

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 «Программирование на Python»**  
**Вариант 29**

Выполнил:  
Саенко Андрей Максимович  
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., канд. технических  
наук, 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. Выполнено задание из лабораторной работы

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 изменится скорость работы рекурсивных версий функций `factorial` и `fib` при использовании декоратора `lru_cache`? Приведите в отчет и обоснуйте полученные результаты.

Результат работы программы:

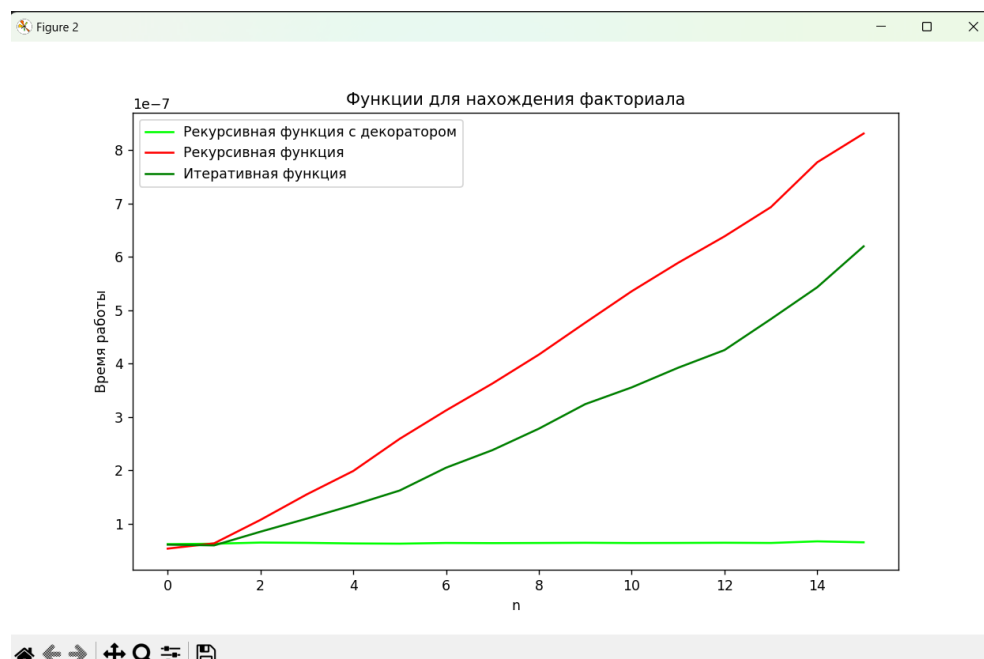


Рисунок 1 – Сравнение времени работы функций для нахождения факториала

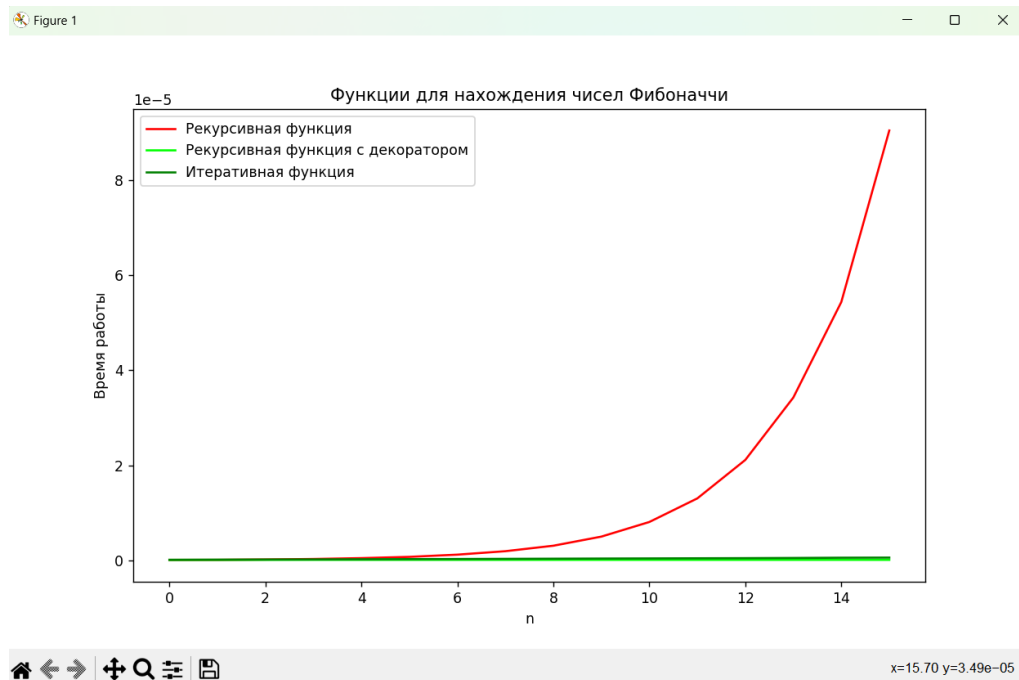


Рисунок 2 – Сравнение времени работы функций для вычисления чисел Фибоначчи

Код программы:

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

```
from functools import lru_cache
import matplotlib.pyplot as plt
import timeit
```

```
def factorial_bad(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n * factorial_bad(n - 1)
```

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

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

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

```
y_factorial_iteration = []
y_factorial_bad = []
y_factorial = []
x = [i for i in range(0,16)]
```

