

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет
«Харківський авіаційний інститут»

Кафедра систем управління літальних апаратів

Практична робота №4

з дисципліни «Об'єктно-орієнтоване проектування СУ»

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

ХАІ.301. 175. 328. 08 ПР

Виконав студент гр. _____ 318

_____ Глєбова Каріна
(підпис, дата) (П.І.Б.)

Перевірив

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

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 користувацьких 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*.

Task 1. Визначити клас `Point_18`, 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,

б) 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).

Орег 18. Створити список з чотирьох точок, порахувати відстань між другою і четвертою, пересунути третю на 47 вгору.

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

Завдання 5. Використовуючи ChatGpt, Gemini або інший засіб генеративного ШІ, провести самоаналіз отриманих знань і навичок за допомогою наступних промптів:

1. «Ти - викладач, що приймає захист моєї роботи. Задай мені тестових питань з 4 варіантами відповіді і 5 відкритих питань. Це мають бути завдання <середнього> рівня складності на розвиток критичного та інженерного мислення. Питання мають відноситись до коду, що є у файлі звіту, і до теоретичних відомостей, що є у <презентації> лекції»
2. «Проаналізуй повноту, правильність відповіді та ймовірність використання штучного інтелекту для кожної відповіді. Оціни кожне питання у 5-бальній шкалі, віднімаючи 60% балів там, де ймовірність відповіді з засобом ШІ висока. Обчисли загальну середню оцінку»
3. «Проаналізуй код у звіті, і додай опис і приклади коду з питань, які є в теоретичних відомостях, але не реалізовано у коді при вирішенні завдань»

Проаналізуйте задані питання, коментарі і оцінки, надані ШІ. Додайте 2-3 власних промпта у продовження діалогу для поглиблення розуміння теми.

ВИКОНАННЯ РОБОТИ

Завдання 1.

Вирішення задачі Task 1.

Визначення класу Point_18, який реалізує абстракцію з атрибутами.

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

p1, p2, p3 – початкові точки класу Point_18;

Вихідні дані (ім'я, опис, тип):

(_repr_): Point_18(x, y) – точки після зміни, клас Point_18.

Діаграма активності завдання task_1 показана на рис. 1.

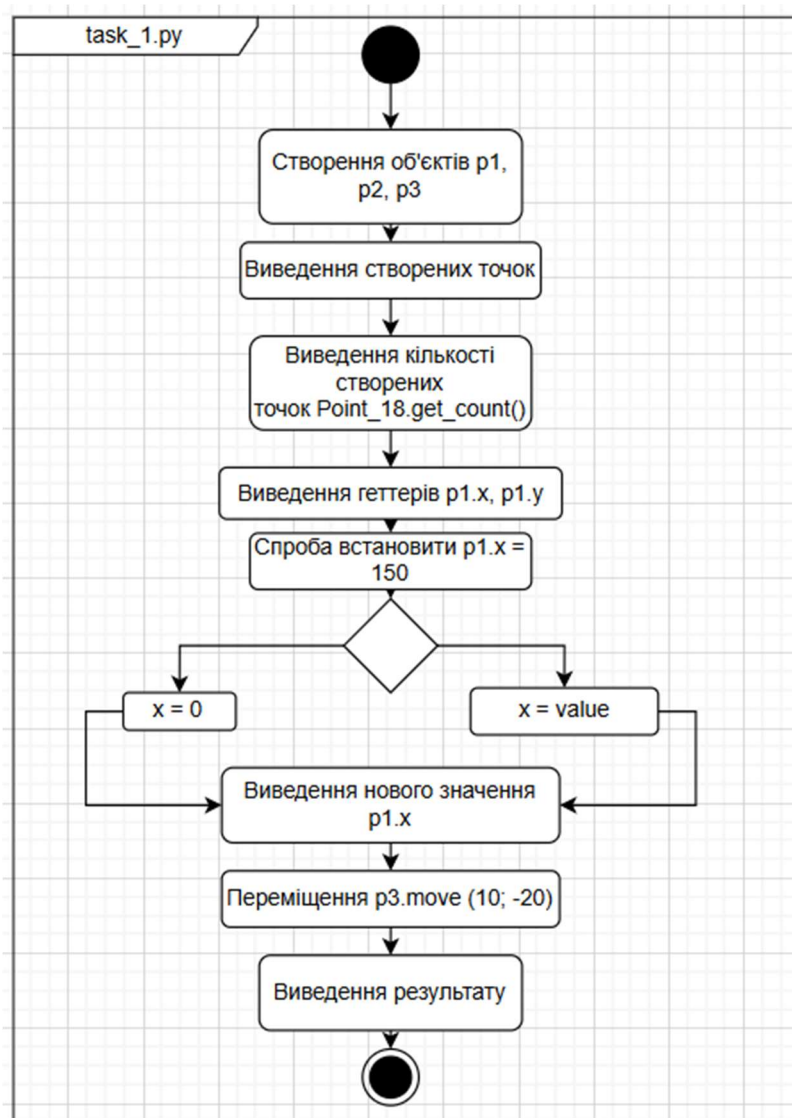


Рисунок 1 – Діаграма активності головної функції task_1

Лістинг коду вирішення задачі task_1.py наведено в дод. А (стор. 10-15).
Екран роботи програми показаний на рис. Б.1 (дод. Б, стор. 16-18)

Завдання 2.

Вирішення задачі Oper 18.

Створення списку з чотирьох точок та розрахунок відстані між другою та четвертою.

Вихідні дані (ім'я, опис, тип):

points_before – початкові координати точок (x, y), тип даних list(Points_18);

points_after – список змінених координат точок (x, y), тип даних list(Points_18);

dist – відстань між другою та четвертою точками у списку точок після зміни, тип float.

Діаграма активності завдання task_2 показана на рис. 2.

