Технології програмування

Лабораторна робота №3

**ООП**

**Новик Максим Вікторович, КБ-231**

**Мета роботи**: 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.

Текст програми разом зі звітом розмістити в директорії lab\_03. Директорію lab\_03 розмістити в директорії, що використовується для виконання практичних завдань по кожній лекції та має назву **TP-KB-23[1 or 2]-Name-Surname**.

**Хід виконання завдання:**

Як і було зазначено в завданні, було перероблено програму з лабораторної роботи №2 під використання класів замість словників і розбити програму на зручні модулі.

Текст програми main.py:

|  |
| --- |
| import sys  from CSVManager import CSVManager  from .studentList import StudentList  def main():      if len(sys.argv) > 1:          file\_name = sys.argv[1]      else:          file\_name = "log.csv"      file\_manager = CSVManager(file\_name)      students = StudentList()      students.data\_list = file\_manager.loadCSV()      while True:          choice = input("Specify the action [C create, U update, D delete, P print, S save, X exit and save]: ").upper()          match choice:              case "C":                  students.addNewElement()              case "U":                  students.updateElement()              case "D":                  students.deleteElement()              case "P":                  students.printAllList()              case "S":                  file\_manager.saveCSV(students.data\_list)                  print("Data saved.")              case "X":                  file\_manager.saveCSV(students.data\_list)                  print("Exiting. Data saved.")                  break              case \_:                  print("Wrong choice")  if \_\_name\_\_ == "\_\_main\_\_":      main() |

Текст програми studentList.py:

|  |
| --- |
| from .student import Student  class StudentList:      def \_\_init\_\_(self):          self.data\_list = []      def printAllList(self):          for elem in self.data\_list:              print(elem)      def addNewElement(self):          name = input("Enter student name: ")          phone = input("Enter student phone: ")          age = input("Enter student age: ")          email = input("Enter student email: ")          newItem = Student(name, phone, age, email)          insertPosition = 0          for item in self.data\_list:              if name > item.name:                  insertPosition += 1              else:                  break          self.data\_list.insert(insertPosition, newItem)          print("New element has been added")      def deleteElement(self):          name = input("Please enter name to be deleted: ")          deletePosition = -1          for item in self.data\_list:              if name == item.name:                  deletePosition = self.data\_list.index(item)                  break          if deletePosition == -1:              print("Element was not found")          else:              del self.data\_list[deletePosition]              print(f"Element with name {name} has been deleted.")      def updateElement(self):          name = input("Please enter name to be updated: ")          insertPosition = -1          for item in self.data\_list:              if name == item.name:                  insertPosition = self.data\_list.index(item)                  break          if insertPosition == -1:              print("Name not found")          else:              old\_element = self.data\_list[insertPosition]              print(old\_element)              newname = input("Please enter student new name or skip: ") or old\_element.name              newphone = input("Please enter student new phone or skip: ") or old\_element.phone              newage = input("Please enter student new age or skip: ") or old\_element.age              newemail = input("Please enter student new email or skip: ") or old\_element.email              updateElement = Student(newname, newphone, newage, newemail)              del self.data\_list[insertPosition]              updatePosition = 0              for item in self.data\_list:                  if newname > item.name:                      updatePosition += 1              self.data\_list.insert(updatePosition, updateElement)              print("Element has been updated.") |

Текст програми student.py:

|  |
| --- |
| class Student:      def \_\_init\_\_(self, name, phone, age, email):          self.name = name          self.phone = phone          self.age = age          self.email = email      def \_\_str\_\_(self):          return f"Student name is {self.name}, Phone is {self.phone}, Age is {self.age}, Email is {self.email}" |

Текст програми student.py:

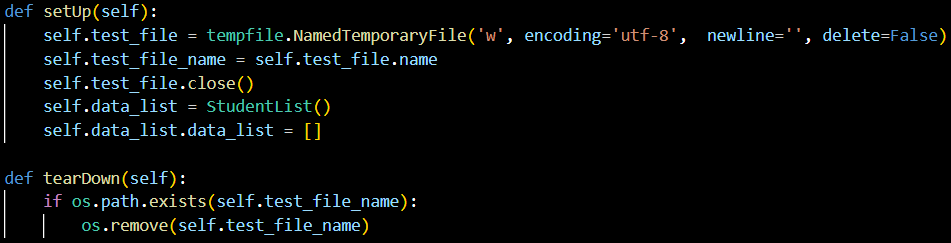
|  |
| --- |
| import csv  import os  from .student import Student  class CSVManager:      def \_\_init\_\_(self, file\_path):          self.file\_path = os.path.join(os.path.dirname(\_\_file\_\_), file\_path)      def loadCSV(self):          data\_list = []          try:              with open(self.file\_path, mode="r") as file:                  reader = csv.DictReader(file)                  for row in reader:                      data\_list.append(Student(                          name = row.get("name", "Unknown"),                          phone = row.get("phone", "Unknown"),                          age = row.get("age", "Unknown"),                          email = row.get("email", "Unknown")                      ))              print(f"Data loaded.")          except FileNotFoundError:              print("File not found. Starting with an empty list.")          return data\_list      def saveCSV(self, data\_list):          with open(self.file\_path, "w", newline="") as file:              fieldnames = ["name", "phone", "age", "email"]              writer = csv.DictWriter(file, fieldnames=fieldnames)              writer.writeheader()              for student in data\_list:                  writer.writerow({"name": student.name, "phone": student.phone, "age": student.age, "email": student.email}) |

При виконанні завдання було внесено мінімум змін до програми лабораторної роботи №2.

**UNIT-тести**

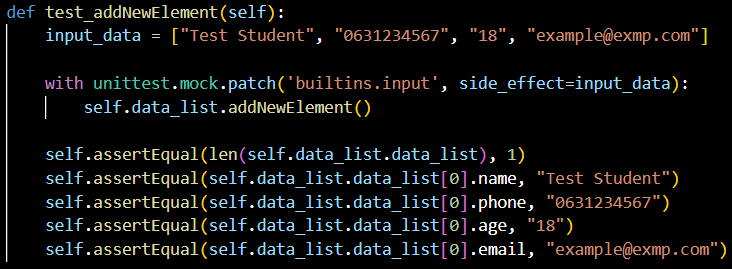
Ініціалізація тестів (setUp і tearDown):

* Використовується тимчасовий файл, створений за допомогою модуля tempfile.
* У методі setUp ініціалізується порожній список студентів data\_list і створюється тимчасовий файл для тестування.
* Метод tearDown видаляє тимчасовий файл після завершення тесту, забезпечуючи чистоту середовища.



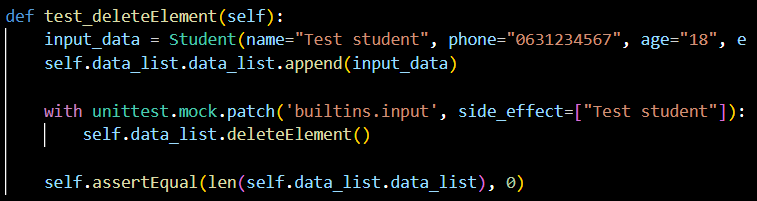
Тестування функції addNewElement():

* Симулюється введення даних через unittest.mock.patch, що дозволяє автоматизувати тестування без втручання користувача.
* Перевіряється, чи елемент коректно додається до списку, і чи його атрибути відповідають очікуваним значенням (name, phone, age, email).



