

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

Институт цифрового развития
Кафедра инфокоммуникаций

«Рекурсия в языке Python»

ОТЧЕТ
по лабораторной работе №12
дисциплины
«Основы программной инженерии»

Выполнил:

Гълбачева Доротея Андреева
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

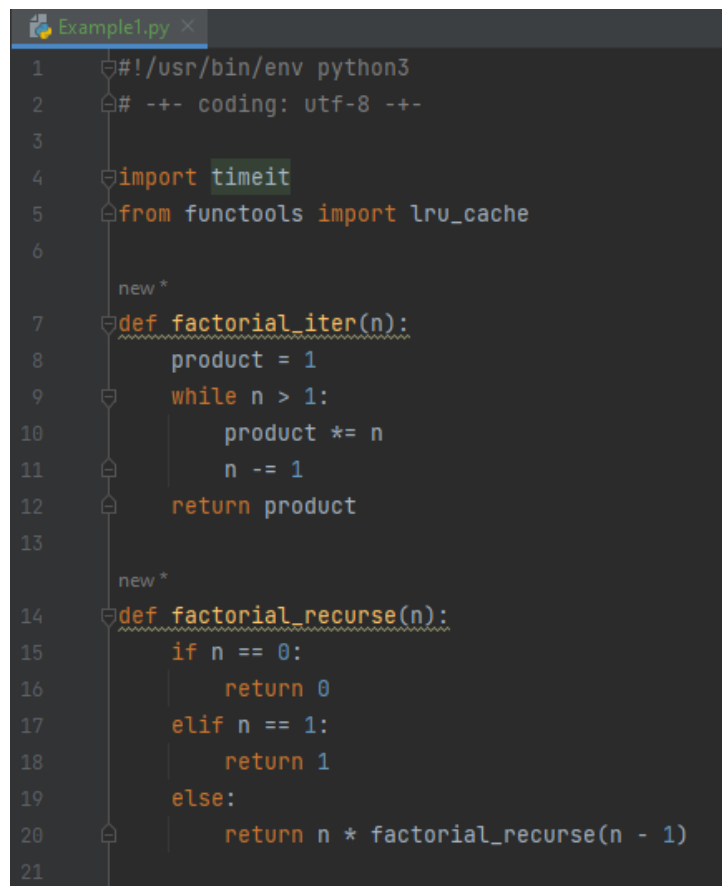
Цель работы: приобретение навыков по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

Выполнения лабораторной работы:

1. Проработка примеров из лабораторной работы:

Задание №1:

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.



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import timeit
5  from functools import lru_cache
6
7  new *
8  def factorial_iter(n):
9      product = 1
10     while n > 1:
11         product *= n
12         n -= 1
13     return product
14
15  new *
16  def factorial_recurse(n):
17     if n == 0:
18         return 0
19     elif n == 1:
20         return 1
21     else:
22         return n * factorial_recurse(n - 1)
```

Рисунок 12.1 - Код программы задания №1

```

new *
22 @lru_cache
23 def factorial_rec_lru(n):
24     if n == 0:
25         return 1
26     elif n == 1:
27         return 1
28     else:
29         return n * factorial_recurse(n - 1)
30
31 if __name__ == '__main__':
32     print("Время затраченное на итеративную версию")
33     print(f'{timeit.timeit(lambda: factorial_iter(500), number=10000)},\n')
34     print("Время затраченное на рекурсивную версию")
35     print(f'{timeit.timeit(lambda: factorial_recurse(500), number=10000)},\n')
36     print("Время затраченное на рекурсивную версию с lru_cache")
37     print(f'{timeit.timeit(lambda: factorial_rec_lru(500), number=10000)}\n')

```

Рисунок 12.2 - Код программы задания №1

```

Example1 x
C:\Users\user\LabR_12\venv\Scripts\python.exe C:\Users
Время затраченное на итеративную версию
4.138645800063387,

Время затраченное на рекурсивную версию
7.554331700084731,

Время затраченное на рекурсивную версию с lru_cache
0.006980399833992124

Process finished with exit code 0

```

Рисунок 12.3 - Результат работы программы задания №1

Задание №2:

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. Приведите полученные результаты в отчет.

```

Example1.py x Example2.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import timeit
5
6  new *
7  class recursion(object):
8      new *
9      def __init__(self, func):
10         self.func = func
11
12     new *
13     def __call__(self, *args, **kwargs):
14         result = self.func(*args, **kwargs)
15         while callable(result):
16             result = result()
17         return result
18
19     new *
20     def call(self, *args, **kwargs):
21         return lambda: self.func(*args, **kwargs)
22
23     new *
24     @recursion
25     def factorial_opt(n, acc=1):
26         if n == 0:

```

Рисунок 12.3 - Код программы задания №2

```

22         return acc
23     return factorial(n - 1, n * acc)
24
25     new *
26     def factorial(n, acc=1):
27         if n == 0:
28             return acc
29         return factorial(n - 1, n * acc)
30
31     if __name__ == '__main__':
32         print("Время работы программы с использованием интроспекции")
33         print(f'{timeit.timeit(lambda: factorial_opt(250), number=10000)}\n')
34         print("Время работы программы без использования интроспекции")
35         print(timeit.timeit(lambda: factorial(250), number=10000))

```

Рисунок 12.4 - Код программы задания №2

