1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. [Институт компьютерных наук и кибербезопасности](https://dl.spbstu.ru/)

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. **«Управление ФС/реестром посредством CLI-приложения на ЯП Python**»
2. по дисциплине «Цифровая культура»
3. Выполнил
4. студент гр. 5151004/30001 Лясович С.Д.

<*подпись*>

1. Преподаватель
2. асс. преподавателя Писков А.А.

<*подпись*>

1. Санкт-Петербург
2. 2025
3. **Цель работы**

Получение навыков управления файловой системой и реестром с использованием языка программирования Python.

1. **Постановка задачи**

Реализовать консольное приложение на языке программирования Python, которое будет поддерживать операции управления ФС и реестром.

1. **Описание решения**

В ходе лабораторной работы были установлены язык программирования Python и среда программирования PyCharm.

Далее необходимо было реализовать консольное приложение. Первым ltkjv были подключены модули, допустимые по заданию, и проинициализированы аргументы командной строки (рисунок 1).

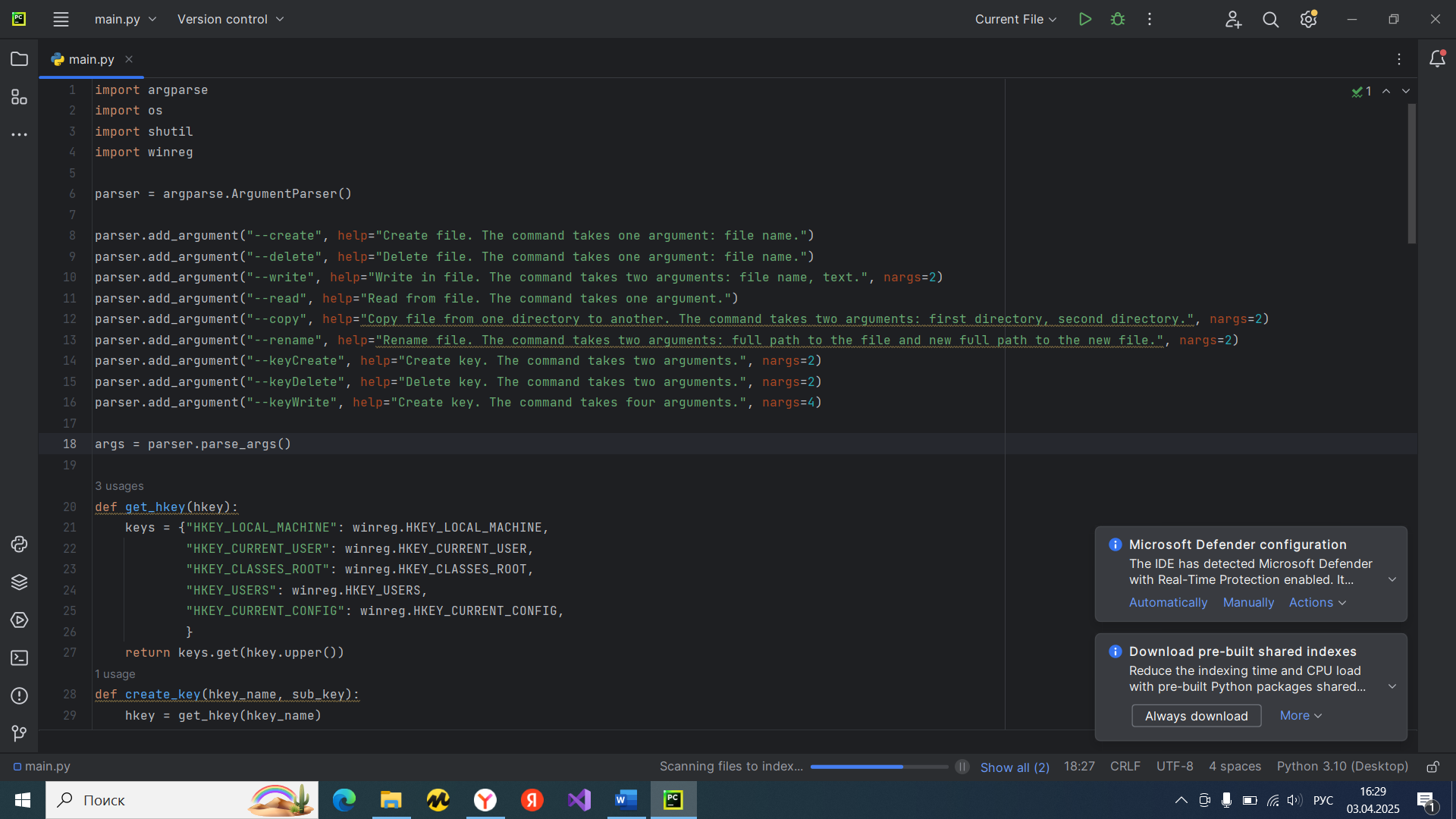


Рисунок 1 — Подключённые модули и проинициализированные аргументы командной строки

Далее, были написаны функции, отвечающие за работу с ключом реестра (рисунок 2).

