

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №12 по дисциплине «Основы
программной инженерии»

Выполнил:
Мамонтов Д.В.,
2 курс, группа ПИЖ-б-о-20-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г

ХОД РАБОТЫ

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 def rec(n):
6     if n == 1:
7         return 1
8     return n + rec(n - 1)
9
10
11 ▶ if __name__ == '__main__':
12     n = int(input("Enter n: "))
13     sum = 0
14     for i in range(1, n + 1):
15         sum += i
16     print(f"Сумма без рекурсии: {sum}")
17     print(f"Сумма с рекурсией: {rec(n)}")
```

Рисунок 1 – код программы

```
Enter n: 5
Сумма без рекурсии: 15
Сумма с рекурсией: 15

Process finished with exit code 0
```

Рисунок 2 – результат работы программы

```
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      from functools import lru_cache
5      import timeit
6
7
8      @lru_cache
9      def factorial_rec(n, acc=1):
10         if n == 0:
11             return acc
12         return factorial_rec(n - 1, n * acc)
13
14
15     @lru_cache
16     def fib_rec(i, current=0, next=1):
17         if i == 0:
18             return current
19         else:
20             return fib_rec(i - 1, next, current + next)
21
22
23     def factorial_iter(n):
24         if n == 0 or n == 1:
25             return 1
26         fact = 1
27         for i in range(1, n + 1):
28             fact *= i
29         return fact
```

Рисунок 3 – код программы

```

32 def fib_iter(n):
33     a = 0
34     b = 1
35     for i in range(n):
36         c = a + b
37         a = b
38         b = c
39     return a
40
41
42 if __name__ == '__main__':
43     number = int(input("Enter the number to calculate: "))
44     start_time = timeit.default_timer()
45     factorial_rec(number)
46     print("Recursive factorial time is: ", timeit.default_timer() - start_time)
47     start_time = timeit.default_timer()
48     factorial_iter(number)
49     print("Iterative factorial time is :", timeit.default_timer() - start_time)
50     start_time = timeit.default_timer()
51     fib_rec(number)
52     print("Recursive Fibonacci time is :", timeit.default_timer() - start_time)
53     start_time = timeit.default_timer()
54     fib_iter(number)
55     print("Iterative Fibonacci time is :", timeit.default_timer() - start_time)

```

Рисунок 4 – код программы (продолжение)

```

Enter the number to calculate: 300
Recursive factorial time is: 0.0016073000151664019
Iterative factorial time is : 8.240004535764456e-05
Recursive Fibonacci time is : 0.0005071000196039677
Iterative Fibonacci time is : 3.290001768618822e-05

Process finished with exit code 0

```

Рисунок 6 – результат работы программы при lru_cache

```
Enter the number to calculate: 300
Recursive factorial time is: 0.0016073000151664019
Iterative factorial time is : 8.240004535764456e-05
Recursive Fibonacci time is : 0.0005071000196039677
Iterative Fibonacci time is : 3.290001768618822e-05

Process finished with exit code 0
```

Рисунок 7 – результат работы программы без lru_cache

```
1  #!/usr/bin/env python3
2  #- coding: utf-8 -*-
3
4  import timeit
5  import sys
6
7
8  class TailRecurseException(Exception):
9      def __init__(self, args, kwargs):
10         self.args = args
11         self.kwargs = kwargs
12
13
14  def tail_call_optimized(g):
15      def func(*args, **kwargs):
16         f = sys._getframe()
17         if f.f_back and f.f_back.f_back and f.f_back.f_back.f_code == f.f_code:
18             raise TailRecurseException(args, kwargs)
19         else:
20             while True:
21                 try:
22                     return g(*args, **kwargs)
23                 except TailRecurseException as e:
24                     args = e.args
25                     kwargs = e.kwargs
26
27         func.__doc__ = g.__doc__
28         return func
29
30
31  @tail_call_optimized
32  def factorial_rec(n, acc=1):
```

Рисунок 8 – код программы

```
33         if n == 0:
34             return acc
35     return factorial_rec(n - 1, n * acc)
36
37
38     @tail_call_optimized
39     def fib_rec(i, current=0, next=1):
40         if i == 0:
41             return current
42         else:
43             return fib_rec(i - 1, next, current + next)
44
45
46     def factorial_iter(n):
47         if n == 0 or n == 1:
48             return 1
49         fact = 1
50         for i in range(1, n + 1):
51             fact *= i
52         return fact
53
54
55     def fib_iter(n):
56         a = 0
57         b = 1
58         for i in range(n):
59             c = a + b
60             a = b
61             b = c
62         return a
```

Рисунок 9 – код программы (продолжение)

