ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Кафедра систем управління літальними апаратами

Лабораторна робота № 4

з дисципліни «Об'єктно-орієнтоване проектування СУ»

Тема: «Реалізація класу і робота з об'єктами»

ХАІ.301 . 3.320.4 ЛР

Виконав студент гр. 320

_____Семеняга Ігор___
(підпис, дата) (П.І.Б.)

Перевірив

_____ к.т.н., доц. О. В. Гавриленко
(підпис, дата) (П.І.Б.)

2024

МЕТА РОБОТИ

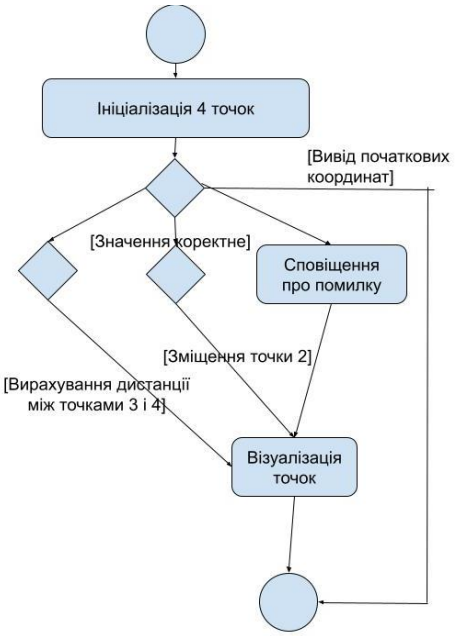
Застосувати теоретичні знання з основ програмування на мові Python з використанням об'єктів і класів, навички використання бібліотеки для візуалізації масивів даних, і навчитися розробляти скрипти для роботи з об'єктами призначених для користувача класів.

ПОСТАНОВКА ЗАДАЧІ

Завдання 1. Визначити клас Point_n, який реалізує
абстракцію з атрибутами:
1) дві дійсні координати точки на площині (властивості, приховані змінні екземпляра),
− для кожної метод-геттер (повертає відповідну координату),
− для кожної метод-сеттер (записує відповідну координату, якщо вона у межах [-100, 100], інакше – дорівнює 0))
2) кількість створених екземплярів точки (змінна класу),
3) метод класу (повертає кількість створених примірників),
4) конструктор з двома параметрами (за замовчуванням),
5) деструктор, що виводить відповідне повідомлення,
6) метод, що змінює координати точки з двома вхідними дійсними параметрами:
− зсув по x,
− зсув по y.
Завдання 2. Виконати операції з об'єктами даного класу.
Завдання 3. Використовуючи пакет matplotlib, відобразити створені об'єкти в графічному вікні до і після змін.
Завдання 4. Зберегти координати точок у текстовому файлі у форматі:

ВИКОНАННЯ РОБОТИ

Завдання 1. Вирішення задачі 1, №4



Створити список з чотирьох точок, порахувати відстань між четвертою та третьою, пересунути другу на 17 вліво.

Вхідні дані (ім’я, опис, тип, обмеження):

Point_n(10, 20), int

Point_n(30, 40), int

Point_n(-50, 60), int

Point_n(70, -80), int

Вихідні дані (ім’я, опис, тип):

print(f"Відстань між точками 4 і 3: {distance:.2f}") float

print("\nКоординати точок збережено у файл 'points.txt'."), txt

Алгоритм вирішення

Рисунок 1 – Алгоритм роботи завдання

Лістинг коду вирішення задачі наведено в дод. А (стор. 7). Екран роботи програми показаний на рис. Б.8.

ВИСНОВКИ

Було застосовано теоретичні знання з основ програмування на мові Python, зокрема використання об'єктів і класів. Закріплено навички роботи з бібліотекою для візуалізації масивів даних, таких як Matplotlib чи Seaborn. Отримано практичний досвід розробки скриптів для створення, модифікації та використання об'єктів призначених для користувача класів, що дозволило краще зрозуміти принципи об'єктно-орієнтованого програмування.

