

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 №12**  
**дисциплины «Основы программной инженерии»**

Выполнила:  
Ламская Ксения Вячеславовна  
2 курс, группа ПИЖ-б-о-22-1,  
09.03.04 «Программная инженерия»,  
направленность (профиль) «Разработка и  
сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Доцент кафедры инфокоммуникаций  
Воронкин Роман Александрович

---

(подпись)

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

Ставрополь, 2023 г.

**Тема:** 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.


### Порядок выполнения работы

#### 1. Создание репозитория GitHub.

#### Create a new repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner *	Repository name *
 ksenia-lamskaya ▾	/ 12laba
	✓ 12laba is available.

Great repository names are short and memorable. Need inspiration? How about **ideal-octo-chainsaw** ?

Description (optional)

- ☒  **Public**  
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**  
You choose who can see and commit to this repository.

Initialize this repository with:

- ☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: Python ▾


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

2. Самостоятельно изучите работу со стандартным пакетом Python `timeit`. Оцените спомощью 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`?

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

from timeit import timeit
from functools import lru_cache

import sys

@lru_cache
def factorial_recursion(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n * factorial_recursion(n - 1)

@lru_cache
def fib_recursion(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib_recursion(n - 2) + fib_recursion(n - 1)

def factorial_iterable(n):
    product = 1
    while n > 1:
        product *= n
        n -= 1
    return product

def fib_iterable(n):
    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
    return a

if __name__ == "__main__":
    sys.setrecursionlimit(5000)
    n = 35
    setup1 = """from __main__ import fib_recursion"""
    setup2 = """from __main__ import factorial_recursion"""
    timer = timeit(stmt=f'fib_recursion({n})', number=10, setup=setup1)
    print('Время выполнения рекурсивной функции с @lru_cache: ', {timer})
    timer = timeit(stmt=f'factorial_recursion({n})', number=10, setup=setup2)
    print('Время выполнения рекурсивной функции с @lru_cache: ', {timer})
```

Рисунок 2.1 – Код программы

```

PS C:\Ksen\12laba\tasks> & c:/Users/irbis/AppData/Local/Programs/Python/Python311/python
Время выполнения рекурсивной функции с @lru_cache: {0.00020379992201924324}
Время выполнения рекурсивной функции с @lru_cache: {0.00024870014749467373}
PS C:\Ksen\12laba\tasks> & c:/Users/irbis/AppData/Local/Programs/Python/Python311/python
Время выполнения рекурсивной функции с @lru_cache: {0.00016530021093785763}
Время выполнения рекурсивной функции с @lru_cache: {0.0001938000787049532}
PS C:\Ksen\12laba\tasks> & c:/Users/irbis/AppData/Local/Programs/Python/Python311/python
Время выполнения рекурсивной функции с @lru_cache: {0.0003396999090909958}
Время выполнения рекурсивной функции с @lru_cache: {0.00022600009106099606}
PS C:\Ksen\12laba\tasks> & c:/Users/irbis/AppData/Local/Programs/Python/Python311/python
Время выполнения рекурсивной функции с @lru_cache: {0.00015050009824335575}
Время выполнения рекурсивной функции с @lru_cache: {0.00018190010450780392}
PS C:\Ksen\12laba\tasks>

```

Рисунок 2.2 – Вывод программы

3. Самостоятельно проработайте пример с оптимизацией хвостовых вызовов в Python. С помощью пакета timeit оцените скорость работы функций factorial и fib с использованием интроспекции стека и без использования интроспекции стека.

```

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

import sys
from timeit import timeit

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

def tail_call_optimized(g):
    """
    Эта программа показывает 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 as 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)

@tail_call_optimized
def fib(i, current=0, nxt=1):
    if i == 0:
        return current
    else:
        return fib(i - 1, nxt, current + nxt)

if __name__ == '__main__':
    n = 30
    setup1 = """from __main__ import factorial"""
    setup2 = """from __main__ import fib"""
    timer = timeit(stmt=f'factorial({n})', number=10, setup=setup1)
    print(f"Время выполнения функции factorial(): {timer}")
    timer = timeit(stmt=f'fib({n})', number=10, setup=setup2)
    print(f"Время выполнения функции fib(): {timer}")

```

Рисунок 3.1 – Код программы

```

Время выполнения рекурсивной функции с @lru_cache: {0.00018190010450780392}
PS C:\Ksen\12laba\tasks> & c:/Users/irbis/AppData/Local/Programs/Python/Python3
Время выполнения функции factorial(): 0.004120799945667386
Время выполнения функции fib(): 0.0031516999006271362
PS C:\Ksen\12laba\tasks>

```

Рисунок 3.2 – Вывод программы

4.

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

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

можно проверить правильность результата.

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

def calculate_combination(m, n):
    if m == 0 or m == n: #базовый случай
        return 1

    if m < 0 or m > n:
        return 0

    return calculate_combination(m, n-1) + calculate_combination(m-1, n-1)

if __name__ == '__main__':
    m = int(input('Введите m: '))
    n = int(input('Введите n: '))
    res = calculate_combination(m, n)
    print(f'C({m}, {n}) = {res}')
```

Рисунок 4.1 – Код программы

```
Введите m: 7
Введите n: 8
C(7, 8) = 8
PS C:\Ksen\12laba\tasks> & c:/Users/irbis/AppData/Local/Programs/Python/Python311/python.exe c:/Ksen/12laba/t
1.py
Введите m: 3
Введите n: 4
C(3, 4) = 4
PS C:\Ksen\12laba\tasks> & c:/Users/irbis/AppData/Local/Programs/Python/Python311/python.exe c:/Ksen/12laba/t
1.py
Введите m: 3
Введите n: 7
C(3, 7) = 35
```

Рисунок 4.2 – Вывод программы

## Ответы на контрольные вопросы

1. Рекурсия - это процесс, при котором функция вызывает саму себя. Она используется для решения задач, которые могут быть разбиты на более мелкие подзадачи. Рекурсия позволяет писать более компактный и легко читаемый код.
2. База рекурсии - это условие, при котором рекурсивный процесс завершается. Это условие должно быть определено внутри рекурсивной функции, чтобы избежать бесконечной рекурсии.
3. Стек программы - это область памяти, которая используется для хранения информации о вызовах функций. При вызове функции информация о вызове помещается в стек, а при возврате из функции информация удаляется из стека. Это позволяет программе сохранять контекст выполнения функций и возвращаться к ним позже.
4. Максимальную глубину рекурсии можно получить с помощью функции `sys.getrecursionlimit()`.
5. Если число рекурсивных вызовов превысит максимальную глубину рекурсии в Python, будет вызвано исключение `RecursionError`.
6. Максимальную глубину рекурсии можно изменить с помощью функции `sys.setrecursionlimit()`.
7. Декоратор `lru_cache` используется для кэширования результатов выполнения функции. Он сохраняет результаты выполнения функции в памяти и возвращает их при повторном вызове функции с теми же аргументами. Это может значительно ускорить выполнение функции в случае, если она вызывается многократно с одними и теми же аргументами.
8. Хвостовая рекурсия - это рекурсия, при которой вызов рекурсивной функции является последней операцией в функции. Оптимизация хвостовых вызовов заключается в замене рекурсивной функции на итеративную функцию, что может улучшить производительность и избежать переполнения стека вызовов.

