

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 ЛАБОРАТОРНОЙ РАБОТЕ №12
дисциплины «Основы программной инженерии»

Выполнил:
Мелтонян Одиссей
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил: Воронкин Р. А.

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

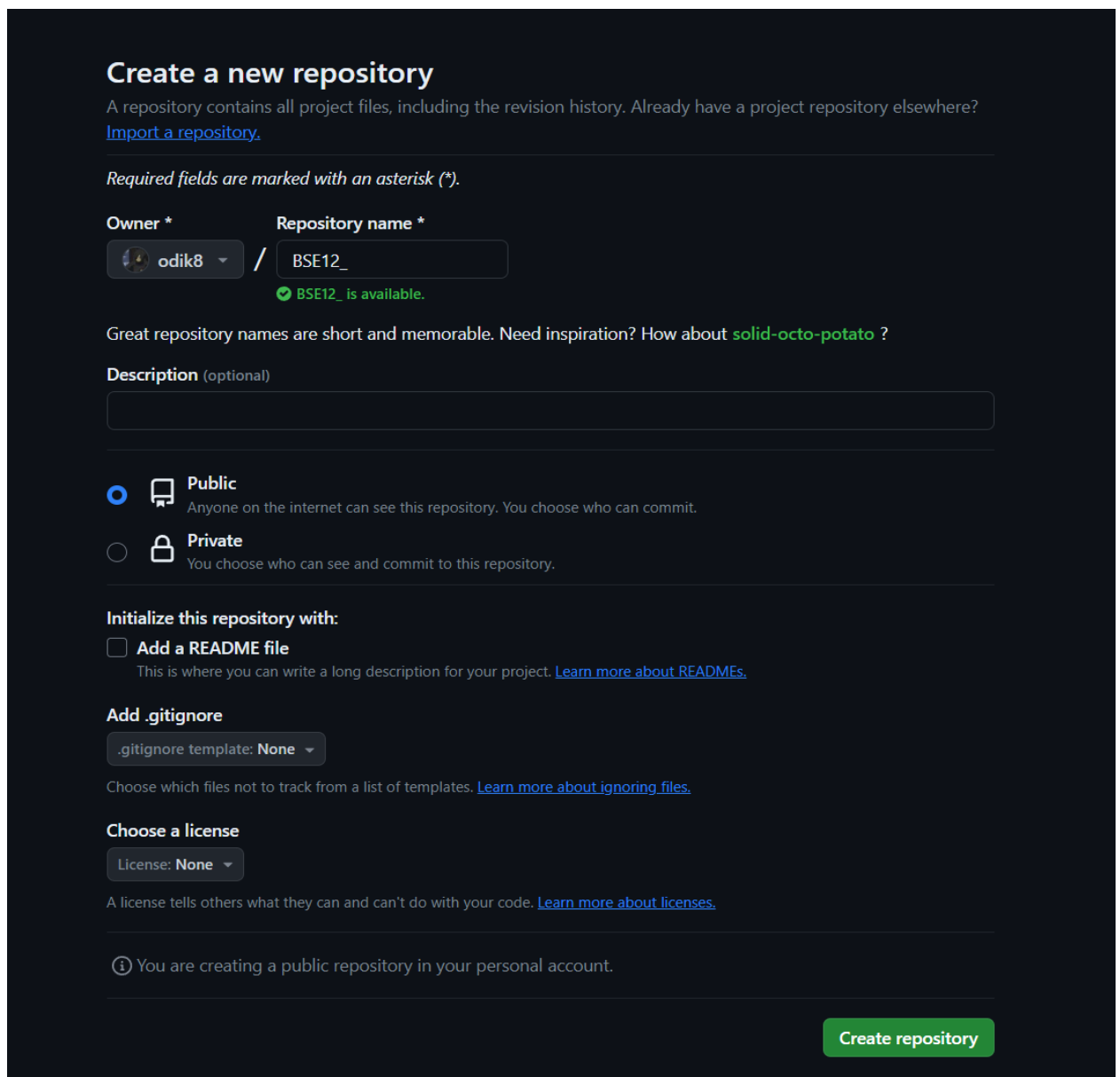
Ставрополь, 2023 г

Тема: Рекурсия в языке Python

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

Ход работы:

1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.




The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and explains that a repository contains all project files. Below this, it states 'Required fields are marked with an asterisk (*)'. The 'Owner' field is set to 'odik8' and the 'Repository name' field is 'BSE12_'. A green checkmark indicates 'BSE12_ is available.'. There is a suggestion for 'solid-octo-potato'. The 'Description' field is empty. Under 'Visibility', 'Public' is selected. The 'Initialize this repository with:' section has 'Add a README file' checked. The '.gitignore' template is set to 'None'. The 'License' is set to 'None'. A note at the bottom says 'You are creating a public repository in your personal account.' and a green 'Create repository' button is at the bottom right.

