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

**Лабораторна робота №2**

**Тема: 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.

**Суть завдання:**

1. **завантаження** початкових данних для довідника з csv файлу. Приклад csv файлу (lab2.csv), та оригінальній файл формату xlsx розміщені поряд з вказівками. Ім’я файлу для завантаження даних повинно передаватись в якості параметрів командного рядку.
2. **зберігання** довідника студентів у форматі CSV перед виходом із програми.
3. покриття основного функціоналу програми **Юніт тестами**.

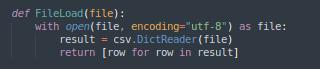
**Теоретичні відомості:** Параметри командного рядка є механізмом введення даних в програму через консольне викликання. У Python це реалізується за допомогою модуля sys та його об'єкта argv. Ці параметри передаються при запуску програми та можуть використовуватися для налаштувань або визначення вхідних даних. Після підключення модуля sys, аргументи командного рядка стають доступними через список argv, де нульовий елемент містить ім'я програми, а подальші - передані параметри. Робота з файлами в Python реалізується через вбудовані функції, такі як open(), яка служить для відкриття файлів з можливістю вказання режиму доступу (читання, запис, тощо). Файл потрібно закривати після використання, але можна скористатися ключовим словом with, щоб автоматично закрити файл після виконання блоку коду. Для роботи з файлами у форматі CSV використовується модуль csv, який надає функціонал для зручного читання та запису даних в цьому форматі. Наприклад, DictReader може використовуватися для читання CSV-файлу у вигляді словників. Юніт-тестування є практикою в розробці програмного забезпечення, спрямованою на перевірку окремих блоків коду на коректність та відповідність специфікаціям. У Python для юніт-тестування можна використовувати різні бібліотеки, одна з яких - pytest. Тести служать для забезпечення правильної роботи окремих функціональних одиниць програми та можуть бути використані для автоматичного перевірки коду під час розробки.

**Хід роботи:**

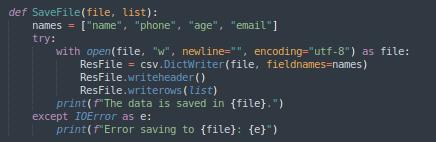
1. Я скопіював код із минулої лабораторної роботи та імпортував бібліотеку CSV для роботи з файлами у CSV форматі.



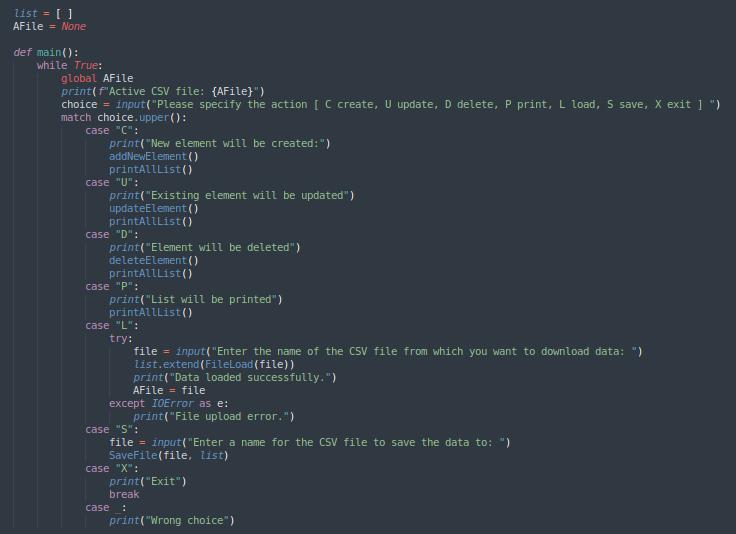
2. Я написав функцію FileLoad, яка призначена для читання даних з CSV-файлу, в неї передається значення file, в якому міститься назва фалу для читання. В середині цієї функції спочатку відбувається відкриття файлу у режимі читання з вказанням кодування ‘utf-8’ і після чого виконується csv.DictReader для читання файлу і повернення словника для кожного рядка, після чого відбувається повернення списку словників, де кожен словник відповідає одному рядку з прочитаного файлу.



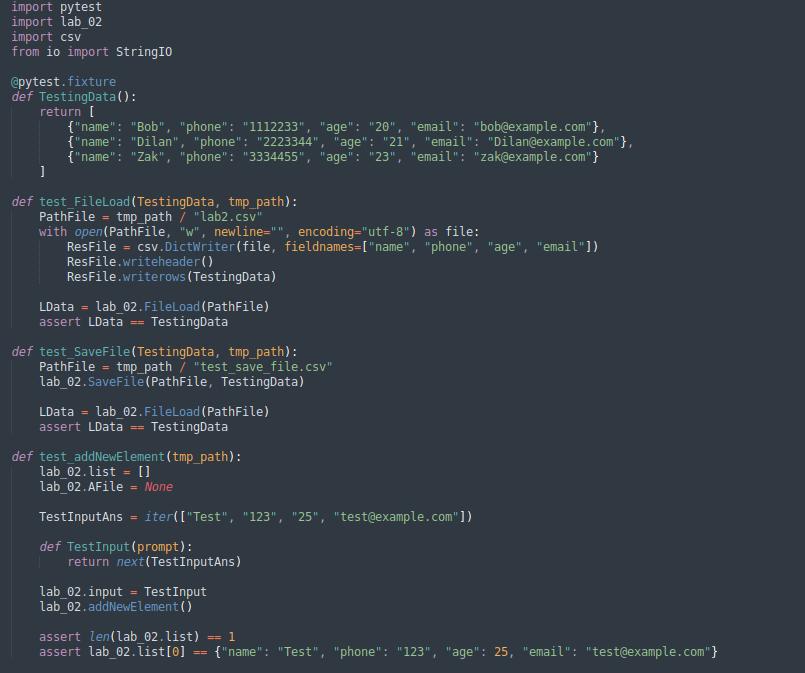
3. Далі я написав функцію SaveFile, яка призначена для запису даних у CSV-файл, в неї я передаю значення file та list, file - це назва файлу, в який треба записати дані, а list - список словників, що містять дані для запису. Спочатку ця функція відкриває файл у режимі запису із кодуванням ‘utf-8’, або створює його, якщо він не існує, після чого використовується csv.DictWriter для створення об’єкта запису CSV із вказанням назв полів, після чого викликається writeheader для запису заголовку CSV файлу. Потім використовується writerows для запису рядків зі списку в файл. Також я додав перевірку помилок і якщо не вдається зберети файл, то виводиться повідомлення про помилку збереження файлу.

