            new_value = json.loads(new_result.data)["bar"]
            self.assertEqual(new_value, base_value + 1)

        def test_read_a_counter(self):
            self.client.post('/counters/bar')
            # Read the counter
            result = self.client.get('/counters/bar')
            self.assertEqual(result.status_code, status.HTTP_200_OK)
            value = json.loads(result.data)["bar"]
            self.assertEqual(value, 1)
```

Figure 2: Counter Test

Repos:
https://github.com/angadb27/tdd
https://github.com/angadb27/jpacman
https://github.com/angadb27/test_coverage