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.

**Хід виконання завдання:**

**Створення класу Student**

* Реалізовано клас Student з атрибутами: ім'я, телефон, email, вік.
* Додано метод \_\_str\_\_() для зручного виводу інформації про студента.

Створення класу StudentList

Реалізовано методи для роботи зі списком студентів:

add\_student() – додавання нового студента.

remove\_student() – видалення студента за ім'ям.

update\_student() – оновлення інформації про студента.

display\_students() – виведення списку студентів.

**Створення класу Utils**

* Реалізовано методи для роботи з файлами:

load\_from\_file() – завантаження даних зі students.csv.

save\_to\_file() – збереження списку студентів у файл.

**Реалізація main.py**

* Завантаження списку студентів із файлу.
* Демонстрація основних операцій (додавання, оновлення, видалення).
* Збереження змін у файл після завершення роботи.

**Організація коду**

* Кожен клас збережено у відповідному файлі:

student.py – клас Student.

student\_list.py – клас StudentList.

utils.py – клас Utils.

main.py – запуск програми.

**Написання юніт-тестів**

* Розроблено тести для перевірки методів роботи зі студентами та файлами.

Текст програми:

Main.py -

|  |
| --- |
| from utils import Utils  from student import Student  def main():    student\_list = Utils.load\_from\_file("students.csv")  print("\nТекущий список студентов:")  student\_list.display\_students()    print("\nДобавляем нового студента...")  student\_list.add\_student(Student("Иван Иванов", "123-456", "[ivan@example.com](mailto:ivan@example.com)", 20))  print("\nСписок студентов после добавления:")  student\_list.display\_students()    print("\nОбновляем данные студента...")  student\_list.update\_student("Иван Иванов", phone="789-012", [email="ivan\_ivanov@example.com](mailto:email="ivan_ivanov@example.com)")  print("\nСписок студентов после обновления:")  student\_list.display\_students()    print("\nУдаляем студента...")  student\_list.remove\_student("Иван Иванов")  print("\nСписок студентов после удаления:")  student\_list.display\_students()    print("\nСохраняем изменения...")  Utils.save\_to\_file("students.csv", student\_list)  print("\nДанные сохранены. Конечный список студентов:")  student\_list.display\_students()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Student.py-

|  |
| --- |
| class Student:  def \_\_init\_\_(self, name: str, phone: str, email: str, age: int):  self.name = name  self.phone = phone  self.email = email  self.age = age  def \_\_str\_\_(self):  return f"{self.name}, {self.age} лет, Телефон: {self.phone}, Email: {self.email}" |

student\_list.py -

|  |
| --- |
| from student import Student  class StudentList:  def \_\_init\_\_(self):  self.students = []  def add\_student(self, student: Student):  self.students.append(student)  def remove\_student(self, name: str):  self.students = [s for s in self.students if s.name != name]  def update\_student(self, name: str, \*\*kwargs):  for student in self.students:  if student.name == name:  student.name = kwargs.get("name", student.name)  student.phone = kwargs.get("phone", student.phone)  student.email = kwargs.get("email", student.email)  student.age = kwargs.get("age", student.age)  def display\_students(self):  for student in self.students:  print(student) |

Посилання на github:

<https://github.com/worhels/lab/tree/main/lab_KB-232/lab_03>

Знімок екрану з посилання на github:

