НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Звіт
із лабораторної роботи №1
із дисципліни «Автоматизація тестування
програмного забезпечення»
на тему
Модульне тестування (unit testing) програм на мові Python

Виконав:
студент групи КМ-81
Донченко Б. М.

Керівник:
асистент Громова В. В.

Київ — 2020

ЗМІСТ

1 ПОСТАНОВКА ЗАДАЧІ

Мета роботи: з`ясувати, що таке автоматичне тестування; ознайомитися з модулями Python, які допомагають провести автоматичне тестування; закріпити отриманні знання на практиці шляхом написання тестового модуля, що реалізує тест-кейси.

Варіант: №8

Завдання:

1. Скласти програму, яка здійснює перетворення величин з радіанної міри в градусну і навпаки. Програма повинна запитувати, яке перетворення потрібно здійснити, і виконати вказану дію. Всі величини вводити з клавіатури.

2. У одновимірному масиві, що складається з n елементів, обчислити середнє арифметичне значення елементів масиву.

2 ОПИС ПРОГРАМИ

Програма: prog1.py

В даній програмі реалізовано перетворення величин з радіанної міри в градусну і навпаки, за допомогою бібліотеки math, а саме функцій radian() та degrees(). Залежно від взятої функції ми створюємо ті функції, які ми будемо потім тестувати(rad_to_grad() та grad_to_rad(), відповідно). Як і сказано в умові, користувач обирає, яку операцію має виконати програма.

Програма: prog1_test.py

Для початку тестування імпортуємо потрібну бібліотеку unittest та програму, яку тестуємо, prog_1. Створюємо клас Lab1Test для створення потрібних test case.

Для кожного test case у класі TestLab створюється окремий метод , в якому перевіряється певна умова. Для уточнення умов див. Розробка тест-кейсів.

Програма:prog2.py

В даній програмі було реалізовано знаходження середнього арифметичного для списку стандартними функціями Python, а саме sum() та len(). Створюємо функцію list_task(), в якій і реалізоване саме завдання. Як і сказано в умові, користувач сам вводить дані в список.

Програма: prog2_test.py

Програма реалізована аналогічно до prog1_test.py.

2 ФОРМУВАННЯ ТЕСТ КЕЙСІВ

Далі буде наведена таблиця всіх реалізованих тест-кейсів в програмі prog1_test.py.

Функція rad_to_grad()

Номер тесту	Мета створення	Вхідні дані	Очікуваний результат
1	Перевірити коректну роботу програми	1	57.3
2	Перевірити коректну роботу програми	2	114.59
3	Перевірити поведінку програми при введенні неправильного типу дані	“s”	Помилка TypeError, адже в функцію потрібно передавати дані типу int, float
4	Перевірити поведінку програми, якщо в функцію не передано жодного аргументу	“”	Помилка TypeError, адже в функцію не передано аргументів

Функція grad_to_rad()

Номер тесту	Мета створення	Вхідні дані	Очікуваний результат
1	Перевірити коректну роботу програми	30	0.52
2	Перевірити коректну роботу програми	90	1.57
3	Перевірити поведінку програми при введенні неправильного типу дані	“v”	Помилка TypeError, адже в функцію потрібно передавати дані типу int, float
4	Перевірити поведінку програми, якщо в функцію не передано жодного аргументу	“”	Помилка TypeError, адже в функцію не передано аргументів

Наступна таблиця демонструє усі тест-кейси в програмі prog2_test.py.

Номер тесту	Мета створення	Вхідні дані	Очікуваний результат
1	Перевірити коректну роботу програми	[1, 5, 3, 7, 9]	5.0
2	Перевірити коректну роботу програми з даними типу float	[1.6, 5.2, 3.2, 7.5, 9.4	5.38
3	Перевірити коректну роботу програми з від'ємними значеннями	[-1, -5, -8, -6, -9]	-5.8
4	Перевірити поведінку програми, якщо в функцію передано дані іншого типу	[1, 5, "3", 7, 9]	Помилка TypeError, адже в функцію потрібно передавати дані типу int, float
5	Перевірити поведінку програми при порожньому списку	[]	Помилка TypeError, адже в функцію не передано аргументів

Оскільки ці дві програми, як ми бачимо по тест-кейсам, є лінійними(відсутні гілки рішень), тоді можна зробити висновок, шо виключні ситуації виникають лише при некоректних данних.

