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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №2.9

Дисциплина: «Программирование на Python»
Тема: «Рекурсия в языке Python»

Выполнил:
Епифанов Алексей Александрович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем
», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

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

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.

2. Выполнил задание: самостоятельно изучите работу со стандартным пакетом Python timeit. Оцените с помощью этого модуля скорость работы итеративной и рекурсивной версий функций factorial и fib . Во сколько раз измениться скорость работы рекурсивных версий функций factorial и fib при использовании декоратора lru_cache? Приведите в отчет и обоснуйте полученные результаты.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
from functools import lru_cache
import timeit
def factorial_rec(n):
    """
    Функция, вычисляющая факториал рекурсивно
    """
    if n == 0:
        return 1
    else:
        return n * factorial_rec(n - 1)
def fib_rec(n):
    """
    Функция, вычисляющая число фибонначи рекурсивно
    """
    if n == 0 or n == 1:
        return n
    else:
        return fib_rec(n - 2) + fib_rec(n - 1)
@lru_cache
def factorial_rec_lru(n):
    """
    Функция, вычисляющая факториал рекурсивно
    Оптимитизирована с использованием lru_cache
    """
    if n == 0:
```

```
        return 1
    else:
        return n * factorial_rec_lru(n - 1)
@lru_cache
def fib_rec_lru(n):
    """
    Функция, вычисляющая число фибонначи рекурсивно
    Оптимитизирована с использованием lru_cache
    """
    if n == 0 or n == 1:
        return n
    else:
        return fib_rec_lru(n - 2) + fib_rec_lru(n - 1)
def factorial_iter(n):
    """
    Функция, вычисляющая факториал итеративно
    """
    product = 1
    while n > 1:
        product *= n
        n -= 1
    return product
def fib_iter(n):
    """
    Функция, вычисляющая число фибонначи итеративно
    """
    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
    return a
def create_graph(b, c, namegraph):
    """
    Создание графика из точек и настройка окна
    """
    plt.scatter(b, c, s=5)
    plt.title(namegraph)
    plt.xlabel("Число, переданное в функцию")
    plt.ylabel("Время работы функции")
def func_time(case_func, size):
    """
    Замер времени выполнения функций для разных случаев
    """
    time = []
    repeat = 50
```