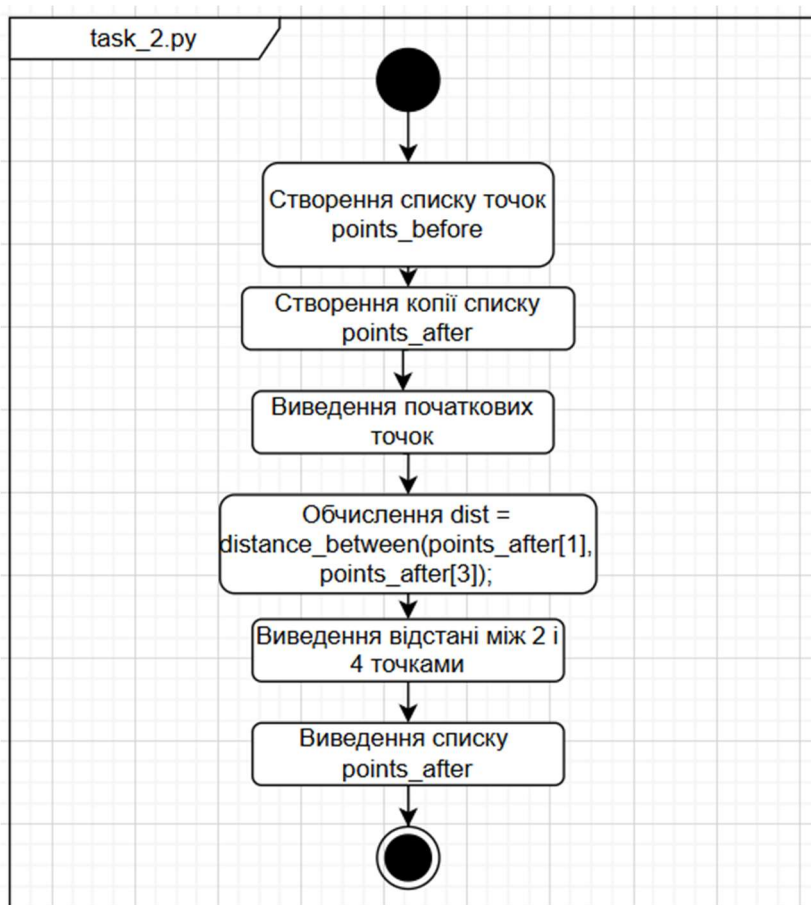


Рисунок 2 – Діаграма активності головної функції task_2

Лістинг коду вирішення задачі task_2.py наведено в дод. А (стор. 10-15).
Екран роботи програми показаний на рис. Б.2 (дод. Б, стор. 16-18)

Завдання 3.

Відображення створених об'єктів в графічному вікні до і після змін.

Вхідні дані (ім'я, опис, тип):

x_before – координати x точок до зміни, тип list(Points_18);

y_before – координати y точок до зміни, тип list(Points_18);

Вихідні дані (ім'я, опис, тип):

x_after – координати x точок після зміни, тип list(Points_18);

y_after – координати y точок після зміни, тип list(Points_18).

Діаграма активності завдання task_3 показана на рис. 3.

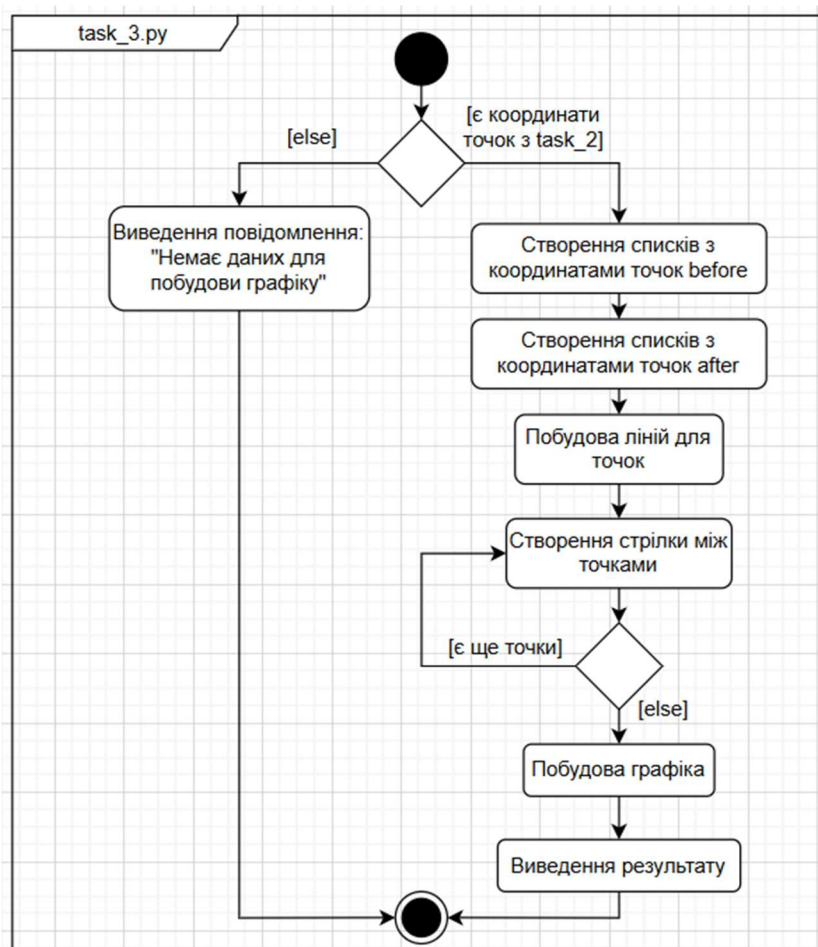


Рисунок 3 – Діаграма активності головної функції task_3

Лістинг коду вирішення завдання завдання 3 наведено в дод. А (стор. 10-15).

Екран роботи програми показаний на рис. Б.3, Б4. (дод. Б, стор. 16-18)

Завдання 4.

Збереження координат точок у текстовому файлі у форматі.

Вхідні дані(ім'я, опис, тип):

points точки, координати яких потрібно записати у вихідний файл, тип list(Points_18);

variant_number – номер варіанту, що використовується для визначення парності, дійсний тип, ціле число.

Вихідні дані(ім'я, опис, тип):

task_4_output.txt – вихідний текстовий файл із записаними координатами точок.

Діаграма активності завдання task_4 показана на рис. 4.