ВИСНОВКИ

У ході лабораторної роботи ми з`ясувати, що таке автоматичне тестування; ознайомилися з модулями Python, які допомагають провести автоматичне тестування; закріпили отриманні знання на практиці шляхом написання тестового модуля, що реалізує тест-кейси. З написаних програмам стало зрозуміло, якщо відсутня гілка рішень тоді програма буде мати виключні ситуації лише через некоректні дані.

ДОДАТОК 1. ТЕКСТ ПРОГРАМ

1.1 prog1.py

```
import math
print("Варіант 28")
def rad_to_grad(x):
    res = math.degrees(x)
    return round(res, 2)
def grad_to_rad(x):
    res = math.radians(x)
    return round(res, 2)
choice = str(input("""
1) Перевод из радиан в градусы;
2) Перевод из градусов в радианы.
Ваш выбор: """))
if choice == "1":
    radian = float(input("Введите радианы: "))
    result = rad_to_grad(radian)
    print(result)
elif choice == '2':
    degrees = float(input("Введите градусы: "))
    result = grad_to_rad(degrees)
    print(result)
else:
    print("Данной операции нету!")
```

1.2 prog1_test.py

```
import unittest
from prog1 import rad_to_grad, grad_to_rad

class Lab1Test(unittest.TestCase):
        #Проверка на 1
    def test_rad_to_grad_correct1(self):
        self.assertEqual(rad_to_grad(1), 57.3)
        #Проверка на 2
    def test_rad_to_grad_correct2(self):
        self.assertEqual(rad_to_grad(2), 114.59)
        #Проверка на "s"
    def test_rad_to_grad_error1(self):
        self.assertRaises(TypeError, rad_to_grad, "s")
        #Проверка на "testing"
    def test_rad_to_grad_error2(self):
        self.assertRaises(TypeError, rad_to_grad, "testing")
        #Проверка на ""
    def test_rad_to_grad_empty(self):
        self.assertRaises(TypeError, rad_to_grad, "")

        #Проверка на 30
    def test_grad_to_rad_correct1(self):
        self.assertEqual(grad_to_rad(30), 0.52)
        #Проверка на 90
    def test_grad_to_rad_correct2(self):
        self.assertEqual(grad_to_rad(90), 1.57)
        #Проверка на "v"
    def test_grad_to_rad_error1(self):
        self.assertRaises(TypeError, grad_to_rad, "v")
        #Проверка на "auto"
    def test_grad_to_rad_error2(self):
        self.assertRaises(TypeError, grad_to_rad, "auto")
        #Проверка на ""
    def test_grad_to_rad_empty(self):
        self.assertRaises(TypeError, grad_to_rad, "")
if __name__ == '__main__':
    unittest.main()
```

1.2 prog2.py

```
print("Варінт 28")
def list_task(array):
    res = sum(array)/len(array)
    return res

N = int(input("Количество элементов в списке: "))
arr = []
for i in range(N):
    number = float(input("Введите число: "))
    arr.append(number)
print(list_task(arr))
```

1.2 prog2_test.py

```
import unittest
from prog2 import list_task

class Lab1Test(unittest.TestCase):
    #Проверка на обычный список
    def test_task2_correct1(self):
        arr = [1, 5, 3, 7, 9]
        self.assertAlmostEqual(list_task(arr), 5.0)
    #Проверка на список с числами с плавующей точкой
    def test_task2_correct2(self):
        arr = [1.6, 5.2, 3.2, 7.5, 9.4]
        self.assertAlmostEqual(list_task(arr), 5.38)
    #Проверка на минусовые числа
    def test_task2_correct3(self):
        arr = [-1, -5, -8, -6, -9]
        self.assertAlmostEqual(list_task(arr), -5.8)
    #Проверка на пустой список
    def test_task2_empty(self):
        arr = []
        self.assertRaises(ZeroDivisionError, list_task, arr)
    #Проверка на другие типы данных
    def test_task2_string(self):
        arr = [1, 5, "3", 7, 9]
        self.assertRaises(TypeError, list_task, arr)


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

ДОДАТОК 2. СКРІНШОТИ ТЕСТУВАННЯ ПРОГРАМ

