

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

**Кафедра инфокоммуникаций
«Рекурсия в языке Python»**

**Отчет по лабораторной работе № 2.9
по дисциплине «Программирование на Python»**

Выполнил студент группы ИВТ-б-о-21-1

Шайдеров Дмитрий Викторович.

«4 » октября 2022г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

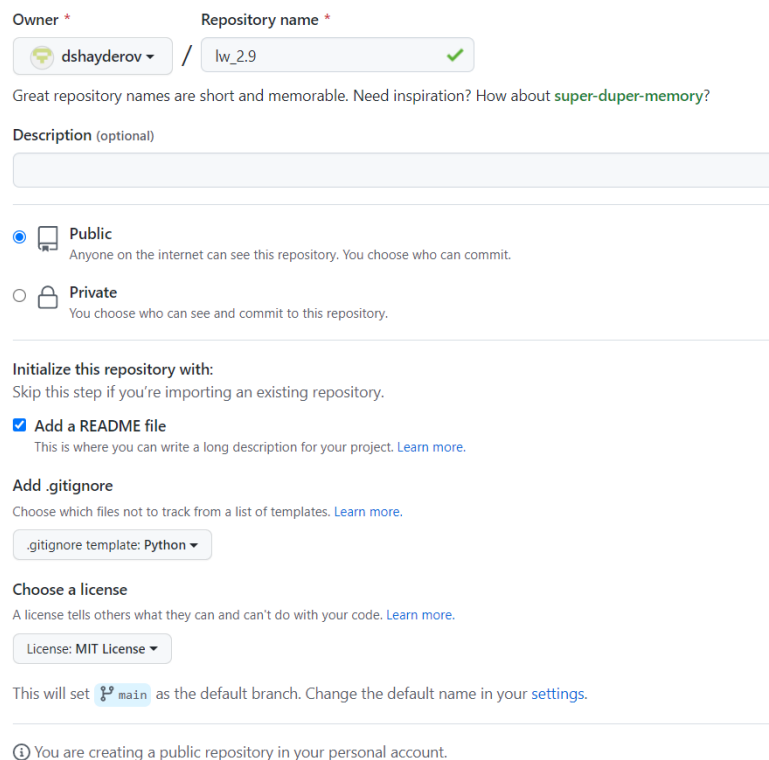
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2022

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

Порядок выполнения работы:

1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.



Owner * Repository name *

dshayderov / lw_2.9

Great repository names are short and memorable. Need inspiration? How about [super-duper-memory](#)?

Description (optional)

☒ 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](#).

① You are creating a public repository in your personal account.

Рисунок 1 - Создание репозитория

2. Выполните клонирование созданного репозитория.

```
Asus@LAPTOP-09A994CG MINGW64 /c/My projects/3/lw_2.8 (main)
$ cd "C:\My projects\3"

Asus@LAPTOP-09A994CG MINGW64 /c/My projects/3
$ git clone https://github.com/dshayderov/lw_2.9.git
Cloning into 'lw_2.9'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 11 (delta 2), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (11/11), done.
Resolving deltas: 100% (2/2), done.
```

Рисунок 2 - Клонирование репозитория

3. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

```

Asus@LAPTOP-09A994CG MINGW64 /c/My projects/3
$ cd "C:\My projects\3\lw_2.9"

Asus@LAPTOP-09A994CG MINGW64 /c/My projects/3/lw_2.9 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

Asus@LAPTOP-09A994CG MINGW64 /c/My projects/3/lw_2.9 (develop)
$ |

```

Рисунок 3 - Ветвление по модели git-flow

4. Создайте проект PyCharm в папке репозитория.

Этот компьютер > OS (C:) > My projects > 3 > lw_2.9 >				
Имя	Дата изменения	Тип	Размер	
Project	09.10.2022 23:36	Папка с файлами		
.gitignore	09.10.2022 23:34	Файл "GITIGNORE"	2 КБ	
LICENSE	09.10.2022 23:34	Файл	2 КБ	
README.md	09.10.2022 23:34	Файл "MD"	1 КБ	

Рисунок 4 - Проект PyCharm

5. Проработайте пример лабораторной работы. Создайте для него отдельный модуль языка Python. Зафиксируйте изменения в репозитории.

```

primer x
"C:\My projects\3\lw_2.9\Project\venv\Scripts\python.exe" "C:/My projects/3/lw_2.9/Project/primer.py"
28462596809170545189064132121198688901480514017027992307941799942744113400037644437729907867577847758158
Process finished with exit code 0

```

Рисунок 5 - Результат выполнения примера

6. 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.

```
zadanie_1 x
"C:\My projects\3\lw_2.9\Project\venv\Scripts\python.exe" "C:/My
Результат рекурсивного факториала: 8.59999999997499e-06
Результат рекурсивного числа Фибоначи: 6.3000000000007494e-06
Результат итеративного факториала: 6.200000000011752e-06
Результат итеративного числа Фибоначи: 6.3000000000007494e-06
Результат факториала с декоратором: 5.900000000003125e-06
Результат числа Фибоначи с декоратором: 5.900000000003125e-06
```

Рисунок 6 - Результат выполнения задания 1

При помощи декоратора `lru_cache` можно мемоизировать результат функции (запечатлеть результаты для конкретных наборов аргументов). Такое кеширование позволяет экономить время и ресурсы, если тяжёлая функция вызывается периодически с более или менее одинаковым набором аргументов.

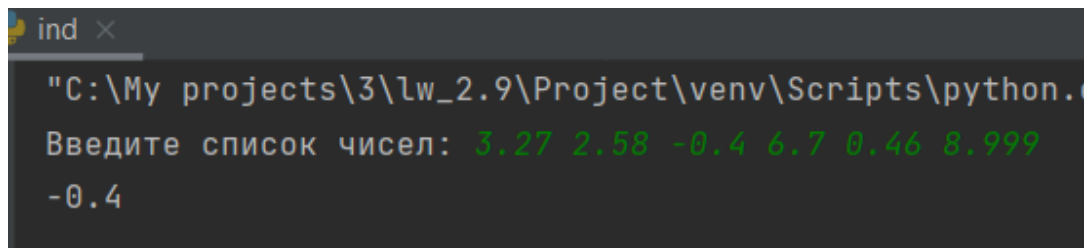
7. Самостоятельно проработайте пример с оптимизацией хвостовых вызовов в Python. С помощью пакета `timeit` оцените скорость работы функций `factorial` и `fib` с использованием интроспекции стека и без использования интроспекции стека. Приведите полученные результаты в отчет.

```
zadanie_2 x
"C:\My projects\3\lw_2.9\Project\venv\Scripts\python.exe" "C:/My project
Результат факториала: 7.39999999997686e-06
Результат числа Фибоначи: 6.200000000004813e-06
Результат факториала с интроспекцией стека: 7.900000000005125e-06
Результат числа Фибоначи с интроспекцией стека: 1.03000000000475e-05
```

Рисунок 7 - Результат выполнения задания 2

8. Решите индивидуальное задание. (Вариант 26/12)

Дан список X из n вещественных чисел. Найти минимальный элемент списка, используя вспомогательную рекурсивную функцию, находящую минимум среди последних элементов списка X , начиная с n -го.



```
ind x
"C:\My projects\3\lw_2.9\Project\venv\Scripts\python.
Введите список чисел: 3.27 2.58 -0.4 6.7 0.46 8.999
-0.4
```

Рисунок 8 - Результат выполнения индивидуального задания

Контрольные вопросы:

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.

Вывод: были приобретены навыки по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.