Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Кафедра Систем Управления и Информатики

Лабораторная работа №1
Вариант №2

Выполнил
Тарабанов Алексей Вячеславович
Проверил
Мусаев А.А.

Санкт-Петербург,
2022

!!!*Код всех программ можно просмотреть, перейдя по приложенной ссылке на гитхаб.!!!

Задание №1

Описание задания:

Реализовать алгоритм фасетного поиска для определенной группы объектов (Таблица 1). Реализовать алгоритм не менее чем для 16 объектов. Пользователь может отвечать на вопросы только «да» и «нет».

Пример:

Дано: 1, 2, 32, 13.

-Вы загадали цифру, но не число?

-Да.

-Оно является четным?

-Да.

Ответ: 2

Решение:

В ходе выполнения задания я использовал сложный словарь, состоящий из списков. В качестве ключа я использовал имя птицы, а в качестве значения вводил их качество, по которым будет произведён фасетный поиск. Он изображен на рисунке 1.

```
birds = {"Утка": ["Летает","Плавает","Маленькая","Короткие", "Короткий", "Не розовая"],
        "Пингвин": ["Не летает", "Плавает", "Большой", "Короткие", "Короткий", "Не розовая"],
        "Страус": ["Не летает", "Не плавает", "Большой", "Длинные", "Короткий", "Не розовая"],
        "Орёл": ["Летает", "Не плавает", "Большой", "Короткие", "Короткий", "Не розовая"],
        "Ласточка": ["Летает", "Не плавает", "Маленькая", "Короткие", "Короткий", "Не розовая"],
        "Фламинго": ["Летает", "Не плавает", "Большой", "Длинные", "Длинный", "Розовая"],
        "Аист": ["Летает", "Не плавает", "Большой", "Длинные", "Длинный", "Не розовая"],
        "Киви": ["Не летает", "Не плавает", "Маленькая", "Короткие", "Длинный", "Не розовая"],
        "Лебедь": ["Летает", "Плавает", "Большой", "Короткие", "Короткий", "Не розовая"],
        "Щеголь": ["Летает", "Не плавает", "Маленькая", "Длинные", "Длинный", "Не розовая"],
        "Индюк": ["Не летает", "Не плавает", "Большой", "Короткие", "Короткий", "Не розовая"],
        "Какаду-инка(розовый попугай)": ["Летает", "Не плавает", "Большой", "Короткие", "Короткий", "Розовая"],
        "Калибри": ["Летает", "Не плавает", "Маленькая", "Короткие", "Длинный", "Не розовая"],
        "Дятел": ["Летает", "Не плавает", "Большой", "Короткие", "Длинный", "Не розовая"],
        "Пеликан": ["Летает", "Плавает", "Большой", "Короткие", "Длинный", "Не розовая"],
        "Розовый танагровый певун": ["Летает", "Не плавает", "Маленькая", "Короткие", "Короткий", "Розовая"]}
```

Рисунок 1 – Сложный список с характеристиками.

Далее я реализую рекурсивную функцию с ограниченным кол-во вопросом и нумерацией, и в связи с ответом пользователя удаляю из словаря тех птиц, чьи характеристики не совпадают с входящей информацией. Нумерация создана для того, чтобы с каждым вопросом брать нужное значение из списка качеств. (Рисунок 2)

```
def Q(birds, Qwest, Atrue, Afalse, num):
        if num > 1:
                global varAns
        else:
                varAns = list(birds)
        Qw = input(Qwest)
        if Qw == "да":
                for k, v in birds.items(): # Мы перебераем птиц по характеристикам
                        if str(v[num - 1]) == Afalse: # Есть ли у птицы качество?
                                if str(k) in varAns: # Есть ли такая птица в списке?
                                        varAns.remove(str(k)) # Удаляем эту птицу
        elif Qw == "нет":
                for k, v in birds.items(): # Мы перебераем птиц по характеристикам
                        if str(v[num - 1]) == Atrue: # Есть ли у птицы качество?
                                if str(k) in varAns: # Есть ли такая птица в списке?
                                        varAns.remove(str(k)) # Удаляем эту птицу
        else:
                print('Вы можете ответить только "Да" или "Нет".')
                return Q(birds, Qwest, Atrue, Afalse, num)
        if len(varAns) == 1:
                return varAns
        if num == 1:
                Q(birds, "Ваша птица плавает? ", "Плавает", "Не плавает", 2)
        if num == 2:
                Q(birds, "Ваша птица маленькая? ", "Маленькая", "Большой", 3)
        if num == 3:
                Q(birds, "Ваша птица имеет длинные ноги(относительно тела)? ", "Длинные", "Короткие", 4)
        if num == 4:
                Q(birds, "У вашей птицы длинный клюв? ", "Длинный", "Короткий", 5)
        if num == 5:
                Q(birds, "Ваша птица розовая? ", "Розовая", "Не розовая", 6)
        return varAns
```

Рисунок 2 – Рекурсивная функция с вопросами.

Далее я вызываю эту функцию, а также делаю проверку на существование птицы пользователя в данном списке. Это можно увидеть на рисунке 3.

```
print("\n Загадайте одну из нижеперечисленных птиц.\n")
print(list(birds),"\n")
print("Ответье на вопросы:")

you_bird = Q(birds, "Ваша птица летает? ", "Летает", "Не летает", 1)

if not you_bird:
        print("Такой птицы нет, может вы ошиблись?")
else:
        print("\n Ваша птица: ", you_bird)
```

Рисунок 3 – Вызов функции и проверка.

Результат выполнения программы:

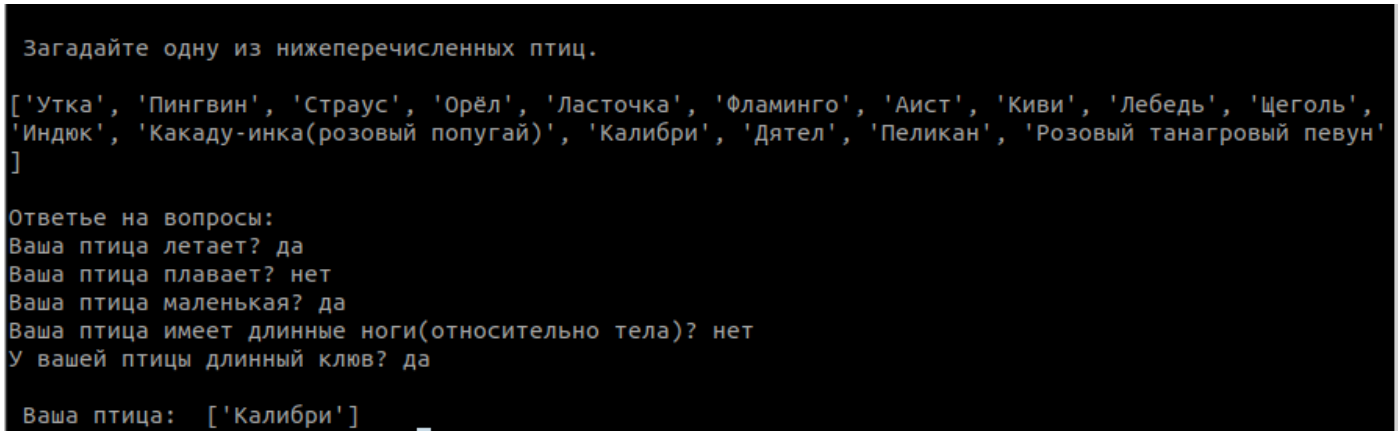


Рисунок 4 – Результат выполнения программы.

Вывод:

Реализован алгоритм фасетного поиска, с помощью рекурсивной функции, пробегающейся по сложному словарю, содержащему вложенные списки.

Задание №2

Описание задания:

Пользователь задает любое количество чисел с экрана, разделяя их запятыми. Реализовать алгоритм, который распределяет числа на натуральные, целые, рациональные, вещественные, комплексные, четные, нечетные и простые. Обратите внимание, что цифры могут попадать в несколько категорий.

Решение:

Для решения поставленной задачи я создавал разные списки для каждого типа числа. Их можно увидеть на рисунках 5-11.

```
# поиск натуральных чисел

num_nat = []
for i in num_list:
        if i.isnumeric():
                num_nat.append(int(i))
print("\nНатуральные числа: ", num_nat)
```

Рисунок 5 – Выбор натуральных чисел.

```
# поиск целых чисел

num_cel = []
for i in num_list:
        if i.isdigit() or i[1:].isdigit():
                num_cel.append(int(i))
print("Целые числа: ", num_cel)
```

Рисунок 6 – Выбор целые чисел.

```
# поиск рациональных чисел

num_rac = []
for i in num_list:
        if i.isdigit() or i[1:].isdigit():
                num_rac.append(int(i))
        elif i != "P" and i != "e":
                num_rac.append(float(i))
print("Рациональные числа: ", num_rac)
```

Рисунок 7 – Выбор рациональные чисел.

```
# поиск вещественных чисел

num_vec = []
for i in num_list:
        if i.isdigit() or i[1:].isdigit():
                num_vec.append(int(i))
        elif i == "P" or i == "e":
                num_vec.append(i)
        else:
                num_vec.append(float(i))
print("Вещественные числа: ", num_vec)
```

Рисунок 8 — Выбор вещественных чисел.

```
# поиск чётных чисел

num_chet = []
for i in num_list:
        if (i.isdigit() or i[1:].isdigit()):
                if int(i) % 2 == 0 and int(i) != 0:
                        num_chet.append(int(i))
print("Чётные числа: ", num_chet)
```

Рисунок 9 — Выбор чётных чисел.

```
# поиск нечётных чисел

num_nechet = []
for i in num_list:
        if (i.isdigit() or i[1:].isdigit()):
                if int(i) % 2 != 0 and int(i) != 0:
                        num_nechet.append(int(i))
print("Нечётные числа: ", num_nechet)
```

Рисунок 10 — Выбор нечётных чисел.

```
# поиск простых чисел

num_pro = []
count_del = 0
for i in num_nat:
        for j in range(2, i):
                if i % j == 0:
                        count_del = count_del + 1
        if count_del == 0:
                num_pro.append(i)
        else:
                count_del = 0

print("Простые числа: ", num_pro)
```

Рисунок 11 — Выбор простых чисел.

Так же я реализовал функцию перевода из строки чисел через запятую. Его можно увидеть на рисунке 12

```
def perevod(num, num_list):
        b = ""
        for i in num:
                if "0" <= i <= "9" or i == "-" or i == "." or i == "P" or i == "e":
                        b += i
                elif i == ",":
                        num_list.append(b)
                        b = ""
        num_list.append(b)
        return(num_list)
```

Рисунок 12 — перевод из строки ввода в список.

Результат выполнения программы:

```
1, 2, 3, 4, 5, 6, 7, -1, -2, -3, -4, -5, -6, 0, -0, 1.5, 1.25, -99.9999, e, P

Натуральные числа:  [1, 2, 3, 4, 5, 6, 7, 0]
Целые числа:  [1, 2, 3, 4, 5, 6, 7, -1, -2, -3, -4, -5, -6, 0, 0]
Рациональные числа:  [1, 2, 3, 4, 5, 6, 7, -1, -2, -3, -4, -5, -6, 0, 0, 1.5, 1.25, -99.9999]
Вещественные числа:  [1, 2, 3, 4, 5, 6, 7, -1, -2, -3, -4, -5, -6, 0, 0, 1.5, 1.25, -99.9999, 'e', 'P']
Чётные числа:  [2, 4, 6, -2, -4, -6]
Нечётные числа:  [1, 3, 5, 7, -1, -3, -5]
Простые числа:  [1, 2, 3, 5, 7, 0]
```

Рисунок 13 — результат выполнения программы.

Вывод:

Написан алгоритм распределения чисел по категориям, с помощью

условных операторов.

Задание №3

Описание задания:

Пользователь задает массив натуральных чисел. Реализовать для них алгоритмы сортировки следующими методами: пузырьковый, гномий, блочный, пирамидальный. Проанализировать достоинства и недостатки данных методов.

Решение:

В ходе решения я отсортировал список пользователя 4 разными способами сортировки изображённые на рисунках 14-17.

```
# Пузырьковая сортировка

num_buble = num_list
def buble_sort(num_buble):
        swapped = True
        while swapped == True:
                swapped = False
                for i in range(0, len(num_buble)-1):
                        if num_buble[i] > num_buble[i+1]:
                                num_buble[i], num_buble[i+1] = num_buble[i+1], num_buble[i]
                                swapped = True
        return num_buble
print("Пузырьковая сортировка: ", buble_sort(num_buble))
```

Рисунок 14 – Пузырьковая сортировка.

Для понимания и реализации этот алгоритм – простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: (On^2).

```
# Гномья сортировка

num_gnome = num_list
def gnume_sort(num_gnome):
        for i in range(0, len(num_gnome)-1):
                if num_gnome[i] > num_gnome[i+1]:
                        num_gnome[i], num_gnome[i+1] = num_gnome[i+1], num_gnome[i]
                        for j in range(i, 0, -1):
                                if num_gnome[j] < num_gnome[j-1]:
                                        num_gnome[j], num_gnome[j-1] = num_gnome[j-1],
num_gnome[j]
                        gnume_sort(num_gnome)
        return num_gnome
print("Гномья сортировка: ", gnume_sort(num_gnome))
```

Рисунок 15 – Гномья сортировка.

Он сочетает в себе сортировку вставка и пузырьковую сортировку. Как уже упоминалось, алгоритм не очень быстрый, но размер кода крошечный по сравнению с другими алгоритмами.

```
num_bucket = num_list
def bucket_sort(num_bucket):
        max_value = max(num_bucket)
        min_num = min(num_bucket)
        max_num = max(num_bucket)
        size = max_num/len(num_bucket)

        buckets_list= []
        for x in range(len(num_bucket)):
                buckets_list.append([])

        for i in range(len(num_bucket)):
                j = int(num_bucket[i] / size)
                if j != len (num_bucket):
                        buckets_list[j].append(num_bucket[i])
                else:
                        buckets_list[len(num_bucket) - 1].append(num_bucket[i])

        for z in range(len(num_bucket)):
                buble_sort(buckets_list[z])

        final_output = []
        for x in range(len (num_bucket)):
                final_output = final_output + buckets_list[x]
        return final_output
print("Блочная сортировка: ", bucket_sort(num_bucket))
```

Рисунок 16 – Блочная сортировка.

Преимущества: относится к классу быстрых алгоритмов с линейным временем исполнения O(N) (на удачных входных данных). **Недостатки**: сильно деградирует при большом количестве мало отличных элементов, или же на неудачной функции получения номера корзины по содержимому элемента.

```python
# Пирамидальная сортировка

num_heap = num_list
def heap_sort(alist):
    def parent(i):
        return (i - 1) // 2

    def left(i):
        return 2 * i + 1

    def right(i):
        return 2 * i + 2

    def max_heapify(alist, index, size):
        l = left(index)
        r = right(index)
        if (l < size and alist[l] > alist[index]):
            largest = l
        else:
            largest = index
        if (r < size and alist[r] > alist[largest]):
            largest = r
        if (largest != index):
            alist[largest], alist[index] = alist[index], alist[largest]
            max_heapify(alist, largest, size)

    def build_max_heap(alist):
        length = len(alist)
        start = parent(length - 1)
        while start >= 0:
            max_heapify(alist, index=start, size=length)
            start = start - 1

    build_max_heap(alist)
    for i in range(len(alist) - 1, 0, -1):
        alist[0], alist[i] = alist[i], alist[0]
        max_heapify(alist, index=0, size=i)

    return alist

print("Пирамидальная сортировка: ", heap_sort(num_heap))
```