```
    N = [i for i in range(30)]
    for n in N:
        timer = timeit.timeit(
            lambda: case_func[0](n),
            number=repeat
        ) / repeat
        time.append(timer)
    plt.figure(case_func[1], size)
    plt.subplots_adjust(left=0.25)
    # Создание графиков
    create_graph(N, time, case_func[1])
if __name__ == '__main__':
    # Настройка размера окон
    dpi = 100
    width_inches = (1680 / dpi) / 4
    height_inches = (850 / dpi) / 2
    size = (width_inches, height_inches)
    functions = {
        factorial_rec: "Факториал рекурсивный",
        factorial_rec_lru: "Факториал рекурсивный с @lru_cache",
        factorial_iter: "Факториал итеративный",
        fib_rec: "Fibonacci рекурсивный",
        fib_rec_lru: "Fibonacci рекурсивный с @lru_cache",
        fib_iter: "Fibonacci итеративный"
    }
    for func_case in functions.items():
        func_time(func_case, size)
    # Показ графиков†
    plt.show()
```

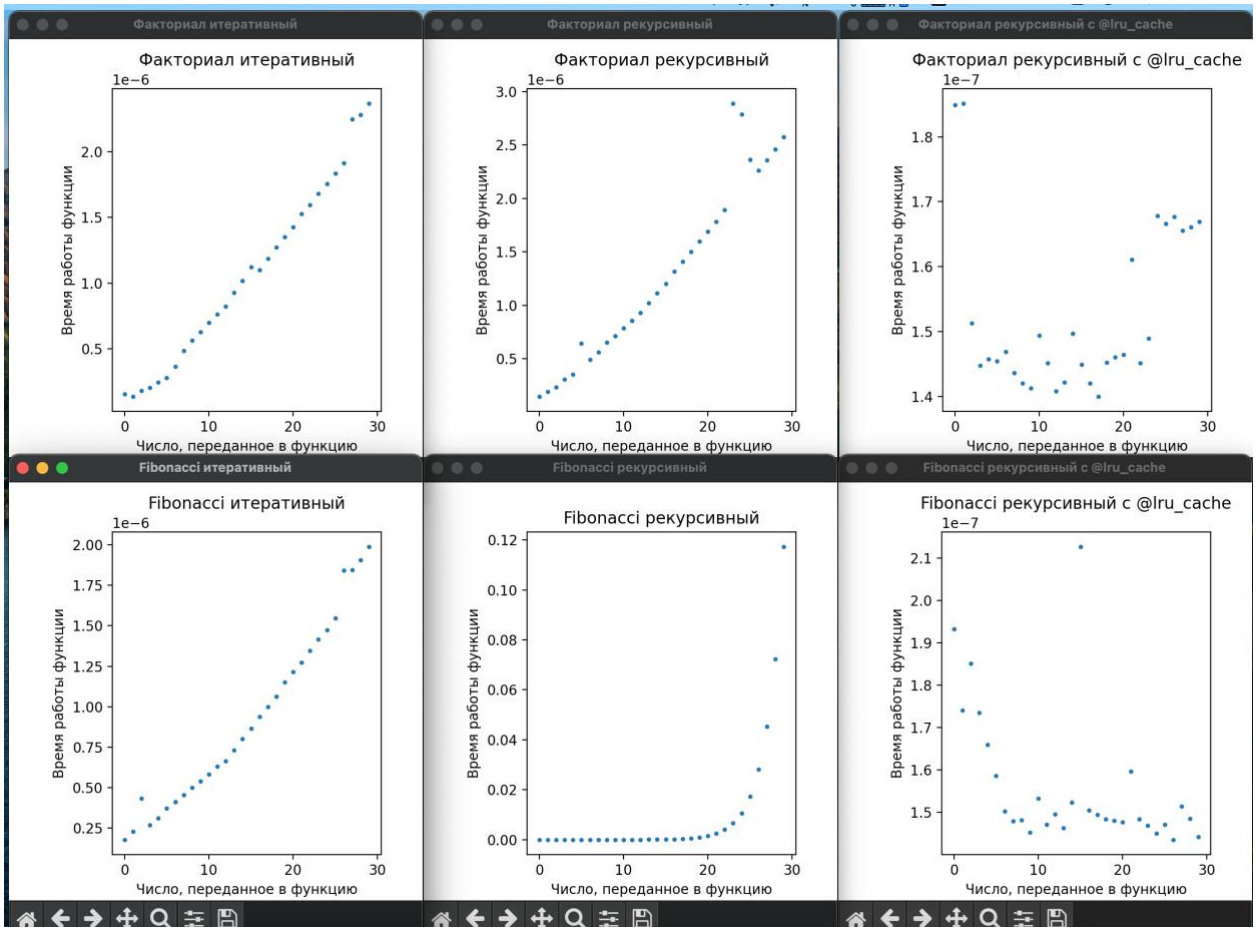

Рисунок 1. Вывод программы file1

На данных графиках четко видно, что при использовании декоратора @lru_cache функции ускоряются в несколько десятков раз, особенно функция поиска числа Фибоначчи, это происходит потому, что функция поиска числа Фибоначчи выполняет много лишних одинаковых подсчетов, это как раз и оптимизирует декоратор @lru_cache

3. Выполнил индивидуальное задание вариант 10: Опишите рекурсивную функцию, которая по заданным вещественному x и целому вычисляет величину согласно формуле:

$$x^n = \begin{cases} 1, & n = 0, \\ 1/x^{|n|}, & n < 0, \\ x \cdot (x^{n-1}), & n > 0. \end{cases}$$

Рисунок 2. Формула из индивидуального задания

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

