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:
Шальнев Владимир Сергеевич,
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:

```
F:\pythonProject>git clone https://github.com/HAXF13D/laboratory-12/
Cloning into 'laboratory-12'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 11 (delta 1), reused 0 (delta 0), pack-reused 5
Unpacking objects: 100% (11/11), done.

F:\pythonProject>cd laboratory-12/
```

Клонирование репозитория

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

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


def main():
    n = int(input("Введите n = "))
    summa = 0
    for i in range(1, n + 1):
        summa += i
    print(f"Сумма посчитаная без рекурсии = {summa}")
    print(f"Сумма посчитанная с помощью рекурсии = {recursion(n)}")


if __name__ == '__main__':
    main()
```

Пример 1

```
Введите n = 4
Сумма посчитаная без рекурсии = 10
Сумма посчитанная с помощью рекурсии = 10
```

Значение №1

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

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)


def main():
    n = int(input("Введите n = "))
    print(f"Вычисление {n} числа Фибоначи с помощью рекурсии = {fib(n)}")


if __name__ == '__main__':
    main()
```

Пример 2

```
Введите n = 6
Вычисление 6 числа Фибоначи с помощью рекурсии = 8
```

Значение №1

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

def cursing(depth):
    try:
        cursing(depth + 1)  # actually, re-cursing
    except RuntimeError as RE:
        print('I recursed {} times!'.format(depth))


if __name__ == '__main__':
    cursing(0)
```

Пример 3

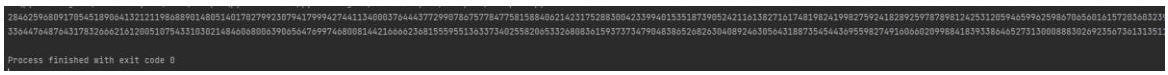
```
I recursed 998 times!

Process finished with exit code 0
```

Значение №1

```python
#!/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(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(n, acc=1):
    """calculate a factorial"""
    if n == 0:
        return acc
    return factorial(n - 1, n * acc)


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


if __name__ == '__main__':
    print(factorial(10000))
    print(fib(10000))
```

Пример 4



Значение №1

```
F:\pythonProject\laboratory-12>git status
On branch develop
Your branch is up to date with 'origin/develop'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        examples/

nothing added to commit but untracked files present (use "git add" to track)

F:\pythonProject\laboratory-12>git add .

F:\pythonProject\laboratory-12>git commit -m "Examples"
[develop 023b2d9] Examples
 4 files changed, 103 insertions(+)
 create mode 100644 examples/example1.py
 create mode 100644 examples/example2.py
 create mode 100644 examples/example3.py
 create mode 100644 examples/example4.py
```

Коммит изменений

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

import timeit


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


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


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


def main():

    number = 250

    start_time = timeit.default_timer()
```

```
    factorial_rec(number)
    print("Execution factorial_rec time is :", timeit.default_timer() -
start_time)

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

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

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


if __name__ == '__main__':
    main()
```

Задача 1

```
Execution factorial_rec time is : 0.0003475319999999976
Execution factorial_iter time is : 2.5891999999999582e-05
Execution fib_rec time is : 4.5219999999998595e-05
Execution fib_iter time is : 1.3492999999996369e-05
```

Значение при отсутствии lru_cache

```
Execution factorial_rec time is : 0.0004930370000000017
Execution factorial_iter time is : 2.625999999999534e-05
Execution fib_rec time is : 9.919099999999847e-05
Execution fib_iter time is : 1.3856999999999065e-05
```

Значение при lru_cache

```
#!/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)


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


def main():

    number = 250

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

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

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

    start_time = timeit.default_timer()
```

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


if __name__ == '__main__':
    main()
```

Задача 2

```
Execution factorial_rec time is : 0.0003475319999999976
Execution factorial_iter time is : 2.5891999999999582e-05
Execution fib_rec time is : 4.5219999999998595e-05
Execution fib_iter time is : 1.3492999999996369e-05
```

Значение при отсутствии tail_call_optimized

```
Execution factorial_rec time is : 0.0003963979999999957
Execution factorial_iter time is : 2.5161999999995244e-05
Execution fib_rec time is : 0.0003664950000000014
Execution fib_iter time is : 1.3856999999999065e-05
```

Значение при tail_call_optimized

```
F:\pythonProject\laboratory-12>git add .

