

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 №2.9

Дисциплина: «Программирование на Python»

Тема: «Рекурсия в языке Python»

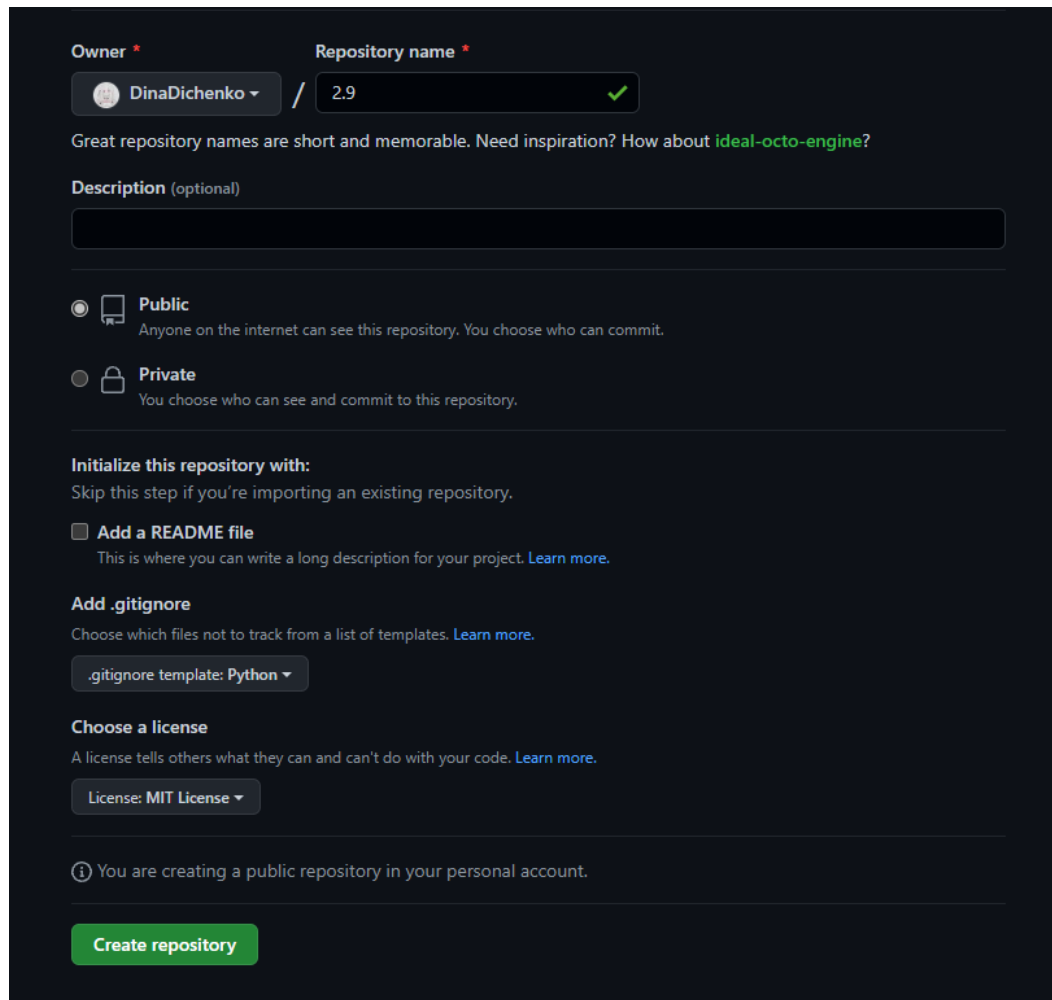
Выполнила: студентка 2
курса, группы ИВТ-б-о-21-1
Диченко Дина Алексеевна

Ставрополь 2022

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

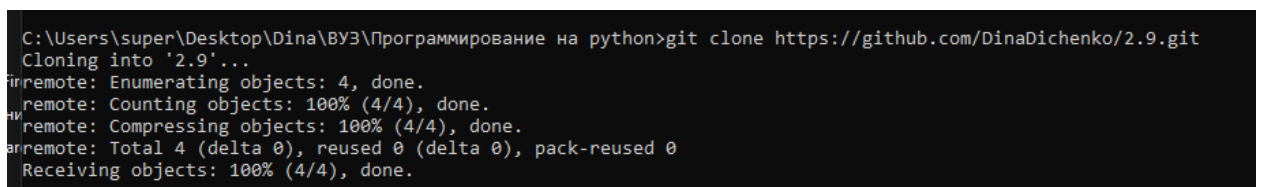
Практическая часть:

1. Создала общедоступный репозиторий и клонировала его.



