МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯРОССИЙСКОЙ 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 студент
2 курса, группы ПИЖ-б-о-20-1
Тотубалина С.С.
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2021 г

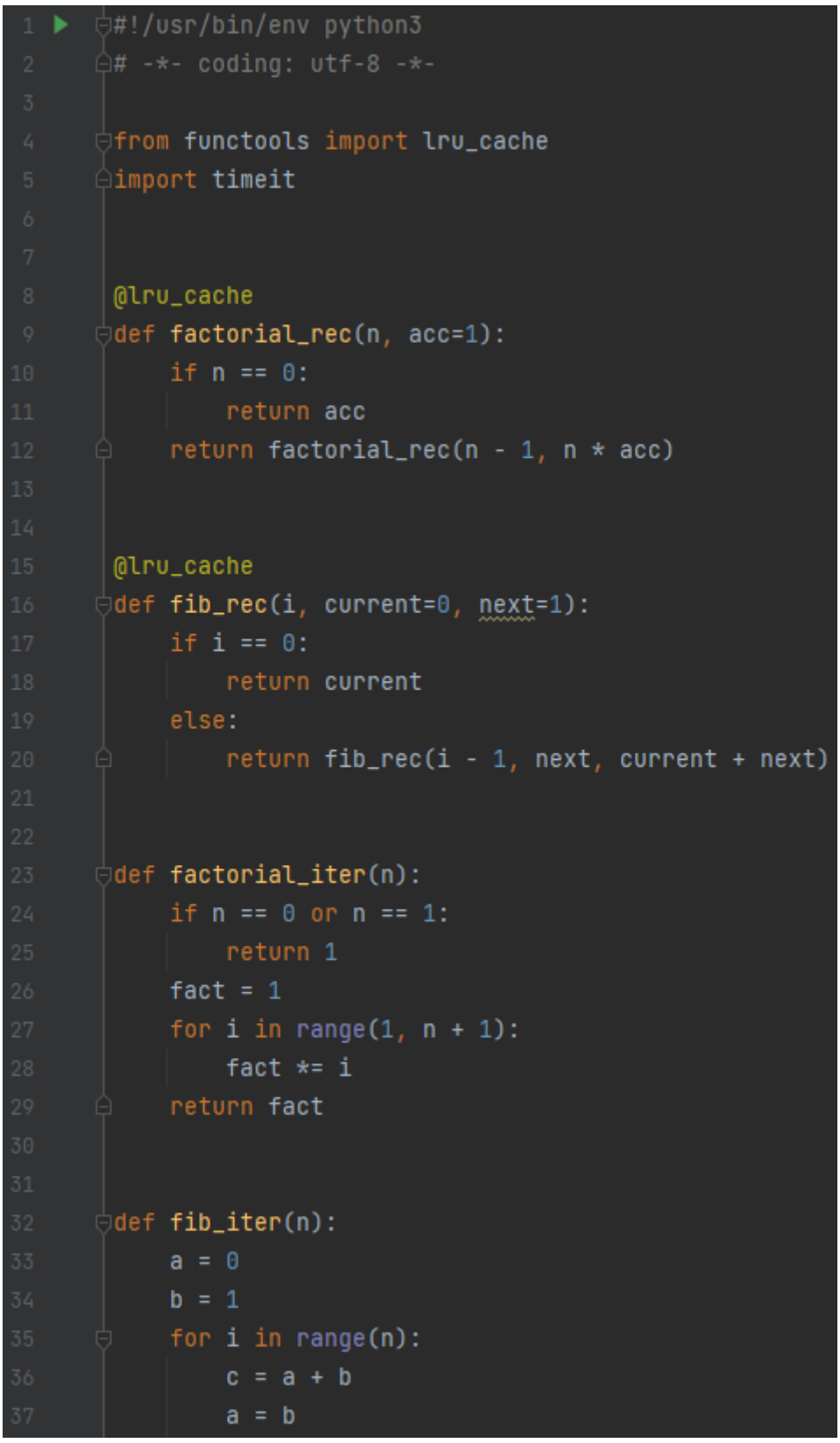
Ход работы

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


def rec(n):
    if n == 1:
        return 1
    return n + rec(n - 1)


if __name__ == '__main__':
    n = int(input("Enter n: "))
    sum = 0
    for i in range(1, n + 1):
        sum += i
    print(f"Сумма без рекурсии: {sum}")
    print(f"Сумма с рекурсией: {rec(n)}")
```

Рис. 1 – код программы lab.12_ex.1.py

