

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**
**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе №12

Дисциплина: «Программирование на Python»

Тема: «Рекурсия в языке Python»

Выполнил: студент 2 курса

группы ИВТ-б-о-21-1

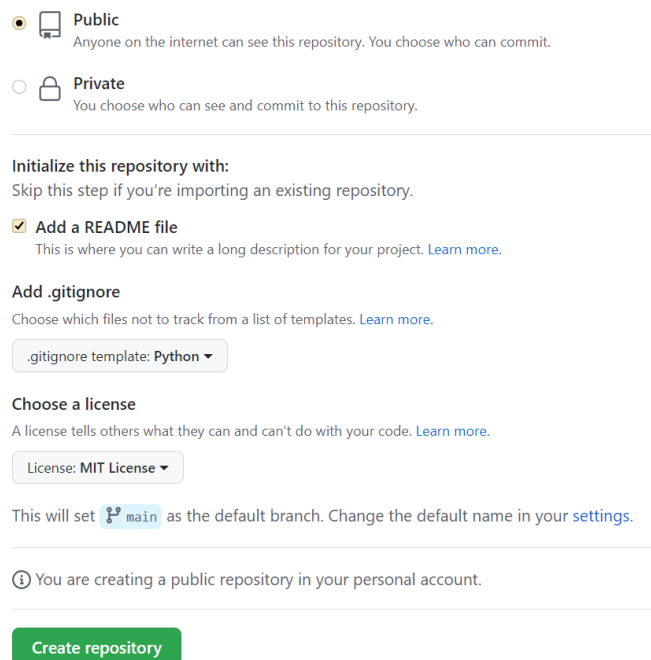
Уланбекова Айканыш Уланбековна

Ставрополь 2022

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

Выполнение работы:

1. Создала репозиторий в GitHub «rep 2.6» в который добавила .gitignore, который дополнила правила для работы с IDE PyCharm с ЯП Python, выбрала лицензию MIT, клонировала его на лок. сервер и организовала в соответствии с моделью ветвления git-flow.



☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **Python** ▼

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: **MIT License** ▼

This will set **main** as the default branch. Change the default name in your [settings](#).

(i) You are creating a public repository in your personal account.

Create repository

Рисунок 1.1 Создание репозитория

```
C:\Users\User>cd C:\Users\User\Desktop\2 кypc Python\lab 12
C:\Users\User\Desktop\2 кypc Python\lab 12>git clone https://github.com/aikanyshkauanbekova/lab12.git
Cloning into 'lab12'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 4.50 KiB | 1.50 MiB/s, done.
Resolving deltas: 100% (1/1), done.
C:\Users\User\Desktop\2 кypc Python\lab 12>
```

Рисунок 1.2 Клонирование репозитория

```

C:\Users\User\Desktop\2 кypc Python\lab 10\lab-10>git flow init
which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/User/Desktop/2 кypc Python/lab 10/lab-10/.git/hooks]
C:\Users\User\Desktop\2 кypc Python\lab 10\lab-10>

```

Рисунок 1.3 Организация репозитория в соответствии с моделью ветвления git-flow



```

.gitignore – Блокнот
Файл Правка Формат Вид Справка
.idea/
# Created by https://www.toptal.com/developers/gitignore/api/python,pycharm
# Edit at https://www.toptal.com/developers/gitignore?templates=python,pycharm

### PyCharm ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio,
# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839

# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
.idea/**/dictionaries
.idea/**/shelf

# AWS User-specific
.idea/**/aws.xml

```

Рисунок 1.4 Изменение .gitignore

2. Создал проект PyCharm в папке репозитория, проработал примеры ЛР.

```

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

# Пример обратного отсчета, написанного с использованием хвостовой рекурсии

def countdown(n):
    if n == 0:
        print("Blastoff!")
    else:
        print(n)
        countdown(n-1)

```

Рисунок 2.1 Пример 1

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

"""
Любое вычисление, которое может быть выполнено с использованием итерации,
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:
"""

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)
```

Рисунок 2.2 Пример 2

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

# Эта программа показывает 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):
    """
    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)
```

Рисунок 2.3 Пример 3

3. Выполнил задания.

