Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

«Лабораторная работа 2.9. Рекурсия в языке Python»

ОТЧЕТ
по лабораторной работе №12
дисциплины
«Основы программной инженерии»

Выполнил:
Луценко Дмитрий Андреевич
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2022 г.

Лабораторная работа 2.9. Рекурсия в языке Python

Цель работы: приобретение навыков по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

Индивидуальное задание

4. Создайте рекурсивную функцию, печатающую все возможные перестановки для целых чисел от 1 до N.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


def print_numbers(a):
    if len(a) == 1:
        return [a]

    temp = []

    for i in a:
        new_a = [x for x in a if x != i]
        z = print_numbers(new_a)

        for t in z:
            temp.append([i] + t)

    a = temp
    return a
```

Рисунок 1 – Функция с перестановкой чисел

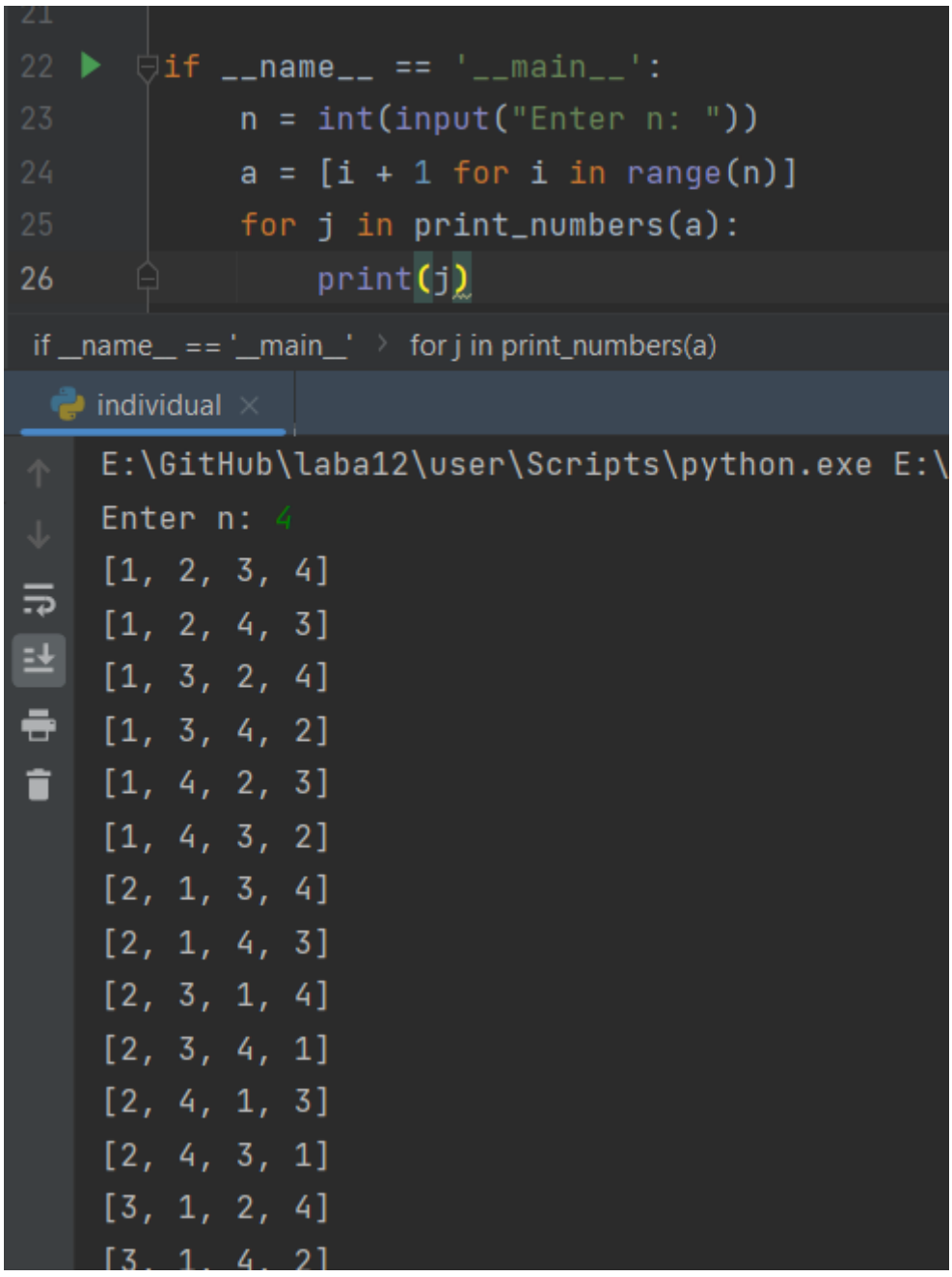

Рисунок 2 – Вызов функции print_numbers и результат работы программы

Вывод: были приобретены навыков по работе с рекурсивными функциями при написании

программ с помощью языка программирования Python версии 3.x.

Ответы на контрольные вопросы

1. Для чего нужна рекурсия? Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя.

2. Что называется базой рекурсии? У рекурсии, как и у математической индукции, есть база — аргументы, для которых значения функции определены

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций? Максимальная глубина рекурсии ограничена движком JavaScript. Точно можно рассчитывать на 10000 вложенных вызовов, некоторые интерпретаторы допускают и больше, но для большинства из них 100000 вызовов – за пределами возможностей. Существуют автоматические оптимизации, помогающие избежать переполнения стека вызовов («оптимизация хвостовой рекурсии»), но они ещё не поддерживаются везде и работают только для простых случаев.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python? Функция sys._getrecursionlimit() возвращает текущее значение предела рекурсии, максимальную глубину стека интерпретатора Python.

5. Что произойдет если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python? Произойдет RunTimeError.

6. Как изменить максимальную глубину рекурсии в языке Python? С помощью функции setrecursionlimit() модуля sys.

7. Каково назначение декоратора lru_cache? Декоратор @lru_cache() модуля functools оборачивает функцию с переданными в нее аргументами и запоминает возвращаемый результат соответствующий этим аргументам. Такое поведение может сэкономить время и ресурсы, когда дорогая или связанная с вводом/выводом функция периодически вызывается с одинаковыми аргументами.

8. Что такое хвостовая рекурсия? Как проводится оптимизация хвостовых вызовов? Хвостовая рекурсия — частный случай рекурсии, при

котором любой рекурсивный вызов является последней операцией перед возвратом из функции. Оптимизация хвостовой рекурсии выглядит так:

```
class TailRecurseException:
    def __init__(self, args, kwargs):
        self.args = args
        self.kwargs = kwargs


def tail_call_optimized(g):
"""
Эта программа показыает работу декоратора, который производит оптимизацию
хвостового вызова. Он делает это, вызывая исключение, если оно является его
прародителем, и перехватывает исключения, чтобы подделать оптимизацию хвоста.

Эта функция не работает, если функция декоратора не использует хвостовой вызов.
"""

    def func(*args, **kwargs):
        f = sys._getframe()
        if f.f_back and f.f_back.f_back and f.f_back.f_back.f_code == f.f_code:
            raise TailRecurseException(args, kwargs)
        else:
            while True:
                try:
                    return g(*args, **kwargs)
                except TailRecurseException, e:
                    args = e.args
                    kwargs = e.kwargs

    func.__doc__ = g.__doc__
    return func
```

```
@tail_call_optimized
def factorial(n, acc=1):
    "calculate a factorial"
    if n == 0:
        return acc

    return factorial(n-1, n*acc)


if __name__ == '__main__':
    print(factorial(10000))
# выводит большое число,
# но не доходит до лимита рекурсии
```