**Лабораторна робота №3**

**Тема:** ООП.

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

**Завдання до лабораторної роботи**

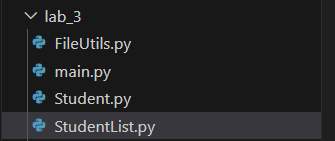
Переробити функціональність телефонного довідника студентів групи, що був розроблений у Лабораторній роботі №2 використовуючи принципи ООП:

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.

**Хід роботи**

Ознайомившись з теоретичними відомостями та завданням розпочав виконувати лабораторну роботу.

Створив нові файли: FileUtils, main, Student, StudentList

****

Переписав довідник з використанням принципів ООП.

Файл **Student**: клас для представлення студента:

|  |
| --- |
| class Student:      def \_\_init\_\_(self, name, phone, email, address):          self.name = name          self.phone = phone          self.email = email          self.address = address      def update(self, name=None, phone=None, email=None, address=None):          if name:              self.name = name          if phone:              self.phone = phone          if email:              self.email = email          if address:              self.address = address      def \_\_str\_\_(self):          return f"Student name: {self.name}, Phone: {self.phone}, Email: {self.email}, Address: {self.address}" |

Файл **StudentList**: клас для управління списком студентів групи:

|  |
| --- |
| from Student import Student  class StudentList:      def \_\_init\_\_(self):          self.students = []      def add\_student(self, student):          self.students.append(student)          self.students.sort(key=lambda x: x.name)        def delete\_student(self, name):          for student in self.students:              if student.name == name:                  self.students.remove(student)                  print(f"Student {name} has been deleted.")                  return          print("Student not found.")      def update\_student(self, name, new\_name=None, new\_phone=None, new\_email=None, new\_address=None):          for student in self.students:              if student.name == name:                  student.update(new\_name, new\_phone, new\_email, new\_address)                  print(f"Student {name}'s details have been updated.")                  return          print("Student not found.")      def print\_all(self):          if not self.students:              print("The student list is empty.")          else:              for student in self.students:                  print(student)      def find\_student(self, name):          for student in self.students:              if student.name == name:                  return student          return None |

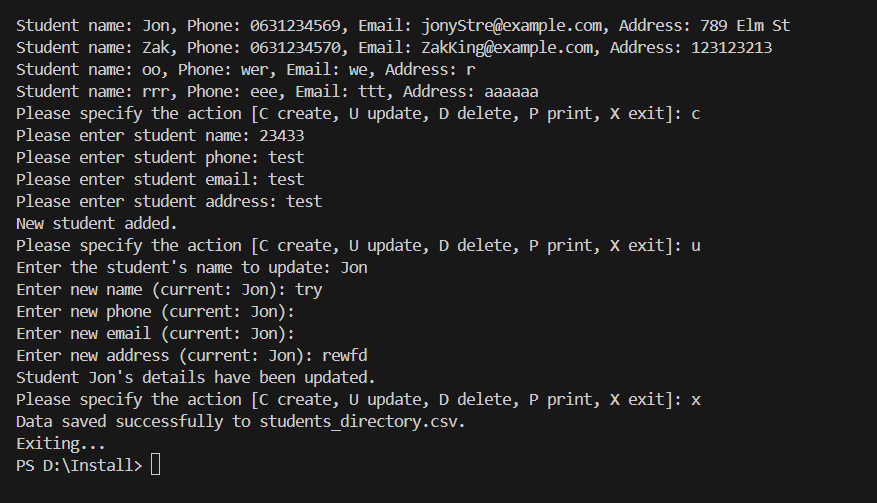
Файл **FileUtils**: клас для збереження та завантаження інформації з файлу.

|  |
| --- |
| import csv  from Student import Student  class FileUtils:      @staticmethod      def load\_from\_csv(filename, student\_list):          try:              with open(filename, mode='r', newline='', encoding='utf-8') as file:                  reader = csv.DictReader(file)                  for row in reader:                      student = Student(row["name"], row["phone"], row["email"], row["address"])                      student\_list.add\_student(student)              print(f"Data loaded successfully from {filename}.")          except FileNotFoundError:              print(f"File {filename} not found.")          except Exception as e:              print(f"Error loading file: {e}")      @staticmethod      def save\_to\_csv(filename, student\_list):          try:              with open(filename, mode='w', newline='', encoding='utf-8') as file:                  fieldnames = ["name", "phone", "email", "address"]                  writer = csv.DictWriter(file, fieldnames=fieldnames)                  writer.writeheader()                  for student in student\_list.students:                      writer.writerow({                          "name": student.name,                          "phone": student.phone,                          "email": student.email,                          "address": student.address                      })              print(f"Data saved successfully to {filename}.")          except Exception as e:              print(f"Error saving file: {e}") |

