**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления»**

# **Курс «Базовые компоненты интернет-технологий»**

## Отчёт по лабораторной работе №5

## «Модульное тестирование в Python.»

Выполнил:

студент группы ИУ5-33Б Кузнецов В. А.

подпись: , дата:

Проверил:

лектор Гапанюк Ю. Е.

подпись: , дата:

2022 г.

**Задание:**

1. Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4.
2. Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования.
3. Разработайте модульные тесты. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк (не менее 3 тестов).
   * BDD - фреймворк (не менее 3 тестов).

**Текст программы:**

Field\_test.py

**import** unittest  
**from** lab\_python\_fp.field **import** field  
  
  
**class** TestField(unittest.TestCase):  
 **def** setUp(self):  
 self.goods = [  
 {**'title'**: **'Ковер'**, **'price'**: 2000, **'color'**: **'green'**},  
 {**'title'**: **'Диван для отдыха'**, **'price'**: 5300, **'color'**: **'black'**}  
 ]  
  
 **def** test\_one\_argument(self):  
 result = list(field(self.goods, **'title'**))  
 answer = [**'Ковер'**, **'Диван для отдыха'**]  
 self.assertEqual(result, answer)  
  
 **def** test\_many\_arguments(self):  
 result = list(field(self.goods, **'title'**, **'price'**))  
 answer = [{**'title'**: **'Ковер'**, **'price'**: 2000}, {**'title'**: **'Диван для отдыха'**, **'price'**: 5300}]  
 self.assertEqual(result, answer)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

test\_unique.py

**import** unittest  
**from** lab\_python\_fp.unique **import** Unique  
**from** lab\_python\_fp.gen\_random **import** gen\_random  
  
  
**class** TestUnique(unittest.TestCase):  
 **def** test\_numbers(self):  
 data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 result = list(Unique(data))  
 answer = [1, 2]  
 self.assertEqual(result, answer)  
  
 **def** test\_random\_generator(self):  
 data = gen\_random(10, 1, 3)  
 result = set(Unique(data))  
 answer = set(range(1, 4))  
 self.assertTrue(answer.issubset(result))  
  
 **def** test\_letters(self):  
 data = [**'a'**, **'A'**, **'b'**, **'B'**, **'a'**, **'A'**, **'b'**, **'B'**]  
 result = list(Unique(data))  
 answer = [**'a'**, **'A'**, **'b'**, **'B'**]  
 self.assertEqual(result, answer)  
  
 **def** test\_letters\_ignoring\_case(self):  
 data = [**'a'**, **'A'**, **'b'**, **'B'**, **'a'**, **'A'**, **'b'**, **'B'**]  
 result = list(Unique(data, ignore\_case=**True**))  
 answer = [**'a'**, **'b'**]  
 self.assertEqual(result, answer)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

test\_gen\_random.feature

**Feature**: Testing generator gen\_random  
 **Scenario**: Testing a range with positive boundaries  
 **Given** amount of numbers 10 and minimum 1 and maximum 10  
 **When** we generate numbers  
 **Then** we have to get the specified number of numbers in a given range  
  
 **Scenario**: Testing a range with negative boundaries  
 **Given** amount of numbers 10 and minimum -10 and maximum -1  
 **When** we generate numbers  
 **Then** we have to get the specified number of numbers in a given range  
  
 **Scenario**: Testing a range with positive and negative boundaries  
 **Given** amount of numbers 10 and minimum -10 and maximum 10  
 **When** we generate numbers  
 **Then** we have to get the specified number of numbers in a given range

test\_gen\_random.py

**from** behave **import** Given, When, Then  
**from** lab\_python\_fp.gen\_random **import** gen\_random  
  
  
@Given(**'amount of numbers {amount} and minimum {min\_} and maximum {max\_}'**)  
**def** step\_impl(context, amount, min\_, max\_):  
 context.amount = int(amount)  
 context.min = int(min\_)  
 context.max = int(max\_)  
 context.generator = gen\_random(context.amount, context.min, context.max)  
  
  
@When(**'we generate numbers'**)  
**def** step\_impl(context):  
 context.numbers = list(context.generator)  
  
  
@Then(**'we have to get the specified number of numbers in a given range'**)  
**def** step\_impl(context):  
 **assert** len(context.numbers) == context.amount  
 **assert** all(context.min <= i <= context.max **for** i **in** context.numbers)

**Пример выполнения**





