Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО - КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра инфокоммуникаций

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №12**

**дисциплины «Программирование на Python»**

**Вариант 23**

|  |  |  |
| --- | --- | --- |
|  | Выполнил:  Омонкулов Исомиддин Валижон угли  2 курс, группа ИВТ-б-о-22-1,  09.03.01 «Информатика и вычислительная техника», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | Руководитель практики:  Воронкин Р. А.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | |  | |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ставрополь, 2024

**Тема:** Рекурсия в языке Python

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

**Задание.** 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.

**Листинг:**

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
  
import timeit  
from functools import lru\_cache  
  
  
# Итеративная версия функции factorial  
def factorial\_iterative(n):  
 result = 1  
 for i in range(1, n+1):  
 result \*= i  
 return result  
  
  
# Рекурсивная версия функции factorial  
def factorial\_recursive(n):  
 if n == 0 or n == 1:  
 return 1  
 else:  
 return n \* factorial\_recursive(n-1)  
  
  
# Итеративная версия функции fib  
def fib\_iterative(n):  
 a, b = 0, 1  
 for \_ in range(n):  
 a, b = b, a + b  
 return a  
  
  
# Рекурсивная версия функции fib  
def fib\_recursive(n):  
 if n <= 1:  
 return n  
 else:  
 return fib\_recursive(n-1) + fib\_recursive(n-2)  
   
  
# Рекурсивная версия функции factorial с использованием lru\_cache  
@lru\_cache  
def factorial\_recursive\_lru(n):  
 if n == 0 or n == 1:  
 return 1  
 else:  
 return n \* factorial\_recursive\_lru(n-1)  
   
  
# Рекурсивная версия функции fib с использованием lru\_cache  
@lru\_cache  
def fib\_recursive\_lru(n):  
 if n <= 1:  
 return n  
 else:  
 return fib\_recursive\_lru(n-1) + fib\_recursive\_lru(n-2)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Оценка скорости работы итеративной и рекурсивной версий функций  
 print("Factorial iterative:", timeit.timeit(lambda: factorial\_iterative(10), number=100000))  
 print("Factorial recursive:", timeit.timeit(lambda: factorial\_recursive(10), number=100000))  
 print("Fib iterative:", timeit.timeit(lambda: fib\_iterative(10), number=100000))  
 print("Fib recursive:", timeit.timeit(lambda: fib\_recursive(10), number=100000))  
  
 # Оценка скорости работы рекурсивных версий функций с использованием lru\_cache  
 print("Factorial recursive with lru\_cache:", timeit.timeit(lambda: factorial\_recursive\_lru(10), number=100000))  
 print("Fib recursive with lru\_cache:", timeit.timeit(lambda: fib\_recursive\_lru(10), number=100000))

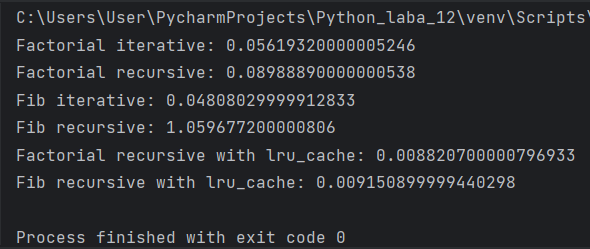


Рисунок 1. Результат программы

**Пояснение.** При запуске этого скрипта вы увидите результаты оценки скорости работы различных версий функций. Для оценки изменения скорости работы рекурсивных версий функций при использовании декоратора lru\_cache, можно сравнить результаты оценки скорости работы рекурсивных функций до и после применения lru\_cache.

Результаты могут показать значительное улучшение скорости работы рекурсивных функций после применения lru\_cache, ***так как кэширование результатов предыдущих вызовов позволяет избежать повторных вычислений.***

***number*** в функции timeit.timeit() - это количество выполнений, которые нужно сделать для оценки времени выполнения. Например, если мы установим number=100000, то функция будет выполнена 100000 раз, и затем будет измерено общее время выполнения. Это помогает получить более точные измерения времени выполнения, особенно для быстрых операций.

**Индивидуальное задание.** Создайте функцию, подсчитывающую сумму элементов массива по следующему алгоритму: массив делится пополам, подсчитываются и складываются суммы элементов в каждой половине. Сумма элементов в половине массива подсчитывается по тому же алгоритму, то есть снова путем деления пополам. Деления происходят, пока в получившихся кусках массива не окажется по одному элементу и вычисление суммы, соответственно, не станет тривиальным.

**Листинг:**

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
  
def recursive\_sum(arr):  
 if len(arr) == 1:  
 return arr[0]  
 else:  
 mid = len(arr) // 2  
 left\_sum = recursive\_sum(arr[:mid])  
 right\_sum = recursive\_sum(arr[mid:])  
 return left\_sum + right\_sum  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 input\_arr = [int(x) for x in input("Введите элементы массива через пробел: ").split()]  
 print("Сумма элементов массива:", recursive\_sum(input\_arr))

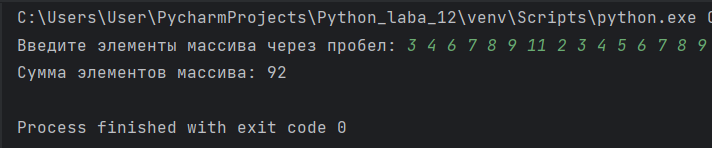


Рисунок 2. Результат программы

**Вывод:** в ходе выполнения данной лабораторной работы были приобретены навыки пользования рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

**Ответы на контрольные вопросы**

1. Рекурсия - это процесс, при котором функция вызывает саму себя. Она используется для решения задач, которые могут быть разбиты на более простые подзадачи того же типа. Рекурсия делает код более читаемым и понятным, особенно для задач, связанных с древовидными или иерархическими структурами данных

2. База рекурсии - это условие, при котором рекурсивные вызовы функции прекращаются. Без базы рекурсии функция будет вызывать саму себя бесконечно.

3. Стек программы - это структура данных, используемая для хранения информации о вызовах функций в программе. При вызове функции информация о текущем состоянии функции (аргументы, локальные переменные и адрес возврата) помещается в стек. Когда функция завершает свою работу, информация извлекается из стека.

4. Текущее значение максимальной глубины рекурсии в Python можно получить с помощью sys.getrecursionlimit().

5. Если число рекурсивных вызовов превысит максимальную глубину рекурсии в Python, будет возбуждено исключение RecursionError.

6. Максимальную глубину рекурсии в Python можно изменить с помощью функции sys.setrecursionlimit() из модуля sys. Однако изменение этого значения может повлиять на производительность и стабильность программы, поэтому следует быть осторожным при его изменении.

7. Декоратор lru\_cache используется для кэширования результатов вызовов функции, чтобы избежать повторных вычислений при одинаковых аргументах. Это может значительно улучшить производительность функций, особенно в случае рекурсивных вызовов.

8. Хвостовая рекурсия - это тип рекурсии, при котором рекурсивный вызов является последней операцией в функции. Оптимизация хвостовых вызовов заключается в том, что компилятор или интерпретатор может заменить рекурсивные вызовы на циклы, что уменьшает использование памяти и улучшает производительность.