Create a new repository

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

Required fields are marked with an asterisk (*).


Owner * / Repository name *


 odik8 / BSE12_

✔ BSE12_ is available.

Great repository names are short and memorable. Need inspiration? How about **solid-octo-potato** ?

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:

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

Add .gitignore


.gitignore template: **None**

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

Choose a license

License: **None**

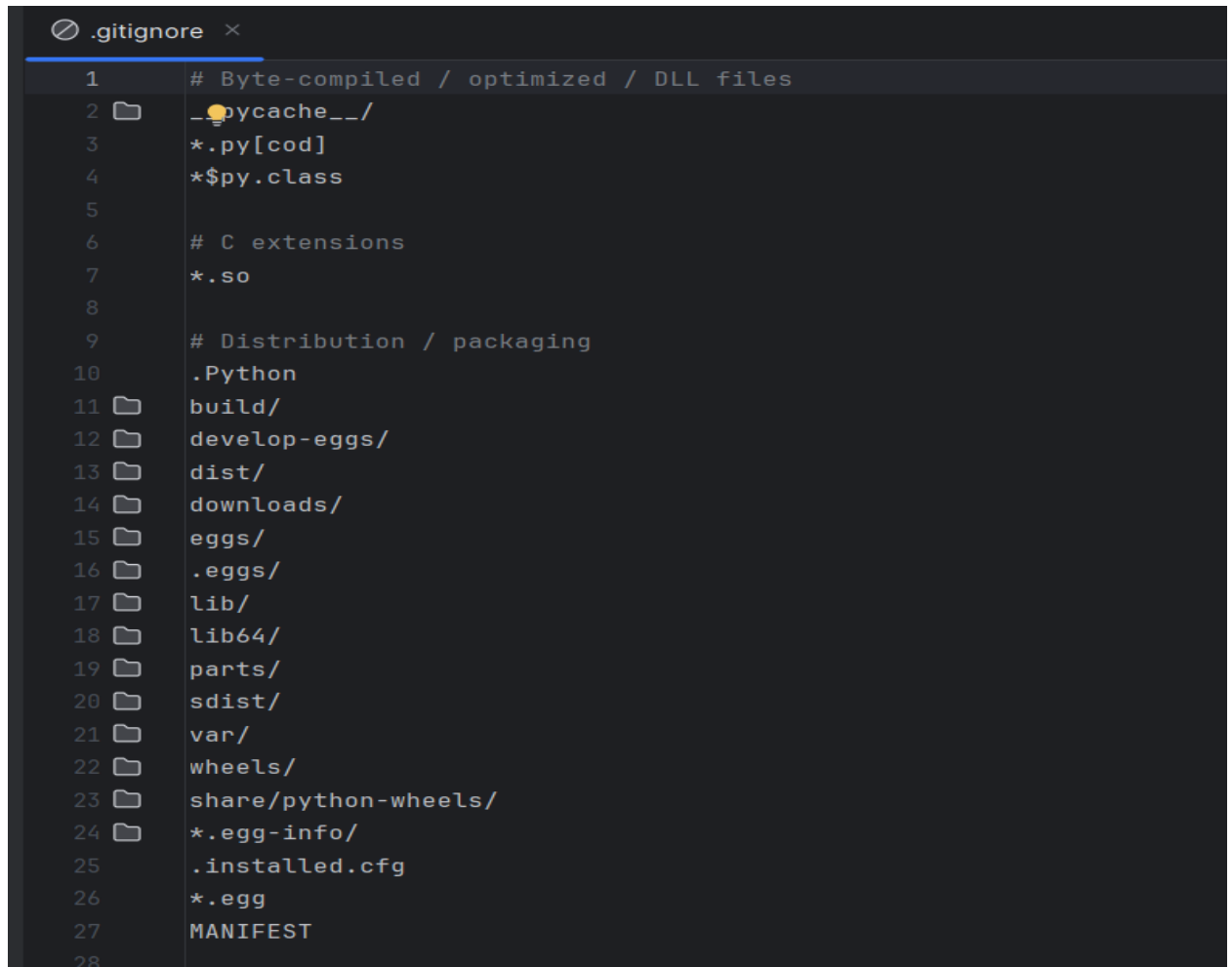
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

