

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

Кафедра систем управління літальними апаратами

Лабораторна робота № 2

з дисципліни «Об'єктно-орієнтоване програмування СУ»

Тема: «Розробка структурованих програм з розгалуженням та
повтореннями»

ХАІ.301 .174. 312ст.2 ЛР

Виконав студент гр. _____ 312ст

Твердохліб Максим Анатолійович

(підпис, дата)

(П.І.Б.)

Перевірів

_____ к.т.н., доц. О. В. Гавриленко

_____ ас. В. О. Білозерський

(підпис, дата)

(П.І.Б.)

2024

Лабораторна робота №2

З дисципліни «Об'єктно-орієнтоване програмування авіаційно-транспортних систем»

Тема: «Розробка структурованих програм з розгалуженням та повторенням»

МЕТА РОБОТИ

Вивчити теоретичний матеріал щодо синтаксису на мові Python і поданням у вигляді UML діаграм діяльності алгоритмів з розгалуження та циклами, а також навчитися використовувати функції, інструкції умовного переходу і циклів для реалізації інженерних обчислень.

ПОСТАНОВКА ЗАДАЧІ

Завдання 1. Вирішити завдання на алгоритми з розгалуженням.

Завдання 2. Дано дійсні числа (x_i, y_i) , $i = 1, 2, \dots, n$, – координати точок на площині. Визначити кількість точок, що потрапляють в геометричну область заданого кольору (або групу областей).

Завдання 3. Дослідити ряд на збіжність. Умова закінчення циклу обчислення суми прийняти у вигляді: $|u_n| < \epsilon$ або $|u_n| > G$ де ϵ – мала величина для переривання циклу обчислення суми сходиться ряду ($\epsilon = 10^{-5} \dots 10^{-20}$); g – величина для переривання циклу обчислення суми розходиться ряду ($g = 10^2 \dots 10^5$).

Завдання 4. Для багаторазового виконання будь-якого з трьох зазначених вище завдань на вибір розробити циклічний алгоритм організації меню в командному вікні.

ВИКОНАННЯ РОБОТИ

Завдання 1. Вирішення задачі (if) 13

Вхідні дані:

Користувач вводить три дійсних числа: a , b , c

Вихідні дані:

Програма виводить середнє з трьох чисел (тобто число, розташоване між найменшим і найбільшим)

