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

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

Виконав:
студент групи КМ-82
Бубела Д. В.

Керівник:
асистент
Громова В. В.

Київ — 2020

ЗМІСТ

МЕТА РОБОТИ

Ознайомитися з фреймворком unittest зі стандартної бібліотеки Python. Провести автоматизоване модульне тестування (unit testing) програм на мові Python складеними за індивідуальним завданням вказаними в «Загальних інструкціях до лабораторних робіт» відповідно до номеру.

Завдання 1

2) Увести з клавіатури координати двох точок A (x1, y1) і B (x2, y2). Скласти алгоритм, який визначає, яка з точок знаходиться ближче до початку координат.

Завдання 2

2) У одновимірному масиві, що складається з n елементів, обчислити суму елементів масиву, розташованих між мінімальним і максимальним елементами.

Розробити тестові випадки для повного покриття варіантів поведінки програми.

1 ПОСТАНОВКА ЗАДАЧІ

Завдання 1

Програма повинна приймати ввід користувача для 2х точок по дві координати кожна. Потрібно створити функцію, що реалізовуватиме алгоритм, який визначає яка з точок знаходиться ближче до початку координат та повертає результат.

Завдання 2

Програма повинна реалізовувати функцію, що визначає мінімальний та максимальний елемент в масиві представленому списком та рахує суму між ними (включно).

Модульні тести повинні бути в окремому файлі та тестувати програму, що знаходиться в іншому файлі. Тести не повинні залежати один від одного.

2 ОСНОВНА ЧАСТИНА

Програми будуть створені таким чином, що дозволяло б їх тестування фреймворком «unittest», тобто основна логічна частина має бути оформлена в функцію, що повертає значення, за яким можна перевірити коректність роботи програми або алгоритму.

Завдання 1

Згідно умові завдання введення даних повинне бути користувачем, тобто з ’stdin’. Для забезпечення передачі корректних даних функції, що буде їх опрацьовувати програма 1, має бути створена додаткова функція fl_input, що не пропускає далі некоректного вводу від користувача. Також повинна бути реалізація інформування користувача про необхідні дії для забезпечення роботи програми. Основну логічну задачі програми буде виконувати функція closer(A, B). Вона приймає дві точки, на яких власне і буде застосовуватись мета програми. Коли відстань від одної з них до початку координат буде більшою за відстань іншої точки до початку координат — вона буде додаватись до спеціального списку c[] з точками з найменшою відстанню до початку координат. Якщо найменшої точки не буде існувати — список c[] буде пустим. Функція повертає список найменших точок.

Для роботи з тестами буде створено клас Prog_1Test що буде дочірнім від класу unittest.TestCase. Всередині класу будуть використовуватись методи assertEqual та assertRaises. Тестування запускається за допомогою unittest.main() коли програма тестування виконується як основна програма виконання.

Завдання 2

Буде створена функція sum_min_max(lst), яка буде визначати мінімальний

елемент з масиву, максимальний, зберігатиме їх індекси та рахуватиме суму слайсу списку з першим і останнім елементом, мінімуму та максимуму початкового списку відповідно. В завданні чітко не вказані дії при існуванні кількох максимумів та/або мінімумів, тому програма перевірятиметься на знаходження суми в будь-яких варіантах, тобто мінімум може вибиратися з мінімумів довільно, так само і з максимумами. Конкретна реалізація буде шукати перше входження відповідних елементів. Пошук індексів елементів здійснюватиметься функціями min та max з порядкових номерів елементів з додатковим аргументом функції що порівнює елементи потрібного списку.

Для роботи з тестами буде створено клас Prog_2Test що буде дочірнім від класу unittest.TestCase. Всередині класу будуть використовуватись методи assertEqual, assertIn та assertRaises. Тестування запускається за допомогою unittest.main() коли програма тестування виконується як основна програма виконання.

