**Звіт про виконання практичних завдань до лекцій з курсу Технології програмування на мові Python**

Звіт до Лабораторної роботи №3

Об’єктно-орієнтовне програмування.

**Мета роботи:** Використовуючи теоретичне підґрунтя про ООП у мові Python переробити програму телефонного довідника студентів використовуючи принципи ООП для формування відомостей про студентів.

Під час виконання Лабораторної роботи №3 було надано варіанти рішення до наступних задач:

Переробити функціональність телефонного довідника студентів групи, що був розроблений у Лабораторній роботі №2 використовуючи принципи ООП:

1) розробити клас Студент групи з відповідними атрибутами;

2) розробити клас Список групи, має містити не словники, як виконано в лабораторній роботі №2, а об’єкти класу Студент групи; додавання нового запису, видаленні існуючого чи зміна даних має бути виконана через методи класу Список групи.

3) розробити клас для роботи з файлами для зчитування початкової інформації про список групи та збереження інформації по завершенню програми.

4) список студентів має містити не словники, як виконано в лабораторній роботі №2, а об’єкти класу Студент групи;

5) описання всіх класів мають міститися в окремих файлах, що мають відповідні імена(наприклад Studen, StudentList, Utils)

6) основний функціонал програми має бути покритий Юніт тестами.

**Клас Студент групи з відповідними атрибутами**

Для виконання цього завдання я створив модуль Student.py. Цей модуль оголошує клас Student (студент) з описом його атрибутiв та методiв.

Текст програми Student.py:

class Student:

    def \_\_init\_\_(self, name: str, phone: str, kurs: str, group: str):

        self.name = name

        self.phone = phone

        self.kurs = kurs

        self.group = group

    def \_\_str\_\_(self):

        return f"Name: {self.name}, Phone: {self.phone}, Course: {self.kurs}, Group: {self.group}"

    def update(self, name=None, phone=None, kurs=None, group=None):

        if name:

            self.name = name

        if phone:

            self.phone = phone

        if kurs:

            self.kurs = kurs

        if group:

            self.group = group

Метод \_\_init\_\_ викликається автоматично при створенні нового об’єкту класу Student (студента). Цьому методу передаються параметри ім’я студента, телефон та група, які стають атрибутами об’єкта студент.

Метод \_\_str\_\_ викликається автоматично при виводі на екран командою print об’кта класу Student (студента). Цьому методу передається параметр – об’єкт класу Student.

Метод update викликається при оновлені даних об’кта класу Student (студента). Цьому методу передаються параметри ім’я студента дані якого треба оновити а також нові значення його атрибутів нові ім’я, телефон, курс, група.

**Клас Список групи StudentList**

Для виконання цього завдання я створив модуль StudentList.py. У цьому модулі оголошується клас StudentList (список студентів) та зібрані методи обробки списку студентів, без використання імпорту/експорту списку із/в файл CSV (додавання, видалення, редагування студента та друк списку студентів).

Текст програми StudentList.py:

from Student import Student

class StudentList:

    def \_\_init\_\_(self):

        self.students = []

    def addNewElement(self, student: Student):

        self.students.append(student)

        print(f"Student {student.name} is appened.")

    def deleteElement(self, name: str):

        deletePosition = -1

        for student in self.students:

            if name == student.name:

                deletePosition = self.students.index(student)

                break

        if deletePosition == -1:

            print("Element was not found")

        else:

            print("Delete position " + str(deletePosition))

            del self.students[deletePosition]

        return

    def updateElement(self, student\_name: str, name1, phone1, kurs1, group1):

        for student in self.students:

            if student.name == student\_name:

                student.update(name1, phone1, kurs1, group1)

                print(f"Student data {student\_name} is updated.")

                return

        print(f"Student name: {student\_name} not found.")

    def printAllList(self):

        for student in self.students:

            print(student)

Цей код реалізує клас StudentList, який представляє список студентів. Кожен студент в цьому списку зберігається як об'єкт класу Student. Клас StudentList надає кілька методів для додавання, видалення, оновлення та відображення студентів у списку.

