

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 №14
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

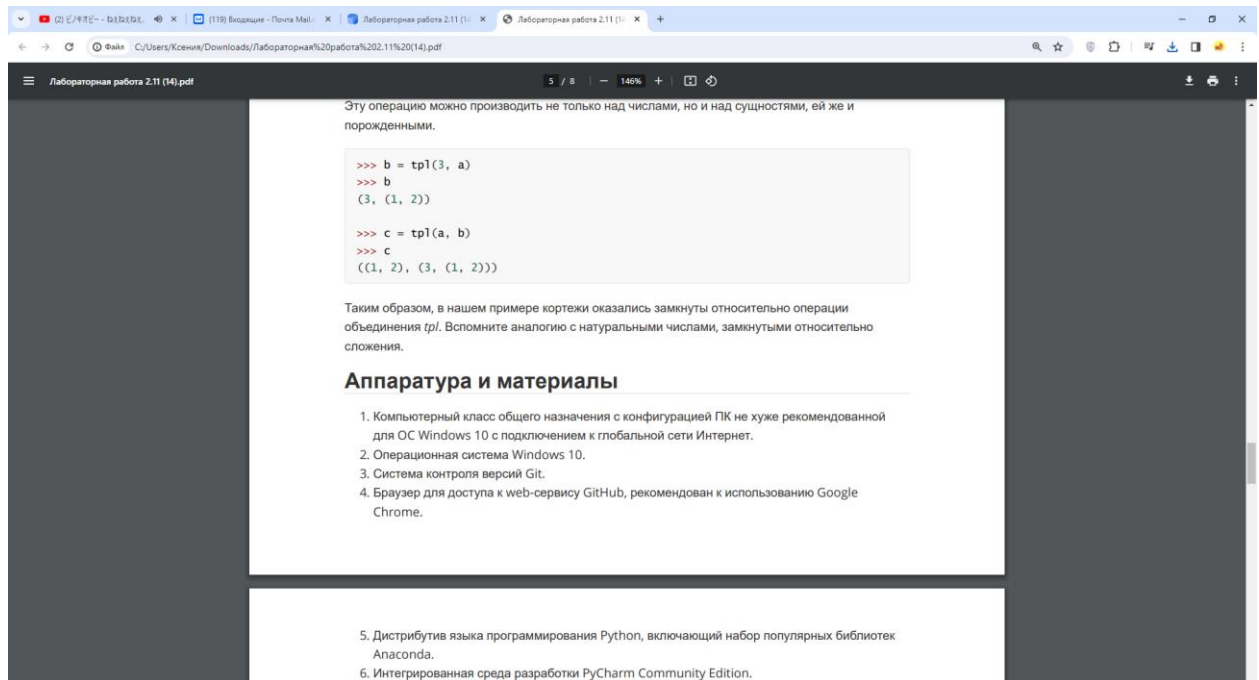


Рисунок 1.1 – Изучение материала для лабораторной работы

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 * MrFinichек / Repository name * PythonIsBetterThanJavasc
✔ PythonIsBetterThanJavascript is available.

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

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.](#)

① You are creating a public repository in your personal account.

[Create repository](#)