Рисунок 17 — Пирамедальная сортировка

Результат выполнения программы:

```
Введите массив чисел: 4 7 3 1 2 8 9 5 6
--------------------
Пузырьковая сортировка:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
Гномья сортировка:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
Блочная сортировка:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
Пирамидальная сортировка:  [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Вывод:

Реализованы разные виды сортировок. У каждого из представленных алгоритмов
сортировки есть свои достоинства и недостатки, и каждый из них имеет свою область
применения.

Задание №4

Описание задания:

С А третьекурсниками часто происходит событие В. Зная вероятность N, что данное событие произойдет с ними за С дней (N вводится пользователем при запуске программы для каждого из А), определите вероятность того, что за D дней данная ситуация произойдет только с третьекурсником Е. Реализовать алгоритм для решения данной задачи.

Решение:

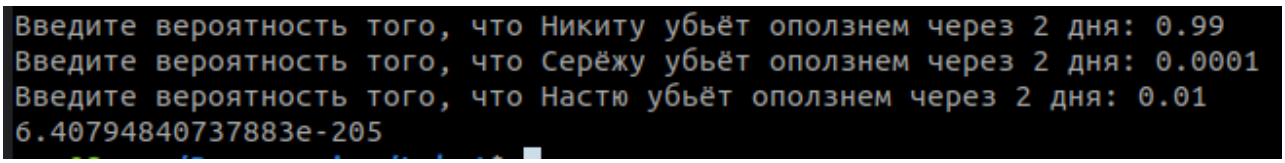
В ходе выполнения задания, были использованы математические формулы из теории вероятности.

Изначально просчитывается вероятность происшествия за 1 день, а после, перемножается на кол-во дней и вероятность происшествия искомого человека перемножается с вероятностью отсутствия происшествия у других. Это можно увидеть на рисунке 18.

```
from math import sqrt

Nik = float(input("Введите вероятность того, что Никиту убьёт оползнем через 2 дня: "))
Ser = float(input("Введите вероятность того, что Серёжу убьёт оползнем через 2 дня: "))
Nas = float(input("Введите вероятность того, что Настю убьёт оползнем через 2 дня: "))

od_Nik = (1-Nik)**(1/2)
od_Ser = (1-Ser)**(1/2)
od_Nas = (1-Nas)**(1/2)

A = (od_Nik**202) * (1 - od_Ser**202) * (1 - od_Nas**202)

print(A)
```

Рисунок 18 – расчёт вероятностей.

Результат выполнения программы:



Вывод:

Никакого алгоритма не используется, используются только формулы для подсчёта вероятностей.

Задание №5

Описание задания:

Реализовать алгоритм, заполняющий таблицу неповторяющимися координатами x и y. Количество координат n равно квадратному корню из номера варианта помноженному на 10 и округленному в большую сторону. Диапазон значений координат вводится пользователем при запуске программы. Для четных вариантов таблица формируется в Excel или другом оффлайновом аналоге. Для нечетных вариантов таблица формируется в таблицах google. Для заданных координат реализовать алгоритм метода наименьших квадратов (не используя готовые библиотеки для МНК) и построить гарфик (библиотека matplotlib).

Решение:

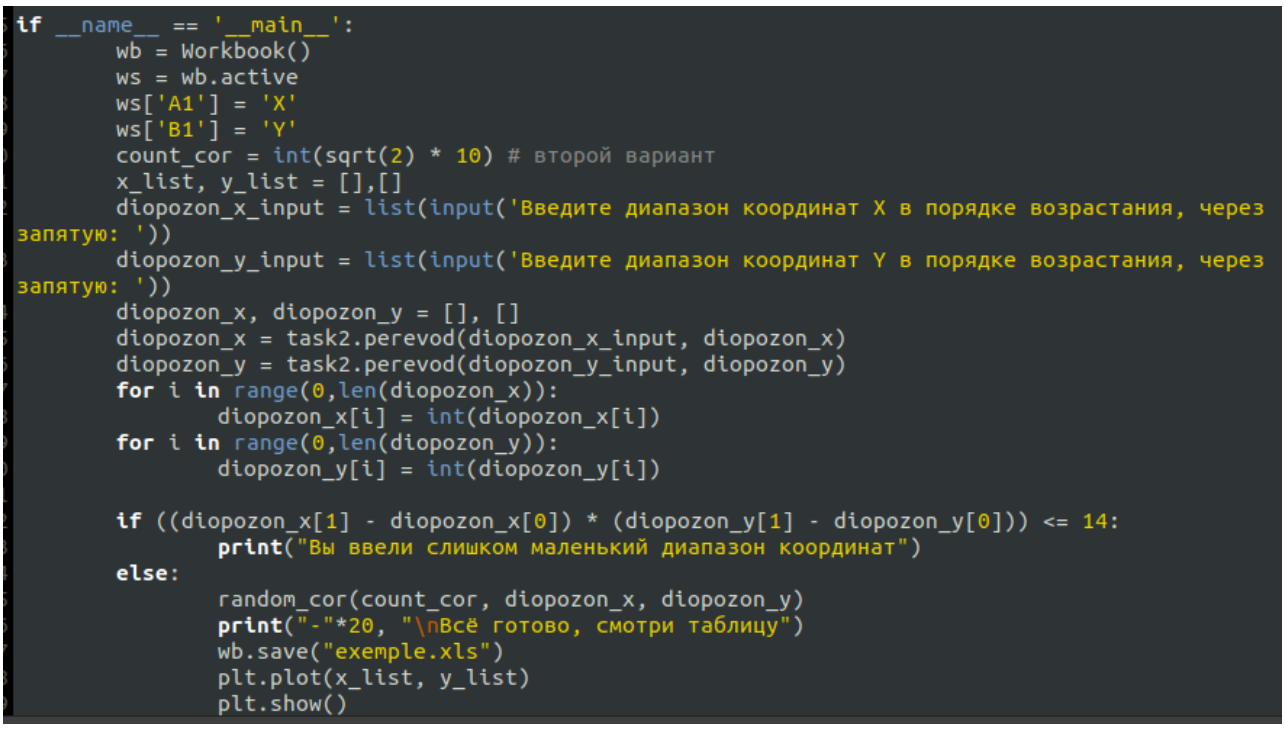
Изначально я реализовал создание рандомных координат и проверку на их повторение, это изображено на рисунке 19. После я сделал функцию метода наименьших квадратов, которую можно увидеть на рисунке 20. Потом я запросил диапазон координат и используя функцию генерации рандомных координат в заданном диапазоне, с помощью библиотек вывел данные в exel таблицу, это можно увидеть на рисунке 21.

