12Министерство 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:
Кожуховский Виктор Андреевич
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 Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

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.

Методика и порядок выполнения работы

1. Изучил теоретический материал работы.

2.Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.

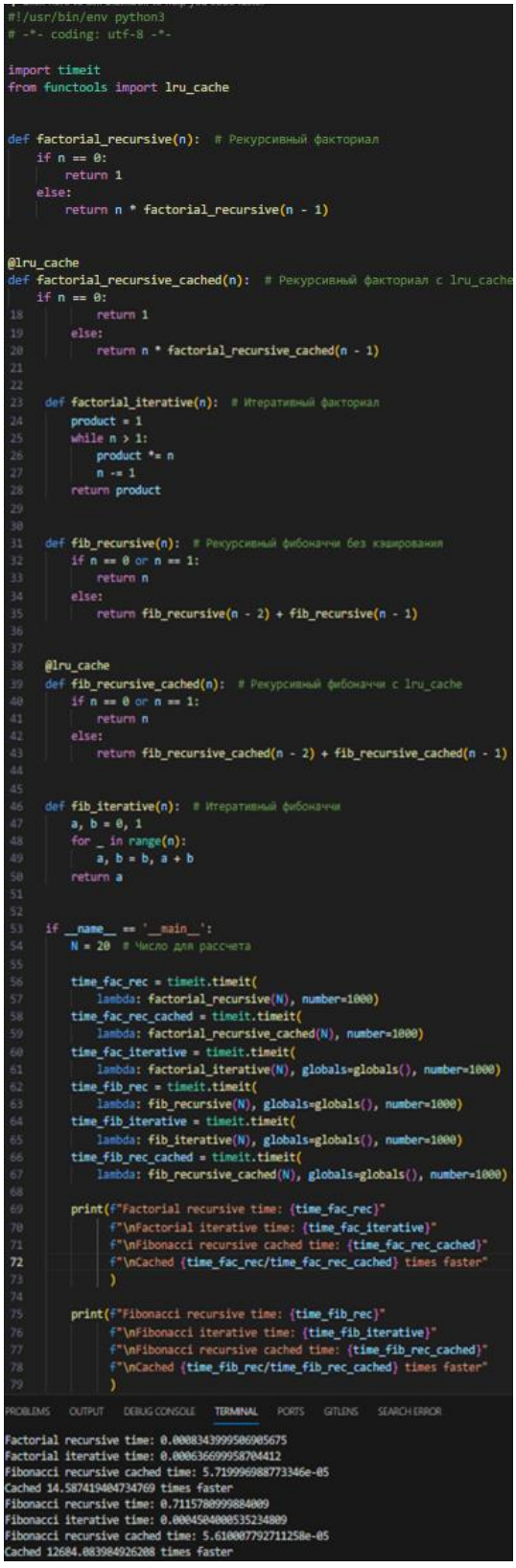
3. Выполнил клонирование созданного репозитория.

4. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.

5. Организовал свой репозиторий в соответствие с моделью ветвления git-flow.

6. Создал проект в папке репозитория.

7. 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.



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

import timeit
from functools import lru_cache


def factorial_recursive(n):  # Рекурсивный факториал
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)


@lru_cache
def factorial_recursive_cached(n):  # Рекурсивный факториал с lru_cache
    if n == 0:
        return 1
    else:
        return n * factorial_recursive_cached(n - 1)


def factorial_iterative(n):  # Итеративный факториал
    product = 1
    while n > 1:
        product *= n
        n -= 1
    return product


def fib_recursive(n):  # Рекурсивный фибоначчи без кэширования
    if n == 0 or n == 1:
        return n
    else:
        return fib_recursive(n - 2) + fib_recursive(n - 1)


