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.

**Хід виконання завдання до лабораторної роботи**

1. Прочитав суть завдання.

2. Відкрив графічний редактор.

3. Написав код, який виконує завдання.

4. Написав звіт про виконання завдання.

5. Створив і виконав Юніт тести.

6. Завантажив всі зміни для розгляду в гілку Labs.

Код файлу lab\_03.py:

|  |
| --- |
| import sys  from Utils import FileUtils  from StudentList import CommandList  def main():  Command\_List = CommandList()  Utils = FileUtils()  if len(sys.argv) > 1:  try:  file = sys.argv[1]  Utils.LoadFile(file, Command\_List)  print("Data loaded successfully.")  except IOError as e:  print("File upload error.")  else:  print("No CSV file specified.")  while True:  choice = input("Please specify the action [ C create, U update, D delete, P print, S save, X exit ] ")  match choice.upper():  case "C":  print("New element will be created:")  Command\_List.addNewElement()  case "U":  print("Existing element will be updated")  name = input("Please enter name to be updated: ")  Command\_List.updateElement(name)  case "D":  print("Element will be deleted")  name = input("Please enter name to be deleted: ")  Command\_List.deleteElement(name)  case "P":  print("List will be printed")  Command\_List.printAllList()  case "S":  file = input("Enter a name for the CSV file: ")  Utils.SaveFile(file, Command\_List)  case "X":  print("Exit")  break  case \_:  print("Wrong choice")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Код файлу Student.py:

|  |
| --- |
| class StudentUnit:  def \_\_init\_\_(self, name, phone, age, email):  self.name = name  self.phone = phone  self.age = age  self.email = email |

Код файлу StudentList.py:

|  |
| --- |
| from Student import StudentUnit  class CommandList:  def \_\_init\_\_(self):  self.students = []  def printAllList(self):  for student in self.students:  text = f"Student name is {student.name}, Phone is {student.phone}, Age is {student.age}, Email is {student.email}"  print(text)  def addNewElement(self):  name = input("Please enter student name: ")  phone = input("Please enter student phone: ")  age = int(input("Please enter student age: "))  email = input("Please enter student email: ")  student = StudentUnit(name, phone, age, email)  self.students.append(student)  self.students.sort(key=lambda x: (x.name, x.age))  print("New element has been added")  def deleteElement(self, name):  deletePosition = -1  for item in self.students:  if name == item.name:  deletePosition = self.students.index(item)  break  if deletePosition == -1:  print("Element was not found")  else:  print(f"Delete position {str(deletePosition)}")  del self.students[deletePosition]  return  def updateElement(self, name):  for index, student in enumerate(self.students):  if name == student.name:  print(f"Student {name} found")  new\_name = input("Please enter new name: ")  if new\_name and new\_name != name:  del self.students[index]  new\_student = StudentUnit(new\_name, student.phone, student.age, student.email)  insert\_position = 0  for item in self.students:  if new\_name > item.name:  insert\_position += 1  else:  break  self.students.insert(insert\_position, new\_student)  new\_phone = input("Please enter updated phone: ")  if new\_phone:  new\_student.phone = new\_phone  while True:  try:  new\_age = input("Please enter updated age: ")  if new\_age:  new\_student.age = int(new\_age)  break  except ValueError:  print("Wrong student age")  new\_email = input("Please enter updated email: ")  if new\_email:  new\_student.email = new\_email  print("Element has been updated")  return  print("Student not found") |

Код файлу Utils.py:

|  |
| --- |
| import csv  from Student import StudentUnit  class FileUtils:  @staticmethod  def LoadFile(file, Student\_list):  Student\_list.students = []  with open(file, encoding="utf-8") as file:  result = csv.DictReader(file)  Student\_list.students.extend([StudentUnit(row["name"], row["phone"], row["age"], row["email"]) for row in result])  return Student\_list  @staticmethod  def SaveFile(file, Student\_list):  rows = ["name", "phone", "age", "email"]  try:  with open(file, "w", newline="", encoding="utf-8") as file:  result = csv.DictWriter(file, fieldnames=rows)  result.writeheader()  result.writerows([vars(student) for student in Student\_list.students])  print(f"The data is saved")  except IOError as e:  print(f"Error saving") |

