**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Кафедра Систем Управления и Информатики

# Лабораторная работа №1

## Вариант №6

Выполнил:

Таякин Д.Р.

Проверил:

Мусаев А.А.

Санкт-Петербург, 2022

## 1 Задание

В первом задании нужно было реализовать алгоритм фасетного поиска для языковой группы. В качестве языков я выбрал языки программирования.

Для работы фасетного поиска был создан объект Language, который содержит имя языка, и словарь из нескольких параметров. Данные не являются полностью корректными, так как большинство языков программирования мультипарадигмальные. На рисунке 1.1 представлена часть кода первого задания, содержащий собранные данные и объект Language.

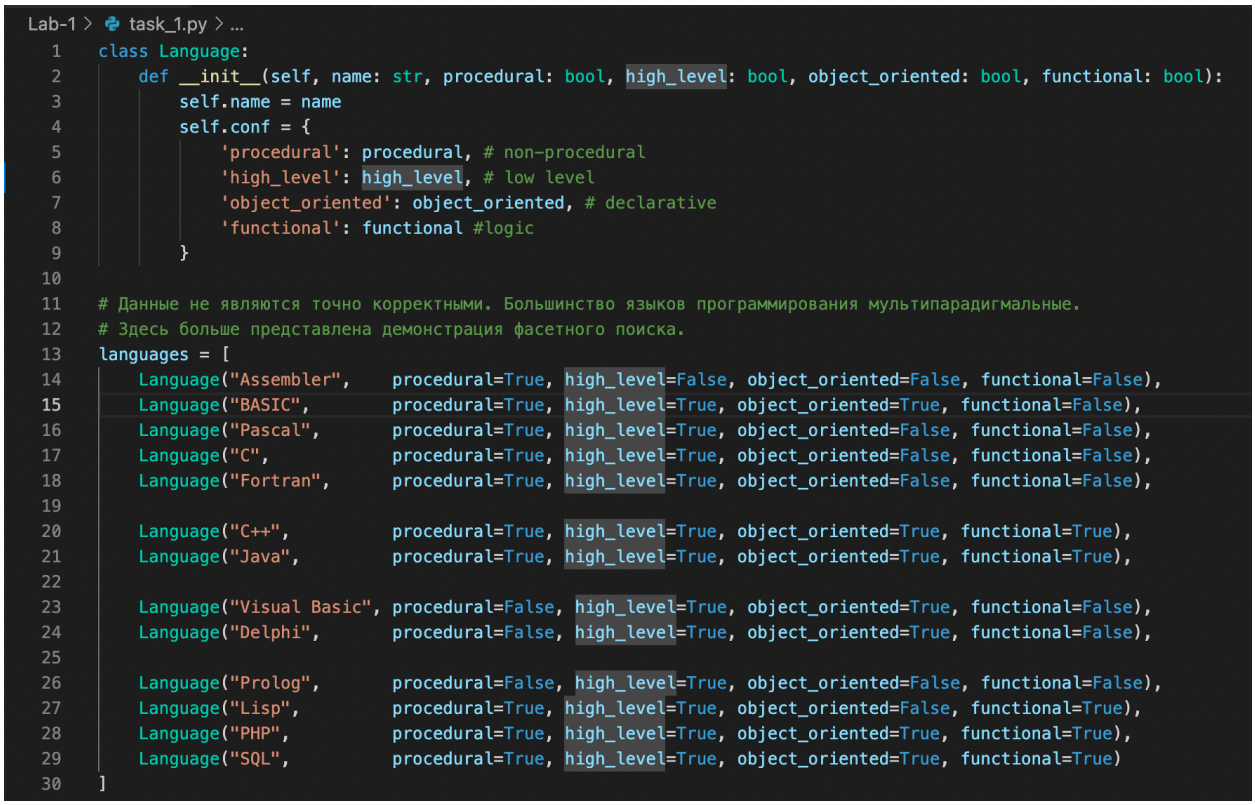


Рисунок 1.1 – Данные, используемые в первом задании

На рисунке 1.2 представлена вторая часть кода основной логики фасетного поиска. В качестве параметров используются тот же по типу словарь, что и в объекте Language. Через цикл пользователю задается вопрос в зависимости от ключевого слова параметра в словаре. Затем через цикл фильтруем по параметрам.

```python
if __name__ == "__main__":
    conf = {
        'procedural': True,
        'high_level': True,
        'object_oriented': False,
        'functional': False
    }

    def handle_ans(v) -> bool:
        return v == "Да"

    for key in conf:
        if key == "procedural":
            conf[key] = handle_ans(input("Является ли язык процедурным? "))
        elif key == "high_level":
            conf[key] = handle_ans(input("Является ли язык высокоуровневым? "))
        elif key == "object_oriented":
            conf[key] = handle_ans(input("Является ли язык объектно-ориентированным? "))
        elif key == "functional":
            conf[key] = handle_ans(input("Является ли язык функциональным? "))

    selected_languages = []
    for lang in languages:
        if conf == lang.conf:
            selected_languages.append(lang.name)

    print(f"Ответ: {', '.join(selected_languages)}")
```

Рисунок 1.2 – Основная логика фасетного поиска

## 2 Задание

Во втором задании нужно было реализовать алгоритм распределения чисел натуральные, целые, рациональные, вещественные, комплексные, четные, нечетные и простые.

На рисунке 2.1 показана первая часть кода, состоящая из принятие ввода данных пользователем и распределение этих данных по массивам в словаре. Также показана основная функция to_int, принимающая за аргумент строку. Данная функция превращает строку в целое число. В качестве приморов, она может перевести такие типы данных в число: “1/1”, “6.0”, “1”.

Lab-1 > task_2.py > is_complex

```python
numbers = list(input().split(","))

sorted_nums = {