```
Варіант 28

1) Перевод из радиан в градусы;
2) Перевод из градусов в радианы.
Ваш выбор: 1
Введите радианы: 30
1718.87
.........
----------------------------------------------------------------------
Ran 10 tests in 0.079s

OK
>>>
```

Рисунок 2.1. Тестування першої програми

```
Варінт 28
Количество элементов в списке: 3
Введите число: 1
Введите число: 3
Введите число: 6
3.3333333333333335
.....
----------------------------------------------------------------------
Ran 5 tests in 0.032s

OK
>>>
```

Рисунок 2.2. Тестування другої програми

ДОДАТОК 3. ВІДПОВІДІ НА ЗАПИТАННЯ

1. Що таке специфікація програмного продукту?

Специфікація програмного продукту - якомога повний перелік вимог до поведінки програмного забезпечення, що розробляється; повний і точний опис функцій і обмежень майбутнього програмного забезпечення.

2. Що таке тестові дані, тест-кейс?

Тестові дані – набір записів даних для ініціалізації змінних тестового сценарію.

Тест-кейс – алгоритм, що описує сукупність кроків, конкретних умов та параметрів, необхідних для перевірки реалізації функції, що тестується чи її частини.

3. Що таке модульне тестування (unit testing)?

Це метод тестування ПЗ, що полягає у тестуванні окремих модулів програми.

4. Для чого потрібне модульне тестування (unit testing)?

Модульне тестування необхідно для впевненості в тому, що модуль програми працює так, як це було заплановано архітектурно і відповідає усім вимогам.

5. Яке місце модульного тестування в життєвому циклі розробки ПЗ?

Модульні тести дозволяють переконатись у правильній роботі написаного модуля протягом усього циклу розробки ПЗ.

6. Які переваги і недоліки модульного тестування?

Переваги: простота створення та документації, можливість швидкої модифікації.

Недоліки: зі збільшенням складності модуля для тестування збільшується складність написання тесту, а також з’являється проблема неповного покриття гілок рішень, складність тестування модулів, що залежать від усієї системи.

7. Які межі застосування модульного тестування?

Зазвичай – лише модулі, які можуть працювати незалежно від інших частин програми.

8. До якого типу тестування відноситься модульне тестування?

Модульне тестування відноситься до раннього типу тестування.

9. Хто зазвичай займається розробкою тест-кейсів для модульного тестування?

Зазвичай розробкою займається програміст ПЗ.

10. Яка специфіка створення тестових даних і тест-кейсів для модульного тестування?

При створенні тест-кейсу потрібно описувати вхідні дані, які використовуються для даного тестування, а також очікуваний результат, до якого ці дані призведуть. Також повинен бути описаний чіткий алгоритм виконання дій.

11. Якими будуть тест-кейси для модульного тестування програми, що обчислює суму двох дійсних чисел a і b?

- Додавання чисел типу int
- Додавання чисел типу float
- Додавання чисел у експоненціальній формі
- Некоректні дані (додавання даних типу string)

12. Якими будуть тест-кейси для модульного тестування програми, що обчислює частку двох дійсних чисел a і b?

- Ділення int на int
- Ділення на нуль
- Ділення даних типу float
- Ділення даних типу string

13. Чи документуються помилки, знайдені під час модульного тестування в системі менеджменту помилок (Bug Tracking System)?

Залежить від конкретного інструментарію, що використовується. Проте загальна відповідь – ні.

14. Що таке «розробка через тестування» (test-driven development)?

Це техніка розробки ПЗ, що полягає в написанні спочатку тесту, що покриває бажані зміни, а лише потім коду, що дозволить пройти цей тест.