[illegible]

```

y_fib_iteration.append(time_fib_iteration)

time_factorial = ((timeit.timeit(lambda: factorial(i),
                                number = 100000))/100000)
y_factorial.append(time_factorial)

time_factorial_bad = ((timeit.timeit(lambda: factorial_bad(i),
                                    number = 100000))/100000)
y_factorial_bad.append(time_factorial_bad)

time_factorial_iteration = ((timeit.timeit(lambda: factorial_iteration(i),
                                           number = 100000))/100000)
y_factorial_iteration.append(time_factorial_iteration)

i += 1

plt.figure(figsize=(10,6))
plt.figure(1)
plt.title("Функции для нахождения чисел Фибоначчи")
plt.plot(x,y_fib_bad, color="red")
plt.plot(x,y_fib,color="lime")
plt.plot(x,y_fib_iteration,color="green")
plt.xlabel("n")
plt.ylabel("Время работы")
plt.legend(['Рекурсивная функция', 'Рекурсивная функция с декоратором',
           'Итеративная функция'])

plt.figure(figsize=(10,6))
plt.figure(2)
plt.title("Функции для нахождения факториала")
plt.legend("")
plt.plot(x,y_factorial,color="lime")
plt.plot(x, y_factorial_bad, color="red")
plt.plot(x, y_factorial_iteration, color="green")
plt.xlabel("n")
plt.ylabel("Время работы")
plt.legend(['Рекурсивная функция с декоратором', 'Рекурсивная функция',
           'Итеративная функция'])
plt.show()

```

Скорость работы рекурсивных функций при использовании декоратора `lru_cache` значительно возрастает. В рассмотренных примерах скорость работы рекурсивных функций с данным декоратором была либо приблизительно равна скорости работы итеративных функций, либо превышала её. Это связано с тем, что использование этого декоратора позволяет сократить объём лишних вычислений.

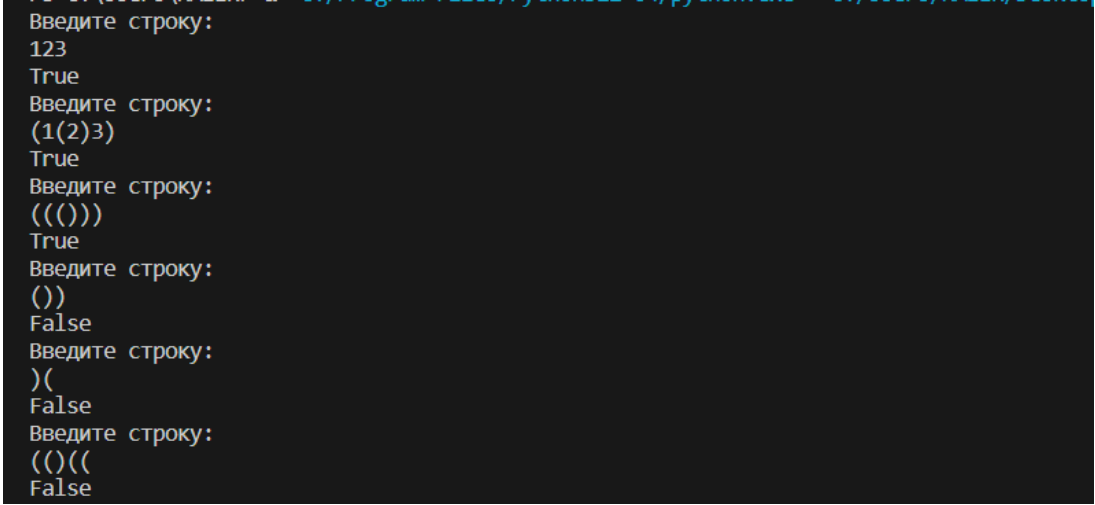
## 2. Выполнено индивидуальное задание

Напишите рекурсивную функцию, проверяющую правильность расстановки скобок в строке.

При правильной расстановке выполняются условия:

- 1) Количество открывающих и закрывающих скобок равно;
- 2) Внутри любой пары открывающая – соответствующая закрывающая скобка, скобки расставлены правильно.

Результат работы программы:



```
Введите строку:
123
True
Введите строку:
(1(2)3)
True
Введите строку:
((()))
True
Введите строку:
()
False
Введите строку:
)(
False
Введите строку:
(()((
False
```

Рисунок 3 – Результат работы программы

Код программы:

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

def skobka(s):
    if s.find("(") == -1 and s.rfind(")") == -1:
        return True
    elif (s.find("(") == -1 or s.rfind(")") == -1
          or s.rfind(")") < s.find("(")):
        return False
    else:
        return skobka(s[s.find("(")+1:s.rfind(")"]])

if __name__ == '__main__':
    while True:
        n = input("Введите строку:\n")
        if n == "exit":
            break
        print(skobka(n))
```

## Ответы на контрольные вопросы

### 1. Для чего нужна рекурсия?

Рекурсия нужна, когда требуется выполнить последовательность из одинаковых действий.

### 2. Что называется базой рекурсии?

База 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 программы – структура данных, используемая для хранения вызовов программы, куда при вызове функции помещается информация о текущем состоянии, а при завершении работы функции информация извлекается и управление передаётся вызывающей функции.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python?

Для получения текущего значения максимальной глубины рекурсии необходимо использовать функцию `sys.getrecursionlimit()`.

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

Если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python, то возникнет ошибка `RecursionError`.

### 6. Как изменить максимальную глубину рекурсии в языке Python?

Для изменения максимальной глубины рекурсии в языке Python используется функция `sys.setrecursionlimit(<значение глубины рекурсии>)`.

### 7. Каково назначение декоратора `lru_cache`?

Декоратор `@lru_cache()` 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. Подобный вид рекурсии примечателен тем, что может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода функции.

Оптимизация хвостовой рекурсии проводится путём преобразования её в плоскую итерацию.

#### Вывод

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