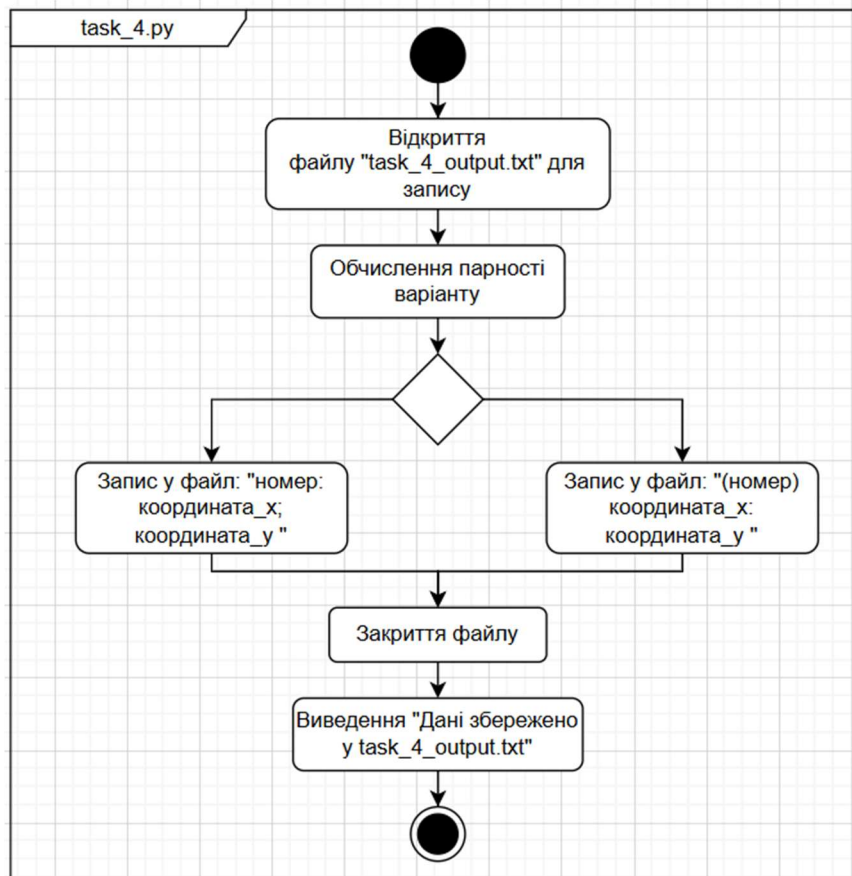


Рисунок 4 – Діаграма активності головної функції task_4

Лістинг коду вирішення завдання завдання 4 наведено в дод. А (стор. 10-15).

Екран роботи програми показаний на рис. Б.5, Б.6. (дод. Б, стор. 16-18)

Діаграма класів для всіх завдань наведена на рис. 5.

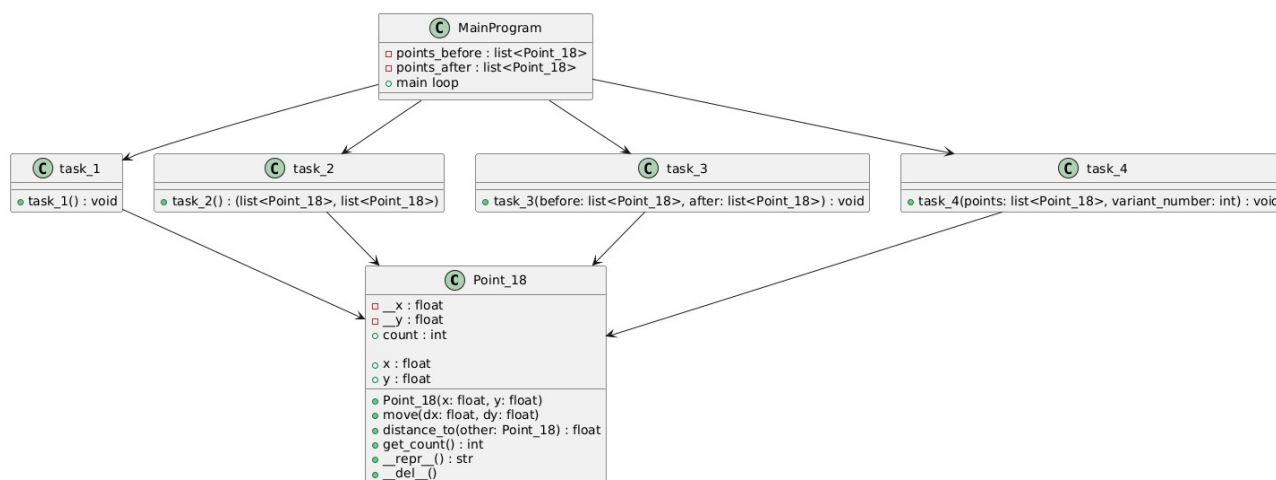


Рисунок 5 – Діаграма класів для всіх завдань

Діалог з ChatGPT наведено в дод. В (стор. 19-47)

ВИСНОВКИ

На практичній роботі було опрацьовано теоретичний матеріал з основ програмування на мові Python з використанням об'єктів і класів, а також застосовані навички використання бібліотеки для візуалізації масивів даних. Були реалізовано та закріплено на практиці розробка скриптів для роботи з об'єктами користувацьких класів.

ДОДАТОК А

Лістинг коду програми

```
#task_1.py

import math

class Point_18:
    """Клас моделює точку на координатній площині."""

    # Змінна класу – спільна для всіх створених об'єктів.
    # Використовується для підрахунку кількості екземплярів.
    count = 0

    def __init__(self, x=0.0, y=0.0):
        """
        Конструктор – викликається при створенні нового об'єкта.
        Встановлює координати x та y через сеттери.
        Також збільшує лічильник створених об'єктів.
        """
        self.x = x          # запис через setter (перевірка меж)
        self.y = y
        Point_18.count += 1 # збільшуємо лічильник об'єктів

    @property
    def x(self):
        """Повертає приховану змінну __x."""
        return self.__x

    @property
    def y(self):
        """Повертає приховану змінну __y."""
        return self.__y

    @x.setter
    def x(self, value):
        """
        Встановлює координату x.
        Якщо значення виходить за межі [-100, 100] – автоматично замінюється на 0.
        """
        if -100 <= value <= 100:
            self.__x = value
        else:
            self.__x = 0

    @y.setter
    def y(self, value):
        """
        Встановлює координату y.
        Те саме правило обмеження: дозволяються лише значення [-100; 100].
        """
        if -100 <= value <= 100:
            self.__y = value
        else:
            self.__y = 0

    def __del__(self):
```

```

    """
    Деструктор – викликається перед видаленням об'єкта.
    Зменшує лічильник об'єктів і виводить повідомлення.
    """
    Point_18.count -= 1
    print(f"Point_18 object deleted. Now count = {Point_18.count}")

def move(self, dx, dy):
    """
    Змінює координати точки.
    dx – зміщення по осі X
    dy – зміщення по осі Y
    """
    self.x = self.x + dx
    self.y = self.y + dy

def distance_to(self, other):
    """
    Обчислює відстань між двома точками за формулою:
    sqrt( (x1 - x2)^2 + (y1 - y2)^2 )
    """
    return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

def __repr__(self):
    """
    Повертає текстове представлення об'єкта.
    Використовується при друку в списках, відладці тощо.
    """
    return f"Point_18({self.x}, {self.y})"

@classmethod
def get_count(cls):
    """Повертає кількість створених екземплярів класу."""
    return cls.count

def task_1():
    """Демонстрація роботи класу Point_18 (завдання 1)."""

    print("\n--- Завдання 1: Демонстрація класу Point_18 ---")

    # Створення трьох точок
    p1 = Point_18(10, 20)      # звичайні координати
    p2 = Point_18(-150, 5)     # x виходить за межі → буде замінено на 0
    p3 = Point_18(50, 50)      # коректні значення

    print("Створені точки:", p1, p2, p3)

    # Виведення кількості створених точок
    print("Кількість створених точок:", Point_18.get_count())

    # Перевірка геттерів
    print("Перевірка геттерів:", p1.x, p1.y)

    # Спроба присвоїти некоректне значення x
    print("Спроба встановити x = 150 (поза межами)")
    p1.x = 150
    print("Новий x:", p1.x)    # має стати 0

    # Демонстрація роботи move()
    print("Переміщаємо третю точку на (10, -20)")

