МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования «Санкт-петербургский государственный морской технический университет»

ФАКУЛЬТЕТ ЦИФРОВЫХ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ

Кафедра Киберфизических систем

ЛАБОРАТОРНАЯ РАБОТА

По дисциплине «Программирование»

Выполнил: Носов Валерий

Проверил:

Санкт-Петербург

2024

Оглавление

[1. Цели и формулировка задачи 3](#_Toc187676633)

[2. Результаты работы 4](#_Toc187676634)

[2.1. Реализация программы с использованием функционального программирования языка Python 4](#_Toc187676635)

[2.1.1. Ход работы 4](#_Toc187676636)

[2.1.2. Демонстрация работы программы 9](#_Toc187676637)

[2.1.3. Листинг кода 11](#_Toc187676638)

[ЗАКЛЮЧЕНИЕ 13](#_Toc187676639)

[Список использованных источников 14](#_Toc187676640)

# Цели и формулировка задачи

Цели:

1. Научиться работать с файлами;
2. Научиться работать с несколькими функциями, а не только с main;
3. Реализовать решение через рекурсивную функцию.

Задача:

Даны **N** целых чисел **X1, X2, …, XN**. Требуется расставить между ними знаки **+** и **-** так, чтобы значение получившегося выражения было равно заданному целому **S**.

Входные данные: считываются с файла через пробел, в следующем порядке:

N X1 X2 ... XN S  
**2 <= N <= 30**, **0 <= Xi <= 5\*107**, **-109 <= S <= 109**

Необходимо вывести в файл полученное равенство с расставленными знаками, либо **no solution**, если не найдено решения. Если найдено несколько решений, то необходимо вывести **любое**. Решение, только через рекурсию!

Пример:

Input:

3 2 3 4 3

Output:

2-3+4=3

# Результаты работы

## **Реализация программы с использованием функционального программирования языка Python**

## Ход работы

Первым делом следует считать входные данные из файла. Так как название и формат файла не указаны в задаче, я буду использовать текстовый файл **input.txt** со следующим содержанием:

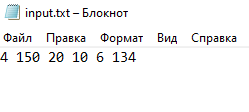


Рис. 1. Содержание **input.txt**.

Как и требовалось, первым числом определяется количество чисел, используемых в решении, а последним – ответ, который должен получиться. Все числа между ними – числа, над которыми будут проводиться арифметические операции.

Для считывания данных из файла я использую контекстный менеджер **with,** который автоматически открывает файл **input.txt** на чтение в кодировке **UTF-8**, после чего считывает входные данные и автоматически закрывает файл:



Рис. 2. Чтение данных из файла.

После этого все полученные данные я записываю в переменные:

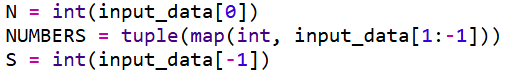


Рис. 3. Распределение входных данных.

Так как входные данные являются неизменными, то есть константами, их названия я пишу прописными буквами, а для чисел, над которыми будут производиться операции сложения и деления, я использую неизменяемый тип данных – кортеж (**tuple**). Также вместо того, чтобы перебирать эти числа циклом, я беру срез от исходного списка и с помощью функции **map** перевожу каждое значения в целочисленный тип данных (**int**), таким образом кортеж **NUMBERS** хранит в себе числа типа **int**, которые в дальнейшем будут использоваться для арифметических операций.

Основная идея решения заключается в том, чтобы создать все комбинации «+» и «-» длины **N-1** (так как один знак ставится между двумя числами, то количество знаков на один меньше количества этих чисел), а после перебирать эти комбинации, пока не получится нужный результат.