The screenshot shows the GitHub 'Create repository' form. At the top, the 'Owner' is set to 'DinaDichenko' and the 'Repository name' is '2.9', which is marked as valid with a green checkmark. Below this, there is a text input field for a 'Description (optional)'. The 'Public' option is selected with a radio button, indicating that anyone on the internet can see and commit to the repository. Under the 'Initialize this repository with:' section, the 'Add a README file' checkbox is unchecked. The 'Add .gitignore' section shows a dropdown menu for the '.gitignore template' set to 'Python'. In the 'Choose a license' section, the 'License' dropdown is set to 'MIT License'. At the bottom, a green button labeled 'Create repository' is visible. A small information icon and text at the bottom state: 'You are creating a public repository in your personal account.'

Рисунок 1. Создание репозитория



```
C:\Users\super\Desktop\Dina\ВУЗ\Программирование на python>git clone https://github.com/DinaDichenko/2.9.git
Cloning into '2.9'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок 2. Клонирование репозитория

2. Дополнила файл .gitignore.

```
# Created by https://www.toptal.com/developers/gitignore/api/python,visualstudio
# Edit at https://www.toptal.com/developers/gitignore?templates=python,visualstudio

### Python ###
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[od]
*$py.class

# C extensions
*.so
.idea/
.idea\

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
*.egg-info/
```

Рисунок 3. Изменение файла .gitignore

3. Организовала репозиторий в соответствии с git-flow.

```
C:\Users\super\Desktop\Dina\ВУЗ\Программирование на python\2.9>git flow init

Which branch should be used for bringing forth production releases?
- develop
- main
Branch name for production releases: [main] main

Which branch should be used for integration of the "next release"?
- develop
Branch name for "next release" development: [develop] develop

How to name your supporting branch prefixes?
Feature branches? [feature/] fea
Bugfix branches? [bugfix/] bug
Release branches? [release/] rel
Hotfix branches? [hotfix/] hot
Support branches? [support/] sup
Version tag prefix? [] ver
Hooks and filters directory? [C:/Users/super/Desktop/Dina/ВУЗ/Программирование на python/2.9/.git/hooks] hook
```

Рисунок 4. Организация репозитория в соответствии с git-flow

4. 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?

```
Recursive factorial: 0.00014099999998506973
Recursive Fibonacci numbers: 0.00014359999977386906
Iterative factorial: 0.0001403000001118926
Iterative Fibonacci numbers: 0.00013989999979457934
Factorial with decorator: 0.0001401999998051906
Fibonacci numbers with decorator: 0.00013949999993201345
Press any key to continue . . .
```

Рисунок 5. Результат работы программы

Быстрее вычислять с декоратором, т.к. функция, используемая внутри него, меняется вне его.

5. 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.

```
Factorial: 0.00013889999991079094
Fibonacci numbers: 0.00014069999997445848
factorial with stack introspection: 0.000140100000040798332
Fibonacci numbers with stack introspection: 0.000140299999965714524
Press any key to continue . . .
```

Рисунок 5. Результат работы программы

6. Решила индивидуальное задание.

Напечатать в обратном порядке последовательность чисел, признаком конца которой является 0.

```
6
8
4
9
0
[0, 9, 4, 8, 6]
Для продолжения нажмите любую клавишу . . .
```

Рисунок 6. Результат работы программы

Ответы на вопросы:

1. Для чего нужна рекурсия?

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. Что называется базой рекурсии?

Аргументы, для которых значения функции определены (элементарные задачи).

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

Стек вызовов — 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.

При возврате из подпрограммы или обработчика прерывания, адрес возврата снимается со стека и управление передается на следующую инструкцию приостановленной (под-)программы.

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

```
sys.getrecursionlimit().
```

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

```
RuntimeError: Maximum Recursion Depth Exceeded
```

6. Как изменить максимальную глубину рекурсии в языке Python?

Можно изменить предел глубины рекурсии с помощью вызова:

```
sys.setrecursionlimit(limit).
```

7. Каково назначение декоратора `lru_cache` ?

`lru_cache` можно использовать для уменьшения количества 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.

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