Технології програмування

Лабораторна робота №2

**Робота з файлами. Юніт тести.**

**Мета роботи**: Використовуючи теоретичне підґрунтя про роботу з файлами та тестування коду у мові Python розширити програму телефонного довідника студентів додавши функціонал, що буде вказано в завданні до лабораторної роботи.

## Теоретичні відомості

### Параметри командного рядка

Одним із механізмом визначення параметрів необхідних для виконання програми – використання аргументів командного рядка. Для можливості використання аргументів командного необхідно підключити модуль **argv**

Розглянемо приклад:

```
from sys import argv

print(f"Script name: {argv[0]}")
print(f"Input parameter: {argv[1]}")
```

В самому початку підключаються модуль **argv**, зо забезпечує можливість використання параметрів командного рядка. Все, що було вказано користувачем під час запуску програми зберігається в список доступ до елементів якого здійснюється використовуючи індекси.

Запустивши програму на виконання наступним чином:

```
python lab2.py lab2.csv
```

Отримаємо результат:

```
Script name: lab2.py
Input parameter: lab2.csv
```

Під нульовим індексом завжди зберігається ім'я програми, яка запускається. Починаючи з індексу один розміщуються параметри які буди вказані під час запуску програми на виконання.

**Робота з файлами**

Для роботи з файлами існують набір стандартних функцій, тож для відкриття, закриття, читання та запису інформації достатньо використовувати готовий функціонал.

Для того, щоб відкрити файл необхідно виконати функцію open() (https://docs.python.org/3/library/functions.html#open). Функція повертає посилання на файл, яке можна використовувати для наступних маніпуляцій. При закінченні використання файлу, його необхідно закрити.

```
file_name = argv[1]
file = open(file_name, "r")

file.close()
```

Проте, використовуючи ключове слово **with** можна не використовувати **close()** в кінці використання файлу.

Після відкриття файлу існує можливість пройти всі рядки, що в ньому збережені. Розглянемо приклад:

```
file_name = argv[1]
with open(file_name, "r") as file:
    for line in file:
        print(f"line from file: {line}")
```

Отримаємо результат

```
line from file: Name,Phone
line from file: Bob,1112233
line from file: Dilan,2223344
line from file: Zak,3334455
```

Для запису даних в файл використовується функція **write()**

```
with open(file_name, "a") as file:
   file.write("New student name,1231213")
```

Для роботи з файлами формату **CSV** розроблено однойменний модуль, який одразу може сформувати словники з вхідних даних, що збережені в форматі CSV (https://docs.python.org/3/library/csv.html).

Функція **DictReader**() забезпечу читання файлу у вигляді словників

Розглянемо приклад:

```
file_name = argv[1]

students = []

with open(file_name) as file:
    reader = csv.DictReader(file)
    for row in reader:
        students.append({"Name":row["Name"],
"Phone":row["Phone"]})

print(students)
```

Отримаємо результат

```
[{'Name': 'Bob', 'Phone': '1112233'}, {'Name':
'Dilan', 'Phone': '2223344'}, {'Name': 'Zak',
'Phone': '3334455'}]
```

Для запису даних в CSV файл необхідно виконати дещо більше дій: відкрити файл для запису, створити обект класу **DictWriter**(), записати верхній рівень з назвою стовбців та безпосередньо записати данні в файл.

```
import csv
```

```
with open("lab2_out.csv", "w", newline='') as
csvfile:
    fieldnames = ["Name", "Age"]
    writer = csv.DictWriter(csvfile,
fieldnames=fieldnames)
    writer.writeheader()
    writer.writerow({'Name': 'Ihor', 'Age': '37'})
```

З прикладами програм можна ознайомитись в файлі **lab2_sample_io.py**

Приклад CSV файлу **lab2.csv**

**Юніт тести**

Процес виконання завдання може включати розбиття великої задачі на маленькі під задачі, при цьому результатом виконання маленької частина має бути певна кількість функцій, сумарний результат виконання яких задовольняє критеріям виділеної під задачі. Для того, щоб впевнитись, що написаний код виконує саме те, що вказано в завданні, розробник формує набір тестів яким піддається написаний в рамках виконання завдання код. **Тестування коду – написання коду для тестування коду.** При цьому поняття Юніт тест означає тестування окремого функціоналу, наприклад однієї функції (юніту).

