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:
Чернова Софья Андреевна,
2 курс, группа ПИЖ-б-о-20-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2021 г

1. Ход работы

1.1 Пример 1 (рис. 1, 2)

```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)}")
```

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

1.2 Задача 1 (рис. 3-7)

```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):
```

Рисунок 3 – код программы

```
    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 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()
```

Рисунок 4 – код программы (продолжение)

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

Рисунок 5 – код программы (окончание)

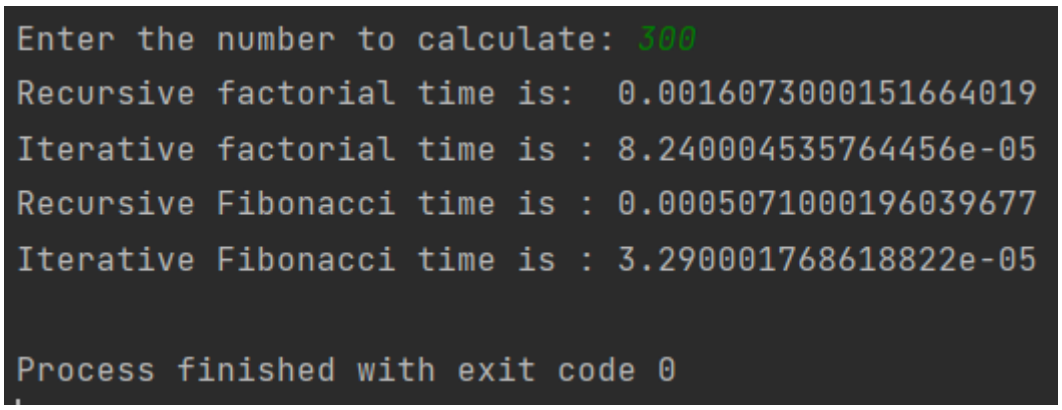



Рисунок 6 – результат работы программы при lru_cache

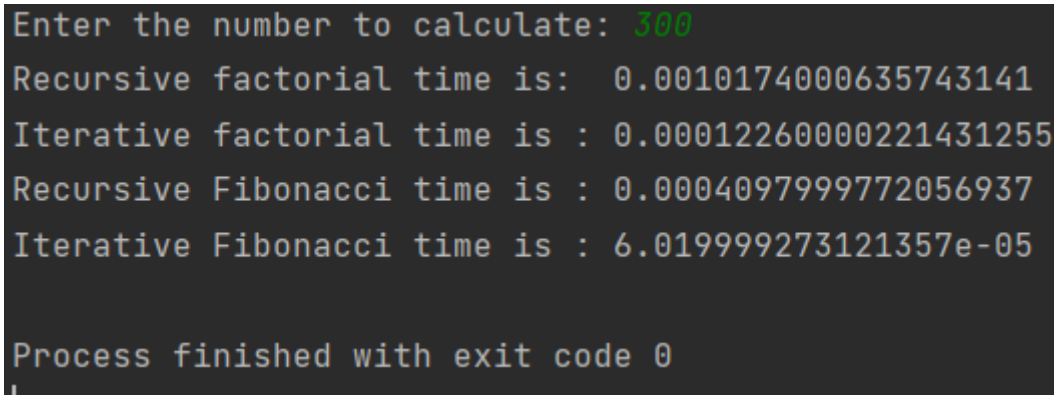



Рисунок 7 – результат работы программы без lru_cache