```
def recursive_function(x, n):
    """
    Рекурсивная функция вычисляющая x^n
    """
    if n == 0:
        return 1
    elif n < 0:
        return 1 / recursive_function(x, abs(n))
    else:
        return x * recursive_function(x, n-1)
if __name__ == '__main__':
    x, n = 2, 16
    x_n = recursive_function(x, n)
    print("{}^{} = {}".format(x, n, x_n))
```

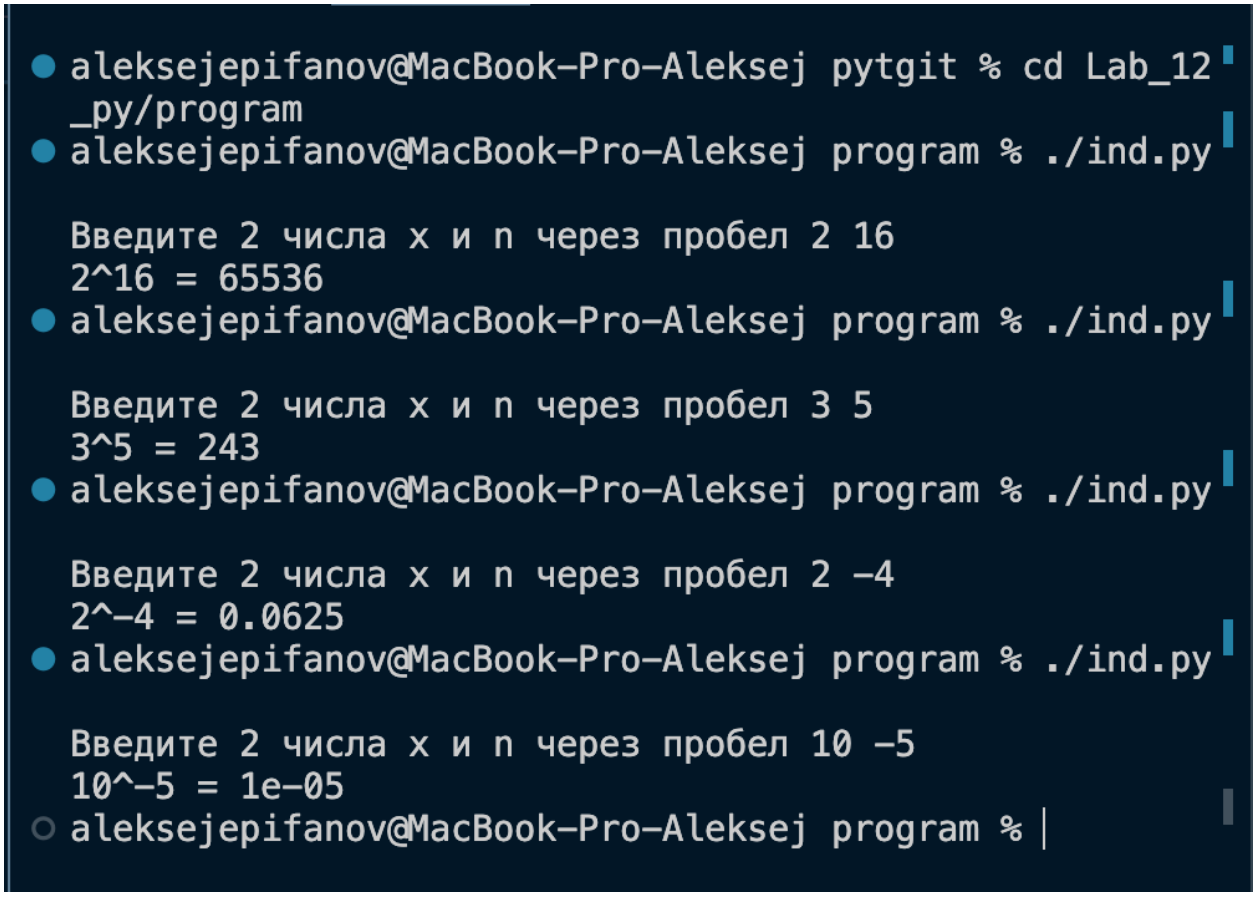

Рисунок 3. Запуск программы индивидуального задания

Ответы на контрольные вопросы:

1. Для чего нужна рекурсия?

У рекурсии есть несколько преимуществ в сравнении с первыми двумя методами. Рекурсия занимает меньше времени, чем выписывание 1 + 2 + 3 на сумму от 1 до 3, рекурсия может работать в обратную сторону.

Принимая во внимание, что цикл for работает строго вперед, иногда рекурсивное решение проще, чем итеративное решение. Это очевидно при реализации обращения связанного списка.

2. Что называется базой рекурсии?

Базовый случай — это возврат значения, не обращаясь к самой функции. Базовый случай необходим, без него была бы бесконечная рекурсия.

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций?

Стек программы (или вызовов) — это структура данных, используемая компьютерной программой для управления вызовами функций во время их выполнения. При вызове функции текущее состояние программы, включая локальные переменные и адрес возврата, помещается в стек. Стек программы обеспечивает управление выполнением функций в порядке их вызова и позволяет программе возвращаться к предыдущим состояниям после завершения работы

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 :

RuntimeError: Maximum Recursion Depth Exceeded

В Python 3.5 ошибка стала называться RecursionError, которая является производной от RuntimeError.

6. Как изменить максимальную глубину рекурсии в языке Python? Можно изменить предел глубины рекурсии с помощью вызова:

sys.setrecursionlimit(limit).

7. Каково назначение декоратора lru_cache ?

Декоратор lru_cache используется для кэширования результатов вызовов функций. "LRU" расшифровывается как «Least Recently Used». Когда функция вызывается с определенными аргументами, результат вызова сохраняется в кэше. Если функция вызывается с теми же аргументами впоследствии, результат вызова извлекается из кэша вместо того, чтобы вычисляться заново, что может ускорить выполнение программы.

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 результате выполнения работы были приобретены навыки по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.