**Лабораторна робота №3**

Завдання: 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 класах, що мають відповідні імена(наприклад Student, StudentList, Utils).
6. основний функціонал програми має бути покритий Юніт тестами.

Виконання завдання:

1. Розробити клас Студент групи.

Створює клас Student і визначаємо атрибути в ньому, яким буде присвоюватися name, phone, email, group, загалом дані студента. Код цього класу можна побачити на Рис. 2. вверху.

1. Розробити клас Список групи.

Створюємо клас StudentList і далі в ньому визначаємо атрибут list\_students, якому присвоюємо пустий список, який ми наповнимо потім даними. Перенесемо у цей клас функції з попередньої лабораторної роботи addNewElement, deleteElement, updateElement, printAllList. Переробимо функціонал так, що, наприклад, додавати елемент не словаром у масив, а об’єктом класу. Тобто, ми тепер викликаємо клас Student і передаємо аргументи, введені користувачем дані про студента. Переробимо далі логіку вставки об’єкта, щоб ми перебирали не ключу словаря, а по атрибуту класу. Щодо інших функцій, то дії аналогічні, там, де ми шукали, перебирали, отримували дані за відповідними ключами словарів, тепер це буде за відповідними атрибутами класів. Код щодо цього завдання на Рис.2 та Рис. 3.

1. Розробити клас для роботи з файлами для зчитування початкової інформації про список групи та збереження інформації по завершенню програми.

Клас назвемо Utils. У нього переносимо з попередньої

лабораторної роботи дві функції save\_csv() та load\_csv(). І також переписуємо під взаємодію з класами замість взаємодії з словарами. У функції load\_csv() cписок студентів тепер створюється шляхом створення екземпляру класу StudentList. І далі вже додаємо елемента в список, який є атрибутом в класі StudentList, а елемент створюється шляхом створення об’єкту класу Student і передачею відповідних зчитаних аргументів по атрибутам(name, phone, email, group). У функції save\_csv() переписуємо перебір значень по ключам на перебір значень по атрибутам. І далі записуються дані в файл. save\_csv() та load\_csv() визначаємо як статичні методи за допомогою @staticmethod для того, щоб викликати ці дві функції(методи) без створення екземпляру класу Utils, це виключно для зручності виклику методів з класу. Переписування функціоналу завершено. Не забудемо змінити виклики функцій в main(), щоб все працювало. Код цього завдання зображено на Рис. 3. та Рис. 4.

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

Ця вимога є виконаною. Я перейшов з словників на

об’єкти класу.

1. описання всіх класів мають міститися в окремих класах, що мають відповідні імена(наприклад Student, StudentList, Utils).

Ця вимога є виконаною. Я розділив функціонал, тепер

функції, що виконують додавання, оновлення, видалення, вивід записів, то вони знаходяться в StudentList. Атрибути студента в класі Student. А функції, що відповідають за збереження та завантаження даних у класі Utils.

1. основний функціонал програми має бути покритий Юніт-тестами.