```

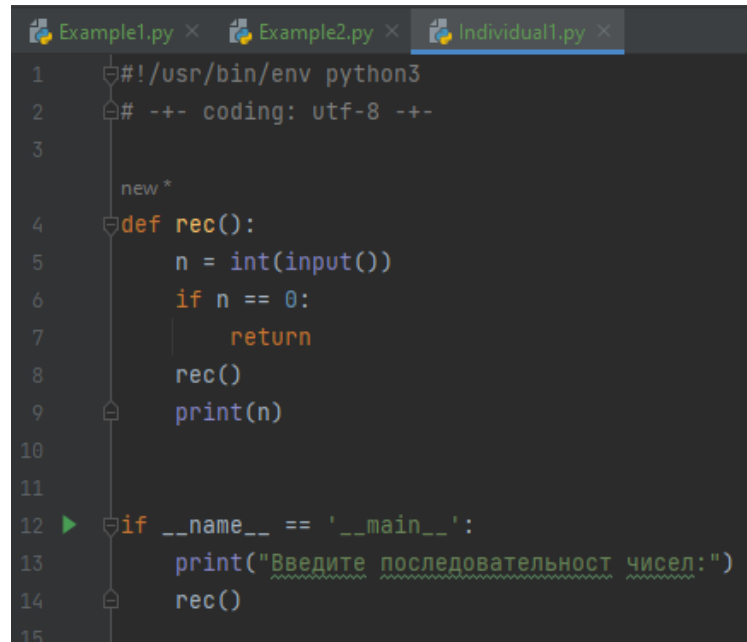
Run: Example2 x
C:\Users\user\LabR_12\venv\Scripts\python.exe C:\Users\
Время работы программы с использованием интроспекции
2.6328843999654055
Время работы программы без использования интроспекции
3.023416900075972

```

Рисунок 12.5 – Результат работы программы задания №2

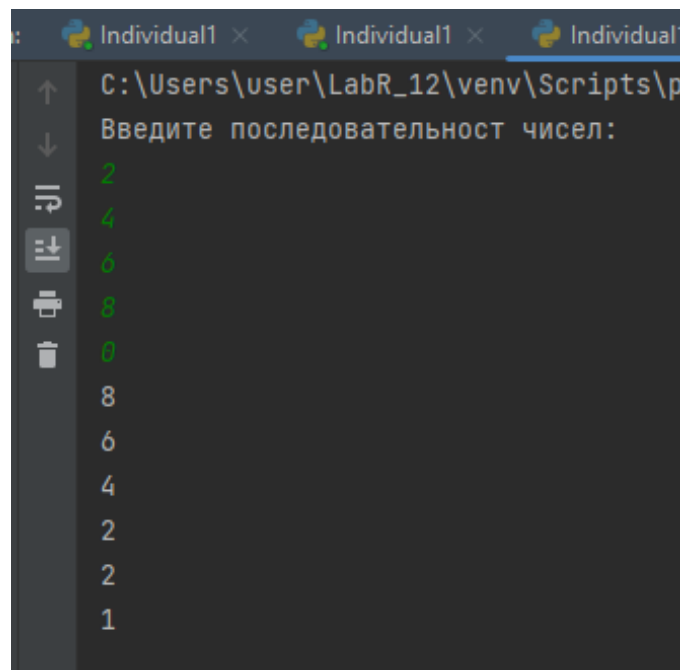
2. Индивидуальные задания:

Напечатать в обратном порядке последовательность чисел, ризнаком конца которой является 0.



```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  def rec():
5      n = int(input())
6      if n == 0:
7          return
8      rec()
9      print(n)
10
11
12 if __name__ == '__main__':
13     print("Введите последовательность чисел:")
14     rec()
15
```

Рисунок 12.6 - Код программы индивидуального задания №1



```
C:\Users\user\LabR_12\venv\Scripts\p
Введите последовательность чисел:
2
4
6
8
0
8
6
4
2
2
1
```

Рисунок 12.7 – Результат работы программы индивидуального задания №1

3. Контрольные вопросы:

1. Для чего нужна рекурсия?

Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя.

2. Что называется базой рекурсии?

У рекурсии, как и у математической индукции, есть база — аргументы, для которых значения функции определены

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

Максимальная глубина рекурсии ограничена движком JavaScript. Точно можно рассчитывать на 10000 вложенных вызовов, некоторые интерпретаторы допускают и больше, но для большинства из них 100000 вызовов — за пределами возможностей. Существуют автоматические оптимизации, помогающие избежать переполнения стека вызовов («оптимизация хвостовой рекурсии»), но они ещё не поддерживаются везде и работают только для простых случаев.

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

Функция `sys.getrecursionlimit()` возвращает текущее значение предела рекурсии, максимальную глубину стека интерпретатора Python.

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

Ошибка `RuntimeError`

6. Как изменить максимальную глубину рекурсии в языке Python?

С помощью функции `sys.setrecursionlimit()` модуля `sys`

7. Каково назначение декоратора `lru_cache` ?

Декоратор `@lru_cache()` модуля `functools` 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. Оптимизация хвостовой рекурсии выглядит так:

```
class recursion(object):
    "Can call other methods inside..."
    def __init__(self, func):
        self.func = func

    def __call__(self, *args, **kwargs):
        result = self.func(*args, **kwargs)
        while callable(result): result = result()
        return result

    def call(self, *args, **kwargs):
        return lambda: self.func(*args, **kwargs)

@recursion
def sum_natural(x, result=0):
    if x == 0:
        return result
    else:
        return sum_natural.call(x - 1, result + x)

# Даже такой вызов не заканчивается исключением
# RuntimeError: maximum recursion depth exceeded
print(sum_natural(1000000))
```