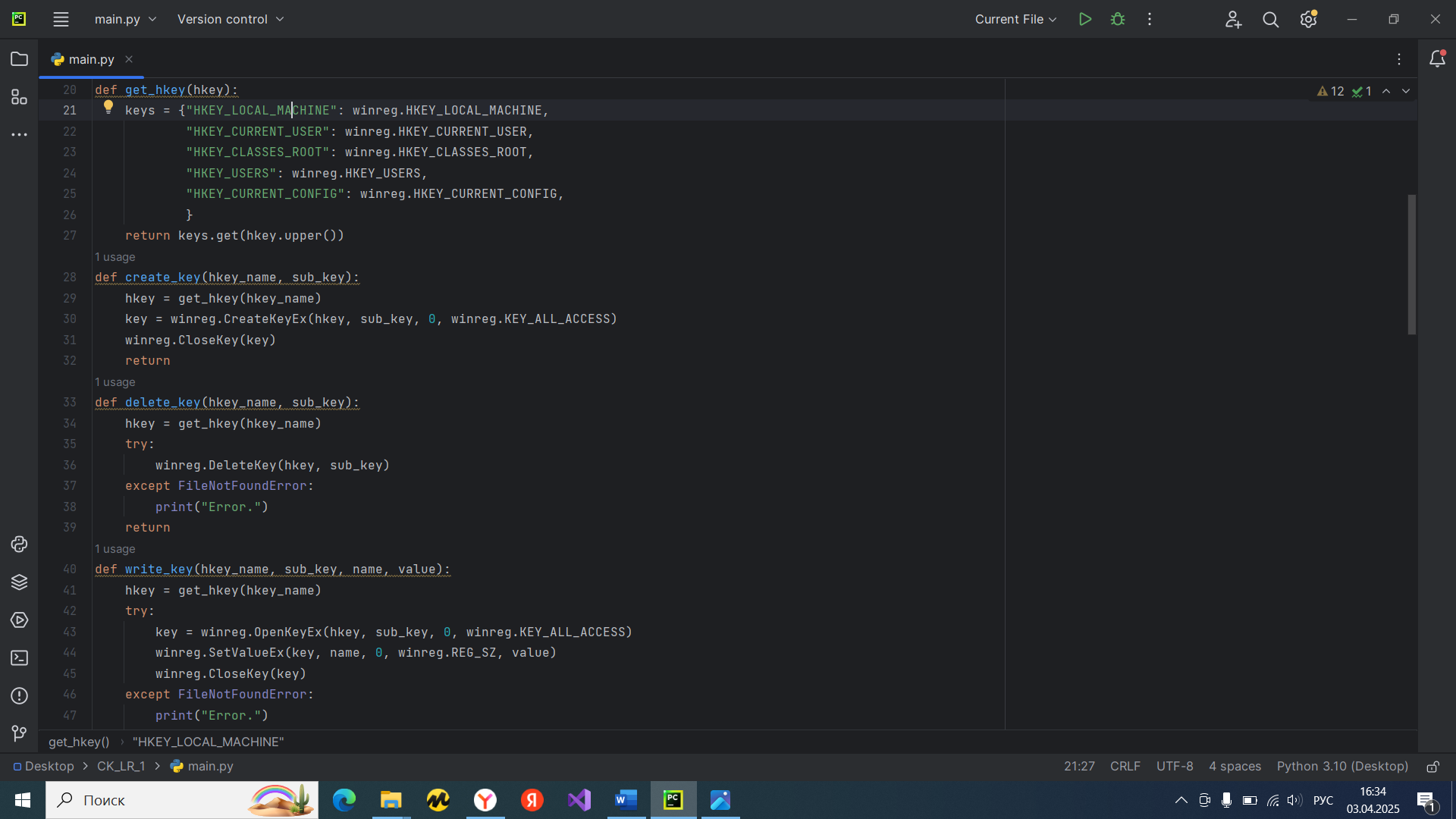


Рисунок 2 — Функции, отвечающие за работу с ключом реестра

На третьем этапе выполнения лабораторной работы, необходимо было реализовать функции работы с файлами, реализованные функции представлены на рисунках 3 и 4.