```
def random_cor(count_cor, diopozon_x, diopozon_y):
        cor_list = []
        global x_list, y_list
        for i in range(1, count_cor+1):
                x = 'A' + str(i+1)
                y = 'B' + str(i+1)
                while ws[x].value == None:
                        a = random.randint(diopozon_x[0], diopozon_x[1])
                        b = random.randint(diopozon_y[0], diopozon_y[1])
                        if [a,b] in cor_list:
                                continue
                        else:
                                ws[x] = str(a)
                                ws[y] = str(b)
                                x_list.append(a)
                                y_list.append(b)
                                cor_list.append([a,b])
        print("-"*20,"\n Метод наименьших квадратов\n",MNK(x_list, y_list))
```

Рисунок 19 — функция генерации рандомных координат с проверкой на идентичность.

```
def MNK(x_list, y_list):
        x_mid = sum(x_list)/len(x_list)
        y_mid = sum(y_list)/len(y_list)
        x_otklon = 0
        num = 0
        for i in range(len(x_list)):
                x_otklon = x_otklon + (x_list[i] - x_mid)
                num += (x_list[i] - x_mid)*(y_list[i] - y_mid)
        b1, b2 = 0, 0

        if x_otklon != 0:
                b2 = num/(x_otklon**2)
                b1 = y_mid - (b2*x_mid)
                return "Yi = " + str(b1) + "+" + str(b2) + "* Xi"
        else:
                return "Yi = Xi"
```

Рисунок 20 — Метод наименьших квадратов.



```
if __name__ == '__main__':
        wb = Workbook()
        ws = wb.active
        ws['A1'] = 'X'
        ws['B1'] = 'Y'
        count_cor = int(sqrt(2) * 10) # второй вариант
        x_list, y_list = [],[]
        diopozon_x_input = list(input('Введите диапазон координат X в порядке возрастания, через
запятую: '))
        diopozon_y_input = list(input('Введите диапазон координат Y в порядке возрастания, через
запятую: '))
        diopozon_x, diopozon_y = [], []
        diopozon_x = task2.perevod(diopozon_x_input, diopozon_x)
        diopozon_y = task2.perevod(diopozon_y_input, diopozon_y)
        for i in range(0,len(diopozon_x)):
                diopozon_x[i] = int(diopozon_x[i])
        for i in range(0,len(diopozon_y)):
                diopozon_y[i] = int(diopozon_y[i])

