Студент группы ИС-26 Шишкин Максим

Практическое занятие №6

Тема: составление программ с функциями в IDE PyCharm Community.

Цели практического занятия: закрепить усвоенные знания, понятия, алгоритмы, основные принципы составления программ, приобрести навыки составление программ со списками в IDE PyCharm Community.

Постановка задачи: 1.

Дан список A размера N. Найти максимальный элемент из его элементов с нечетными номерами: A1, A3, A5, ...

Тип алгоритма: циклический

Текст программы:

```python
#Дан список A размера N. Найти максимальный элемент из его элементов с
#нечетными номерами: A1, A3, A5, ... .


import random

def generate_random_list(n):
    """Генерирует случайный список длиной n"""
    return [random.randint(0, 100) for _ in range(n)]  #Создаем список из n случайных целых чисел в диапазоне от 0 до 100

def find_max_odd_index_element(lst):
    """Находит максимальный элемент с нечетными индексами в списке lst"""
    if len(lst) < 2:
        raise ValueError("Список должен содержать хотя бы два элемента")  #Проверяем, что список содержит хотя бы два элемента

    odd_elements = lst[1::2]  #Берем элементы с нечетными индексами (начиная с первого)
    if not odd_elements:
        raise ValueError("В списке нет элементов с нечетными индексами")  #Проверяем, что в списке есть элементы с нечетными индексам

    return max(odd_elements)  #Возвращаем максимальный элемент из списка с нечетными индексами

try:
    N = int(input("Введите длину списка N: "))  #Запрашиваем у пользователя длину списка
    if N <= 0:
        raise ValueError("Длина списка должна быть положительным числом")  #Проверяем, что длина списка положительная

    A = generate_random_list(N)  #Генерируем случайный список длиной N
    print("Случайный список:", A)  #Выводим сгенерированный список

    max_odd_element = find_max_odd_index_element(A)  #Находим максимальный элемент с нечетным индексом
    print("Максимальный элемент с нечетным индексом:", max_odd_element)  #Выводим найденный элемент

except ValueError as e:
    print(f"Ошибка: {e}")  #Обрабатываем возможные ошибки ввода
```

Протокол программ:

Введите длину списка N: 4

Случайный список: [47, 2, 40, 11]

Максимальный элемент с нечетным индексом: 11

Process finished with exit code 0

Постановка задачи: 2.

Дан целочисленный список A размера N (< 15). Переписать в новый целочисленный список B все элементы с порядковыми номерами, кратными трем (3, 6, ...), и вывести размер полученного списка B и его содержимое. Условный оператор не использовать.

Тип алгоритма: Циклический

Текст программы:

```
#Дан целочисленный список A размера N (< 15). Переписать в новый целочисленный
#список B все элементы с порядковыми номерами, кратными трем (3, 6, ...), и вывести
#размер полученного списка B и его содержимое. Условный оператор не
#использовать.


import random

def generate_random_list(n):
    """Генерирует случайный список длиной n, содержащий целые числа в диапазоне от -50 до 50."""
    return [random.randint(-50, 50) for _ in range(n)]

def extract_every_third_element(lst):
    """Извлекает элементы из списка lst, стоящие на позициях, кратных 3 (то есть 3-й, 6-й, 9-й и т.д.)."""
    return lst[2::3]

try:
    N = int(input("Введите длину списка N (от 1 до 14): "))  #Запрос длины списка у пользователя
    if N < 1 or N > 14:
        raise ValueError("Длина списка должна быть в диапазоне от 1 до 14")  #Проверка корректности введенного значения

    A = generate_random_list(N)  #Генерация случайного списка длиной N
    print("Исходный список A:", A)  #Вывод исходного списка

    B = extract_every_third_element(A)  #Извлечение элементов, стоящих на позициях, кратных 3

    print("Размер списка B:", len(B))  #Вывод размера результирующего списка
    print("Элементы списка B:", B)  #Вывод элементов результирующего списка

except ValueError as e:
    print(f"Произошла ошибка: {e}")  #Обработка возможной ошибки ввода
```

Протокол программ:

Введите длину списка N (от 1 до 14): 11

Исходный список A: [7, -24, 41, -37, 47, 33, 1, -17, 26, 32, -37]

Размер списка B: 3

Элементы списка B: [41, 33, 26]

Process finished with exit code 0

Постановка задачи: 3.

Дано множество A из N точек (N > 2, точки заданы своими координатами x, y). Найти наименьший периметр треугольника, вершины которого принадлежат различным точкам множества A, и сами эти точки (точки выводятся в том же порядке, в котором они перечислены при задании множества A). Расстояние R между точками с координатами

(x1, y1) и (x2, y2) вычисляется по формуле: R = √(x2 – x1)2 + (y2 – y1)2. Для хранения данных о каждом наборе точек следует использовать по два списка: первый список для хранения абсцисс, второй — для хранения ординат.