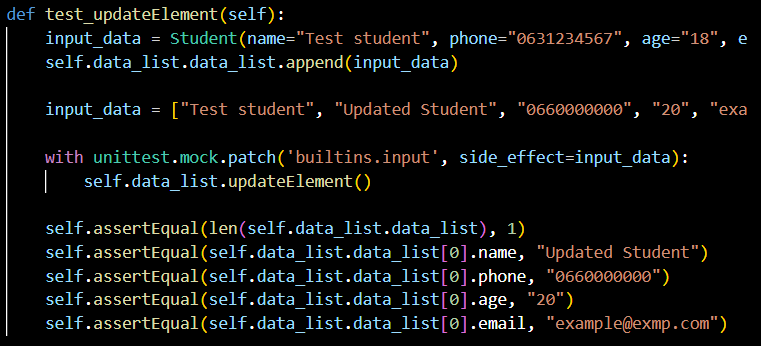
Тестування функції deleteElement():

* У список додається тестовий об’єкт типу Student.
* Імітується введення назви студента для видалення, після чого перевіряється, чи список порожній.



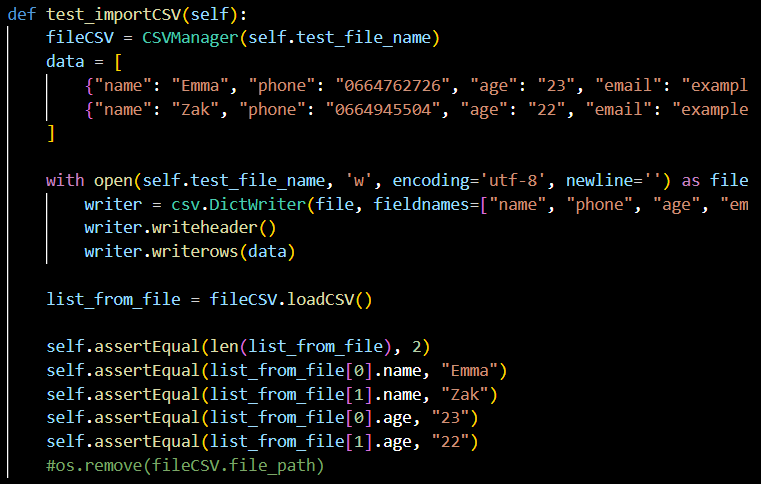
Тестування функції updateElement():

* Спочатку додається студент зі старими даними.
* Імітується введення нових даних, включно з пропуском оновлення певних полів.
* Перевіряється, чи оновлені значення відповідають введеним.



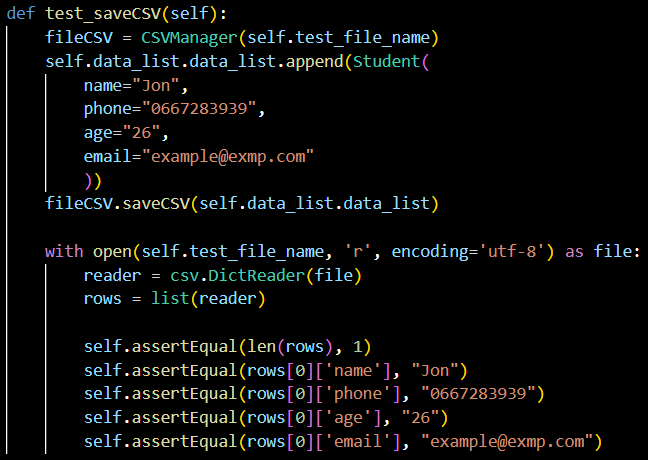
Тестування функції loadCSV():

* Створюється CSV-файл із даними для двох студентів.
* Файл зчитується через функцію loadCSV із класу CSVManager.
* Перевіряється коректність кількості студентів та значення їхніх атрибутів.
* Команда os.remove(fileCSV.file\_path) закоментована та залишена на випадок, якщо тестовий файл не видаляється.



Тестування функції saveCSV():

* До списку додається студент із заданими даними.
* Список зберігається в CSV-файл через функцію saveCSV.
* Потім файл зчитується, і перевіряються значення збережених даних.