Алгоритм вирішення показано на рис. 1

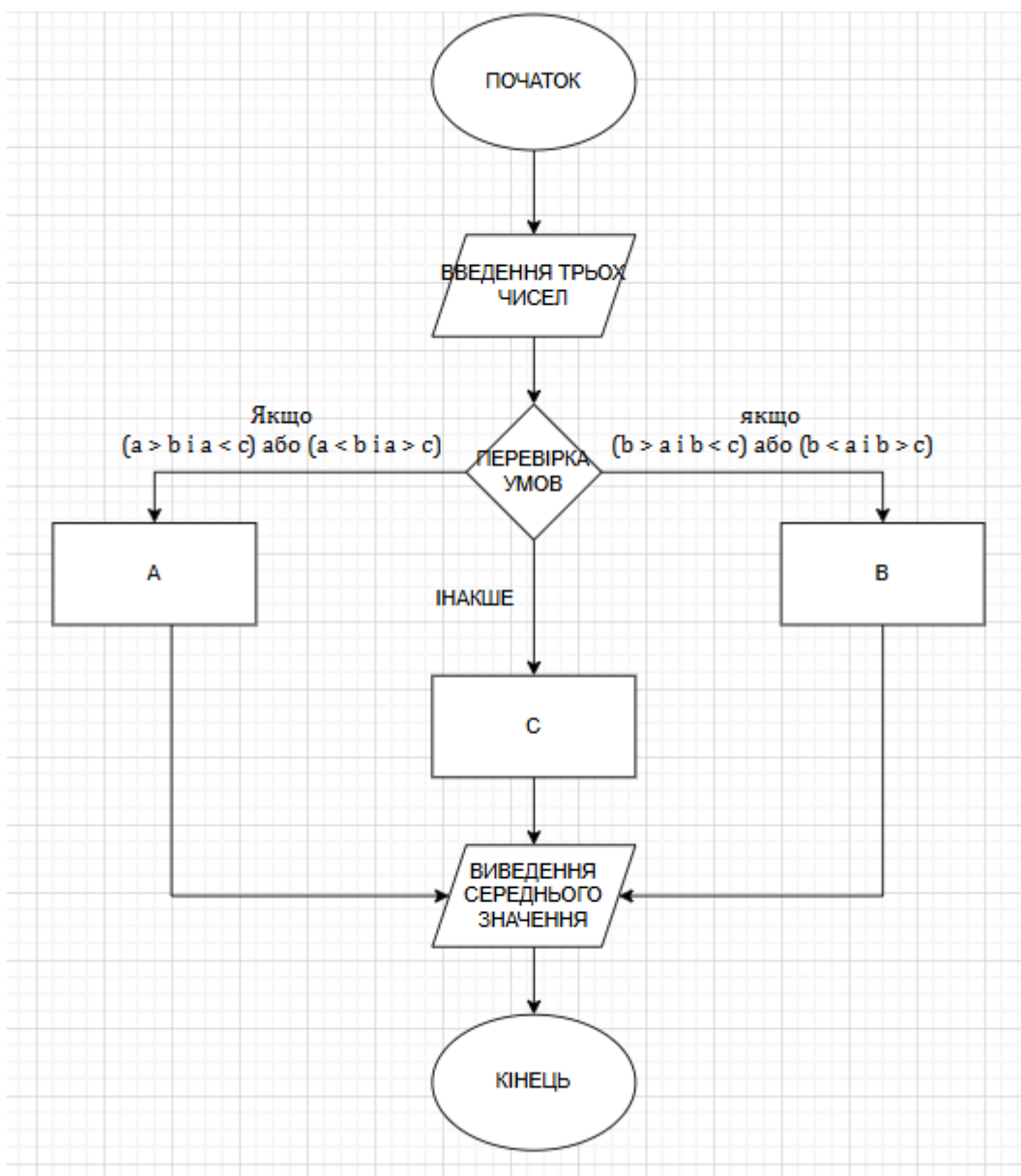


Рисунок 1 – Блок схема алгоритма вирішення до Завдання 1

Лістинг коду вирішення задачі наведено в дод. А (стор. 8). Екран роботи програми показаний на рис. Б.1

Завдання 2. Вирішення задачі Т.2.геом.обл. 18

Вхідні дані: Радіус: Точки:

Вихідні дані: Кількість точок, що потрапляють у помаранчеву область

Алгоритм вирішення показано на рис. 2

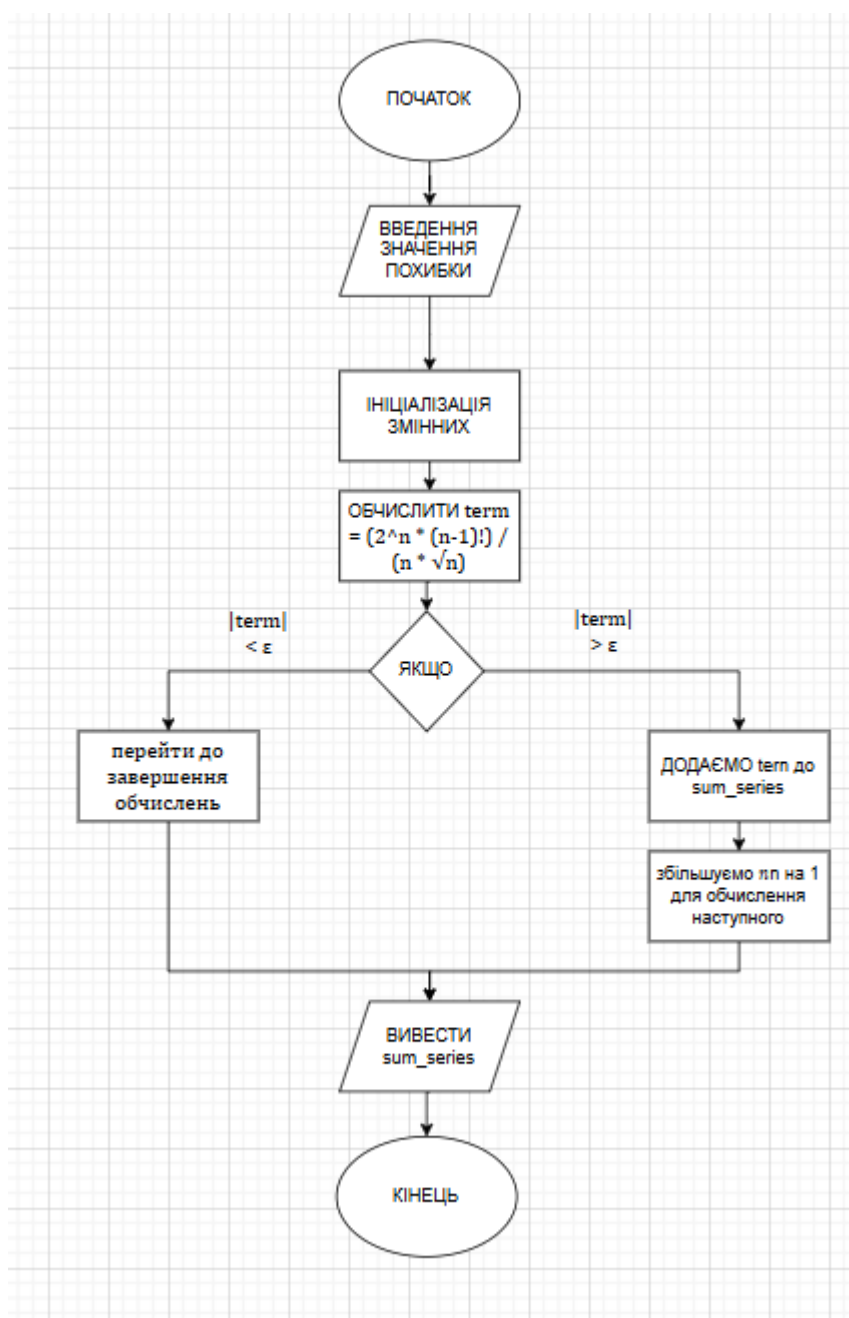


Рисунок 2 – Блок схема алгоритма вирішення до Завдання 2

Лістинг коду вирішення задачі наведено в дод. А (стор.8). Екран роботи програми показаний на рис. Б.2

Завдання 3. Вирішення задачі (Т.3.ряди) 17

Вхідні дані:

Похибка ϵ для точності обчислення суми

Вихідні дані:

Значення суми ряду

Алгоритм вирішення показано нижче

Встановлюємо початкове значення для лічильника $n=1$

Ініціалізуємо змінну для суми: $\text{sum_series} = 0$.

Визначаємо значення похибки ϵ , яка визначає точність обчислень.

Повторюємо обчислення для кожного члена ряду до тих пір, поки значення члена ряду не стане меншим за ϵ .

Для кожного значення n , обчислюємо чисельник члена ряду:

$\text{numerator} = 2^n \cdot (n-1)!$

Обчислюємо знаменник члена ряду: $\text{denominator} = n \cdot \sqrt[n]{n}$

Обчислюємо поточний член ряду: $\text{term} = \frac{\text{numerator}}{\text{denominator}}$

Якщо $|\text{term}| < \epsilon$, зупиняємо цикл обчислення (ряд збігся).

Інакше додаємо значення поточного члена до суми:

$\text{sum_series} += \text{term}$

Збільшуємо n на 1 і переходимо до наступної ітерації.

