САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №4

по курсу «Алгоритмы и структуры данных»

Тема: Стек, очередь, связанный список

Выполнила:

Просветова Валерия Дмитриевна

К3141

Проверила:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

Задача №1: Стек

Задача №4: Скобочная последовательность.Версия 2

Задача №6: Очередь с минимум

Задача №7: Максимум в движущейся последовательности

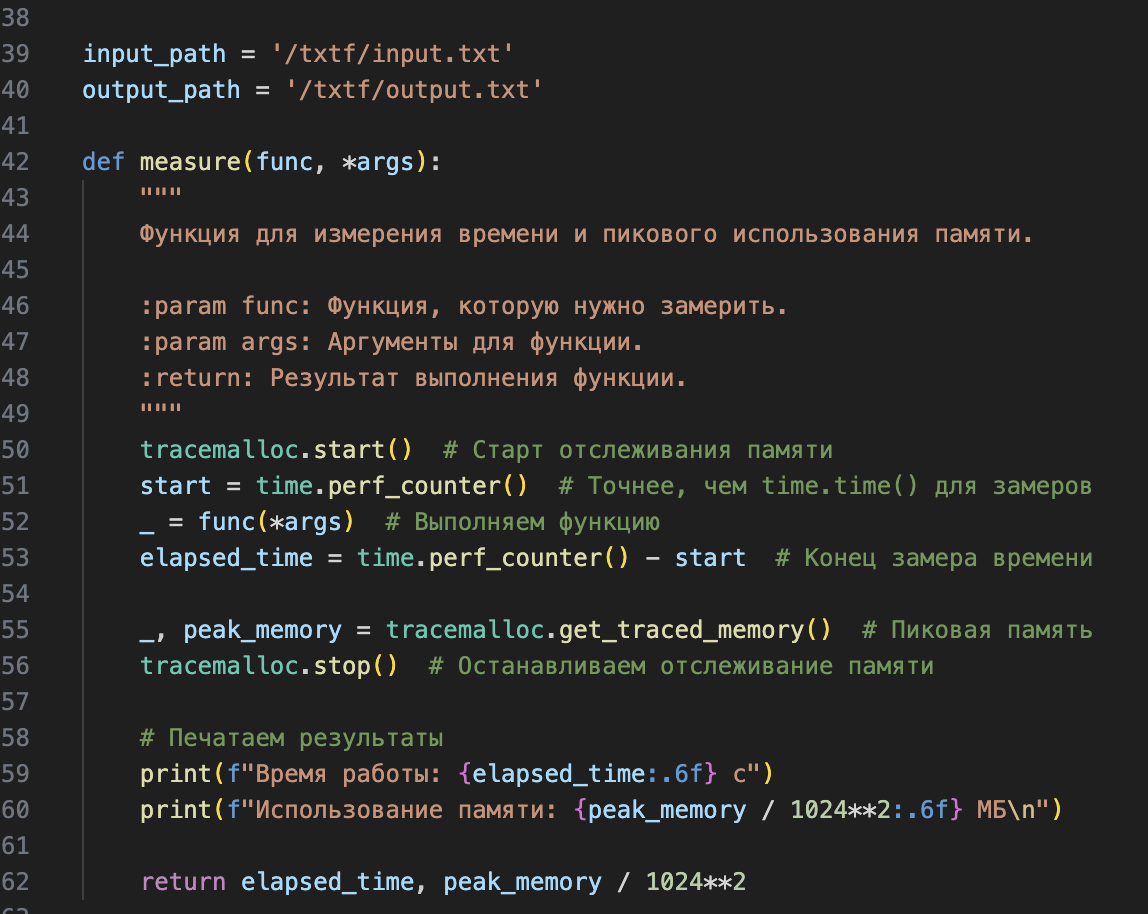
Задача №8: Постфиксная запись

Задача №13: Реализация стека, очереди и связанных списков

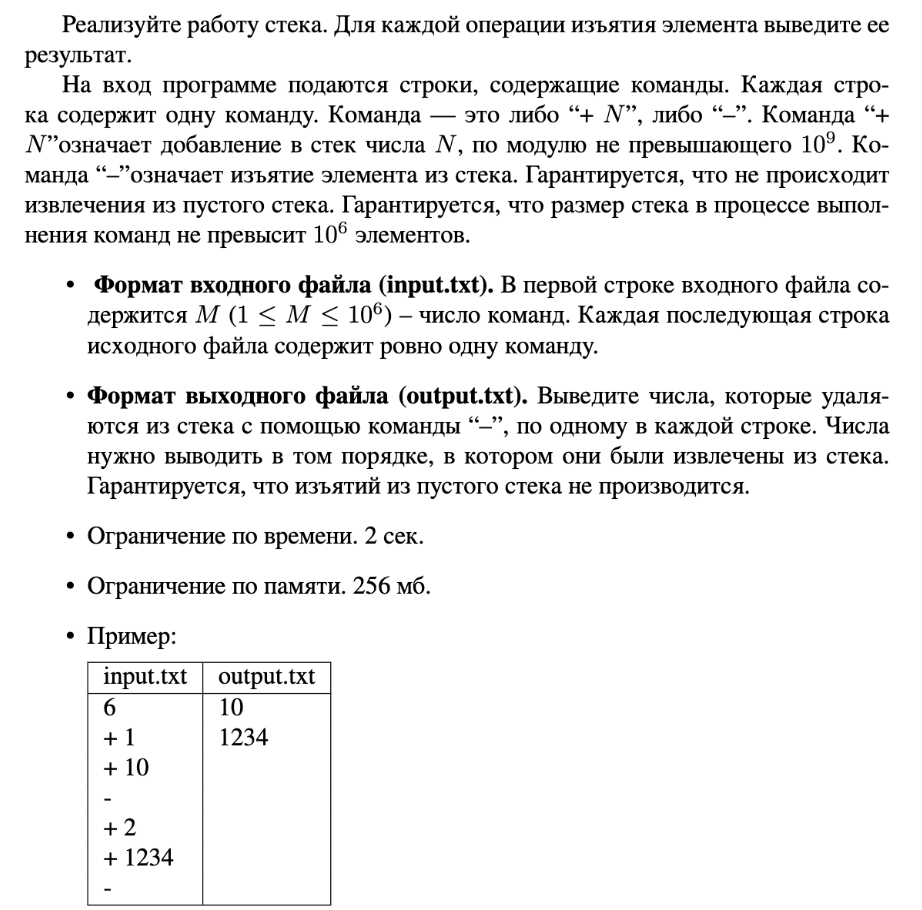
Вывод

Использовала файл utils.py для исключения повторения строчек кода:

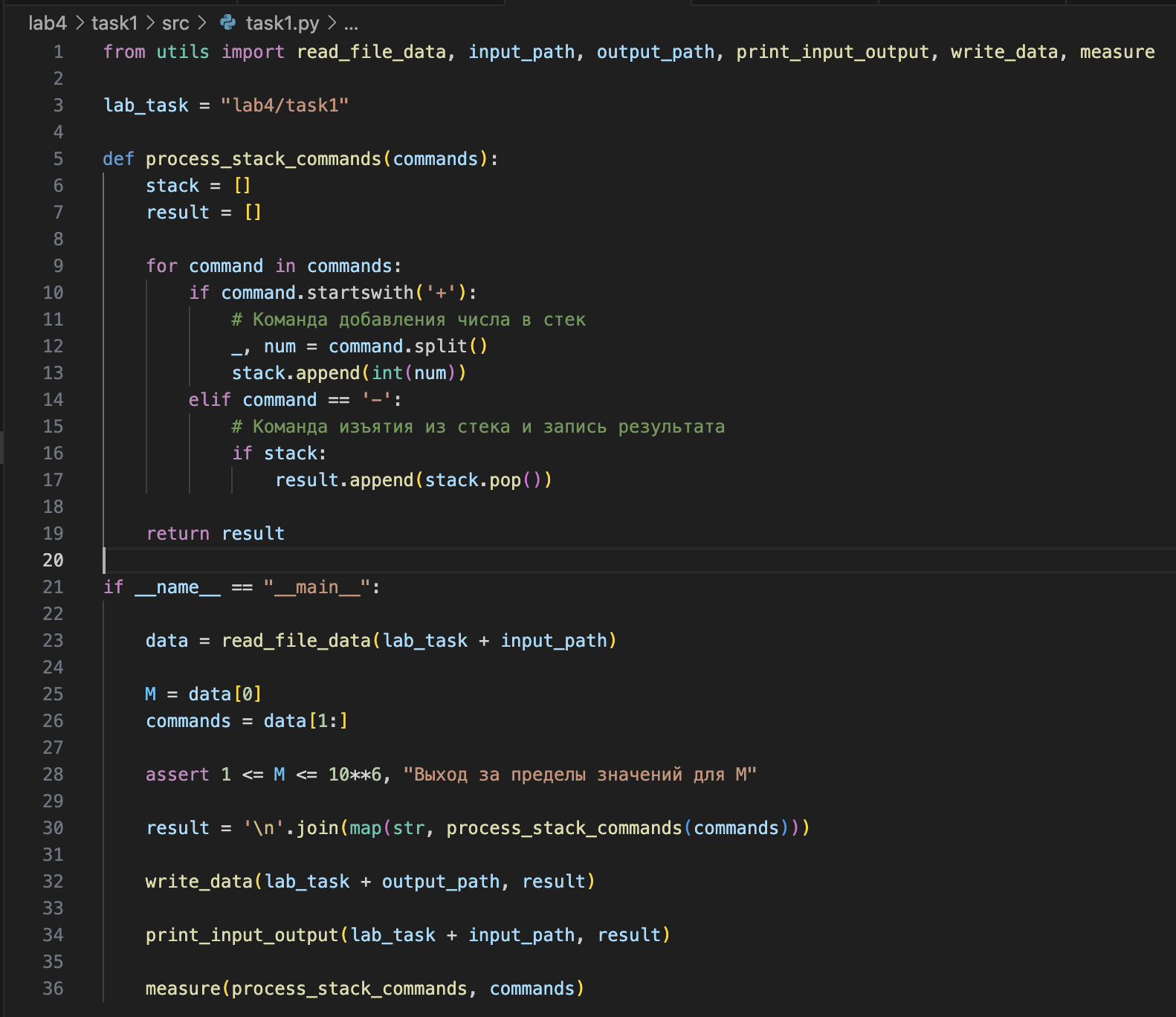




Задача №1:



Решение:



**Шаг 1: Создание пустого стека и списка результатов**

В этом шаге я создаю два пустых списка: stack и result. Список stack будет использоваться для хранения чисел, а список result будет использоваться для хранения результатов выполнения команд.