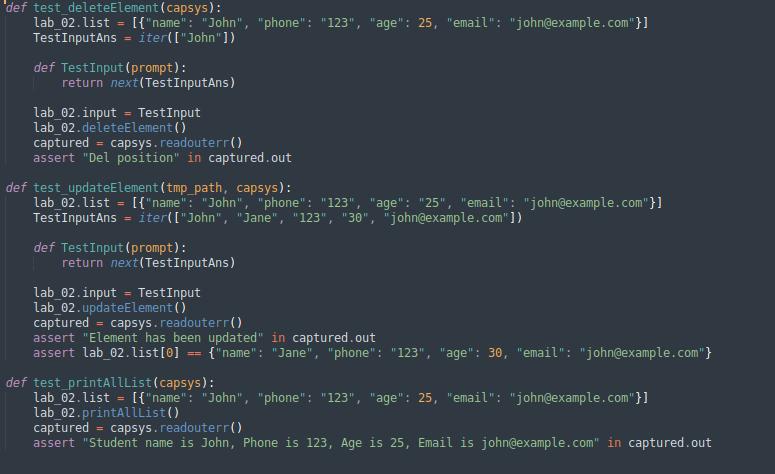


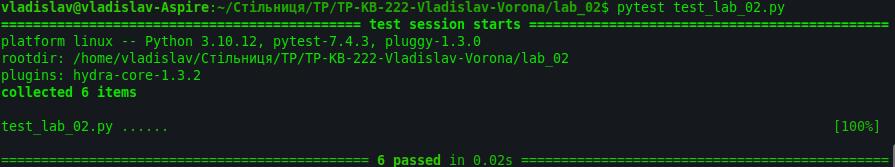
4. Я відредагував функцію main, додавши до неї інформацію про активний CSV файл та можливість вибору збереження та завантаження CSV файлів.



5. Далі я виконав Unit-тести, для цього я створив окремий файл під назвою test\_lab\_02.py і почав там писати код для тестування. Спочатку я імпортував всі потрібні модулі та сам файл lab\_02. Потім я створив фікстуру TestingData, в середині якої знаходяться тестові дані у форматі списків. Далі я створив функціюtest\_FileLoad, яка використовує фіксуру TestingData та tmp\_path, яка є іншою вбудованою фікстурою pytest для створення тимчасових файлів. Ця функція створює тимчасовий CSV-файл та записує в його дані з TestingData, після чого викликає функцію FileLoad із модуля lab\_02, який вже завантажує дані за файлу, після чого відбувається перевірка, чи завантажені дані співпадають з TestinData. Далі я написав функцію test\_SaveFile, яка схоже на функцію test\_FileLoad, але перевіряє роботу функції SaveFile, вона зберігає тестові дані за допомгою SaveFile в тимчасовий файл, потім завантажує і перевіряє, чи дані співпадають. Потім я написав функцію test\_addNewElement, в якій відбуваєтсья тестування функції addNewElement із lab\_02, тут модифікується вбудована функція input, щоб при виклику поверталась певна послідовність значень, далі викликається функція addNewElement, яка вводить новий елемент в список з викорситанням вказаних введених даних і після чого відбувається перевірка списку на наявність нового елементу. Далі я написав функції test\_deleteElement та test\_updateElement, які схожі із функцією addNewElement, тільки тестуються інші функції і відбувається трішки інша перевірка списку. Далі я перевіряю функцію test\_printAllList, яка тестує функцію printAllList, вона змінює список lab\_02.list і викликає цю функцію, після чого виведені дані перевіряються на вірність.







**Висновок:** під час виконання цієї лабораторної роботи я розширив функціонал програми довідника студентів, використовуючи теоретичні відомості з роботи з файлами та тестування коду у мові Python. Використовуючи модуль csv, я реалізував завантаження даних для довідника з csv файлу та збереження його у форматі CSV . Окрім цього, я використав бібліотеку pytest для написання юніт-тестів, які перевіряють коректність роботи основних функцій програми. Тести дозволили переконатися в правильності роботи функцій, таких як завантаження та збереження файлів, а також додавання, видалення та оновлення елементів у довіднику. Під час виконання тестів, я використовував фіксовані дані та тимчасові файли для перевірки різних сценаріїв використання програми, цей підхід дозволяє забезпечити стабільність та надійність програмного коду. Я отримав практичні навички роботи з файлами в мові програмування Python, а також здобув вміння використовувати юніт-тести для перевірки функціональності коду.