Рисунок 2.1 – Настройка репозитория

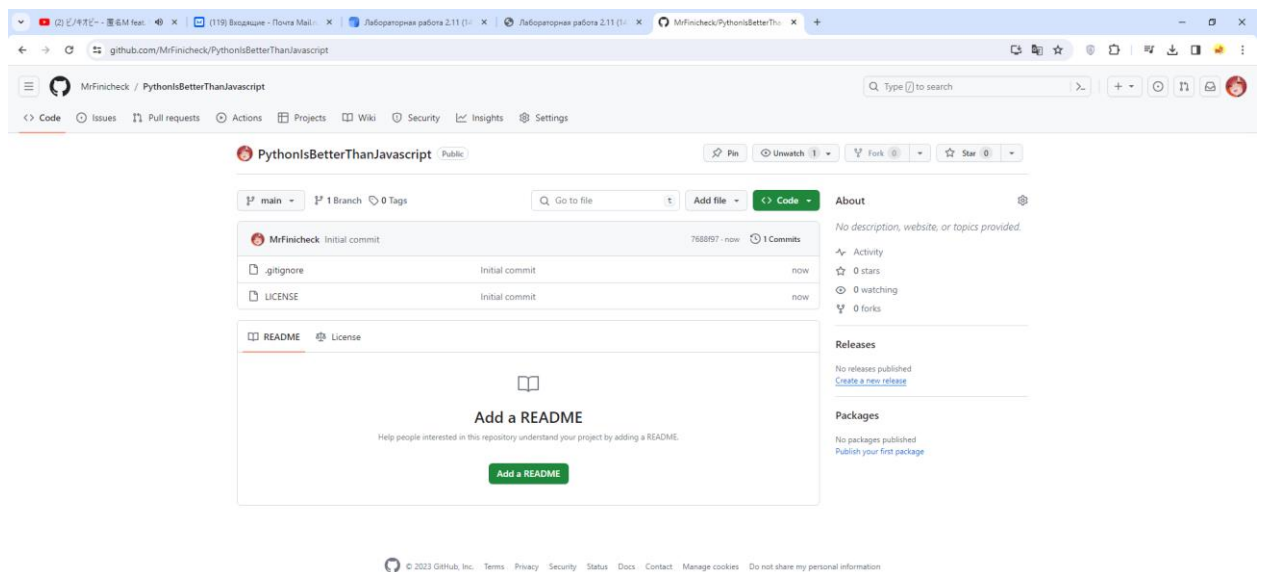


Рисунок 2.2 – Готовый репозиторий

3. Выполняю клонирование созданного репозитория

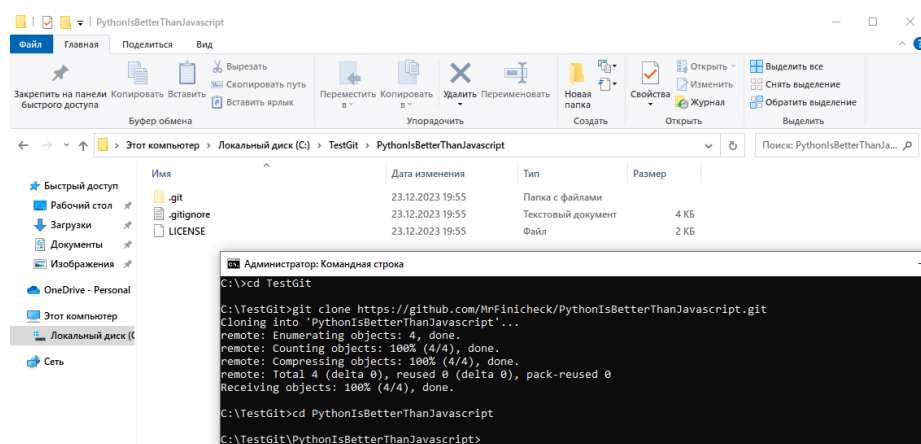
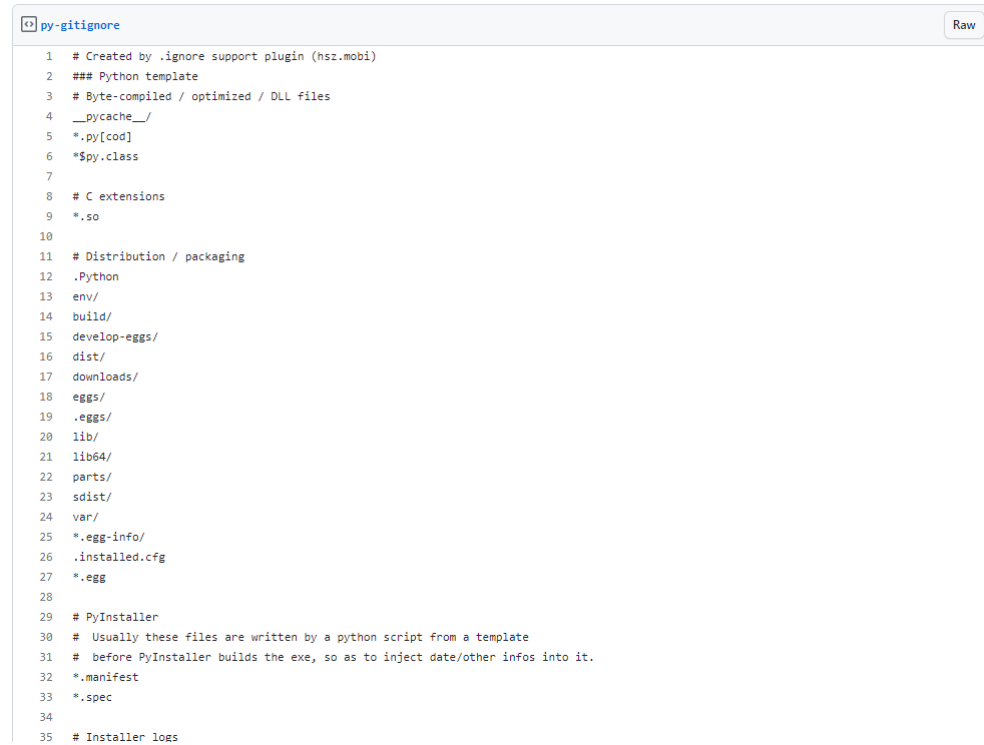


