Exercises chapter 11: Testing your code

Saul SL

June 2023

1 Exercise 11-1 City, Country

Write a function that accepts two parameters: a city name and a country name. The function should return a single string of the form City, Country, such as Santiago, Chile. Store the function in a module called `city_functions.py`. Create a file called `test_cities.py` that tests the function you just wrote (remember that you need to import unittest and the function you want to test). Write a method called `test_city_country()` to verify that calling your function with values such as 'santiago' and 'chile' results in the correct string. Run `test_cities.py`, and make sure `test_city_country()` passes.

2 Exercise 11-2 Population

Modify your function so it requires a third parameter, population. It should now return a single string of the form City, Country – population xxx, such as Santiago, Chile – population 5000000. Run test _cities.py again. Make sure `test_city_country()` fails this time. Modify the function so the population parameter is optional. Run test _cities.py again, and make sure `test_city_country()` passes again. Write a second test called `test_city_country_population()` that verifies you can call your function with the values 'santiago', 'chile', and 'population=5000000'. Run `test_cities.py` again, and make sure this new test passes.

2.1 city_functions.py

```python
def format_city(name, country, population=None):
    """Format the location of a city
    """
    if population:
        name_f = f"{name.title()}, {country.title()} - {population}"
    else:
        name_f = f"{name.title()}, {country.title()}"
    return name_f
```

2.2 test_cities.py

```python
def format_city(name, country, population=None):
    """Format the location of a city
    """
    if population:
        name_f = f"{name.title()}, {country.title()} - {population}"
    else:
        name_f = f"{name.title()}, {country.title()}"
    return name_f
```

3 11-3 Employee

Write a class called Employee. The `__init__()` method should take in a first name, a last name, and an annual salary, and store each of these as attributes. Write a method called `give_raise()` that adds $5,000 to

the annual salary by default but also accepts a different raise amount. Write a test case for Employee. Write two test methods, `test_give_default_raise()` and `test_give_custom_raise()`. Use the `setUp()` method so you don't have to create a new employee instance in each test method. Run your test case, and make sure both tests pass.

3.1 employee

```
class Employee():
    """Definies a default employee
    """

    def __init__(self, first_name, last_name, annu_salary):
        # super(Employee, self).__init__()
        self.first_name = first_name
        self.last_name = last_name
        self.annu_salary = annu_salary

    def give_raise(self, amount=5000):
        self.annu_salary += int(amount)
```

3.2 test_employee

```
import unittest
import employee


class EmployeeTestCase(unittest.TestCase):
    """Test functions defined in the employee class
    """

    def setUp(self):
        self.worker = employee.Employee('tim', 'jones', 50_000)

    def test_raise_default(self):
        self.worker.give_raise()
        salary = self.worker.annu_salary
        self.assertEqual(salary, 55_000)

    def test_raise_custom(self):
        self.worker.give_raise(20_000)
        salary = self.worker.annu_salary
        self.assertEqual(salary, 70_000)


if __name__ == '__main__':
    unittest.main()
```