

**Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Институт цифрового развития
Кафедра инфокоммуникаций**

**ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2.9
дисциплины
«Программирование на Python»**

Выполнила студентка группы
ИВТ-б-о-21-1
Яковлева Е.А. « » _____ 20__ г.
Подпись студента _____
Работа защищена
« » _____ 20__ г.

Проверил доцент
Кафедры инфокоммуникаций,
старший преподаватель
Воронкин Р.А.

(подпись)

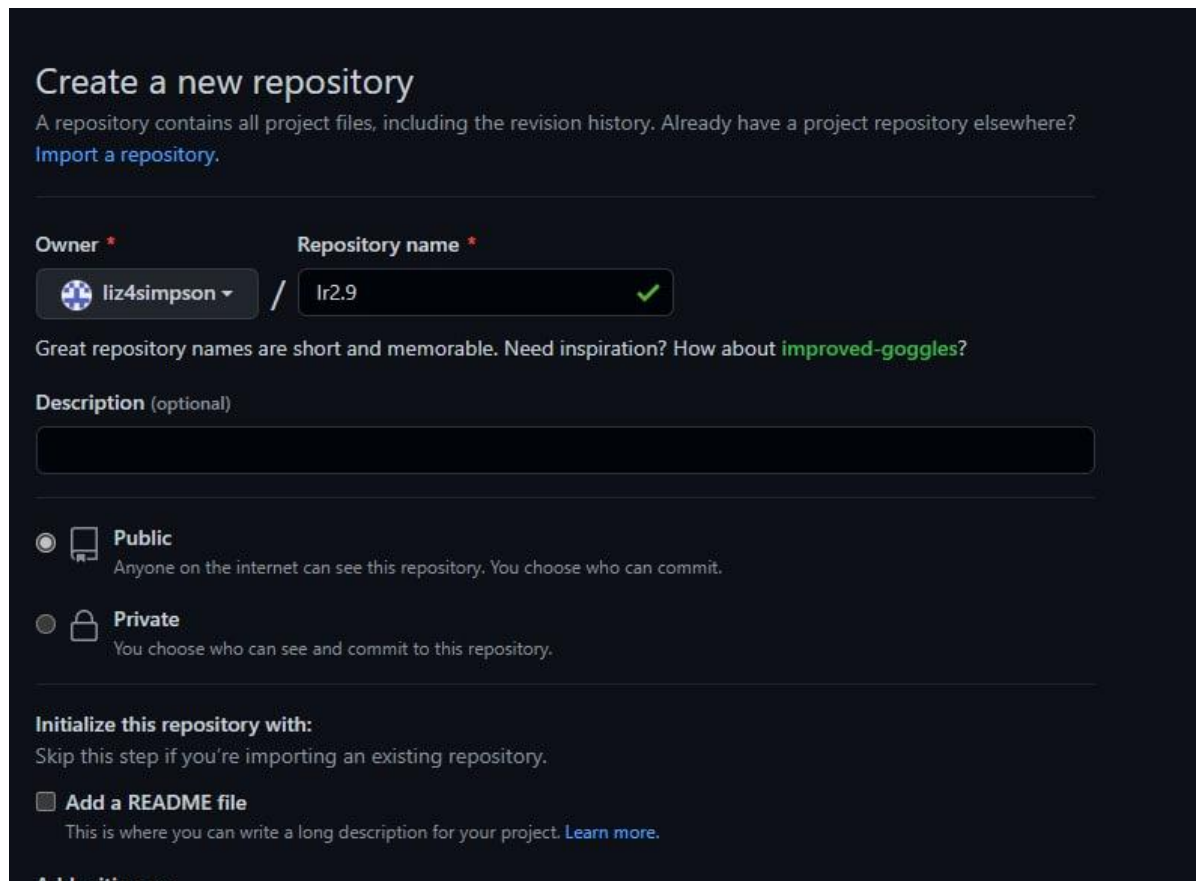
Ставрополь, 2022 г.