```
Enter n: 3
Сумма без рекурсии: 6
Сумма с рекурсией: 6

Process finished with exit code 0
```

Рис. 2 – результат работы программы lab.12_ex.1.py



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

from functools import lru_cache
import timeit


@lru_cache
def factorial_rec(n, acc=1):
    if n == 0:
        return acc
    return factorial_rec(n - 1, n * acc)


@lru_cache
def fib_rec(i, current=0, next=1):
    if i == 0:
        return current
    else:
        return fib_rec(i - 1, next, current + next)


def factorial_iter(n):
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact


def fib_iter(n):
    a = 0
    b = 1
    for i in range(n):
        c = a + b
        a = b
```

Рис. 3 – код программы lab.12_ex.2.py

```
        b = c
    return a


if __name__ == '__main__':
    number = int(input("Enter the number to calculate: "))
    start_time = timeit.default_timer()
    factorial_rec(number)
    print("Recursive factorial time is: ",
          timeit.default_timer() - start_time
          )

    start_time = timeit.default_timer()
    factorial_iter(number)
    print("Iterative factorial time is :",
          timeit.default_timer() - start_time
          )

    start_time = timeit.default_timer()
    fib_rec(number)
    print("Recursive Fibonacci time is :",
          timeit.default_timer() - start_time
          )

    start_time = timeit.default_timer()
    fib_iter(number)
    print("Iterative Fibonacci time is :",
          timeit.default_timer() - start_time
          )
```

Рис. 4 – код программы lab.12_ex.2.py

```
Enter the number to calculate: 100
Recursive factorial time is:  0.00021259999999756474
Iterative factorial time is : 1.9399999999336615e-05
Recursive Fibonacci time is : 5.940000000137502e-05
Iterative Fibonacci time is : 8.400000002239949e-06

Process finished with exit code 0
```

Рис. 5 – пример работы программы lab.12_ex.2.py с lru_cache

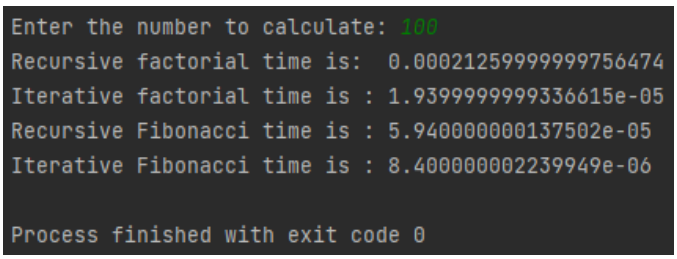



Рис. 6 – пример работы программы lab.12_ex.2.py без lru_cache

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

import timeit
import sys


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


def tail_call_optimized(g):
    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



def factorial_rec(n, acc=1):
    if n == 0:
        return acc
    return factorial_rec(n - 1, n * acc)



def fib_rec(i, current=0, next=1):
    if i == 0:
        return current
    else:
```

Рис. 7 – код программы lab.12_ex.3.py

```
    else:
        return fib_rec(i - 1, next, current + next)


def factorial_iter(n):
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact


def fib_iter(n):
    a = 0
    b = 1
    for i in range(n):
        c = a + b
        a = b
        b = c
    return a


if __name__ == '__main__':
    number = int(input("Enter the number: "))

    start_time = timeit.default_timer()
    factorial_rec(number)
    print("Recursive factorial time is: ",
          timeit.default_timer() - start_time
          )

    start_time = timeit.default_timer()
    factorial_iter(number)
    print("Iterative factorial time is :",
          timeit.default_timer() - start_time
          )

    start_time = timeit.default_timer()
    fib_rec(number)
    print("Recursive Fibonacci time is :",
          timeit.default_timer() - start_time
```

Рис. 8 – код программы lab.12_ex.3.py