**Шаг 2: Обработка команд и проверка команды на добавление числа в стек**

**for command in commands:**

**if command.startswith('+'):**

Здесь начинаю обрабатывать каждую команду в списке commands. Мы используем цикл for для перебора команд. Проверяю, начинается ли команда с символа +. Если да, то это означает, что нужно добавить число в стек.

**Шаг 3: Разделение команды на число и добавление в стек**

**\_, num = command.split()**

**stack.append(int(num))**

Разбиваю command на два элемента: символ + и число. Я использую метод split() для разделения команды на два элемента. Затем преобразовала число в целое с помощью функции int() и добавила его в стек с помощью метода append().

**Шаг 4: Проверка команды на изъятие из стека**

**elif command == '-':**

В этом шаге проверяю, равна ли команда символу -. Если да, то это означает, что нужно извлечь число из стека.

**Шаг 5: Извлечение числа из стека и добавление в список результатов**

**if stack:**

**result.append(stack.pop())**

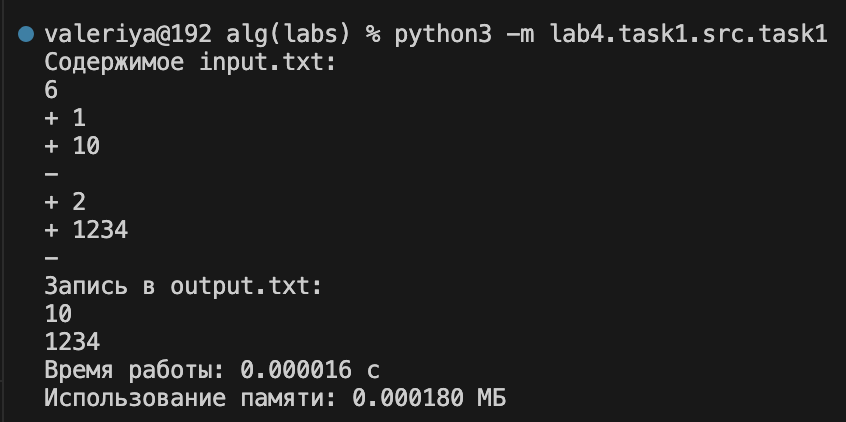
Проверяю, что стек не пустой. Если стек не пустой, то извлекаю число из стека с помощью метода pop() и добавляю его в список result с помощью метода append().

**Шаг 7: Возвращение результата**

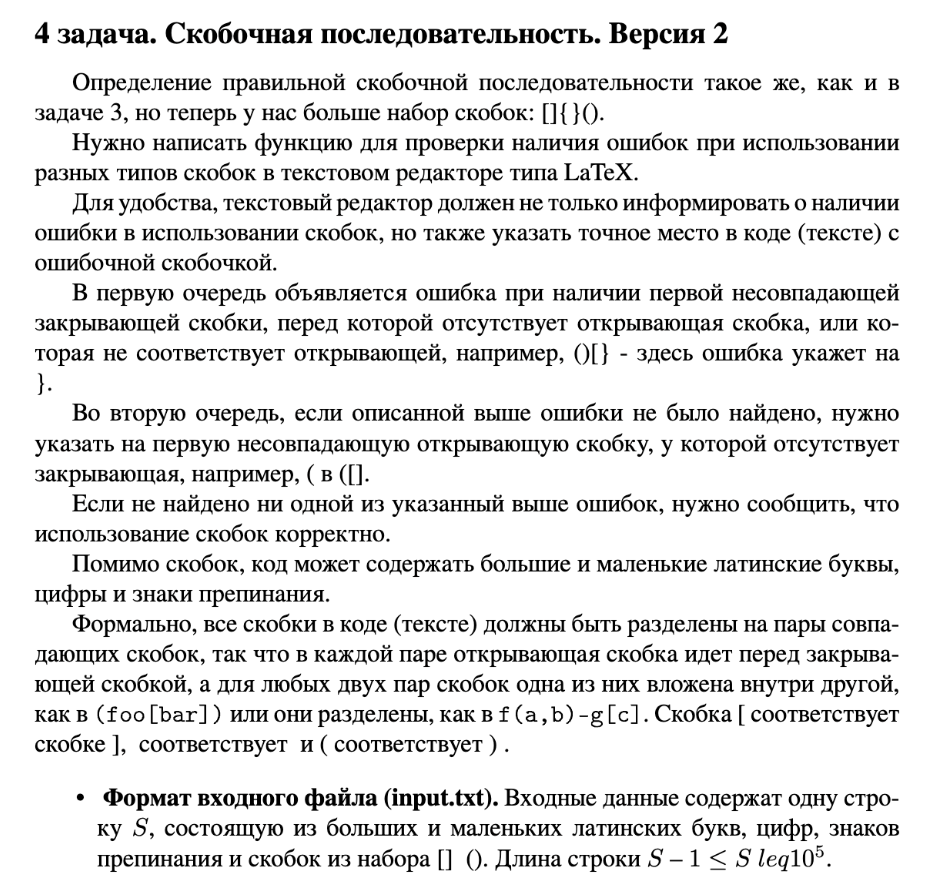
Возвращаю список result, который содержит результаты выполнения команд.

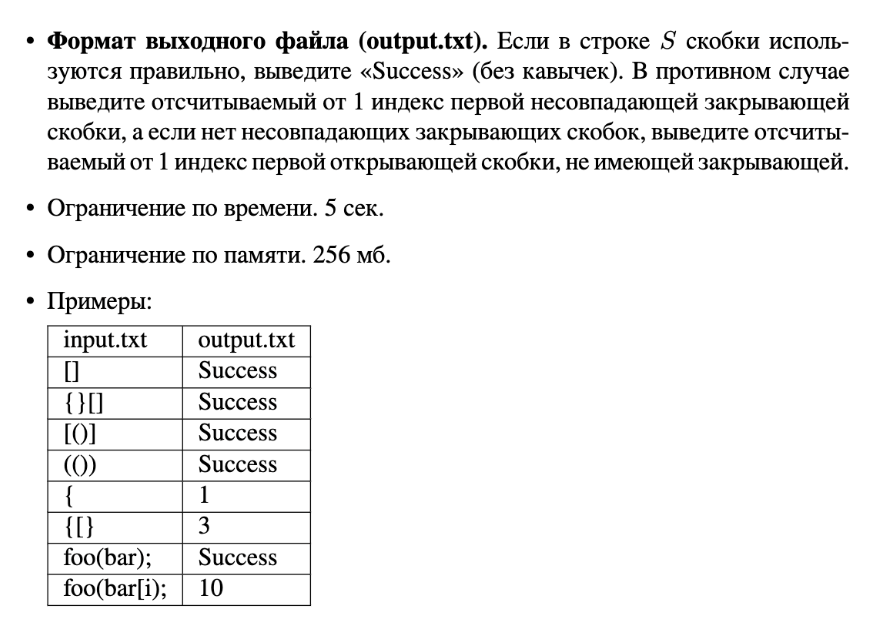
**Шаг 8:**

С помощью utils.py считываю данные из input.txt и записываю результат в output.txt

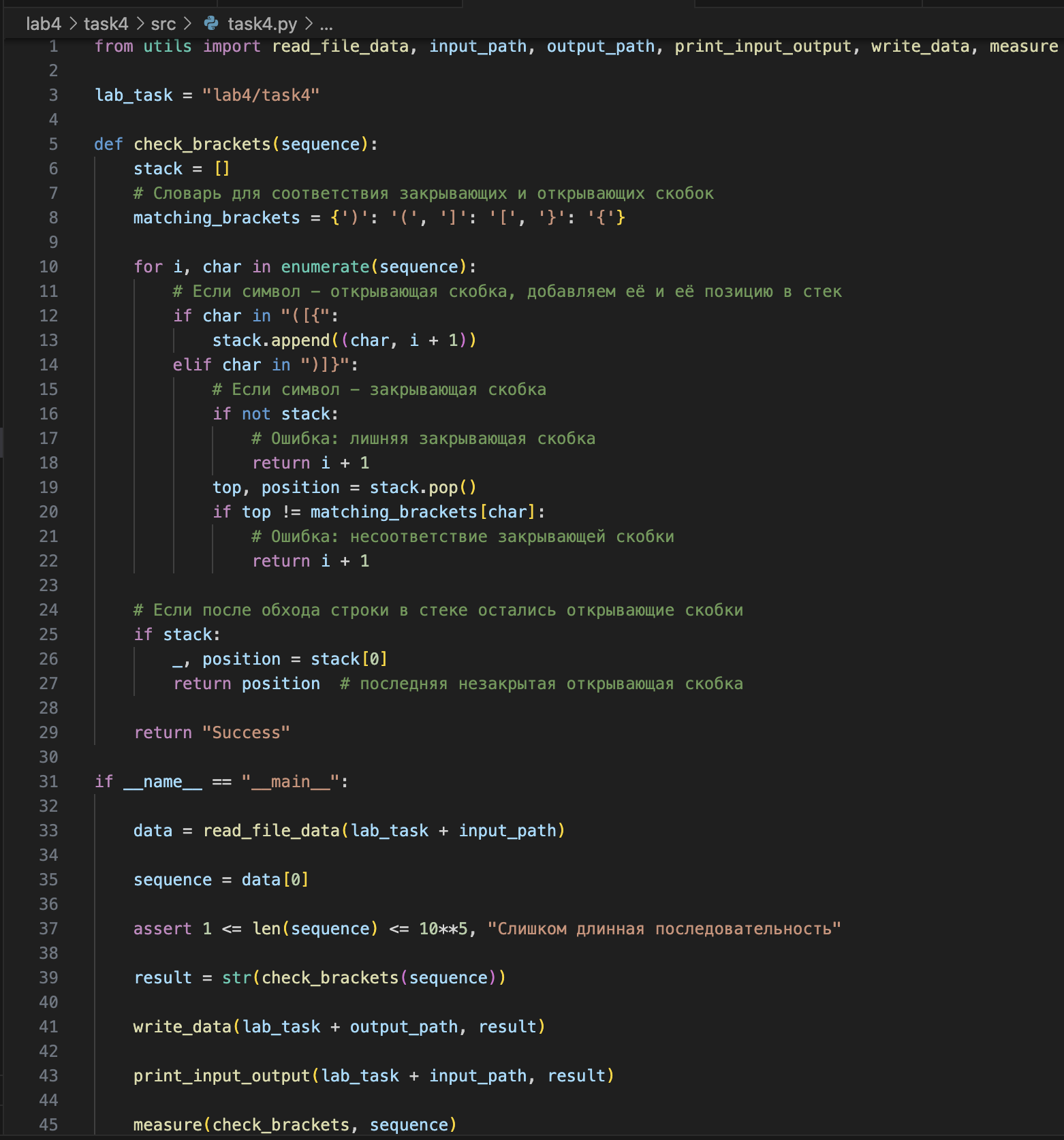


Задача №4





Решение:



**Шаг 1: Создание пустого стека**

stack = []

В этом шаге я создаю пустой стек, который будет использоваться для хранения открывающих скобок и их позиций.

**Шаг 2: Создание словаря соответствия скобок**

**matching\_brackets = {')': '(', ']': '[', '}': '{'}**

Создаю словарь, который хранит соответствие между закрывающимися и открывающимися скобками.

**Шаг 3: Обход строки**

**for i, char in enumerate(sequence):**

Начинаю обход строки sequence с помощью цикла for.Я использую функцию enumerate для получения позиции каждого символа в строке.

**Шаг 4: Проверка открывающих скобок**

**if char in "([{":**

**stack.append((char, i + 1))**

Проверяю, является ли текущий символ открывающей скобкой. Если да, то добавляю ее в стек вместе с ее позицией.

**Шаг 5: Проверка закрывающих скобок**

**elif char in ")]}":**

**if not stack:**

**return i + 1**

**top, position = stack.pop()**

**if top != matching\_brackets[char]:**

**return i + 1**

В этом шаге я проверяю, является ли текущий символ закрывающей скобкой. Если да, то :

* Стек не пустой. Если он пустой, то это означает, что закрывающая скобка не соответствует ни одной открывающей скобке, и я возвращаю позицию ошибки.
* Верхний элемент стека соответствует текущей закрывающей скобке. Если они не соответствуют, то это означает, что закрывающая скобка не соответствует открывающей скобке, и я возвращаю позицию ошибки.

**Шаг 6: Проверка оставшихся открывающих скобок**

**if stack:**

**\_, position = stack[0]**

**return position**

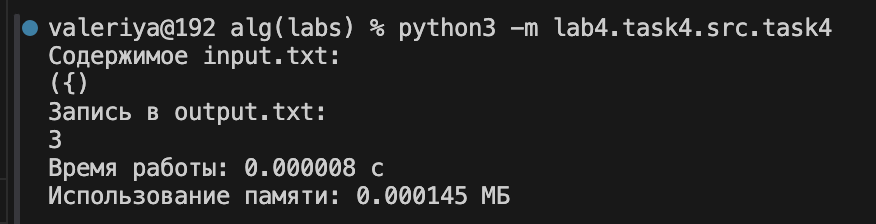
Проверяю, что после обхода строки в стеке остались открывающие скобки. Если да, то это означает, что эти скобки не были закрыты, и возвращаю позицию последней незакрытой открывающей скобки.

**Шаг 7: Возвращение результата**

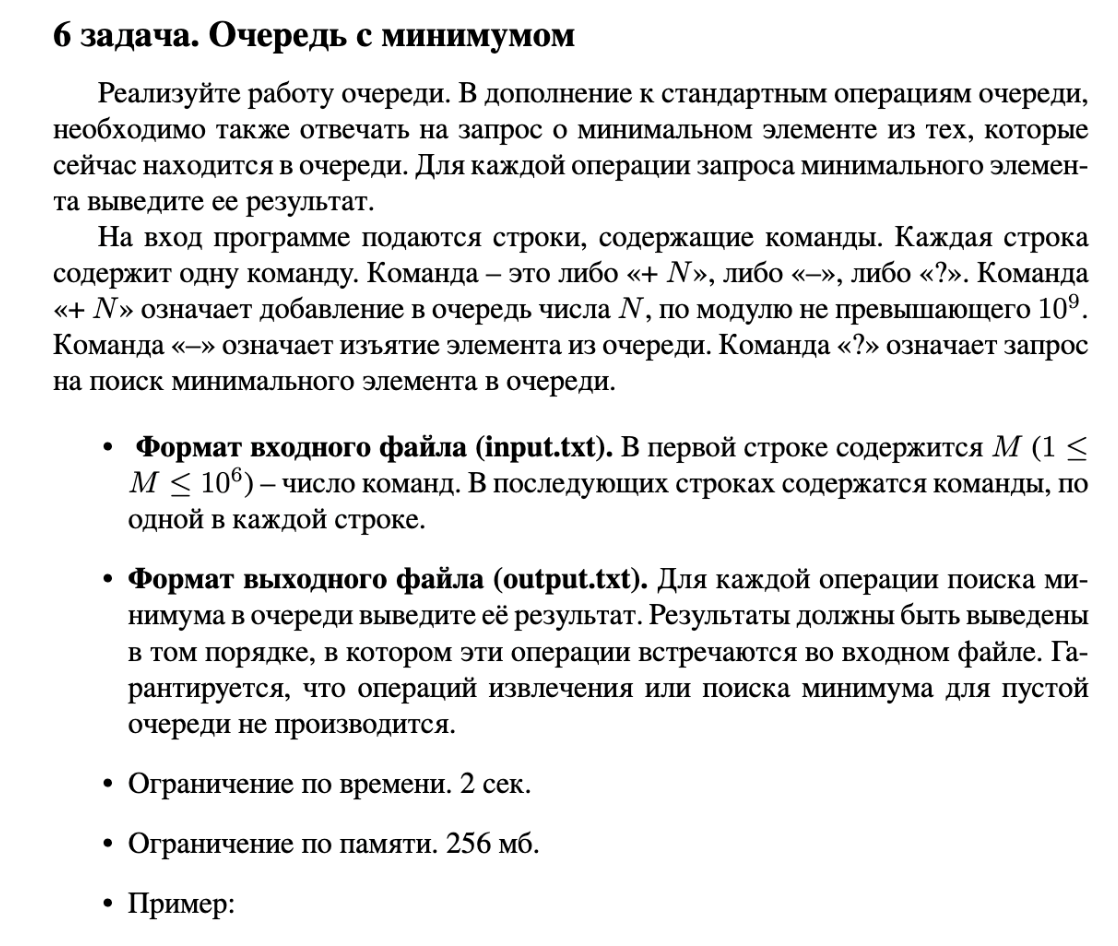
**return "Success"**

Здесь возвращаю строку "Success", если все скобки были правильно использованы.