    "natural": [],
    "integer": [],
    "rational": [],
    "real": [],
    "comp": [],
    "even": [],
    "odd": [],
    "prime": []
}

def to_int(num):
    try:
        if "/" in str(num):
            split_num = str(num).split("/")
            if float(split_num[0]) / float(split_num[1]) == 1:
                return float(split_num[0]) / float(split_num[1])

        if float(num).is_integer():
            return int(num)
        return None
    except:
        return None
```

Рисунок 2.1 – Первая часть кода

На рисунке 2.2 показана вторая часть кода, где представлены функции фильтрующие числа по нужным группам.

```
27  def is_complex(num):
28      if "j" in num:
29          sorted_nums["comp"].append(num)
30
31  def is_even_odd(num):
32      if to_int(num):
33          sorted_nums["even"].append(num) if to_int(num) % 2 == 0 else sorted_nums["odd"].append(num)
34
35  def is_natural(num):
36      n = to_int(num)
37      if n != None and n > 0:
38          sorted_nums["natural"].append(num)
39
40  def is_rational(num):
41      if "/" in num or "." in num or num == "0":
42          sorted_nums["rational"].append(num)
43
44  def is_real(num):
45      if "j" not in num:
46          sorted_nums["real"].append(num)
47
48  def is_prime(num):
49      n = to_int(num)
50      if n is None:
51          return False
52      if n <= 1:
53          return False
54      if n <= 3:
55          return True
56      if n % 2 == 0 or n % 3 == 0:
57          return False
58      i = 5
59      while(i * i <= n) :
60          if (n % i == 0 or n % (i + 2) == 0):
61              return False
62          i = i + 6
63      return True
64
65  for num in numbers:
66      is_natural(num)
67
68      if to_int(num):
69          sorted_nums["integer"].append(num)
70
71      is_even_odd(num)
72      is_complex(num)
73      is_rational(num)
74      is_real(num)
75
76      if is_prime(num):
77          sorted_nums["prime"].append(num)
78
79  for key, value in sorted_nums.items():
80      print(f"{key}:", ", ".join(value))
```

Рисунок 2.2 – Вторая часть кода

## 3 Задание

В третьем задании нужно было реализовать алгоритмы сортировки чисел: пузырьковый, гномий, блочный, пирамидальный и проанализировать достоинства и недостатки.

```
3   def bubble_sort(arr):
4       sorted_arr = arr.copy()
5       n = len(sorted_arr)
6       for i in range(n):
7           swapped = False
8           for j in range(0, n-i-1):
9               if sorted_arr[j] > sorted_arr[j+1]:
10                  sorted_arr[j], sorted_arr[j+1] = sorted_arr[j+1], sorted_arr[j]
11                  swapped = True
12          if swapped == False:
13              break
14      return sorted_arr
```

