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**

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – код программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – результат работы программы

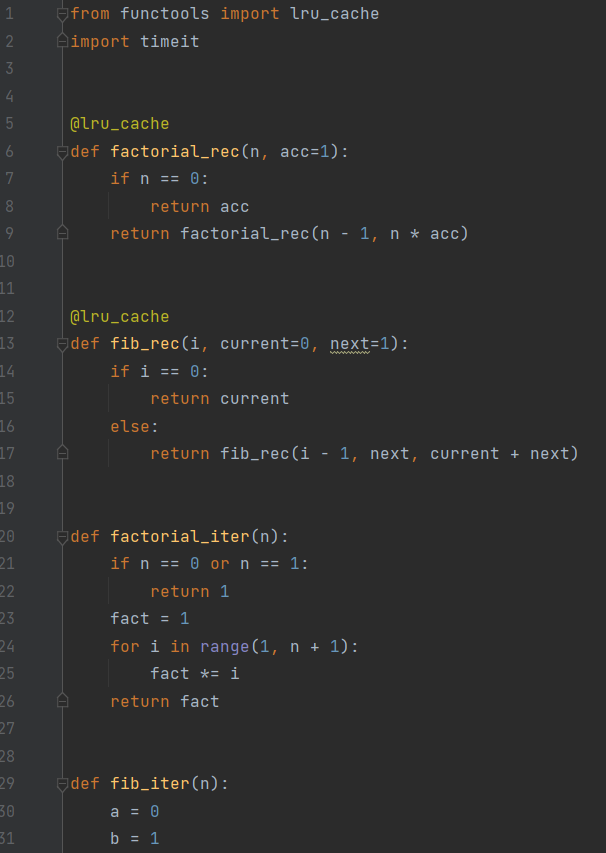


Рисунок 3 – код программы

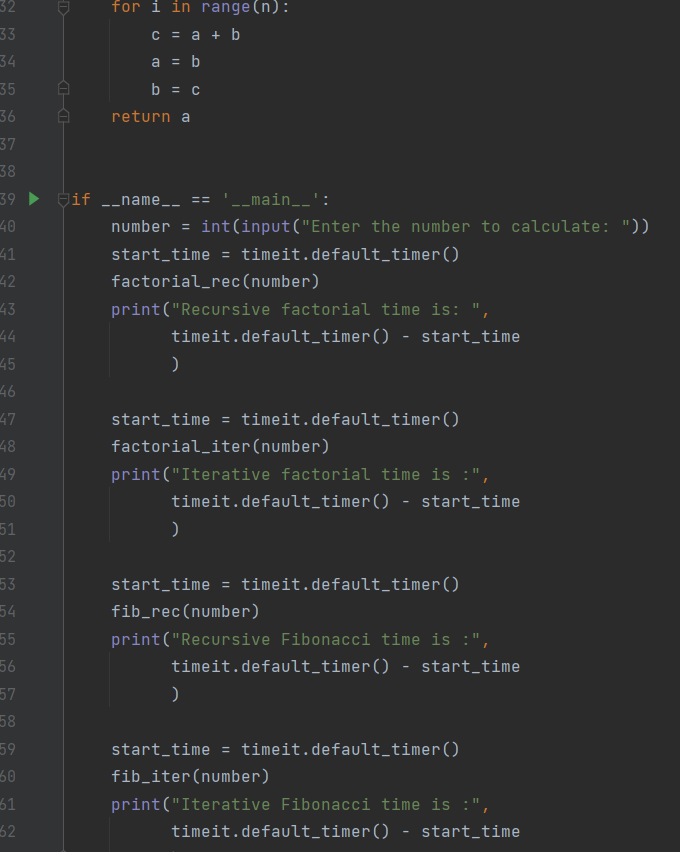


Рисунок 4 – код программы (продолжение)

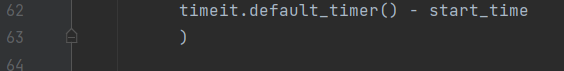


Рисунок 5 – код программы (конец)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 – результат работы программы при lru\_cache

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 – результат работы программы без lru\_cache

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 – код программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 – код программы (продолжение)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 10 – код программы (конец)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 – результат работы программы с tail\_call\_optimized

Изображение выглядит как текст

Автоматически созданное описание

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:

def summary(mas, inx):  
 if int(mas[inx]) > 0:  
 return summary(mas, inx + 1) + int(mas[inx])  
 else:  
 return 0  
  
  
ms = input()  
print(summary(ms.split(), 0))

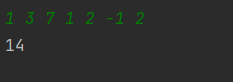


Рисунок 1 – результат работы программы

**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

1. 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

1. 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.

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

import sys

print(sys.getrecursionlimit())

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

Возникает исключение RuntimeError :

RuntimeError: Maximum Recursion Depth Exceeded

1. Как изменить максимальную глубину рекурсии в языке Python?

sys.setrecursionlimit(1500)

1. 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.

1. 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