**Шаг 8: Считываю данные из input.txt и записываю их в output.txt**

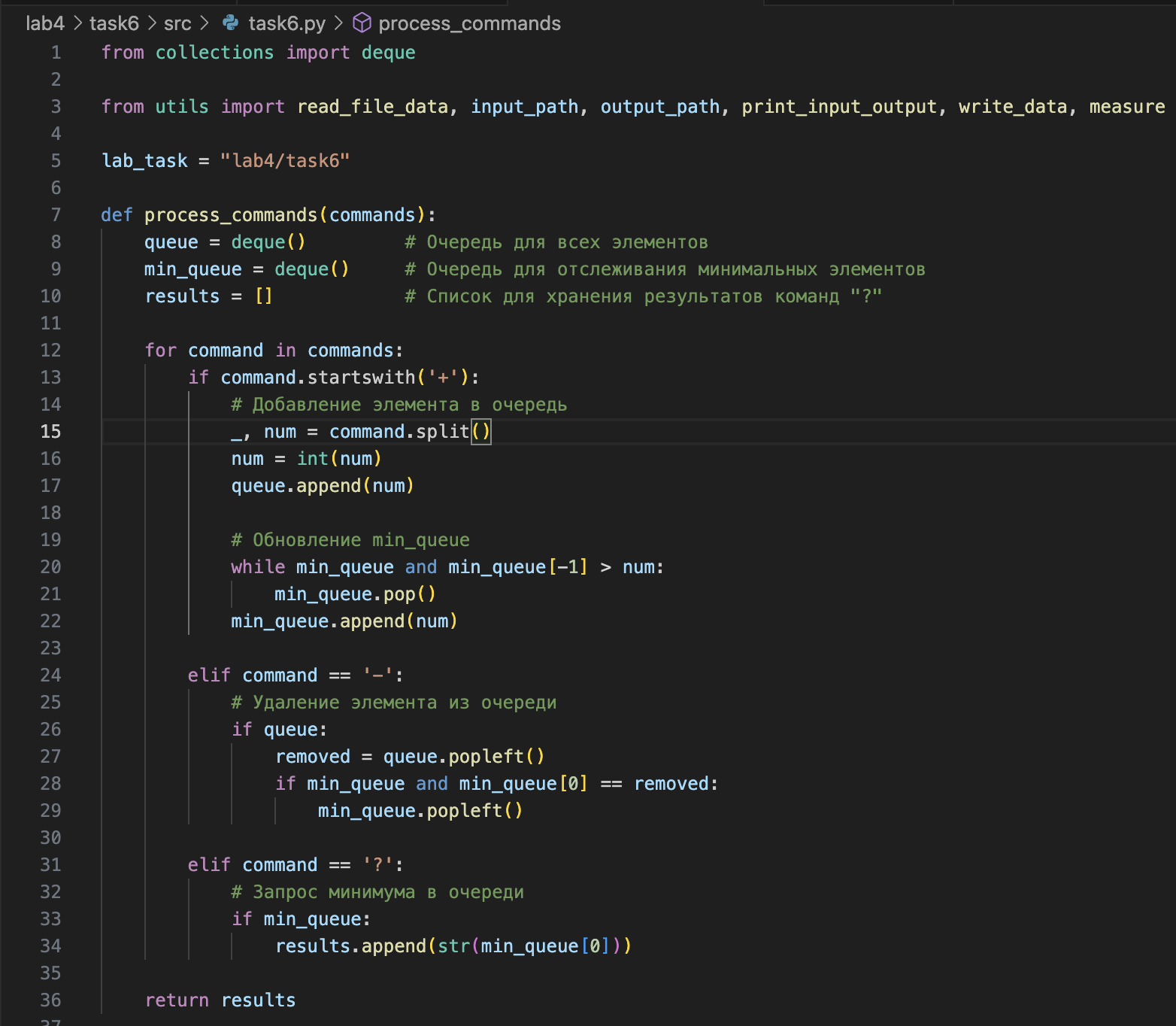
****

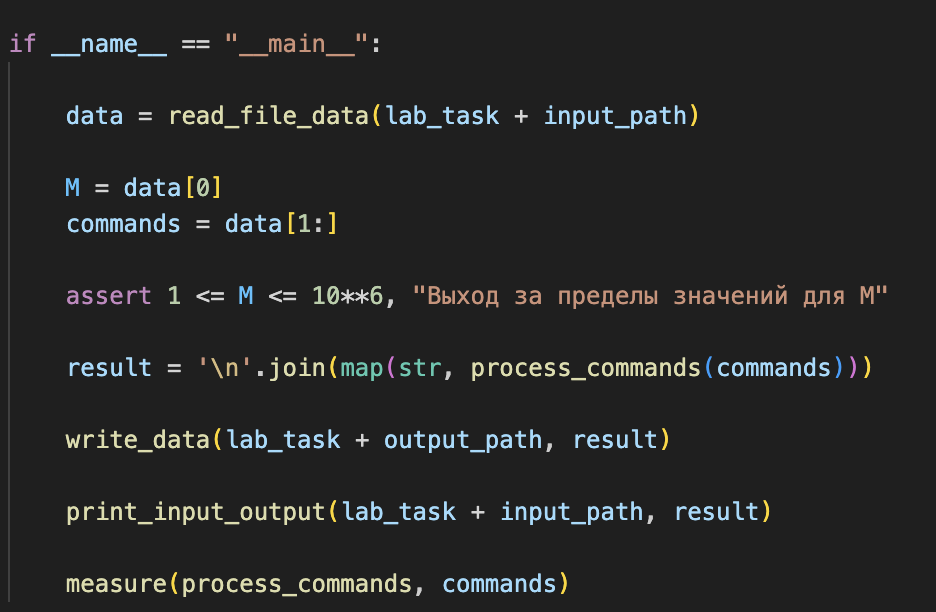
Задача №6

****

****

Решение:





**Шаг 1: Создание пустых очередей и списка результатов**

**queue = deque()**

**min\_queue = deque()**

**results = []**

В этом шаге я создаю три пустых структуры данных:

* queue: очередь для хранения всех элементов
* min\_queue: очередь для хранения минимальных элементов
* results: список для хранения результатов команд "?"

**Шаг 2: Обработка команд**

**for command in commands:**

Я начинаю обрабатывать каждую команду в списке commands.

**Шаг 3: Добавление элемента в очередь**

**if command.startswith('+'):**

**\_, num = command.split()**

**num = int(num)**

**queue.append(num)**

**# Обновление min\_queue**

**while min\_queue and min\_queue[-1] > num:**

**min\_queue.pop()**

**min\_queue.append(num)**

Я проверяю, начинается ли команда с символа "+". Если да, то :

* Разделяю command на два элемента: символ "+" и число
* Преобразую число в целое
* Добавляю число в очередь queue
* Обновляю очередь min\_queue для хранения минимальных элементов

**Шаг 4: Удаление элемента из очереди**

**elif command == '-':**

**# Удаление элемента из очереди**

**if queue:**

**removed = queue.popleft()**

**if min\_queue and min\_queue[0] == removed:**

**min\_queue.popleft()**

Тут проверяю, равна ли команда символу "-". Если да, то :

* Удаляю элемент из очереди queue
* Если удаленный элемент является минимальным элементом в очереди min\_queue, то удаляю его из очереди min\_queue

**Шаг 5: Запрос минимума в очереди**

**elif command == '?':**

**# Запрос минимума в очереди**

**if min\_queue:**

**results.append(str(min\_queue[0]))**

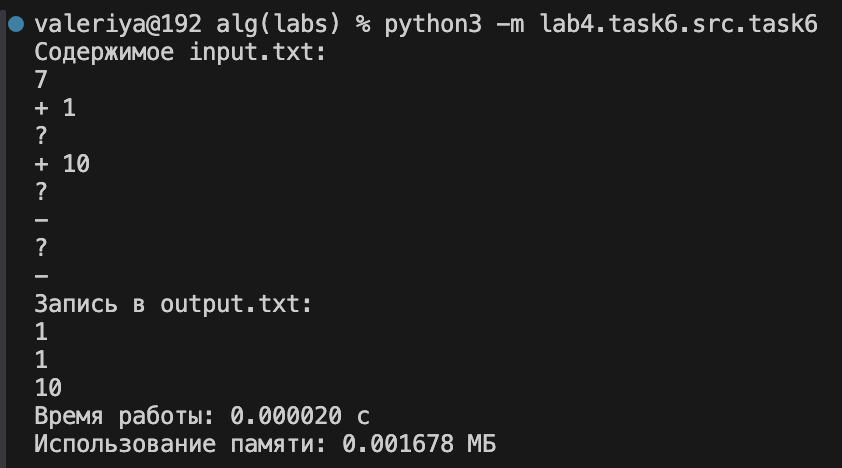
Я анализирую, равна ли команда символу "?". Если да, то :

* Проверяю, что очередь min\_queue не пуста
* Если очередь min\_queue не пуста, то добавляю минимальный элемент в список results

