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 №2.9
дисциплины «Основы программной инженерии»

Выполнил:
Баратов Семен Григорьевич
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Преподаватель:
Воронкин Р.А., канд. тех. 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.

Результаты выполнения

1. Создали репозиторий с лицензией MIT, добавили в .gitignore необходимые правила для работы с IDE PyCharm, клонировали репозиторий, организовали репозиторий в соответствии с моделью git-flow.

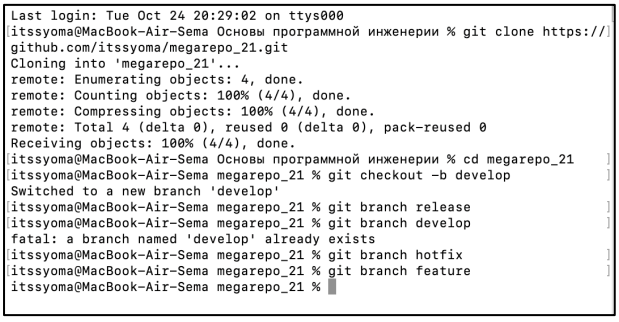
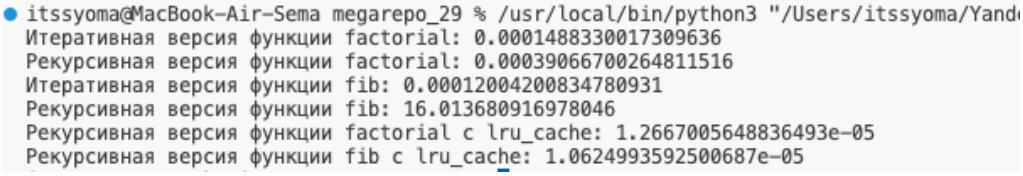


Рисунок 1 – Работа с репозиторием в командной строке.

2. Выполнили задание №1. 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


# 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)


# Рекурсивная версия функции factorial с использованием lru_cache
@lru_cache
def factorial_cached(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_cached(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 fib с использованием lru_cache
@lru_cache
def fib_cached(n):
    if n <= 1:
        return n
    else:
        return fib_cached(n-1) + fib_cached(n-2)


if __name__ == "__main__":
    # Оценка скорости работы итеративной версии функции factorial
    print("Итеративная версия функции factorial:", timeit.timeit('factorial_iterative(30)', globals=globals(), number=100))

    # Оценка скорости работы рекурсивной версии функции factorial
    print("Рекурсивная версия функции factorial:", timeit.timeit('factorial_recursive(30)', globals=globals(), number=100))

    # Оценка скорости работы итеративной версии функции fib
    print("Итеративная версия функции fib:", timeit.timeit('fib_iterative(30)', globals=globals(), number=100))

    # Оценка скорости работы рекурсивной версии функции fib
    print("Рекурсивная версия функции fib:", timeit.timeit('fib_recursive(30)', globals=globals(), number=100))

    # Оценка скорости работы рекурсивной версии функции factorial с lru_cache
    print("Рекурсивная версия функции factorial с lru_cache:", timeit.timeit('factorial_cached(30)', globals=globals(), number=100))

    # Оценка скорости работы рекурсивной версии функции fib с lru_cache
    print("Рекурсивная версия функции fib с lru_cache:", timeit.timeit('fib_cached(30)', globals=globals(), number=100))
```

Рисунок 2 – Код программы.



