

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

**Отчет по лабораторной работе №2.9
Рекурсия в языке Python
по дисциплине «Технологии программирования»**

Выполнил студент группы ИВТ-б-о-20-1

Ищенко Т.С. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверила Воронкин Р.А. _____

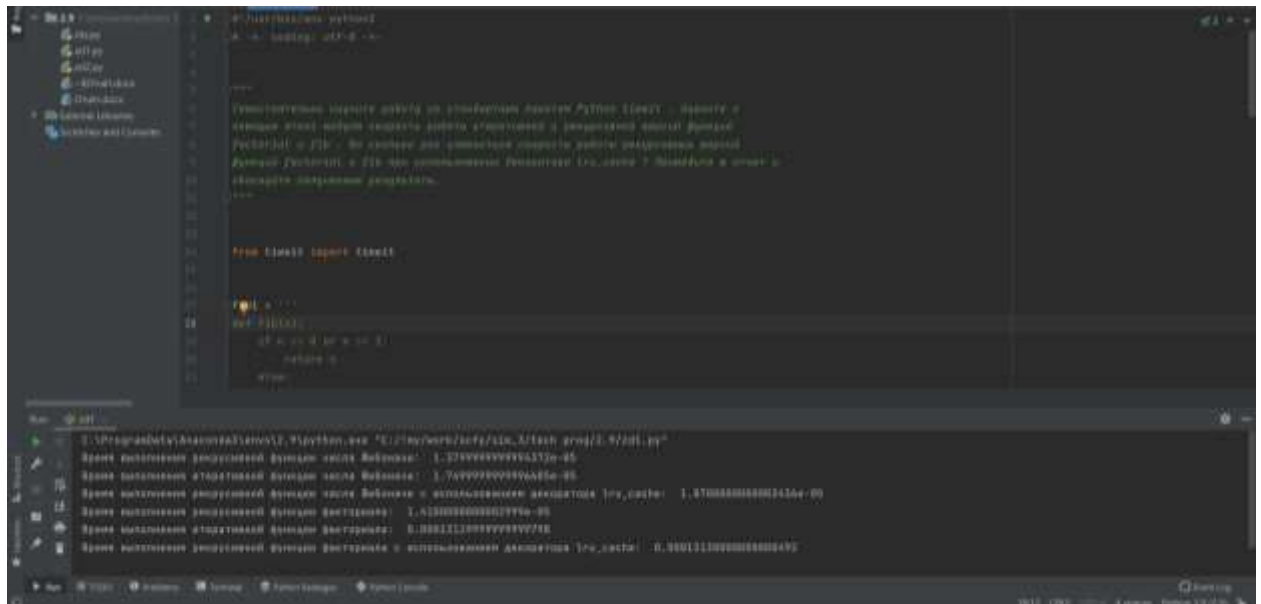
(подпись)

Ставрополь 2021

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

Ход работы:

1. В ходе выполнения первого задания я изучил стандартный пакет `timeit` и оценил с помощью этого модуля скорость работы интерактивной и рекурсивной версии функции `factorial` и `fib`. Выполнение рекурсивной функции поиска числа Фибоначчи оказалось быстрее, чем интерактивной и с использованием декоратора `lru_cache`. А вот при нахождении факториала быстрее всего оказалась рекурсивная функция факториала с использованием декоратора `lru_cache`.



The screenshot shows a Jupyter Notebook with the following code and output:

```
from timeit import timeit

# Прокомментируйте строку ниже, если вы используете пакет Python 3.x. Задайте n
# значением, равным скорости работы интерактивной и рекурсивной версии функций
# factorial и fib. Во сколько раз скорость работы интерактивной версии
# функции factorial и fib при использовании декоратора lru_cache? Задайте n, чтобы
# скорость декоратора была равна скорости рекурсивной функции.
n = 1000000

from timeit import timeit

def fib(n):
    if n <= 1:
        return n
    return fib(n-1) + fib(n-2)

def factorial(n):
    if n <= 1:
        return n
    return n * factorial(n-1)
```

Output:

```
Время выполнения рекурсивной функции числа Фибоначчи: 1.3744444444444444
Время выполнения интерактивной функции числа Фибоначчи: 1.7444444444444444
Время выполнения рекурсивной функции числа Фибоначчи с использованием декоратора lru_cache: 1.8700000000000001
Время выполнения рекурсивной функции факториала: 1.6100000000000001
Время выполнения интерактивной функции факториала: 0.00011100000000000001
Время выполнения рекурсивной функции факториала с использованием декоратора lru_cache: 0.00011100000000000001
```

Рисунок 1 – Результат выполнения первого задания

2. Самостоятельно проработал пример с оптимизацией хвостовых вызовов с помощью пакета `timeit` и оценил скорость работы функции `factorial` и `fib`. При выполнении функции `factorial()` с использованием интроспекции стека время оказалось быстрее, чем при обычном использовании функции. Но время выполнения функции `fib()` без использования стека оказалось быстрее, чем с использованием.