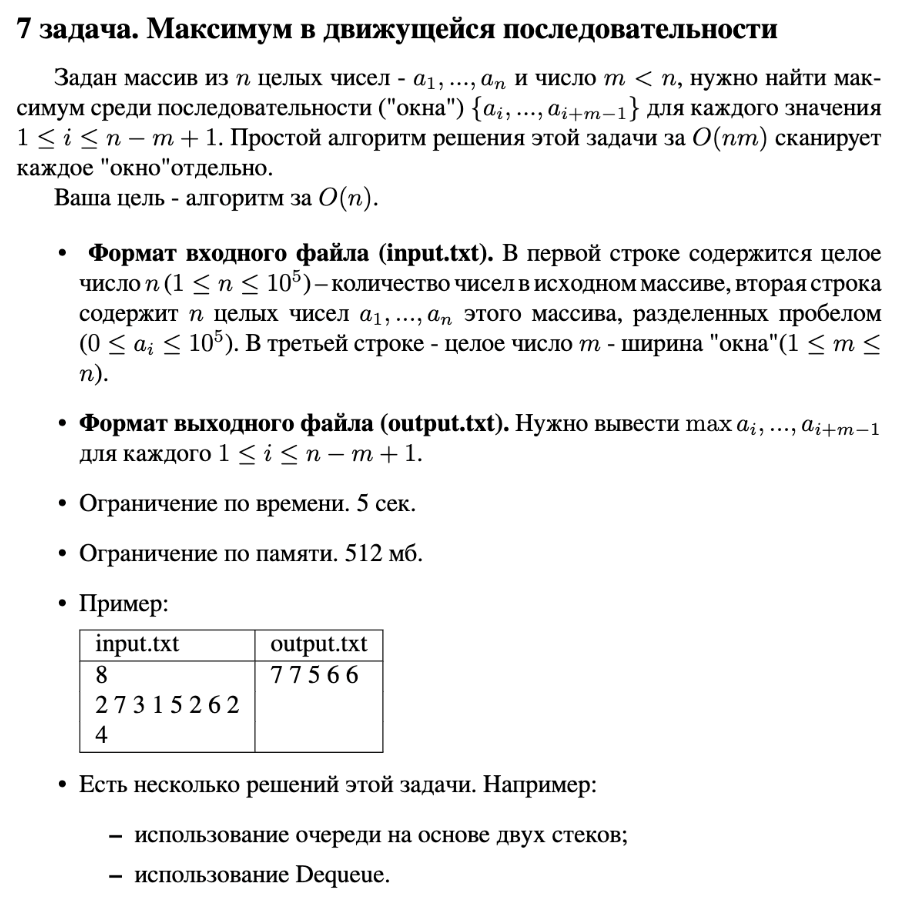
**Шаг 6: Возвращение результатов**

В этом шаге я возвращаю список results, который содержит результаты выполнения команд "?".

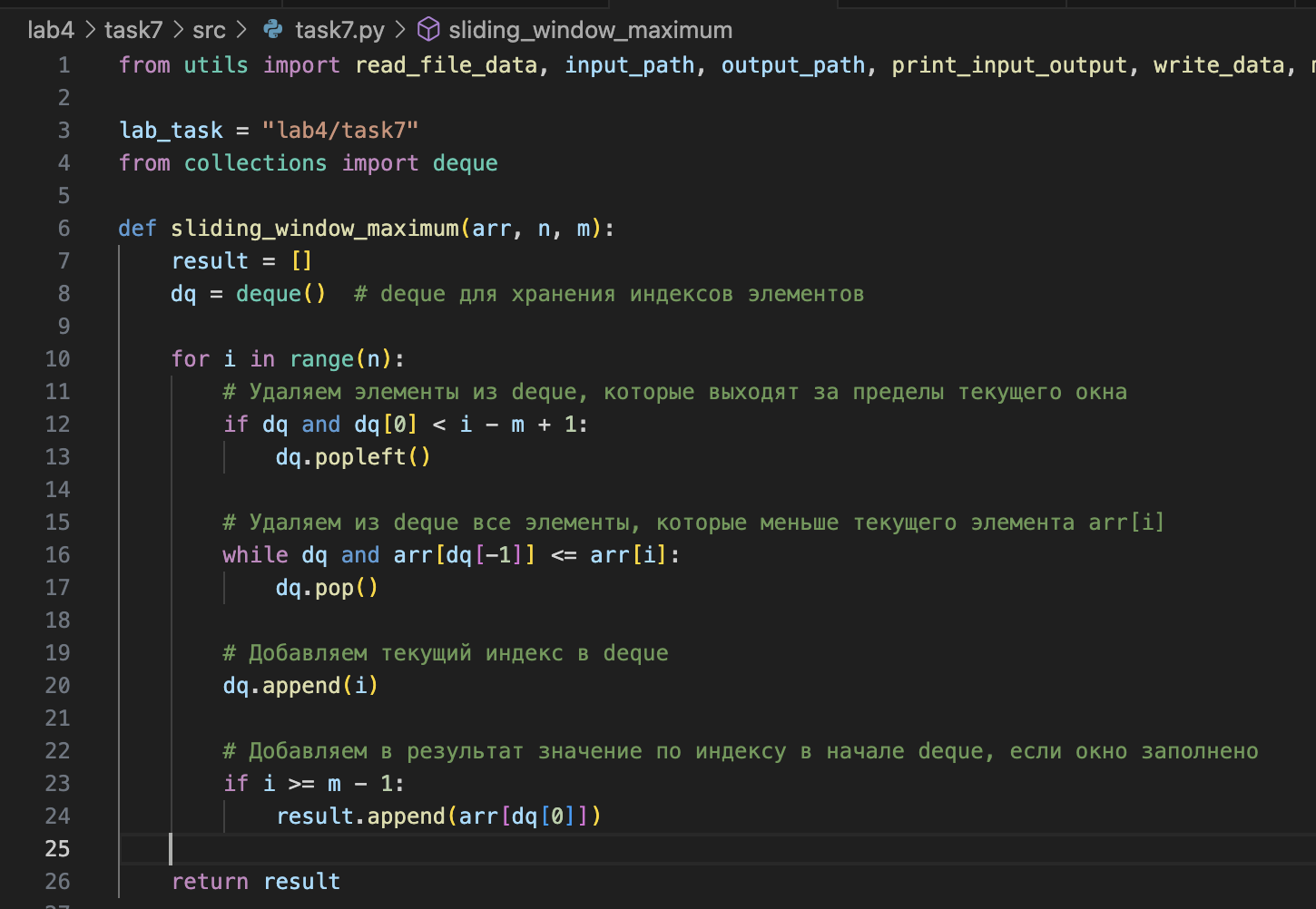
**Шаг 7: Считываю данные из input.txt и записываю их в output.txt**

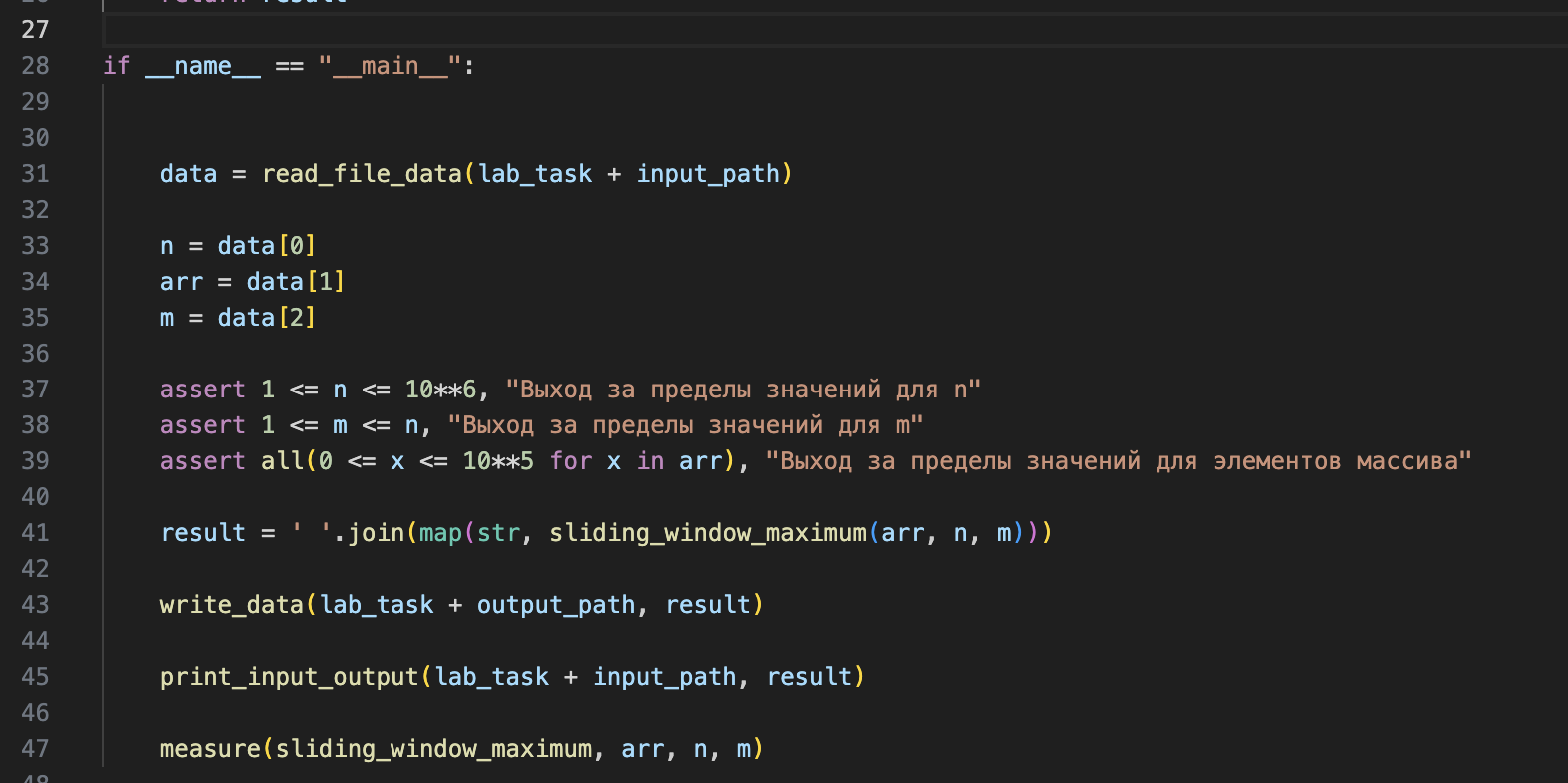
****

Задача №7



Решение:





Эта функция sliding\_window\_maximum принимает массив arr, размер окна m и размер массива n, и возвращает массив максимальных значений в каждом окне размера m.

