

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 ПРАКТИЧЕСКОЙ РАБОТЕ №2.9
дисциплины «Программирование на Python»

Выполнил:
Степанов Леонид Викторович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение
средств вычислительной техники
и автоматизирование систем»,
очная форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., канд. техн. наук,
доцент, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

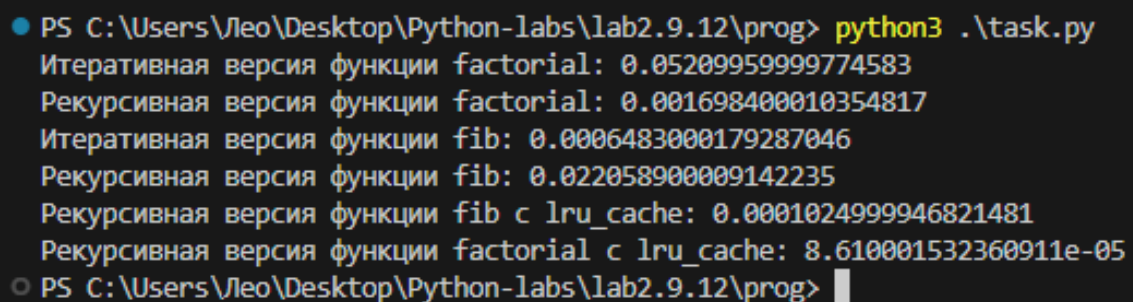
Ставрополь, 2023 г.

Тема: Рекурсия в языке Python

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

Порядок выполнения работы:

1. Создал файл (task.py), в котором изучил работу со стандартным пакетом timeit. Оценил 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?



```
PS C:\Users\Leo\Desktop\Python-labs\lab2.9.12\prog> python3 .\task.py
Итеративная версия функции factorial: 0.05209959999774583
Рекурсивная версия функции factorial: 0.001698400010354817
Итеративная версия функции fib: 0.0006483000179287046
Рекурсивная версия функции fib: 0.022058900009142235
Рекурсивная версия функции fib с lru_cache: 0.0001024999946821481
Рекурсивная версия функции factorial с lru_cache: 8.610001532360911e-05
PS C:\Users\Leo\Desktop\Python-labs\lab2.9.12\prog>
```

Рисунок 1 – Результат работы программы task.py

Исходя из полученных результатов из рис. 1, можно сделать вывод о том, что итеративная версия функции factorial работает медленнее рекурсивной, а итеративная функция fib быстрее рекурсивной. Скорость рекурсивных функций fib и factorial с lru_cache в много раз быстрее, чем без.

Индивидуальное задание (Вариант 7): 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.

```
prog > ind.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  def sum_of_midle(list_of_nums):
6      ...
7      Функция, которая считает сумму элементов списка
8      ...
9      if len(list_of_nums) == 1:
10         return list_of_nums[0]
11     else:
12         middle = len(list_of_nums) // 2
13         left_half_sum = sum_of_midle(list_of_nums[:middle])
14         right_half_sum = sum_of_midle(list_of_nums[middle:])
15         return left_half_sum + right_half_sum
16
17
18 def main():
19     ...
20     Главная функция программы.
21     ...
22     print(sum_of_midle([1, 2, 3, 4, 5]))
23
24
25 if __name__ == "__main__":
26     main()
27
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Code - prog + v

```
PS C:\Users\Leo\Desktop\Python-labs\lab2.9.12\prog> python3 .\ind.py
15
PS C:\Users\Leo\Desktop\Python-labs\lab2.9.12\prog>
```

Рисунок 2 – Результат работы программы ind.py

Вывод: в ходе выполнения работы были приобретены навыки по работе с рекурсивными функциями при написании программ с помощью языка Python версии 3.x.

Контрольные вопросы

1. Для чего нужна рекурсия?

Рекурсия используется в программировании для решения задач, которые могут быть разбиты на более мелкие подзадачи того же типа. Она позволяет функции вызывать саму себя, что упрощает решение сложных задач путем

разделения их на более простые подзадачи. Рекурсия также широко используется в алгоритмах, таких как алгоритмы обхода деревьев и графов.

2. Что называется базой рекурсии?

Условие, при котором рекурсивные вызовы функции прекращаются и начинается возврат из рекурсивных вызовов. Это базовый случай, который предотвращает бесконечное выполнение рекурсивной функции и обеспечивает завершение процесса.

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций?

Стек программы – это структура данных, которая хранит информацию о вызовах функций во время выполнения программы. При вызове функции, информация о текущем состоянии функции, такая как локальные переменные и адрес возврата, помещается в стек. Когда функция завершает выполнение, информация извлекается из стека, и управление передается обратно вызывающей функции. Это позволяет программе возвращаться к предыдущему состоянию после завершения выполнения функции.

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

В языке Python можно получить текущее значение максимальной глубины рекурсии с помощью функции `sys.getrecursionlimit()`. Она возвращает текущее максимальное количество рекурсивных вызовов, которое может быть выполнено до возникновения ошибки "RecursionError".

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

Если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python, возникнет ошибка "RecursionError". Это произойдет, когда программа пытается выполнить больше рекурсивных вызовов, чем разрешено текущей максимальной глубиной рекурсии.

6. Как изменить максимальную глубину рекурсии в языке Python?

Максимальную глубину рекурсии в языке Python можно изменить с помощью функции `sys.setrecursionlimit()`. Однако, изменение этого значения должно быть осуществлено с осторожностью, так как слишком большая глубина рекурсии может привести к переполнению стека и ошибкам выполнения.

7. Каково назначение декоратора `lru_cache` ?

Используется для кеширования результатов вызовов функции с определенными аргументами. Он сохраняет результаты предыдущих вызовов функции, чтобы избежать повторных вычислений при повторных вызовах с теми же аргументами. Это может значительно улучшить производительность функций, особенно при выполнении тяжелых вычислений.

8. Что такое хвостовая рекурсия? Как проводится оптимизация хвостовых вызовов?

Это особый вид рекурсии, при котором рекурсивный вызов является последней операцией в функции. Оптимизация хвостовых вызовов заключается в том, что компилятор или интерпретатор может заменить рекурсивный вызов на цикл, что позволяет избежать увеличения стека вызовов. Это позволяет снизить использование памяти и улучшить производительность программы.