Для создания этих комбинаций я использую функцию **combinations():**

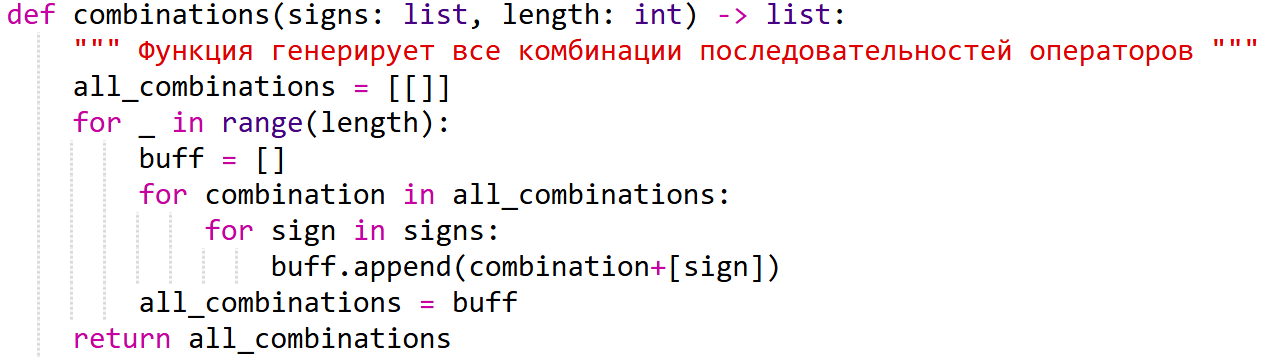


Рис. 4. Функция **combinations()**.

Данная функция принимает на вход в качестве первого аргумента список **signs**, содержащий в себе операторы, используемые в решении, в моём случае: «+-». Вторым же аргументом является длина, а точнее количество требуемых операторов, и как уже ранее говорилось, это **N-1**.

Функция N-1 раз перебирает знаки, каждый раз добавляя к комбинации новый и записывая промежуточный результат в **buff**, и в конце концов **all\_combinations** будет содержать в себе все возможные комбинации. После выполнения всех действий, функция возвращает список **all\_combinations**.

Используя эту функцию, я получаю список всех комбинаций:



Рис. 5. Получение всех комбинаций знаков.

Далее я вызываю функцию **solve()** – основную функцию всей программы, и записываю результат в переменную **result:**



Рис. 6. Запись ответа в **result**.

Перед тем, как разобрать функцию **solve()**, следует разобрать ещё две функции: **get\_result()** и **render\_result\_string()**.

Для начала разберём функцию **get\_result()**:

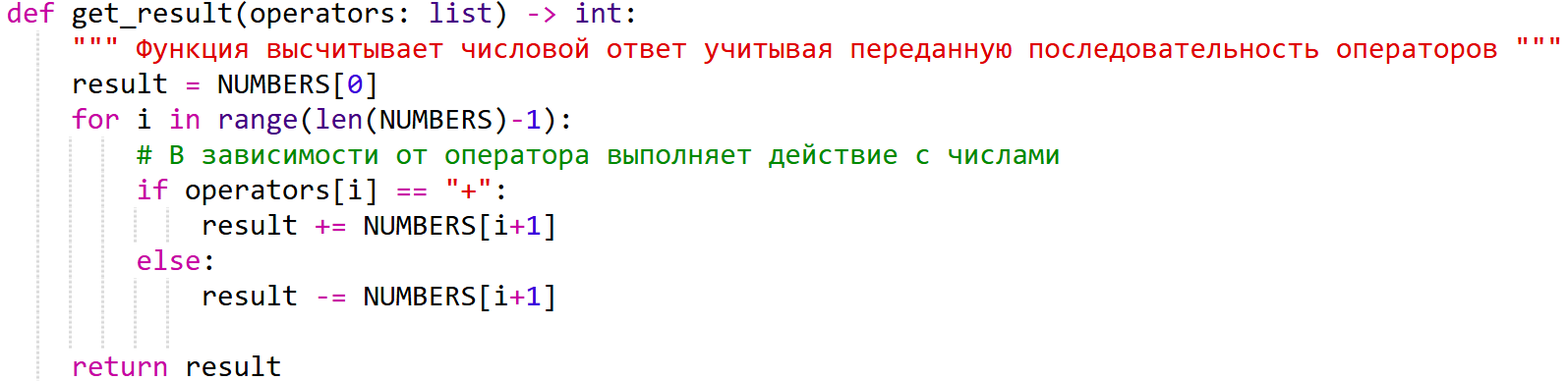


Рис. 6. Функция **get\_result()**.

Как и сказано в **docstring** (строка документации, обрамлённая тройными кавычками), данная функция получает на вход последовательность операторов (арифметических знаков) и, считывая их, выполняет действие с числами, возвращая результат.

