

Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



Звіт

до лабораторної роботи №9

З дисципліни: «Кросплатформні засоби програмування»

На тему: «ОСНОВИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ У PYTHON»

Варіант 5

Виконала:

ст. гр. КІ-305

Гринь С.М.

Прийняв:

Іванов Ю.С.

Львів 2023

Мета роботи: оволодіти навиками реалізації парадигм об'єктно-орієнтованого програмування використовуючи засоби мови Python.

Завдання:

1. Написати та налагодити програму на мові Python згідно варіанту. Програма має задовольняти наступним вимогам:
 - класи програми мають розміщуватися в окремих модулях в одному пакеті;
 - точка входу в програму (main) має бути в окремому модулі;
 - мають бути реалізовані базовий і похідний класи предметної області згідно варіанту;
 - програма має містити коментарі.
2. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
3. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
4. Дати відповідь на контрольні запитання.

Варіант 5

Базовий клас: Машина;

Похідний клас: Вантажна машина.

Код програми:

Код у файлі machine.py

```
class Machine:
```

```
    def __init__(self, make, model, year):
```

```
        self.make = make # Виробник
```

```
        self.model = model # Модель
```

```
        self.year = year # Рік випуску
```

```
    def start(self):
```

```
        print(f'{self.year} {self.make} {self.model} почала рух.")
```

```
    def stop(self):
```

```
        print(f'{self.year} {self.make} {self.model} зупинилася.")
```

Код у файлі truck.py

```
from machine import Machine # Імпортуємо базовий клас
```

```
class Truck(Machine):
```

```
    def __init__(self, make, model, year, payload_capacity):
```

```
        super().__init__(make, model, year) # Викликаємо конструктор базового класу
```

```

        self.payload_capacity = payload_capacity # Вантажопідйомність

    def load(self):
        print(f"Вантажна машина з вантажопідйомністю {self.payload_capacity} кг завантажуються.")

    def unload(self):
        print("Розвантаження вантажної машини.")

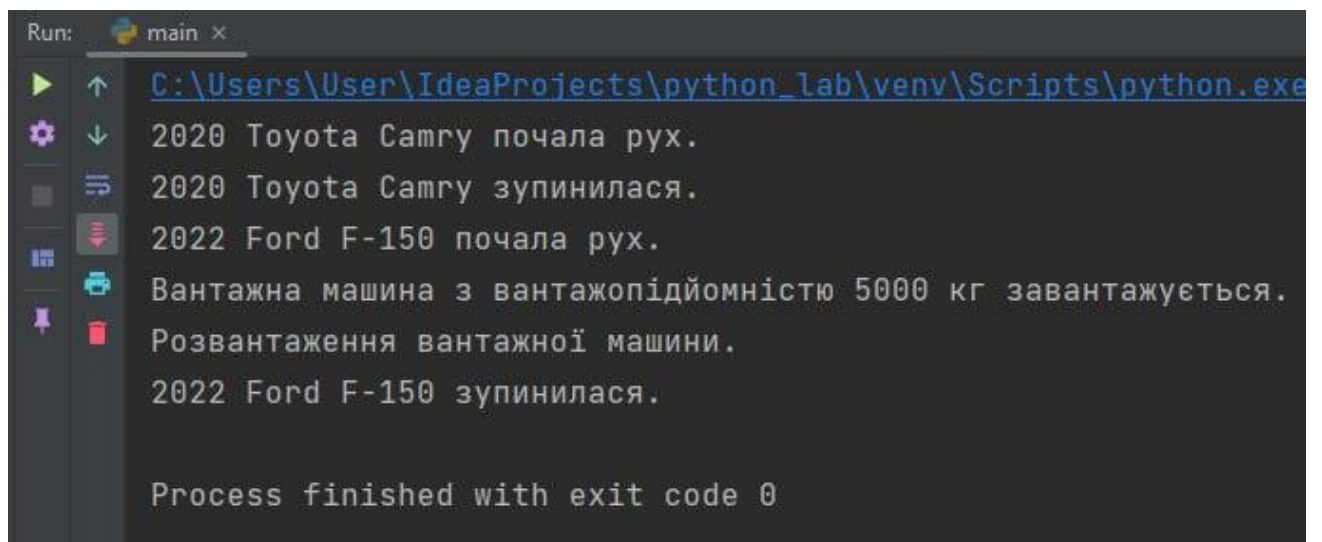
# Код у файлі main.py
from machine import Machine
from truck import Truck

if __name__ == "__main__":
    car = Machine("Toyota", "Camry", 2020) # Створюємо екземпляр базового класу
    car.start()
    car.stop()

    truck = Truck("Ford", "F-150", 2022, 5000) # Створюємо екземпляр похідного класу
    truck.start()
    truck.load()
    truck.unload()
    truck.stop()