Після завершення циклу виводимо значення суми

sum_series

Лістинг коду вирішення задачі наведено в дод. А (стор.8). Екран роботи програми показаний на рис. Б.3

Завдання 4

Вирішення показано нижче:

Створюємо два окремих файли: основний скрипт для організації меню `main_script.py` та модуль з функцією для виконання обчислень `task_module.py`.

В основному скрипті `main_script.py` організуємо функцію `main_menu()` для виведення меню та циклічного виконання програми:

Виводимо меню з вибором дій.

Запитуємо у користувача вибір (1 або 0).

Якщо вибір – 1, викликаємо функцію `find_middle_number()` з модуля `task_module.py` для обчислення середнього числа з трьох введених чисел.

Якщо вибір – 0, виходимо з циклу та завершуємо програму.

Якщо введено некоректне значення, виводимо повідомлення про помилку та просимо повторити вибір.

У модулі `task_module.py` створюємо функцію `find_middle_number()`:

Запитуємо у користувача три числа через пробіл, перевіряючи правильність введення.

Якщо введено три числа, визначаємо середнє з них – число, яке розташоване між найменшим та найбільшим.

Порівнюємо введені числа, щоб знайти середнє, використовуючи умовні конструкції:

Якщо число a знаходиться між b і c , воно є середнім.

Якщо b знаходиться між a і c , воно є середнім.

Інакше середнє – це c .

Виводимо результат (середнє число) користувачу.

Якщо користувач вводить некоректні значення, виводимо повідомлення про помилку.

Запускаємо `main_script.py`, який організує меню та дозволяє виконувати завдання повторно до моменту завершення програми за бажанням користувача.

Лістинг коду вирішення задачі наведено в дод. А. стор.8 Екран роботи програми показаний на рис. Б.4

Висновок

У цій лабораторній роботі були реалізовані алгоритми для вирішення завдань з розгалуженнями та циклами, що включають перевірку точок на потрапляння в геометричну область, дослідження рядів на збіжність, а також реалізацію меню для багаторазового виконання програм. Було успішно застосовано структури розгалуження та цикли для вирішення інженерних завдань на Python, а також перевірено коректність введених даних і обробку помилок.

Додаток А

Лістинг коду програми для завдання (if) 13

```
def task_if1():
    """If 13. Дано три числа. Знайти середнє з них (тобто число, розташоване
    між найменшим і найбільшим).
    try:
        # Введення трьох чисел
        a, b, c = map(int, input("Введіть три числа через пробіл: ").split())
        # Логіка для знаходження середнього числа
        numbers = [a, b, c]
        numbers.sort() # Сортиємо числа
        middle = numbers[1] # Середнє число - це друге в відсортованому
        списку

        print("Середнє число: ", middle)
    except ValueError:
        print("INTEGER expected!") # Помилка, якщо введено некоректні
        значення
    # Виклик функції
    task_if1()
```

Лістинг коду програми для завдання (Т.2.геом.обл.) 18

```
def count_points_in_area(points, r):
    """
    Рахує кількість точок, що потрапляють в помаранчеву область.
    :param points: список координат точок [(x1, y1), (x2, y2), ...]
    :param r: радіус (межа по x)
    :return: кількість точок в області
    """
    count = 0
    for x, y in points:
        # Перевірка умов для помаранчевої області
        if 0 < x <= r and y < 0 and y >= -x:
            count += 1
    return count

# Приклад використання
points = [(1, -1), (2, -2), (3, -1), (-1, -1), (1, 1)]
```



```

r = 3 # Наприклад, радіус r
result = count_points_in_area(points, r)
print(f"Кількість точок у помаранчевій області: {result}")