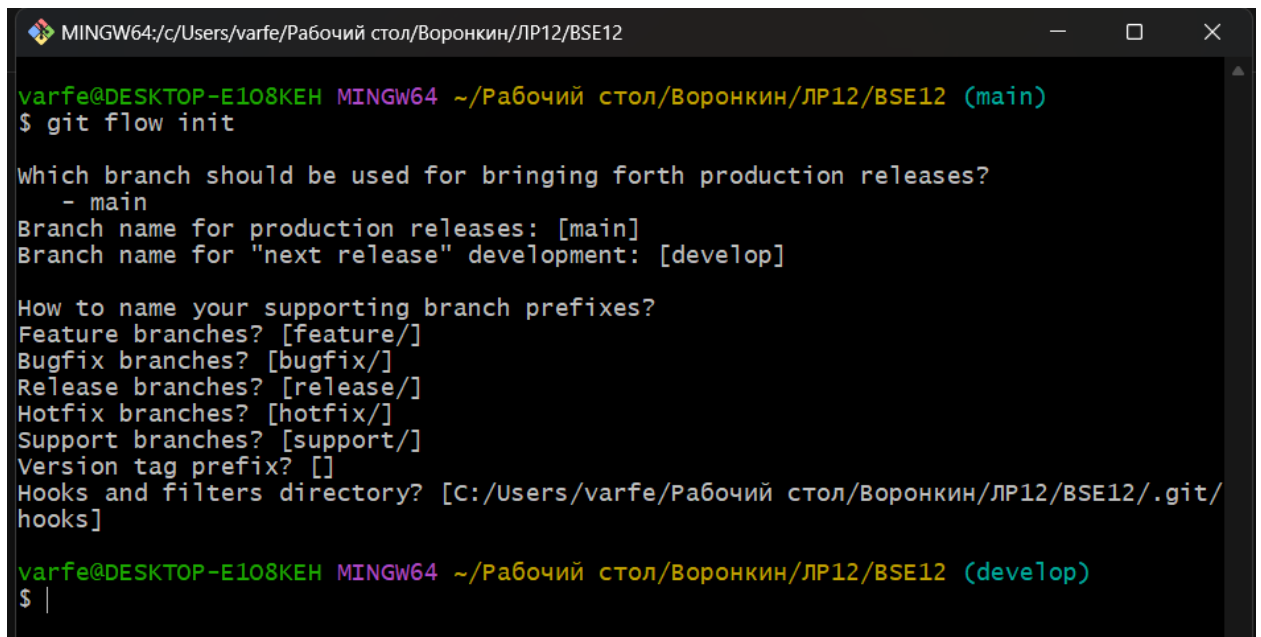
3. Выполнил клонирование созданного репозитория.
4. Дополнил файл .gitignore необходимыми правилами для работы с IDE PyCharm.



```
.gitignore
1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
```

Рисунок 2 – Файл .gitignore

5. Организовал свой репозиторий в соответствии с моделью ветвления git-flow.



```
MINGW64:/c:/Users/varfe/Рабочий стол/Воронкин/ЛР12/BSE12
varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/Воронкин/ЛР12/BSE12 (main)
$ 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]

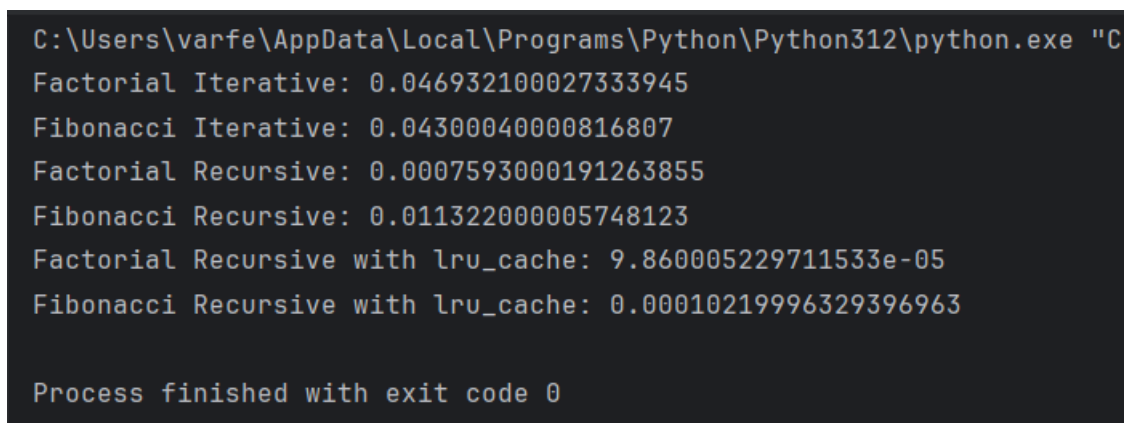
How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/varfe/Рабочий стол/Воронкин/ЛР12/BSE12/.git/hooks]

varfe@DESKTOP-E108KEN MINGW64 ~/Рабочий стол/Воронкин/ЛР12/BSE12 (develop)
$ |
```

Рисунок 3 – Инициализация git-flow

6. Создал проект PyCharm в папке репозитория.