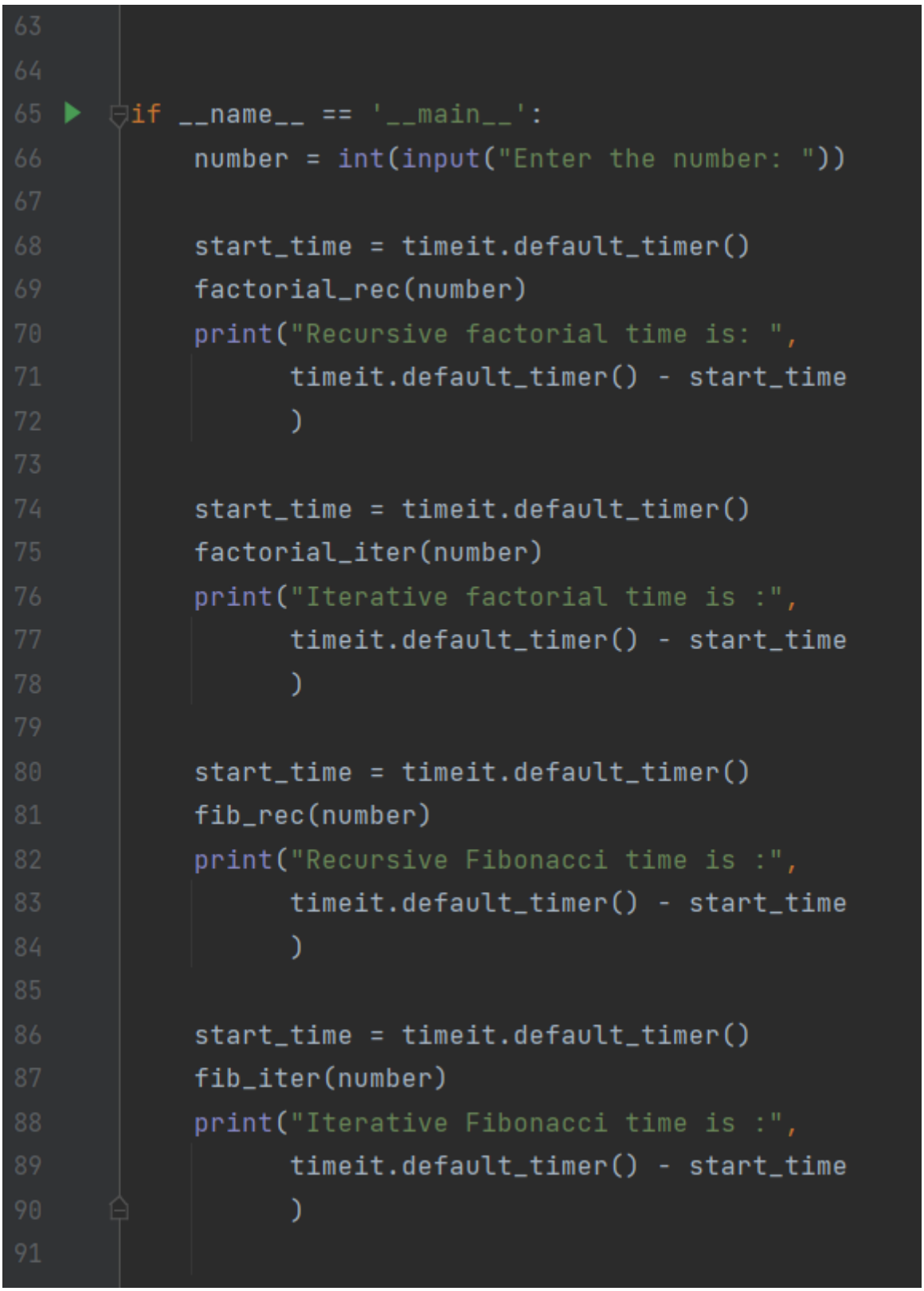
1.3 Задача 2 (рис. 8-12)

```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


@tail_call_optimized
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 (окончание)

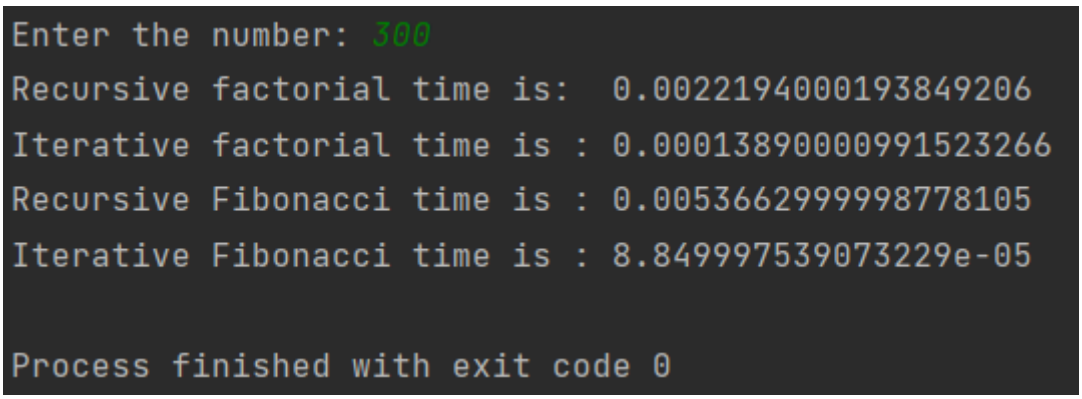



Рисунок 11 – результат работы программы с tail_call_optimized

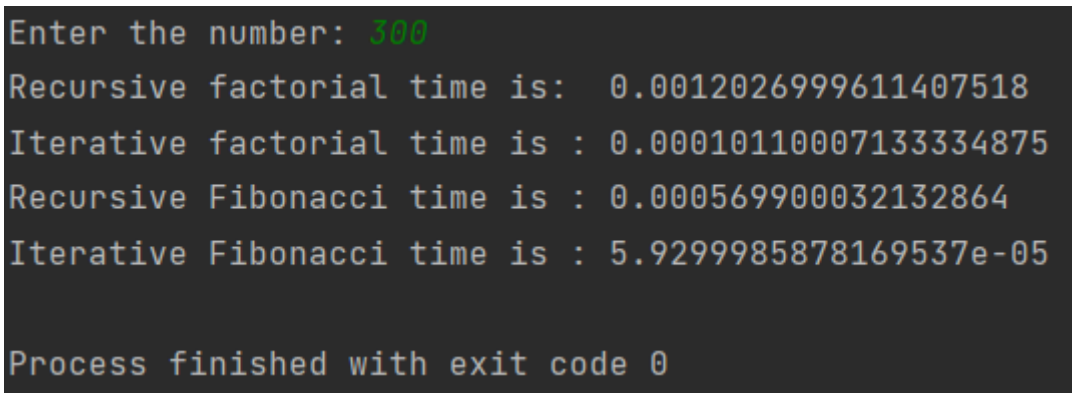



Рисунок 12 – результат работы программы без tail_call_optimized

1.4 Индивидуальное задание №10 (рис. 13 - 16)

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


x = float(input("Write x: "))
n = int(input("Write n: "))


def rec(x, n):
    if n == 0:
        print(1)
        return
    elif n < 0:
        x = 1/x**(abs(n))
        print(x)
        return
    elif n > 0:
        x = x * (x**(n-1))
        print(x)
        return


if __name__ == '__main__':
    rec(x, n)
```

