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

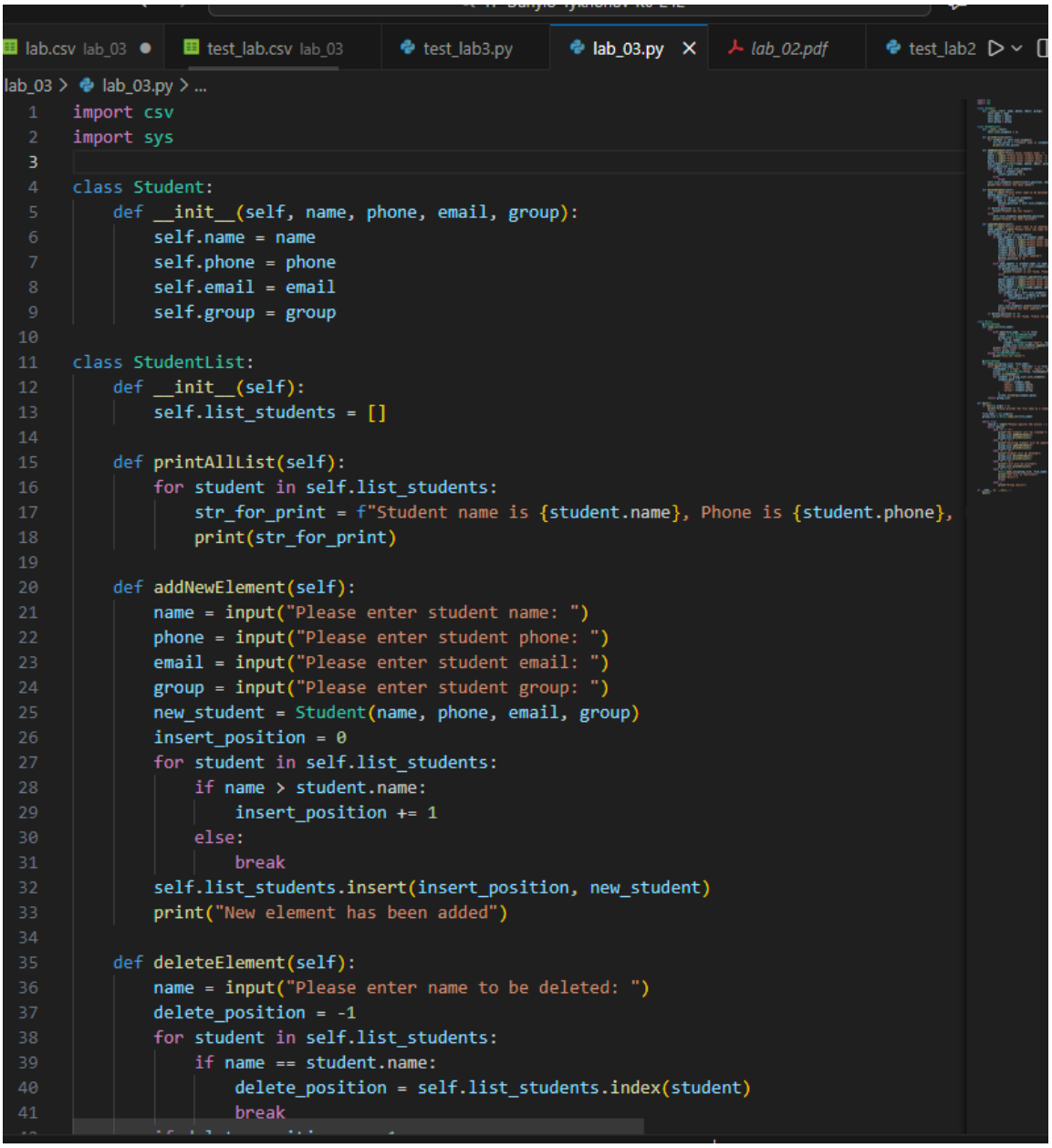
Переробити функціональність телефонного довідника студентів групи, що був розроблений у Лабораторній роботі №2 використовуючи принципи ООП:

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**.

Текст програми:


lab.csv lab_03
test_lab.csv lab_03
test_lab3.py
lab_03.py
lab_02.pdf
test_lab2
lab_03 > lab_03.py > ...
import csv
import sys
class Student:
def __init__(self, name, phone, email, group):
self.name = name
self.phone = phone
self.email = email
self.group = group
class StudentList:
def __init__(self):
self.list_students = []
def printAllList(self):
for student in self.list_students:
str_for_print = f"Student name is {student.name}, Phone is {student.phone},
print(str_for_print)
def addNewElement(self):
name = input("Please enter student name: ")
phone = input("Please enter student phone: ")
email = input("Please enter student email: ")
group = input("Please enter student group: ")
new_student = Student(name, phone, email, group)
insert_position = 0
for student in self.list_students:
if name > student.name:
insert_position += 1
else:
break
self.list_students.insert(insert_position, new_student)
print("New element has been added")
def deleteElement(self):
name = input("Please enter name to be deleted: ")
delete_position = -1
for student in self.list_students:
if name == student.name:
delete_position = self.list_students.index(student)
break

```
42          if delete_position == -1:
43              print("Element was not found")
44          else:
45              self.list_students.pop(delete_position)
46              print("Element has been deleted")
47
48      def updateElement(self):
49          name = input("Please enter name to be updated: ")
50          name_update = input("Please enter new name for this student: ")
51          delete_position = -1
52          for student in self.list_students:
53              if name_update == name == student.name:
54                  phone_update = input("Please enter new phone for this student: ")
55                  email_update = input("Please enter new email for this student: ")
56                  group_update = input("Please enter new group for this student: ")
57                  student.phone = phone_update
58                  student.email = email_update
59                  student.group = group_update
60                  print("Element has been updated")
61                  delete_position = -2
62                  break
63              elif name_update != student.name and name == student.name:
64                  delete_position = self.list_students.index(student)
65                  if delete_position == -1:
66                      print("Student is not found. Please try again correctly.")
67                  else:
68                      self.list_students.pop(delete_position)
69                  phone_update = input("Please enter new phone for this student: ")
70                  email_update = input("Please enter new email for this student: ")
71                  group_update = input("Please enter new group for this student: ")
72                  new_student = Student(name_update, phone_update, email_update, group_upd
73                  insert_position = 0
74                  for student_up in self.list_students:
75                      if name_update > student_up.name:
76                          insert_position += 1
77                      else:
78                          break
79                  self.list_students.insert(insert_position, new_student)
```

```
                self.list_students.insert(insert_position, new_student)
                print("Element has been updated")
                break
        if delete_position == -1:
            print("Student is not found. Please try again correctly.")

class Utils:
    @staticmethod
    def load_csv(file_name):
        try:
            with open(file_name, 'r') as file:
                reader = csv.DictReader(file)
                group_list = StudentList()
                for row in reader:
                    student = Student(row["name"], row["phone"], row["email"], row["group"])
                    group_list.list_students.append(student)
            print("Data loaded successfully!")
            return group_list
        except FileNotFoundError:
            print("File not found!")

    @staticmethod
    def save_csv(group_list, file_name):
        with open(file_name, 'w', newline='') as file:
            fieldnames = ["name", "phone", "email", "group"]
            writer = csv.DictWriter(file, fieldnames=fieldnames)
            writer.writeheader()
            for student in group_list.list_students:
                student_data = {
                    "name": student.name,
                    "phone": student.phone,
                    "email": student.email,
                    "group": student.group
                }
                writer.writerow(student_data)
        return group_list

def main():
    if len(sys.argv) < 2:
        print("Please provide the file name as a command line argument!")
        return
    file_name = sys.argv[1]
    group_list = Utils.load_csv(file_name)

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