«from Student import Student» імпортує клас Student з модуля Student, що дозволяє використовувати його в класі StudentList.

class StudentList - це оголошення класу StudentList, який буде зберігати список студентів як об’єкти класу Student.

Метод \_\_init\_\_ - це конструктор класу, який викликається при створенні нового об'єкта класу.

self.students = [] - ініціалізує атрибут students як порожній список студентів. Цей список буде використовуватись для зберігання об’єктів класу Student (студентів).

Метод addNewElement - дозволяє додавати нового студента до списку. Параметр student - це об'єкт класу Student.

self.students.append(student) додає переданого студента в список students.

Останньою командою виводиться повідомлення, що студент було успішно доданий до списку.

Метод deleteElement() - цей метод видаляє студента зі списку за вказаним ім'ям.

Параметр name - ім'я студента, якого потрібно видалити.

Спочатку змінна, в якій буде збережено індекс елемента для видалення DeletePosition = -1

for student in self.students: - команда перебирає весь список студентів послідовно.

if name == student.name: - якщо ім’я студента для видалення співпало з іменем студента в списку – в DeletePosition зберігається його індекс і вихід з циклу по: break

Якщо DeletePosition ==-1 то студента з іменем name в списку не знайдено,

Інакше – повідомлення про видалення знайденого студента (вказується його індекс), і фактичне видалення його зі списку:

del self.students[deletePosition]

Метод updateElement() - цей метод дозволяє оновити дані студента за вказаним ім'ям.

Параметри: student\_name - це ім'я студента, дані якого потрібно оновити.

name1, phone1, kurs1, group1 – це атрибути студента (ім’я, телефон, курс, група), які будуть використовуватись для оновлення даних студента.

Цикл for student in self.students: перебирає всіх студентів у списку.

Якщо ім'я студента співпало з переданим, викликається метод update об'єкта Student для оновлення даних. (Метод update визначений в класі Student).

Після успішного оновлення, виводиться повідомлення про це.

Якщо студент не знайдений, виводиться відповідне повідомлення.

Метод printAllList() - цей метод виводить список всіх студентів на екран.

Якщо список студентів порожній (перевірка через if not self.students), виводиться повідомлення, що список порожній.

Якщо студенти є в списку, кожен студент виводиться через цикл for student in self.students:.

Для кожного студента викликається метод \_\_str\_\_() класу Student (який визначений в класі Student), щоб коректно відобразити інформацію про студента.

**Клас для роботи з файлами**

Для виконання цього завдання я створив модуль FileUtils.py. У цьому модулі оголошується клас FileUtils та знаходяться методи роботи з файлами CSV (загрузка списку та вигрузка списку в файл CSV)

Текст програми FileUtils.py:

import csv

from StudentList import StudentList

from Student import Student

class FileUtils:

    @staticmethod

    def loadDataFromCsv(file\_name: str, student\_list: StudentList):

        try:

            with open(file\_name, mode='r') as file:

                reader = csv.DictReader(file)

                for row in reader:

                    student = Student(

                        name=row['name'],

                        phone=row['phone'],

                        kurs=row['kurs'],

                        group=row['group']

                    )

                    student\_list.addNewElement(student)

                print(f"List students is loaded from file: {file\_name}.")

        except FileNotFoundError:

            print(f"File {file\_name} not found.")

        except Exception as e:

            print(f"Error reading file: {e}")

    @staticmethod

    def saveDataToCsv(file\_name: str, student\_list: StudentList):

        try:

            with open(file\_name, mode='w') as file:

                fieldnames = ["name", "phone", "kurs", "group"]

                writer = csv.DictWriter(file, fieldnames=fieldnames)

                writer.writeheader()

                for student in student\_list.students:

                    writer.writerow({

                        "name": student.name,

                        "phone": student.phone,

                        "kurs": student.kurs,

                        "group": student.group

                    })

                print(f"Students list is saved to file: {file\_name}.")

        except Exception as e:

            print(f"Error reading file: {e}")

Цей модуль містить два статичних методи класу FileUtils, які працюють із файлами у форматі CSV для завантаження та збереження списку студентів. Перші 3 команди імпортують бібліотеку CSV для роботи з файлами CSV, далі імпортується клас StudentList (список студентів) з модуля StudentList.py, третя команда імпортує клас Student (студент) із модуля Student. class FileUtils - це оголошення класу FileUtils.