```

Лістинг коду програми для завдання (Т.3.ряди) 17

```

import math

def compute_series(epsilon=1e-10):
    """
    Обчислює нескінченний ряд:
     $S = \sum (2^n * (2n-1)!) / \sqrt{n!}$  до збіжності з точністю epsilon.
    :param epsilon: точність обчислення
    :return: значення суми ряду
    """
    # Початкові значення
    s = 0 # Сума
    n = 1 # Перший індекс
    term = 2 / math.sqrt(1) # Початковий член ряду (для n=1)
    while abs(term) > epsilon:
        s += term # Додаємо поточний член до суми
        # Рекурсивно обчислюємо наступний член ряду
        term *= (2 * (2 * n) * (2 * n - 1)) / ((n + 1) * math.sqrt(n + 1))

        # Переходимо до наступного індексу
        n += 1
        # Перевірка на занадто довге виконання
        if n > 10000:
            print("Досягнуто ліміту ітерацій.")
            break
    return s

# Використання функції
result = compute_series()
print(f"Значення ряду: {result}")

```

Лістинг коду програми для задачі Завдання 4

```

# script-file
import math

def task_if1():
    """

```

Задача: визначити кількість точок, які потрапляють в область заданого кола (варіант 18).

```

"""
# Координати точок (xi, yi), можна змінювати
points = [
    (1, 1),
    (2, -2),
    (-1, 3),
    (0.5, 0.5),
    (-1.5, -1.5),
    (0, -1),
]

# Радіус кола (завдання використовує змінну "r")
r = 2

# Геометрична область: коло нижньої половини координатної системи
def is_in_region(x, y):
    """
    Перевірка, чи потрапляє точка в область.
    Для варіанта 18 це нижня половина жовтої області.
    """
    return x**2 + y**2 <= r**2 and y < 0

# Підрахунок точок, що потрапляють у задану область
count = sum(1 for x, y in points if is_in_region(x, y))
print(f"Кількість точок у жовтій області (варіант 18): {count}")

# Основний цикл
choice = int(input("Please, choose the task 1-3 (0-EXIT): "))
while choice:
    if choice == 1:
        task_if1()

    elif choice == 3:
        # Цей блок можна використовувати для інших завдань
        print("Task 3 placeholder")

    else:
        print("Wrong task number!")

    choice = int(input("Please, choose the task again (0-EXIT): "))

print("Good bye!")

```

Додаток Б

Скрін-шрти вікна виконання програми

```

1 def task_if13():
2     """If 13. Дано три числа. Знайти середнє з них (тобто число, розташоване між
   найменшим і найбільшим)."""
3
4     try:
5         # Введення трьох чисел
6         a, b, c = map(int, input("Введіть три числа через пробіл: ").split())
7         # Логіка для знаходження середнього числа
8         numbers = [a, b, c]
9         numbers.sort() # Сортуємо числа
10        middle = numbers[1] # Середнє число - це друге в відсортованому списку
11
12        print("Середнє число: ", middle)
13    except ValueError:
14        print("INTEGER expected!") # Помилка, якщо введено некоректні значення
15
16    # Виклик функції
17    task_if13()
  
