

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»

Выполнил:
Говоров Егор Юрьевич
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., канд. технических
наук, 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.

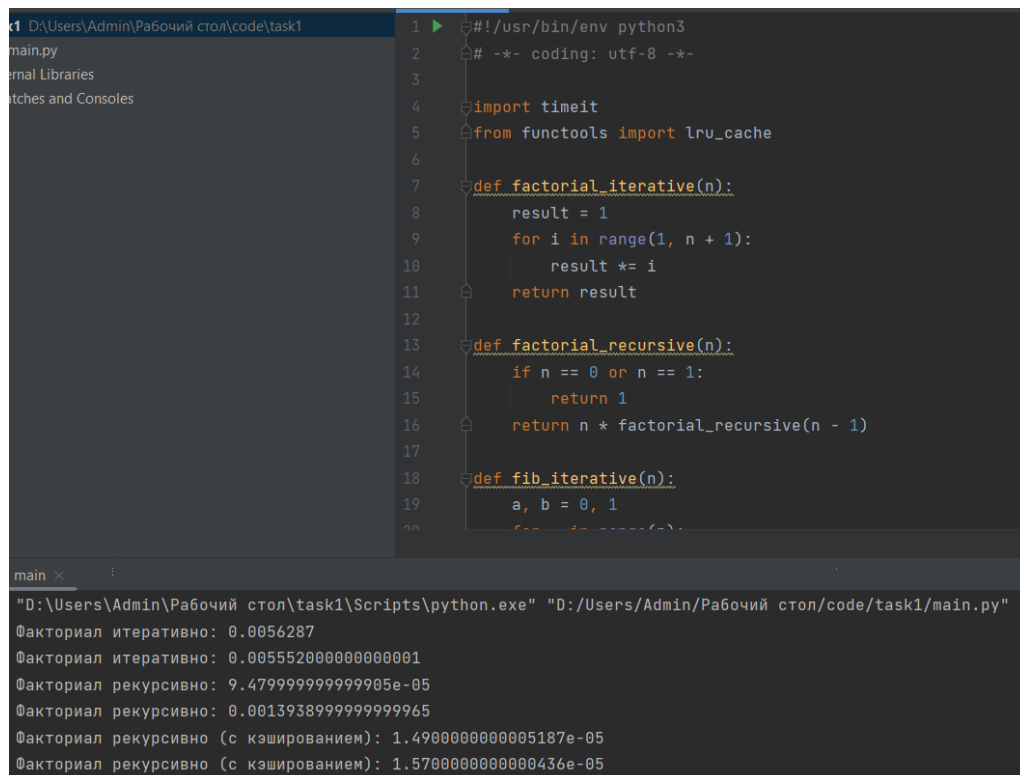
Ход работы

1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python. Выполнил клонирование созданного репозитория.

2. Дополнил файл .gitignore необходимыми правилами.

3. Организовал созданный репозиторий в соответствие с моделью ветвления git-flow.

5. Решил следующую задачу: 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.



