Технології програмування

Лабораторна робота №3

**ООП**

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

**Теоретичні відомості**

**ООП та мова Python**

Python має безліч вбудованих типів, наприклад, int, str і так далі, які ми можемо використовувати у програмі. Але Python також дозволяє визначати власні типи за допомогою класів. Клас є деякою сутністю. Конкретним здійсненням класу є об'єкт.

Можна ще провести таку аналогію. У нас у всіх є деяке уявлення про людину, яка має ім'я, вік, якісь інші характеристики Людина може виконувати деякі дії - ходити, бігати, думати і т.д. Тобто це уявлення, яке включає набір характеристик та дій, можна назвати класом. Конкретне втілення цього шаблону може відрізнятися, наприклад, одні мають одне ім'я, інші - інше ім'я. І реально існуюча людина представлятиме об'єкт цього класу.

Клас визначається за допомогою ключового слова:

|  |
| --- |
| class class\_name:  attributes  methods |

Усередині класу визначаються його атрибути, які зберігають різні характеристики класу, та методи – функції класу.

Приклад найпростішого класу:

|  |
| --- |
| class Person:  pass |

У разі визначено клас Person, який умовно представляє людини. В даному випадку в класі не визначається жодних методів чи атрибутів. Однак оскільки в ньому має бути щось визначено, то як замінник функціоналу класу застосовується оператор **pass**. Цей оператор застосовується, коли синтаксично необхідно визначити певний код, проте ми не хочемо його, і замість конкретного коду вставляємо оператор **pass**.

Після створення класу, можна визначити об'єкти цього класу. Наприклад:

|  |
| --- |
| class Person:  pass    tom = Person()  bob = Person() |

Після визначення класу Person створюються два об'єкти класу Person – tom і bob. Для створення об'єкта застосовується спеціальна функція – конструктор, яка називається як ім'я класу і яка повертає об'єкт класу. Тобто у цьому випадку виклик Person() представляє виклик конструктора. Кожен клас за замовчуванням має конструктор без параметрів.

**Методи класу**

Методи класу фактично представляють функції, які визначені всередині класу і які визначають його поведінку. Наприклад, визначимо клас Person з одним методом:

|  |
| --- |
| class Person:  def say\_hello(self):  print("Hello")    tom = Person()  tom.say\_hello() |

Тут визначено метод say\_hello(), який умовно виконує вітання – виводить рядок на консоль. При визначенні методів будь-якого класу слід враховувати, що всі вони повинні приймати як перший параметр посилання на поточний об'єкт, який відповідно до умов називається self. Через це посилання всередині класу ми можемо звернутися до функціональності об'єкта. Але при самому виклик методу цей параметр не враховується.

Використовуючи ім'я об'єкта, ми можемо звернутися до його способів. Для звернення до методів застосовується нотація точки – після імені об'єкта ставиться точка і після неї йде виклик методу. Наприклад, звернення до методу say\_hello() для виведення привітання на консоль:

|  |
| --- |
| tom.say\_hello() |

У результаті ця програма виведе на консоль рядок "Hello".

Якщо метод повинен приймати інші параметри, вони визначаються після параметра self, і за виклику подібного методу їм необхідно передати значення:

|  |
| --- |
| class Person:  def say(self, message):  print(message)      tom = Person()  tom.say("Hello, World!") |

Тут визначено метод say(). Він приймає два параметри: self і message. І другим параметром - message при виклику методу необхідно передати значення.

**self**

Через ключове слово self можна звертатися всередині класу до функціональності поточного об'єкта. Наприклад, визначимо два методи у класі Person:

|  |
| --- |
| class Person:    def say(self, message):  print(message)    def say\_hello(self):  self.say("Hello, world")      tom = Person()  tom.say\_hello() |

Тут в одному методі - say\_hello() викликається інший метод - say(). Оскільки метод say() приймає крім self ще параметри (параметр message), то за виклику методу цього параметра передається значення.

**Конструктори**

Для створення класу об'єкта використовується конструктор. Так, вище коли ми створювали об'єкти класу Person, ми використовували за замовчуванням конструктор, який не приймає параметрів і який неявно мають всі класи:

|  |
| --- |
| tom = Person() |

Однак ми можемо явно визначити в класах конструктор за допомогою спеціального методу, який називається **\_\_init\_\_()** (по два прочерки з кожної сторони). Наприклад, змінимо клас Person, додавши до нього конструктор:

|  |
| --- |
| class Person:  # конструктор  def \_\_init\_\_(self):  print("Person creating")    def say\_hello(self):  print("Hello")      tom = Person()  tom.say\_hello() |

Отже, тут у коді класу Person визначено конструктор та метод say\_hello(). Як перший параметр конструктор, як і методи, також приймає посилання на поточний об'єкт - self. Зазвичай конструктори застосовуються визначення дій, які мають здійснюватися під час створення об'єкта.

Тепер під час створення об'єкта буде здійснено виклик конструктора **\_\_init\_\_()** з класу Person, який виведе на консоль рядок " Person creating ".