```
itssyoma@MacBook-Air-Sema megarepo_29 % /usr/local/bin/python3 "/Users/itssyoma/Yand
Итеративная версия функции factorial: 0.0001488330017309636
Рекурсивная версия функции factorial: 0.00039066700264811516
Итеративная версия функции fib: 0.00012004200834780931
Рекурсивная версия функции fib: 16.013680916978046
Рекурсивная версия функции factorial с lru_cache: 1.2667005648836493e-05
Рекурсивная версия функции fib с lru_cache: 1.0624993592500687e-05
```

Рисунок 3 – Результат выполнения программы.

Итеративные версии функций factorial и fib работают быстрее, чем их рекурсивные версии из-за отсутствия накладных расходов на вызов функций.

Использование декоратора lru_cache для рекурсивных функций factorial и fib значительно улучшает их производительность за счет кэширования результатов предыдущих вызовов.

3. Выполнили задание №2. Самостоятельно проработайте пример с оптимизацией хвостовых вызовов в Python. С помощью пакета timeit оцените скорость работы функций factorial и fib с использованием интроспекции стека и без использования интроспекции стека. Приведите полученные результаты в отчет.

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

import sys, timeit

class TailRecurseException(Exception):
    def __init__(self, args, kwargs):
        self.args = args
        self.kwargs = kwargs


def tail_call_optimized(g):
    """
    Эта программа показывает работу декоратора, который производит оптимизацию
    хвостового вызова. Он делает это, вызывая исключение, если оно является его
    прародителем, и перехватывает исключения, чтобы подделать оптимизацию хвоста.

    Эта функция не работает, если функция декоратора не использует хвостовой вызов.
    """

    def func(*args, **kwargs):
        f = sys._getframe()
        if f.f_back and f.f_back.f_back and f.f_back.f_back.f_code == f.f_code:
            raise TailRecurseException(args, kwargs)
        else:
            while 1:
                try:
                    return g(*args, **kwargs)
                except TailRecurseException as e:
                    args = e.args
                    kwargs = e.kwargs
    func.__doc__ = g.__doc__
    return func


# Функция factorial
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n-1)


# Функция factorial для оптимизации хвостовых вызовов
@tail_call_optimized
def factorial_tail_recursive(n, acc=1):
    if n == 0:
        return acc
    else:
        return factorial_tail_recursive(n-1, acc*n)


# Функция fib
def fib_recursive(n):
    if n <= 1:
        return n
    else:
        return fib_recursive(n-1) + fib_recursive(n-2)


# Функция fib для оптимизации хвостовых вызовов
@tail_call_optimized
def fib_tail_recursive(n, a=0, b=1):
    if n == 0:
        return a
    else:
        return fib_tail_recursive(n-1, b, a+b)


if __name__ == "__main__":

    # Оценка скорости работы функции factorial с использованием интроспекции стека
    print("Функция factorial без использования интроспекции стека:", timeit.timeit('factorial_recursive(30)', globals=globals(), number=10))

    # Оценка скорости работы функции factorial без использования интроспекции стека
    print("Функция factorial с использованием интроспекции стека:", timeit.timeit('factorial_tail_recursive(30)', globals=globals(), number=10))

    # Оценка скорости работы функции fib с использованием интроспекции стека
    print("Функция fib без использования интроспекции стека:", timeit.timeit('fib_recursive(30)', globals=globals(), number=10))

    # Оценка скорости работы функции fib без использования интроспекции стека
    print("Функция fib с использованием интроспекции стека:", timeit.timeit('fib_tail_recursive(30)', globals=globals(), number=10))
```

Рисунок 4 – Код программы.

```
itssyoma@MacBook-Air-Sema megarepo_29 % /usr/local/bin/python3 "/Users/itssyoma/
Функция factorial без использования интроспекции стека: 5.750000127591193e-05
Функция factorial с использованием интроспекции стека: 0.00041491599404253066
Функция fib без использования интроспекции стека: 1.648254875006387
Функция fib с использованием интроспекции стека: 0.000203417002921924
```

Рисунок 5 – Результат выполнения программы.

4. Выполнили индивидуальное задание (вариант 1). Напишите рекурсивную функцию, проверяющую правильность расстановки скобок в строке. При правильной расстановке выполняются условия:

количество открывающих и закрывающих скобок равно.

внутри любой пары открывающая – соответствующая закрывающая скобка, скобки расставлены правильно.

Примеры неправильной расстановки:)(, ())(, ())(() и т. п.

Рисунок 6 – Код и результат выполнения программы.

Ответы на контрольные вопросы

1. Для чего нужна рекурсия?

Рекурсия используется в программировании для решения задач, которые могут быть удобно разбиты на более мелкие подзадачи того же типа. Она позволяет вызывать функцию из самой себя, что упрощает решение некоторых задач.

2. Что называется базой рекурсии?

База рекурсии - это условие, при котором рекурсивная функция прекращает свою работу и начинает возвращать значения. Это условие обычно проверяется в начале функции и предотвращает бесконечную рекурсию.

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

Стек программы - это структура данных, используемая для хранения информации о вызовах функций во время выполнения программы. Каждый раз, когда функция вызывается, информация о ее состоянии (аргументы, локальные переменные и адрес возврата) помещается в стек. При завершении функции эта информация удаляется из стека.

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

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

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

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

6. Как изменить максимальную глубину рекурсии в языке Python?

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 использовании.

7. Каково назначение декоратора lru_cache?

Декоратор lru_cache используется для кэширования результатов вызовов функции с использованием алгоритма "наименее недавно используемый" (LRU - least recently used). Это позволяет избежать повторных вычислений при повторных вызовах функции с теми же аргументами.

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

Хвостовая рекурсия - это форма рекурсии, при которой рекурсивный вызов является последней операцией в функции. Оптимизация хвостовых вызовов позволяет компилятору заменить рекурсивные вызовы на циклы, что

уменьшает использование стека и позволяет избежать проблем с переполнением стека при большой глубине рекурсии.