```

```

p3.move(10, -20)
print("p3 після зміни:", p3)

print("---- Кінець демонстрації завдання 1 ----")

```

```
# task_2.py
```

```
# Імпорт класу Point_18 з попереднього завдання
from task_1 import Point_18
```

```

def task_2():
    print("\n--- Завдання 2: Варіант 18 ---")

    # Створюємо список точок "до змін".
    # Кожен елемент – це окремий об'єкт Point_18 з відповідними координатами.
    points_before = [
        Point_18(10, 10),
        Point_18(-5, -20),
        Point_18(0, 0),
        Point_18(30, -7)
    ]

    # Створюємо список "після змін".
    points_after = [
        Point_18(p.x, p.y) for p in points_before
    ]

    # Виводимо початкові точки
    print("Початкові точки:", points_before)

    # Обчислюємо відстань між другою та четвертою точкою у списку points_after.
    # Індеси: 1 → друга точка, 3 → четверта.
    dist = points_after[1].distance_to(points_after[3])
    # Вивід відстані з округленням до трьох знаків після коми.
    print(f"Відстань між точками 2 і 4: {dist:.3f}")

    # Переміщуємо третю точку (індекс 2) на вектор (0, +47)
    # Тобто змінюється тільки координата y.
    points_after[2].move(0, 47)

    # Виводимо оновлені значення після переміщення
    print("Після зміни:", points_after)

    # Повертаємо обидва списки,
    # щоб інші частини програми могли використати їх
    return points_before, points_after

```

```
# task_3.py
```

```
import matplotlib.pyplot as plt
```

```

def task_3(before, after):
    """
    Має точки до (before) і після (after) змін.
    – before, after: списки об'єктів Point_18 (або сумісних)
    """

```

```

# Перевірка, чи передані списки точок.
# Якщо даних немає – графік не будується.
if before is None or after is None:
    print("Немає даних для побудови графіка.")
    return

print("\n--- Завдання 3: Графік ---")

# Витягуємо координати X і Y з початкових точок
x_before = [p.x for p in before]
y_before = [p.y for p in before]

# Витягуємо координати X і Y зі змінених точок
x_after = [p.x for p in after]
y_after = [p.y for p in after]

# Створюємо нову фігуру для графіка з розміром 8×6
plt.figure(figsize=(8, 6))

# Малюємо ламані лінії та точки "до змін"
# marker='o' – точки круглі
plt.plot(x_before, y_before, marker='o', linestyle='-', label="До змін")

# Малюємо ламані лінії та точки "після змін"
# marker='x' – точки хрестики
plt.plot(x_after, y_after, marker='x', linestyle='-', label="Після змін")

# Малюємо стрілки між відповідними точками "до" і "після"
# Це дозволяє наочно побачити, як кожна точка перемістилася.
# zip(before, after) автоматично обмежує ітерацію
# до найкоротшого списку (захист від різної довжини).
for b, a in zip(before, after):
    plt.annotate(
        '',
        xy=(a.x, a.y),          # координати нової точки
        xytext=(b.x, b.y),      # координати старої точки
        arrowprops=dict(arrowstyle='->', linewidth=0.8)
    )

# Включаємо сітку для кращого візуального сприйняття координат
plt.grid(True)

# Додаємо легенду ("До змін", "Після змін")
plt.legend()

# Підписи осей
plt.xlabel("X")
plt.ylabel("Y")

# Заголовок графіка
plt.title("Положення точок до і після зміни координат (лінії + стрілки)")

# Встановлюємо однаковий масштаб по обох осях,
# щоб стрілки та переміщення не спотворювалися
plt.axis('equal')

# Відображаємо графік на екрані
plt.show()

```

```
def task_4(points, variant_number):
    print("\n--- Завдання 4: Запис у файл ---")

    # Визначаємо, чи є номер варіанта парним.
    # Якщо парний → формат запису один, якщо ні → інший.
    is_even = (variant_number % 2 == 0)

    # Відкриваємо файл task_4_output.txt у режимі запису.
    # encoding="utf-8" – щоб українські символи зберігалися правильно.
    with open("task_4_output.txt", "w", encoding="utf-8") as f:

        # enumerate дає пари (номер_елемента, сам_елемент)
        # start=1 → нумерація починається з 1, а не з 0
        for i, p in enumerate(points, start=1):

            # Якщо варіант парний – один формат
            if is_even:
                # Формат: (1) x: y
                f.write(f"({i}) {p.x}: {p.y}\n")

            # Якщо варіант непарний – інший формат
            else:
                # Формат: 1: x; y
                f.write(f"{i}: {p.x}; {p.y}\n")

    # Після виходу з with файл автоматично закриється
    print("Дані збережено у task_4_output.txt")
```

```
#script-file
```

```
from task_1 import task_1
from task_2 import task_2
from task_3 import task_3
from task_4 import task_4

points_before = None
points_after = None

while True:
    print("\n=====")
    print("1 – Демонстрація класу Point_18 (Завдання 1)")
    print("2 – Виконати завдання 2 (Варіант 18)")
    print("3 – Показати графік точок (Завдання 3)")
    print("4 – Зберегти точки у файл (Завдання 4)")
    print("0 – Вийти")
    print("=====")

    choice = input("Виберіть задачу (0-4): ")

    if choice == "1":
        task_1()

    elif choice == "2":
        points_before, points_after = task_2()

    elif choice == "3":
        if points_before is None or points_after is None:
            print("Спочатку виконайте завдання 2!")
        else:
            task_3(points_before, points_after)
```

```
elif choice == "4":
    if points_after is None:
        print("Спочатку виконайте завдання 2!")
    else:
        task_4(points_after, 18)

elif choice == "0":
    print("Вихід з програми...")
    break

else:
    print("Невірний вибір.")
```

ДОДАТОК Б

Скрін-шоти вікна виконання програми

```
=====
1 - Демонстрація класу Point_18 (Завдання 1)
2 - Виконати завдання 2 (Варіант 18)
3 - Показати графік точок (Завдання 3)
4 - Зберегти точки у файл (Завдання 4)
0 - Вийти
=====
Виберіть задачу (0-4): 1

--- Завдання 1: Демонстрація класу Point_18 ---
Створені точки: Point_18(10, 20) Point_18(0, 5) Point_18(50, 50)
Кількість створених точок: 3
Перевірка геттерів: 10 20
Спроба встановити x = 150 (поза межами)
Новий x: 0
Переміщаємо третю точку на (10, -20)
p3 після зміни: Point_18(60, 30)
--- Кінець демонстрації завдання 1 ---
Point_18 object deleted. Now count = 2
Point_18 object deleted. Now count = 1
Point_18 object deleted. Now count = 0
```