Для мові Python існує стороння бібліотека **pytest**, яка реалізує механізм написання тестів. Детальний опис бібліотеки доступний за посиланням https://docs.pytest.org/en/stable/

Для початку використання бібліотеки необхідно виконати інсталяцію

```
pip install pytest
```

Розглянемо в якості прикладу завдання на написання програми калькулятор, що запитує у користувача два параметри над якими необхідно виконати дії. В даному прикладі операції додавання та множення винесені

в окремі функції. Саме ці функціями будуть виступати окремими юнітами для тестування.

```
def add(a, b):
    return a + b

def mul(a, b):
    return a * b

def main():
    a = int(input("What is a: "))
    b = int(input("What is b: "))
    print(add(a, b))
    print(mul(a, b))

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

Перед початком написання тестів необхідно відмітити існування ключового слова **assert**, що надає можливість перевірити на правдивість вказану умову. Також необхідно вказати, згідно конвенції назва юніт тесту має починатися зі слова **test_**

Наведемо приклад файлу з тестами:

```
from lab2_sample_calc import add
from lab2_sample_calc import mul

def test_add():
    assert add(2, 2) == 4
    assert add(3, 2) == 5
    assert add(4, 2) == 6

def test_mul_positive_both():
    assert mul(2, 2) == 4
    assert mul(3, 2) == 6
    assert mul(4, 2) == 8

def test_mul_positive_and_negative():
```

```
    assert mul(2, -2) == -4
    assert mul(-3, 2) == -6
    assert mul(4, -2) == -8

def test_mul_negative_both():
    assert mul(-2, -2) == 4
    assert mul(-3, -2) == 6
    assert mul(-4, -2) == 8
```

З самого початку необхідно підключити файл і функції які будуть піддаватись тестуванню. Далі відбувається тестування юніту додавання та юніту множення. Зверніть увагу, що для юніту множення розроблено три незалежних тести.

Приклади розміщені в файлах **lab2_sample_calc.py** та **lab2_sample_calc_test.py**

Для запуску тестів використовується наступна команда:

```
pytest lab2_sample_calc_test.py
```

Приклад результату роботи

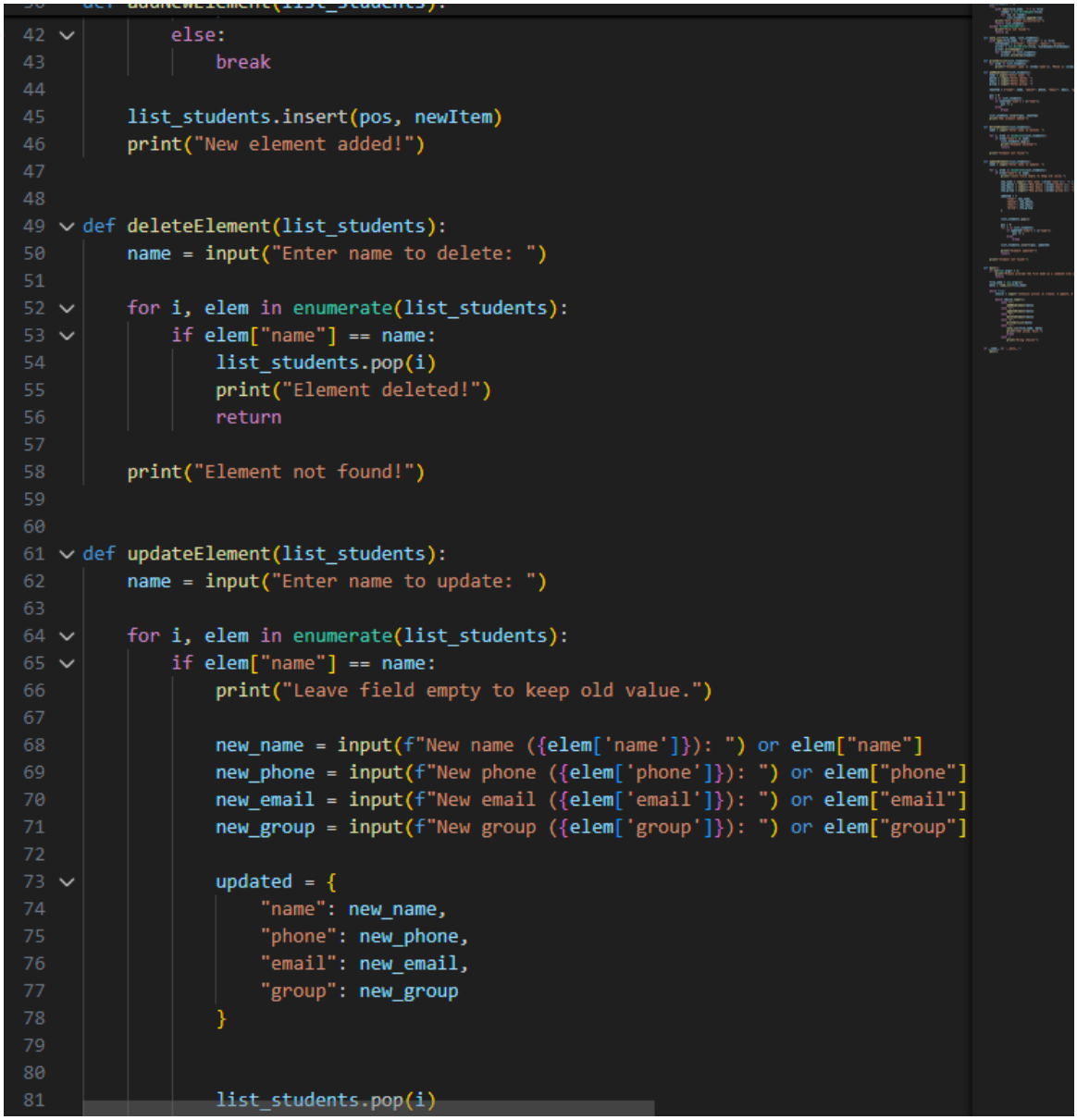
```
PS W:\TP-Danylo-Tykhonov-K6-242> pytest
========================================= test session starts =========================================
platform win32 -- Python 3.13.7, pytest-9.0.1, pluggy-1.6.0
rootdir: W:\TP-Danylo-Tykhonov-K6-242
collected 4 items

lab_02\test_lab2.py ....                                                                      [100%]

========================================== 4 passed in 0.38s ==========================================
PS W:\TP-Danylo-Tykhonov-K6-242>
```

lab_02.py U | test_lab2.py 5, U | lab.csv U | test_lab.csv U | task1.py pr2

lab_02 > test_lab2.py > TestLab2 > test_addNewElement

```python
import unittest
import os
from unittest.mock import patch
from lab_02 import *

class TestLab2(unittest.TestCase):
    def setUp(self):
        self.list_students = [
    {"name": "Bob", "phone": "0631234567", "email": "bob@gmail.com", "group": "CS
    {"name": "Emma", "phone": "0637457545", "email": "emma@gmail.com", "group": "
    {"name": "Michael", "phone": "0994747444", "email": "michael@gmail.com", "gro
    {"name": "Zak", "phone": "0635545467", "email": "zak@gmail.com", "group": "CE
]

