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:
Середа Кирилл Витальевич
1 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Лабораторная работа 2.9. Рекурсия в языке Python

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

Ход выполнения:

1) Задание 1 – самостоятельное изучение timeit и lru_cache

Timeit предназначен для измерения времени выполнения функции. Декоратор lru_cache из модуля functools может быть использован для кэширования результатов выполнения функций и предотвращения повторного выполнения функции для одних и тех же входных данных.

Провел эксперименты и сравнил различные методы:

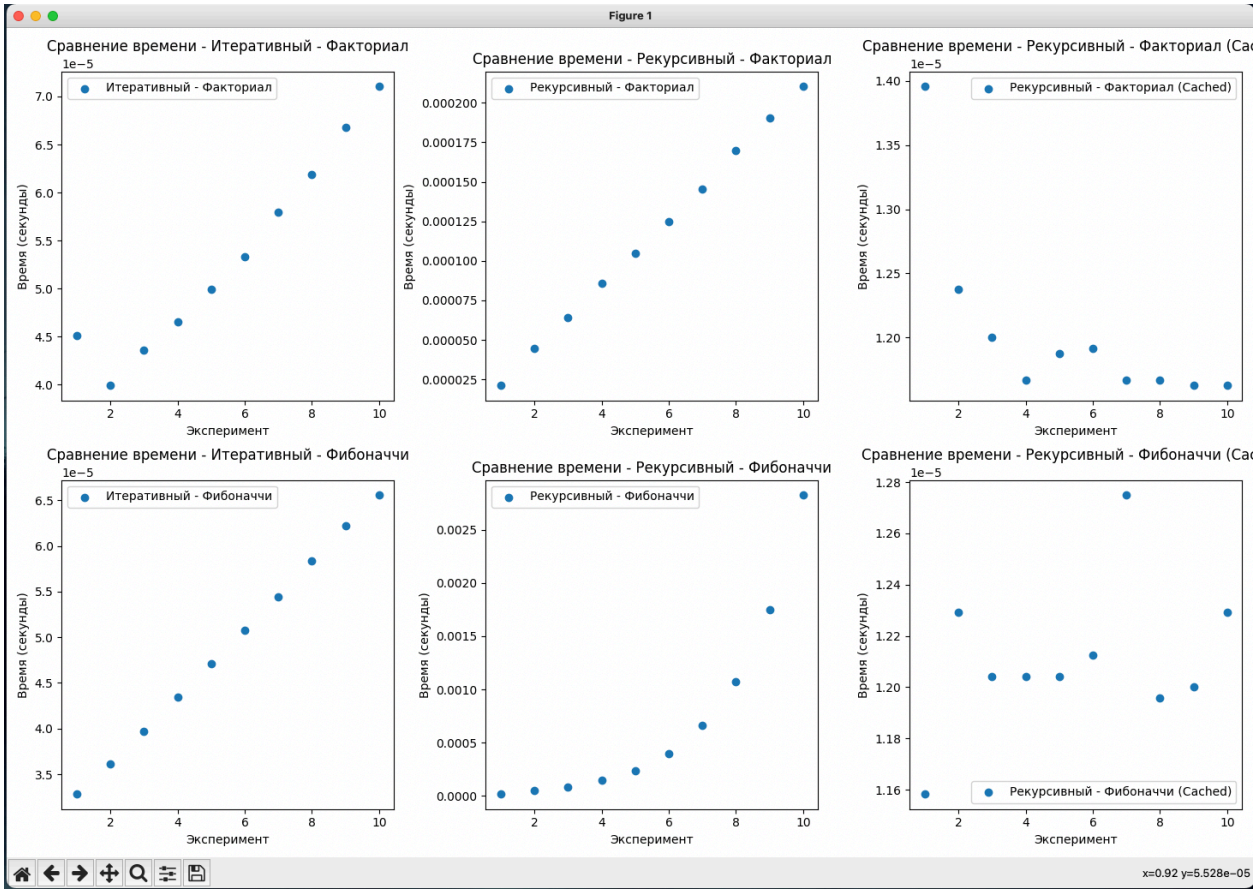

Рисунок 1 – Результаты экспериментов

Как мы можем заметить итеративный метод постоянно возрастает как при вычислении Факториала, так и при вычислении чисел фибоначчи. Рекурсивный метод при вычислении факториала также постоянно возрастает, но при это имеет гораздо меньшее время выполнения, однако при вычислении Фибоначчи существует некоторое плато времени выполнения, после которого время начинает возрастать. При использовании lru_cache

время выполнения не возрастает также как при других методах, а находится в районе погрешности, с некоторыми отклонениями от нормы.

Вывод: рекурсивный метод при использовании lru_cache является наиболее эффективным.

