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 3**

Виконав: ст. гр. КІ-305

Гнідець В.М.

Прийняв:

Іванов Ю.С.

Львів 2023

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

**Завдання:**

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 3:** *Пес*

**Код програми**

**Arctic\_dog.py**

from dog import Dog  
class ArcticDog(Dog):  
 def \_\_init\_\_(self, name, breed, country):  
 *"""  
 Ініціалізує собаку з ім'ям, породою, і областю в якій вона допомагає лікарям.  
 Успадковуєтсья від класу dog  
 """* super().\_\_init\_\_(name, breed)  
 self.country = country  
 self.helping\_level = 0  
  
  
 def train(self, hours):  
 *"""Симулює, що арктична собака тренуєтсья і підвищує свій рівень навичок."""* self.helping\_level += hours \* 3  
 return f"{self.name} тренувався {hours} годин. Рівень навичок: {self.helping\_level}"  
  
 #def getHealht(self):  
 #return f"{self.name} збільшив на {self.health} очок здоров'я"  
  
 def perform\_duty(self):  
 *"""Симулює, що собака виконує завданя і витрачає енергію."""* if self.energy\_level >= 30 and self.helping\_level >= 50:  
 self.energy\_level -= 30  
 self.helping\_level -= 50  
 return f"{self.name} успішно виконав завдання. Рівень енергії: {self.energy\_level}. Рівень навичок: {self.helping\_level}"  
 else:  
 return f"{self.name} не готовий виконати завадання."

**Dog\_health.py**

class dog\_health:  
 def \_\_init\_\_(self, health):  
 *"""Створює собаку з ім'ям і породою."""* self.health = health  
  
  
 def getHealht(self):  
 return "Я здоровий!"

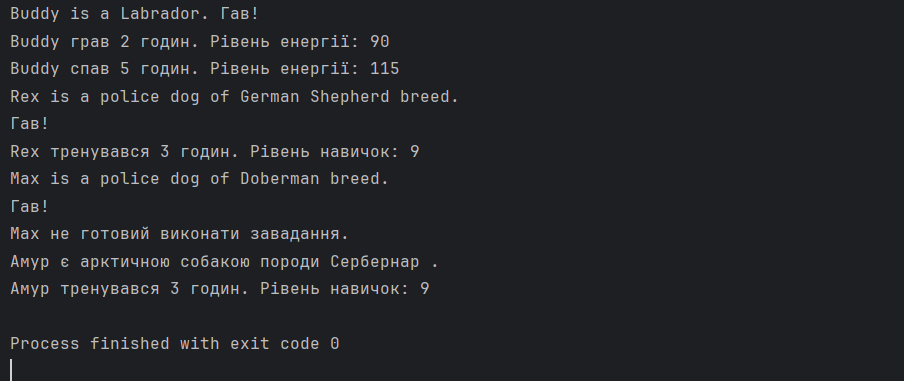
**Dog.py**

class Dog:  
 def \_\_init\_\_(self, name, breed):  
 *"""Створює собаку з ім'ям і породою."""* self.name = name  
 self.breed = breed  
 self.energy\_level = 100  
  
 def bark(self):  
 *"""Повертає звук який робить собака."""* return "Гав!"  
  
 def play(self, hours):  
 *"""Симулює, що собака грає і витрачає енергію."""* energy\_consumed = hours \* 5  
 if self.energy\_level >= energy\_consumed:  
 self.energy\_level -= energy\_consumed  
 return f"{self.name} грав {hours} годин. Рівень енергії: {self.energy\_level}"  
 else:  
 return f"{self.name} дуже змучений щоб грати."  
  
 def sleep(self, hours):  
 *"""Симулює, що собака спить і відновлює енергію."""* self.energy\_level += hours \* 5  
 return f"{self.name} спав {hours} годин. Рівень енергії: {self.energy\_level}"

**Main.py**

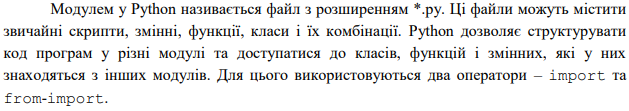
from dog import Dog  
from arctic\_dog import ArcticDog  
from dog\_health import dog\_health  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 dog1 = Dog("Buddy", "Labrador")  
 print(f"{dog1.name} is a {dog1.breed}. {dog1.bark()}")  
 print(dog1.play(2)) # Собака грає 2 години  
 print(dog1.sleep(5)) # Собака спить 5 годин  
  