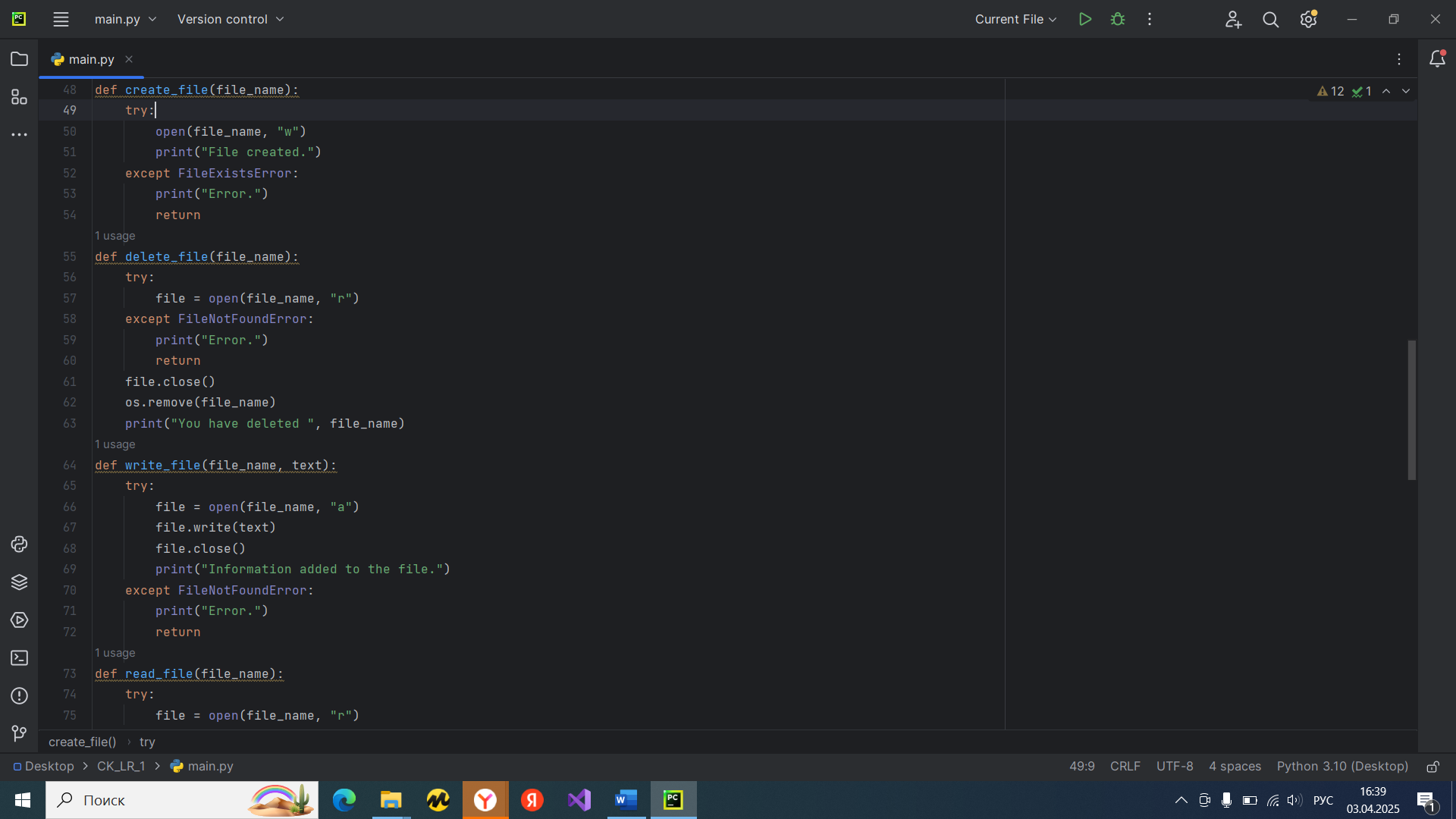


Рисунок 3 — Функции, отвечающие за работу с файлом часть 1

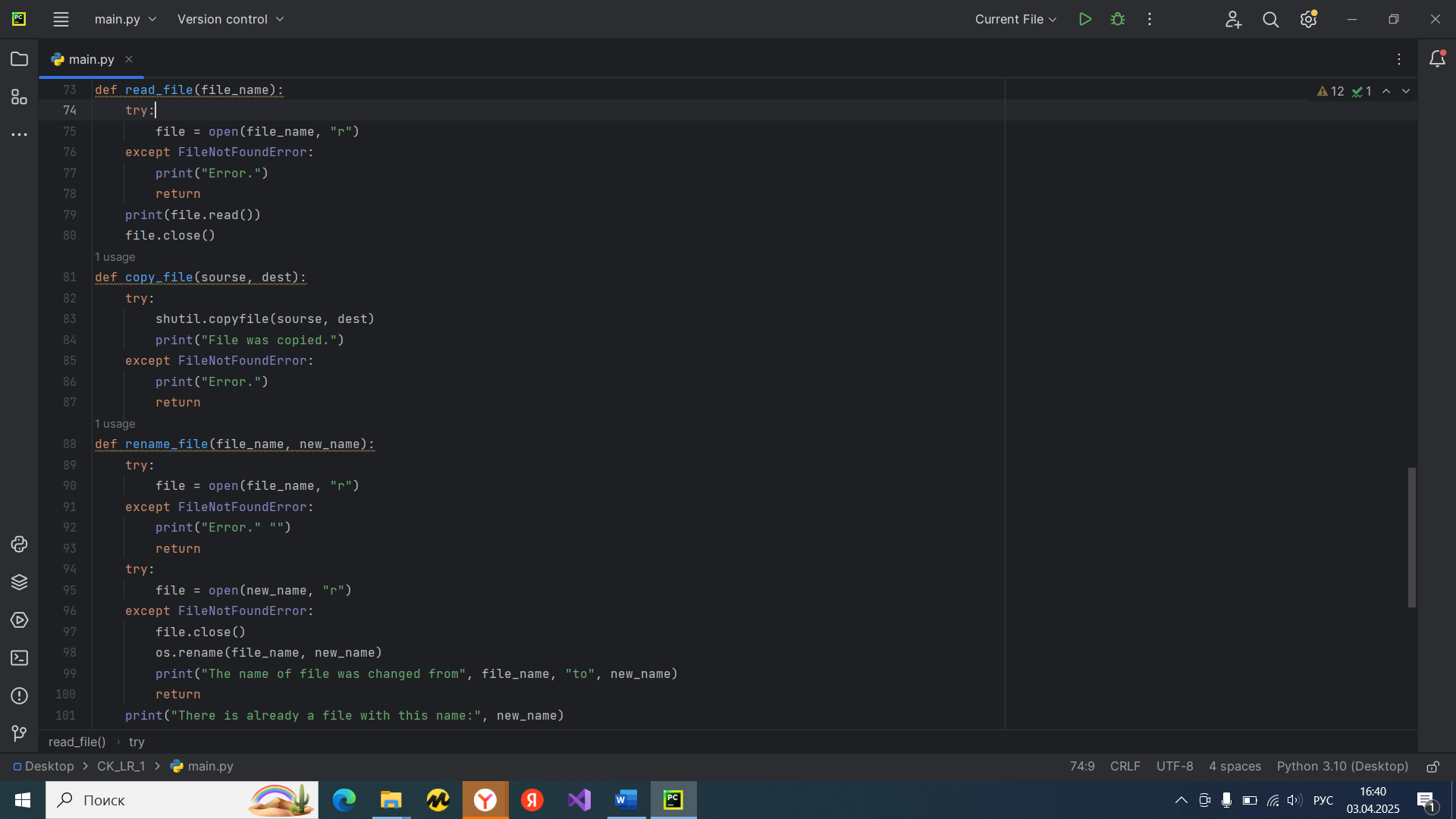


Рисунок 4 — Функции, отвечающие за работу с файлом часть 2

1. **Ответы на контрольные вопросы**
2. В компилируемых языках программирования обработка кода происходит путём преобразования программного кода в машинный код, когда в интерпретируемых языках программный код выполняется построчно во время запуска. Помимо этого, существуют различия в скорости: компилируемые языки работают быстрее интерпретируемых, и портативности: компилируемые языки менее портативны нежели интерпретируемые.

К компилируемым языкам программирования можно отнести C, C++, Go, Rust, Pascal, Fortran, а к интерпретируемым Python, JavaScript, Ruby, PHP, Bash.

1. Помимо использования модуля argparse, входные данные для консольного приложения могут быть считаны через интерактивный ввод (input()); через стандартный ввод (stdin); через API-запросы (HTTP, WebSocket); через файлы.
2. Разница между конструкцией «open()...close()» и командой «with open() as» в Python заключается в безопасности управления файловыми ресурсами. Так как «open() + close()» требует ручного закрытия файла, то существует вероятность того, что между «open()» и «close()» произойдёт ошибка, следовательно файл останется открытым, что может привести к утечке ресурсов. Когда, при использовании команды «with open() as» файл закрывается автоматически , даже если возникла ошибка.
3. Установка модулей для Python может происходить через: pip — стандартный менеджер пакетов Python; из исходного кода (GitHub, локальные файлы); вручную; в виртуальное окружение.
4. **Выводы**

В данной лабораторной работе были получены навыки управления файловой системой и реестром с использованием языка программирования Python.