Рисунок Б.1 – Екран виконання програми для вирішення завдання завдання 1

```
=====
1 - Демонстрація класу Point_18 (Завдання 1)
2 - Виконати завдання 2 (Варіант 18)
3 - Показати графік точок (Завдання 3)
4 - Зберегти точки у файл (Завдання 4)
0 - Вийти
=====
Виберіть задачу (0-4): 2

--- Завдання 2: Варіант 18 ---
Початкові точки: [Point_18(10, 10), Point_18(-5, -20), Point_18(0, 0), Point_18(30, -7)]
Відстань між точками 2 і 4: 37.336
Після зміни: [Point_18(10, 10), Point_18(-5, -20), Point_18(0, 47), Point_18(30, -7)]
```

Рисунок Б.2 – Екран виконання програми для вирішення завдання 2


```

=====
1 – Демонстрація класу Point_18 (Завдання 1)
2 – Виконати завдання 2 (Варіант 18)
3 – Показати графік точок (Завдання 3)
4 – Зберегти точки у файл (Завдання 4)
0 – Вийти
=====
Виберіть задачу (0-4): 3
Спочатку виконайте завдання 2!

```

Рисунок Б.3 – Екран виконання програми для завдань 3 або 4 без вирішення Завдання 2

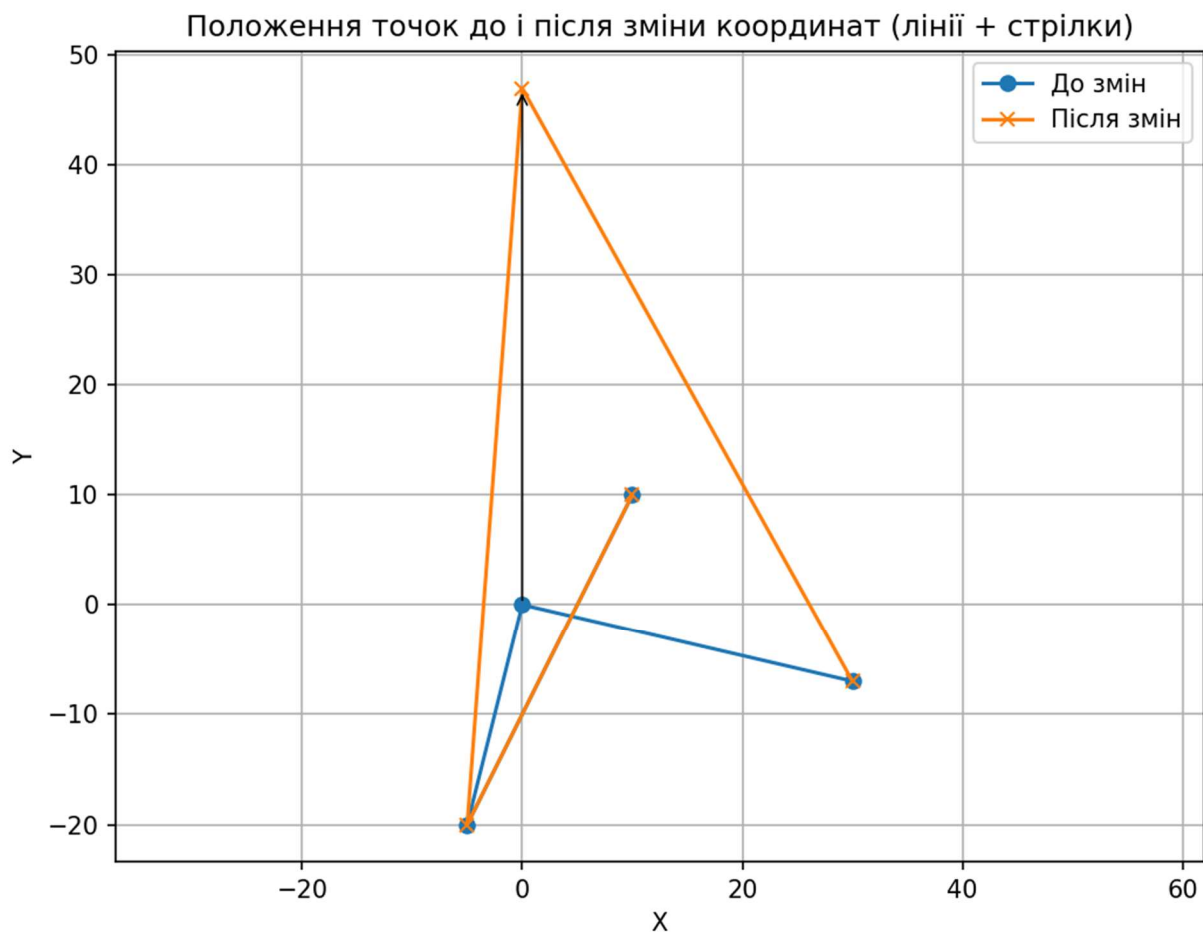
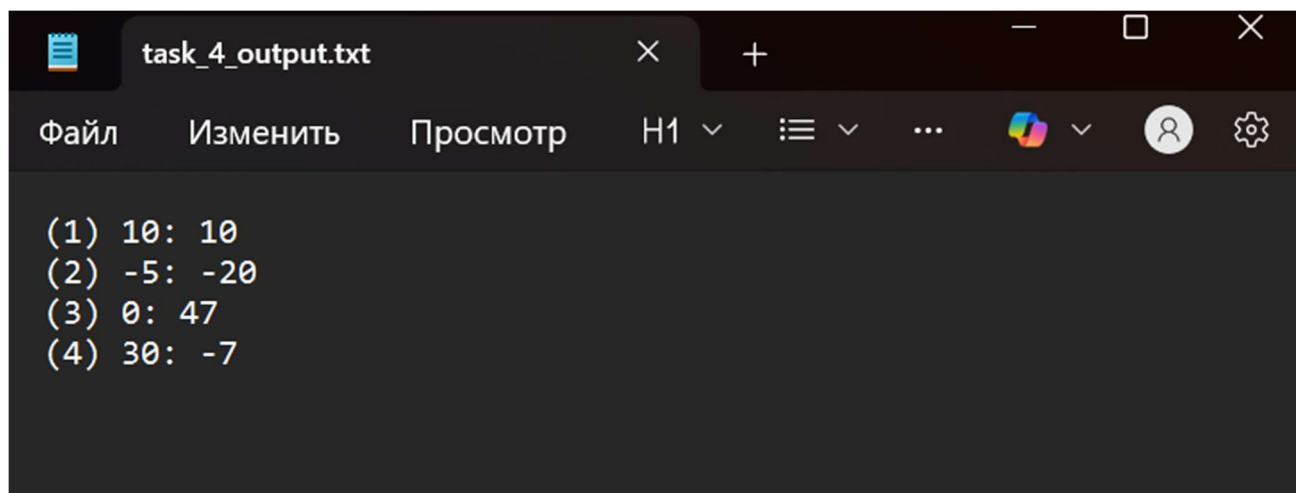