3 РОЗРОБКА ТЕСТ-КЕЙСІВ

Таблиця 3.1 – Тест-кейси завдання 1

Тест	Мета	Вхідні данні	Очікуваний результат
simple_equality	Перевірити 2 точки з однаковою відстанню	(2.2, 3.3), (-3.3, -2.2)	[]
int_compatibility	Перевірити виконання функції з цілими числами (чи поведінка однакова)	(22, 33), (-31, -24)	[(-31, -24)]
big_numbers	Перевірити виконання функції на числах більших ніж зазвичай (переповнення розрядної сітки)	(1000000000000000000.2, -9000000000000000000000.4), (1221221, 32312321)),	[(1221221, 32312321)]
negative_test	Перевірити виконання функції з від’ємними числами (чи враховує знак)	(-9.0, 9.0), (1.1, 1.1)	[(1.1, 1.1)]
bad_data	Перевірити виконання функції з некоректними даними	(-9.0, 9.0), (1.1, ’o’)	TypeError

Таблиця 3.2 – Тест-кейси завдання 2

Тест	Мета	Вхідні данні	Очікуваний результат
straight_sum	Перевірити виконання при стандартних даних	[1, -6, 0, 34, 434]	462
empty	Перевірити виконання при пустому списку	[]	0
one	Перевірити виконання при співпадінні максимального і мінімального елементів	[41]	41
two	Перевірити виконання при 2х елементаз в порядку зростання	[1, 2]	3
reverse_sum	Перевірити виконання при умові, що мінімум знаходиться в списку після максимума	[1, 434, 0, 34, -6]	462
float_compatibility	Перевірити виконання на дійсних числах	[1.32, -6.23, 0, 34, 434.001]	461.770999 99999996
negative_test	Перевірити виконання при всіх від’ємних числах	[-794, -43, -8989, -11, -89]	-9000
multiple_minmax	Перевірити виконання при кількох максимумах, мінімумах	[-89, 1, 90, -89, 2, 90]	або 2, або 5, або 1, або 3

Продовження таблиці 3.2

Тест	Мета	Вхідні данні	Очікуваний результат
nested_lst	Перевірити виконання при вкладених списках, в специфікації не задано обробку вкладених списків	[12, [21, 32]]	TypeError
bad_type_sum	Перевірити правильність суми при некоректних даних між мінімальним і максимальним елементами	[1, ’o’, 2]	TypeError
bad_type_cmp	Перевірити правильність суми при некоректних даних, що не знаходиться між мінімальним і максимальним елементами, так як невідомо як порівнювати букву з цифрою	[1, 2, ’o’]	TypeError

ВИСНОВКИ

Ми ознайомилися з фреймворком unittest, провели автоматизоване модульне тестування зроблених відповідно до завдання програм, склавши до них модульні тести в окремому файлі. Програми з обидвох завдань поводять себе передбачувано, та так як їх було задумано. В завданні 1 програму тестують 5 модульних тестів з назвами відповідно до деталей, що вони перевіряють, так само в завданні 2 з 11ма модульними тестами. Програми тести проходять успішно, що зображено на скрінах В.1 В.2

ПЕРЕЛІК ПОСИЛАНЬ

1. «Програмування на мові PYTHON: інструкції до виконання лабораторних робіт з дисципліни «Програмування-1. Основи програмування» / КПІ ім. Ігоря Сікорського; укладач В. В. Громова.

2. Python Software Foundation. unittest — Unit testing framework [Електронний ресурс] / Python Software Foundation.. – 2020. -– Режим доступу до ресурсу: https://docs.python.org/3/library/unittest.html.

Додаток А

Текст програм, які тестуються

Лістинг файлу prog_1.py

```
def closer(A, B):
    c = []
    if A[0]**2 + A[1]**2 < B[0]**2 + B[1]**2:
        c.append(A)
    if A[0]**2 + A[1]**2 > B[0]**2 + B[1]**2:
        c.append(B)
    return c


def fl_input(message):
    isCorrect = False
    while not isCorrect:
        try:
            fl = float(input(message))
        except ValueError:
            print("Невірний формат вводу")
        else:
            isCorrect = True
    return fl


if __name__ == '__main__':
    print("Визначення точки з ")
    print("Введіть дані точки A (x1, y1)")
    x1 = fl_input("x1 = ")
    y1 = fl_input("y1 = ")
    print("Введіть дані точки B (x2, y2)")
    x2 = fl_input("x2 = ")
    y2 = fl_input("y2 = ")
    c = closer((x1, y1), (x2, y2))
    if len(c) == 0:
        print("Точки знаходяться на однаковій відстані")
    else:
        print(f"Точка з координатами {c[0]} знадиться ближче до початку координат")
```

Лістинг файлу prog_2.py

```
def sum_min_max(lst):
    if not lst:
        return 0
    minmax_idx = [min(range(len(lst)), key=lst.__getitem__),
                  max(range(len(lst)), key=lst.__getitem__)]
    minmax_idx.sort()
    return sum(lst[minmax_idx[0]:minmax_idx[1] + 1])
```