        if ((diopozon_x[1] - diopozon_x[0]) * (diopozon_y[1] - diopozon_y[0])) <= 14:
                print("Вы ввели слишком маленький диапазон координат")
        else:
                random_cor(count_cor, diopozon_x, diopozon_y)
                print("-"*20, "\nВсё готово, смотри таблицу")
                wb.save("exemple.xls")
                plt.plot(x_list, y_list)
                plt.show()
```

Рисунок 21 – Запрос диапазона координат, запись в exel таблицу, вывод графика.

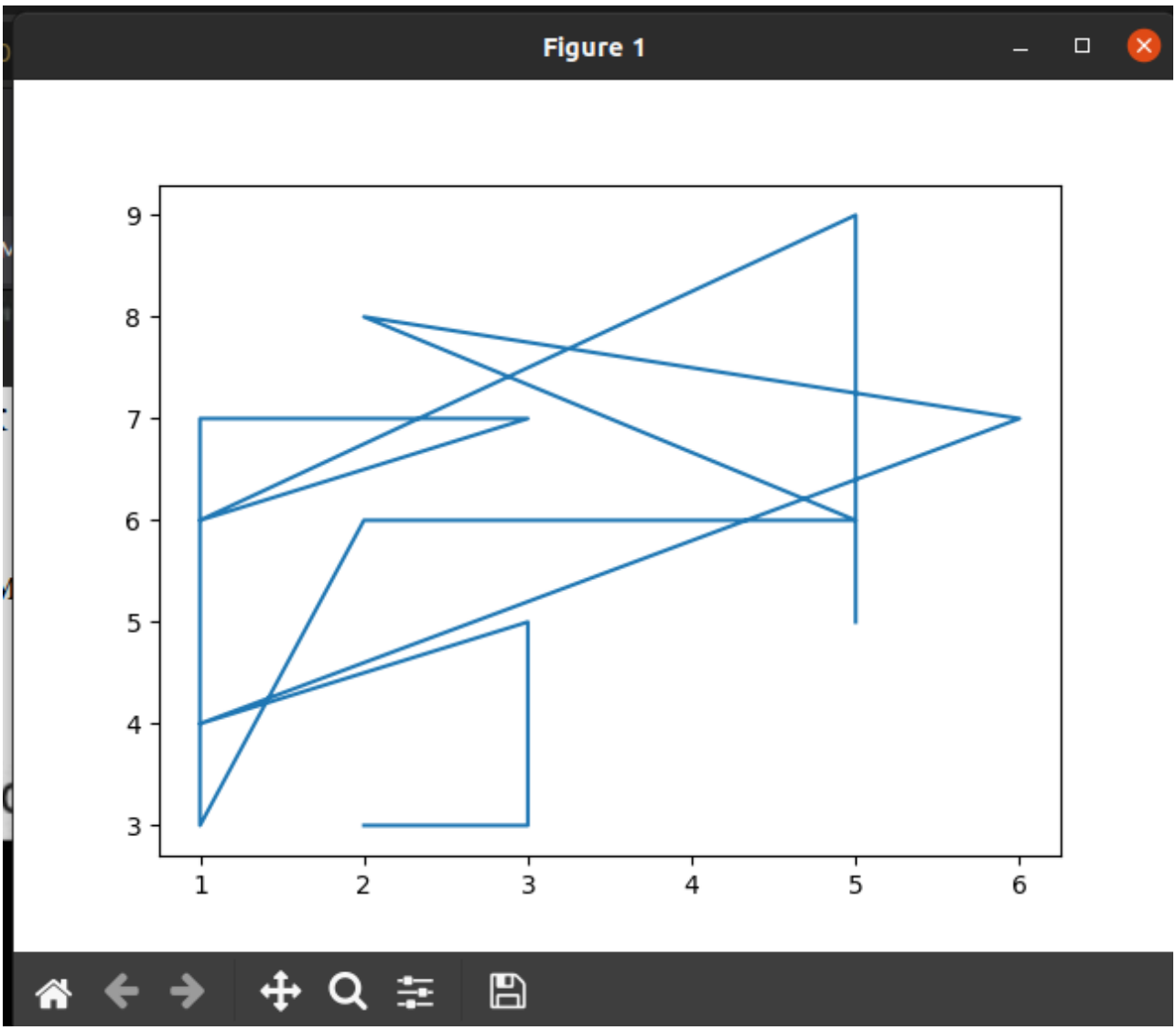


Рисунок 22 — График(рандомных координат).

	A	B
1	X	Y
2	5	5
3	5	9
4	1	6
5	3	7
6	1	7
7	1	3
8	2	6
9	5	6
10	2	8
11	6	7
12	1	4
13	3	5
14	3	3
15	2	3
16		

Рисунок 23 – Exel таблица.

Результат выполнения программы:

```
Введите диапазон координат X в порядке возрастания, через запятую: 1, 6
Введите диапазон координат Y в порядке возрастания, через запятую: 3, 9
--------------------
 Метод наименьших квадратов
 Yi = -5.536269968343695e+31+1.9376944889202932e+31* Xi
--------------------
Всё готово, смотри таблицу
```

Вывод:

Для расчёта коэффициентов прямой используется метод наименьших квадратов.

Задание №6

Описание задания:

Без применения готовых библиотек, написать алгоритм, который позволяет вводить матрицу (указывается размер и значения элементов), а затем по желанию пользователя выполнять возведение в квадрат (если возможно),транспонировать (если возможно), для квадратной матрицы находить определитель.

Решение:

В ходе выполнения задания, я сделал 4 функции. Первая функция помогает создать матрицу и заполнить ей, а следующие три позволяют применять к ней какие-либо действия на выбор пользователя. Все функции изображены на рисунках 24-27. Так же программа узнают количество столбцов и строк в матрице пользователя.

```
def create_matrix():
        count_stolb = int(input("Введите ко-во столбцов: "))
        count_str = int(input("Введите ко-во строк: "))
        count_a = 0
        a_list = []
        matrix = []
        for i in range(count_str):
                a_list = []
                matrix.append(a_list)
                for j in range(count_stolb):
                        count_a += 1
                        A = "A" + str(count_a)
                        a_list.append(A)
                print(a_list)

        count_a = 0
        for i in range(len(matrix)):
                for j in range(len(matrix[i])):
                        count_a += 1
                        matrix[i][j] = input("Введите значение "+"A"+str(count_a)+": ")
        print("-"*20, "\nВаша матрица:")
        for i in range(len(matrix)):
                print(matrix[i])
        return matrix
```

Рисунок 24 – Функция создания и заполнения матрицы.

```
def kvadrat(matrix):
        K_matrix = []
        for i in range(len(matrix)):
                K_line = []
                for j in range(len(matrix[i])):
                        K_line.append(0)
                K_matrix.append(K_line)
        for i in range(len(matrix)):
                for j in range(len(matrix[i])):
                        for z in range(len(matrix)):
                                K_matrix[i][j] += int(matrix[i][z]) * int(matrix[z][j])
        return K_matrix
```

Рисунок 25 — Функция возведения матрицы в квадрат.

