

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 11

Рекурсия в языке Python

**по дисциплине «Технологии программирования и
алгоритмизации»**

Выполнила студентка группы ИВТ-б-о-20-1

Хацукова А.И. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2021

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

Ход работы:

1. Изучила теоретическую часть лабораторной работы, проработала примеры и приступила к выполнению первого общего задания:

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

"""
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.
"""

from timeit import timeit

fib1 = ""
def fib(n):
    if n == 0 or n == 1:
        return n
```

```

        else:
            return fib(n - 2) + fib(n - 1)
'''

factorial1 = """
def factorial1(n):
    if n == 0:
        return 1
    else:
        return n * factorial1(n-1)
'''

fib2 = """
def fib2(n):
    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
    return a
'''

fib3 = """
from functools import lru_cache
@lru_cache
def fib3(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib3(n - 2) + fib3(n - 1)
'''

factorial2 = """
def factorial2(n):
    product = 1

```

```

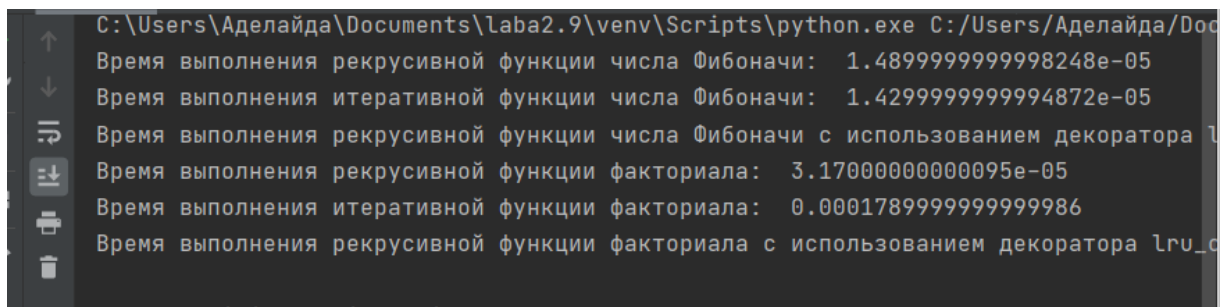
while n > 1:
    product *= n
    n -= 1
return product
"""

factorial3 = """
from functools import lru_cache
@lru_cache
def factorial3(n):
    if n == 0:
        return 1
    else:
        return n * factorial3(n-1)
"""

if __name__ == '__main__':
    print("Время выполнения рекурсивной функции числа Фибоначи: ",
          timeit(setup=fib1, number=1000))
    print("Время выполнения итеративной функции числа Фибоначи: ",
          timeit(setup=fib2, number=1000))
    print("Время выполнения рекурсивной функции числа Фибоначи с"
          " использованием декоратора lru_cache: ",
          timeit(setup=fib3, number=1000))
    print("Время выполнения рекурсивной функции факториала: ",
          timeit(setup=factorial1, number=1000))
    print("Время выполнения итеративной функции факториала: ",
          timeit(setup=factorial2, number=10000))
    print("Время выполнения рекурсивной функции факториала с"

```

```
" использованием декоратора lru_cache: ",  
timeit(setup=factorial3, number=10000))
```



```
C:\Users\Аделаида\Documents\laba2.9\venv\Scripts\python.exe C:/Users/Аделаида/Doc  
Время выполнения рекурсивной функции числа Фибоначи: 1.489999999998248e-05  
Время выполнения итеративной функции числа Фибоначи: 1.4299999999994872e-05  
Время выполнения рекурсивной функции числа Фибоначи с использованием декоратора l  
Время выполнения рекурсивной функции факториала: 3.170000000000095e-05  
Время выполнения итеративной функции факториала: 0.0001789999999999986  
Время выполнения рекурсивной функции факториала с использованием декоратора lru_c
```

Рисунок 1 – Результат выполнения первого задания

2. Второе общее задание:

```
#!/usr/bin/env python3  
  
# -*- coding: utf-8 -*-  
  
"""  
  
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.  
"""  
  
from timeit import timeit  
  
c1 = """  
def factorial(n, acc=1):  
    if n == 0:
```

```

        return acc

    return factorial(n-1, n*acc)
"""
c2 = """
def fib(i, current = 0, next = 1):
    if i == 0:
        return current
    else:
        return fib(i - 1, next, current + next)
"""
c3 = """
class TailRecurseException:
    def __init__(self, args, kwargs):
        self.args = args
        self.kwargs = kwargs
def tail_call_optimized(g):
    def func(*args, **kwargs):
        f = sys._getframe()
        while f and f.f_code.co_filename == f:
            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

```