Перший метод loadDataFromCSV Зчитує список студентів із CSV-файлу та додає їх до списку.

file\_name: str - параметр, який приймає шлях до CSV-файлу, з якого потрібно завантажити дані.

student\_list: StudentList - параметр, що представляє об'єкт класу StudentList, куди будуть додаватися об'єкти студентів.

Метод намагається відкрити файл для читання і завантажити в нього інформацію про студентів.

Використовується команда with для відкриття файлу. Це гарантує, що файл буде автоматично закритий після завершення роботи з ним цієї команди.

Параметр mode='r' означає, що файл відкривається для читання.

reader = csv.DictReader(file) - використовується клас csv.DictReader для читання файлу. Цей клас автоматично розпізнає перший рядок файлу як назви колонок і перетворює кожен рядок в словник, де ключі - це імена колонок.

Далі йде перебір кожного рядка файлу (який є словником з полями name, phone, kurs, group).

Для кожного рядка створюється новий об'єкт класу Student, використовуючи значення з відповідних колонок CSV-файлу.

Команда student\_list.addNewElement(student) додає створеного студента до списку студентів через метод addNewElement об'єкта student\_list.

Далі виводиться повідомлення про успішне завантаження даних з файлу.

Далі обробка помилок завантаження командою Except.

Другий метод saveDataToCSV()- зберігає список студентів в CSV

Команда with open() відкриває файл для запису (mode='w'). Якщо файл уже існує, він буде перезаписаний.

Команда fieldnames = ["name", "phone", "kurs", "group"] задає список назв колонок, які ми хочемо зберігати в CSV-файлі.

Створюється об'єкт writer для запису в CSV за допомогою csv.DictWriter, який дозволяє записувати дані у вигляді словників.

Команда writeheader() записує заголовки колонок у файл.

Цикл перебирає по одному список студентів student\_list.

Командою writerow для кожного студента із списку створюється словник з його атрибутами і записується цей словник як рядок у файл.

Виводиться повідомлення про успішне збереження даних у файл. Командою except Exception Обробляється будь-яка помилка, яка виникне під час запису в файл.

**Список студентів який містить не словники, а об’єкти класу Студент групи**

Для виконання цього завдання я створив модуль main\_3.py. Це головний модуль програми, в якому описано діалог користувача з програмою, та з якого викликаються всі інші модулі та методи вищеназваних класів.

Текст програми main\_3.py:

import StudentList

from Student import Student

import FileUtils

def main():

    student\_list = StudentList.StudentList()

    file\_name = "lab2.csv"

    FileUtils.FileUtils.loadDataFromCsv(file\_name, student\_list)

    while True:

        chouse = input("Please specify the action [ C create, U update, D delete, P print,  X exit ] ")

        match chouse:

            case "C" | "c":

                print("New element will be created:")

                name = input("Name Student: ")

                phone = input("Phone: ")

                kurs = input("Course: ")

                group = input("Group: ")

                student = Student(name, phone, kurs, group)

                student\_list.addNewElement(student)

                student\_list.printAllList()

            case "U" | "u":

                print("Existing element will be updated")

                name\_upd = input("Enter student Name, what need update: ")

                print("Enter new data (leave the field empty, if don't want change):")

                new\_name = input("New Name: ")

                phone = input("New Phone: ")

                kurs = input("New Course: ")

                group = input("New Group: ")

                student\_list.updateElement(student\_name=name\_upd, name1=new\_name, phone1=phone, kurs1=kurs, group1=group)

            case "D" | "d":

                name = input("Please enter name to be deleted: ")

                student\_list.deleteElement(name)

            case "P" | "p":

                print("List will be printed")

                student\_list.printAllList()

            case "X" | "x":