Текст програми:

|  |
| --- |
| import unittest  import unittest.mock  import os  import tempfile  import csv  from .student import Student  from .studentList import StudentList  from .CSVManager import CSVManager  class TestStudenList(unittest.TestCase):      def setUp(self):          self.test\_file = tempfile.NamedTemporaryFile('w', encoding='utf-8',  newline='', delete=False)          self.test\_file\_name = self.test\_file.name          self.test\_file.close()          self.data\_list = StudentList()          self.data\_list.data\_list = []      def tearDown(self):          if os.path.exists(self.test\_file\_name):              os.remove(self.test\_file\_name)      def test\_addNewElement(self):          input\_data = ["Test Student", "0631234567", "18", "example@exmp.com"]            with unittest.mock.patch('builtins.input', side\_effect=input\_data):              self.data\_list.addNewElement()          self.assertEqual(len(self.data\_list.data\_list), 1)          self.assertEqual(self.data\_list.data\_list[0].name, "Test Student")          self.assertEqual(self.data\_list.data\_list[0].phone, "0631234567")          self.assertEqual(self.data\_list.data\_list[0].age, "18")          self.assertEqual(self.data\_list.data\_list[0].email, "example@exmp.com")      def test\_deleteElement(self):          input\_data = Student(name="Test student", phone="0631234567", age="18", email="example@exmp.com")          self.data\_list.data\_list.append(input\_data)          with unittest.mock.patch('builtins.input', side\_effect=["Test student"]):              self.data\_list.deleteElement()          self.assertEqual(len(self.data\_list.data\_list), 0)      def test\_updateElement(self):          input\_data = Student(name="Test student", phone="0631234567", age="18", email="example@exmp.com")          self.data\_list.data\_list.append(input\_data)          input\_data = ["Test student", "Updated Student", "0660000000", "20", "example@exmp.com"]          with unittest.mock.patch('builtins.input', side\_effect=input\_data):              self.data\_list.updateElement()            self.assertEqual(len(self.data\_list.data\_list), 1)          self.assertEqual(self.data\_list.data\_list[0].name, "Updated Student")          self.assertEqual(self.data\_list.data\_list[0].phone, "0660000000")          self.assertEqual(self.data\_list.data\_list[0].age, "20")          self.assertEqual(self.data\_list.data\_list[0].email, "example@exmp.com")      def test\_importCSV(self):          fileCSV = CSVManager(self.test\_file\_name)          data = [              {"name": "Emma", "phone": "0664762726", "age": "23", "email": "example@exmp.com"},              {"name": "Zak", "phone": "0664945504", "age": "22", "email": "example@exmp.com"}          ]          with open(self.test\_file\_name, 'w', encoding='utf-8', newline='') as file:              writer = csv.DictWriter(file, fieldnames=["name", "phone", "age", "email"])              writer.writeheader()              writer.writerows(data)          list\_from\_file = fileCSV.loadCSV()            self.assertEqual(len(list\_from\_file), 2)          self.assertEqual(list\_from\_file[0].name, "Emma")          self.assertEqual(list\_from\_file[1].name, "Zak")          self.assertEqual(list\_from\_file[0].age, "23")          self.assertEqual(list\_from\_file[1].age, "22")          #os.remove(fileCSV.file\_path)      def test\_saveCSV(self):          fileCSV = CSVManager(self.test\_file\_name)          self.data\_list.data\_list.append(Student(              name="Jon",              phone="0667283939",              age="26",              email="example@exmp.com"              ))          fileCSV.saveCSV(self.data\_list.data\_list)          with open(self.test\_file\_name, 'r', encoding='utf-8') as file:              reader = csv.DictReader(file)              rows = list(reader)              self.assertEqual(len(rows), 1)              self.assertEqual(rows[0]['name'], "Jon")              self.assertEqual(rows[0]['phone'], "0667283939")              self.assertEqual(rows[0]['age'], "26")              self.assertEqual(rows[0]['email'], "example@exmp.com")  if \_\_name\_\_ == '\_\_main\_\_':      unittest.main() |

**Висновок:**

Під час виконання лабораторної роботи я використовував теоретичне підґрунтя про ООП у мові Python переробивши програму телефонного довідника студентів використовуючи принципи ООП для формування відомостей про студентів.

**Посилання на github:**

https://github.com/GAGGAGX/TP-KB-231-Novyk-Maksym

**Знімок екрану з посилання на github:**