Рисунок 3.1 – Клонирование репозитория на локальный диск

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

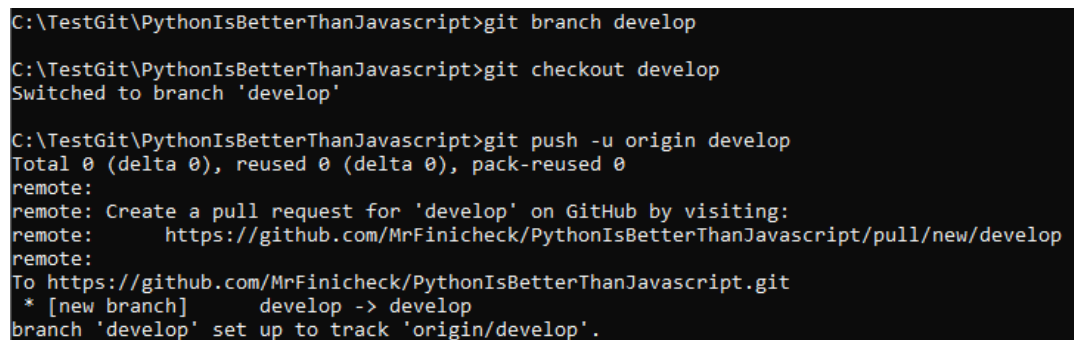
python pycharm gitignore

A screenshot of a PyCharm IDE window showing a .gitignore file. The window title is 'py-gitignore'. The file content is a standard Python .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 *.spec. The file is numbered from 1 to 35.

```
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 a series of git commands and their output. The commands are: 'git branch develop', 'git checkout develop', and 'git push -u origin develop'. The output shows the branch being created and pushed to the remote repository. The terminal text is as follows:

```
C:\TestGit\PythonIsBetterThanJavascript>git branch develop
C:\TestGit\PythonIsBetterThanJavascript>git checkout develop
Switched to branch 'develop'
C:\TestGit\PythonIsBetterThanJavascript>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/PythonIsBetterThanJavascript/pull/new/develop
remote:
To https://github.com/MrFinicheck/PythonIsBetterThanJavascript.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.
```

Рисунок 5.1 – Создание ветки develop от ветки main

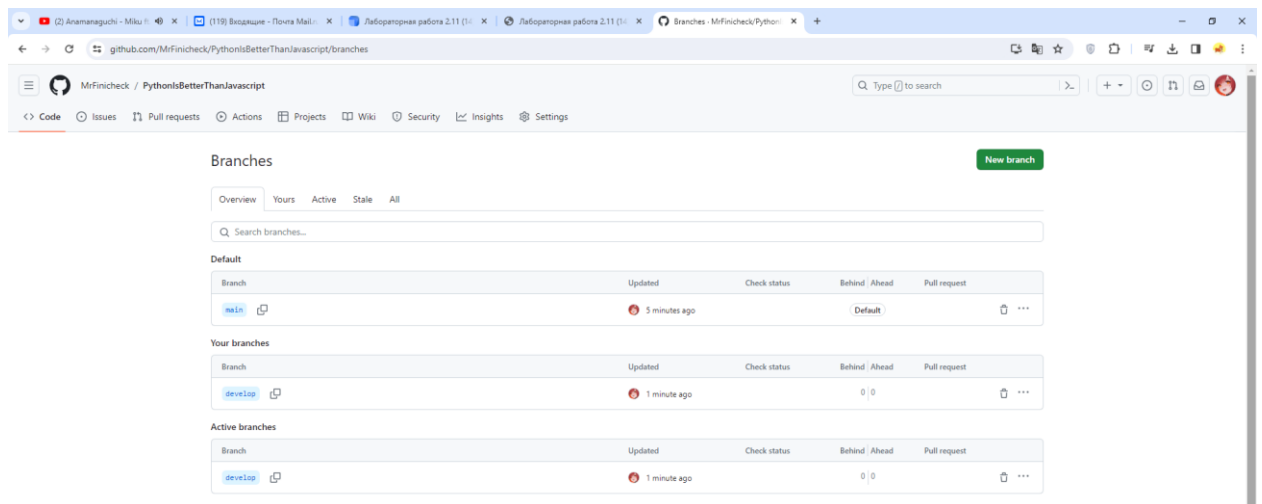


Рисунок 5.2 – Ветка develop на GitHub

6. Создала проект PyCharm в папке репозитория

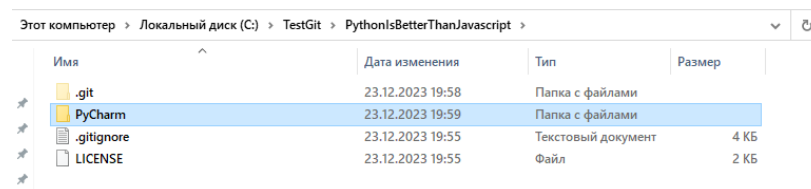


Рисунок 6.1 – Репозиторий с проектом PyCharm

7. Проработала примеры лабораторной работы.

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

def fun1(a):
    x = a * 3

    def fun2(b):
        nonlocal x
        return b + x

    return fun2

if __name__ == "__main__":
    test_fun = fun1(4)
    test_fun(7)
    print(test_fun)
```