```

Введіть три числа через пробіл: 1 3 5
Середнє число: 3

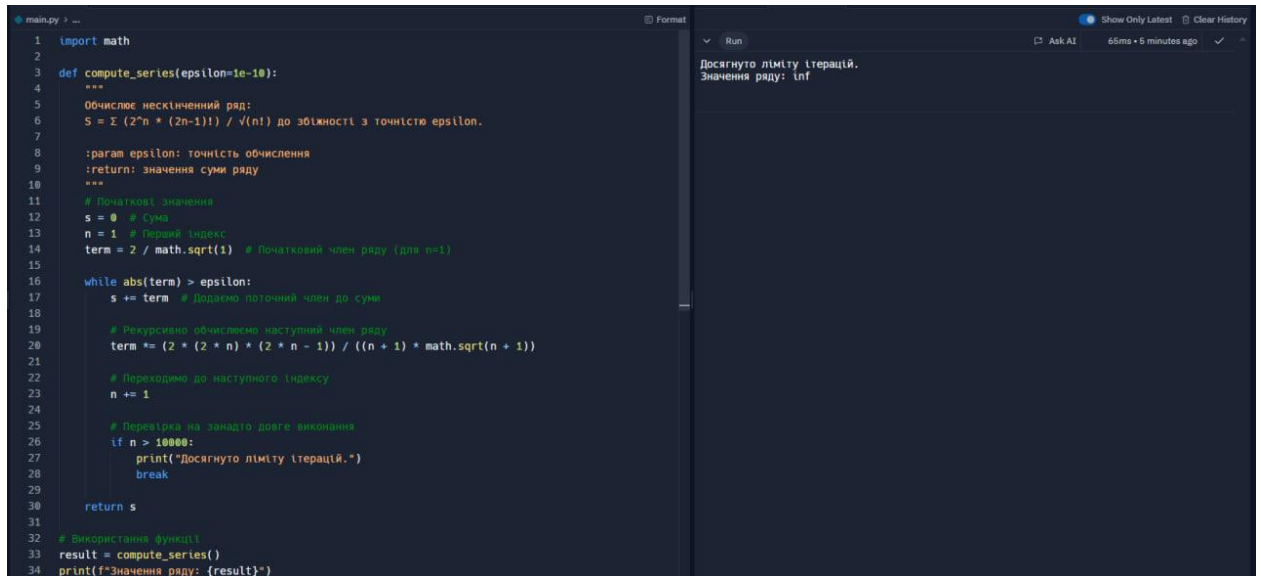
Рисунок Б.1 – Екран виконання програми для вирішення завдання (if) 13

```

1 def count_points_in_area(points, r):
2     """
3     Рахує кількість точок, що потрапляють в помаранчеву область.
4
5     :param points: список координат точок [(x1, y1), (x2, y2), ...]
6     :param r: радіус (межа по x)
7     :return: кількість точок в області
8     """
9     count = 0
10    for x, y in points:
11        # Перевірка умов для помаранчевої області
12        if 0 < x <= r and y < 0 and y >= -x:
13            count += 1
14    return count
15
16
17 # Приклад використання
18 points = [(1, -1), (2, -2), (3, -1), (-1, -1), (1, 1)]
19 r = 3 # Наприклад, радіус r
20 result = count_points_in_area(points, r)
21 print(f"Кількість точок у помаранчевій області: {result}")
  
```

Кількість точок у помаранчевій області: 3

Рисунок Б.2 – Екран виконання програми для вирішення завдання
(Т.2.геом.обл.) 18



```

1  import math
2
3  def compute_series(epsilon=1e-10):
4      """
5      Обчислює нескінченний ряд:
6       $S = \sum (2^n * (2n-1)! / \sqrt{n!})$  до збіжності з точністю epsilon.
7
8      :param epsilon: точність обчислення
9      :return: значення суми ряду
10     """
11     # Початкові значення
12     s = 0 # Сума
13     n = 1 # Перший індекс
14     term = 2 / math.sqrt(1) # Початковий член ряду (для n=1)
15
16     while abs(term) > epsilon:
17         s += term # Додаємо поточний член до суми
18
19         # Рекурсивно обчислюємо наступний член ряду
20         term *= (2 * (2 * n) * (2 * n - 1)) / ((n + 1) * math.sqrt(n + 1))
21
22         # Переходимо до наступного індексу
23         n += 1
24
25         # Перебірка на занадто довге виконання
26         if n > 10000:
27             print("Досягнуто ліміту ітерацій.")
28             break
29
30     return s
31
32 # Використання функції
33 result = compute_series()
34 print(f"Значення ряду: {result}")

```

Run
Досягнуто ліміту ітерацій.
Значення ряду: inf
Ask AI 66ms • 5 minutes ago

Рисунок Б.3 – Екран виконання програми для вирішення завдання (Т.3.ряди)

17

```

Please, choose the task 1-3 (0-EXIT): 1
Кількість точок у жовтій області (варіант 18): 1
Please, choose the task again (0-EXIT): 2
Wrong task number!
Please, choose the task again (0-EXIT): 3
Task 3 placeholder
Please, choose the task again (0-EXIT):

```

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