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 изменится скорость работы рекурсивных версий функций factorial и fib при

использовании декоратора lru_cache? Приведите в отчет и обоснуйте полученные результаты.

```
Результат рекурсивного факториала: 1.9899976905435324e-05
Результат рекурсивного числа Фибоначи: 1.9400031305849552e-05
Результат итеративного факториала: 1.92999723367393e-05
Результат итеративного числа Фибоначи: 1.9199971575289965e-05
Результат факториала с декоратором: 2.6799971237778664e-05
Результат числа Фибоначи с декоратором: 2.0000035874545574e-05

Process finished with exit code 0
```

Рисунок 3.1 Вывод программы задания

Быстрее вычислять с декоратором, так как функция используется внутри него меняясь вне его.

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

```
Результат факториала: 1.9300030544400215e-05
Результат числа Фибоначи: 1.839996548369527e-05
Результат факториала с интроспекцией стека: 1.880002673715353e-05
Результат числа Фибоначи с интроспекцией стека: 2.4200009647756815e-05

Process finished with exit code 0
```

Рисунок 3.2 Рез-т выполнения задания №2

Индивидуальное задание:

4. (9 вариант). Выполнил индивидуальное задание.

9. Даны целые числа m и n , где $0 \leq m \leq n$, вычислить, используя рекурсию, число сочетаний C_n^m по формуле: $C_n^0 = C_n^n = 1$, $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$ при $0 \leq m \leq n$.
Воспользовавшись формулой

$$C_n^m = \frac{n!}{m!(n-m)!} \quad (1)$$

можно проверить правильность результата.

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

def fum(m, n):
    if m == 0 or m == n:
        return 1
    if 0 <= m <= n:
        return fum(m, n - 1) + fum(m - 1, n - 1)

m = int(input())
n = int(input())

if __name__ == '__main__':
    print(fum(m, n))
```

Рисунок 4.1 Вывод программы индивидуального задания

5. Сделал коммит, выполнил слияние с веткой main, и запустил изменения в уд. репозиторий.

```
C:\Users\User>cd C:\Users\User\Desktop\2 кypc Python\lab 12\lab12
C:\Users\User\Desktop\2 кypc Python\lab 12\lab12>git add .
C:\Users\User\Desktop\2 кypc Python\lab 12\lab12>git commit -m "key"
[main daf2d73] key
6 files changed, 259 insertions(+)
create mode 100644 indiv.py
create mode 100644 primer1.py
create mode 100644 primer2.py
create mode 100644 primer3.py
create mode 100644 zadanie1.py
create mode 100644 zadanie2.py
C:\Users\User\Desktop\2 кypc Python\lab 12\lab12>git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 2.83 KiB | 1.42 MiB/s, done.
Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/aikanyshkauanbekova/lab12.git
6816ba0..daf2d73 main -> main
C:\Users\User\Desktop\2 кypc Python\lab 12\lab12>_
```

Рисунок 5.1 Сохранение

Контр. вопросы и ответы на них:

1. Для чего нужна рекурсия?

В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия). Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

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 — это структура данных, в которой элементы хранятся в порядке поступления.

Стек хранит последовательность данных. Связаны данные так: каждый элемент указывает на тот, который нужно использовать следующим. Это линейная связь — данные идут друг за другом и нужно брать их по очереди. Из середины стека брать нельзя.

Главный принцип работы стека — данные, которые попали в стек недавно, используются первыми. Чем раньше попал — тем позже используется. После использования элемент стека исчезает, и верхним становится следующий элемент.

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

Функция `sys.getrecursionlimit()` возвращает текущее значение предела рекурсии, максимальную глубину стека интерпретатора Python. Этот предел предотвращает бесконечную рекурсию от переполнения стека языка C и сбоя Python. Это значение может быть установлено с помощью `sys`.

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`.

6. Как изменить максимальную глубину рекурсии в языке Python?

С помощью `sys.setrecursionlimit(число)`.

7. Каково назначение декоратора `lru_cache`?

Функция `lru_cache` предназначена для мемоизации (предотвращения повторных вычислений), т. е. кэширует результат в памяти. Полезный инструмент, который уменьшает количество лишних вычислений.

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.

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.