Тема: Работа с рекурсией в языке Python

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

Порядок выполнения работы:



Создала общедоступный репозиторий на GitHub, в котором использованы лицензия MIT и язык программирования Python.



Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Owner * **Repository name ***

 liz4simpson / lr2.9 

Great repository names are short and memorable. Need inspiration? How about [improved-goggles?](#)

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 citations](#)

Рисунок 1. Создание репозитория

Выполнила клонирование созданного репозитория.

```
C:\Users\Elizaveta\Desktop\git>git clone https://github.com/liz4simpson/lr2.9.git
Cloning into 'lr2.9'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
Receiving objects: 100% (10/10), 5.14 KiB | 526.
remote: Total 10 (delta 2), reused 0 (delta 0), pack-reused 0
Resolving deltas: 100% (2/2), done.
```

Рисунок 2. Клонирование репозитория

Организовала свой репозиторий в соответствии с моделью ветвления git-flow.

```

C:\Users\Elizaveta\Desktop\git\lr2.9>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/Elizaveta/Desktop/git/lr2.9/.git/hooks]

```

Рисунок 3. Организация репозитория согласно модели ветвления get-flow

2. Создала проект PyCharm в папке репозитория, проработала примеры из лабораторной работы.

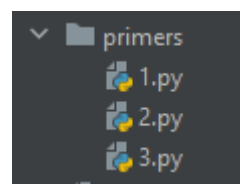


Рисунок 4. Проработанные примеры

```

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

# Пример обратного отсчета, написанного с и

liz4simpson

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

```

1 x

```

C:\nana\lr2.9\Scripts\python.exe C:\Users\Eliz
Process finished with exit code 0

```

Рисунок 5.1. Результат выполнения программы

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """
5     Любое вычисление, которое может быть выполнено
6     быть выполнено с использованием рекурсии. Вот
7     значения в последовательном контейнере, напри
8     использованием хвостовой рекурсии:
9 """
10
11 liz4simpson
12 def find_max(seq, max_so_far):
13     if not seq:
```

Рисунок 5.2. Результат выполнения программы

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 # Эта программа показывает работу декоратора, кото
5 # хвостового вызова. Он делает это, вызывая исклю
6 # прародителем, и перехватывает исключения, чтобы
7
8 import sys
9
10 liz4simpson
11 class TailRecurseException(Exception):
12     liz4simpson
13     def __init__(self, args, kwargs):
```

Рисунок 5.3. Результат выполнения программы

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 6. Результат выполнения программы

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 7. Результат выполнения программы

Индивидуально задание

(вар-25)

Решить следующую задачу: задан список положительных чисел, признаком конца которых служит отрицательное число. Используя рекурсию, подсчитать количество чисел и их сумму.

```
>> 12 14 -4 -6 10 -5
Сумма положительных элементов: 36
Количество положительных элементов: 3
```

Рисунок 8. Результат выполнения программы

4. Сделала коммит, выполнила слияние с веткой main, и запустила изменения в удаленный репозиторий.

```
C:\Users\Elizaveta\Desktop\git\lr2.9>git add .
C:\Users\Elizaveta\Desktop\git\lr2.9>git commit -m "added prjct"
[develop cceelb6] added prjct
8 files changed, 267 insertions(+)
create mode 100644 Idividual/var11.py
create mode 100644 exercise/1.py
create mode 100644 exercise/2.1.py
create mode 100644 lr2.9.docx
create mode 100644 primers/1.py
```

Рисунок 11. Фиксация и коммит файлов

```
C:\Users\Elizaveta\Desktop\git\lr2.9>git push --set-upstream origin develop
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 4 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (13/13), 605.96 KiB | 12.37 MiB/s, done.
Total 13 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/liz4simpson/lr2.9/pull/new/develop
remote:
To https://github.com/liz4simpson/lr2.9.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.

C:\Users\Elizaveta\Desktop\git\lr2.9>
C:\Users\Elizaveta\Desktop\git\lr2.9>git checkout main
Unlink of file 'lr2.9.docx' failed. Should I try again? (y/n) y
Unlink of file 'lr2.9.docx' failed. Should I try again? (y/n) y
Updating files: 100% (8/8), done.
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Рисунок 4.1 коммит и пуш изменений и переход на ветку main