```

Результат роботи програми:



```

Run: main x
C:\Users\User\IdeaProjects\python_lab\venv\Scripts\python.exe
2020 Toyota Camry почала рух.
2020 Toyota Camry зупинилася.
2022 Ford F-150 почала рух.
Вантажна машина з вантажопідйомністю 5000 кг завантажуються.
Розвантаження вантажної машини.
2022 Ford F-150 зупинилася.

Process finished with exit code 0

```

Контрольні запитання

1. Що таке модулі?

Модулем у Python називається файл з розширенням *.py. Ці файли можуть містити звичайні скрипти, змінні, функції, класи і їх комбінації. Python дозволяє структурувати код програм у різні модулі та доступатися до класів, функцій і змінних, які у них знаходяться з інших модулів. Для цього використовуються два оператори – `import` та `from-import`.

2. Як імпортувати модуль?

Оператор `import` дозволяє імпортувати модуль повністю, та доступатися до нього через назву модуля. Вона може бути вказана у будь-якому місці програми перед звертанням до елементів, які у ній містяться, але зазвичай її вказують на початку модуля. Для звертання до елементів модуля треба вказати назву модуля і після крапки вказати до якого елементу ви хочете звернутися.

Синтаксис

```
import назва_модуля
назва_модуля.елемент_модуля
```

3. Як оголосити клас?

Клас оголошується за допомогою ключового слова `class` після якого йде назва класу.

4. Що може міститися у класі?

Клас може містити:

- дані, які належать класу (статичні дані-члени класу);
- дані, які належать об'єкту класу;
- методи, які належать класу (статична методи);
- методи, які належать об'єкту класу

5. Як називається конструктор класу?

У мові програмування Python конструктор класу називається `__init__`. Цей метод викликається автоматично при створенні нового об'єкта класу і використовується для ініціалізації атрибутів об'єкта. Наприклад:

class МійКлас:

```
def __init__(self, атрибут1, атрибут2):
    self.атрибут1 = атрибут1
    self.атрибут2 = атрибут2
об'єкт = МійКлас(значення1, значення2)
```

6. Як здійснити спадкування?

Спадкування призначене для розширення функціональності існуючих класів шляхом утворення нових класів на базі вже існуючих. У Python усі класи спадкуються неявно від класу `object`. Python дозволяє реалізовувати як одинарне так і множинне спадкування. Для реалізації спадкування класи, які слід успадкувати вказуються у круглих дужках через кому після назви класу, який оголошується:

```
class <назва_класу> (<базовий_клас_1>, <базовий_клас_2>, ...):  
    <тіло класу>
```

7. Які види спадкування існують?

У мові програмування Python існують два основних види спадкування:

1.Однорівневе спадкування (Single Inheritance): Це найпростіший вид спадкування, де клас успадковує властивості та методи від одного базового класу. Один клас може мати лише один безпосередній батьківський клас. Наприклад:

```
python Copy code  
  
class БатьківськийКлас:  
    def метод_батьківського_класу(self):  
        pass  
  
class ДочірнійКлас(БатьківськийКлас):  
    def метод_дочірнього_класу(self):  
        pass
```

2.Багаторівневе спадкування (Multiple Inheritance): Python підтримує спадкування від більше, ніж одного класу. Це означає, що клас може успадковувати властивості і методи від декількох базових класів. Наприклад:

```
python Copy code  
  
class БатьківськийКлас1:  
    def метод_батьківського_класу1(self):  
        pass  
  
class БатьківськийКлас2:  
    def метод_батьківського_класу2(self):  
        pass  
  
class ДочірнійКлас(БатьківськийКлас1, БатьківськийКлас2):  
    def метод_дочірнього_класу(self):  
        pass
```

Це дозволяє створювати більш складні ієрархії класів, але також може призводити до конфліктів імен методів, які потрібно розв'язувати.