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import timeit
5 from functools import lru_cache
6
7 def factorial_iterative(n):
8     result = 1
9     for i in range(1, n + 1):
10         result *= i
11     return result
12
13 def factorial_recursive(n):
14     if n == 0 or n == 1:
15         return 1
16     return n * factorial_recursive(n - 1)
17
18 def fib_iterative(n):
19     a, b = 0, 1
20     for i in range(n):
21         a, b = b, a + b
22     return a
```

main x

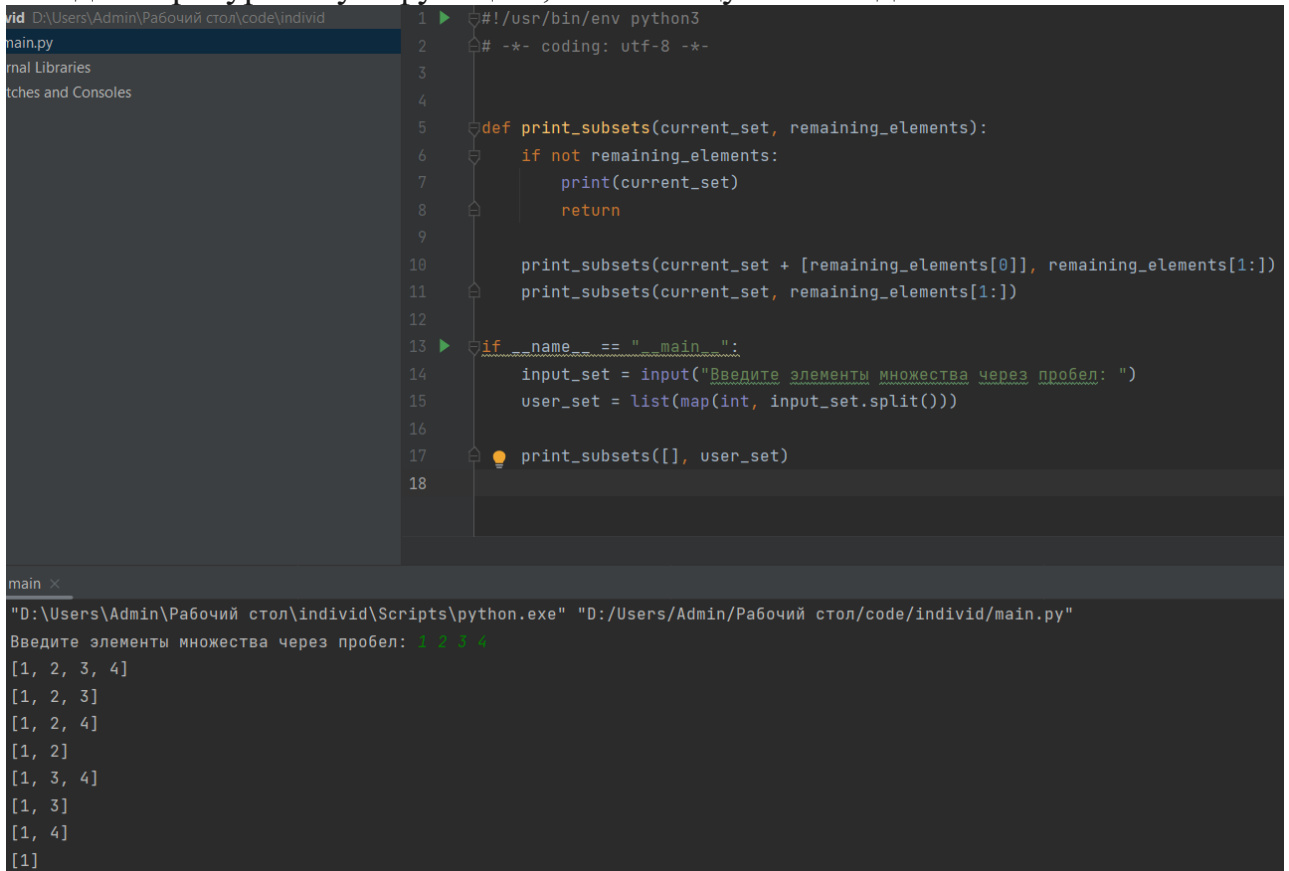
```
"D:\Users\Admin\Рабочий стол\task1\Scripts\python.exe" "D:/Users/Admin/Рабочий стол/code/task1/main.py"
Факториал итеративно: 0.0056287
Факториал итеративно: 0.005552000000000001
Факториал рекурсивно: 9.479999999999905e-05
Факториал рекурсивно: 0.001393899999999965
Факториал рекурсивно (с кэшированием): 1.49000000000005187e-05
Факториал рекурсивно (с кэшированием): 1.5700000000000436e-05
```

Рисунок 1. Результат работы программы из задачи 1

Анализируя полученные результаты времени работы функций, можно сказать, что итеративный подход в вычислении чисел Фибоначчи и факториалов на несколько порядков эффективнее, чем рекурсивный. Использование декоратора `@lru_cache` крайне значительно уменьшает время работы, это достигается путем оптимизации одинаковых повторяющихся подсчетов рекурсивной функции.

7. Выполнил индивидуальное задание (Вариант 5)

Создайте рекурсивную функцию, печатающую все подмножества множества



```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 def print_subsets(current_set, remaining_elements):
6     if not remaining_elements:
7         print(current_set)
8         return
9
10    print_subsets(current_set + [remaining_elements[0]], remaining_elements[1:])
11    print_subsets(current_set, remaining_elements[1:])
12
13 if __name__ == "__main__":
14     input_set = input("Введите элементы множества через пробел: ")
15     user_set = list(map(int, input_set.split()))
16
17     print_subsets([], user_set)
18
```

main x

"D:\Users\Admin\Рабочий стол\individ\Scripts\python.exe" "D:/Users/Admin/Рабочий стол/code/individ/main.py"

Введите элементы множества через пробел: 1 2 3 4

[1, 2, 3, 4]
[1, 2, 3]
[1, 2, 4]
[1, 2]
[1, 3, 4]
[1, 3]
[1, 4]
[1]

Рисунок 2. Результат работы программы из индивидуального задания

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 кэширования 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 программ.

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