                FileUtils.FileUtils.saveDataToCsv("lab2.csv",student\_list)

                print("Exit")

                break

            case \_:

                print("Wrong chouse")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

У цьому завданні мені довелося внести зміни до програми main\_3.py – основної програми для управління списком студентів, щоб вона могла працювати із створеними мною класами. Ця програма забезпечує діалог з користувачем та основні функції обробки списку студентів (додавання, видалення, оновлення інформації про студентів, перегляд поточного списку студентів, а також завантажувати з CSV та зберігати зміни у CSV-файл)

Спочатку йде імпорт всіх модулів, необхідних для роботи main\_3.py

import StudentList - імпортується модуль StudentList, що містить клас StudentList, який дозволяє управляти списком студентів.

from Student import Student - імпортується клас Student, який містить інформацію про одного студента (ім'я, телефон, курс, група).

import FileUtils - імпортується модуль FileUtils, який містить методи для завантаження та збереження даних студентів у CSV-файл lab\_2.csv.

Основна функція main(). Ця функція містить основний цикл програми, який дозволяє користувачу вибирати операції для управління списком студентів.

Спочатку йде створення списку студентів

student\_list = StudentList.StudentList() - створюється екземпляр класу StudentList, який ініціалізує порожній список студентів.

file\_name = "lab2.csv" - визначається ім'я файлу, з якого буде завантажено або в який буде збережено список студентів.

Далі за допомогою методу loadDataFromCSV класу FileUtils, програма намагається завантажити список студентів з файлу lab2.csv у об'єкт student\_list.

Далі йде основний цикл програми. Цей цикл працює нескінченно, поки користувач не вибере опцію X для збереження даних і виходу з програми. Програма виводить список доступних операцій для користувача.

Якщо користувач вибрав опцію C (create), програма запитує ім'я, телефон, курс і групу студента. Створюється новий об'єкт класу Student із введеними даними. Цей об'єкт додається до списку студентів через метод addNewElement() класу Student\_List Увесь список виводиться на екран методом printAllList() класу Student\_List.

Якщо користувач вибрав опцію D (delete), програма запитує ім'я студента, якого потрібно видалити. Викликається метод deleteElement() классу student\_list для видалення студента зі списку.

Якщо користувач вибрав опцію U(update), програма запитує ім'я студента name\_upd, якого потрібно оновити, та нові дані (ім'я new\_name, телефон, курс, група). Викликається метод updateElement, класу student\_list який оновлює дані студента в списку. Якщо якийсь з параметрів не введений, програма залишить старі значення для цих полів.

Якщо користувач вибрав опцію P(print), програма виводить поточний список студентів на екран, викликаючи метод printAllList класу student\_list.

Якщо користувач вибрав опцію X(exit), програма викликає метод saveDataToCsv() класу FileUtils для збереження поточного списку студентів у файл lab2.csv.

Далі програма виводить повідомлення про успішне збереження даних і завершує цикл (і тим самим програму) через break.

Якщо користувач вводить некоректний варіант(будь-яка буква крім c,d,u,p,x), програма виводить повідомлення про невірний вибір і пропонує зробити новий вибір.

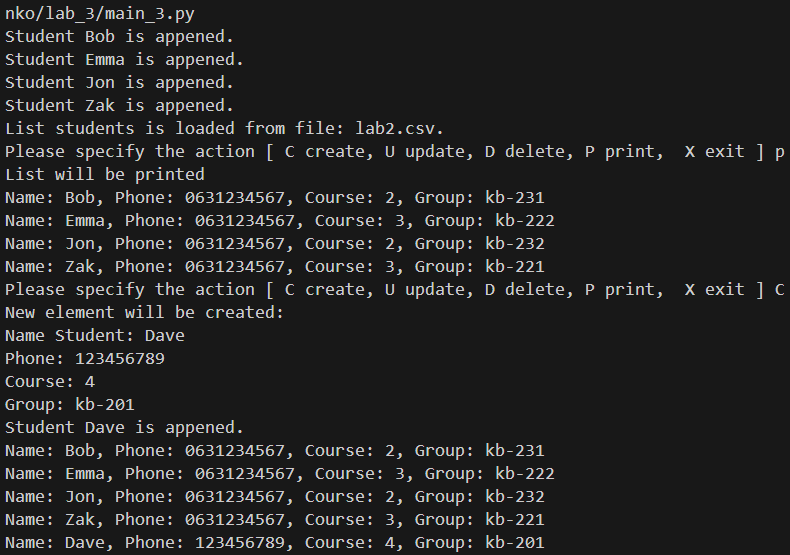
У кінці йде блок:

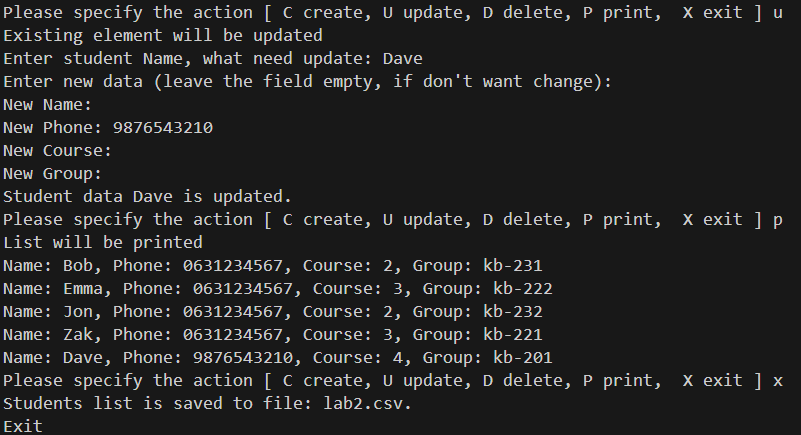
if \_\_name\_\_ == "\_\_main\_\_":

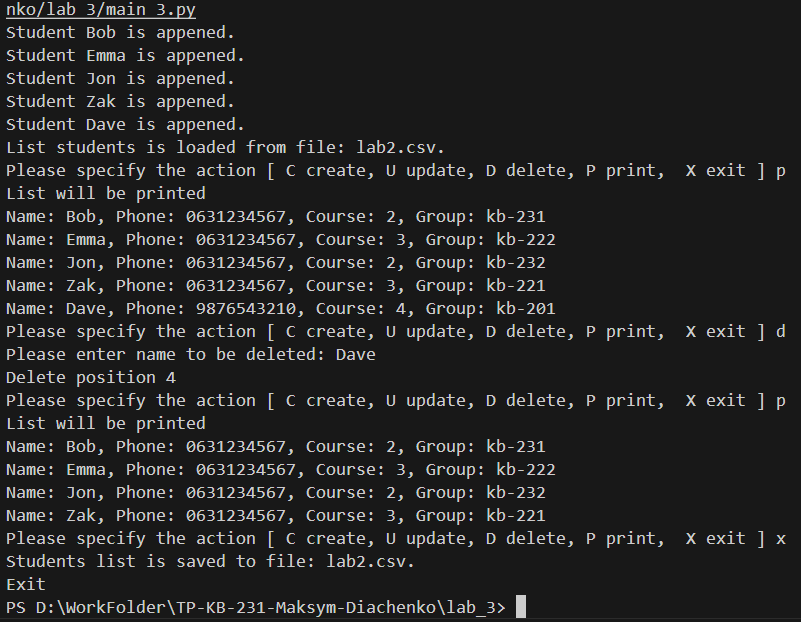
main()

Цей блок гарантує, що програма запуститься тільки при безпосередньому виконанні скрипту, а не при його імпорті як модуль.

Результат роботи основної програми:







**Юніт-тести основного функціоналу програми**

Для виконання цього завдання я створив модулі StudentList.py та FileUtils.py. Для створення юніт-тестів для основного функціоналу програмного забезпечення достатньо зробити два тести файлів StudentList.py і FileUtils.py, адже саме в них знаходяться основні модулі/методи обробки списку студентів. Тестування класу Student модуля Student.py буде виконано автоматично, так як атрибути та методи цього класу використовуються практично в усіх функціях програмного забезпечення.