    def test_addNewElement(self):
        with patch('builtins.input', side_effect=['Denys', '0673838838', 'denys@g
            addNewElement(self.list_students)
        self.assertEqual(self.list_students[1]["name"], "Denys")
        self.assertEqual(self.list_students[2]["name"], "Emma")


    def test_updateElement(self):
        with patch('builtins.input', side_effect=['Emma', 'William', '06757577575
            updateElement(self.list_students)
        self.assertEqual(self.list_students[2]["name"], "William")
        self.assertNotIn({"name": "Emma", "phone": "0637457545", "email": "emma@g


    def test_deleteElement(self):
        with patch('builtins.input', side_effect=['Michael']):
            deleteElement(self.list_students)
        self.assertNotIn({"name": "Michael", "phone": "0994747444", "email": "mic

    def test_save_csv(self):
        try:
            save_csv('test_lab2.csv', self.list_students)
            self.assertTrue(os.path.isfile('test_lab2.csv'))
            data = load_csv('test_lab2.csv')
            self.assertEqual(data, self.list_students)
        finally:
            pass
```

```python
import csv
import sys


def load_csv(file_name):
    list_students = []
    try:
        with open(file_name, 'r') as file:
            reader = csv.DictReader(file)
            for row in reader:
                list_students.append(row)
        print("Data loaded successfully!")
        return list_students
    except FileNotFoundError:
        print("File not found!")
        return []

def save_csv(file_name, list_students):
    with open(file_name, 'w', newline='') as file:
        fieldnames = ["name", "phone", "email", "group"]
        writer = csv.DictWriter(file, fieldnames=fieldnames)
        writer.writeheader()
        for student in list_students:
            writer.writerow(student)

def printAllList(list_students):
    for elem in list_students:
        print(f"Student name is {elem['name']}, Phone is {elem['phone']}, Email:

def addNewElement(list_students):
    name = input("Enter name: ")
    phone = input("Enter phone: ")
    email = input("Enter email: ")
    group = input("Enter group: ")

    newItem = {"name": name, "phone": phone, "email": email, "group": group}

    pos = 0
    for s in list_students:
        if newItem["name"] > s["name"]:
            pos += 1
```



```
        else:
            break

    list_students.insert(pos, newItem)
    print("New element added!")


def deleteElement(list_students):
    name = input("Enter name to delete: ")

    for i, elem in enumerate(list_students):
        if elem["name"] == name:
            list_students.pop(i)
            print("Element deleted!")
            return

    print("Element not found!")


def updateElement(list_students):
    name = input("Enter name to update: ")

    for i, elem in enumerate(list_students):
        if elem["name"] == name:
            print("Leave field empty to keep old value.")

            new_name = input(f"New name ({elem['name']}): ") or elem["name"]
            new_phone = input(f"New phone ({elem['phone']}): ") or elem["phone"]
            new_email = input(f"New email ({elem['email']}): ") or elem["email"]
            new_group = input(f"New group ({elem['group']}): ") or elem["group"]

            updated = {
                "name": new_name,
                "phone": new_phone,
                "email": new_email,
                "group": new_group
            }


            list_students.pop(i)
```

```
            pos = 0
            for s in list_students:
                if updated["name"] > s["name"]:
                    pos += 1
                else:
                    break

            list_students.insert(pos, updated)

            print("Element updated!")
            return

    print("Student not found!")


def main():
    if len(sys.argv) < 2:
        print("Please provide the file name as a command line argument!")
        return

    file_name = sys.argv[1]
    data = load_csv(file_name)

    while True:
        choice = input("\nChoose action [C create, U update, D delete, P print, )

        match choice.lower():
            case "c":
                addNewElement(data)
            case "u":
                updateElement(data)
            case "d":
                deleteElement(data)
            case "p":
                printAllList(data)
            case "x":
                save_csv(file_name, data)
                print("CSV saved. Exit.")
                break
            case _:
```

```
            case _:
                print("Wrong")


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