Тип алгоритма: Циклический

Текст программы:

```
#Дано множество A из N точек (N > 2, точки заданы своими координатами x, y). Найти
#наименьший периметр треугольника, вершины которого принадлежат различным
#точкам множества A, и сами эти точки (точки выводятся в том же порядке, в котором
#они перечислены при задании множества A).
#Расстояние R между точками с координатами (x1, y1) и (x2, y2) вычисляется по формуле:
#R = √(x2 – x1)2 + (y2 – y1)2.
#Для хранения данных о каждом наборе точек следует использовать по два списка: первый
#список для хранения абсцисс, второй — для хранения ординат.


import random
from itertools import combinations  #Импортируем функцию для создания комбинаций
from math import sqrt  #Импортируем функцию для вычисления квадратного корня


def generate_random_coordinates(n):

    points_x = [random.uniform(-100, 100) for _ in range(n)]  #Генерация списка случайных координат X
    points_y = [random.uniform(-100, 100) for _ in range(n)]  #Генерация списка случайных координат Y
    return points_x, points_y  #Возвращаем два списка координат


def distance(x1, y1, x2, y2):

    return sqrt((x2 - x1)**2 + (y2 - y1)**2)  #Используем формулу Евклида для расчета расстояния


def perimeter(points_x, points_y, idx1, idx2, idx3):

    d1 = distance(points_x[idx1], points_y[idx1], points_x[idx2], points_y[idx2])  # Расчет длины стороны 1
    d2 = distance(points_x[idx2], points_y[idx2], points_x[idx3], points_y[idx3])  # Расчет длины стороны 2
    d3 = distance(points_x[idx3], points_y[idx3], points_x[idx1], points_y[idx1])  # Расчет длины стороны 3
    return d1 + d2 + d3  #Сумма длин сторон дает периметр треугольника


def smallest_perimeter_triangle(points_x, points_y):

    n = len(points_x)  #Определяем количество точек
    min_perim = float('inf')  #Устанавливаем начальное значение минимального периметра как бесконечность
    result_points = []  #Создаем пустой список для хранения координат вершин

    for comb in combinations(range(n), 3):  #Проходимся по всем возможным комбинациям троек индексов
        perim = perimeter(points_x, points_y, *comb)  #Рассчитываем периметр текущего треугольника
        if perim < min_perim:  #Если найден меньший периметр
```

```
                min_perim = perim  #Обновляем минимальное значение
                result_points = [(points_x[i], points_y[i]) for i in comb]  #Сохраняем координаты вершин

        return min_perim, result_points  #Возвращаем минимальный периметр и координаты вершин


def main():

    try:
        N = int(input("Введите количество точек N (N > 2): "))  #Запрашиваем у пользователя количество точек
        if N <= 2:  # Проверяем, что количество точек больше двух
            raise ValueError("Количество точек должно быть больше 2")  #Если условие нарушено, выбрасываем ошибку

        points_x, points_y = generate_random_coordinates(N)  #Генерируем случайные координаты точек

        min_perim, triangle_points = smallest_perimeter_triangle(points_x, points_y)  #Находим минимальный периметр

        print("\nКоординаты точек:")  #Выводим заголовок
        for i in range(N):  #Проходим по всем точкам
            print(f"{i+1}. ({points_x[i]:.2f}, {points_y[i]:.2f})")  #Выводим каждую точку с номером и координатами

        print(f"\nНаименьший периметр треугольника: {min_perim:.2f}")  #Выводим минимальный периметр
        for i, point in enumerate(triangle_points):  #Проходим по вершинам треугольника
            print(f"Вершина {i+1}: ({point[0]:.2f}, {point[1]:.2f})")  #Выводим координаты каждой вершины

    except ValueError as e:  #Обрабатываем возможные ошибки
        print(f"Произошла ошибка: {e}")  #Сообщение об ошибке


if __name__ == "__main__":
    main()  #Вызов основной функции при запуске скрипта
```

Протокол программ:

Введите количество точек N (N > 2): 10

Координаты точек:

1. (10.10, -71.79)
2. (-70.75, -15.50)
3. (71.52, 17.60)
4. (-44.13, 0.72)
5. (42.14, 41.15)
6. (-63.03, -93.71)
7. (64.55, -5.51)
8. (40.80, -95.79)
9. (-88.65, 14.66)
10. (3.77, -46.84)

Наименьший периметр треугольника: 112.90

Вершина 1: (-70.75, -15.50)

Вершина 2: (-44.13, 0.72)

Вершина 3: (-88.65, 14.66)

Process finished with exit code 0

Вывод: В процессе работы я закрепил полученные ранее навыки, приобрел новые навыки в использование списков и работы с ними, научился создавать программы с использованием библиотеки random и библиотеки math в IDE PyCharm Community.