Основні тести які варто виконати для StudentList.py – це Додавання студента; Видалення студента (існуючого та неіснуючого); Оновлення даних студента (існуючого та неіснуючого).

Основні тести які варто виконати для FileUtils.py – це Завантаження студентів з файлу CSV; Збереження списку студентів у файл CSV; Перевірка обробки помилок (наприклад, файл не знайдено).

Для юніт-тестів модуля StudentList.py я створив модуль Test\_StudentList.py

Текст програми Test\_StudentList.py:

import unittest

from StudentList import StudentList

from Student import Student

class TestStudentList(unittest.TestCase):

    def setUp(self):

        self.student\_list = StudentList()

        self.student = Student(name="Ivan", phone="123456789", kurs="1", group="kb-241")

        self.student\_list.addNewElement(self.student)

    def test\_add\_student(self):

        self.assertEqual(len(self.student\_list.students), 1)

        self.assertEqual(self.student\_list.students[0].name, "Ivan")

    def test\_remove\_student(self):

        self.student\_list.deleteElement("Ivan")

        self.assertEqual(len(self.student\_list.students), 0)

    def test\_remove\_student\_not\_found(self):

        self.student\_list.deleteElement("Zahar")

        self.assertEqual(len(self.student\_list.students), 1)

    def test\_update\_student(self):

        self.student\_list.updateElement("Ivan", name1="Ivan1", phone1="111111", kurs1="2", group1="kb-232")

        updated\_student = self.student\_list.students[0]

        self.assertEqual(updated\_student.name, "Ivan1")

        self.assertEqual(updated\_student.phone, "111111")

    def test\_update\_student\_not\_found(self):

        self.student\_list.updateElement("Zahar", name1="Zahar1", phone1=None, kurs1=None, group1=None)

        self.assertEqual(len(self.student\_list.students), 1)

        self.assertEqual(self.student\_list.students[0].name, "Ivan")

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

Спочатку йде імпорт необхідних модулів:

import unittest – імпортується стандартний модуль Python для написання юніт-тестів.

from StudentList import StudentList - імпорт класу StudentList списку студентів з модуля StudentList.py, який ми тестуємо.

from Student import Student - імпорт класу Student (студента) з модуля Student.py, який який використовується при створенні об’єкта Студент.

Далі створюється клас TestStudentList, який успадковує unittest.TestCase. Це дозволяє використовувати методи бібліотеки unittest для створення та виконання тестів.

Метод setup - це спеціальний метод, який викликається перед виконанням кожного тесту. Він створює 1 екземпляр класу Student(name="Ivan", phone="123456789", kurs="1", group="kb-241") із заданими атрибутами.

У цьому методі створюється:

Порожній список студентів self.student\_list.

Об'єкт студента self.student з атрибутами:

Ім'я: Ivan

Телефон: 123456789

Курс: 1

Група: kb-241

Студент додається до списку для перевірки методів класу StudentList.

Таким чином перед кожним тестом початковий список студентів StudentList складається з 1 студента, створеного методом setUp()

Далі йде тест додавання студента до списку. self.assertEqual - це метод, який перевіряє, чи два значення рівні.

Цим методом виконуються 2 перевірки:

Чи список студентів містить 1 елемент після додавання студента “Ivan”.

Чи ім'я першого студента у списку відповідає очікуваному (“Ivan”).

Саме додавання студента в цьому методі не виконується, т.я. студента автоматично додано при виконанні методу setUp() і виклику функції addNewElement() перед початком роботи методу test\_add\_student(). Тобто фактично в методі test\_add\_student() тестується робота функції addNewElement() класу Studentlist()

Далі йде тест видалення студента. Викликається метод deleteElement(), класу StudentList(), який видаляє студента з ім'ям “Ivan”. Виконується перевірка: після видалення студента список має стати порожнім.

Далі йде тест видалення студента, якого немає в списку. Викликається метод deleteElement() класу StudentList() для студента, якого немає у списку (“Zahar”). Перевірка: список не змінюється (довжина залишається =1).