Рисунок 7.1 – Код программы EXERCISE1.py в IDE PyCharm

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

def mul(a):
    def helper(b):
        return a * b

    return helper

if __name__ == "__main__":
    mul(5)(2)
```

Рисунок 7.2 – Код программы EXERCISE2.py в IDE PyCharm

8. Выполнила индивидуальное задание.

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

def replace_duplicates(replace_char):
    def inner_func(string):
        result = string[0]
        for i in range(1, len(string)):
            if string[i] != string[i-1]:
                result += string[i]
            else:
                result += replace_char
        return result

    return inner_func

if __name__ == "__main__":
    string_to_transform = input("Введите строку с "
                                "повторяющимися символами: ")

    said_char = input("Введите символ: ")

    replacement_func = replace_duplicates(said_char)
    transformed_string = replacement_func(string_to_transform)

    print(f"Исправленная строка: {transformed_string}")
```

Рисунок 8.1 – Код программы EXERCISE8.py в IDE PyCharm

9. Зафиксировала изменения в репозитории.

```
C:\TestGit\PythonIsBetterThanJavascript>git add PyCharm
C:\TestGit\PythonIsBetterThanJavascript>git commit -m"add files"
[develop 6a520c3] add files
4 files changed, 62 insertions(+)
create mode 100644 PyCharm/excerise/EXERCISE1.py
create mode 100644 PyCharm/excerise/EXERCISE2.py
create mode 100644 PyCharm/individual/request.py
create mode 100644 "PyCharm/individual/\320\222\320\260\321\200\320\270\320\260\320\275\321\2026.txt"
```

Рисунок 9.1 – Коммит файлов в репозитории git

Контрольные вопросы

1. Что такое замыкание?

Замыкание (closure) в программировании — это функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся ее параметрами.

2. Как реализованы замыкания в языке программирования Python?

Замыкание (closure) в Python — это функция, которая запоминает значения в окружающей (внешней) области видимости, даже если эта область видимости больше не существует. Замыкание возникает, когда внутри функции определены вложенные функции, и вложенная функция ссылается на переменные из внешней функции.

3. Что подразумевает под собой область видимости Local?

Эту область видимости имеют переменные, которые создаются и используются внутри функций

4. Что подразумевает под собой область видимости Enclosing?

Суть данной области видимости в том, что внутри функции могут быть вложенные функции и локальные переменные, так вот локальная переменная функции для ее вложенной функции находится в enclosing области видимости.

5. Что подразумевает под собой область видимости Global?

Переменные области видимости global – это глобальные переменные уровня модуля (модуль – это файл с расширением .py).

6. Что подразумевает под собой область видимости Build-in?

Уровень Python интерпретатора. В рамках этой области видимости находятся функции `open`, `len` и т. п., также туда входят исключения. Эти сущности доступны в любом модуле Python и не требуют предварительного импорта. `Builtin` – это максимально широкая область видимости.

7. Как использовать замыкания в языке программирования Python?

Использование замыканий в Python обычно связано с созданием функций внутри других функций и возвратом этих вложенных функций. Замыкания полезны, когда вам нужно передать часть контекста (переменные) в функцию, чтобы она могла использовать их даже после того, как внешняя функция завершила свою работу.

8. Как замыкания могут быть использованы для построения иерархических данных?

Замыкания в Python можно использовать для построения иерархических данных, таких как деревья или вложенные структуры. Замыкания позволяют сохранять состояние внешней функции внутри вложенной функции, что обеспечивает уровень иерархии.