МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №2.9 по дисциплине основы программной инженерии

Выполнил:
Кожухов Филипп Денисович,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной
безопасности,Воронкин Р.А.

Отчет защищен с оценкой_____Дата защиты_____

Ставрополь, 2021 г.

ВЫПОЛНЕНИЕ:

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


def recursion(n):
    if n == 1:
        return 1
    return n + recursion(n - 1)


if __name__ == '__main__':
    n = int(input("Enter n: "))
    summary = 0
    for i in range(1, n + 1):
        summary += i
    print(f"Iterative sum: {summary}")
    print(f"Recursion sum: {recursion(n)}")
```

Рисунок 1 - Код задания №1

```
Enter n: 462
Iterative sum: 106953
Recursion sum: 106953

Process finished with exit code 0
```

Рисунок 2 - Вывод программы

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

import timeit
import sys


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


def tail_call_optimized(g):
    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 True:
                try:
                    return g(*args, **kwargs)
                except TailRecurseException as e:
                    args = e.args
                    kwargs = e.kwargs

    func.__doc__ = g.__doc__
    return func


@tail_call_optimized
def factorial_rec(n, acc=1):
    if n == 0:
        return acc
    return factorial_rec(n - 1, n * acc)


@tail_call_optimized
def fib_rec(i, current=0, next=1):
    if i == 0:
        return current
    else:
        return fib_rec(i - 1, next, current + next)
```

```python
def factorial_iter(n):
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact


def fib_iter(n):
    a = 0
    b = 1
    for i in range(n):
        c = a + b
        a = b
        b = c
    return a


if __name__ == '__main__':
    number = int(input("Enter the number: "))

    start_time = timeit.default_timer()
    factorial_rec(number)
    print("Recursive factorial time is: ",
          timeit.default_timer() - start_time
          )

    start_time = timeit.default_timer()
    factorial_iter(number)
    print("Iterative factorial time is :",
          timeit.default_timer() - start_time
          )

    start_time = timeit.default_timer()
    fib_rec(number)
    print("Recursive Fibonacci time is :",
          timeit.default_timer() - start_time
          )

    start_time = timeit.default_timer()
    fib_iter(number)
    print("Iterative Fibonacci time is :",
          timeit.default_timer() - start_time
          )
```

Рисунок 3 - Код задания №2

```
Enter the number: 42
Recursive factorial time is:  0.00016028699999992568
Iterative factorial time is : 1.2633000000317196e-05
Recursive Fibonacci time is : 0.00014331099999997932
Iterative Fibonacci time is : 7.1059999999789625e-06

Process finished with exit code 0
```

Рисунок 4 - Вывод программы

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Вариант 5

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

def subset(my_set, check=None):
    if check is None:
        check = []
    if len(my_set) != 0 and my_set not in check:
        check.append(my_set)
        print(set(my_set))
        for i, elem in enumerate(my_set):
            subset(my_set[0:i] + my_set[i + 1:len(my_set)], check)


if __name__ == '__main__':
    start_set = {int(i) for i in input("Enter the set: ").split()}
    list_set = list(start_set)
    subset(list_set)
```

Рисунок 5 - Код индивидуального задания

```
Enter the set: 1 2 4
{1, 2, 4}
{2, 4}
{4}
{2}
{1, 4}
{1}
{1, 2}

Process finished with exit code 0
```

Рисунок 6 - Вывод программы

ОТВЕТЫ НА ВОПРОСЫ

1. Для чего нужна рекурсия?
Рекурсия может работать в обратную сторону, иногда рекурсивное решение проще, чем итеративное решение. Это очевидно при реализации обращения связанного списка.

2. Что называется базой рекурсии?
База рекурсии – это такие аргументы функции, которые делают задачу настолько простой, что решение не требует дальнейших вложенных вызовов.

3. Самостоятельно изучите что является стеком программы. Как используется стек программы при вызове функций?
Стек функции – в теории вычислительных систем, LIFO-стек, хранящий информацию для возврата управления из подпрограмм (процедур, функций) в программу (или подпрограмму, при вложенных или рекурсивных вызовах) и/или для возврата в программу из обработчика прерывания (в том числе при переключении задач в многозадачной среде).
Когда функция производит вложенный вызов, происходит следующее:
1) Выполнение текущей функции приостанавливается.
2) Контекст выполнения, связанный с ней, запоминается в специальной структуре данных – стеке контекстов выполнения.
3) Выполняются вложенные вызовы, для каждого из которых создаётся свой контекст выполнения.
4) После их завершения старый контекст достаётся из стека, и выполнение внешней функции возобновляется с того места, где она была остановлена.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python?
Выполнить команду sys.getrecursionlimit()

5. Что произойдет если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python?
Когда предел достигнут, возникает исключение RuntimeError : RuntimeError: Maximum Recursion Depth

6. Как изменить максимальную глубину рекурсии в языке Python?
Нужно выполнить команду: sys.setrecursionlimit(limit)

7. Каково назначение декоратора lru_cache ?
Декоратор @lru_cache() модуля functools оборачивает функцию с переданными в нее аргументами и запоминает возвращаемый результат соответствующий этим аргументам. Такое поведение может сэкономить время и ресурсы, когда дорогая или связанная с вводом/выводом функция периодически вызывается с одинаковыми аргументами.

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. Оптимизация хвостового вызова (TCO) — это способ автоматического сокращения рекурсии в рекурсивных функциях. Для оптимизации, к примеру, можно постараться каждый раз при вызове функции сохранять только кадр стека внутренней вызываемой функции, что может значительно сэкономить память.