```

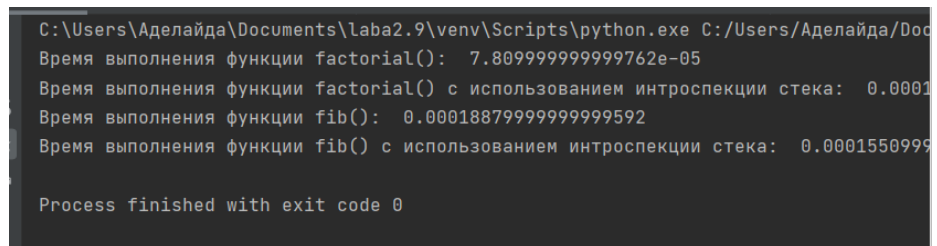
def factorial(n, acc=1):
    if n == 0:
        return acc
    return factorial(n-1, n*acc)
"""
c4 = """
class TailRecurseException:
    def __init__(self, args, kwargs):
        self.args = args
        self.kwargs = kwargs
def tail_call_optimized(g):
    def func(*args, **kwargs):
        f = sys._getframe()
        while f and f.f_code.co_filename == f:
            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 fib(i, current = 0, next = 1):
    if i == 0:
        return current
    else:
        return fib(i - 1, next, current + next)

```

```

if __name__ == '__main__':
    print("Время выполнения функции factorial(): ",
          timeit(setup=c1, number=10000))
    print("Время выполнения функции factorial() с"
          " использованием интроспекции стека: ",
          timeit(setup=c3, number=10000))
    print("Время выполнения функции fib(): ",
          timeit(setup=c2, number=10000))
    print("Время выполнения функции fib() с"
          " использованием интроспекции стека: ",
          timeit(setup=c4, number=10000))

```



```

C:\Users\Аделаида\Documents\laba2.9\venv\Scripts\python.exe C:/Users/Аделаида/Doc
Время выполнения функции factorial(): 7.809999999999762e-05
Время выполнения функции factorial() с использованием интроспекции стека: 0.0001
Время выполнения функции fib(): 0.00018879999999999592
Время выполнения функции fib() с использованием интроспекции стека: 0.0001550999
Process finished with exit code 0

```

Рисунок 2 – Результат выполнения второго задания

Исходя из полученных результатов можно убедиться, что использование декоратора существенно уменьшает время на обработку данных, в результате чего увеличивается скорость работы программы.

3. После приступила к выполнению индивидуального задания по варианту (20 (6) вариант)

```

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

def fun(n, list_n):

```



```

if n == 0:
    print(list_n[1], end="")
    for i in list_n[2:]:
        print(f'{{{i}}}', end="")
    print("")
else:
    x = list_n[len(list_n) - 1]
    if n >= x:
        for i in range(max(1, x), n + 1):
            fun(n - i, list_n + [i])

if __name__ == '__main__':
    num = int(input('Введите число: '))
    fun(num, [0])

```

```

C:\Users\Аделаида\Documents\laba2.9\venv\Scripts\python.exe C:/Users/Аделаид
Введите число: 3
1+1+1
1+2
3
Process finished with exit code 0

```

Рисунок 3 – Результат выполнения индивидуального задания

Контрольные вопросы:

1. Для чего нужна рекурсия?

Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя. Никакого парадокса здесь нет – компьютер лишь последовательно выполняет встретившиеся ему в программе команды и, если встречается вызов процедуры, просто начинает выполнять эту функцию. Без разницы, какая функция дала команду это делать.

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 в Python – это линейная структура данных, в которой данные расположены объектами друг над другом. Он хранит данные в режиме LIFO (Last in First Out). Данные хранятся в том же порядке, в каком на кухне тарелки располагаются одна над другой. Мы всегда выбираем последнюю тарелку из стопки тарелок. В стеке новый элемент вставляется с одного конца, и элемент может быть удален только с этого конца.

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`.

6. Как изменить максимальную глубину рекурсии в языке Python?

Изменить максимальную глубину рекурсии можно с помощью `sys.setrecursionlimit()`.

7. Каково назначение декоратора `lru_cache` ?

Декоратор `lru_cache` является полезным инструментом, который можно использовать для уменьшения количества лишних вычислений. Декоратор оборачивает функцию с переданными в нее аргументами и запоминает возвращаемый результат, соответствующий этим аргументам.

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