Рисунок Б.4 – Екран виконання програми для вирішення завдання 3

```
=====
1 - Демонстрація класу Point_18 (Завдання 1)
2 - Виконати завдання 2 (Варіант 18)
3 - Показати графік точок (Завдання 3)
4 - Зберегти точки у файл (Завдання 4)
0 - Вийти
=====
Виберіть задачу (0-4): 4

--- Завдання 4: Запис у файл ---
Дані збережено у task_4_output.txt
```

Рисунок Б.5 – Екран виконання програми для вирішення завдання 4



```
(1) 10: 10
(2) -5: -20
(3) 0: 47
(4) 30: -7
```

Рисунок Б.6 – Екран виконання програми для вирішення завдання 4

ДОДАТОК В

Діалог у ChatGPT для самоаналізу виконання роботи

«Ти - викладач, що приймає захист моєї роботи. Задай мені тестових питань з 4 варіантами відповіді і 5 відкритих питань. Це мають бути завдання <середнього> рівня складності на розвиток критичного та інженерного мислення. Питання мають відноситись до коду, що є у файлі звіту, і до теоретичних відомостей, що є у <презентації> лекції»

ChatGPT сказав:

Готово! Нижче — тестові питання (з 4 варіантами) та 5 відкритих питань, складені на основі коду з вашого звіту

Захист п.р. №4

та теорії з презентації

pyt_lect 4 dark

.

Рівень — **середній**, орієнтований на **критичне та інженерне мислення**.

✓ I. Тестові питання (10 шт., по 4 варіанти)

1. Що є прихованою (hidden) змінною у класі `Point_18`?

- A. `self.x` і `self.y`
 - B. `__x` і `__y`
 - C. `Point_18.count`
 - D. `dx` і `dy`
-

2. Який механізм ООП використовується у вашому класі для контролю доступу до координат?

- A. Наслідування
 - B. Поліморфізм
 - C. Інкапсуляція через властивості (`@property`)
 - D. Агрегація
-

3. Яку роль виконує метод `move()` у задачі 2?

- A. Створює копії точок
 - B. Змінює координати точки — модифікатор стану
 - C. Є селектором
 - D. Виводить об'єкт у консоль
-

4. Що повертає метод `distance_to()`?

- A. Скалярний добуток
 - B. Дистанцію за формулою Манхеттена
 - C. Евклідову відстань між двома точками
 - D. Кортеж зміщень
-

5. Чому при створенні `Point_18(-150, 5)` значення `x` стає 0?

- A. Виникає помилка
 - B. Спрацьовує деструктор
 - C. Спрацьовує `setter` і замінює значення поза `[-100, 100]`
 - D. Така поведінка вбудована в Python
-

6. Який принцип ООП реалізовано через лічильник `count` у класі?

- A. Агрегація
 - B. Класова змінна для спільного стану
 - C. Композиція
 - D. Поліморфізм
-

7. Що означає `__del__()` у Python згідно з презентацією?

- A. Перевантажений оператор
 - B. Метод для читання властивостей
 - C. Деструктор — викликається перед знищенням об'єкта
 - D. Модифікатор доступу
-

8. Який режим відкриття файлу використано в `task_4`?

- A. `"rb"`
 - B. `"w"`
 - C. `"r+"`
 - D. `"wb+"`
-

9. Який UML-елемент відповідає методу `move()`?

- A. Атрибут класу
 - B. Публічна операція (method)
 - C. Приватна змінна
 - D. Агрегований компонент
-

10. Що робить цикл у `task_3` з `annotate()`?

- A. Вимірює відстані
 - B. Малює стрілки переміщення для кожної точки
 - C. Зберігає точки у файл
 - D. Генерує нові координати
-

✓ II. Відкриті питання (5 шт.)

1. Поясніть, як у вашому класі реалізована інкапсуляція. Чому використано саме `@property` замість звичайних публічних змінних?

2. Наведіть приклад ситуації, де метод `distance_to()` можна було б розширити або перевантажити. Як це співвідноситься з поліморфізмом?

3. Чому у `task_2` створюється окремий список `points_after`, хоча початкові точки можна було б змінювати напрямую? Які ризики або помилки допомагає уникнути така практика?

4. Опишіть роль UML-діаграми класів у контексті вашого проєкту. Які її елементи відповідають частинам вашого коду?

5. Як би ви реалізували успадкування (inheritance) у вашому проєкті? Наведіть приклад похідного класу від `Point_18` з додатковою функціональністю.

Якщо потрібно — можу також:

- ✓ скласти **ключ відповідей**,
- ✓ створити **глибший аналіз**,
- ✓ згенерувати **наступні 2–3 промпти** (завдання 5),
- ✓ допомогти сформувати відповіді на ці питання.

Источники

Вы сказали:

Тести: 1. В, 2. С, 3. В, 4. С, 5. С, 6. В, 7. С, 8. В, 9. В, 10. В

Питання:

1. Інкапсуляція зроблена через клас із прихованими координатами x та y , а `@property` використовується для того, аби можна було автоматично виправляти неправильні значення.
2. Можна додати відстань у 3д вимірі.
3. Точки `points_after` - точки зі зміненими координатами, тобто вже зовсім іншакші точки від тих, що були (`points_before`). Вони потрібні також для того, щоб зробити графік, на якому видно їхню зміну положень.
4. UML-діаграма зображує структуру програми та взаємозв'язок між методами або навіть різними частина в цій програмі один між один.
5. -

ChatGPT сказав:

Нижче — **аналіз ваших відповідей**, оцінювання за 5-бальною шкалою (з урахуванням правила «*віднімати 60% за ймовірну участь III*»), а також загальна середня оцінка. Наприкінці — **поради щодо покращення відповідей**, включно з пунктом 5, який ви не дописали.

✓ I. Перевірка тестів (10 питань)

Ваші відповіді:

1. В — **вірно**
2. С — **вірно**
3. В — **вірно**
4. С — **вірно**
5. С — **вірно**
6. В — **вірно**
7. С — **вірно**
8. В — **вірно**
9. В — **вірно**
10. В — **вірно**

★ **Результат тестів: 10/10**

✓ II. Аналіз відкритих відповідей

Нижче — аналіз кожної відповіді:

- повнота
- правильність
- ймовірність використання ШІ (за змістом відповідей — низька/середня)
- оцінка (0–5) з відніманням 60% у разі підозри на ШІ

1. Питання про інкапсуляцію

Ваша відповідь:

Інкапсуляція зроблена через клас із прихованими координатами `x` та `y`, а `@property` використовується для того, аби можна було автоматично виправляти неправильні значення.

