

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

Институт цифрового развития  
Кафедра инфокоммуникаций

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.9**  
**дисциплины «Программирование на Python»**

Выполнил:  
Магдаев Даламбек Магомедович  
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 Р.А., доцент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

## **Тема:** Рекурсия в языке Python

**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.

### **Код программы:**

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import timeit
from functools import lru_cache
# Итеративная версия функции factorial
def factorial_iterative(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
# Рекурсивная версия функции factorial
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n-1)
# Итеративная версия функции fibonacci
def fib_iterative(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a
```

```

# Рекурсивная версия функции fibonacci
def fib_recursive(n):
    if n <= 1:
        return n
    else:
        return fib_recursive(n-1) + fib_recursive(n-2)
# Рекурсивная версия функции factorial с использованием lru_cache
@lru_cache
def factorial_recursive_lru(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive_lru(n-1)
# Рекурсивная версия функции fibonacci с использованием lru_cache
@lru_cache
def fib_recursive_lru(n):
    if n <= 1:
        return n
    else:
        return fib_recursive_lru(n-1) + fib_recursive_lru(n-2)
if __name__ == '__main__':
    # Оценка скорости работы итеративной и рекурсивной версий функций
    print("Factorial итеративный:", timeit.timeit(lambda: factorial_iterative(10),
        number=100000))
    print("Factorial рекурсивный:", timeit.timeit(lambda: factorial_recursive(10),
        number=100000))
    print("Fibonacci итеративный:", timeit.timeit(lambda: fib_iterative(10),
        number=100000))
    print("Fibonacci рекурсивный:", timeit.timeit(lambda: fib_recursive(10),
        number=100000))
    # Оценка скорости работы рекурсивных версий функций с использованием
    lru_cache
    print("Factorial рекурсивный с @lru_cache:", timeit.timeit(lambda:
        factorial_recursive_lru(10), number=100000))
    print("Fibonacci рекурсивный с @lru_cache:", timeit.timeit(lambda:
        fib_recursive_lru(10), number=100000))

```

```

PS C:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Python> python
python\9\task1.py
Factorial итеративный: 0.09519190003629774
Factorial рекурсивный: 0.1442439000820741
Fibonacci итеративный: 0.091640800004825
Fibonacci рекурсивный: 2.1525053000077605
Factorial рекурсивный с @lru_cache: 0.021890099975280464
Fibonacci рекурсивный с @lru_cache: 0.02533799991942942
PS C:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Python>

```

Рисунок 1. Вывод программы task1

### 3. Выполнил индивидуальное задание вариант 9:

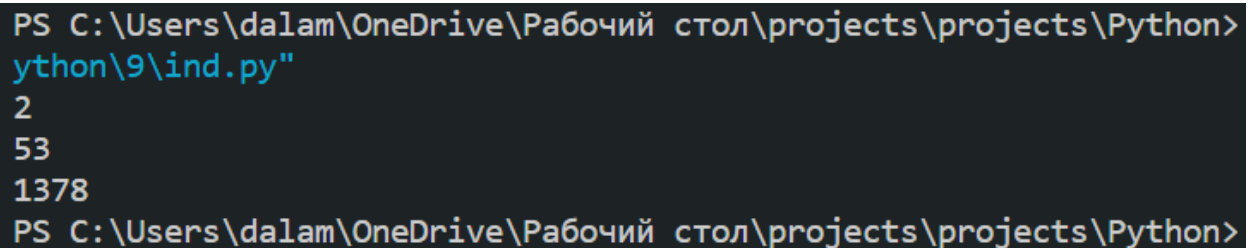
9. Даны целые числа  $m$  и  $n$ , где  $0 \leq m \leq n$ , вычислить, используя рекурсию, число сочетаний  $C_n^m$  по формуле:  $C_n^0 = C_n^n = 1$ ,  $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$  при  $0 \leq m \leq n$ .  
Воспользовавшись формулой

$$C_n^m = \frac{n!}{m!(n-m)!} \quad (1)$$

Рисунок 2. Условие задания

#### Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
def C(m, n):
    if m == n or m == 0:
        return 1
    elif 0 <= m <= n:
        return C(m, n - 1) + C(m - 1, n - 1)
if __name__ == '__main__':
    print(C(int(input()), int(input())))
```



```
PS C:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Python>
python\9\ind.py
2
53
1378
PS C:\Users\dalam\OneDrive\Рабочий стол\projects\projects\Python>
```

Рисунок 3. Результат работы программы

#### 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.