**Лабораторная работа 1. Файловый менеджер**

**Цель лабораторной работы**

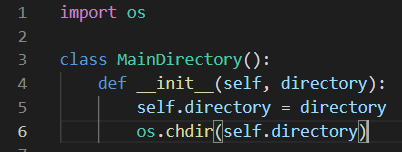
Научиться создавать сложный проект и научиться программно работать с локальными файлами и папками

СНАЧАЛА У МЕНЯ ИДЁТ ПОЯСНЕНИЕ К ПРОГРАММЕ, А ПОТОМ УЖЕ ВСЯ РЕАЛИЗАЦИЯ

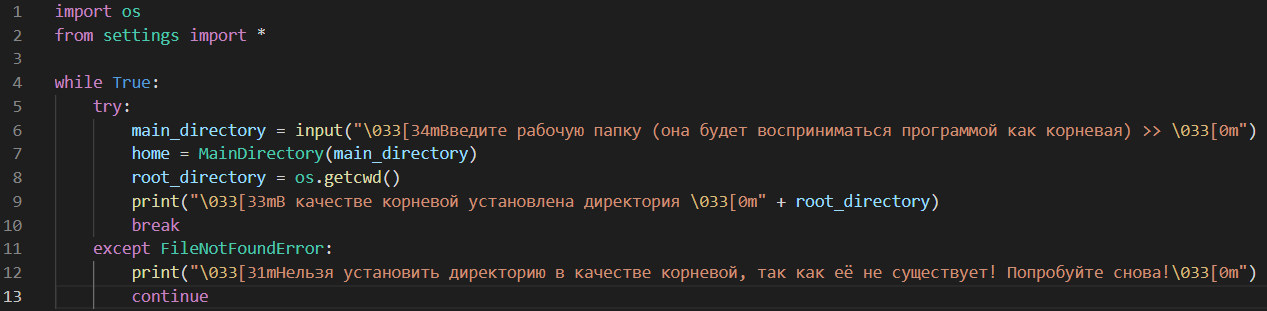
**Указания к выполнению**

1. Расположение рабочей папки должно указываться в настройках файлового менеджера. Настройки должны располагаться в отдельном от основного исходного кода файле.

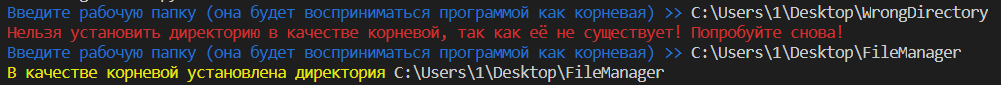
Файл настроек – это settings.py:



Здесь мы пытаемся инициализировать рабочую директорию в ту, которую вводит пользователь (6 строчка). Либо всё получается, и это срабатывает, либо кидается исключение, которое обрабатывается в main.py, где я пускаю бесконечный цикл до того момента, пока не получится установить корневую директорию, при этом я использую импорт файла settings.py как модуля:



Итог (что видит пользователь консольного менеджера):



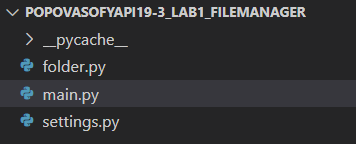
1. Файловый менеджер должен блокировать пользователя от выхода за пределы рабочей папки. Пользователь должен воспринимать рабочую папку как корневую и все действия файлового менеджера должны локализоваться только в пределах этой папки.

Так и есть, вот что будет, если мы захотим перейти за пределы рабочей директории:



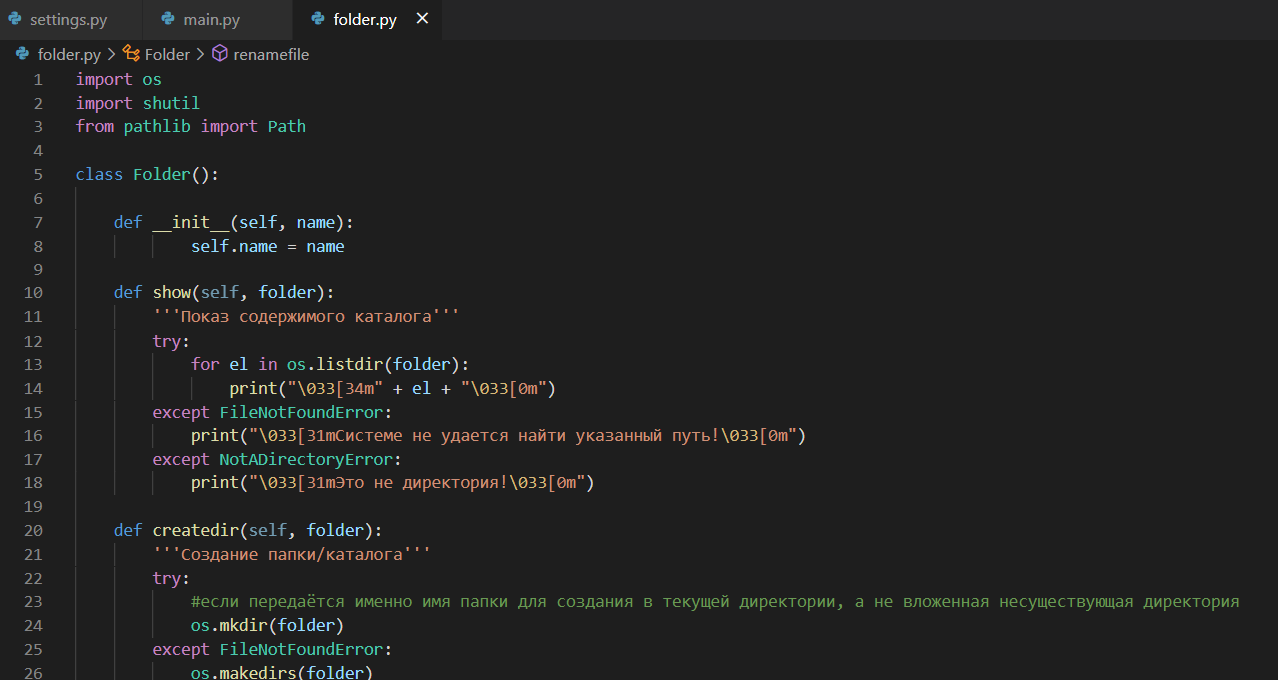
1. Программный проект должен быть оформлен как код на языке программирования Python и располагаться в определенной папке. Проект должен состоять из нескольких файлов. Расположение рабочей папки не должно быть связано с физическим расположением файлов исходного кода.