Далі йде тест оновлення даних студента, який є в списку. Метод update\_student оновлює дані існуючого студента:

Нове ім'я: Ivan1

Новий телефон: 111111

Новий курс: 2

Нова група: kb-232

Перевірка: чи оновлені атрибути студента у списку (name = “Ivan”, phone = “111111”).

Далі йде тест оновлення студента, якого немає в списку. Викликається метод update\_student для неіснуючого студента (“Zahar”). Перевірки: список студентів залишився складом з 1 студента, тобто при оновленні студента, якого немає в списку, новий студент не доданий до списку. Ім’я єдиного студента в списку залишилось “Ivan”

Далі йде виконання тестів

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

Запускає тестування, якщо файл виконується напряму. Модуль unittest автоматично знайде всі методи, що починаються з test\_.

Для тестування роботи з CSV-файлами я створив модуль Test\_FileUtils.py

Текст програми Test\_ FileUtils.py:

import unittest

import os

from FileUtils import FileUtils

from StudentList import StudentList

from Student import Student

class TestFileUtils(unittest.TestCase):

    def setUp(self):

        self.file\_name = "test\_1.csv"

        self.student\_list = StudentList()

        self.student = Student(name="Ivan", phone="123456789", kurs="1", group="KB-241")

        self.student\_list.addNewElement(self.student)

    def test\_saveDataToCsv(self):

        FileUtils.saveDataToCsv(self.file\_name, self.student\_list)

        self.assertTrue(os.path.exists(self.file\_name))

    def test\_loadDataFromCsv(self):

        FileUtils.loadDataFromCsv(self.file\_name, self.student\_list)

        new\_list = StudentList()

        FileUtils.loadDataFromCsv(self.file\_name, new\_list)

        self.assertEqual(len(new\_list.students), 1)

        self.assertEqual(new\_list.students[0].name, "Ivan")

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

Спочатку йде імпорт необхідних модулів

unittest — стандартна бібліотека Python для створення та виконання тестів.

os — використовується для роботи з файловою системою (наприклад, перевірка існування заданих тестових файлів).

FileUtils — модуль, який тестується (клас для роботи з файлами CSV).

StudentList та Student — потрібні для створення об'єктів, які зберігаються у файлі CSV.

Далі йде створення класу для тестів. Клас TestFileUtils успадковує unittest.TestCase, що дозволяє створювати тестові методи з перевіркою функціоналу.

Далі йде метод setup. Цей метод виконується перед кожним тестом. Він створює спільні об'єкти для тестів, щоб уникнути дублювання коду. Наприклад, кожен тест отримує готовий об'єкт StudentList із одним студентом.

У ньому створюються:

Тестовий файл self.file\_name, який буде використовуватися у тестах.

Об'єкт self.student\_list, який містить список студентів.

Студент self.student – екземпляр класу Student() додається до списку student\_list.

Далі йде тест збереження студентів у файл. Викликається метод saveDataToCsv(), щоб зберегти список студентів у файл self.file\_name.

self.assertTrue(os.path.exists(self.file\_name)) перевіряє, чи існує файл file\_name. Якщо файл є, тест методу пройдено успішно.

Далі йде тест завантаження студентів з файлу. Спочатку створюється пустий об’єкт класу StudentList() – порожній список студентів з іменем new\_list. Завантажуються дані з файлу у new\_list за допомогою методу loadDataFromCsv() класу FileUtils(). Перевірки: у списку new\_list має бути один студент; чи ім'я цього студента відповідає очікуваному ("Ivan").

Далі йде виконання тестів

**Висновки.** Під час виконання цієї лабораторної роботи я навчився використовувати засоби об’єктно-орієнтованого програмування для написання програм. Я переробив програми для обробки списку студентів, використовуючи методи об’єктно-орієнтованого програмування. Також я навчився створювати юніт-тести для модулів які використовують класи, об’єкти класів, їх атрибути та методи.

Посилання на github:

<https://github.com/diachenkom/TP-KB-231-Maksym-Diachenko>

Знімок екрану з посилання на github:

