

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»**

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

---

(подпись)

Руководитель практики:  
Воронкин Р. А., доцент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

# Ход работы

## 1. Я изучила теоретический материал работы

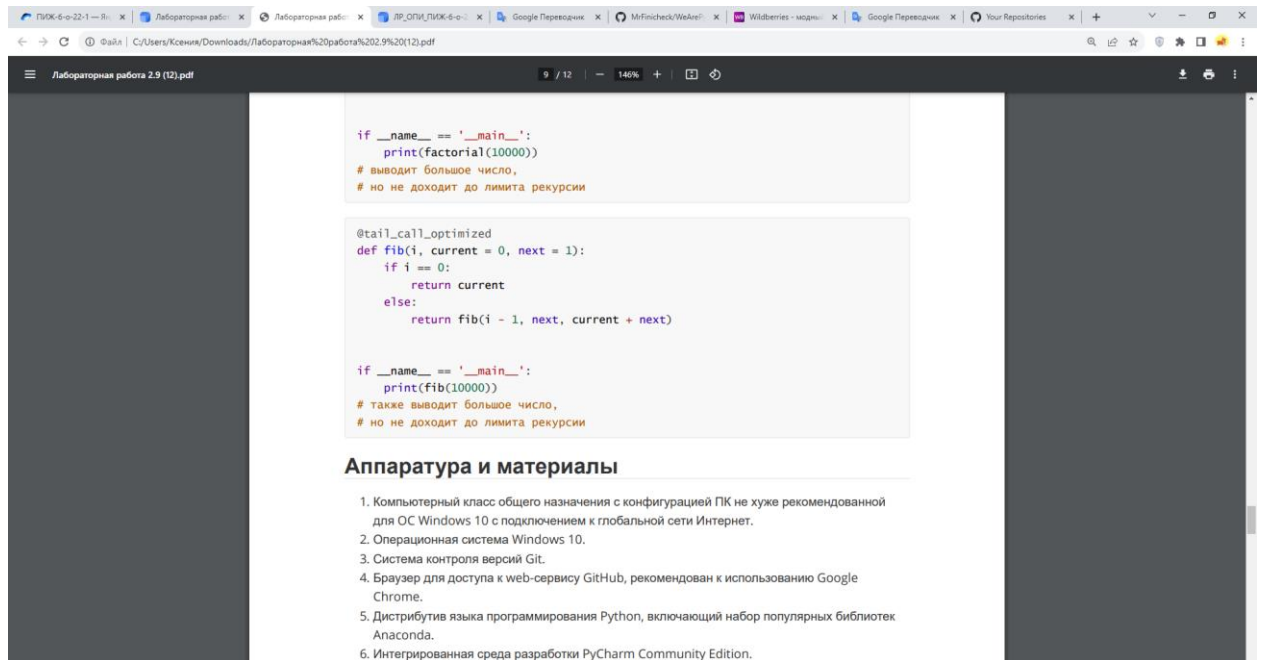


Рисунок 1.1 – Изучение материала для лабораторной работы

## 2. Создала общедоступный репозиторий на GitHub, в котором будет использована 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.](#)

Required fields are marked with an asterisk (\*).

Owner \* MrFincheck / Repository name \* boa  
✓ boa is available.

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

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: Python]

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

Choose a license  
[License: MIT License]

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 2.1 – Настройка репозитория

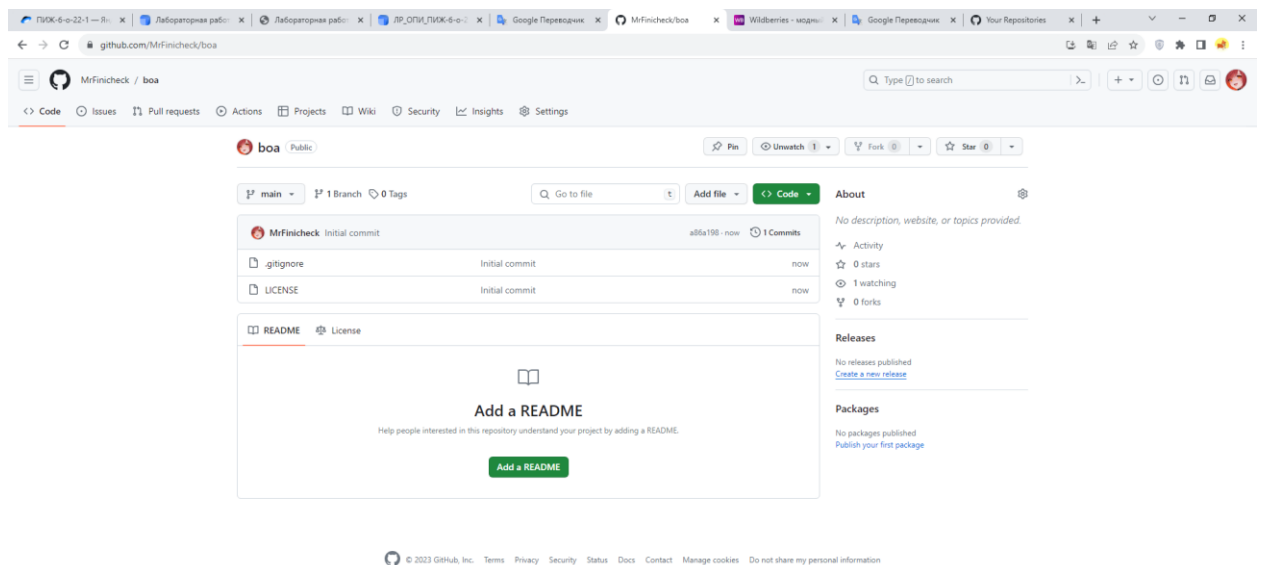


Рисунок 2.2 – Готовый репозиторий

### 3. Выполняю клонирование созданного репозитория

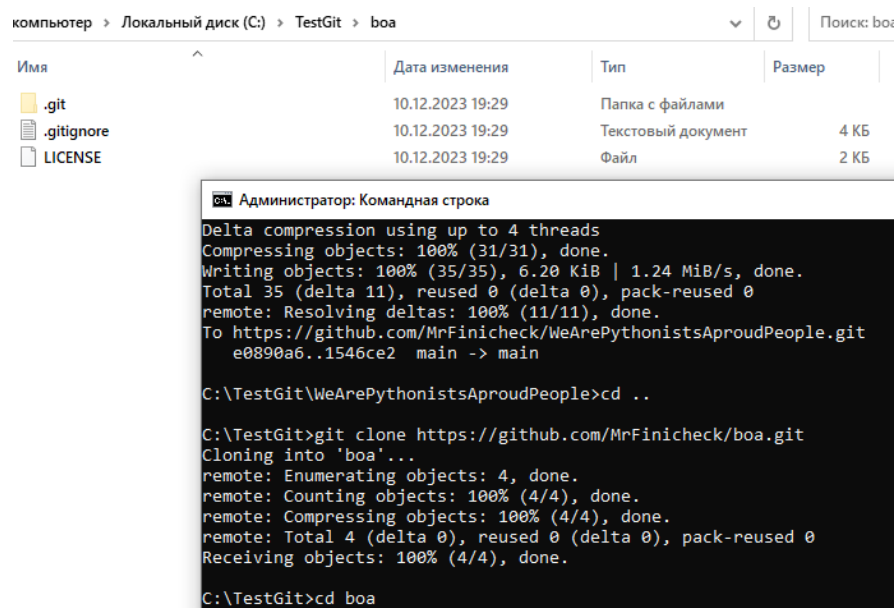
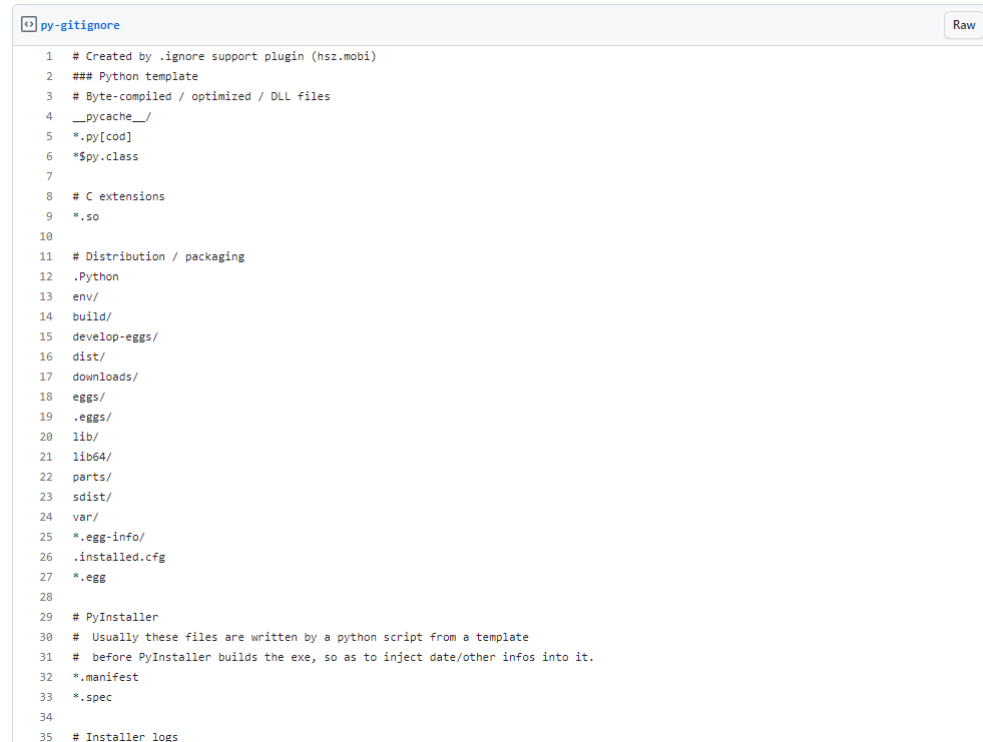


Рисунок 3.1 – Клонирование репозитория на локальный диск

### 4. Дополнила файл .gitignore необходимыми правилами для работы с IDE PyCharm

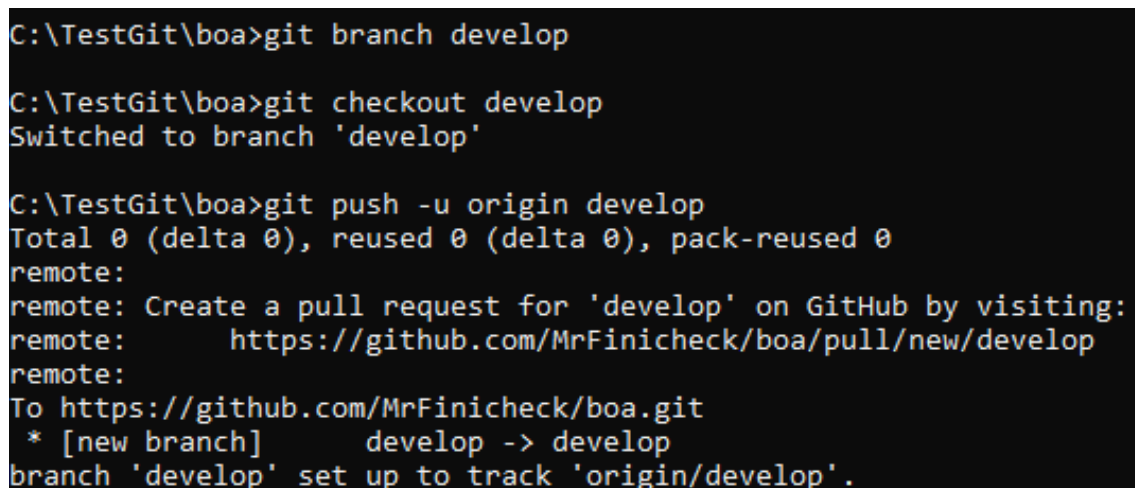
python pycharm gitignore

A screenshot of a PyCharm IDE window showing a .gitignore file. The window title is 'py-gitignore'. The file content is a standard Python project .gitignore template, listing various files and directories to be ignored, such as \_\_pycache\_\_, \*.py[cod], \*.py.class, \*.so, .Python, env/, build/, develop-eggs/, dist/, downloads/, eggs/, .eggs/, lib/, lib64/, parts/, sdist/, var/, \*.egg-info/, .installed.cfg, \*.egg, \*.manifest, \*.spec, and \*.egg-info/. The file is 35 lines long.

```
1 # Created by .ignore support plugin (hsz.mobi)
2 ### Python template
3 # Byte-compiled / optimized / DLL files
4 __pycache__/
5 *.py[cod]
6 *$py.class
7
8 # C extensions
9 *.so
10
11 # Distribution / packaging
12 .Python
13 env/
14 build/
15 develop-eggs/
16 dist/
17 downloads/
18 eggs/
19 .eggs/
20 lib/
21 lib64/
22 parts/
23 sdist/
24 var/
25 *.egg-info/
26 .installed.cfg
27 *.egg
28
29 # PyInstaller
30 # Usually these files are written by a python script from a template
31 # before PyInstaller builds the exe, so as to inject date/other infos into it.
32 *.manifest
33 *.spec
34
35 # Installer logs
```

Рисунок 4.1 – .gitignore для IDE PyCharm

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

A terminal window screenshot showing the steps to create a 'develop' branch from 'main' and push it to GitHub. The commands and their outputs are as follows:

```
C:\TestGit\boa>git branch develop
C:\TestGit\boa>git checkout develop
Switched to branch 'develop'
C:\TestGit\boa>git push -u origin develop
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/MrFinicheck/boa/pull/new/develop
remote:
To https://github.com/MrFinicheck/boa.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.
```

Рисунок 5.1 – Создание ветки develop от ветки main

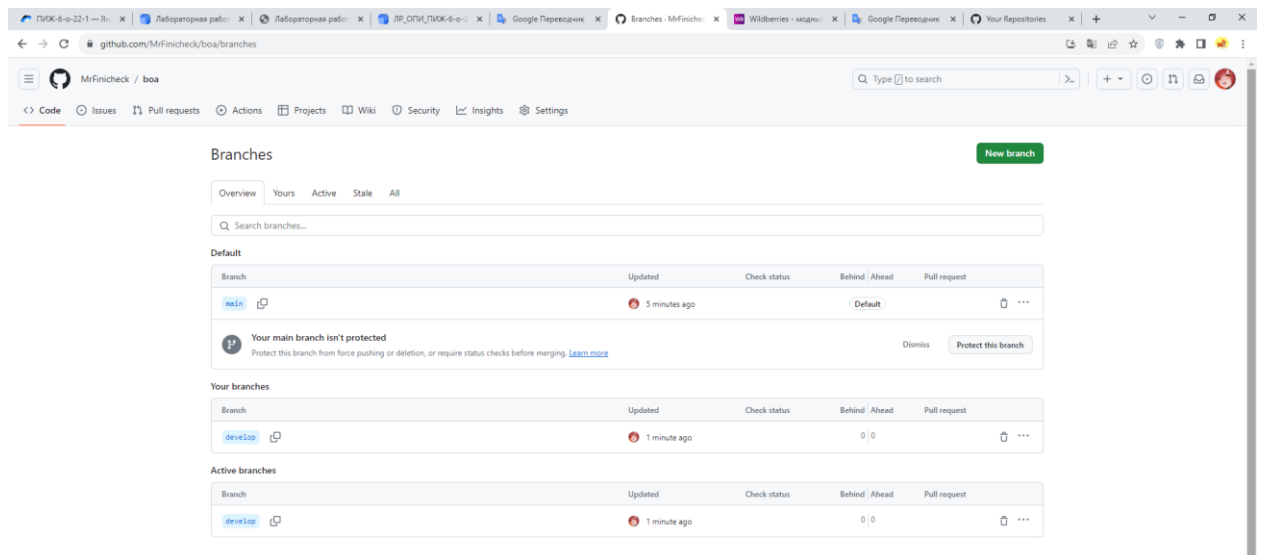


Рисунок 5.2 – Ветка develop на GitHub

## 6. Создала проект PyCharm в папке репозитория

Имя	Дата изменения	Тип	Размер
.git	10.12.2023 19:32	Папка с файлами	
PyCharm	10.12.2023 19:33	Папка с файлами	
.gitignore	10.12.2023 19:29	Текстовый документ	4 КБ
LICENSE	10.12.2023 19:29	Файл	2 КБ

Рисунок 6.1 – Репозиторий с проектом PyCharm

7. Самостоятельно изучила работу со стандартным пакетом Python `timeit`. Оценила с помощью этого модуля скорость работы итеративной и рекурсивной версий функций `factorial` и `fib`.

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

from timeit import timeit
from functools import lru_cache

import sys

# Выполнение функций без использования декоратора.

def factorial_iterable(n):
    # Итеративная версия функции factorial.
    multiply = 1
    while n > 1:
        multiply *= n
        n -= 1
    return multiply

def fib_iterable(n):
    # Итеративная версия функции fib.
    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
```

Рисунок 7.1 – Код программы Task7.py в IDE PyCharm

8. Самостоятельно проработала пример с оптимизацией хвостовых вызовов в Python. С помощью пакета timeit оценила скорость работы функций factorial и fib с использованием интроспекции стека и без использования интроспекции стека. Привела полученные результаты в отчет

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

import sys
from timeit import timeit

# Выполнение функций без использования интроспекции стека.

def factorial(n):
    # Функция для вычисления факториала.
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

def fib(n):
    # Функция для чисел Фибоначчи.
    if n <= 1:
        return n
    else:
        return fib(n - 1) + fib(n - 2)

# Выполнение функций с использованием интроспекции стека.
```

Рисунок 8.1 – Код программы Task8.py в IDE PyCharm

9. Выполнила индивидуальные задания. Привела в отчете скриншоты работы программ решения индивидуального задания.

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

def print_sum_repr(n, sum_repr=[], start=1):
    if n == 0:
        # Если число N равно 0, то печатаем представление суммы.
        print(sum_repr)
        return

    for i in range(start, n + 1):
        if n - i >= 0:
            # Добавляем i к текущему представлению суммы.
            print_sum_repr(n - i, sum_repr + [i], i)

if __name__ == '__main__':
    n = int(input("Введите натуральное число N: "))
    print(f"Возможные представления числа {n} "
          f"в виде суммы других натуральных чисел:")
    print_sum_repr(n)
```

Рисунок 9.1 – Код программы INDIndividual.py в IDE PyCharm

10. Зафиксировала сделанные изменения в репозитории.

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

import math

def cylinder():
    # Определить функцию для нахождения площади круга.
    def circle(radius):
        return math.pi * radius ** 2

    # Ввести радиус и высоту цилиндра.
    radius = float(input("Введите радиус цилиндра: "))
    height = float(input("Введите высоту цилиндра: "))

    # Посчитать площади боковой и всей поверхностей.
    lateral_surface_area = 2 * math.pi * radius * height
    total_surface_area = (lateral_surface_area +
                          2 * circle(radius))

    # Узнать хочет ли получить пользователь всю площадь.
    solution = input("Хотите получить только площадь не "
                    "только боковой поверхности, но и "
                    "цилиндра? (Да/Нет): ")

    if solution.lower() == "да":
        print(f"Площадь боковой поверхности цилиндра: "
              f"{lateral_surface_area}")
    else:
```

Рисунок 10.1 – Коммит файлов в репозитории git

11. Зафиксировала сделанные изменения в репозитории.

```
C:\TestGit\boa>git commit -m"add files"
[develop d1e5272] add files
4 files changed, 178 insertions(+)
create mode 100644 PyCharm/Individual/INDividual.py
create mode 100644 "PyCharm/Individual/\320\222\320\260\321\200\320\270\320\260\320\275\321\2026.txt"
create mode 100644 PyCharm/Tasks/MTask7.py
create mode 100644 PyCharm/Tasks/MTask8.py
```

Рисунок 11.1 – Коммит файлов в репозитории git

## Контрольные вопросы

1. Для чего нужна рекурсия?

Рекурсия функции нужна, когда требуется выполнить последовательность из одинаковых действий.

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 особая область памяти, которая используется для временного хранения данных во время выполнения программы. Стек работает по принципу LIFO (last in, first out). Стек вызовов работает так: при вызове вложенной функции, основная функция, откуда был вызов останавливается и создается блок памяти по новый вызов. В ячейку памяти записываются значения переменных и адрес возврата.

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

Чтобы проверить текущие параметры лимита, нужно запустить: `sys.getrecursionlimit()`

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?

Можно изменить предел глубины рекурсии с помощью вызова: `sys.setrecursionlimit(limit)`

7. Каково назначение декоратора `lru_cache`?

Декоратор `@lru_cache()` модуля `functools` оборачивает функцию с переданными в нее аргументами и запоминает возвращаемый результат соответствующий этим аргументам. Такое поведение может сэкономить время и ресурсы, когда дорогая или связанная с вводом/выводом функция периодически вызывается с одинаковыми аргументами.

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. Чтобы оптимизировать рекурсивные функции, мы можем использовать декоратор `@tail_call_optimized` для вызова нашей функции