Імпортуємо необхідні бібліотеки для юніт-тестів. Оголошуємо клас TestLab3 і аргумент, який він приймає unittest.TestCase. Далі в ньому оголошуємо функцію setUp, у ній створюємо екземпляр класу StudentList(), це буде нашим списком, призначаємо це створення в змінну self.group\_list. Далі створюємо змінну self.test\_data і присвоюємо масив словарів із значеннями, і тут є примітка, що це просто набір даних, які ми хочемо додати в масив різними функціями відповідних класів. Потім прописуємо тести для функції addNewElement() у класі StudentList у функції test\_addNewElement(). Робимо цикл, який перебирає значення self.test\_data і далі передає як side\_effect у patch, у якому викликає через клас функцію(метод) addNewElement(). Далі за такою конструкцією self.assertEqual(self.group\_list.list\_students[2].name, 'Mykola') перевіряємо, що елемент дійсно з’явився в класі StudentList у атрибуті list\_students. Аналогічним чином і інші елементи. Вміст цієї тест-функції скопіюємо в інші тест-функції(deleteElement, updateElement), це через те, що після тесту в нас масив об’єктів буде пустий, і щоб він не був пустий, то ми додаємо елементи, а потім тестуємо на видалення чи оновлення у відповідності, як нам потрібно. Далі тестуємо на оновлення запису за допомогою такого коду:  
with patch('builtins.input', side\_effect=['Anna', 'William', '06712345678', 'william@gmail.com', 'EM-192']): self.group\_list.updateElement() self.assertEqual(self.group\_list.list\_students[2].name, "William") self.assertNotIn("Anna", [student.name for student in self.group\_list.list\_students]).  
Ми передали значення(які типу увів користувач) у patch, далі викликали метод оновлення записів(updateElement) з класу. І далі за допомогою assertEqual() перевіряємо на яку позицію став оновлений елемент з новим ім’ям та за допомогою assertNotIn() ми перевіряємо, чи не залишилася Анна в списку, успіх, якщо її в масиві немає. Потім пишемо тест для методу deleteElement(), тут нічого складного, знову конструкція with patch, side\_effect присвоюємо ім’я Микола(саме його хочемо видалити). І далі в цій конструкції викликаємо метод deleteElement() з класу. І поза цією конструкцією перевіряємо, чи не залишився Микола в списку, аналогічно як з Анною в попередній тест-функції. Далі напишемо тест-функцію test\_save\_csv(), у ній ми протестуємо методи класу Utils(save\_csv та load\_csv). Відразу напишемо конструкцію try… finally… finally залишим пустим, написавши pass. У блоку try оголошуємо дві змінні load\_list\_students та save\_list\_students присвоюємо їм виклик із класу методу завантаження даних з файлу та виклик із класу методу збереження даних у файл відповідно. За допомогою конструкції self.assertTrue(os.path.isfile('test\_lab3.csv')) перевіряємо, чи було збережено файл. І далі перевіряємо за допомогою конструкції self.assertEqual(load\_list\_students, save\_list\_students), чи співпадають завантажені дані зі збереженими даними. Ось і все. Настав час запустити тести і перевірити, що все правильно працює.

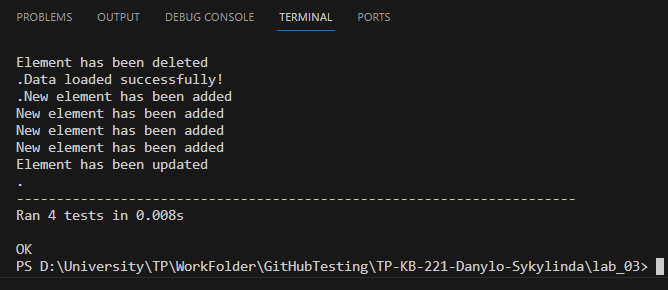


Рис. 1. Показано, що тести проходять успішно.

Як бачимо, тести проходять успішно, а отже зроблено правильно. Тепер лабораторна робота виконана.

Код завдання:

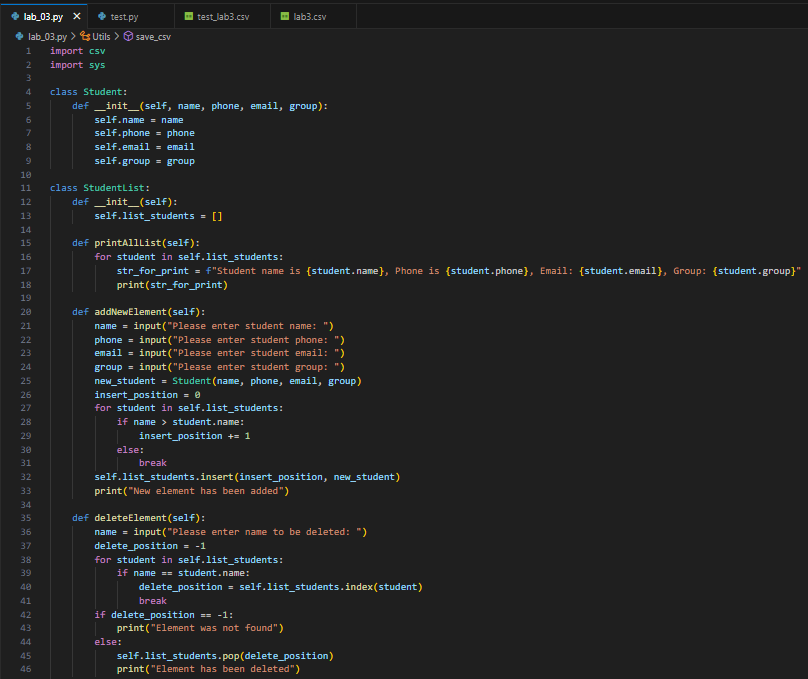


Рис. 2. Код файлу lab\_03.py.

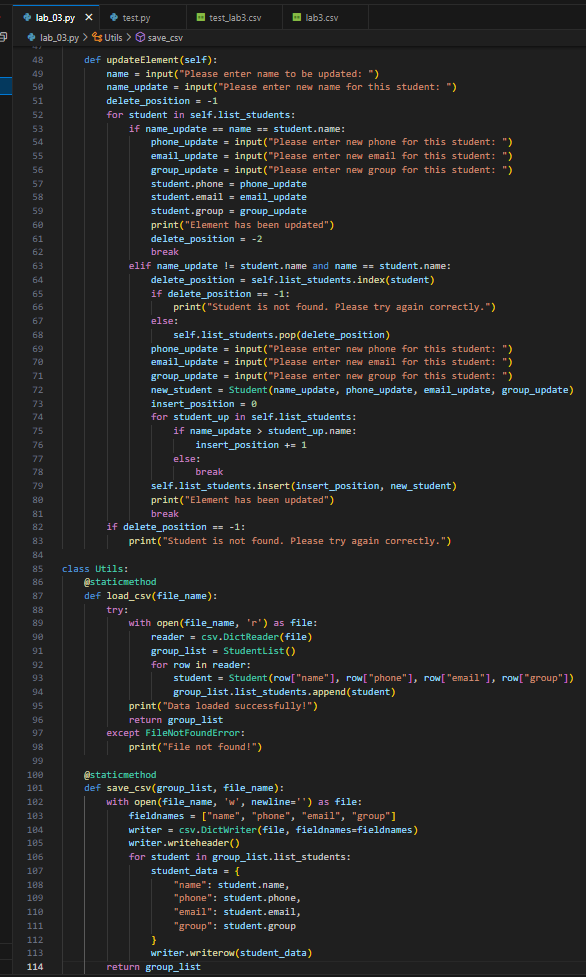


Рис. 3. Продовження коду файлу lab\_03.py.

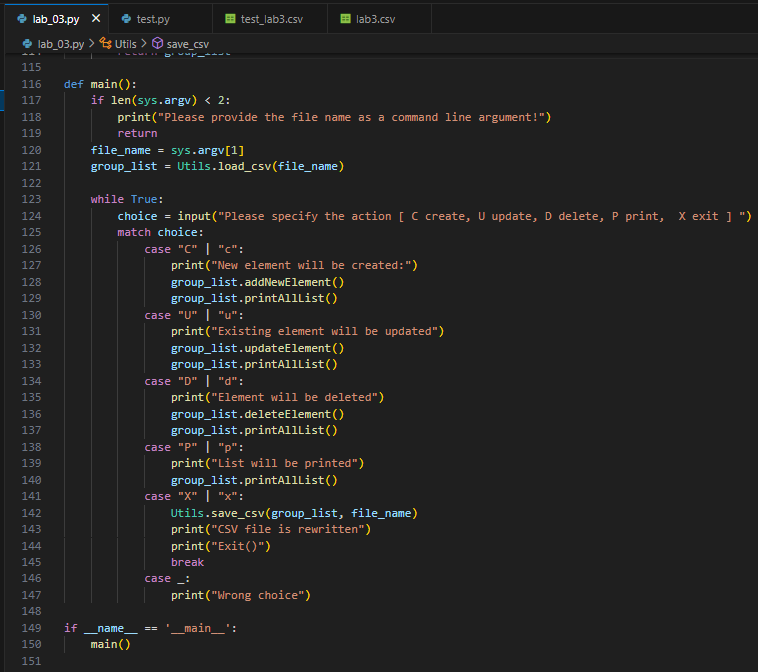


Рис. 4. Продовження коду файлу lab\_03.py.