Рисунок 3.1 – Пузырьковая сортировка

На рисунке 3.1 представлена пузырьковая сортировка. В худшем случае, алгоритм работает за $O(n^2)$. В лучшем за $O(n)$. Пузырьковая сортировка является простейшим алгоритмом и эффективен для небольших массивов.

```
16  def gnome_sort(arr):
17      sorted_arr = arr.copy()
18      i = 0
19      while i < len(sorted_arr):
20          if i == 0:
21              i += 1
22          if sorted_arr[i] >= sorted_arr[i - 1]:
23              i += 1
24          else:
25              sorted_arr[i], sorted_arr[i-1] = sorted_arr[i-1], sorted_arr[i]
26              i -= 1
27      return sorted_arr
```

Рисунок 3.2 – Гномья сортировка

На рисунке 3.2 представлена гномья сортировка, которая похожа на сортировку вставками, но в отличие от последней перед вставкой на нужное место происходит серия обменов, как в сортировке пузырьком. Время работы в худшем случае за $O(n^2)$, а в лучшем за $O(n)$.

На рисунке 3.3 изображена пирамидальная сортировка. Худшеее и лучшее время выполнения: $O(nlogn)$. На почти отсортированных массивах работает столь же долго, как и на хаотических данных.

```
29  def heapify(arr, n, i):
30      largest = i
31      l = 2 * i + 1 # left
32      r = 2 * i + 2 # right
33
34      # if left child > root
35      if l < n and arr[largest] < arr[l]:
36          largest = l
37
38      # if right child > root
39      if r < n and arr[largest] < arr[r]:
40          largest = r
41
42      # change root, if needed
43      if largest != i:
44          arr[i], arr[largest] = arr[largest], arr[i]
45          heapify(arr, n, largest)
46
47  def heap_sort(arr):
48      sorted_arr = arr.copy()
49      n = len(sorted_arr)
50
51      # building a maxheap
52      for i in range(n // 2 - 1, -1, -1):
53          heapify(sorted_arr, n, i)
54
55      # extracting elements
56      for i in range(n - 1, 0, -1):
57          sorted_arr[i], sorted_arr[0] = sorted_arr[0], sorted_arr[i]
58          heapify(sorted_arr, i, 0)
59
60      return sorted_arr
```

Рисунок 3.3 – Пирамидальная сортировка

На рисунке 3.4 представлен блочный алгоритм сортировки, в котором сортируемые элементы распределяются между конечным числом отдельных блоков так, чтобы все элементы в каждом следующем по порядку блоке были всегда больше (или меньше), чем в предыдущем. Преимущества: относится к

классу быстрых алгоритмов с линейным временем исполнения $O(n)$ (на удачных входных данных).

```
def bucket_sort(arr, n):
    s_arr = arr.copy()
    max_ele = max(s_arr)
    min_ele = min(s_arr)

    # range(for buckets)
    rnge = (max_ele - min_ele) / n

    temp = []

    # create empty buckets
    for i in range(n):
        temp.append([])

    for i in range(len(s_arr)):
        diff = (s_arr[i] - min_ele) / rnge - int((s_arr[i] - min_ele) / rnge)

        if diff == 0 and s_arr[i] != min_ele:
            temp[int((s_arr[i] - min_ele) / rnge) - 1].append(s_arr[i])
        else:
            temp[int((s_arr[i] - min_ele) / rnge)].append(s_arr[i])

    # sorting each bucket
    for i in range(len(temp)):
        if len(temp[i]) != 0:
            temp[i].sort()

    # creating sorted array
    k = 0
    for lst in temp:
        if lst:
            for i in lst:
                s_arr[k] = i
                k = k+1

    return s_arr
```

Рисунок 3.4 – Блочная сортировка

## 4 Задание

В 4 задании нужно было реализовать алгоритм для решения задачи с теорией вероятности.

Зная, что с третьекурсниками A происходит событие P за C дней, нам нужно узнать с какой вероятностью не произойдет событие за 1 день с каждым из одногруппников: $P_i = (1 - P)^{1/C}$. Теперь нужно вычислить наступление события за D дней для третьекурсника E по полученной формуле: $P_E = P_1^D * P_2^D * (1 - P_3^D)$.

```python
from math import pow

# дни за которые может с ними произойти событие
C = 6
# дни за которые может произойти событие с Полиной
D = 206

students = {
    "Polyna": 0,
    "Lesha": 0,
    "Sveta": 0
}

ans_chance = 1
for name, chance in students.items():
    n = input(f"Вероятность для {name}: ")
    chance = pow(1 - float(n), (1 / C))

    if name == "Polyna":
        ans_chance *= pow((1 - chance), D)
    else:
        ans_chance *= pow(chance, D)

print(ans_chance)

```

Рисунок 4.1 – Код четвертого задания

## 5 Задание

В данном задании нужно было реализовать алгоритм, заполняющий таблицу неповторяющимися координатами x и y в заданном пользователем диапазоне. Координаты записать в Excel файле. Затем для заданных координат применить метод наименьших квадратов и построить график, используя библеотеку matplotlib.

Lab-1 > task_5.py > create_excel_file

```python
import matplotlib.pyplot as plt
import random, openpyxl
from task_6 import *

n = 25

def create_excel_file(x: list, y: list):
    wb = openpyxl.Workbook()
    sheet = wb.active

    sheet.cell(1, 1, "x")
    sheet.cell(1, 2, "y")

    for i, (coord_x, coord_y) in enumerate(zip(x, y)):
        sheet.cell(2+i, 1, coord_x)
        sheet.cell(2+i, 2, coord_y)

    wb.save("coords.xlsx")

def mnk(x: list, y: list):
    sum_x = sum(x); sum_y=sum(y)
    sum_xy = sum(map(lambda x, y: x * y, x, y))
    sum_x2 = sum(map(lambda x: x ** 2, x))

    det_m = Matrix(2, 2, [sum_x2, sum_x, sum_x, n]).det()
    det_a = Matrix(2, 2, [sum_xy, sum_x, sum_y, n]).det()
    det_b = Matrix(2, 2, [sum_x2, sum_xy, sum_x, sum_y]).det()

    a = det_a / det_m
    b = det_b / det_m

    return a, b

```

Рисунок 5.1 – Функции и используемые модули

На рисунке 5.1 представлена часть кода, где происходит импортирование нужных модулей и объявление двух важных функций:

```python
    def det(self):
        if self.cols != self.rows:
            print('Number of columns must be equal to number of rows.')
            return

        if self.rows == 1:
            return self.mat[0]
        elif self.rows == 2:
            return self.mat[0][0] * self.mat[1][1] - self.mat[1][0] * self.mat[0][1]
        else:
            summ = 0
            for i in range(self.cols):
                minor = self.minor(0, i)
                flat_minor_list = [item for sublist in minor for item in sublist]
                summ += ((-1)**i) * self.mat[0][i] * Matrix(len(minor), len(minor[0]), flat_minor_list).det()
            return summ

    def minor(self, i, j):
        return [row[:j] + row[j+1:] for row in (self.mat[:i] + self.mat[i+1:])]

    def squared(self):
        if self.cols != self.rows:
            print('Number of columns must be equal to number of rows.')
            return

        C = self.zeros_matrix(self.rows, self.cols)
        for i in range(self.rows):
            for j in range(self.cols):
                total = 0
                for ii in range(self.cols):
                    total += self.mat[i][ii] * self.mat[ii][j]
                C[i][j] = total

        return C

    def zeros_matrix(self, rows, cols):
        return [[0 for _ in range(cols)] for _ in range(rows)]
```

Рисунок 6.2 – Вторая часть кода класса Matrix

```python
if __name__ == "__main__":
    def create_matrix():
        print("Create Matrix")

        rows = int(input("Rows: "))
        cols = int(input("Columns: "))
        str_data = input("Enter numbers with spaces: ")

        data = list(map(int, str_data.split(' ')))
        return Matrix(rows, cols, data)

    M = create_matrix()
    while True:
        print("""
Choose an option:
1. square matrix
2. transpose
3. find determinant
        """)
        i = int(input())

        if i == 1:
            print(M.squared())
        elif i == 2:
            print(M.transpose())
        elif i == 3:
            print(M.det())
        else:
            print(f"Can't find {i} option. Choose an option that is listed.")
```

Рисунок 6.3 – Код ввода и обработки данных через функции класса Matrix

создание и запись в Excel файл координат и вычисление коэфициентов a и b. В функци mnk для вычисления определителй я использовал написанный код в 6 задании.

```
35  if __name__ == "__main__":
36      min_v, max_v = map(int, input().split())
37
38      x = []; y = []
39      for _ in range(n):
40          x.append(random.uniform(min_v, max_v))
41          y.append(random.uniform(min_v, max_v))
42
43      create_excel_file(x, y)
44
45      a, b = mnk(x, y)
46
47      plt.plot([min_v, max_v], [a*min_v + b, a*max_v + b], color="#CE7A60")
48
49      plt.scatter(x, y)
50      plt.show()
51
52      print(x, y)
53
```

Рисунок 5.2 – Код, принимающий входные данные и использует функциии для построения графика

## 6 Задание

В данном задании нужно было написать алгоритм, позволяюзий вводить матрицу и применять различные операции: возведение в квадрат, транспонирование, нахождение определителя.

На рисунках 6.1 и 6.2 показаны части кода класса матрицы. В первой части представлен конструктор и метод транспонирования. Во второй части кода представлены функция нахождения детерминанта, функция возведения матрицы в квадрат и вспомогательные функции.

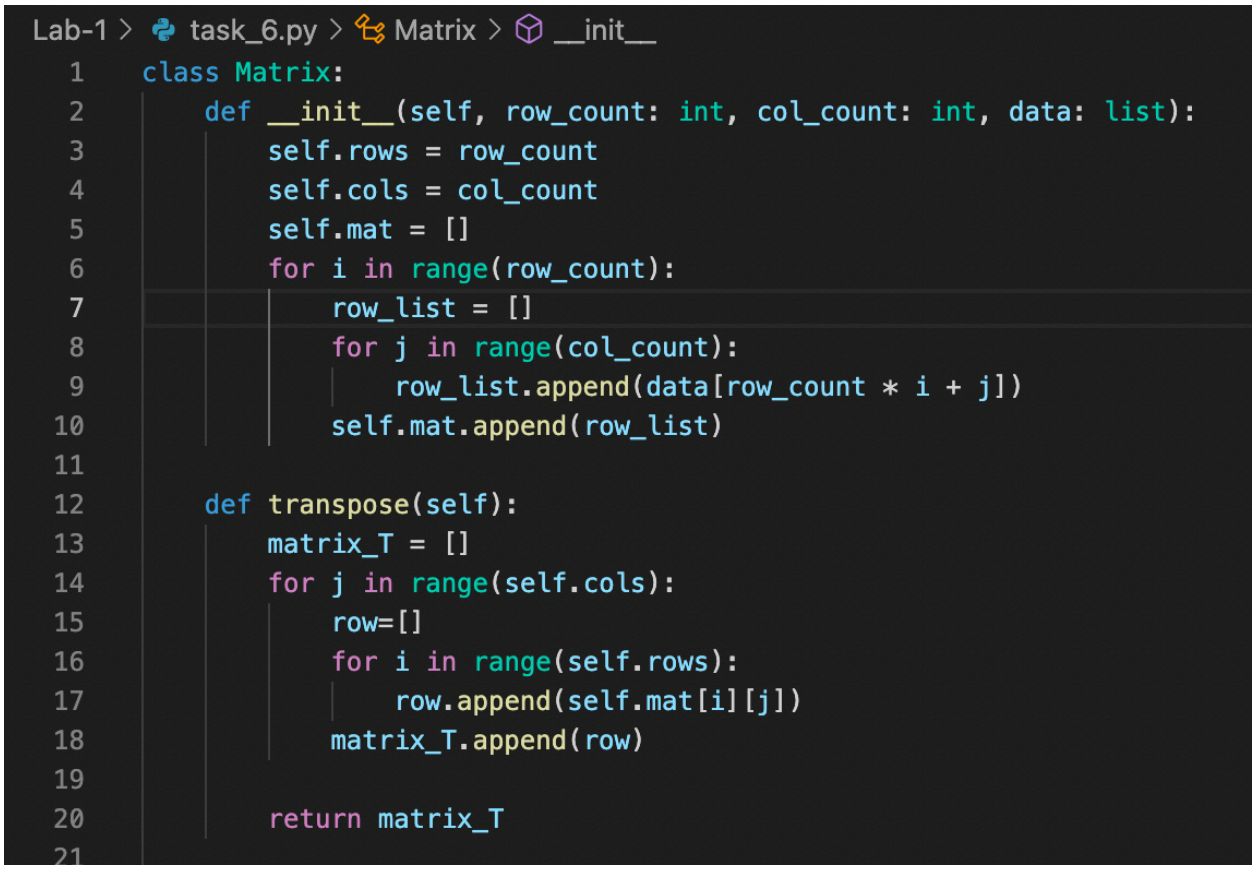


Рисунок 6.1 – Первая часть кода класса Matrix

На рисунке 6.3 представлен код ввода пользователем значений матрицы и обработка последующих действий через класс Matrix.