Крім цього, в Python є також можливість множинного спадкування, коли клас успадковує властивості від декількох класів одночасно, а також спадкування від вбудованих класів, таких як **object**, який є базовим класом для всіх інших класів в Python.

8. Які небезпеки є при множинному спадкуванні, як їх уникнути?

Множинне спадкування в Python може призводити до деяких проблем і небезпек. Основні проблеми, пов'язані з множинним спадкуванням, включають:

1. **Конфлікти імен методів і атрибутів:** Якщо багато базових класів мають методи чи атрибути з однаковими іменами, це може призвести до конфліктів. При виклику такого методу не завжди очевидно, який саме метод буде викликаний. Це може призвести до непередбачуваної поведінки.
2. **Складність розуміння і обслуговування коду:** Багаторівневе спадкування може зробити код складним і важким для розуміння, особливо якщо велика кількість класів участь у спадкуванні.

Для уникнення цих проблем і зменшення небезпек множинного спадкування в Python, можна вживати наступні стратегії:

1. **Використовуйте композицію замість спадкування:** Замість спадкування можна використовувати композицію, коли один клас вкладається в інший як атрибут. Це дозволяє створювати більш гнучкі та менш зв'язані класи.
2. **Інтерфейси і абстрактні класи:** Визначаєте інтерфейси і використовуйте абстрактні класи для вимушення деяких поведінок у дочірніх класах. Це допоможе зробити код більш зрозумілим і впевнитися, що всі класи, які успадковують, реалізують певні методи.
3. **Використовуйте `super()`:** При виклику методів базових класів, використовуйте функцію `super()`, щоб вказати, з якого саме базового класу слід викликати метод. Це допоможе уникнути конфліктів імен методів.
4. **Намагайтесь уникати занадто глибокого спадкування:** Спробуйте уникати занадто глибокого ланцюга спадкування, оскільки це може ускладнити код та зробити його вразливим на зміни.

Дотримуючись цих стратегій, ви можете знизити ризики, пов'язані з множинним спадкуванням в Python і зробити свій код більш читабельним та підтримуваним.

9. Що таке класи-домішки?

Домішки або Mixin – це шаблон проектування, в якому деякий метод базового класу використовує метод, який не визначається у цьому класі. Цей метод призначений для реалізації іншим базовим класом. Клас-домішка або mixin class – це клас, який використовується у цьому шаблоні, надаючи функціональні можливості (методи), але не призначений для самостійного використання у вигляді об'єктів класу. В ідеальному випадку класи-домішки не мають власної ієрархії спадкування і не мають полів, а мають лише методи.

10. Яка роль функції `super()` при спадкуванні?

Функція `super()` в Python грає важливу роль у контексті спадкування. Вона використовується для виклику методів базового класу в класі-нащадку. Роль `super()` полягає в тому, щоб

дозволити класам-нащадкам спільно використовувати методи базового класу, розширюючи або змінюючи їх функціональність.

Основні призначення функції **super()**:

1. **Дозволяє викликати методи базового класу:** Завдяки **super()**, ви можете викликати методи базового класу в класі-нащадку, щоб використовувати функціональність, яку вони надають. Це допомагає уникнути дублювання коду та забезпечує гнучкість в розробці.
2. **Забезпечує правильний порядок виклику методів у багаторівневому спадкуванні:** У випадку багаторівневого спадкування, де клас-нащадок успадковує властивості від декількох базових класів, **super()** допомагає визначити, з якого саме базового класу слід викликати метод. Це важливо для збереження правильного порядку виконання методів у ланцюгу спадкування.
3. **Дозволяє розширювати функціональність базового класу:** Клас-нащадок може викликати методи базового класу за допомогою **super()**, а потім розширити їх функціональність, додавши власний код до методу. Це сприяє перевизначенню методів базового класу. Ось приклад використання функції **super()**:

```
python Copy code

class БатьківськийКлас:
    def метод(self):
        print("Метод з базового класу")

class ДочірнійКлас(БатьківськийКлас):
    def метод(self):
        super().метод() # Виклик методу з базового класу
        print("Метод з класу-нащадка")

дочірній_об'єкт = ДочірнійКлас()
дочірній_об'єкт.метод()
```

У цьому прикладі функція **super().метод()** дозволяє викликати метод базового класу, а потім додавати власний код до методу класу-нащадка.

Висновок: оволоділа навиками реалізації парадигм об'єктно-орієнтованого програмування використовуючи засоби мови Python.