**Шаг 1: Создание пустого списка результатов и deque**

**result = []**

**dq = deque() # deque для хранения индексов элементов**

В этом шаге я создаю пустой список result для хранения результатов и пустую deque dq для хранения индексов элементов.

**Шаг 2: Обход массива**

**for i in range(n):**

Начинаю обход массива arr с помощью цикла for.

**Шаг 3: Удаление элементов из deque, которые выходят за пределы текущего окна**

**if dq and dq[0] < i - m + 1:**

**dq.popleft()**

Здесь я проверяю, что deque dq не пуста и что первый элемент deque dq находится за пределами текущего окна. Если это так, то удалю первый элемент deque dq.

**Шаг 4: Удаление из deque всех элементов, которые меньше текущего элемента arr[i]**

**while dq and arr[dq[-1]] <= arr[i]:**

**dq.pop()**

Тут проверяю, что deque dq не пуста и что последний элемент deque dq меньше или равен текущему элементу arr[i]. Если это так, то удалю последний элемент deque dq.

**Шаг 5: Добавление текущего индекса в deque**

**dq.append(i)**

Добавляю текущий индекс i в deque dq.

**Шаг 6: Добавление в результат значение по индексу в начале deque, если окно заполнено**

**if i >= m - 1:**

**result.append(arr[dq[0]])**

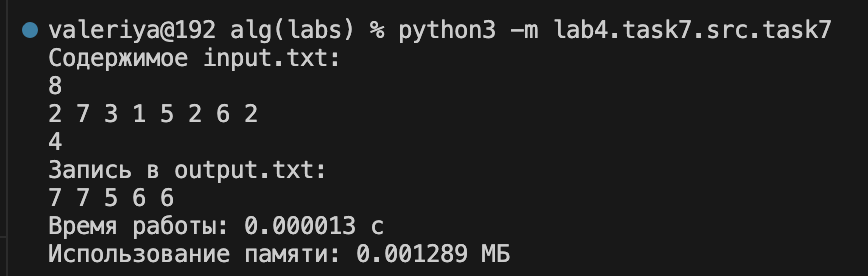
Проверяю, что текущий индекс i больше или равен размеру окна m минус 1. Если это так, то добавляю в результат значение по индексу в начале deque dq.

**Шаг 7: Возвращение результата**

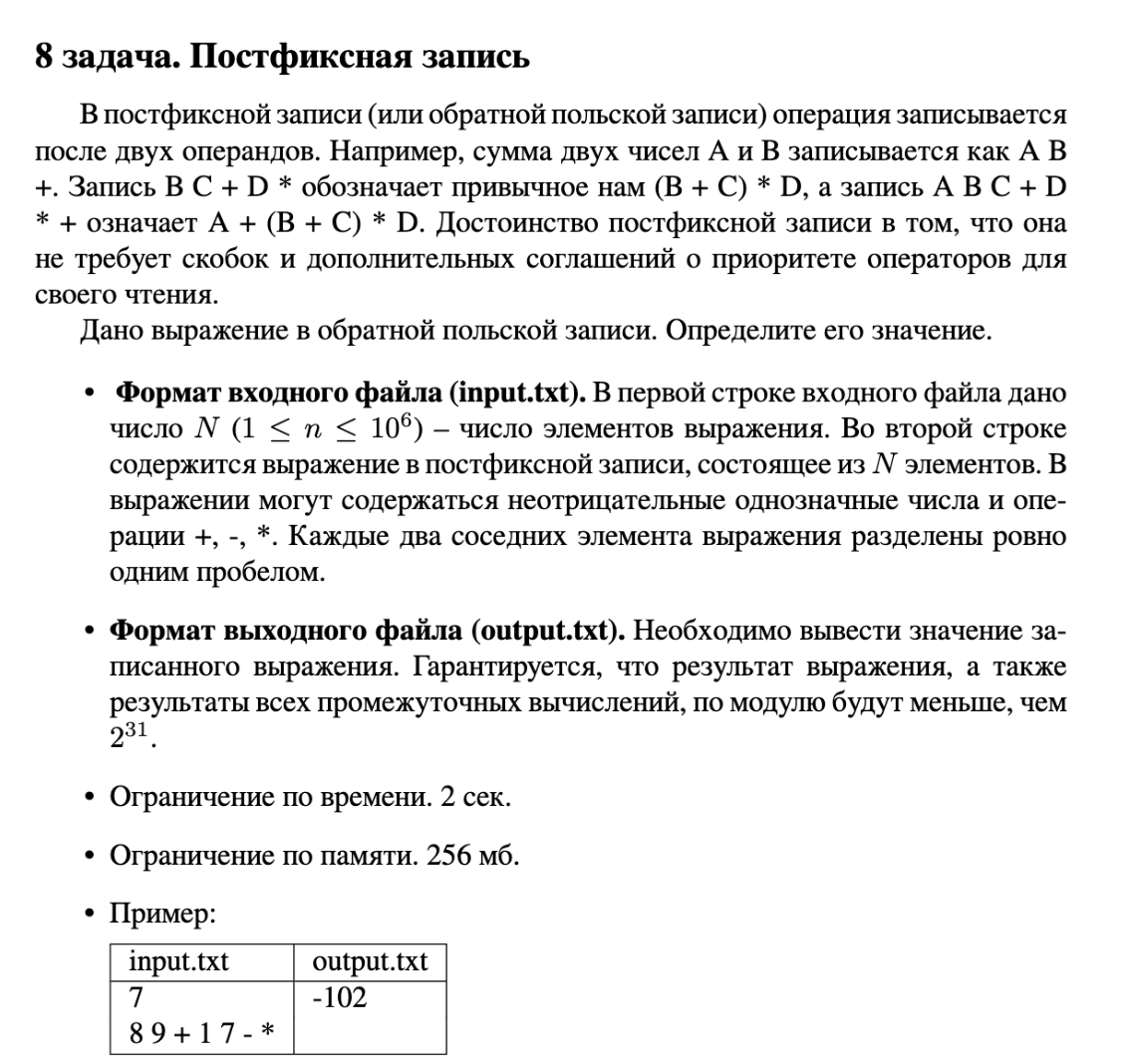
**return result**

Возвращаю список result, который содержит максимальные значения в каждом окне размера m.

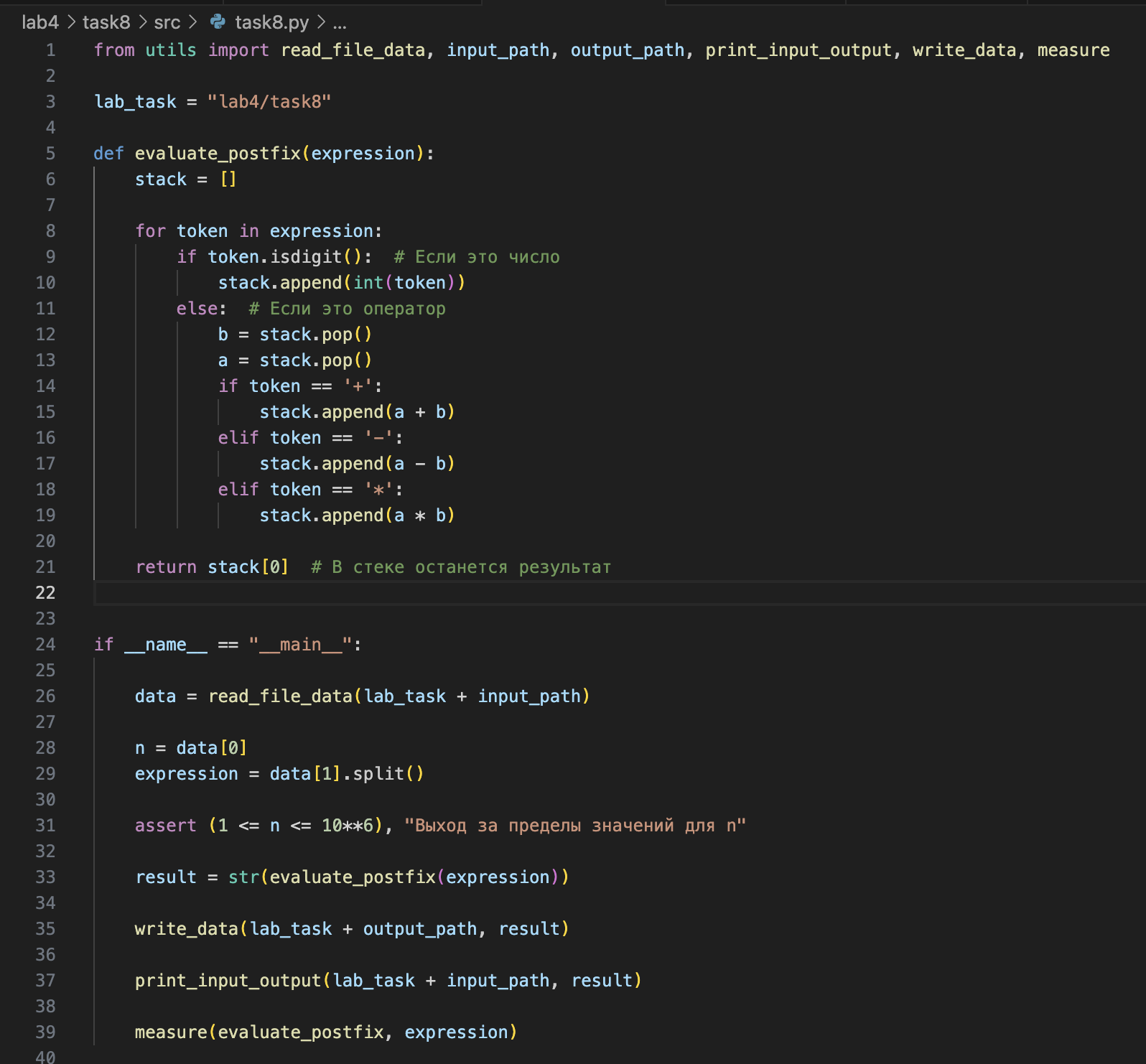
**Шаг 7: Считываю данные из input.txt и записываю их в output.txt**