Аналіз:

- ✓ Правильно описані приховані змінні (`__x`, `__y`).
- ✓ Вірно підкреслено роль `@property`.
- ✚ Можна було додати згадку про *інтерфейсний доступ без зміни синтаксису*, але відповідь достатня.

Ймовірність ШІ: низька

Оцінка: 5/5

2. Питання про розширення `distance_to()`

Ваша відповідь:

Можна додати відстань у 3д вимірі.

Аналіз:

- ✓ Ідея коректна, показує розуміння поліморфізму/розширення.
- ✗ Але відповідь дуже коротка.
- ✗ Не пояснено, *як саме* це пов'язано з поліморфізмом (наприклад, створення підкласу `Point3D`, або перевантаження методу).

Ймовірність ШІ: низька

Оцінка: 3/5

3. Чому `points_after` створюється окремо

Ваша відповідь:

Це інші точки, потрібні для графіка, щоб бачити зміну положень.

Аналіз:

✓ Правильно: розділення «до» і «після».

✓ Правильно: важливо для візуалізації.

✚ Можна було додати: уникнення модифікації оригінальних даних, прозорість побудови графіка.

Ймовірність ШІ: низька

Оцінка: 4/5

4. Питання про UML-діаграму

Ваша відповідь:

UML-діаграма зображує структуру програми та взаємозв'язок між методами і частинами програми.

Аналіз:

✓ В цілому коректно.

✗ Відповідь загальна, не прив'язана до вашого конкретного класу (Point_18).

✗ Можна було згадати: атрибути, методи, рівні доступу.

Ймовірність ШІ: середня (формулювання загальне)

Оцінка (з урахуванням –60%): 2/5

5. Питання про успадкування (inheritance)

Ваша відповідь:

—

Аналіз:

Відповідь відсутня.

Оцінка: 0/5

✓ III. Загальна середня оцінка

Сума балів:

$$5 + 3 + 4 + 2 + 0 = \mathbf{14}$$

$$14 / 5 = \mathbf{2.8 / 5}$$

★ Загальна середня оцінка: **2.8 / 5**

✓ IV. Покращений варіант відповіді на питання 5 (успадкування)

Ось як можна було відповісти:

5. Як би ви реалізували успадкування у проєкті?

Можливий варіант:

```
class ColoredPoint(Point_18):
    def __init__(self, x=0, y=0, color="black"):
        super().__init__(x, y)
        self.color = color

    def __repr__(self):
        return f"ColoredPoint({self.x}, {self.y}, color='{self.color}')
```

Пояснення:

- Це клас-нащадок від Point_18.
 - Успадковує всі методи (move, distance_to, властивості).
 - Додає нову характеристику — колір.
 - Перевизначає __repr__, що демонструє **поліморфізм**.
-

Якщо хочете, можу також:

- ✓ сформулювати *вашу фінальну версію звіту* з цими відповідями
- ✓ підготувати наступні 2–3 промпти (як вимагалось у завданні)
- ✓ розібрати слабкі місця та покращити аргументацію

Сказати «згенеруй промпти»?

Вы сказали:

Розбери слабкі місця

ChatGPT сказав:

Нижче — **детальний розбір слабких місць у ваших відповідях**, з поясненнями, *чому саме це слабе місце і як його покращити*, щоб відповідь виглядала як у сильного студента-інженера, а не як у поспішного.

Розбір базується на вашому коді зі звіту та теорії з лекційних слайдів.

🔍 1. Питання про інкапсуляцію — слабкі місця

Ваш плюс:

Ви правильно описали загальний принцип приховування і логіку `@property`.

Слабкі місця:

1. Не згадано логіку перевірки діапазону — це ключове у вашому класі.
2. Не описано, що `@property` дає можливість **залишити однаковий синтаксис доступу**, але контролювати значення.
3. Не згадано про приховані змінні `__x` і `__y` в контексті *name mangling* — це є в лекції.

Як покращити:

- Уточнити, що інкапсуляція тут не просто «щось приховане», а **контроль цілісності даних**.
 - Пояснити, що `setter` не дає об'єкту перейти у некоректний стан.
-

🔍 2. Питання про розширення `distance_to()`

Ваш плюс:

Ідея з 3D логічна.

Слабкі місця:

1. Відповідь занадто коротка — не показує розуміння *поліморфізму* або *успадкування*.
2. Немає пояснення **як саме** це реалізувати у контексті Python та завдань.
3. Можна було згадати:
 - перевантаження методу в підкласі,
 - використання іншого алгоритму,
 - optional параметри.

Як покращити:

Додати приклад, наприклад:

створити підклас `Point3D`, який успадковує `Point_18` і перевизначає `distance_to()` — це демонструє поліморфізм.

Це одразу показує глибше розуміння.

🔍 3. Чому створюється `points_after` — слабкі місця

Ваш плюс:

Ви правильно описали призначення списку «після зміни».

Слабкі місця:

1. Не згадано, що зміна існуючого списку зруйнувала б можливість побудувати стрілки переміщення.
2. Не сказано про інженерний принцип «не модифікувати оригінальні дані».
3. Не описано, що копіюються *саме об'єкти*, а не посилання (хоч і глибоко не копіюється, це теж варто розуміти).

Як покращити:

Підкреслити важливість контролю станів — "до" і "після" — у візуалізації та тестуванні.

🔍 4. Питання про UML — слабкі місця

Ваш плюс:

Ви розумієте, що UML — це про структуру.

Слабкі місця:

1. Відповідь надто загальна — підходить під будь-яку тему.
2. У завданні потрібно було **пов'язати UML із вашим кодом**, а цього немає.
3. Не згадані елементи UML: атрибути, операції, доступність (public/private), зв'язки.

Як покращити:

Додати конкретику:

- `Point_18` → клас
- `__x`, `__y` → private attributes
- `move()`, `distance_to()` → public operations
- `count` → class attribute
- `get_count()` → class method

Тоді відповідь виглядає професійно, а не поверхово.

🔍 5. Питання про успадкування — слабкі місця

Ваш мінус:

Відповідь відсутня.

Це сильно знизило загальний бал.

Як покращити:

Дати приклад простого підкласу, наприклад:

- `ColoredPoint`
- `MovingPoint`
- `Point3D`
- `PointWithID`

І показати перевизначення хоч одного методу.

☞ Узагальнення слабких місць

Питання	Слабкі місця	Рівень
1	бракує глибини у поясненні property	незн ачне
2	відповідь надто коротка, без поліморфізму	помі тне
3	не враховані принципи інженерного дизайну	легк е
4	занадто загально, не прив'язано до коду	серй озне
5	відсутня відповідь	кри тичне

💬 Якщо хочеш, я можу:

- ✓ Запропонувати ідеальні версії ваших відповідей
- ✓ Переписати їх так, щоб виглядали повністю авторськими, без стороннього стилю
- ✓ Підготувати вас до «захисту» — з моделлю діалогу викладач ↔ студент

Хочеш, щоб я сформував покращений варіант відповідей для подальшого внесення у звіт?

