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»

Выполнил: студент 2 курса
группы ИВТ-б-о-21-1
Урусов Максим Андреевич

Ставрополь 2022

Выполнение работы:

1. Создал репозиторий в GitHub «rep 2.6» в который добавил .gitignore, который дополнил правила для работы с IDE PyCharm с ЯП Python, выбрал лицензию MIT, клонировал его на лок. сервер и организовал в соответствии с моделью ветвления git-flow.

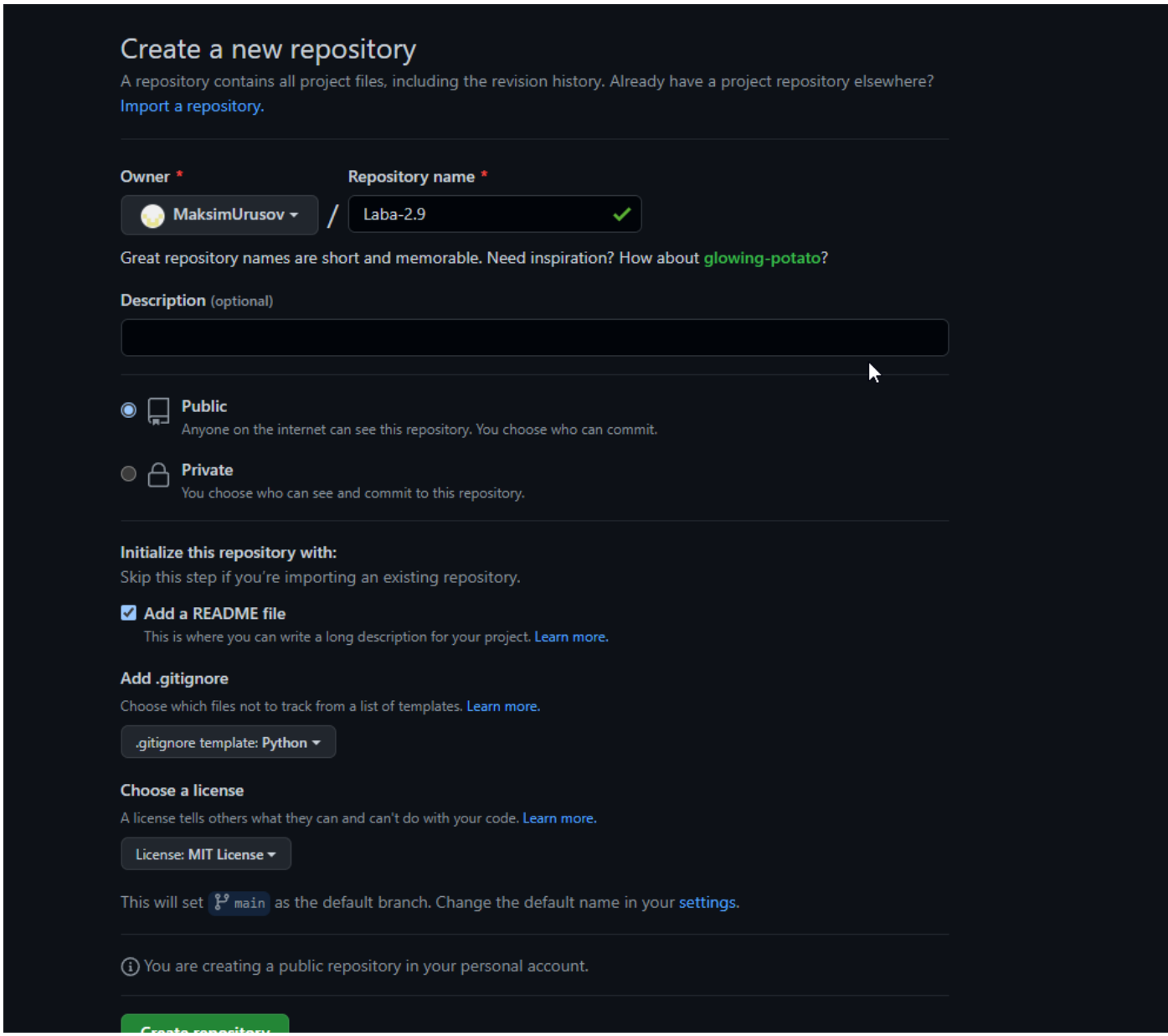

Рисунок 1.1 Создание репозитория

```
C:\Users\den-n>git clone https://github.com/MaksimUrusov/Laba-2.9.git
Cloning into 'Laba-2.9'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 1.2 Клонирование репозитория

```
C:\Users\den-n\Laba-2.9>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/den-n/Laba-2.9/.git/hooks]

C:\Users\den-n\Laba-2.9>
```

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. Выполнение заданий

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 измениться 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

3. 18 вариант (4). Выполнил индивидуальное задание.

```
C:\Users\den-n\AppData\Local\Programs\Python\Python310\python.exe C:/Users/den-n/Laba-2.9/ind.py
Введите число 3
[[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]]

Process finished with exit code 0
```

Рисунок 4.1 Вывод программы индивидуального задания

4. Сделал коммит, выполнил слияние с веткой main, и запушил изменения в уд. репозиторий.

```
D:\учёба\пнп\2.9>git push origin develop
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 8 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 5.70 KiB | 5.70 MiB/s, done.
Total 12 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:      https://github.com/LenkaGolovach/2.9/pull/new/develop
remote:
To https://github.com/LenkaGolovach/2.9.git
 * [new branch]      develop -> develop

D:\учёба\пнп\2.9>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)
```

Рисунок 4.1 коммит и пуш изменений и переход на ветку main

```
D:\учёба\пнп\2.9>git merge develop
Updating 9c21a6a..bf6b013
Fast-forward
 ind.py   | 26 ++++++++++++++++++++++
 prim1.py | 12 ++++++++++
 prim2.py | 18 ++++++++++++++++
 prim3.py | 58 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 zad1.py  | 77 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 zad2.py  | 78 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 6 files changed, 269 insertions(+)
 create mode 100644 ind.py
 create mode 100644 prim1.py
 create mode 100644 prim2.py
 create mode 100644 prim3.py
 create mode 100644 zad1.py
 create mode 100644 zad2.py
```

Рисунок 4.2 Слияние ветки main с develop

Рисунок 4.4 Изменения на удаленном сервере

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 RunTime.

6. Как изменить максимальную глубину рекурсии в языке Python?

С помощью sys.setrecursionlimit(число).

7. Каково назначение декоратора lru_cache?

Функция lru_cache предназначается 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 поэтому адресу. Либо непосредственно перед, либо сразу после возврата из функции стек очищается от параметров.