Задача №8



Решение:



**Шаг 1: Создание пустого стека**

**stack = []**

В этом шаге я создаю пустой стек, который будет использоваться для хранения чисел и промежуточных результатов.

**Шаг 2: Обход выражения**

**for token in expression:**

Здесь начинаю обход выражения по token (числам и операторам).

**Шаг 3: Обработка чисел**

**if token.isdigit(): # Если это число**

**stack.append(int(token))**

Я проверяем, является ли текущий token числом. Если да, то мы преобразуем его в целое число и добавляем в стек.

**Шаг 4: Обработка операторов**

**else: # Если это оператор**

**b = stack.pop()**

**a = stack.pop()**

**if token == '+':**

**stack.append(a + b)**

**elif token == '-':**

**stack.append(a - b)**

**elif token == '\*':**

**stack.append(a \* b)**

Здесь проверяю, является ли текущий token оператором. Если да, то :

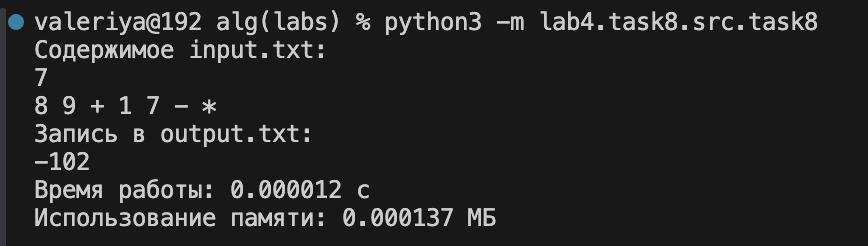
* Извлекаю два последних числа из стека (b и a)
* Выполняю операцию над этими числами в зависимости от оператора (+, -, \*)
* Добавляю результат обратно в стек

**Шаг 5: Возвращение результата**

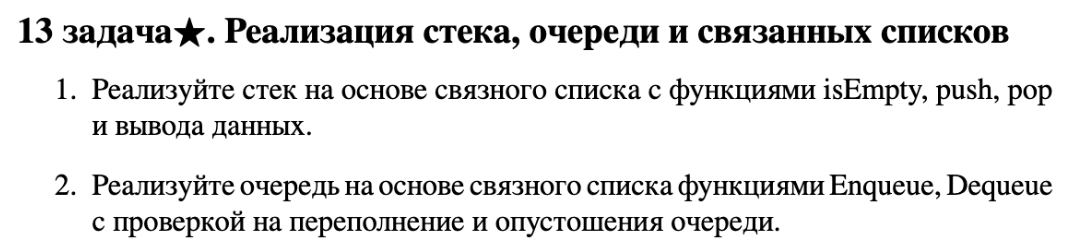
**return stack[0] # В стеке останется результат**

Тут возвращаю единственный элемент, оставшийся в стеке, который является результатом вычисления выражения.

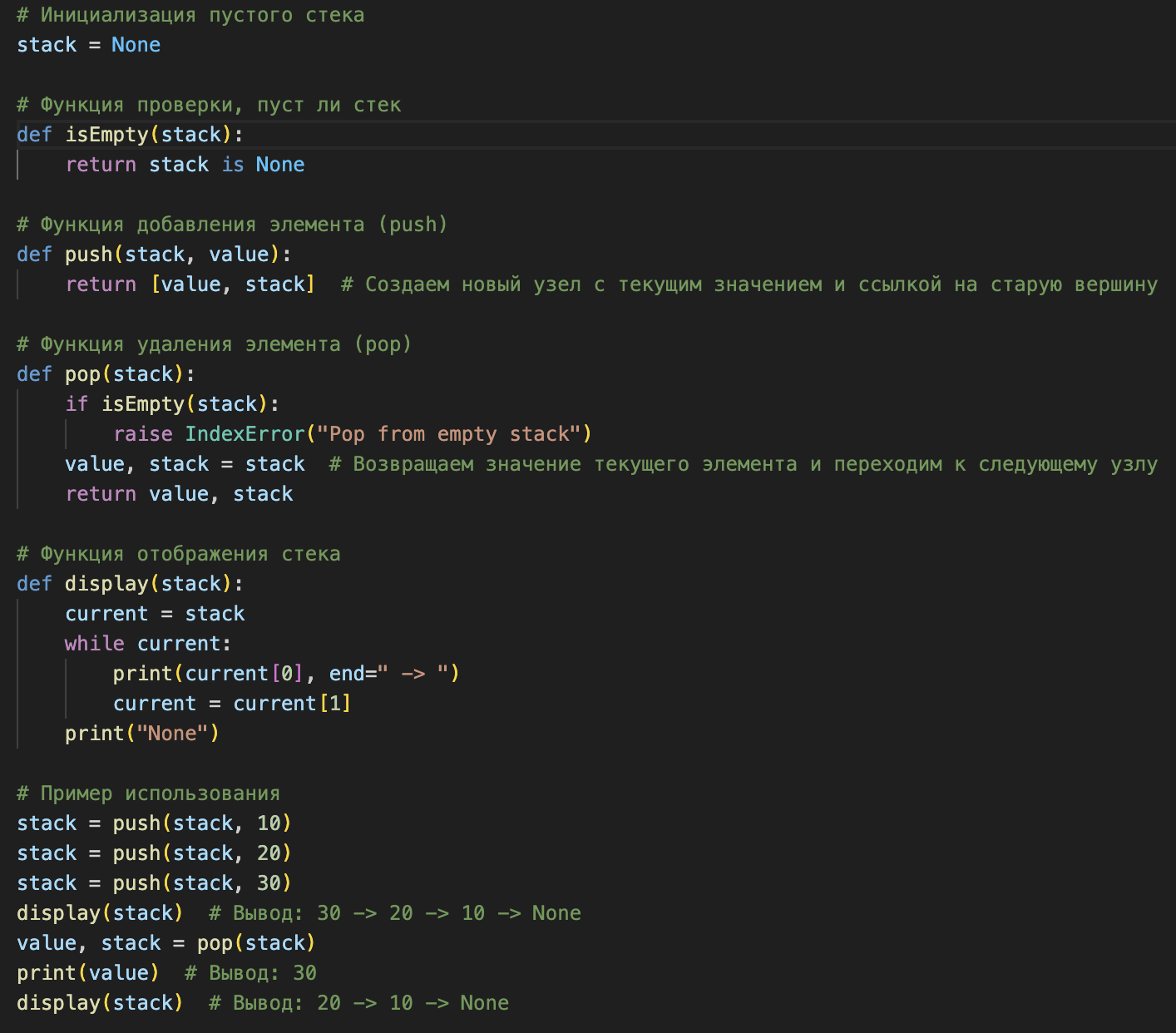
**Шаг 7: Считываю данные из input.txt и записываю их в output.txt**

****

Задача №13



Решение:

1.

Этот код реализует стек на основе связного списка.

**Функция isEmpty**

**def isEmpty(stack):**

**return stack is None**

Эта функция проверяет, пуст ли стек. Если стек равен None, то функция возвращает True, иначе False.

**Функция push**

**def push(stack, value):**

**return [value, stack]** # Создаем новый узел с текущим значением и ссылкой на старую вершину

Эта функция добавляет новый элемент в стек. Она создает новый узел с текущим значением и ссылкой на старую вершину стека. Если стек пуст, то новая вершина будет равна None.

**Функция pop**

**def pop(stack):**

**if isEmpty(stack):**

**raise IndexError("Pop from empty stack")**

**value, stack = stack** # Возвращаем значение текущего элемента и переходим к следующему узлу

**return value, stack**

Эта функция удаляет элемент из стека. Если стек пуст, то функция вызывает исключение IndexError. Иначе, функция возвращает значение текущего элемента и переходит к следующему узлу.