Додаток Б

Текст модулів, що реалізують автоматичне тестування

Лістинг файлу test_prog_1.py

```
import prog_1
import unittest


class Prog_1Test(unittest.TestCase):

    def test_simple_equality(self):
        self.assertEqual(prog_1.closer((2.2, 3.3), (-3.3, -2.2)), [])

    def test_int_compatibility(self):
        self.assertEqual(prog_1.closer((22, 33), (-31, -24)), [(-31, -24)])

    def test_big_numbers(self):
        self.assertEqual(prog_1.closer(
            (10000000000000000000.2, -90000000000000000000.4), (1221221, 32312321)),
            [(1221221, 32312321)])

    def test_negative_test(self):
        self.assertEqual(prog_1.closer((-9.0, 9.0), (1.1, 1.1)), [(1.1, 1.1)])

    def test_bad_data(self):
        self.assertRaises(TypeError, prog_1.closer, (-9.0, 9.0), (1.1, 'o'))


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

Лістинг файлу test_prog_2.py

```
import prog_2
import unittest


class Prog_2Test(unittest.TestCase):

    def test_straight_sum(self):
        self.assertEqual(
            prog_2.sum_min_max(
                [1, -6, 0, 34, 434]
            ),
            462
        )

    def test_empty(self):
        self.assertEqual(
            prog_2.sum_min_max(
                []
            ),
            0
        )

    def test_one(self):
        self.assertEqual(
            prog_2.sum_min_max(
                [41]
            ),
            41
        )

    def test_two(self):
        self.assertEqual(
            prog_2.sum_min_max(
                [1, 2]
            ),
            3
        )

    def test_reverse_sum(self):
        self.assertEqual(
            prog_2.sum_min_max(
                [1, 434, 0, 34, -6]
            ),
            462
        )

    def test_float_compatibility(self):
        self.assertEqual(
            prog_2.sum_min_max(
                [1.32, -6.23, 0, 34, 434.001]
```

```
            ),
            461.77099999999996
        )

    def test_negative_test(self):
        self.assertEqual(
            prog_2.sum_min_max(
                [-794, -43, -8989, -11, -89]
            ),
            -9000
        )

    def test_multiple_minmax(self):
        self.assertIn(
            prog_2.sum_min_max(
                [-89, 1, 90, -89, 2, 90]
            ),
            [2, 5, 1, 3]
        )

    def test_nested_lst(self):
        self.assertRaises(TypeError, prog_2.sum_min_max, [12, [21, 32]])

    def test_bad_type_sum(self):
        self.assertRaises(TypeError, prog_2.sum_min_max, [1, 'o', 2])

    def test_bad_type_cmp(self):
        self.assertRaises(TypeError, prog_2.sum_min_max, [1, 2, 'o'])


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

Додаток В

Скріншоти результатів виконання автотестів

```
~/K/A/L/№/source $ python -m unittest -v test_prog_1.py
test_bad_data (test_prog_1.Prog_1Test) ... ok
test_big_numbers (test_prog_1.Prog_1Test) ... ok
test_int_compatibility (test_prog_1.Prog_1Test) ... ok
test_negative_test (test_prog_1.Prog_1Test) ... ok
test_simple_equality (test_prog_1.Prog_1Test) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

Рисунок В.1 – Тестування програми 1

```
~/K/A/L/№/source $ python -m unittest -v test_prog_2.py
test_bad_type_cmp (test_prog_2.Prog_2Test) ... ok
test_bad_type_sum (test_prog_2.Prog_2Test) ... ok
test_empty (test_prog_2.Prog_2Test) ... ok
test_float_compatibility (test_prog_2.Prog_2Test) ... ok
test_multiple_minmax (test_prog_2.Prog_2Test) ... ok
test_negative_test (test_prog_2.Prog_2Test) ... ok
test_nested_lst (test_prog_2.Prog_2Test) ... ok
test_one (test_prog_2.Prog_2Test) ... ok
test_reverse_sum (test_prog_2.Prog_2Test) ... ok
test_straight_sum (test_prog_2.Prog_2Test) ... ok
test_two (test_prog_2.Prog_2Test) ... ok

----------------------------------------------------------------------
Ran 11 tests in 0.000s

