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 «Программирование на Python»
Вариант №15

Выполнил:
Кенесбаев Хилол Куат улы
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., 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 принаписании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.

2. Выполнил задание: 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.

Код программы:

```
#!/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)
# Итеративная версия функции fibonacci
def fib_iterative(n):
a, b = 0, 1
for _ in range(n):
a, b = b, a + b
return a
```

```
# Рекурсивная версия функции fibonacci
def fib_recursive(n):
if n <= 1:
return n
else:
return fib_recursive(n-1) + fib_recursive(n-2)
# Рекурсивная версия функции factorial с использованием lru_cache
@lru_cache
def factorial_recursive_lru(n):
if n == 0 or n == 1:
return 1
else:
return n * factorial_recursive_lru(n-1)
# Рекурсивная версия функции fibonacci с использованием lru_cache
@lru_cache
def fib_recursive_lru(n):
if n <= 1:
return n else:
return fib_recursive_lru(n-1) + fib_recursive_lru(n-2)
if__name___== '__main__':
# Оценка скорости работы итеративной и рекурсивной версий функций
print("Factorial итеративный:", timeit.timeit(lambda: factorial_iterative(10),
number=100000))
print("Factorial рекурсивный:", timeit.timeit(lambda: factorial_recursive(10),
number=100000))
print("Fibonacci итеративный:", timeit.timeit(lambda: fib_iterative(10),
number=100000))
print("Fibonacci рекурсивный:", timeit.timeit(lambda: fib_recursive(10),
number=100000))
# Оценка скорости работы рекурсивных версий функций с использованием
lru_cache
print("Factorial рекурсивный с @lru_cache:", timeit.timeit(lambda:
factorial_recursive_lru(10), number=100000))
print("Fibonacci рекурсивный с @lru_cache:", timeit.timeit(lambda:
fib_recursive_lru(10), number=100000))
```

```
C:\Users\HP-PC\AppData\Local\Programs\Python\Python312\python.exe C:\Users\HP-PC\PycharmProjects\lab-12\Lab2.9\prog\task1.py
Factorial итеративный: 0.03561200000149256
Factorial рекурсивный: 0.07027869999910763
Fibonacci итеративный: 0.039218500000060885
Fibonacci рекурсивный: 0.7842884999990929
Factorial рекурсивный с @lru_cache: 0.006867900001452654
Fibonacci рекурсивный с @lru_cache: 0.006534199999805423

Process finished with exit code 0
```

Рисунок 1 Вывод программы task1

3. Выполнил индивидуальное задание вариант 10:

Условие задания:

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.

Код программы:

```
def check_brackets(string, idx=0, open_brackets=0):
    # Базовый случай: достигнут конец строки
    if idx == len(string):
        # Если все скобки сбалансированы, количество открывающих и закрывающих скобок равно
        return open_brackets == 0
    # Если открывающая скобка, увеличиваем счетчик открытых скобок
    if string[idx] == '(':
        return check_brackets(string, idx + 1, open_brackets + 1)
    # Если закрывающая скобка, уменьшаем счетчик открытых скобок
    elif string[idx] == ')':
        # Если закрывающей скобки не может быть без соответствующей открывающей
        if open_brackets == 0:
            return False
        return check_brackets(string, idx + 1, open_brackets - 1)
    # Пропускаем другие символы
    return check_brackets(string, idx + 1, open_brackets)

# Примеры использования
print(check_brackets("((()))"))
print(check_brackets("(()))"))
print(check_brackets("())("))
```

```
C:\Users\HP-PC\AppData\Local\Programs\Python\Python312\python.exe C:\Users\HP-PC\Desktop\study\python\пересдача\Lab2.9\prog\ind.py
True
False
False
```

Рисунок 2 Результат работы программы

Ответы на контрольные 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 рекурсиив 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

RuntimeError :

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.