**Функция display**

**def display(stack):**

**current = stack**

**while current:**

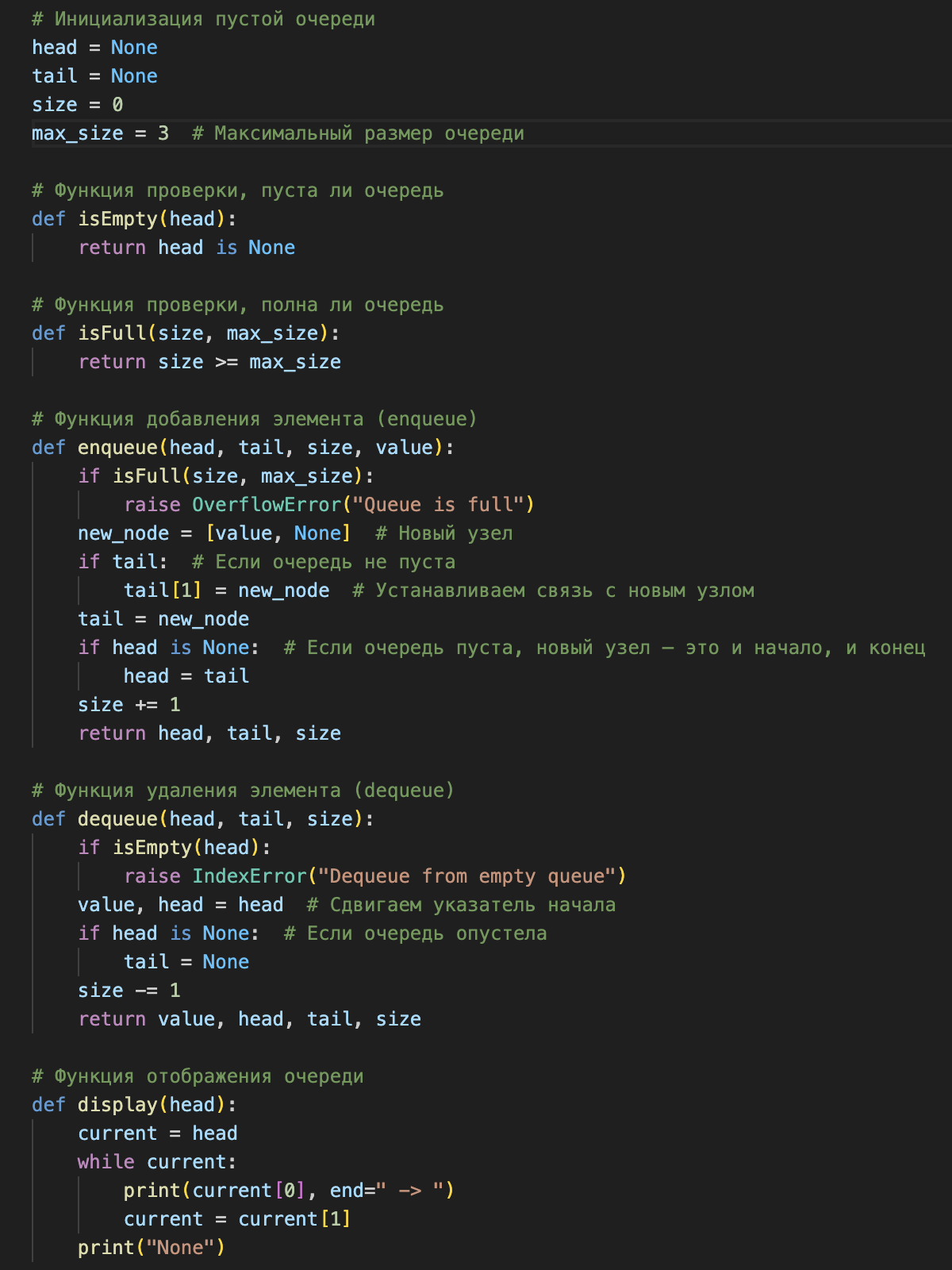
**print(current[0], end=" -> ")**

**current = current[1]**

**print("None")**

Эта функция отображает элементы стека. Она начинает с текущей вершины стека и переходит к следующему узлу, пока не достигнет конца стека. На каждом шаге она выводит значение текущего элемента и стрелку, указывающую на следующий узел.

2.



**Инициализация пустой очереди:**

**head = None**

**tail = None**

**size = 0**

**max\_size = 3** # Максимальный размер очереди

Эта часть кода инициализирует пустую очередь. Переменные head и tail обозначают начало и конец очереди соответственно, а size хранит текущий размер очереди. Переменная max\_size задает максимальный размер очереди.

**Функция проверки, пуста ли очередь:**

**def isEmpty(head):**

**return head is None**

Эта функция проверяет, пуста ли очередь. Если head равен None, то очередь пуста, и функция возвращает True. Иначе, функция возвращает False.

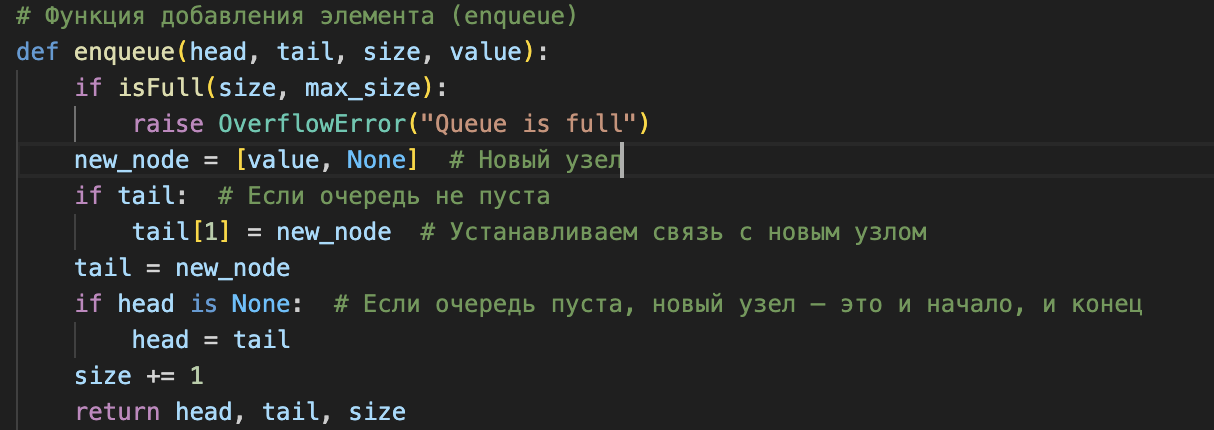
**Функция проверки, полна ли очередь**

**def isFull(size, max\_size):**

**return size >= max\_size**

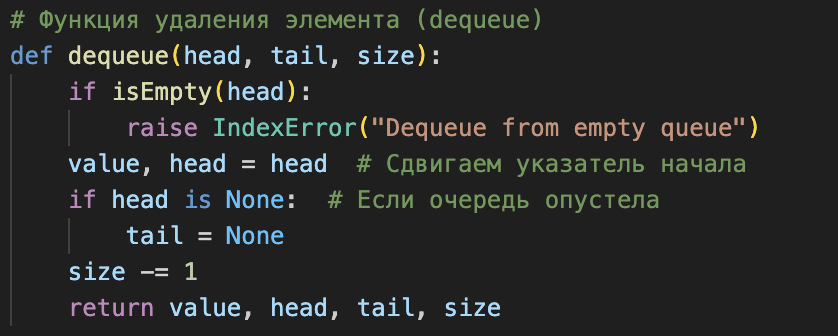
Эта функция проверяет, полна ли очередь. Если текущий размер очереди (size) больше или равен максимальному размеру (max\_size), то очередь полна, и функция возвращает True. Иначе, функция возвращает False.

**Функция добавления элемента (enqueue)**



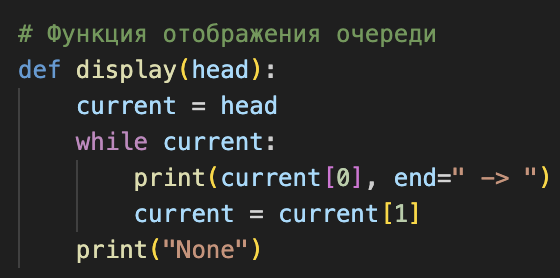
Эта функция добавляет новый элемент в конец очереди. Если очередь полна, функция вызывает исключение OverflowError. Иначе, функция создает новый узел с заданным значением и добавляет его в конец очереди. Если очередь пуста, новый узел становится началом и концом очереди.

**Функция удаления элемента (dequeue)**



Эта функция удаляет элемент из начала очереди. Если очередь пуста, функция вызывает исключение IndexError. Иначе, функция удаляет элемент из начала очереди и возвращает его значение. Если очередь опустела, функция обнуляет указатели начала и конца очереди.

**Функция отображения очереди**



Эта функция выводит элементы очереди на экран. Функция начинает с начала очереди и проходит по всем элементам, выводя их значения и указатели на следующий элемент. Если очередь пуста, функция выводит None.

Вывод:

В данной лабораторной работе я научилась работать со стеком и очередью, а также решать различные задачи с их применением. Решала задачу на очередь на основе связного списка.