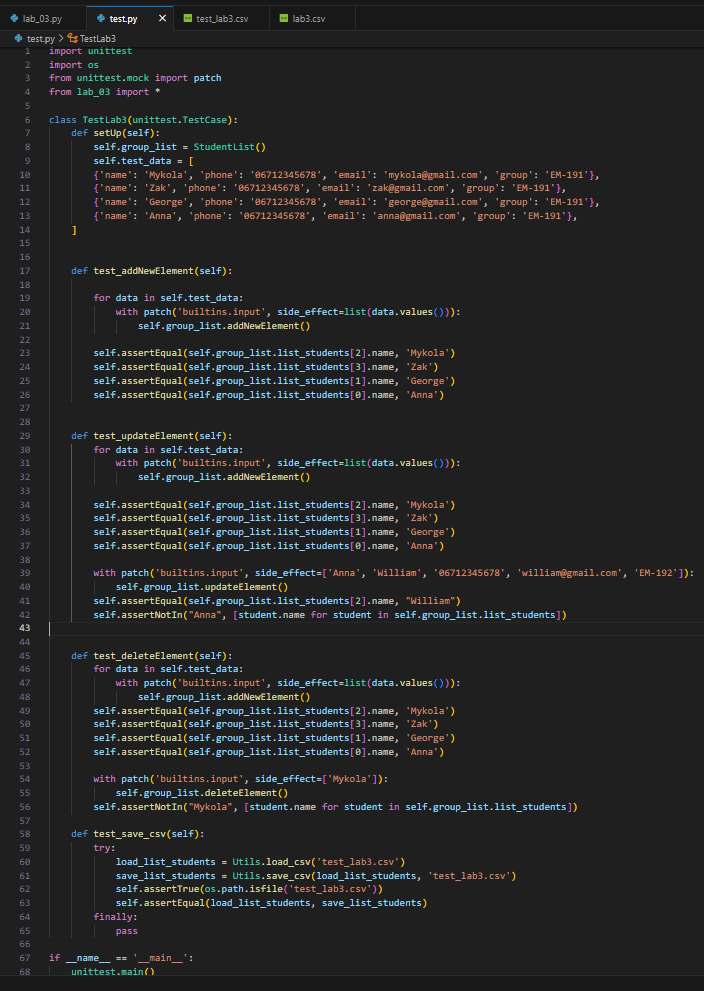


Рис. 5. Код юніт-тестів для коду з файлу lab\_03.py.

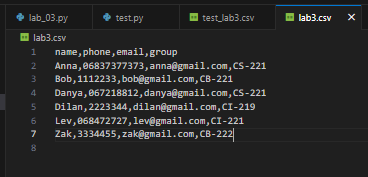


Рис. 6. Дані у файлі lab3.csv для роботи коду lab\_03.py.

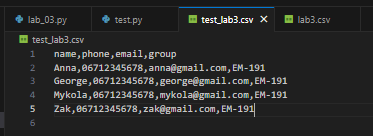


Рис. 7. Дані у файлі test\_lab3.csv для роботи коду юніт-тестів test.py.

GitHub:

Посилання на GitHub на цю папку з файлами лабораторної роботи: <https://github.com/danylosykylinda/TP-KB-221-Danylo-Sykylinda/tree/main/lab_03>.