```
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          )
```

Рис. 9 – код программы lab.12_ex.3.py

```
Enter the number: 10
Recursive factorial time is:  1.039999999985497e-05
Iterative factorial time is : 6.799999999973494e-06
Recursive Fibonacci time is : 5.800000000277805e-06
Iterative Fibonacci time is : 2.4999999999053557e-06

Process finished with exit code 0
```

Рис. 10 – пример работы программы lab.12_ex.3.py с tail_call_optimized

```
Enter the number: 10
Recursive factorial time is:  1.039999999985497e-05
Iterative factorial time is : 6.799999999973494e-06
Recursive Fibonacci time is : 5.800000000277805e-06
Iterative Fibonacci time is : 2.4999999999053557e-06

Process finished with exit code 0
```

Рис. 11 - пример работы программы lab.12_ex.3.py без tail_call_optimized

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  def subset(a, num, s):
6      if num == len(a):
7          print(s)
8          return
9      subset(a, num + 1, s)
10     s += str(a[num]) + ' '
11     subset(a, num + 1, s)
12
13
14 if __name__ == '__main__':
15     my_set = [int(i) for i in input("Enter the set: ").split()]
16     subset(my_set, 0, ' ')
```

Рис. 12 – код программы individual_12.py (Вариант №22)

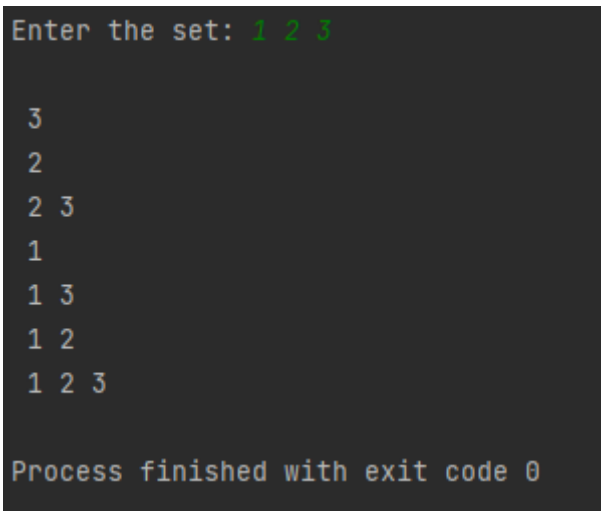



Рис. 13 - результат работы программы individual_12.py (Вариант №22) с тремя элементами

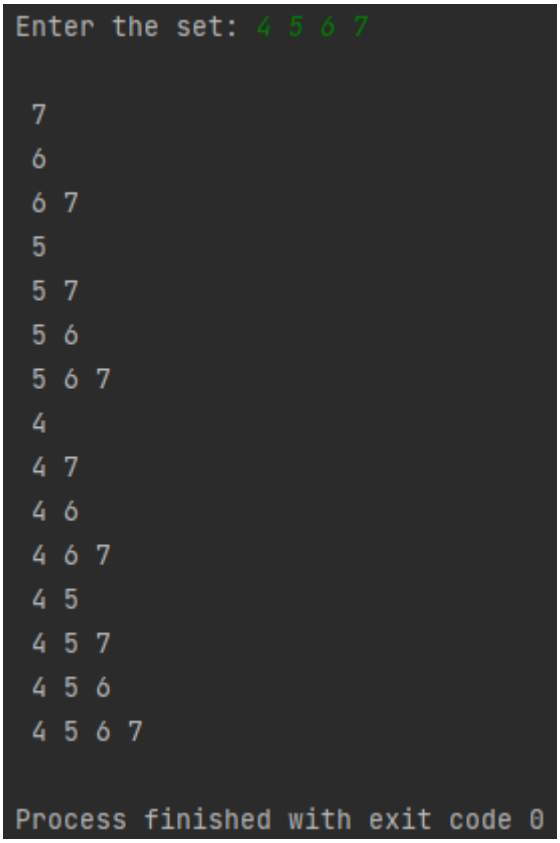



Рис. 14 - результат работы программы individual_12.py (Вариант №22) с четырьмя элементами

Ответы на 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 if n == 0 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