# Створюємо першу піддослідну собаку  
arctic\_dog1 = ArcticDog("Rex", "German Shepherd", "K9 Unit")  
print(f"{arctic\_dog1.name} is a police dog of {arctic\_dog1.breed} breed.")  
print(arctic\_dog1.bark())  
print(arctic\_dog1.train(3)) # Поліцейська собака навчається 3 години  
  
# Створюємо другу піддослідну собаку, яка успішно виконує обов'язки  
arctic\_dog2 = ArcticDog("Max", "Doberman", "K9 Unit")  
arctic\_dog2.energy\_level = 40 # Налаштовуємо рівень енергії  
arctic\_dog2.training\_level = 60 # Налаштовуємо рівень навчання  
print(f"{arctic\_dog2.name} is a police dog of {arctic\_dog2.breed} breed.")  
print(arctic\_dog2.bark())  
print(arctic\_dog2.perform\_duty()) # Поліцейська собака успішно виконує обов'язок  
  
# Створюємо першу піддослідну собаку  
police\_dog1 = ArcticDog(f"Амур", "Сербернар", "Підніжжя західних Альп")  
print(f"{police\_dog1.name} є арктичною собакою породи {police\_dog1.breed} .")  
print(police\_dog1.train(3)) # Арктична собака навчається 3 години

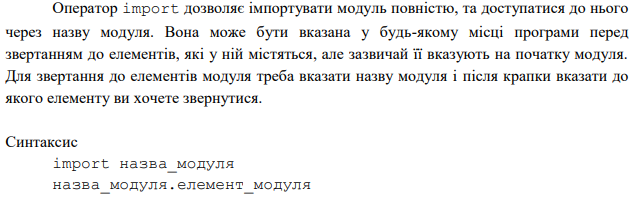
**Результат роботи програми:**

****

**Контрольні запитання**

***1. Що таке модулі?***

 ***2. Як імпортувати модуль?***

******

***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:***

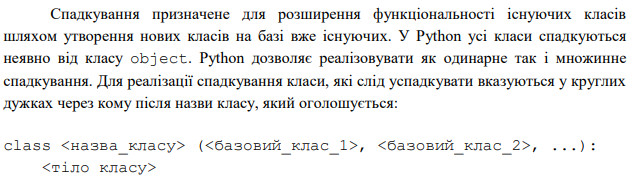
***def \_\_init\_\_(self, атрибут1, атрибут2):***

***self.атрибут1 = атрибут1***

***self.атрибут2 = атрибут2***

***об'єкт = МійКлас(значення1, значення2)***

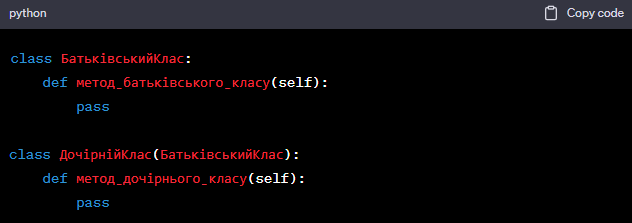
***6. Як здійснити спадкування?***

******

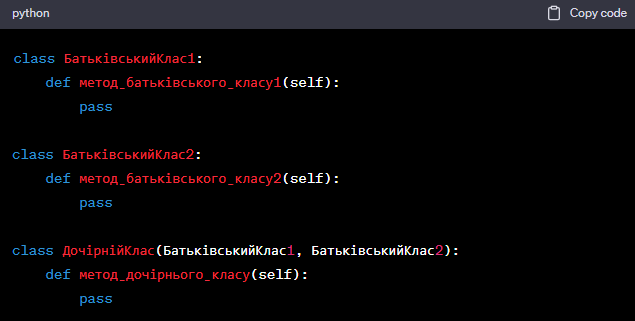
***7. Які види спадкування існують?***

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:

******

**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:

******

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**: Визначайте 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 ім'ям 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.

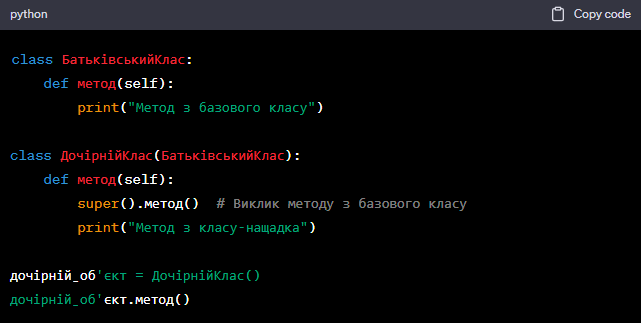
***10. Яка роль функції super() при спадкуванні?***

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()**:

******

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

Початок форми

**Висновок:** оволодів навиками реалізації парадигм об’єктно-орієнтованого програмування використовуючи засоби мови Python.