Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**з дисципліни**

**Кросплатформні засоби програмування**

**Звіт з Лабораторної роботи №** 9

**ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО**

**ПРОГРАМУВАННЯ У PYTHON**

**Виконав:**

КІ-307

Герега Р. О.

**Перевірив:**

Іванов Ю.С

**2023**

**Мета роботи:** оволодіти навиками реалізації парадигм об’єктно-орієнтованого

програмування використовуючи засоби мови Python.

**Теоретичний вступ**

**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.

Оператор import

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

Приклад

Файл my\_print\_module.py

def hello\_print():

print(“Hello”)

Файл printer.py

import my\_print\_module

my\_print\_module.hello\_print ()

Для зручності оригінальну назву модуля можна змінити на псевдонім і доступатися

до елементів модуля за допомогою псевдоніма.

Синтаксис

import оригінальна\_назва\_модуля as псевдонім

Приклад

import my\_print\_module as my\_printe.

**Класи**

Клас оголошується за допомогою ключового слова class після якого йде назва

класу. 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.

Члени класу є лише публічні, проте Python забезпечує механізми, які дозволяють

організувуати області видимості близькі за своєю суттю до protected і private. Це робиться

шляхом використання нижнього підкреслення у назві членів класу. Одинарне нижнє

підкреслення перед назвою члену класу робить за своїми властивостями схожим на

захищений член класу, а подвійне – схожим на приватний член класу. Всі члени класу, що

йому належать мають відступ у розмірі одного табулятора, або 4-ох пробілів від початку

оголошення класу.

Статичні члени-дані класу оголошуються в класі як назва змінної і її початкове

значення.

Дані, які належать об’єкту класу оголошуються в конструкторі з використанням

ключового слова self, яке є посиланням на об’єкт класу:

self.<назва\_змінної> = <початкове значення>

Статичні методи класу оголошуються в класі за правилами оголошення функцій.

Не статичні методи оголошуються в класі за правилами оголошення функцій, перший

параметр якої є обов’язково self:

def <назва\_методу>(self, <параметри>):

тіло методу

Роль конструктора відіграє метод \_\_init\_\_(self, <параметри>).

Доступ до статичних членів класу відбуваєтсья за допомогою назви класу:

<назва класу>.<назва статичного члену класу>.

Доступ до не статичних членів класу відбуваєтсья за допомогою назви об’єкту:

<назва об’єкту>.<назва не статичного члену класу>.

Параметри self передавати у метод при виклику не потрібно. Він передається

неявно як і у інших мовах програмування.

Якщо клас не містить ніяких членів, то він має мати замість них ключову слово

pass

**Варіант 6**



**Завдання:**

1. Написати та налагодити програму на мові Python згідно варіанту. Програма має

задовольняти наступним вимогам:

• програма має розміщуватися в окремому модулі;

• програма має реалізувати функції читання/запису файлів у текстовому і

двійковому форматах результатами обчислення виразів 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 доку

**Код**

from datetime import datetime

class Plane:

def \_\_init\_\_(self, make, model):

self.make = make

self.model = model

self.fuel\_level = 100 # Початковий рівень пального

def fly(self):

if self.fuel\_level > 0:

print(f"[{datetime.now()}][{self.make}][{self.model}] is flying")

self.fuel\_level -= 10

else:

print("Not enough fuel. Please refuel.")

def land(self):

print(f"[{datetime.now()}][{self.make}][{self.model}] is landing")

def refuel(self):

print(f"[{datetime.now()}][{self.make}][{self.model}] is being refueled")

self.fuel\_level = 100 # Повне заповнення пального

def check\_fuel\_level(self):

print(f"[{datetime.now()}][{self.make}][{self.model}] fuel level: {self.fuel\_level}%")

from plane import Plane

from datetime import datetime

class BomberPlane(Plane):

def \_\_init\_\_(self, make, model, weapon):

super().\_\_init\_\_(make, model)

self.weapon = weapon

self.bombs = 10 # Початкова кількість боєприпасів

def drop\_bombs(self):

if self.bombs > 0:

print(f"[{datetime.now()}][{self.make}][{self.model}] is drop bombs")

self.bombs -= 1

else:

print("Out of bombs. Please reload.")

def defend(self):

print(f"[{datetime.now()}][{self.make}][{self.model}] is defending")

def reload\_weapon(self):

print(f"[{datetime.now()}][{self.make}][{self.model}] weapon is being reloaded")

self.bombs = 10 # Повне заповнення боєприпасів

def check\_ammo\_count(self):

from plane import Plane

from bomber import BomberPlane

def main():

# Створення екземпляра базового класу "Літак"

# Створення екземпляра базового класу "Літак"

plane1 = Plane("Boeing", "747")

plane1.fly()

plane1.check\_fuel\_level()

plane1.land()

plane1.refuel()

plane1.check\_fuel\_level()

# Створення екземпляра похідного класу "Військовий літак"

fighter\_plane = BomberPlane("F-16", "Fighting Falcon", "Missiles")

fighter\_plane.fly()

fighter\_plane.check\_fuel\_level()

fighter\_plane.drop\_bombs()

fighter\_plane.check\_ammo\_count()

fighter\_plane.defend()

fighter\_plane.reload\_weapon()

fighter\_plane.check\_ammo\_count()

if \_\_name\_\_ == "\_\_main\_\_":

main()

print(f"[{datetime.now()}][{self.make}][{self.model}] bombs count: {self.bombs}")

**Результат:**

[2023-12-08 01:05:47.791913][Boeing][747] is flying

[2023-12-08 01:05:47.791913][Boeing][747] fuel level: 90%

[2023-12-08 01:05:47.791913][Boeing][747] is landing

[2023-12-08 01:05:47.791913][Boeing][747] is being refueled

[2023-12-08 01:05:47.791913][Boeing][747] fuel level: 100%

[2023-12-08 01:05:47.791913][F-16][Fighting Falcon] is flying

[2023-12-08 01:05:47.791913][F-16][Fighting Falcon] fuel level: 90%

[2023-12-08 01:05:47.791913][F-16][Fighting Falcon] is drop bombs

[2023-12-08 01:05:47.791913][F-16][Fighting Falcon] bombs count: 9

[2023-12-08 01:05:47.791913][F-16][Fighting Falcon] is defending

[2023-12-08 01:05:47.791913][F-16][Fighting Falcon] weapon is being reloaded

[2023-12-08 01:05:47.791913][F-16][Fighting Falcon] bombs count: 10

**Короткий висновок**

Під час виконання даної лабораторної роботи я оволодіd навиками реалізації парадигм об’єктно-орієнтованого

програмування використовуючи засоби мови Python.