@lru_cache
def fib_recursive_cached(n):  # Рекурсивный фибоначчи с lru_cache
    if n == 0 or n == 1:
        return n
    else:
        return fib_recursive_cached(n - 2) + fib_recursive_cached(n - 1)


def fib_iterative(n):  # Итеративный фибоначчи
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a


if __name__ == '__main__':
    N = 20  # Число для рассчета

    time_fac_rec = timeit.timeit(
        lambda: factorial_recursive(N), number=1000)
    time_fac_rec_cached = timeit.timeit(
        lambda: factorial_recursive_cached(N), number=1000)
    time_fac_iterative = timeit.timeit(
        lambda: factorial_iterative(N), globals=globals(), number=1000)
    time_fib_rec = timeit.timeit(
        lambda: fib_recursive(N), globals=globals(), number=1000)
    time_fib_iterative = timeit.timeit(
        lambda: fib_iterative(N), globals=globals(), number=1000)
    time_fib_rec_cached = timeit.timeit(
        lambda: fib_recursive_cached(N), globals=globals(), number=1000)

    print(f"Factorial recursive time: {time_fac_rec}"
          f"\nFactorial iterative time: {time_fac_iterative}"
          f"\nFibonacci recursive cached time: {time_fac_rec_cached}"
          f"\nCached {time_fac_rec/time_fac_rec_cached} times faster"
          )

    print(f"Fibonacci recursive time: {time_fib_rec}"
          f"\nFibonacci iterative time: {time_fib_iterative}"
          f"\nFibonacci recursive cached time: {time_fib_rec_cached}"
          f"\nCached {time_fib_rec/time_fib_rec_cached} times faster"
          )
```

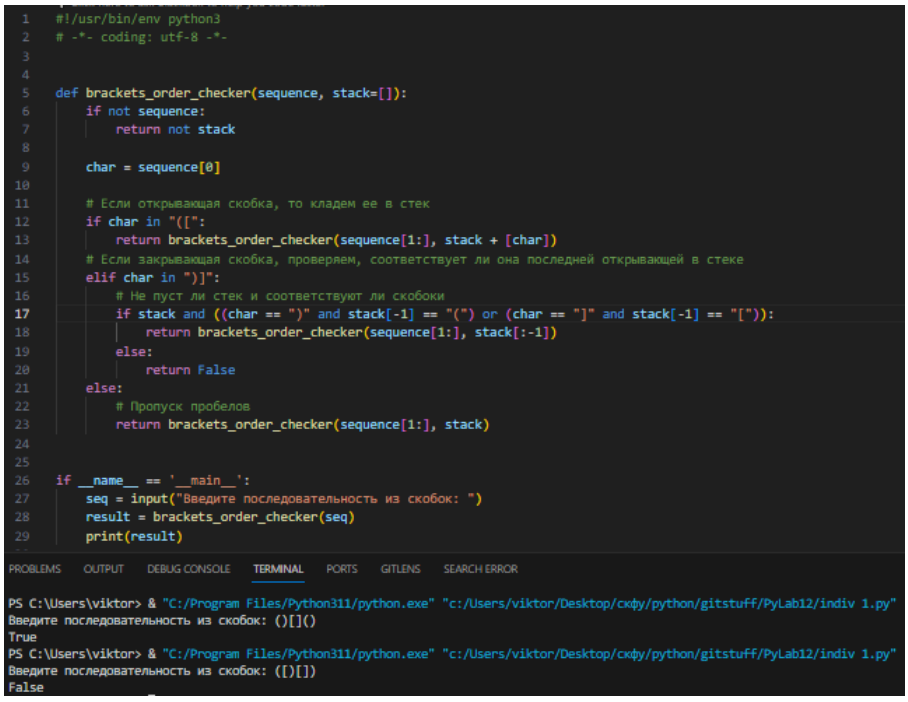
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SEARCH ERROR

Factorial recursive time: 0.0008343999586985675
Factorial iterative time: 0.0006366999587084412
Fibonacci recursive cached time: 5.719996988773346e-05
Cached 14.587419484734769 times faster
Fibonacci recursive time: 0.7115780999884009
Fibonacci iterative time: 0.0004504000535234809
Fibonacci recursive cached time: 5.610087792711258e-05
Cached 12684.083984926208 times faster
```

Рисунок 1. Код для задания 1 и его вывод

8. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

В строке могут присутствовать скобки как круглые, так и квадратные скобки. Каждой открывающей скобке соответствует закрывающая того же типа (круглой – круглая, квадратной- квадратная). Напишите рекурсивную функцию, проверяющую правильность расстановки скобок в этом случае.