Следующая функция – **render\_result\_string()**:

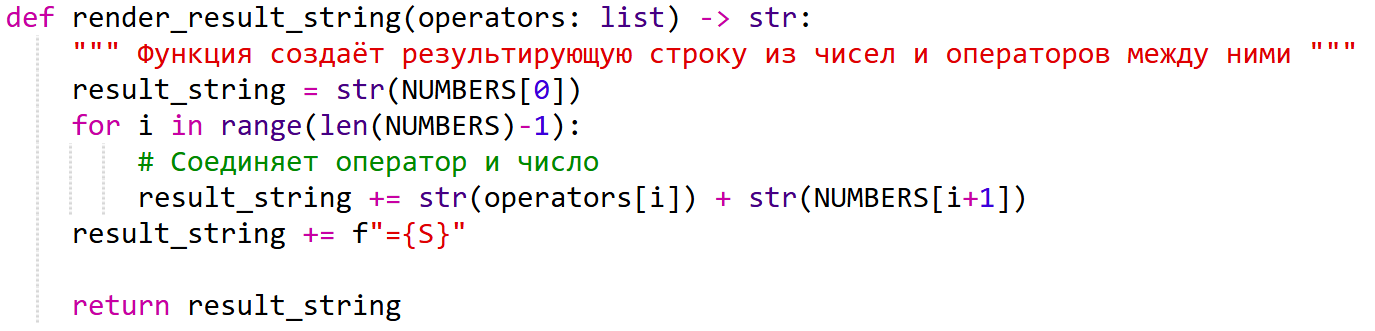


Рис 7. Функция **render\_result\_string()**.

Эта функция принимает на вход так же последовательность операторов, и создаёт строку вида «число{оператор}число=ответ», то есть в моём случае «150-20+10-6=134».

Теперь, когда я рассказал про вспомогательные функции, можно переходить к основной:

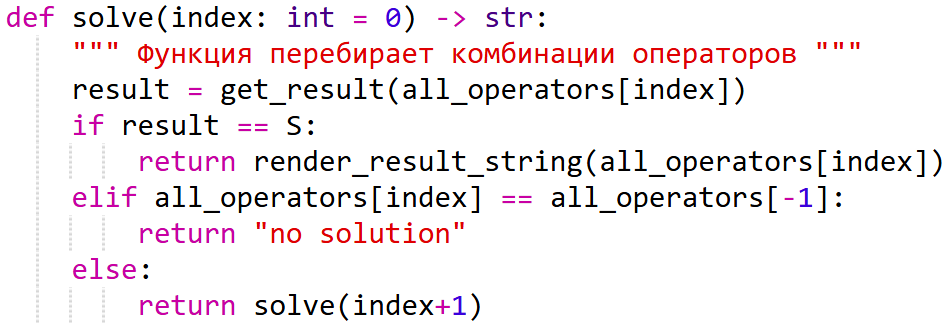


Рис. 8. Функция **solve()**.

Назначение функции совпадает с основной идеей решения задачи. Данная функция является рекурсивной, то есть она вызывает саму себя. У рекурсивной функции должны рассматриваться два случая: базовый и рекурсивный. Базовый случай обеспечивает выход из рекурсии и предотвращает её бесконечное выполнение. В свою очередь рекурсивный случай способствует тому, чтобы функция вызывала саму себя, но с изменёнными аргументами.

Для функции **solve()** существует два базовых случая: если результат функции равен требуемому результату (**result == S**), и если функция использовала уже все комбинации знаков, но так и не нашла ту, которая приводит к ответу (**all\_operators[index] == all\_operators[-1]**). В первом случае функция возвращает значение вспомогательной функции **render\_result\_string()**, которую я уже разобрал, а во втором возвращает **“no solution”**.

Рекурсивным случаем является тот, когда не выполнилось ни одно из предыдущих двух условий: в этом случае функция вызывает саму себя, увеличивая переменную **index** на единицу.

После того, как функция выполнилась и вернула значение, оно записывается в переменную **result**, и с помощью контекстного менеджера **with** ответ записывается в файл, используя ту же кодировку **UTF-8**:

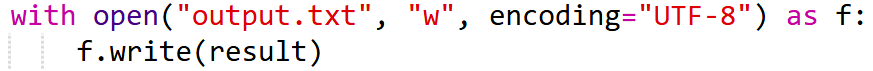


Рис. 9. Запись ответа в **output.txt**.