Скріншот коду з GitHub:

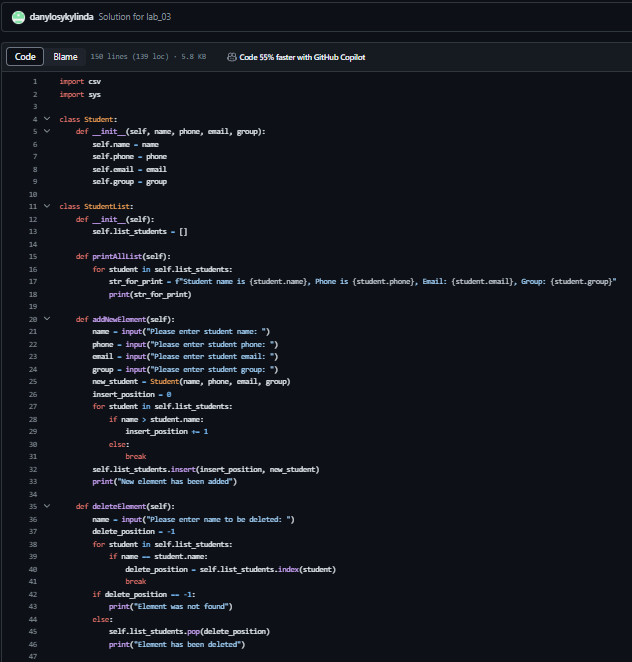


Рис. 8. Код файлу lab\_03.py.



Рис. 9. Продовження коду файлу lab\_03.py.

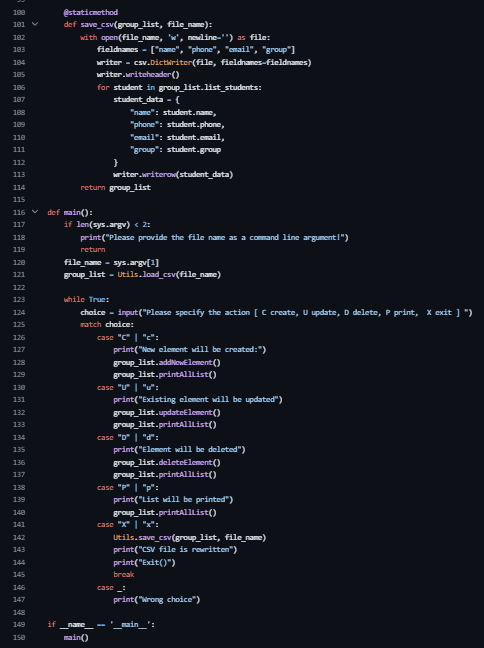


Рис. 10. Продовження коду файлу lab\_03.py.



Рис. 11. Код файлу test.py для юніт-тестів для коду файлу lab\_03.py.

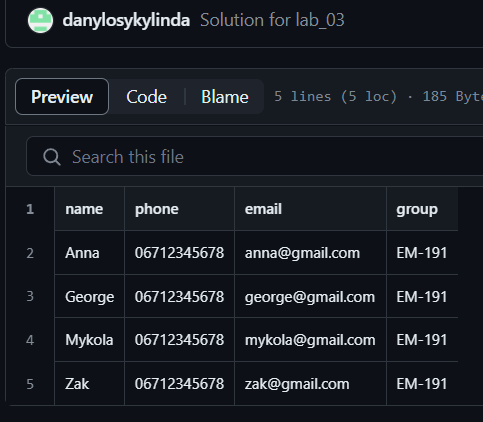


Рис. 12. Файл test\_lab3.csv з початковими даними для юніт-тестів.

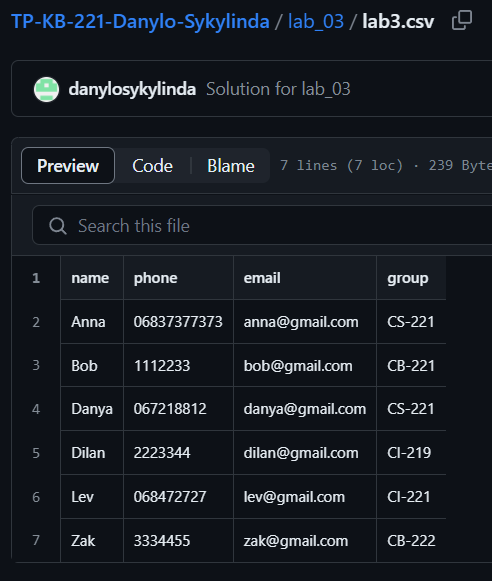


Рис. 13. Файл lab3.csv з початковими даними для роботи коду lab\_03.py.