Код программы:

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

import timeit
from functools import lru_cache
import matplotlib.pyplot as plt
import numpy as np

def factorial_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial_recursive(n - 1)

@lru_cache(maxsize=None)
def factorial_recursive_cached(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial_recursive_cached(n - 1)

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

def fib_recursive(n):
    if n <= 1:
        return n
    return fib_recursive(n - 1) + fib_recursive(n - 2)

@lru_cache(maxsize=None)
def fib_recursive_cached(n):
    if n <= 1:
        return n
    return fib_recursive_cached(n - 1) + fib_recursive_cached(n - 2)

def measure_time(func, *args):
    return timeit.timeit(lambda: func(*args), number=100)

def plot_results(x, y, label, subplot):
    plt.subplot(subplot)
    plt.scatter(x, y, label=label)
    plt.xlabel('Эксперимент')
    plt.ylabel('Время (секунды)')
    plt.title(f'Сравнение времени - {label}')
    plt.legend()


if __name__ == '__main__':
    experiments = 30
    values = np.arange(1, 11)
```

```
    factorial_iterative_times = [measure_time(factorial_iterative, n) for n
in values]
    factorial_recursive_times = [measure_time(factorial_recursive, n) for n
in values]
    factorial_recursive_cached_times =
[measure_time(factorial_recursive_cached, n) for n in values]

    fib_iterative_times = [measure_time(fib_iterative, n) for n in values]
    fib_recursive_times = [measure_time(fib_recursive, n) for n in values]
    fib_recursive_cached_times = [measure_time(fib_recursive_cached, n) for n
in values]

    plt.figure(figsize=(15, 10))

    plot_results(values, factorial_iterative_times, 'Итеративный -
Факториал', 231)
    plot_results(values, factorial_recursive_times, 'Рекурсивный -
Факториал', 232)
    plot_results(values, factorial_recursive_cached_times, 'Рекурсивный -
Факториал (Cached)', 233)

    plot_results(values, fib_iterative_times, 'Итеративный - Фибоначчи', 234)
    plot_results(values, fib_recursive_times, 'Рекурсивный - Фибоначчи', 235)
    plot_results(values, fib_recursive_cached_times, 'Рекурсивный - Фибоначчи
(Cached)', 236)

    plt.tight_layout()
    plt.show()
```

2) Индивидуальное задание 1- Вариант 1: написать программу проверяющую баланс скобок

```
Введите строку: (цвц)(вцвц__)(цв(вц)цв)
Строка '(цвц)(вцвц__)(цв(вц)цв)' сбалансированна.
```

Рисунок 2 – Результат выполнения программы

Код программы:

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

def is_balanced(string, index=0, open_count=0):
    if index == len(string):
        return open_count == 0

    current_char = string[index]

    if current_char == '(':
        return is_balanced(string, index + 1, open_count + 1)
    elif current_char == ')':
        return open_count > 0 and is_balanced(string, index + 1, open_count -
1)
    else:
        return is_balanced(string, index + 1, open_count)

if __name__ == '__main__':
    input_string = input("Введите строку: ")
    result = is_balanced(input_string)

    if result:
```

```
        print(f"Строка '{input_string}' сбалансированна.")
    else:
        print(f"Строка '{input_string}' не сбалансированна.")
```

Ответы на 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, рекурсия может работать в обратную сторону. Принимая во внимание, что цикл for работает строго вперед, иногда рекурсивное решение проще, чем итеративное решение. Это очевидно при реализации обращения связанного списка.

2. Что называется базой рекурсии? Базовый случай — это возврат значения, не обращаясь к самой функции. Базовый случай необходим, без него была бы бесконечная рекурсия.

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций? Стек программы (или вызовов) — это структура данных, используемая компьютерной программой для управления вызовами функций во время их выполнения. При вызове функции текущее состояние программы, включая локальные переменные и адрес возврата, помещается в стек. Стек программы обеспечивает управление выполнением функций в порядке их вызова и позволяет программе возвращаться к предыдущим состояниям после завершения работы.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python? Чтобы проверить текущие параметры лимита, нужно запустить: sys.getrecursionlimit().

5. Что произойдет если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python? Существует предел глубины возможной рекурсии, который зависит от реализации Python. Когда предел достигнут, возникает исключение RecursionError: RuntimeError: Maximum Recursion Depth Exceeded. В Python 3.5 ошибка стала называться RecursionError, которая является производной от RuntimeError.

6. Как изменить максимальную глубину рекурсии в языке Python? Можно изменить предел глубины рекурсии с помощью вызова: sys.setrecursionlimit(limit).

7. Каково назначение декоратора lru_cache? Декоратор lru_cache используется для кэширования результатов вызовов функций. "LRU" расшифровывается как «Least Recently Used». Когда функция вызывается с определенными аргументами, результат вызова сохраняется в кэше. Если функция вызывается с теми же аргументами впоследствии, результат вызова извлекается из кэша вместо того, чтобы вычисляться заново, что может ускорить выполнение программы.

8. 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. Подобный вид рекурсии примечателен тем, что может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода 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.

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