Пример неправильной расстановки: ([ )].



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


def brackets_order_checker(sequence, stack=[]):
    if not sequence:
        return not stack

    char = sequence[0]

    # Если открывающая скобка, то кладем ее в стек
    if char in "([":
        return brackets_order_checker(sequence[1:], stack + [char])
    # Если закрывающая скобка, проверяем, соответствует ли она последней открывающей в стеке
    elif char in ")]":
        # Не пуст ли стек и соответствуют ли скобоки
        if stack and ((char == ")" and stack[-1] == "(") or (char == "]" and stack[-1] == "[")):
            return brackets_order_checker(sequence[1:], stack[:-1])
        else:
            return False
    else:
        # Пропуск пробелов
        return brackets_order_checker(sequence[1:], stack)


if __name__ == '__main__':
    seq = input("Введите последовательность из скобок: ")
    result = brackets_order_checker(seq)
    print(result)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SEARCH ERROR

PS C:\Users\viktor> & "C:/Program Files/Python311/python.exe" "c:/Users/viktor/Desktop/скфу/python/gitstuff/PyLab12/indiv 1.py"
Введите последовательность из скобок: ()[]()
True
PS C:\Users\viktor> & "C:/Program Files/Python311/python.exe" "c:/Users/viktor/Desktop/скфу/python/gitstuff/PyLab12/indiv 1.py"
Введите последовательность из скобок: ([)[])
False
```

Рисунок 2. Код для индивидуального задания и его вывод

9. Зафиксировал сделанные изменения в репозитории.

10. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.

11. Выполнил слияние ветки для разработки с веткой master / main.

12. Отправил сделанные изменения на сервер GitHub.

Вопросы для защиты работы

1. Для чего нужна рекурсия?

Рекурсия используется для решения задач, которые можно разбить на более мелкие, однотипные подзадачи.

2. Что называется базой рекурсии?

Это условие, при котором рекурсия прекращается, чтобы избежать бесконечного цикла вызовов. В каждой рекурсивной функции должен быть

как минимум один базовый случай, и он должен быть достижим от любого другого случая.

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

Стек программы - это область памяти, используемая операционной системой и программой для хранения данных о последовательности вызовов функций и переменных во время выполнения программы. При каждом вызове функции запись о ней (frame) помещается на вершину стека, содержащая информацию о значениях аргументов, локальных переменных и адресе, куда функция должна вернуть управление после своего выполнения. Когда функция завершает выполнение, ее фрейм удаляется со стека, и управление возвращается к точке вызова.

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

sys.getrecursionlimit()

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

Если число рекурсивных вызовов превышает максимальную глубину рекурсии в Python, возникает исключение RecursionError.

6. Как изменить максимальную глубину рекурсии в языке Python?

С помощью функции sys.setrecursionlimit(limit).

7. Каково назначение декоратора lru_cache?

Декоратор lru_cache из модуля functools предоставляет механизм кэширования результатов выполнения функций. Это означает, что результаты вызовов с одними и теми же аргументами сохраняются, и при повторном вызове функции с такими же аргументами они возвращаются из кэша без повторного выполнения самой функции. Это может значительно ускорить выполнения рекурсивных функций за счет избегания повторных вычислений.

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

Хвостовая рекурсия - это случай рекурсии, когда рекурсивный вызов является последним действием функции. При этом функция не должна выполнять никаких действий после рекурсивного вызова.

Оптимизация хвостовых вызовов позволяет переиспользовать стековый фрейм предыдущего вызова для следующего, что снижает общее потребление памяти и уменьшает риск переполнения стека. В некоторых языках программирования, таких как Scheme, хвостовая рекурсия может быть автоматически оптимизирована компилятором.