Рисунок 13 – код программы

```
Write x: 3
Write n: 0
1

Process finished with exit code 0
```

Рисунок 14 – результат работы программы при n = 0

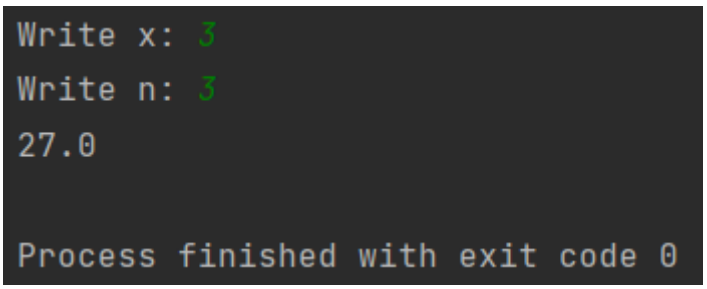



Рисунок 15 – результат работы программы при n > 0

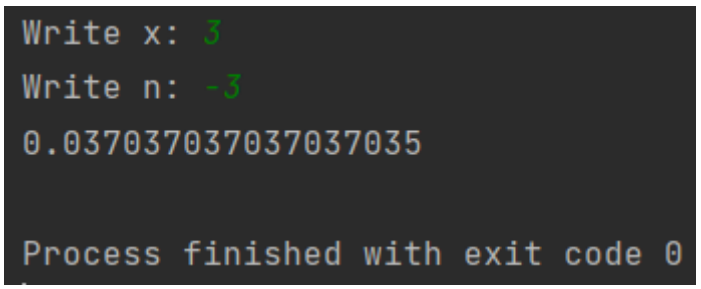



Рисунок 16 – результат работы программы при n < 0

2. Ответы на 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 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