ДОДАТОК А
Лістинг коду програми до задач №11

```
from point import Point_n
import matplotlib.pyplot as plt

def main():
    # Створення чотирьох точок
    points = [
        Point_n(10, 20),
        Point_n(30, 40),
        Point_n(-50, 60),
        Point_n(70, -80)
    ]

    # Виведення координат початкових точок
    print("Координати початкових точок:")
    for i, point in enumerate(points, start=1):
        print(f"Точка {i}: ({point.x}, {point.y})")

    # Обчислення відстані між четвертою та третьою точками
    distance = points[3].distance_to(points[2])
    print(f"Відстань між точками 4 і 3: {distance:.2f}")

    # Переміщення другої точки на 17 вліво
    points[1].move(-17, 0)

    # Виведення координат після змін
    print("\nКоординати після змін:")
    for i, point in enumerate(points, start=1):
        print(f"Точка {i}: ({point.x}, {point.y})")

    # Візуалізація точок до та після змін
    x_initial = [10, 30, -50, 70]
    y_initial = [20, 40, 60, -80]
    x_after = [point.x for point in points]
    y_after = [point.y for point in points]

    plt.figure(figsize=(10, 5))

    # Початкові точки
    plt.subplot(1, 2, 1)
    plt.scatter(x_initial, y_initial, color='blue', label='Початкові точки')
    for i, (x, y) in enumerate(zip(x_initial, y_initial), start=1):
        plt.text(x, y, f" {i}", fontsize=9)
    plt.title("Початкові точки")
    plt.grid()
    plt.legend()

    # Точки після змін
```

```
    plt.subplot(1, 2, 2)
    plt.scatter(x_after, y_after, color='red', label='Точки після змін')
    for i, (x, y) in enumerate(zip(x_after, y_after), start=1):
        plt.text(x, y, f" {i}", fontsize=9)
    plt.title("Точки після змін")
    plt.grid()
    plt.legend()

    plt.show()

    # Збереження точок у текстовий файл
    with open("points.txt", "w") as f:
        for i, point in enumerate(points, start=1):
            f.write(f"({i}) {point.x}:{point.y}\n")
    print("\nКоординати точок збережено у файл 'points.txt'.")

if __name__ == "__main__":
    main()
```

```
import math

class Point_n:
    """
    Клас для представлення точки на площині з атрибутами:
    - координати x і y
    - кількість створених екземплярів класу
    """

    __instances_count = 0  # Змінна класу для підрахунку екземплярів

    def __init__(self, x=0, y=0):
        """
        Конструктор класу. Приймає координати точки.
        """
        self.__x = 0
        self.__y = 0
        self.x = x  # Використовуємо сеттер
        self.y = y  # Використовуємо сеттер
        Point_n.__instances_count += 1

    def __del__(self):
        """
        Деструктор класу. Зменшує кількість екземплярів та виводить
повідомлення.
        """
        Point_n.__instances_count -= 1
        print("Об'єкт з координатами ({self.__x}, {self.__y}) було видалено.")
```

```
@property
def x(self):
    """Геттер для координати x."""
    return self.__x

@x.setter
def x(self, value):
    """Сеттер для координати x."""
    self.__x = value if -100 <= value <= 100 else 0

@property
def y(self):
    """Геттер для координати y."""
    return self.__y

@y.setter
def y(self, value):
    """Сеттер для координати y."""
    self.__y = value if -100 <= value <= 100 else 0

@classmethod
def get_instance_count(cls):
    """
    Метод класу для отримання кількості екземплярів.
    """
    return cls.__instances_count

def move(self, dx, dy):
    """
    Метод для зміщення точки на dx та dy.
    """
    self.x += dx
    self.y += dy

def distance_to(self, other):
    """
    Метод для обчислення відстані до іншої точки.
    """
    return math.sqrt((self.x - other.x) ** 2 + (self.y - other.y) ** 2)
```

ДОДАТОК Б

Скрін-шоти вікна виконання програми

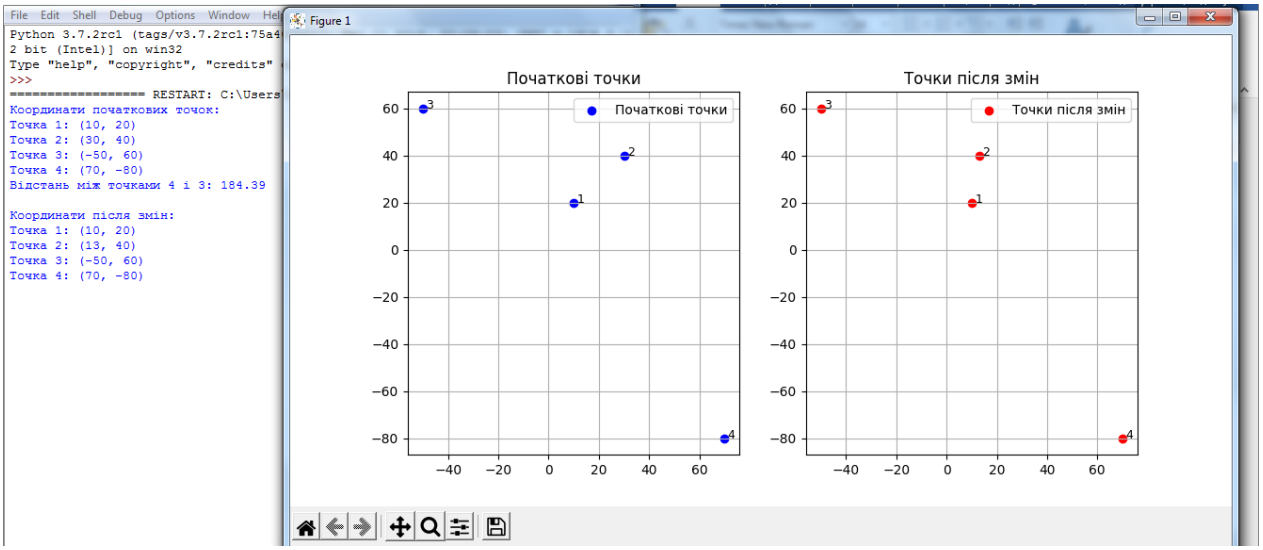


Рисунок Б.1 – Екран виконання програми для вирішення завдання 1-№4