Все условия выполняются, вот структура моего проекта:

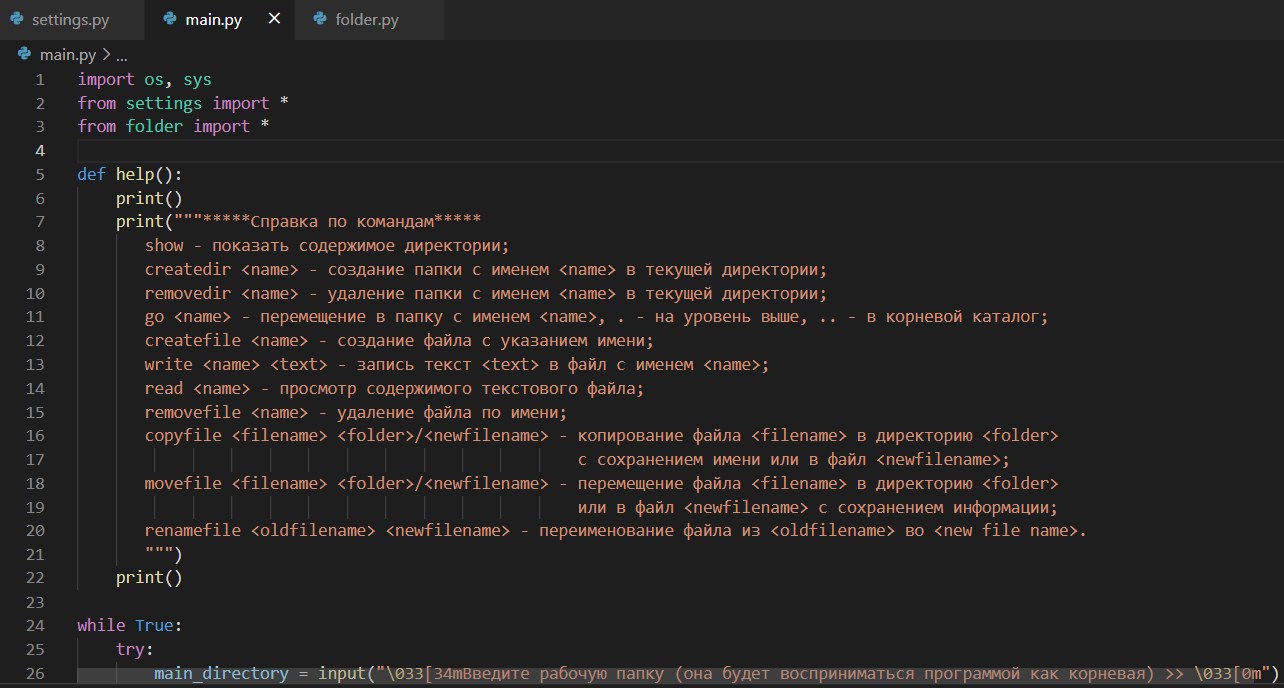


Как выглядят файлы:

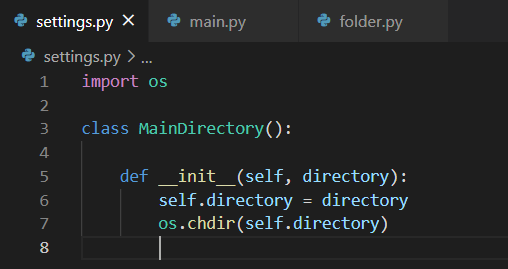
folder.py (здесь инициализируется директория, с которой я работаю, она же выводится как консоль):



main.py (здесь идёт вызов всех функций и методов):

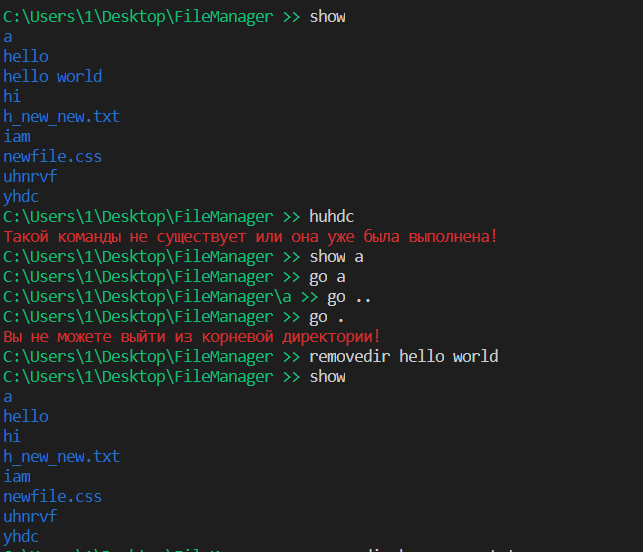


settings.py (изначальная установка рабочего каталога):



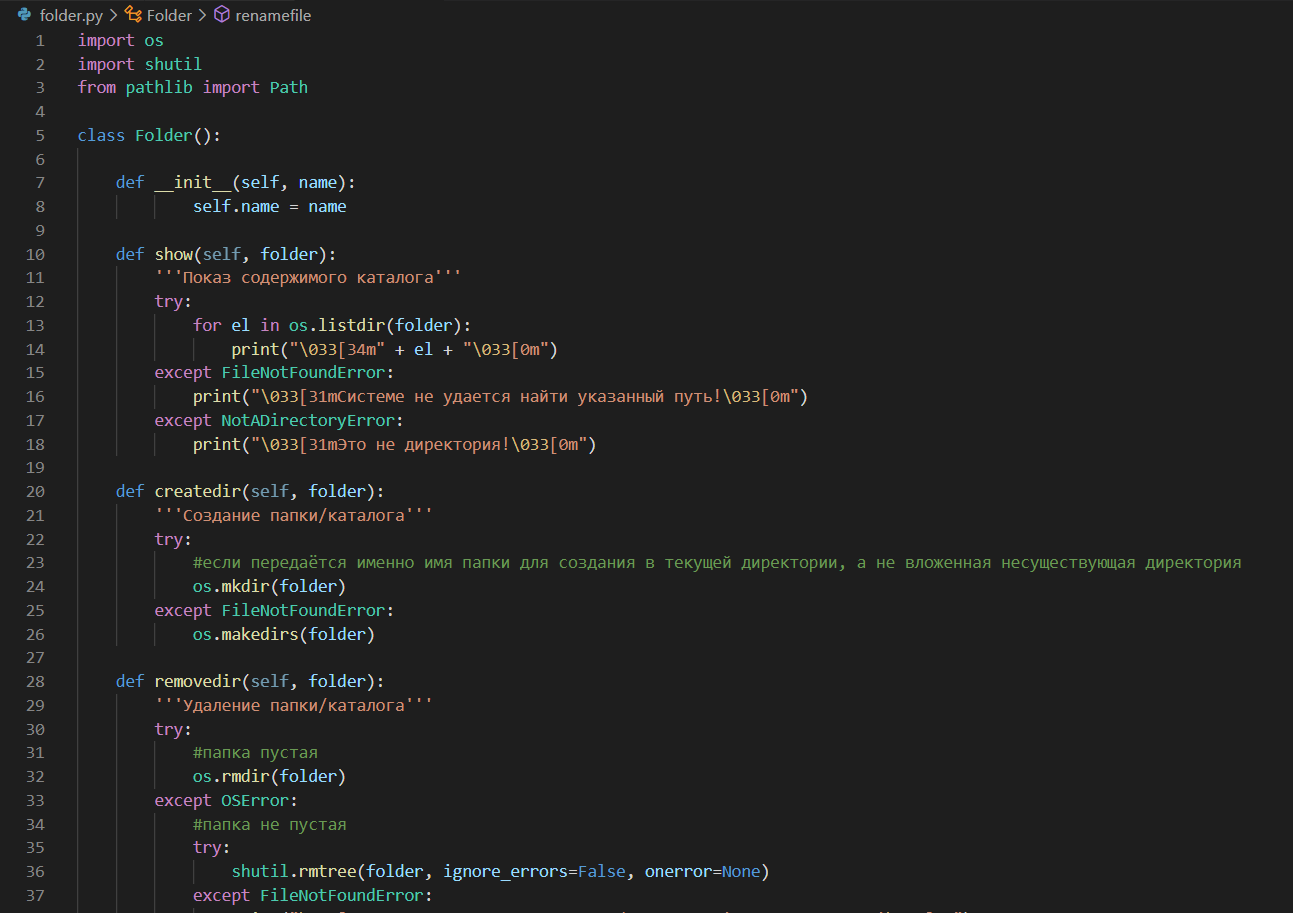
1. Файловый менеджер по умолчанию должен иметь текстовый интерфейс по аналогии с интерфейсом командной строки. Действия пользователя осуществляются вводом с клавиатуры соответствующей команды (по необходимости с параметрами).

Как мою программу видит пользователь:



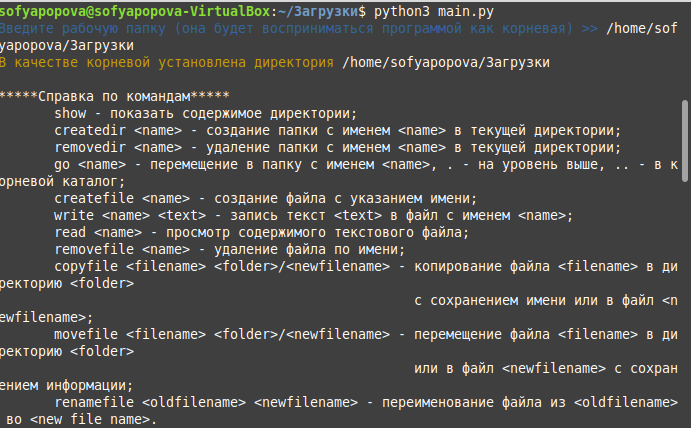
1. Код должен быть организован в виде набора функций или классов, каждая операция файлового менеджера должна быть реализована в отдельной функции или методе класса.

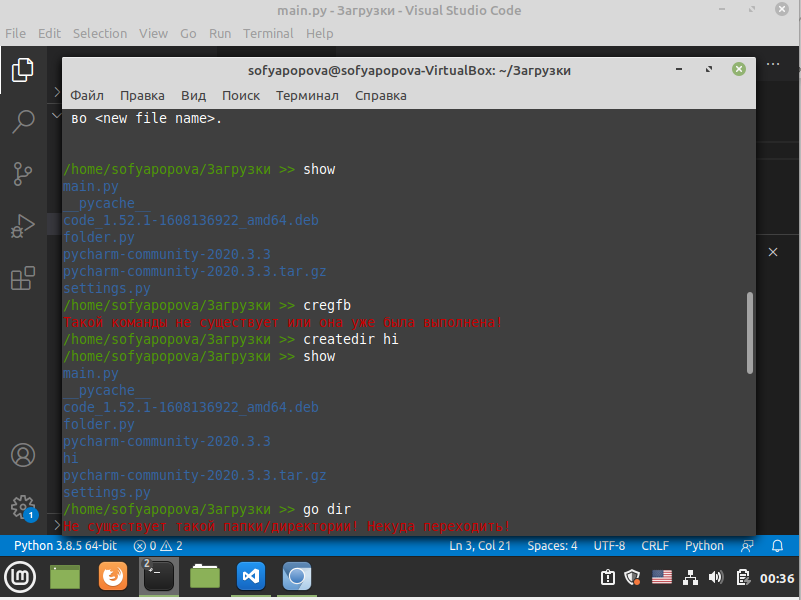
Так и есть, мой файл folder.py, в котором содержится класс Folder, так и выглядит:

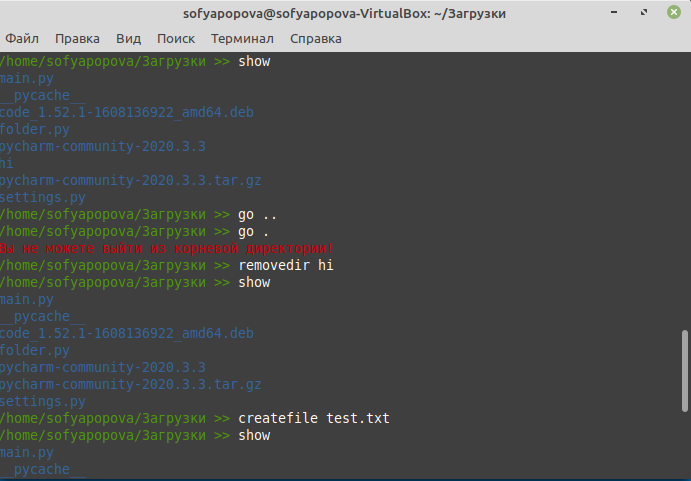


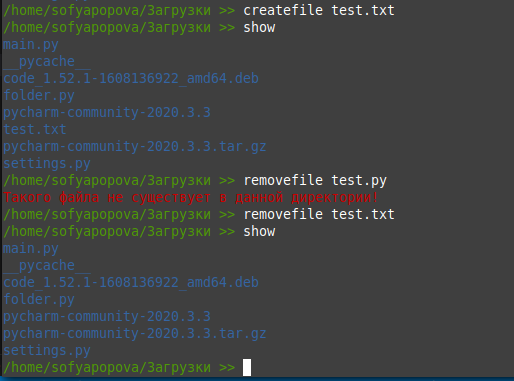
1. Файловый менеджер должен быть кроссплатформенным и работать как в среде Windows, так и в UNIX системах. Необходимо протестировать работоспособность программы в разных ОС. Для кроссплатформенности рекомендуется использовать стандартную библиотеку Python для осуществления файловых операций.

Тестирую на линуксе:







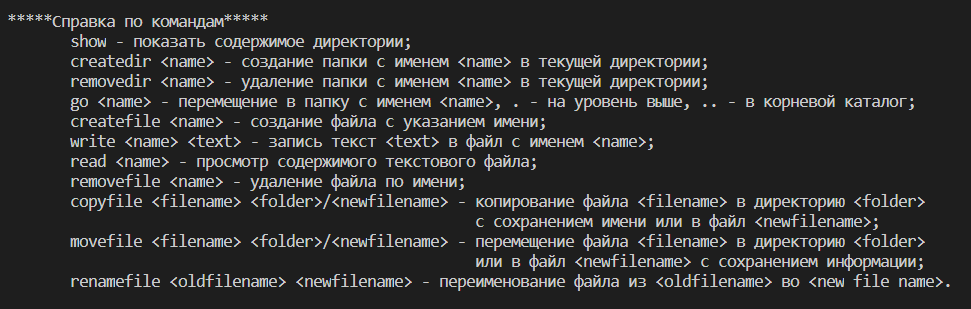


1. Разработка программы должна вестись с использованием СКВ Git. Код должен публиковаться в репозитории на GitHub.

Ссылка на гитхаб: https://github.com/alexsp63/Practicum

1. Перед разработкой программист должен продумать названия и структуру команд для пользователя. Команды не должны повторять команды существующих программных оболочек.

Странное требование, но ок, мои не повторяют:



**Задания для выполнения**

Необходимо создать примитивный файловый менеджер. Программа должна

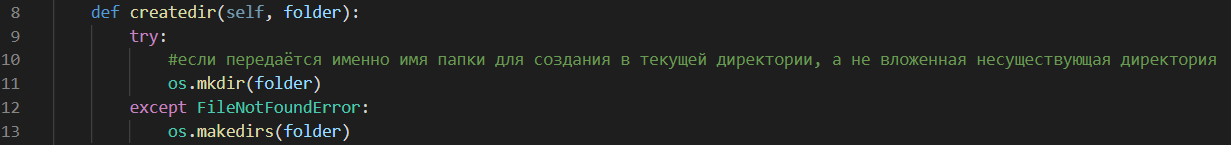
работать в определенной папке (рабочей папки менеджера) и позволять

пользователю выполнять следующие простые действия в пределах рабочей

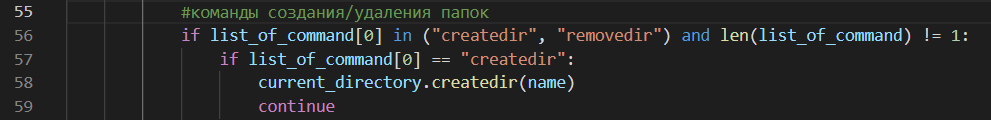
папки:

1. Создание папки (с указанием имени);

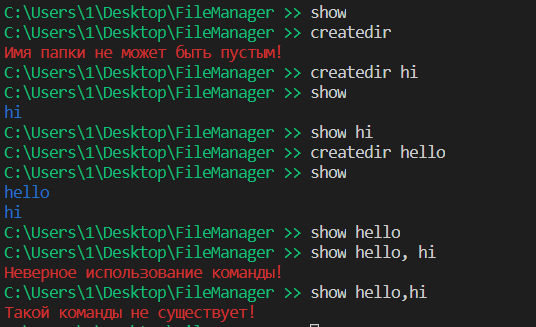
Метод в классе Folder:

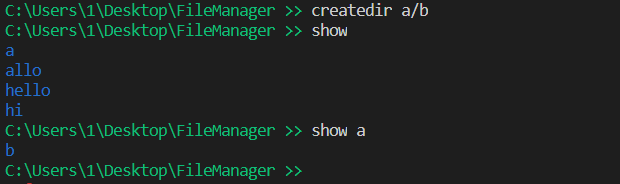


Вызов этого метода в main (+ в исходном коде прописаны все исключения):



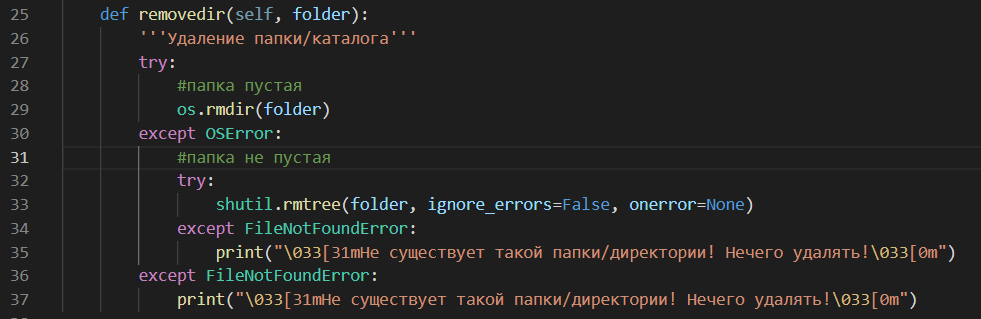
Итог:



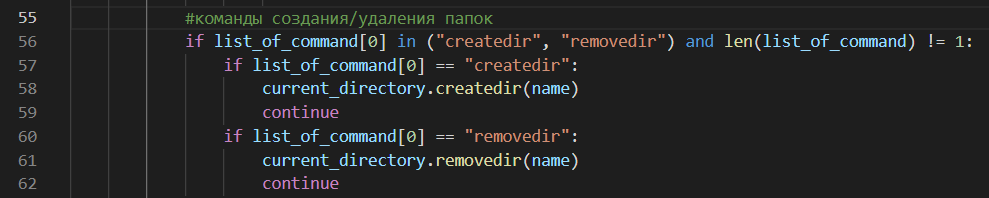


2. Удаление папки по имени;

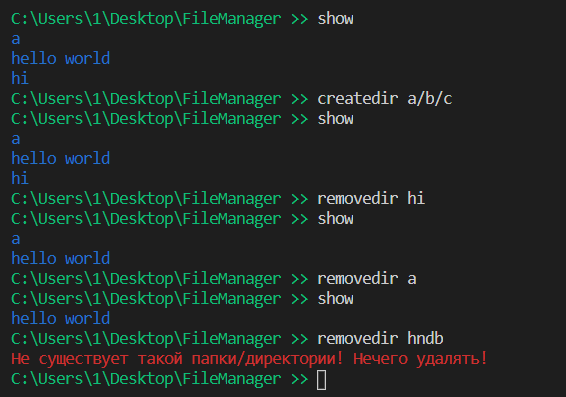
Метод в классе Folder (рассмотрены все случаи):



В main вызывается аналогично предыдущему:



Результат работы:

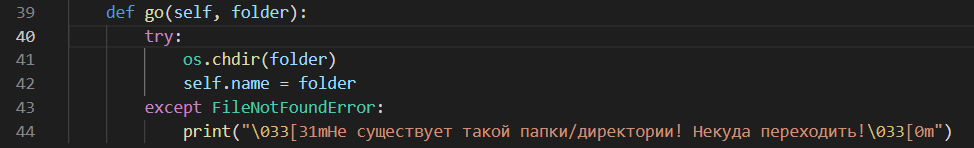


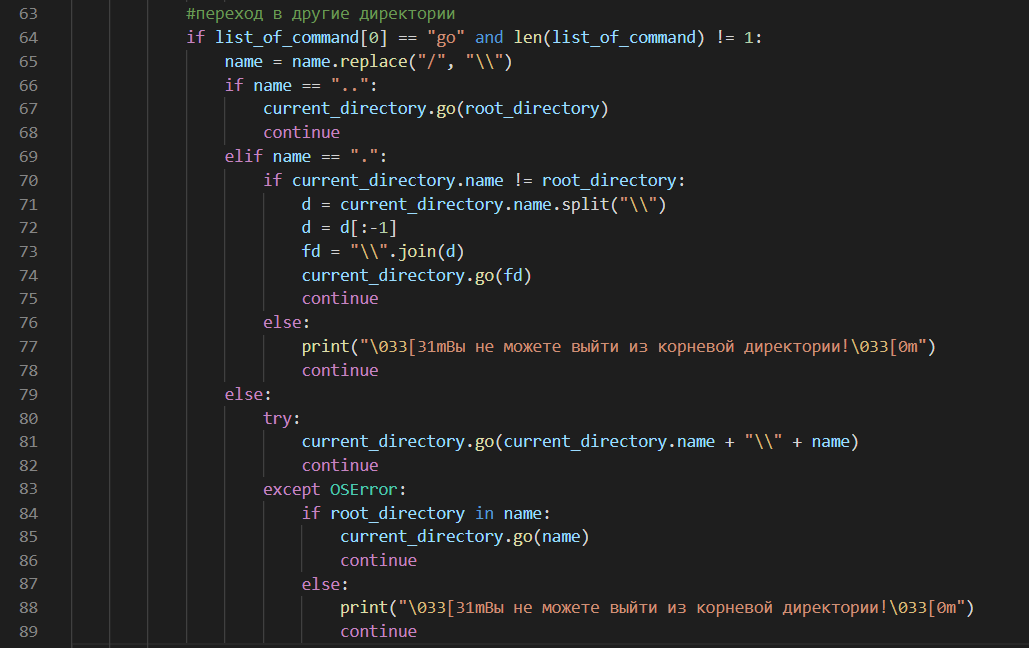
3. Перемещение между папками (в пределах рабочей папки) - заход в папку по имени, выход на уровень вверх;

Также рассмотрела все случаи, больше всего проверок было на то, что нельзя выйти из рабочей/корневой папки:

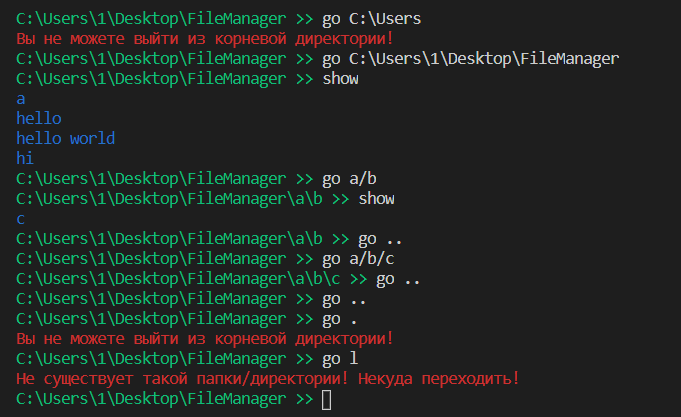
Код метода:

Вызов в main:





И вот так это работает:



4. Создание пустых файлов с указанием имени;

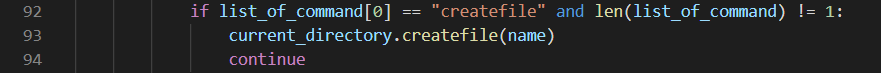
Для создания пустых файлов по имени я добавила импорт в свой файл folder.py:



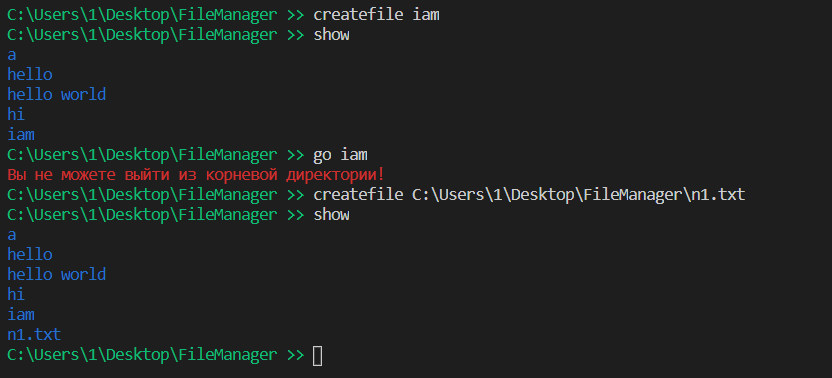
И метод:



А также его вызов:

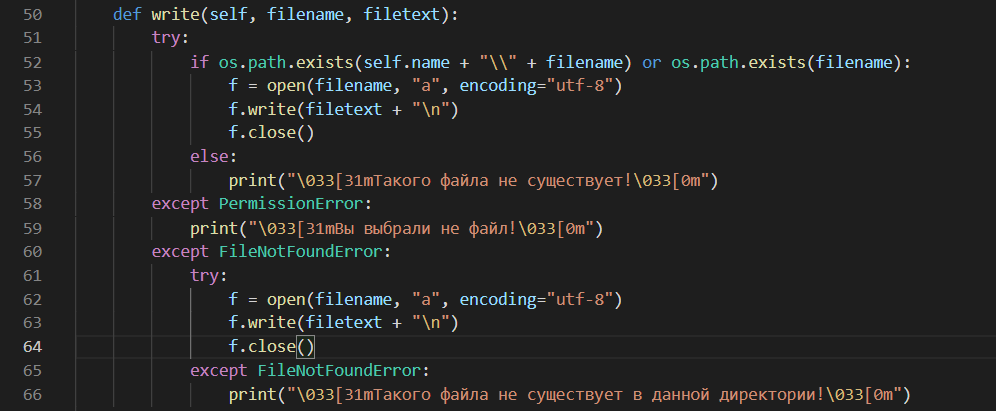


Итог, как это выглядит со стороны пользователя:



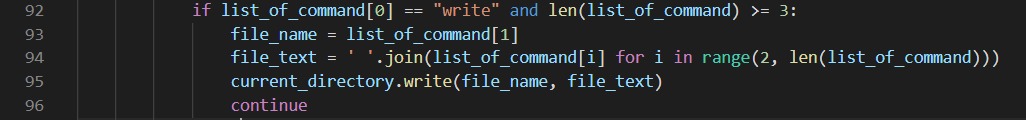
5. Запись текста в файл;

Для начала метод в классе:

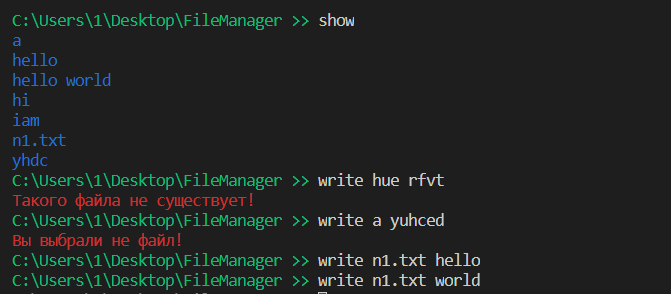


Проверка в 52 строчке стоит, потому что я хочу, чтобы моя программа воспринимала как относительный, так и абсолютный путь, а также обработка исключения на случай, если пользователь выберет не файл, а, например, директорию.

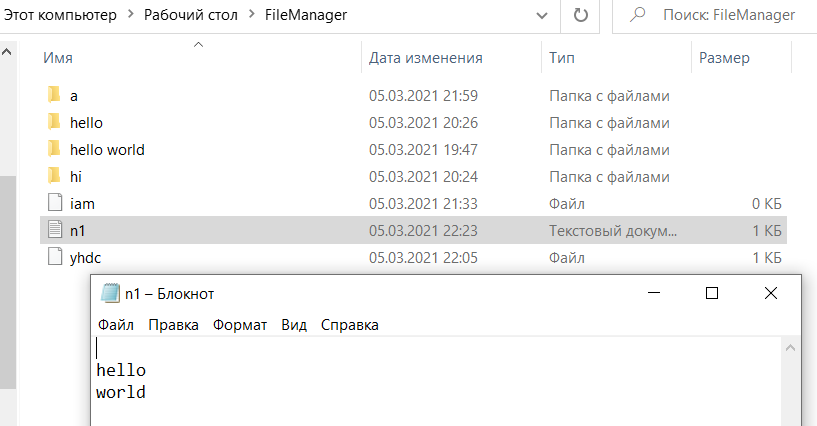
Вызов в main.py:



Теперь проверю, как это всё работает:



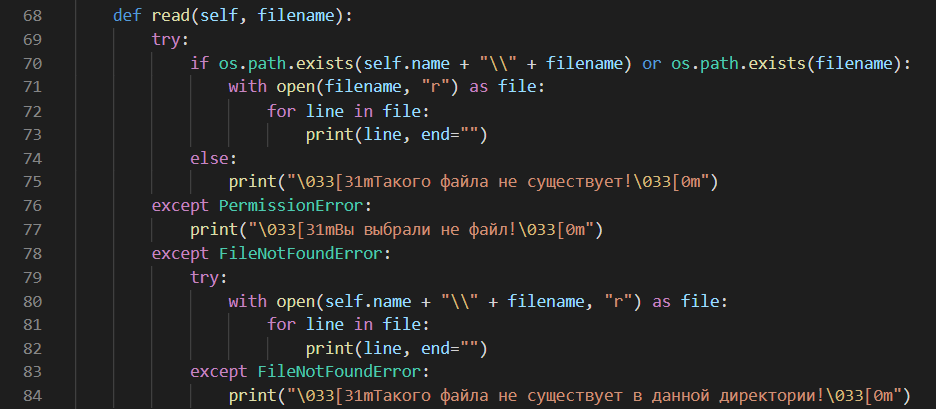
Реализация просмотра содержимого идёт только следующим пунктом, так что пока можно посмотреть содержимое путём открытия его через проводник:



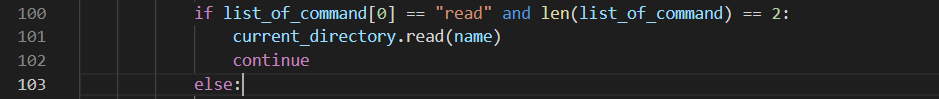
Как можно увидеть, всё хорошо записалось.