F:\pythonProject\laboratory-12>git commit -m "Version with tail_call_optimized"
[develop 886f529] Version with tail_call_optimized
 1 file changed, 87 insertions(+)
 create mode 100644 tasks/task2.py
```

Коммит изменений

```
F:\pythonProject\laboratory-12>git add .

F:\pythonProject\laboratory-12>git commit -m "Version with lru_cache and without tail_call_optimized"
[develop 1ad0e0a] Version with lru_cache and without tail_call_optimized
 1 file changed, 4 insertions(+), 1 deletion(-)
```

Коммит изменений

```
F:\pythonProject\laboratory-12>git add .

F:\pythonProject\laboratory-12>git commit -m "Version without lru_cache and tail_call_optimized"
[develop 8031516] Version without lru_cache and tail_call_optimized
 1 file changed, 61 insertions(+)
 create mode 100644 tasks/task_1.py
```

Коммит изменений

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

import sys
```

```python
def rec_func(a, i=0, k=0):
    if len(a) == k:
        print("В списке нет отрицательных чисел", file=sys.stderr)
        exit(0)
    elif a[i] < 0:
        return 0, k
    else:
        return a[i] + rec_func(a, i + 1, k + 1)[0], rec_func(a, i + 1, k +
1)[1]


def main():
    array = list(map(int, input("Введите список:\n").split()))
    summa, amount = rec_func(array)
    print(f"Сумма чисел = {summa}")
    print(f"Количество чисел = {amount}")


if __name__ == '__main__':
    main()
```

Индивидуальное задание 1

```
Введите список:
1 2 3 -1
Сумма чисел = 6
Количество чисел = 3
```

Значение №1

```
Введите список:
1 2 3 4
В списке нет отрицательных чисел
```

Значение №2

```
F:\pythonProject\laboratory-12>git status
On branch develop
Your branch is ahead of 'origin/develop' by 4 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   tasks/indiv.py
        renamed:    tasks/task2.py -> tasks/task_2.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   tasks/indiv.py


F:\pythonProject\laboratory-12>git add .

F:\pythonProject\laboratory-12>git commit -m "some changes + indiv task"
[develop 0ff8ba8] some changes + indiv task
 2 files changed, 25 insertions(+)
 create mode 100644 tasks/indiv.py
 rename tasks/{task2.py => task_2.py} (100%)
```

Коммит изменений

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:

2. **Что называется базой рекурсии?**

Случай, при котором мы не запускаем в рекурсию, к примеру, во время вычисления факториала базовый случай – это if n == 0 or n == 1: return 1

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

Стек вызовов (от англ. call stack; применительно к процессорам — просто «стек») — 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).

При вызове подпрограммы или возникновении прерывания, в стек заносится адрес возврата — адрес в памяти следующей инструкции приостановленной программы и управление передается подпрограмме или подпрограмме-обработчику. При последующем вложенном или рекурсивном вызове, прерывании подпрограммы или обработчика прерывания, в стек заносится очередной адрес возврата и т. д.

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

```
import sys
print(sys.getrecursionlimit())
```

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

Возникает исключение RuntimeError :

RuntimeError: Maximum Recursion Depth Exceeded

6. **Как изменить максимальную глубину рекурсии в языке Python?**

```
sys.setrecursionlimit(1500)
```

7. **Каково назначение декоратора lru_cache?**

Он 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 хвостовой рекурсии путём преобразования её в плоскую итерацию реализована во многих оптимизирующих компиляторах. В некоторых функциональных языках программирования спецификация

гарантирует обязательную оптимизацию хвостовой рекурсии. Типовой механизм реализации вызова функции основан на сохранении адреса возврата, параметров и локальных переменных функции в стеке и выглядит следующим образом:

1. В точке вызова в стек помещаются параметры, передаваемые функции, и адрес возврата.
2. Вызываемая функция в ходе работы размещает в стеке собственные локальные переменные.
3. По завершении вычислений функция очищает стек от своих локальных переменных, записывает результат (обычно — в один из регистров процессора).
4. Команда возврата из функции считывает из стека адрес возврата и выполняет переход по этому адресу. Либо непосредственно перед, либо сразу после возврата из функции стек очищается от параметров.