МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Лабораторная работа 2.9

Рекурсия в языке Python

Выполнил студент группы ИВТ-б-о-20-1

Симанский М.Ю « »_____20__г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

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

Пример 1

Пример обратного отсчета, написанного с использованием хвостовой рекурсии.

Код

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
def countdown(n):
    if n == 0:
        print("Blastoff!")
    else:
        print(n)
        countdown(n-1)
```

Пример 2

Любое вычисление, которое может быть выполнено с использованием итерации, также может быть выполнено с использованием рекурсии. Вот версия find_max (поиск максимального значения в последовательном контейнере, например списке или кортеже), написанная с использованием хвостовой рекурсии**.**

Код

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
def find_max(seq, max_so_far):
    if not seq:
        return max_so_far
    if max_so_far < seq[0]:
        return find_max(seq[1:], seq[0])
    else:
        return find_max(seq[1:], max_so_far)
```

Пример 3

Хвостовую рекурсию лучше не использовать, поскольку компилятор Python не обрабатывает оптимизацию для хвостовых рекурсивных вызовов. В таких случаях рекурсивное решение использует больше системных ресурсов, чем итеративное.

По умолчанию рекурсивный стек Python не превышает 1000 кадров. Это ограничение можно изменить, установив sys.setrecursionlimit(15000) который быстрее, однако этот метод потребляет больше памяти.

Код

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Эта программа показыает работу декоратора, который производит оптимизацию
# хвостового вызова. Он делает это, вызывая исключение, если оно является его
# прародителем, и перехватывает исключения, чтобы вызвать стек.
import sys

class TailRecurseException:
    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
```

Задание 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import timeit
code1 ='''
```

```
def factorial(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n * factorial(n - 1)
'''
code2 = '''
def fib(n):
    if n == 0 or n == 1:
        return n

    else:
        return fib(n - 2) + fib(n - 1)
'''
print('Результат рекурсивного факториала:', timeit.timeit(setup =
code1,number = 1000))
print('Результат рекурсивного числа Фибоначи:', timeit.timeit(setup =
code2,number = 1000))
code3 ='''
def factorial(n):
    product = 1
    while n > 1:
        product *= n
        n -= 1
    return product
'''
code4 = '''
def fib(n):
    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
    return a
'''
print('Результат итеративного факториала:', timeit.timeit(setup =
code3,number = 1000))
print('Результат итеративного числа Фибоначи:', timeit.timeit(setup =
code4,number = 1000))
code5 ='''
from functools import lru_cache
@lru_cache
def factorial(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n * factorial(n - 1)
'''
code6 = '''
from functools import lru_cache
@lru_cache
def fib(n):
    if n == 0 or n == 1:
        return n

    else:
        return fib(n - 2) + fib(n - 1)
'''
print('Результат факториала с декоратором:', timeit.timeit(setup =
code5,number = 1000))
```

```
print('Результат числа Фибоначи с декоратором:', timeit.timeit(setup =
code6,number = 1000))
```

Результат выполнения

```
Результат рекурсивного факториала: 1.2899999999996248e-05
Результат рекурсивного числа Фибоначи: 1.130000000000575e-05
Результат итеративного факториала: 1.1200000000002874e-05
Результат итеративного числа Фибоначи: 1.1200000000002874e-05
Результат факториала с декоратором: 1.2200000000003874e-05
Результат числа Фибоначи с декоратором: 1.2200000000003874e-05

Process finished with exit code 0
```

Рисунок 1 – Программа выполнена

Задание 2

Индивидуальное задание. Вариант 18(4)

Создайте рекурсивную функцию, печатающую все возможные перестановки для целых чисел от 1 до.

Код

```
def all_var(arr):
    if len(arr) == 1:
        return [arr]  # терминальная ветвь
    else:
        a = arr[0]  # первый элемент списка
        p = all_var(arr[1:])  # все перестановки хвоста
        r = []  # вставляем a в каждую возможную позицию каждой
        for pp in p:  # перестановки хвоста
            for i in range(len(pp)):
                tmp = pp[0 : i] + [a] + pp[i:]
                r.append(tmp)
            r.append(pp + [a])
        return r
n = int(input("Введите число"))
print(all_var([i for i in range(1, n + 1)]))
```

Результат

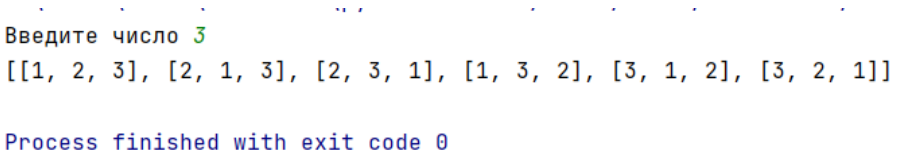

Рисунок 2 – Результат выполнения программы

Ответы на вопросы

1. Функция может содержать вызов других функций. В том числе процедура может вызвать саму себя. Никакого парадокса здесь нет – компьютер лишь последовательно выполняет встретившиеся ему в программе команды и, если встречается вызов процедуры, просто начинает выполнять эту функцию. Без разницы, какая функция дала команду это делать.

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

3. Стек в Python — это линейная структура данных «последним вошел — первым ушел», т.е. элемент, введенный последним, будет первым удаляемым элементом.

4. Чтобы проверить текущие параметры лимита, нужно запустить: sys.getrecursionlimit()

5. Существует предел глубины возможной рекурсии, который зависит от реализации Python. Когда

предел достигнут, возникает исключение RuntimeError

6. sys.setrecursionlimit(число)

7. полезным инструментом является декоратор lru_cache , который можно использовать для уменьшения количества лишних вычислений.

8. 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.