Вы сказали:

Проаналізуй код у звіті, і додай опис і приклади коду з питань, які є в теоретичних відомостях, але не реалізовано у коді при вирішенні завдань

ChatGPT сказав:

Нижче — **глибокий аналіз вашого коду** та **додані описи й приклади програмного коду** для тих положень теорії, які є у лекції (PDF)

pyt_lect 4 dark

, але **не реалізовані у вашому проєкті** (клас `Point_18` та завдання 1–4) зі звіту

Захист п.р. №1

.

✓ 1. Що є в теорії, але відсутнє в коді

Згідно зі структурою лекцій, у теорії присутні такі важливі теми:

Тема	Наявність у вашому коді
Абстракція, інкапсуляція — ✓	реалізовано через <code>@property</code>
Приховані атрибути — ✓	<code>__x__, __y</code>
Класові атрибути — ✓	<code>count</code>
Конструктор та деструктор — ✓	<code>__init__, __del__</code>
Методи класу — ✓	<code>@classmethod</code> <code>get_count()</code>
Робота з файлами — ✓	<code>task_4()</code>

Але відсутні такі важливі елементи теорії:

- ✗ 1. Дескриптори (`__get__, __set__`)
 - ✗ 2. Успадкування (inheritance)
 - ✗ 3. Перевизначення методів (overriding)
 - ✗ 4. Поліморфізм (вузький: однакові назви методів у різних класах)
 - ✗ 5. Перевантаження операторів (`__add__, __eq__,` тощо)
 - ✗ 6. Робота з прихованими атрибутами через `__dict__, __name__, __module__`
 - ✗ 7. Приклад композиції/агрегації
 - ✗ 8. Розширені можливості matplotlib із лекцій (колір, форма маркерів)
-

🔥 Далі — опис і приклади коду, які можна включити у звіт для повноти теорії

✓ 2. Дескриптори (не реалізовано)

У лекції наведено приклади класів-дескрипторів (UserIDDescriptor, UserNameDescriptor). Ваш код не використовує дескриптори, але **можна продемонструвати їх на прикладі координати**.

📖 Приклад, який можна додати у звіт

```
class CoordDescriptor:
    """Дескриптор координат: значення за межами [-100; 100] замінюються на 0."""

    def __get__(self, obj, objtype=None):
        return obj.__dict__.get(self.attr_name)

    def __set__(self, obj, value):
        if -100 <= value <= 100:
            obj.__dict__[self.attr_name] = value
        else:
            obj.__dict__[self.attr_name] = 0

    def __set_name__(self, owner, name):
        self.attr_name = "_" + name

class PointWithDescriptor:
    x = CoordDescriptor()
    y = CoordDescriptor()

    def __init__(self, x, y):
        self.x = x
        self.y = y

p = PointWithDescriptor(150, 20)
print(p.x)    # 0
```

✓ 3. Успадкування (inheritance)

У теорії це розглядається на декількох слайдах, але у звіті цього немає.

☞ Приклад класу-нащадка

```
class ColoredPoint(Point_18):  
    """Похідний клас з додатковим параметром color."""  
  
    def __init__(self, x, y, color="black"):  
        super().__init__(x, y)  
        self.color = color  
  
    def __repr__(self):  
        return f"ColoredPoint({self.x}, {self.y}, '{self.color}')
```

✓ 4. Перевизначення методів (overriding)

У теорії є приклади:

- `myMethod()` у `Parent/Child`
- `__init__` у `Child` класі

Ваш код не містить прикладів overriding.

🔗 Приклад overriding у контексті вашого проекту

```
class LoggingPoint(Point_18):
    def move(self, dx, dy):
        print(f"Moving point from ({self.x}, {self.y}) "
              f"to ({self.x+dx}, {self.y+dy})")
        super().move(dx, dy)
```

✓ 5. Поліморфізм (однакові методи для різних класів)

В лекції зустрічається поняття поліморфізму, але воно не представлено у коді.

☞ Приклад

```
class Point2D:
    def dim(self):
        return 2

class Point3D:
    def dim(self):
        return 3

def print_dim(point):
    print("Dimension:", point.dim())

print_dim(Point2D()) # 2
print_dim(Point3D()) # 3
```

✓ 6. Перевантаження операторів (`__add__`, `__eq__`)

У теорії є розділ про default methods (`__str__`, `__repr__`, etc.) — ви використовуєте лише `__repr__`.

☞ Приклад, який можна додати у звіт

```
class Point_18_Extended(Point_18):
    def __add__(self, other):
        return Point_18_Extended(self.x + other.x, self.y + other.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

p1 = Point_18_Extended(5, 5)
p2 = Point_18_Extended(1, 2)
print(p1 + p2)          # Point_18_Extended(6, 7)
print(p1 == p2)         # False
```

✓ 7. Робота з прихованими атрибутами через built-in class attributes

У теорії є слайд з `__dict__`, `__doc__`, `__name__`, `__module__`.

Ваш код їх не демонструє.

🔗 Приклад

```
print(Point_18.__doc__)
print(Point_18.__module__)
print(Point_18.__dict__)
```

✓ 8. Агрегація / композиція (не реалізовано)

У теорії є розділ про відношення між об'єктами.

☞ Приклад “фігури з точок”

```
class Polygon:
    def __init__(self, points):
        self.points = points  # агрегація: полігон не створює точки

    def perimeter(self):
        total = 0
        for p1, p2 in zip(self.points, self.points[1:] + self.points[:1]):
            total += p1.distance_to(p2)
        return total
```

✓ 9. Matplotlib — приклад із лекції (більш розширений)

У вашому коді використано просту лінію/точки, але не використано:

- колір,
- різні маркери,
- підписування точок,
- стилі ліній.

☞ Приклад

```
plt.plot(x_before, y_before, 'cD-', label="Before")
plt.plot(x_after, y_after, 'mx:', label="After")
plt.grid()
plt.show()
```

☞ Висновок: що потрібно додати у звіт

Ви добре покрили практичну частину, але ваш звіт:

! потребує додаткових прикладів по ключовим розділам теорії:

1. Дескриптори
2. Наслідування
3. Перевизначення методів
4. Поліморфізм
5. Перевантаження операторів
6. `__dict__`, `__name__`, `__module__`
7. Агрегація/Композиція
8. Розширений `matplotlib`

Я вже підготував **готові приклади коду**, які можна:

- винести в **Додаток Г**,
- або додати в пояснювальну записку у вигляді окремого блоку "Розширення теоретичних відомостей".