Файл Юніт тестів:

|  |
| --- |
| import pytest  from pathlib import Path  from io import StringIO  from unittest.mock import patch  from lab\_03 import main  from Utils import FileUtils  from StudentList import CommandList  from Student import StudentUnit  import csv  @pytest.fixture  def TestingData():  return [  {"name": "Bob", "phone": "1112233", "age": "23", "email": "bob@example.com"},  {"name": "Dilan", "phone": "2223344", "age": "18", "email": "Dilan@example.com"},  {"name": "Zak", "phone": "3334455", "age": "20", "email": "zak@example.com"}  ]  def test\_LoadFile(TestingData, tmp\_path):  PathFile = tmp\_path / "lab3.csv"  with open(PathFile, "w", newline="", encoding="utf-8") as file:  ResFile = csv.DictWriter(file, fieldnames=["name", "phone", "age", "email"])  ResFile.writeheader()  ResFile.writerows(TestingData)  LData = FileUtils.LoadFile(str(PathFile), CommandList())  assert [vars(student) for student in LData.students] == TestingData  def test\_SaveFile(TestingData, tmp\_path, capsys):  PathFile = tmp\_path / "test\_save\_file.csv"  StudentL = CommandList()  StudentL.students = [StudentUnit(\*\*data) for data in TestingData]  FileUtils.SaveFile(str(PathFile), StudentL)  captured = capsys.readouterr()  assert "The data is saved\n" in captured.out  LData = FileUtils.LoadFile(str(PathFile), CommandList())  assert [vars(student) for student in LData.students] == [vars(student) for student in StudentL.students]  def test\_updateElement(tmp\_path, capsys, TestingData):  with patch('builtins.input', side\_effect=["John", "123", "30", "john@example.com"]):  StudentL = CommandList()  StudentL.students = [StudentUnit(\*\*TestingData[0])]  with patch('sys.stdout', new\_callable=StringIO) as TestInputAns:  StudentL.updateElement("Bob")  captured = TestInputAns.getvalue()  assert "Element has been updated" in captured  assert vars(StudentL.students[0]) == {"name": "John", "phone": "123", "age": 30, "email": "john@example.com"}  assert sorted(StudentL.students, key=lambda x: (x.name, x.age)) == StudentL.students  def test\_addNewElement(tmp\_path, TestingData):  StudentL = CommandList()  with patch('builtins.input', side\_effect=["Test", "123", "25", "test@example.com"]):  StudentL.addNewElement()  assert len(StudentL.students) == 1  assert vars(StudentL.students[0]) == {"name": "Test", "phone": "123", "age": 25, "email": "test@example.com"}  assert sorted(StudentL.students, key=lambda x: (x.name, x.age)) == StudentL.students  with patch('builtins.input', side\_effect=["AAA", "456", "30", "123@example.com"]):  StudentL.addNewElement()  assert len(StudentL.students) == 2  assert vars(StudentL.students[0]) == {"name": "AAA", "phone": "456", "age": 30, "email": "123@example.com"}  assert sorted(StudentL.students, key=lambda x: (x.name, x.age)) == StudentL.students  with patch('builtins.input', side\_effect=["Test", "222", "35", "test@example.com"]):  StudentL.addNewElement()  assert len(StudentL.students) == 3  assert vars(StudentL.students[2]) == {"name": "Test", "phone": "222", "age": 35, "email": "test@example.com"}  assert sorted(StudentL.students, key=lambda x: (x.name, x.age)) == StudentL.students  def test\_deleteElement(capsys, TestingData):  with patch('builtins.input', return\_value="John"):  StudentL = CommandList()  StudentL.students = [StudentUnit(\*\*TestingData[0])]  with patch('sys.stdout', new\_callable=StringIO) as TestInputAns:  StudentL.deleteElement("John")  captured = TestInputAns.getvalue()  assert "Element was not found\n" in captured  def test\_printAllList(capsys, TestingData):  StudentL = CommandList()  StudentL.students = [StudentUnit(\*\*TestingData[0])]  with patch('sys.stdout', new\_callable=StringIO) as TestInputAns:  StudentL.printAllList()  captured = TestInputAns.getvalue()  assert "Student name is Bob, Phone is 1112233, Age is 23, Email is bob@example.com" in captured |