7. Самостоятельно изучил работу со стандартным пакетом Python timeit. Оценил с помощью этого модуля скорость работы итеративной и рекурсивной версий функций factorial и fib. Скорость работы рекурсивных версий функций factorial и fib при использовании декоратора lru_cache можно наблюдать ниже



```
C:\Users\varfe\AppData\Local\Programs\Python\Python312\python.exe "C
Factorial Iterative: 0.046932100027333945
Fibonacci Iterative: 0.04300040000816807
Factorial Recursive: 0.0007593000191263855
Fibonacci Recursive: 0.011322000005748123
Factorial Recursive with lru_cache: 9.860005229711533e-05
Fibonacci Recursive with lru_cache: 0.00010219996329396963

Process finished with exit code 0
```

Рисунок 4 – Сравнение скоростей функций

9. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

```

2 usages
1 def count_and_sum_recursive(numbers, index=0, count=0, total_sum=0):
2     if index == len(numbers):
3         return count, total_sum
4
5     current_number = numbers[index]
6     if current_number < 0:
7         return count, total_sum
8     else:
9         count += 1
10        total_sum += current_number
11        return count_and_sum_recursive(numbers, index + 1, count, total_sum)
12
13
14 if __name__ == "__main__":
15     numbers_list = list(map(float, input("Вводите числа через пробел: ").split()))
16     result_count, result_sum = count_and_sum_recursive(numbers_list)
17
18     print("Количество чисел:", result_count)
19     print("Сумма чисел:", result_sum)
20

```

Рисунок 5 – Код решения задачи

```

C:\Users\varfe\AppData\Local\Programs\Python\Python3
Вводите числа через пробел: 6 4 3 -4 1 9 0 8 7
Количество чисел: 3
Сумма чисел: 13.0

Process finished with exit code 0

```

Рисунок 6 – Результат работы программы

10. Зафиксировал сделанные изменения в репозитории.
11. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.
12. Выполнил слияние ветки для разработки с веткой main.
13. Отправил сделанные изменения на сервер GitHub.
14. Отправил адрес репозитория GitHub на электронный адрес преподавателя.

Вопросы для защиты работы:

1. Для чего нужна рекурсия? Рекурсия в программировании — это техника, при которой функция вызывает саму себя. Она часто используется для решения задач, которые могут быть разбиты на более простые подзадачи. Рекурсия упрощает написание кода и понимание задачи, особенно в случаях, когда структура данных или задача имеют рекурсивную природу.

2. Что называется базой рекурсии? Базой рекурсии является условие, при котором функция прекращает вызывать саму себя и возвращает результат. Базовый случай не включает в себя рекурсивные вызовы и служит завершающим условием для рекурсии.

3. Стек программы и его использование при вызове функций: Стек программы — это структура данных, которая хранит информацию о вызовах функций в программе. Каждый раз, когда функция вызывается, информация о вызове (локальные переменные, адрес возврата и другие данные) помещается на вершину стека. При завершении функции эта информация удаляется из стека. Стек используется для отслеживания последовательности вызовов функций и их возвратов.

4. Как получить текущее значение максимальной глубины рекурсии в языке Python: Максимальная глубина рекурсии в Python можно получить с помощью `sys.getrecursionlimit()`. Например:

```
import sys print(sys.getrecursionlimit())
```

5. Что произойдет, если число рекурсивных вызовов превысит максимальную глубину рекурсии в языке Python: Произойдет исключение **RecursionError**. Это защитная мера, предотвращающая переполнение стека вызовов.

6. Как изменить максимальную глубину рекурсии в языке Python: Максимальную глубину рекурсии можно изменить с помощью функции `sys.setrecursionlimit(limit)`. Однако, изменение этого значения не

рекомендуется, так как это может привести к нестабильной работе программы. По умолчанию в Python установлено достаточно безопасное значение.

7. Назначение декоратора lru_cache: Декоратор **lru_cache** используется для кеширования результатов вызовов функций. Он сохраняет результаты предыдущих вызовов функции и, при повторном вызове с теми же аргументами, возвращает закешированный результат вместо повторного выполнения функции. Это может существенно улучшить производительность функций, особенно рекурсивных.

8. Хвостовая рекурсия и оптимизация хвостовых вызовов: Хвостовая рекурсия - это вид рекурсии, когда рекурсивный вызов является последней операцией в функции. В Python стандартной оптимизации хвостовых вызовов нет, поэтому хвостовая рекурсия может привести к переполнению стека. Однако, можно использовать оптимизированный декоратор **@tail_recursive** из библиотеки **trampolines**, чтобы преобразовать хвостовую рекурсию в цикл и избежать переполнения стека.