Здесь, так как имя файла для выходных данных так же не задано, я использую текстовый файл **output.txt**.

На этом программа завершается.

## Демонстрация работы программы

Итак, пришло время продемонстрировать работу программы. Как уже было сказано ранее, в качестве входных данных будет использоваться файл **input.txt**:

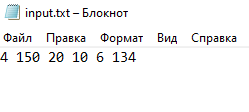


Рис. 10. Входные данные.

До запуска программы в папке присутствуют всего два файла: **main.py** и **input.txt**:

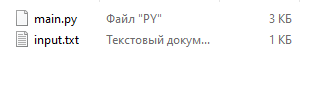


Рис. 11. Содержимое папки до выполнения программы.

Запускаем программу и проверяем содержимое папки:

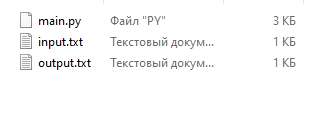


Рис. 12. Содержимое папки после выполнения программы.

Как видно, в папке появился новый файл **output.txt**, который содержит в себе следующие данные:

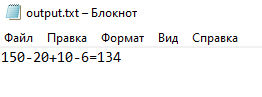


Рис. 13. Выходные данные в **output.txt**.

Это и является ответом на поставленную задачу.

## Листинг кода

def combinations(signs: list, length: int) -> list:

""" Функция генерирует все комбинации последовательностей операторов """

all\_combinations = [[]]

for \_ in range(length):

buff = []

for combination in all\_combinations:

for sign in signs:

buff.append(combination+[sign])

all\_combinations = buff

return all\_combinations

def solve(index: int = 0) -> str:

""" Функция перебирает комбинации операторов """

result = get\_result(all\_operators[index])

if result == S:

return render\_result\_string(all\_operators[index])

elif all\_operators[index] == all\_operators[-1]:

return "no solution"

else:

return solve(index+1)

def get\_result(operators: list) -> int:

""" Функция высчитывает числовой ответ учитывая переданную последовательность операторов """

result = NUMBERS[0]

for i in range(len(NUMBERS)-1):

# В зависимости от оператора выполняет действие с числами

if operators[i] == "+":

result += NUMBERS[i+1]

else:

result -= NUMBERS[i+1]

return result

def render\_result\_string(operators: list) -> str:

""" Функция создаёт результирующую строку из чисел и операторов между ними """

result\_string = str(NUMBERS[0])

for i in range(len(NUMBERS)-1):

# Соединяет оператор и число

result\_string += str(operators[i]) + str(NUMBERS[i+1])

result\_string += f"={S}"

return result\_string

if \_\_name\_\_ == "\_\_main\_\_":

with open('input.txt', 'r', encoding='UTF-8') as f:

input\_data = f.readline().strip().split()

N = int(input\_data[0])

NUMBERS = tuple(map(int, input\_data[1:-1]))

S = int(input\_data[-1])

all\_operators = combinations("+-", N-1)

result = solve()

with open("output.txt", "w", encoding="UTF-8") as f:

f.write(result)

# ЗАКЛЮЧЕНИЕ

# Для выполнения этой работы использовались следуюшие методы: 1) Работа с файлами. Входные данные читались из файла, после чего ответ так же записывался в файл. Такой подход позволяет не взаимодествовать с программой через терминал и не менять значение внутри самой программы, чтобы получить ответ. Вместо этого нужно просто задать входные данные в файл, после чего получить ответ в выходном файле, что очень удобно.

2) **Работа с несколькими функциями.** Разбиение кода на отдельные функции во много раз увеличивает читабельность и понимание кода. Кроме того, это позволяет выполнять одни и те же действия, просто вызвав функцию вместо того, чтобы прописывать код целиком, что так же способствует укорачиванию кода.

3) **Работа с рекурсивной функцией.** В некоторых случаях использование рекурсивной функции может быть удобней, нежели использование циклов. К тому же это сокращает количество строк кода.

Таким образом, используя перечисленные выше решения, мне удалось выполнить поставленную задачу.

# Список использованных источников