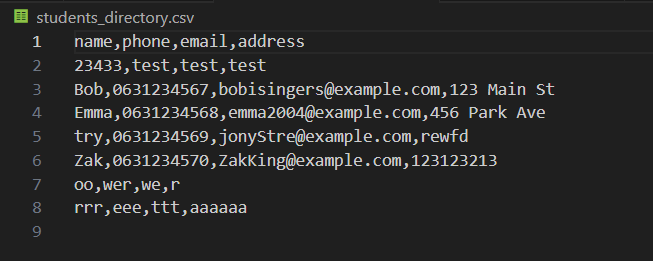
Файл Main - основний файл запуску:

|  |
| --- |
| from StudentList import StudentList  from FileUtils import FileUtils  from Student import Student  def main():      student\_list = StudentList()      # Завантаження даних із файлу      filename = "students\_directory.csv"      FileUtils.load\_from\_csv(filename, student\_list)      while True:          choice = input("Please specify the action [C create, U update, D delete, P print, X exit]: ").lower()          if choice == "c":              name = input("Please enter student name: ")              phone = input("Please enter student phone: ")              email = input("Please enter student email: ")              address = input("Please enter student address: ")              student = Student(name, phone, email, address)              student\_list.add\_student(student)              print("New student added.")          elif choice == "u":              name = input("Enter the student's name to update: ")              new\_name = input(f"Enter new name (current: {name}): ") or None              new\_phone = input(f"Enter new phone (current: {name}): ") or None              new\_email = input(f"Enter new email (current: {name}): ") or None              new\_address = input(f"Enter new address (current: {name}): ") or None              student\_list.update\_student(name, new\_name, new\_phone, new\_email, new\_address)          elif choice == "d":              name = input("Enter the student's name to delete: ")              student\_list.delete\_student(name)          elif choice == "p":              student\_list.print\_all()          elif choice == "x":              FileUtils.save\_to\_csv(filename, student\_list)              print("Exiting...")              break          else:              print("Invalid choice. Please try again.")  if \_\_name\_\_ == "\_\_main\_\_":      main() |

Результат роботи програми:



Збережений файл studends.csv:



Створення файлу юнит тестів.

import unittest

from Student import Student

class TestStudent(unittest.TestCase):

    def test\_student\_creation(self):

        student = Student("Alice", "0631234567", "alice@example.com", "123 Main St")

        self.assertEqual(student.name, "Alice")

        self.assertEqual(student.phone, "0631234567")

        self.assertEqual(student.email, "alice@example.com")

        self.assertEqual(student.address, "123 Main St")

    def test\_update\_student(self):

        student = Student("Alice", "0631234567", "alice@example.com", "123 Main St")

        student.update(phone="0639876543", email="alice\_new@example.com")

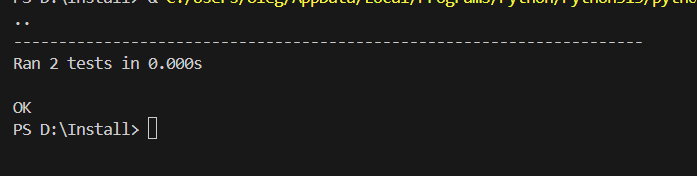
        self.assertEqual(student.phone, "0639876543")

        self.assertEqual(student.email, "alice\_new@example.com")

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

    unittest.main()

Результат виконання був успішним, отже працює правильно:



**Висновок:** В ході виконання цієї лабораторної роботи я вдосконалив попередню версію своєї програми, застосувавши принципи об'єктно-орієнтованого програмування (ООП). Крім того, я покрив основний функціонал коду юніт тестами, що дозволяє забезпечити надійність та стабільність роботи програми. Ці вдосконалення значно покращили структуру і тестованість коду, що робить програму більш ефективною та зручною для подальшого розвитку.