**Атрибути об'єкту**

Атрибути зберігають стан об'єкта. Для визначення та встановлення атрибутів усередині класу можна використовувати слово self. Наприклад, визначимо наступний клас Person:

|  |
| --- |
| class Person:  def \_\_init\_\_(self, name):  self.name = name  self.age = 1  tom = Person("Tom")  print(tom.name)  print(tom.age)  # зміна значення  tom.age = 37  print(tom.age) |

Тепер конструктор класу Person приймає ще один параметр – name. Через цей параметр в конструктор буде передаватися ім'я людини, що створюється. Усередині конструктора встановлюються два атрибути - name і age (умовно ім'я та вік людини).

Якщо ми визначили у класі конструктор \_\_init\_\_, ми вже не зможемо викликати конструктор за замовчуванням. Тепер нам треба викликати наш явним чином оподаткований конструктор \_\_init\_\_, який необхідно передати значення для параметра name:

Далі на ім'я об'єкта ми можемо звертатися до атрибутів об'єкта - отримувати та змінювати їх значення:

|  |
| --- |
| print(tom.name)  tom.age = 37 |

Для звернення до атрибутів об'єкта всередині класу у його методах також застосовується слово self.

**Створення об'єктів**

Кількість об'єктів, що може бути створена – необмежена.

|  |
| --- |
| class Person:    def \_\_init\_\_(self, name):  self.name = name  self.age = 1    def display\_info(self):  print(f"Name: {self.name} Age: {self.age}")      tom = Person("Tom")  tom.age = 37  tom.display\_info() # Name: Tom Age: 37    bob = Person("Bob")  bob.age = 41  bob.display\_info() # Name: Bob Age: 41 |

Тут створюються два об'єкти класу Person: tom та bob. Вони відповідають визначенню класу Person, мають однаковий набір атрибутів та методів, проте їхній стан відрізнятиметься. При виконанні програми Python динамічно визначатиме self - він представляє об'єкт, у якого викликається метод.

**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-22[1 or 2]-Name-Surname**.

**Хід роботи**

student.py: містить клас Student, який використовується для зберігання даних студентів. Клас має конструктор для ініціалізації атрибутів: ім'я, телефон, email і адресу. В методі \_\_str\_\_ я визначаю, як буде виглядати виведення студентів.

student\_list.py: містить клас StudentList, який дозволяє додавати, оновлювати, видаляти та виводити студентів. Ось основні методи цього класу:

* **add\_student**: Додає нового студента після того, як користувач вводить необхідну інформацію (ім'я, телефон, email, адресу). Студент додається в список, при цьому забезпечується сортування за ім'ям.
* **delete\_student**: Дозволяє видалити студента за ім'ям.
* **update\_student**: Оновлює дані студента (телефон, email, адресу) за ім'ям. Після оновлення список студентів сортується за ім'ям.
* **print\_all\_students**: Виводить інформацію про всіх студентів у списку.

file\_manager.py: містить клас FileManager, який відповідає за роботу з файлами CSV. Він має два основні методи:

* **load\_data\_from\_csv**: Завантажує дані з CSV файлу в об'єкти Student.
* **save\_data\_to\_csv**: Зберігає дані студентів в CSV файл.

main.py: Основний файл, який запускає програму. Програма приймає ім'я файлу CSV як аргумент командного рядка. Якщо файл не передано, програма завершується. Далі здійснюється завантаження даних з файлу, після чого в циклі користувач може виконувати операції з додавання, оновлення, видалення або виведення студентів.

test\_lab3.py: Написав юніт тести, які тестують кожну функцію нашої програми. Для тестування вводиться фіктивне введення користувача через monkeypatch та перевіряється, чи правильно працюють методи додавання, оновлення, видалення та виведення студентів.

Код програми:

file\_manager.py:

import csv

from student import Student

class FileManager:

    def \_\_init\_\_(self):

        # Можна додати ініціалізацію, якщо необхідно

        pass

    def load\_data\_from\_csv(self, file\_name):

        try:

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

                reader = csv.DictReader(file)

                students = [Student(row['name'], row['phone'], row['email'], row['address']) for row in reader]

        except FileNotFoundError:

            print(f"File {file\_name} not found. Starting with an empty list.")

            students = []

        return students

    def save\_data\_to\_csv(self, file\_name, students):

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

            fieldnames = ["name", "phone", "email", "address"]

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

            writer.writeheader()

            for student in students:

                writer.writerow({"name": student.name, "phone": student.phone, "email": student.email, "address": student.address})

main.py

import sys

from student\_list import StudentList

from file\_manager import FileManager

def main():

    if len(sys.argv) < 2:

        print("No input file provided. Exiting...")

        sys.exit(1)

    file\_name = sys.argv[1]

    file\_manager = FileManager()

    # Завантажуємо дані з CSV

    students = file\_manager.load\_data\_from\_csv(file\_name)

    # Створюємо об'єкт StudentList з уже завантаженими студентами

    student\_list = StudentList(students)

    while True:

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

        match choice:

            case "C" | "c":

                print("New element will be created:")

                student\_list.add\_student()

                student\_list.print\_all\_students()

            case "U" | "u":

                print("Existing element will be updated:")

                student\_list.update\_student()

                student\_list.print\_all\_students()

            case "D" | "d":

                print("Element will be deleted:")

                student\_list.delete\_student()

                student\_list.print\_all\_students()

            case "P" | "p":

                print("List will be printed:")

                student\_list.print\_all\_students()

            case "X" | "x":

                print("Exiting program...")