6. Просмотр содержимого текстового файла;

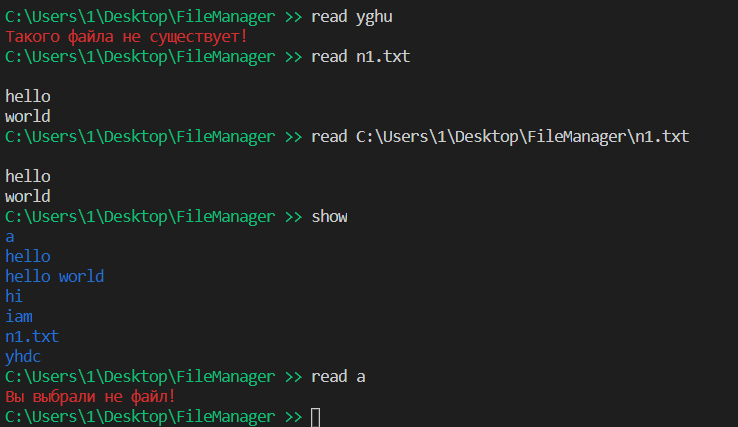
Метод в классе, читаю файл красиво по строкам:



Вызов в main.py:



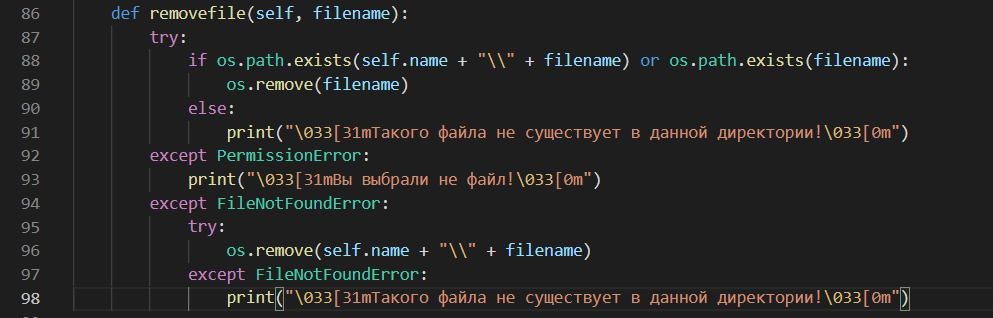
И результат работы:



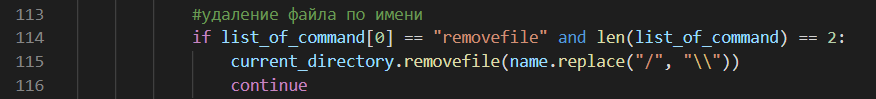
Как можно заметить, содержимое файла совпадает с тем, которое было показано при открытии файла через блокнот.

7. Удаление файлов по имени;

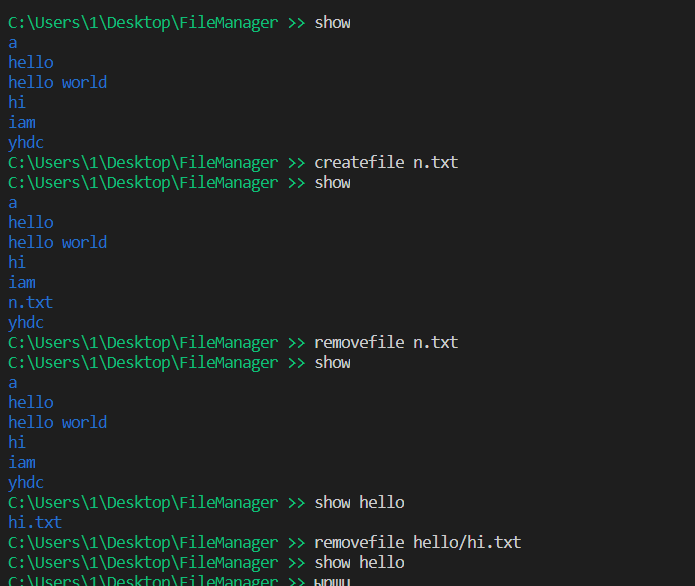
Метод для удаления:



Вызов:

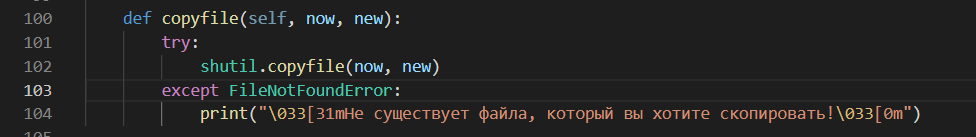


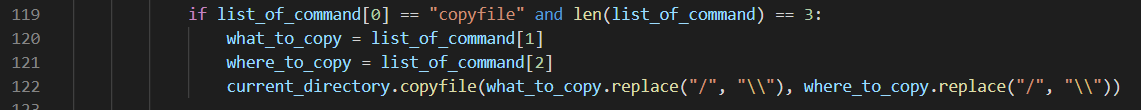
Результат работы:



8. Копирование файлов из одной папки в другую;

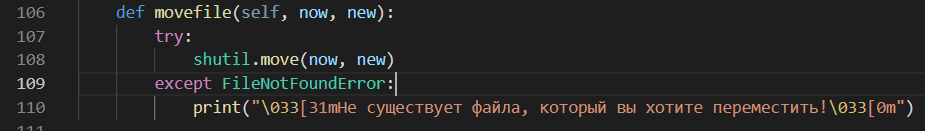
Скриншоты по аналогичной схеме – метод, вызов, вывод.