ПРИЛОЖЕНИЕ А

Листинг реализованной программы

import argparse

import os

import shutil

import winreg

parser = argparse.ArgumentParser()

parser.add\_argument("--create", help="Create file. The command takes one argument: file name.")

parser.add\_argument("--delete", help="Delete file. The command takes one argument: file name.")

parser.add\_argument("--write", help="Write in file. The command takes two arguments: file name, text.", nargs=2)

parser.add\_argument("--read", help="Read from file. The command takes one argument.")

parser.add\_argument("--copy", help="Copy file from one directory to another. The command takes two arguments: first directory, second directory.", nargs=2)

parser.add\_argument("--rename", help="Rename file. The command takes two arguments: full path to the file and new full path to the new file.", nargs=2)

parser.add\_argument("--keyCreate", help="Create key. The command takes two arguments.", nargs=2)

parser.add\_argument("--keyDelete", help="Delete key. The command takes two arguments.", nargs=2)

parser.add\_argument("--keyWrite", help="Create key. The command takes four arguments.", nargs=4)

args = parser.parse\_args()

def get\_hkey(hkey):

keys = {"HKEY\_LOCAL\_MACHINE": winreg.HKEY\_LOCAL\_MACHINE,

"HKEY\_CURRENT\_USER": winreg.HKEY\_CURRENT\_USER,

"HKEY\_CLASSES\_ROOT": winreg.HKEY\_CLASSES\_ROOT,

"HKEY\_USERS": winreg.HKEY\_USERS,

"HKEY\_CURRENT\_CONFIG": winreg.HKEY\_CURRENT\_CONFIG,

}

return keys.get(hkey.upper())

def create\_key(hkey\_name, sub\_key):

hkey = get\_hkey(hkey\_name)

key = winreg.CreateKeyEx(hkey, sub\_key, 0, winreg.KEY\_ALL\_ACCESS)

winreg.CloseKey(key)

return

def delete\_key(hkey\_name, sub\_key):

hkey = get\_hkey(hkey\_name)

try:

winreg.DeleteKey(hkey, sub\_key)

except FileNotFoundError:

print("Error.")

return

def write\_key(hkey\_name, sub\_key, name, value):

hkey = get\_hkey(hkey\_name)

try:

key = winreg.OpenKeyEx(hkey, sub\_key, 0, winreg.KEY\_ALL\_ACCESS)

winreg.SetValueEx(key, name, 0, winreg.REG\_SZ, value)

winreg.CloseKey(key)

except FileNotFoundError:

print("Error.")

def create\_file(file\_name):

try:

open(file\_name, "w")

print("File created.")

except FileExistsError:

print("Error.")

return

def delete\_file(file\_name):

try:

file = open(file\_name, "r")

except FileNotFoundError:

print("Error.")

return

file.close()

os.remove(file\_name)

print("You have deleted ", file\_name)

def write\_file(file\_name, text):

try:

file = open(file\_name, "a")

file.write(text)

file.close()

print("Information added to the file.")

except FileNotFoundError:

print("Error.")

return

def read\_file(file\_name):

try:

file = open(file\_name, "r")

except FileNotFoundError:

print("Error.")

return

print(file.read())

file.close()

def copy\_file(sourse, dest):

try:

shutil.copyfile(sourse, dest)

print("File was copied.")

except FileNotFoundError:

print("Error.")

return

def rename\_file(file\_name, new\_name):

try:

file = open(file\_name, "r")

except FileNotFoundError:

print("Error." "")

return

try:

file = open(new\_name, "r")

except FileNotFoundError:

file.close()

os.rename(file\_name, new\_name)

print("The name of file was changed from", file\_name, "to", new\_name)

return

print("There is already a file with this name:", new\_name)

return

if args.create:

create\_file(args.create)

elif args.delete:

delete\_file(args.delete)

elif args.write:

write\_file(args.write[0], args.write[1])

elif args.read:

read\_file(args.read)

elif args.copy:

copy\_file(args.copy[0], args.copy[1])

elif args.rename:

rename\_file(args.rename[0], args.rename[1])

elif args.keyCreate:

create\_key(args.keyCreate[0], args.keyCreate[1])

elif args.keyDelete:

delete\_key(args.keyDelete[0], args.keyDelete[1])

elif args.keyWrite:

write\_key(args.keyWrite[0], args.keyWrite[1], args.keyWrite[2], args.keyWrite[3])