```
def transpon(matrix):
        T_matrix = []
        for i in range(len(matrix)):
                T_line = []
                for j in range(len(matrix[i])):
                        T_line.append(matrix[j][i])
                T_matrix.append(T_line)
        return T_matrix
```

Рисунок 26 — Функция транспонирования матрицы.

```
def opredel(matrix):
        if len(matrix) == 2:
                return int(matrix[0][0]) * int(matrix[1][1]) - int(matrix[0][1]) * int(matrix[1]
[0])
        else:
                O_matrix = 0
                for i in range(len(matrix)):
                        O_matrix1 = []
                        for j in range(1, len(matrix)):
                                O_line = []
                                for k in range(len(matrix[j])):
                                        if k != i:
                                                O_line.append(matrix[j][k])
                                O_matrix1.append(O_line)
                        O_matrix += array[0][i] * (-1) ** ((i + 1) + (0 + 1)) * det(O_matrix1)
        return O_matrix
```

Рисунок 27 — Функция нахождения определителя матрицы.

Результат выполнения программы:

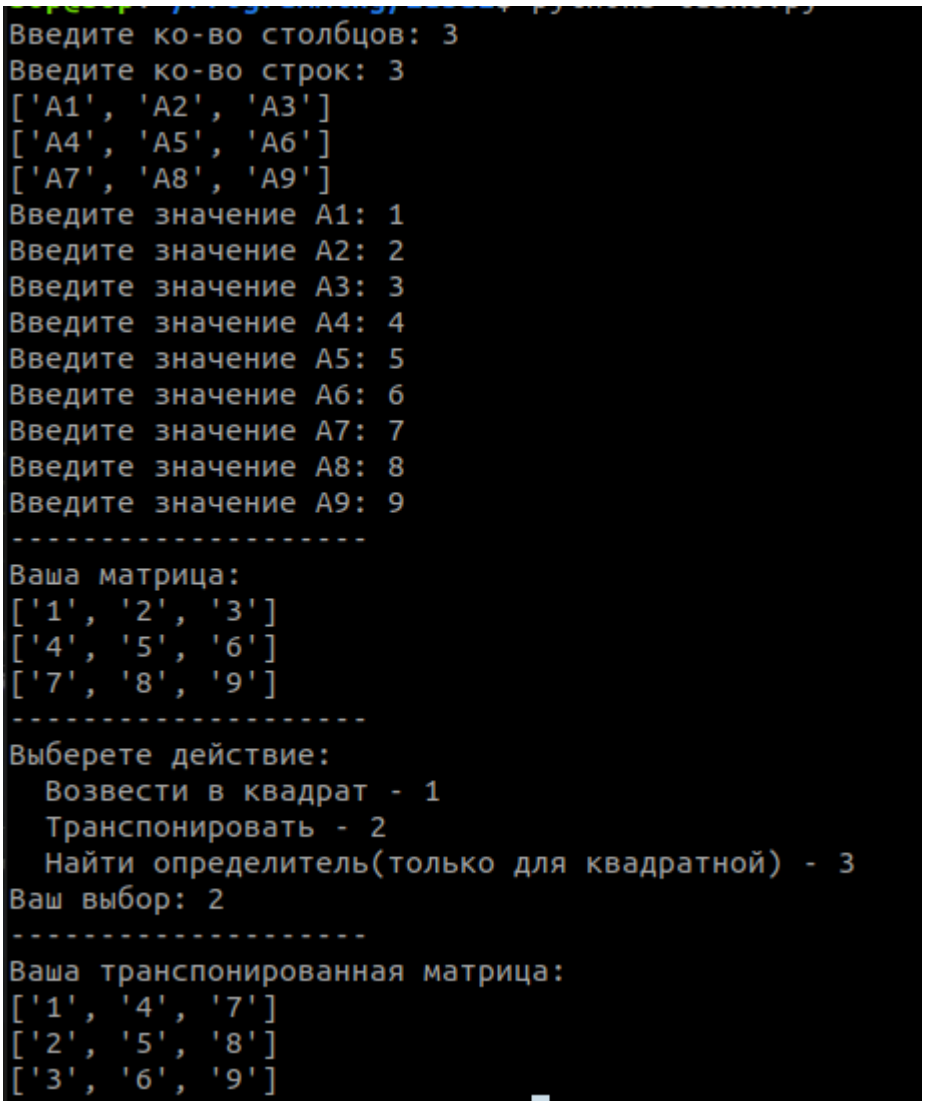


Рисунок 28 – Результат транспонирования матрицы.

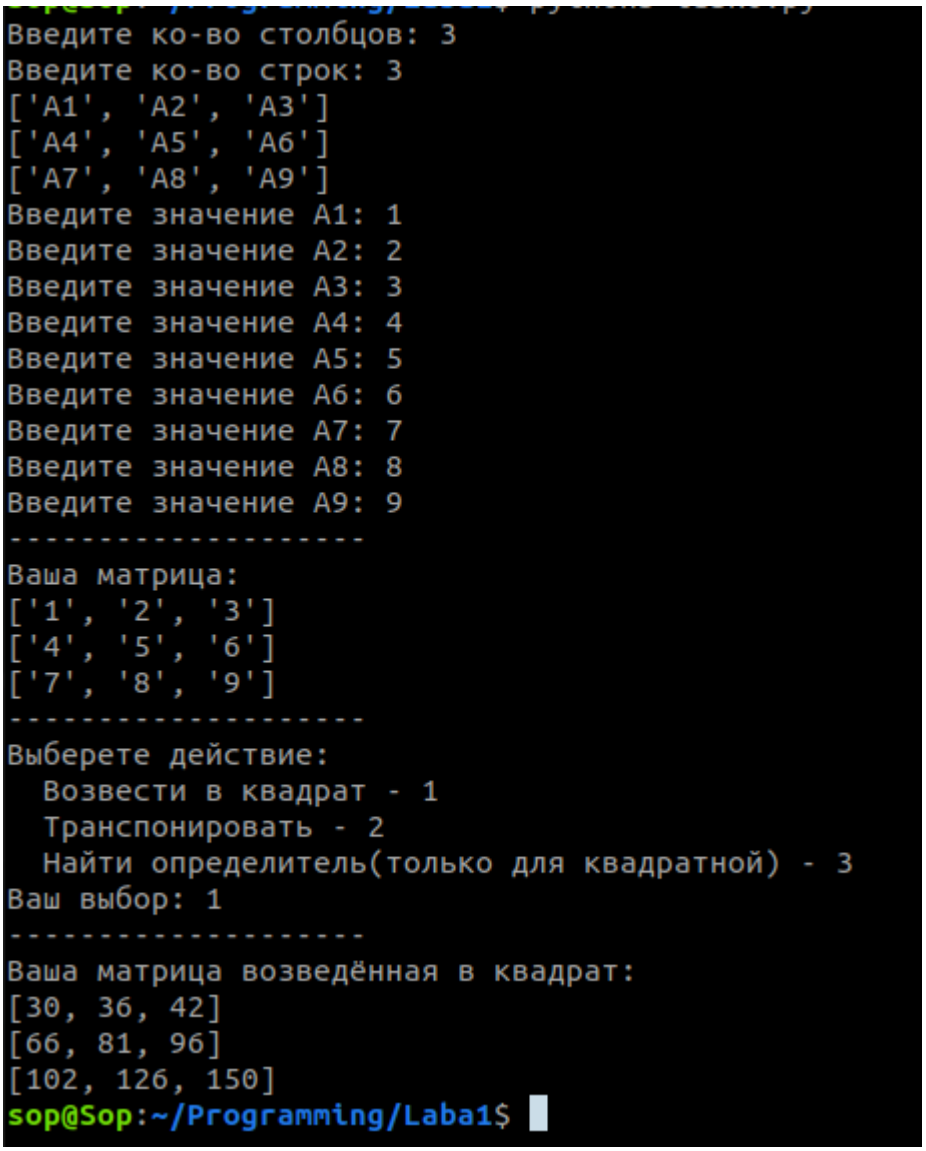


Рисунок 29 – Результат возведения матрицы в квадрат.

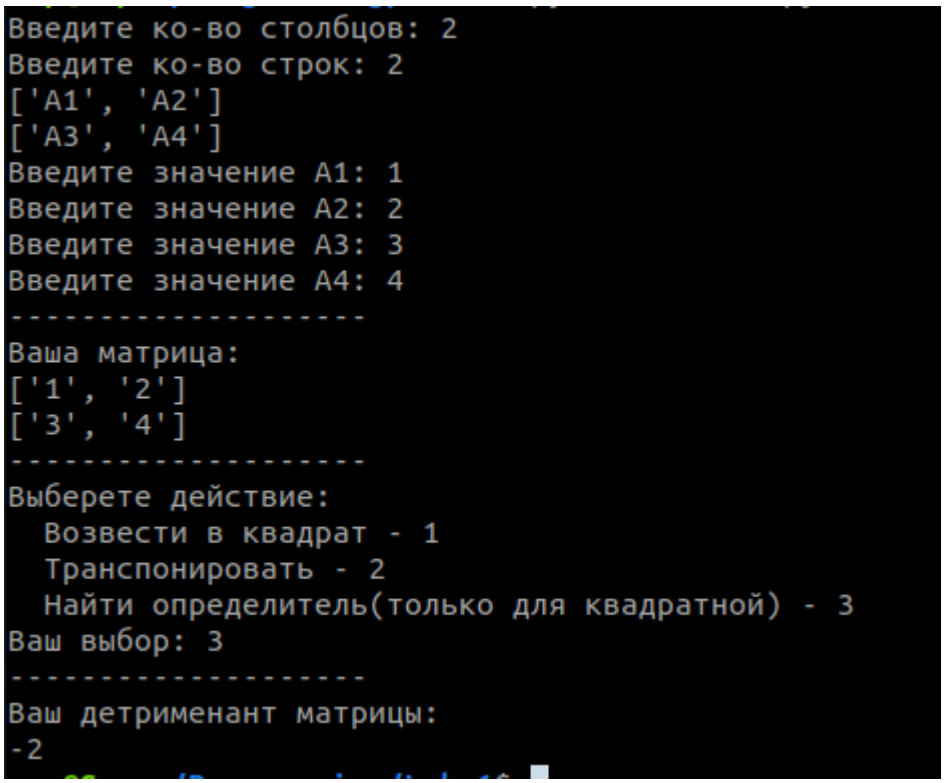


Рисунок 30 – результат нахождения определителя матрицы.

Вывод:

Работа с матрицами без дополнительных библиотек не располагает к себе.