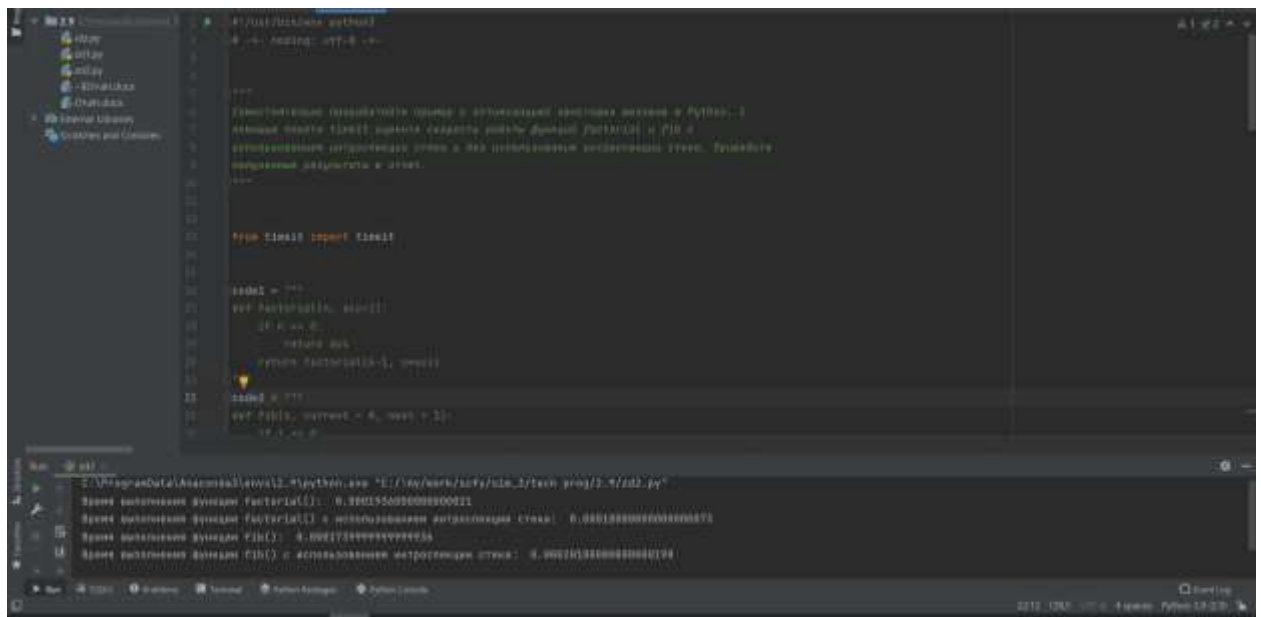


Рисунок 2 – Результат выполнения второго задания

3. Выполнил индивидуальное задание, согласно варианту

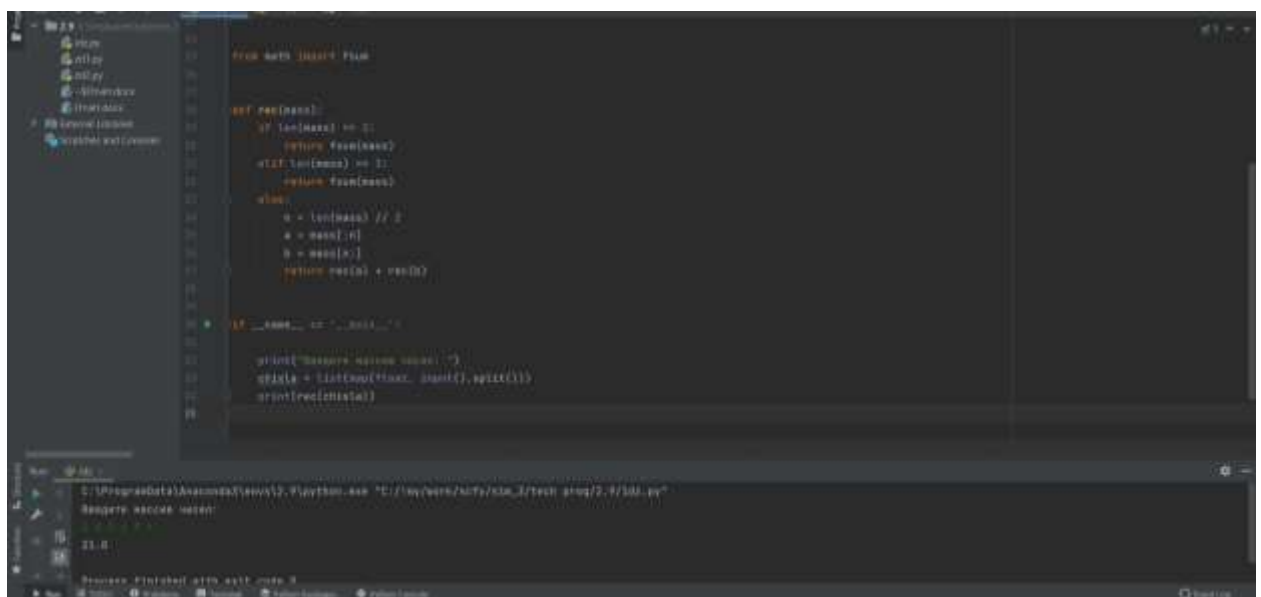


Рисунок 3 – Результат выполнения индивидуального задания.

4. Произвёл проверку на PEEP8

```
(tools) C:\!my\work\scfy\sim_3\tech prog\2.9>flake8
.\zd1.py:9:80: E501 line too long (84 > 79 characters)
.\zd2.py:8:80: E501 line too long (83 > 79 characters)

(tools) C:\!my\work\scfy\sim_3\tech prog\2.9>flake8
.\zd2.py:8:80: E501 line too long (83 > 79 characters)

(tools) C:\!my\work\scfy\sim_3\tech prog\2.9>flake8

(tools) C:\!my\work\scfy\sim_3\tech prog\2.9>
```

Рисунок 4 – Результат работы Flake8

Контрольные вопросы

1. Для чего нужна рекурсия? - Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя. Никакого парадокса здесь нет – компьютер лишь последовательно выполняет встретившиеся ему в программе команды и, если встречается вызов процедуры, просто начинает выполнять эту функцию. Без разницы, какая функция дала команду это делать.

2. Что называется базой рекурсии? – это такие аргументы функции, которые делают задачу настолько простой, что решение не требует дальнейших вложенных вызовов.

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций? - Стек в Python – это линейная структура данных, в которой данные расположены объектами друг над другом. Он хранит данные в режиме LIFO (Last in First Out). Данные хранятся в том же порядке, в каком на кухне тарелки располагаются одна над другой. Мы всегда выбираем последнюю тарелку из стопки тарелок. В стеке новый элемент вставляется с одного конца, и элемент может быть удален только с этого конца.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python? - Чтобы проверить текущие параметры лимита, нужно запустить: `sys.getrecursionlimit()`.

5. Что произойдет если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python? - Существует предел глубины возможной рекурсии, который зависит от реализации Python. Когда предел достигнут, возникает исключение `RuntimeError`.

6. Как изменить максимальную глубину рекурсии в языке Python? - Изменить максимальную глубину рекурсии можно с помощью `sys.setrecursionlimit()`.

7. Каково назначение декоратора `lru_cache` ? - Декоратор `lru_cache` является полезным инструментом, который можно использовать для уменьшения количества лишних вычислений. Декоратор оборачивает функцию с переданными в нее аргументами и запоминает возвращаемый результат, соответствующий этим аргументам.

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. Подобный вид рекурсии примечателен тем, что может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода функции. Оптимизация хвостовой рекурсии путём преобразования её в плоскую итерацию реализована во многих оптимизирующих компиляторах. В некоторых функциональных языках программирования спецификация гарантирует обязательную оптимизацию хвостовой рекурсии.

Вывод: в ходе выполнения лабораторной работы были получены навыки по работе с рекурсией при написании программы с помощью языка Python.