МІНІСТЕРСТВО 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 програм для мобільних систем»

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

ХАІ.301 .151 .322 .4 ЛР

Виконав студент гр. _____322_____

_____Сироватський Дмитро_
(підпис, дата) (П.І.Б.)

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

2023

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 (n – номер варіанту), 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 відповідно до варіанту (див. таб.1).

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:
номер: координата_x; координата_y – для непарних варіантів
(номер) координата_x:координата_y – для парних варіантів

ВИКОНАННЯ РОБОТИ

Вирішення задачі task_23

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

- x (Координата x): Це координата x точки на площині. Тип даних - дійсне число (Float). Обмеження: має бути в діапазоні [-100, 100].
- y (Координата y): Це координата y точки на площині. Тип даних - дійсне число (Float). Обмеження: має бути в діапазоні [-100, 100].
- dx (Зсув по x): Це зсув по осі x. Тип даних - дійсне число (Float). Немає обмежень.
- dy (Зсув по y): Це зсув по осі y. Тип даних - дійсне число (Float). Немає обмежень.
- point1, point2 (Точки): Це об’єкти класу Point_n. Тип даних - об’єкт класу Point_n.

Вихідні дані (ім’я, опис, тип):

- dist (Відстань): Це відстань між двома точками. Тип даних - дійсне число (Float).
- Point_n.count (Кількість точок): Це кількість створених екземплярів класу Point_n. Тип даних - ціле число (Integer).
- Point_n.x, Point_n.y (Координати точки): Це координати точки на площині після зсуву. Тип даних - дійсне число (Float).
- Графік: Це візуалізація точок на площині. Тип даних - графік.

Лістинг коду вирішення задачі наведено в дод. А (стор. 5). Екран роботи програми показаний на рис. Б.1.(стор. 7)

ВИСНОВКИ

В результаті роботи було вивчено створення класів в Python, роботу з об’єктами та візуалізацію даних за допомогою matplotlib. Завдяки цьому отримано практичні навички об’єктно-орієнтованого програмування.

ДОДАТОК А
Лістинг коду програми до задач task_23

```
<

# Імпортуємо необхідні бібліотеки
import math
import matplotlib.pyplot as plt

# Визначаємо клас Point_n
class Point_n:
    # Змінна класу для відслідковування кількості створених екземплярів
    count = 0

    # Конструктор з двома параметрами (за замовчуванням)
    def __init__(self, x=0, y=0):
        #  Встановлюємо координати точки, перевіряючи їх на відповідність
діапазону [-100, 100]
        self.__x = self.__check_coordinate(x)
        self.__y = self.__check_coordinate(y)
        # Збільшуємо кількість створених екземплярів
        Point_n.count += 1

    # Деструктор, що виводить відповідне повідомлення
    def __del__(self):
        print("Об'єкт видалено")

    # Метод для перевірки координат на відповідність діапазону [-100, 100]
    def __check_coordinate(self, value):
        if -100 <= value <= 100:
            return value
        else:
            return 0

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

    # Сеттер для координати x
    @x.setter
    def x(self, value):
        self.__x = self.__check_coordinate(value)

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

    # Сеттер для координати y
```

```
    @y.setter
    def y(self, value):
        self.__y = self.__check_coordinate(value)

    # Метод класу, що повертає кількість створених екземплярів
    @classmethod
    def get_count(cls):
        return cls.count

    # Метод, що змінює координати точки
    def move(self, dx, dy):
        self.x += dx
        self.y += dy

# Функція для розрахунку відстані між двома точками
def distance(point1, point2):
    return math.sqrt((point1.x - point2.x)**2 + (point1.y - point2.y)**2)

# Створюємо список з трьох точок
points = [Point_n(10, 10), Point_n(20, 20), Point_n(30, 30)]

# Розраховуємо відстань між другою і третьою точкою
dist = distance(points[1], points[2])
print(f"Відстань між другою і третьою точкою: {dist}")

# Переміщаємо першу точку на 20 вниз і на 30 вправо
points[0].move(-20, 30)

# Візуалізуємо точки
x_values = [point.x for point in points]
y_values = [point.y for point in points]

plt.plot(x_values, y_values, 'ro')
plt.grid()
plt.show()
>
```

ДОДАТОК Б

Скрін-шоти вікна виконання програми

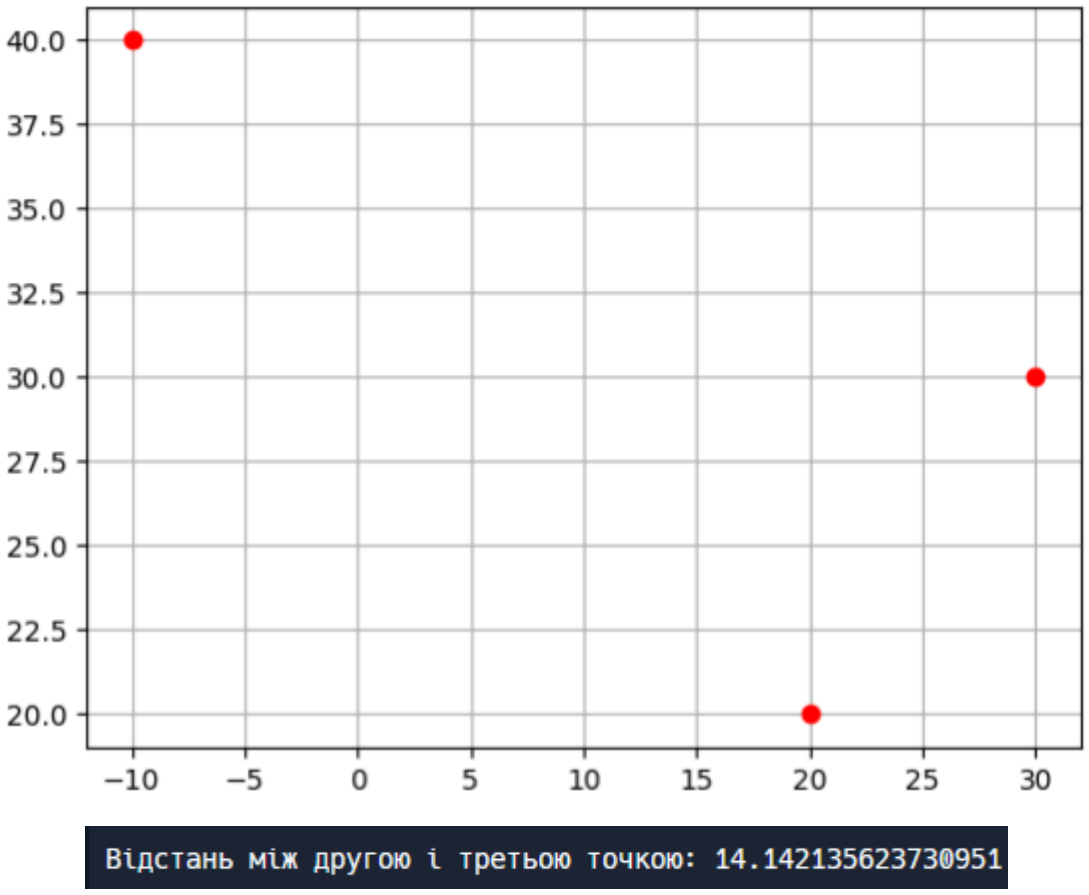

Рисунок Б.1 – Екран виконання програми для вирішення task_23