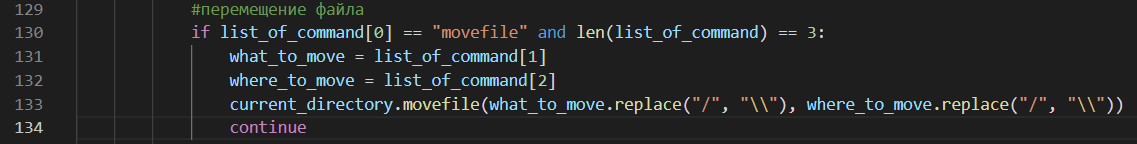


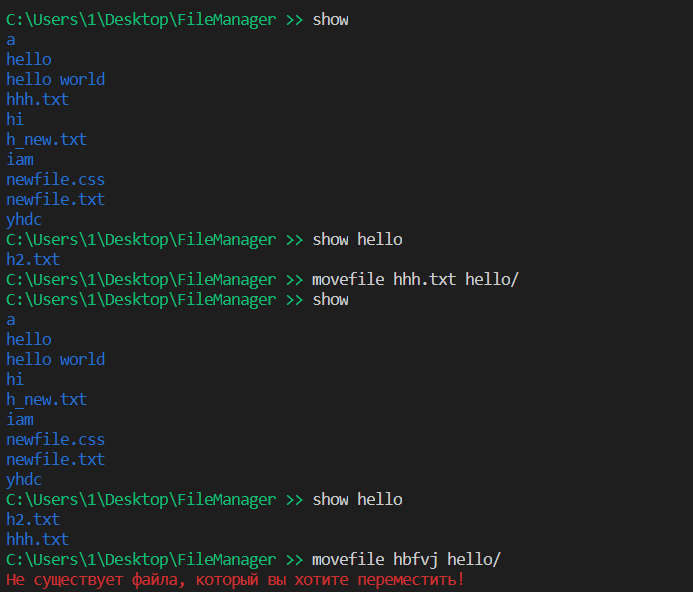




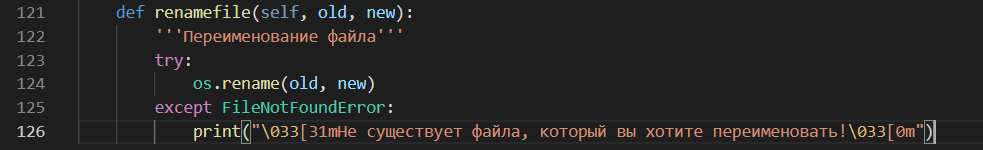
9. Перемещение файлов;

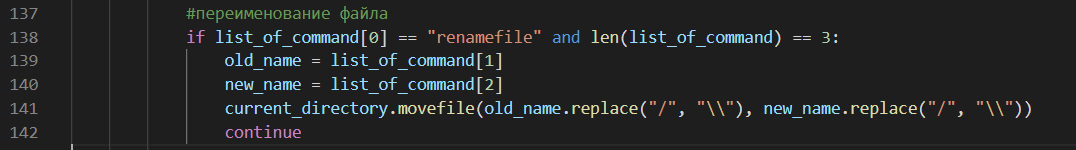


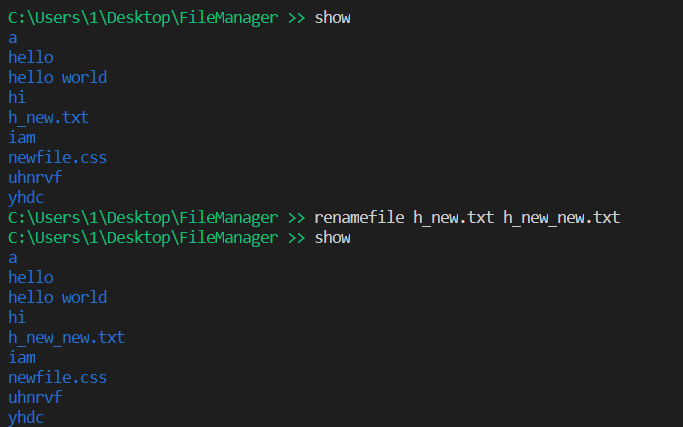




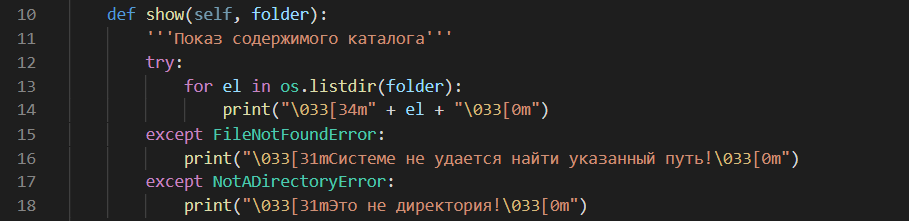
10. Переименование файлов.

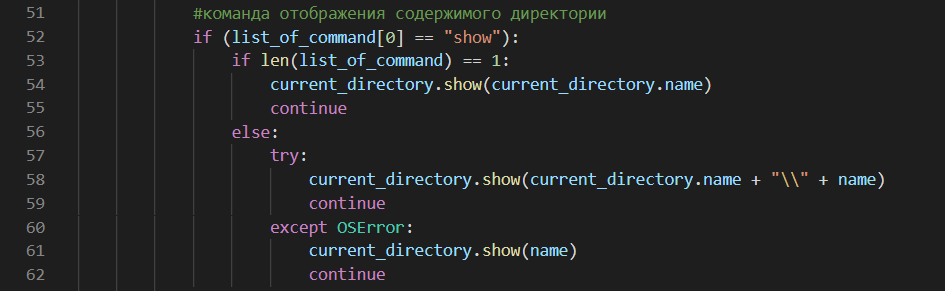






**НУ И ОТ СЕБЯ Я ДОБАВИЛА МЕТОД show – показ содержимого каталога, КОТОРЫЙ Я АКТИВНО ИСПОЛЬЗОВАЛА В ТЕЧЕНИЕ ВСЕГО ТЕСТИРОВАНИЯ ПРОГРАММЫ:**

****

****