## Description

The previous stage was a good start. However, we need directories to store something... and there is not much use in browsing them when you can't even see their content. Let's fix that.

In this stage, we will implement a new command to get quick info about the state of the current directory. The name of the command is `ls`. The user can use `ls` in three different ways:

- `ls` prints the list of files and subdirectories in the current directory.
- `ls -l` prints the list of subdirectories and the list of files, and for each file, it specifies its size in bytes.
- `ls -lh` outputs the size of files in a human-readable format. The size of each file can be indicated not only in bytes but also in kilobytes, megabytes, or gigabytes. This way, it'll be much more convenient to read the list of files and find the largest one in the folder, in case you need that!

## Objectives

In this stage, your program should:

- Support all the commands from the previous stage. The interaction between the user and the program remains the same: the user inputs a command, the program outputs the result, accepts a new command, and so on.
- Provide a new command — `ls` . If the user inputs `ls` without arguments, print the list of subdirectories and files in the current directory.
    - The program should print the names of the files together with extensions. For example, `test.txt` (instead of just `test`)
    - The list of subdirectories should precede the list of files.
    - The order of items in these lists is irrelevant.
    - The program shouldn't output anything if there are no files and subdirectories.
- If the user inputs `ls -l`, the program should output the same lists. However, the list of files should also contain the size of each file **in bytes**. This information can be found via `os.stat(filename).st_size`. The name of the file and its size should be separated by one space character.
- If the user inputs `ls -lh`, the program should output the same lists together with the sizes of the files. However, the size should be printed in human-readable format:
    - the size should be converted to bytes, kilobytes, megabytes, or gigabytes, depending on the file size. Then, the converted size and the unit should be printed. For example, `100MB` or `1GB`.
    - See the table below for details:

| Size of the file | Output format | Example of the output |
| --- | --- | --- |
| < 1024 bytes (< 1 KB) | `<size>B` | `608B`, `900B` |
| 1 KB (1024 B) <= size < 1 MB (1024 KB) | `<size>KB` | `600KB` |
| 1 MB (1024 KB) <= size < 1 GB (1024 MB) | `<size>MB` | `550MB` |
| >= 1 GB (1024 MB) | `<size>GB` | `2GB` |

Note that the number of bytes, kilobytes, megabytes, and gigabytes should be integer. If the calculated size of the file is not an integer, round it to the nearest integer.

# Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

### Example 1:

```
Input the command
> pwd
C:\Users\User
> cd ..
Users
> ls
User
mouse.jpg
> cd User
User
> pwd
C:\Users\User
> quit
```

### Example 2:

```
Input the command
> pwd
C:\Users\User\PycharmProjects\File Manager
> cd ..
PycharmProjects
```

```
> ls
Hangman
Search Engine
Spell Checker
test.txt
readme.md
image.png
> ls -l
Hangman
Search Engine
Spell Checker
test.txt 47302
image.png 5767168
movie.mp4 2684354560
> ls -lh
Hangman
Search Engine
Spell Checker
test.txt 46KB
image.png 5MB
movie.mp4 2GB
> cd Hangman
Hangman
> ls -l
readme.md 0
> ls -lh
readme.md 0B
> quit
```