```

63
64
65 ▶ if __name__ == '__main__':
66     number = int(input("Enter the number: "))
67
68     start_time = timeit.default_timer()
69     factorial_rec(number)
70     print("Recursive factorial time is: ",
71         timeit.default_timer() - start_time
72     )
73
74     start_time = timeit.default_timer()
75     factorial_iter(number)
76     print("Iterative factorial time is :",
77         timeit.default_timer() - start_time
78     )
79
80     start_time = timeit.default_timer()
81     fib_rec(number)
82     print("Recursive Fibonacci time is :",
83         timeit.default_timer() - start_time
84     )
85
86     start_time = timeit.default_timer()
87     fib_iter(number)
88     print("Iterative Fibonacci time is :",
89         timeit.default_timer() - start_time
90     )
91

```

Рисунок 10 – код программы (конец)

```

Enter the number: 300
Recursive factorial time is: 0.0022194000193849206
Iterative factorial time is : 0.00013890000991523266
Recursive Fibonacci time is : 0.005366299998778105
Iterative Fibonacci time is : 8.849997539073229e-05

Process finished with exit code 0

```

Рисунок 11 – результат работы программы с tail_call_optimized

```

Enter the number: 300
Recursive factorial time is: 0.0012026999611407518
Iterative factorial time is : 0.00010110007133334875
Recursive Fibonacci time is : 0.000569900032132864
Iterative Fibonacci time is : 5.9299985878169537e-05

Process finished with exit code 0

```

Рисунок 12 – результат работы программы без tail_call_optimized

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Задание:

Задан список положительных чисел, признаком конца которых служит отрицательное число. Используя рекурсию, подсчитать количество чисел и их сумму.

Код:

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

def summary(mas, inx):
    if int(mas[inx]) > 0:
        return summary(mas, inx + 1) + int(mas[inx])
    else:
        return 0

if __name__ == '__main__':
    ms = input()
    print(summary(ms.split(), 0))
```



```
1 3 7 1 2 -1 2
14
```

Рисунок 1 – результат работы программы

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, и рекурсия может работать в обратную сторону

2. Что называется базой рекурсии?

Случай, при котором мы не запускаем в рекурсию, к примеру, во время вычисления факториала базовый случай – это $\text{if } n == 0 \text{ or } n == 1:$
`return 1`

3. Самостоятельно изучите что является стеком программы.

Как используется стек программы при вызове функций?

Стек вызовов (от англ. call stack; применительно к процессорам — просто «стек») — в теории вычислительных систем, LIFO-стек, хранящий информацию для возврата управления из подпрограмм (процедур, функций) в программу (или подпрограмму, при вложенных или рекурсивных вызовах) и/или для возврата в программу из

обработчика прерывания (в том числе при переключении задач в многозадачной среде).

При вызове подпрограммы или возникновении прерывания, в стек заносится адрес возврата — адрес в памяти следующей инструкции приостановленной программы и управление передается подпрограмме или подпрограмме-обработчику. При последующем вложенном или рекурсивном вызове, прерывании подпрограммы или обработчика прерывания, в стек заносится очередной адрес возврата и т. д.

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

```
import sys
print(sys.getrecursionlimit())
```

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

Возникает исключение `RuntimeError` :
`RuntimeError: Maximum Recursion Depth Exceeded`

6. Как изменить максимальную глубину рекурсии в языке Python?

```
sys.setrecursionlimit(1500)
```

7. Каково назначение декоратора `lru_cache`?

Он 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 путём преобразования её в плоскую итерацию реализована во многих оптимизирующих компиляторах. В некоторых функциональных языках программирования спецификация гарантирует обязательную оптимизацию хвостовой рекурсии. Типовой механизм реализации вызова функции основан на сохранении адреса возврата, параметров и локальных переменных функции в стеке и выглядит следующим образом:

1) В точке вызова в стек помещаются параметры, передаваемые функции, и адрес возврата.

2) Вызываемая функция в ходе работы размещает в стеке собственные локальные переменные.

3) По завершении вычислений функция очищает стек от своих локальных переменных, записывает результат (обычно — в один из регистров процессора).

4) Команда возврата из функции считывает из стека адрес возврата и выполняет переход по этому адресу. Либо непосредственно перед, либо сразу после возврата из функции стек очищается от параметров