```
C:\Users\Elizaveta\Desktop\git\lr2.9>git merge develop
Updating fc79502..ccee1b6
Fast-forward
 Individual/var11.py | 24 ++++++
 exercise/1.py       | 77 ++++++
 exercise/2.1.py     | 78 ++++++
 lr2.9.docx          | Bin 0 -> 651890 bytes
 primers/1.py        | 12 ++++++
 primers/2.py        | 18 ++++++
 primers/3.py        | 58 ++++++
 ~$lr2.9.docx        | Bin 0 -> 162 bytes
8 files changed, 267 insertions(+)
create mode 100644 Individual/var11.py
create mode 100644 exercise/1.py
create mode 100644 exercise/2.1.py
create mode 100644 lr2.9.docx
create mode 100644 primers/1.py
create mode 100644 primers/2.py
create mode 100644 primers/3.py
create mode 100644 ~$lr2.9.docx
```

Рисунок 12. Слияние ветки develop с main

```
C:\Users\Elizaveta\Desktop\git\lr2.9>git push
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/liz4simpson/lr2.9.git
 fc79502..ccee1b6  main -> main

C:\Users\Elizaveta\Desktop\git\lr2.9>
```

Рисунок 13. Отправка изменений на удаленный репозиторий

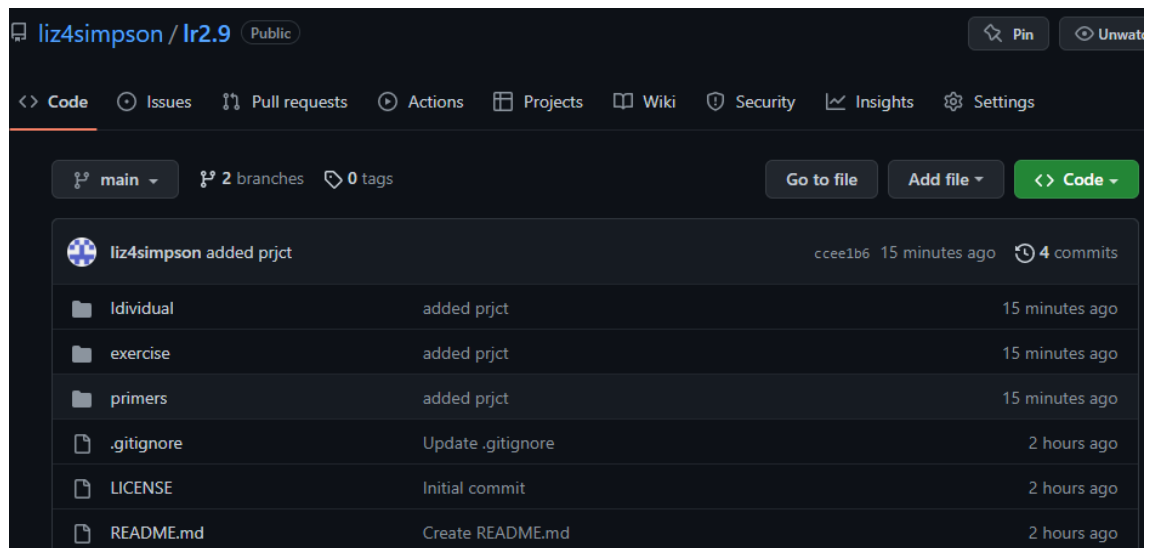


Рисунок 14. Изменения на удаленном репозитории

Вывод: в результате выполнения лабораторной работы были приобретены навыки для работы с рекурсией при написании программ с помощью языка программирования Python версии 3.x

Ответы на 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.