OK
```

Рисунок В.2 – Тестування програми 2

Додаток Г

Відповіді на контрольні запитання

1) Що таке специфікація програмного продукту?

Специфікація вимог до ПЗ (software requirements specification, spec) - якомога повний перелік вимог до поведінки програмного забезпечення, що розробляється; повний і точний опис функцій і обмежень майбутнього програмного забезпечення. Специфікація зазвичай містить декілька типів вимог: системні, вимоги користувача, експлуатаційні та інші.

2) Що таке тестові дані, тест-кейс?

Тестові дані (test data) — це дані, які існують на початку виконання тесту і впливають на його роботу, або ж зазнають впливу зі сторони тестованої системи або компоненту. Тестові дані можуть бути: створені тестувальником, згенеровані спеціальною програмою, модифіковані реальні дані; повністю реальні дані.

Тест-кейс / тестовий випадок / тестова ситуація / тестовий сценарій / тест (test case) — 1) набір вхідних даних, умов виконання і очікуваних результатів, розроблений з метою перевірки певної функції або поведінки програмного забезпечення; 2) документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується, або її частини, тобто формальний запис тест-кейсу.

3) Що таке модульне тестування (unit testing)?

Модульне тестування (англ. Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою.

4) Для чого потрібно модульне тестування (unit testing)?

Модульне тестування потрібно для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

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

Йде після етапу кодування й передує етапу супроводу і експлуатації. Включає виконання комплексного тестування всієї програмної системи спеціальною групою і виправлення помилок.

6) Які переваги і недоліки модульного тестування?

Модульне тестування допомагає знайти помилки раніше в циклі розробки ПЗ, що робить розробку дешевшою та швидшою. Дозволяє програмісту, коли він буде змінювати код (проводити рефакторинг) бути впевненим, що модуль працює вірно. Може бути застосоване в інтеграційному тестуванні: тестування окремих модулів та сукупності цих модулів робить інтеграційне тестування легшим. Модульні тести являють собою специфічний вид документації до системи. Розробники можуть подивитися на модульний тест, щоб дізнатися про функції, що виконує модуль, та як його застосовувати. Під час розробки програмного забезпечення методом TDD (Test-driven development), модульний тест стає частиною дизайну. Кожен тест визначає потрібні класи та методи, їх очікувану поведінку.

Тестування програмного забезпечення не може знайти всіх помилок у програмі. У більшості програм неможливо прорахувати кожен варіант виконання. Це також вірно для модульного тестування. Крім того, модульне тестування, власне, повинне тестувати тільки модулі. Так що цей вид тестування не зможе знайти інтеграційні помилки та інші: помилки архітектури, проблеми з витримкою навантажень на ПЗ. Як і будь-який вид тестування, модульне тестування може визначити лише наявність помилок, а не їх відсутність. Тестування ПЗ — це

комбінаторна задача. Наприклад, кожне логічне судження повинно мати не менш двох тестів: один перевіряє результат істинно, а другий хибно. Внаслідок цього, часто на кожний рядок коду доводиться написати від 3 до 5 рядків тесту. Це вимагає багато часу та грошей, які часто не варті результату.

7) Які межі застосування модульного тестування?

Модульне тестування є головною частиною деяких методологій програмування, як от екстремальне програмування. Воно служить для ізоляції окремих частин програми, тому може бути використане як основа архітектури розробки.

8) До якого типу тестування відноситься модульне тестування?

Модульне тестування відноситься до тестів, які перевіряють функціональність певного розділу коду, зазвичай на функціональному рівні. Його відносять до тестування методом білої скриньки.

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

Зазвичай розробкою тест-кейсів займається програміст, який пише програму. Це робиться перед власне написанням коду або після вже в автоматичному або ручному режимі.

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

Тестові дані вибираються таким чином, щоб при їх багаторазовому введенні в тести програми — вона давала однаковий передбачуваний результат. Тест-кейси і тестові дані підготовлюються програмістом, що тестує дану функцію, до якої застосовується тестування.

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

Таблиця Г.1 – Тест-кейси. Сума a b

Класи даних	a	b	Очікуваний результат
Коректні дані	2	3	5
Некоректні дані	1	s	TypeError

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

Таблиця Г.2 – Тест-кейси. частка a b

Класи даних	a	b	Очікуваний результат
Коректні дані	4	-2	-2
Коректні дані	0	1124325342542355	0
Некоректні дані	1	0	ZeroDivisionError
Некоректні дані	1	y	TypeError

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

Не документуються, оскільки модульне тестування використовується в процесі розробки програмістами, а не окремо користувачами.

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

Технологія розробки програмного забезпечення, яка використовує короткі ітерації розробки, що починаються з попереднього написання тестів, які визначають необхідні покращення або нові функції. Кожна ітерація має на меті розробити код, який пройде ці тести.