                # Зберігаємо дані в CSV

                file\_manager.save\_data\_to\_csv(file\_name, student\_list.get\_all\_students())

                break

            case \_:

                print("Wrong choice")

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

    main()

student\_list.py

from student import Student

class StudentList:

    def \_\_init\_\_(self, students=None):

        self.students = students if students is not None else []

    def add\_student(self):

        name = input("Please enter student name: ")

        phone = input("Please enter student phone: ")

        email = input("Please enter student email: ")

        address = input("Please enter student address: ")

        new\_student = Student(name, phone, email, address)

        insert\_position = 0

        for student in self.students:

            if name > student.name:

                insert\_position += 1

            else:

                break

        self.students.insert(insert\_position, new\_student)

        print(f"Student {name} has been added.")

    def delete\_student(self):

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

        delete\_position = -1

        for index, student in enumerate(self.students):

            if student.name == name:

                delete\_position = index

                break

        if delete\_position == -1:

            print(f"Student {name} not found.")

        else:

            del self.students[delete\_position]

            print(f"Student {name} has been deleted.")

    def update\_student(self):

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

        student\_found = False

        for student in self.students:

            if student.name == name:

                student\_found = True

                new\_name = input(f"Enter new name (current: {student.name}): ")

                student.name = new\_name.strip() or student.name

                phone = input(f"Enter new phone (current: {student.phone}): ")

                student.phone = phone.strip() or student.phone

                email = input(f"Enter new email (current: {student.email}): ")

                student.email = email.strip() or student.email

                address = input(f"Enter new address (current: {student.address}): ")

                student.address = address.strip() or student.address

                break

        if not student\_found:

            print(f"Student {name} not found.")

        else:

            self.students.sort(key=lambda x: x.name)

            print(f"Student {new\_name} has been updated.")

    def print\_all\_students(self):

        for student in self.students:

            print(f"Student name is {student.name}, Phone is {student.phone}, Email is {student.email}, Address is {student.address}")

    def get\_all\_students(self):

        return self.students

student.py

class Student:

    def \_\_init\_\_(self, name, phone, email, address):

        self.name = name

        self.phone = phone

        self.email = email

        self.address = address

    def \_\_str\_\_(self):

        return f"Student(name={self.name}, phone={self.phone}, email={self.email}, address={self.address})"

test\_lab3.py

import pytest

from student import Student

from student\_list import StudentList

# Тест для додавання студента

def test\_add\_student(monkeypatch):

    inputs = iter(["Alice", "0631112222", "alice@example.com", "321 Cedar St"])

    monkeypatch.setattr('builtins.input', lambda \_: next(inputs))

    student\_list = StudentList()

    student\_list.add\_student()

    assert len(student\_list.students) == 1

    assert student\_list.students[0].name == "Alice"

    assert student\_list.students[0].phone == "0631112222"

    assert student\_list.students[0].email == "alice@example.com"

    assert student\_list.students[0].address == "321 Cedar St"

# Тест для видалення студента

def test\_delete\_student(monkeypatch):

    inputs = iter(["Alice", "0631112222", "alice@example.com", "321 Cedar St", "Alice"])

    monkeypatch.setattr('builtins.input', lambda \_: next(inputs))

    student\_list = StudentList()

    student\_list.add\_student()

    assert len(student\_list.students) == 1

    student\_list.delete\_student()

    assert len(student\_list.students) == 0

# Тест для оновлення студента

def test\_update\_student(monkeypatch):

    add\_inputs = iter(["Alice", "0631112222", "alice@example.com", "321 Cedar St"])

    update\_inputs = iter(["Alice", "Bob", "123", "new\_email@example.com", "New Address"])

    monkeypatch.setattr('builtins.input', lambda \_: next(add\_inputs))

    student\_list = StudentList()

    student\_list.add\_student()

    assert len(student\_list.students) == 1

    monkeypatch.setattr('builtins.input', lambda \_: next(update\_inputs))

    student\_list.update\_student()

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

    assert updated\_student.name == "Bob"

    assert updated\_student.phone == "123"

    assert updated\_student.email == "new\_email@example.com"

    assert updated\_student.address == "New Address"

def test\_print\_all\_students(monkeypatch, capsys):

    add\_inputs = iter(["Alice", "0631112222", "alice@example.com", "321 Cedar St"])

    monkeypatch.setattr('builtins.input', lambda \_: next(add\_inputs))

    student\_list = StudentList()

    student\_list.add\_student()

    assert len(student\_list.students) == 1

    student\_list.print\_all\_students()

    captured = capsys.readouterr()

    assert "Student name is Alice" in captured.out

    assert "Phone is 0631112222" in captured.out

    assert "Email is alice@example.com" in captured.out

    assert "Address is 321 Cedar St" in captured.out