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 програмування СУ

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

ХАІ.301 . <спец.>. 322. 1 ЛР

Виконав студент гр. \_\_\_\_\_\_322\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кулагін Олексій\_\_\_

(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.у». 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 (записуює 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 х,

− зсув по у.

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\_х; координата\_у – для непарних варіантів

(номер) координата\_х:координата\_у – для парних варіантів

Код в лістингу програми повинен містити докладні коментарі !!!

У звіті повинно бути дві діаграми: Діаграма класів і діаграма активності

для основного сценарію роботи з об'єктами.

# ВИКОНАННЯ РОБОТИ

Завдання 1. Вирішення задачі 17  
Вхідні дані (ім’я, опис, тип, обмеження):

· point1: координати x та y першої точки (тип: float, діапазон значень для x і y: [-100, 100]).

· point2: координати x та y другої точки (тип: float, діапазон значень для x і y: [-100, 100]).

· point3: координати x та y третьої точки (тип: float, діапазон значень для x і y: [-100, 100]).

· dx, dy: значення зсуву для переміщення першої точки (тип: float, діапазон значень для зсуву: будь-яке значення).

Вихідні дані (ім’я, опис, тип):

· Кількість створених точок: ціле число (тип: int) — виводиться після створення точок.

· Відстань між другою та третьою точками: число з плаваючою точкою (тип: float).

· Нова позиція першої точки після зміщення: об'єкт типу Point\_17, що відображає координати після зміщення.

· Збереження координат точок у текстовому файлі: файл points.txt, що містить координати точок у форматі "номер: x; y" (для непарних варіантів) або "номер: x:y" (для парних варіантів).

Лістинг коду вирішення задачі наведено в дод. А.  
 Екран роботи програми показаний на рис. Б.

# ВИСНОВКИ Було вивчено синтаксис визначення і виклику функцій у Python та особливості роботи з послідовностями. Закріплено на практиці знання з документації бібліотеки numpy. Отримано навички реалізації бібліотеки функцій, які структурують вирішення завдань з підходом «згори – до низу», що відпрацьовано в коді програми. ДОДАТОК А

Лістинг коду програми до задач Point17

import math

import matplotlib.pyplot as plt # type: ignore

class Point\_17:

    # Лічильник для кількості екземплярів

    point\_count = 0

    def \_\_init\_\_(self, x=0, y=0):

        # Ініціалізація координат точки

        self.\_\_x = x

        self.\_\_y = y

        Point\_17.point\_count += 1  # Збільшуємо лічильник при створенні нового екземпляра

    def \_\_del\_\_(self):

        # Деструктор: зменшуємо лічильник при видаленні екземпляра

        Point\_17.point\_count -= 1

        print(f"Object with coordinates ({self.\_\_x}, {self.\_\_y}) destroyed!")

    @property

    def x(self):

        # Геттер для координати x

        return self.\_\_x

    @x.setter

    def x(self, value):

        # Сеттер для координати x з перевіркою меж

        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 з перевіркою меж

        self.\_\_y = value if -100 <= value <= 100 else 0

    @classmethod

    def get\_point\_count(cls):

        # Метод класу для отримання кількості точок

        return cls.point\_count

    def move(self, dx, dy):

        # Метод для зсуву точки на dx по осі x та dy по осі y

        self.x += dx

        self.y += dy

    def distance\_to(self, other):

        # Метод для розрахунку відстані між поточною точкою та іншою

        return math.sqrt((self.\_\_x - other.x) \*\* 2 + (self.\_\_y - other.y) \*\* 2)

    def \_\_str\_\_(self):

        # Строкове представлення точки

        return f"Point({self.\_\_x}, {self.\_\_y})"

# Створюємо список з трьох точок

point1 = Point\_17(10, 20)

point2 = Point\_17(30, 40)

point3 = Point\_17(50, 60)

# Виводимо кількість створених точок

print(f"Total points created: {Point\_17.get\_point\_count()}")

# Порахувати відстань між другою та третьою точками

distance = point2.distance\_to(point3)

print(f"Distance between point2 and point3: {distance}")

# Переміщаємо першу точку на 50 одиниць вгору

point1.move(0, 50)

print(f"After moving, point1 is at: {point1}")

# Виведемо кількість точок після змін

print(f"Total points after moving: {Point\_17.get\_point\_count()}")

# Створюємо графік до змін

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

plt.scatter([point1.x, point2.x, point3.x], [point1.y, point2.y, point3.y], color='blue')

plt.text(point1.x, point1.y, 'Point1', fontsize=12, ha='right')

plt.text(point2.x, point2.y, 'Point2', fontsize=12, ha='right')

plt.text(point3.x, point3.y, 'Point3', fontsize=12, ha='right')

plt.title('Points before moving')

plt.grid(True)

plt.show()

# Створюємо графік після змін

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

plt.scatter([point1.x, point2.x, point3.x], [point1.y, point2.y, point3.y], color='red')

plt.text(point1.x, point1.y, 'Point1', fontsize=12, ha='right')

plt.text(point2.x, point2.y, 'Point2', fontsize=12, ha='right')

plt.text(point3.x, point3.y, 'Point3', fontsize=12, ha='right')

plt.title('Points after moving')

plt.grid(True)

plt.show()

# Зберігаємо координати точок у файл

with open("points.txt", "w") as f:

    if Point\_17.get\_point\_count() % 2 != 0:  # Для непарних варіантів

        f.write(f"1: {point1.x}; {point1.y}\n")

        f.write(f"2: {point2.x}; {point2.y}\n")

        f.write(f"3: {point3.x}; {point3.y}\n")

    else:  # Для парних варіантів

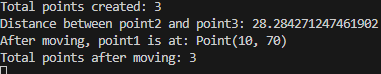
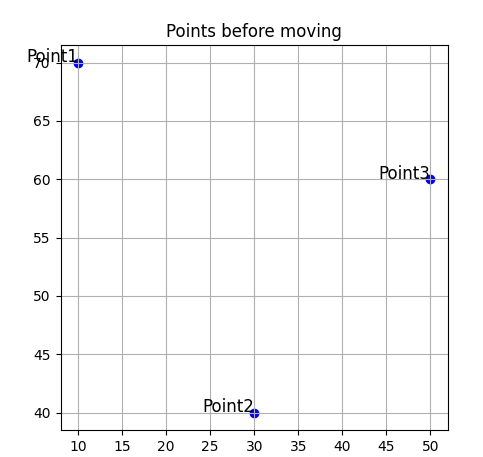
        f.write(f"1: {point1.x}:{point1.y}\n")

        f.write(f"2: {point2.x}:{point2.y}\n")

        f.write(f"3: {point3.x}:{point3.y}\n")

print("Coordinates saved to 'points.txt'.")

ДОДАТОК Б

Скрін-шоти вікна виконання програми Point17  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
ДОДАТОК Б.2

Скрін-шоти вікна виконання програми Point17  