```

    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:")
                group_list.addNewElement()
                group_list.printAllList()
            case "U" | "u":
                print("Existing element will be updated")
                group_list.updateElement()
                group_list.printAllList()
            case "D" | "d":
                print("Element will be deleted")
                group_list.deleteElement()
                group_list.printAllList()
            case "P" | "p":
                print("List will be printed")
                group_list.printAllList()
            case "X" | "x":
                Utils.save_csv(group_list, file_name)
                print("CSV file is rewritten")
                print("Exit()")
                break
            case _:
                print("Wrong choice")

if __name__ == '__main__':
    main()
```

```python
import csv
import sys

class Student:
    def __init__(self, name, phone, email, group):
        self.name = name
        self.phone = phone
        self.email = email
        self.group = group

class StudentList:
    def __init__(self):
        self.list_students = []

    def printAllList(self):
        for student in self.list_students:
            str_for_print = f"Student name is {student.name}, Phone is {student.phone}, Email: {student.email}, Group: {student.group}"
            print(str_for_print)

    def addNewElement(self):
        name = input("Please enter student name: ")
        phone = input("Please enter student phone: ")
        email = input("Please enter student email: ")
        group = input("Please enter student group: ")
        new_student = Student(name, phone, email, group)
        insert_position = 0
        for student in self.list_students:
            if name > student.name:
                insert_position += 1
            else:
                break
        self.list_students.insert(insert_position, new_student)
        print("New element has been added")

    def deleteElement(self):
        name = input("Please enter name to be deleted: ")
        delete_position = -1
        for student in self.list_students:
            if name == student.name:
                delete_position = self.list_students.index(student)
                break
        if delete_position == -1:
            print("Element was not found")
        else:
            self.list_students.pop(delete_position)
            print("Element has been deleted")

    def updateElement(self):
```

```
        name = input("Please enter name to be updated: ")
        name_update = input("Please enter new name for this student: ")
        delete_position = -1
        for student in self.list_students:
            if name_update == name == student.name:
                phone_update = input("Please enter new phone for this student: ")
                email_update = input("Please enter new email for this student: ")
                group_update = input("Please enter new group for this student: ")
                student.phone = phone_update
                student.email = email_update
                student.group = group_update
                print("Element has been updated")
                delete_position = -2
                break
            elif name_update != student.name and name == student.name:
                delete_position = self.list_students.index(student)
                if delete_position == -1:
                    print("Student is not found. Please try again correctly.")
                else:
                    self.list_students.pop(delete_position)
                phone_update = input("Please enter new phone for this student: ")
                email_update = input("Please enter new email for this student: ")
                group_update = input("Please enter new group for this student: ")
                new_student = Student(name_update, phone_update, email_update, group_update)
                insert_position = 0
                for student_up in self.list_students:
                    if name_update > student_up.name:
                        insert_position += 1
                    else:
                        break
                self.list_students.insert(insert_position, new_student)
                print("Element has been updated")
                break
        if delete_position == -1:
            print("Student is not found. Please try again correctly.")

class Utils:
    @staticmethod
    def load_csv(file_name):
        try:
            with open(file_name, 'r') as file:
                reader = csv.DictReader(file)
                group_list = StudentList()
                for row in reader:
                    student = Student(row["name"], row["phone"], row["email"], row["group"])
                    group_list.list_students.append(student)
```

```
            print("Data loaded successfully!")
            return group_list
        except FileNotFoundError:
            print("File not found!")

    @staticmethod
    def save_csv(group_list, file_name):
        with open(file_name, 'w', newline='') as file:
            fieldnames = ["name", "phone", "email", "group"]
            writer = csv.DictWriter(file, fieldnames=fieldnames)
            writer.writeheader()
            for student in group_list.list_students:
                student_data = {
                    "name": student.name,
                    "phone": student.phone,
                    "email": student.email,
                    "group": student.group
                }
                writer.writerow(student_data)
        return group_list

def main():
    if len(sys.argv) < 2:
        print("Please provide the file name as a command line argument!")
        return
    file_name = sys.argv[1]
    group_list = Utils.load_csv(file_name)

    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:")
                group_list.addNewElement()
                group_list.printAllList()
            case "U" | "u":
                print("Existing element will be updated")
                group_list.updateElement()
                group_list.printAllList()
            case "D" | "d":
                print("Element will be deleted")
                group_list.deleteElement()
                group_list.printAllList()
            case "P" | "p":
                print("List will be printed")
```

```
                group_list.printAllList()
            case "X" | "x":
                Utils.save_csv(group_list, file_name)
                print("CSV file is rewritten")
                print("Exit()")
                break
            case _:
                print("Wrong choice")

if __name__ == '__main__':
    main()
```