Programming (COMP4008-PRG)
4th exercise

Thorsten Altenkirch and Isaac Triguero

November 16, 2021

This exercise is about object oriented programming in Python. Your task is to use objects to implement a representation of a simple file system. In computer science, a file system is a data structure that the Operating System uses to control how data is stored and retrieved from a hard (or solid state) drive. A file system could be represented as a tree, in which files and directories may be organised as a hierarchy, so that, directories may contain subdirectories. For example, a file `gatos.jpg` may be under `Isaac` directory, but that is a subdirectory of the home folder. In file systems, a directory is just a file, but it is a special kind of file that contains a list of files. Here is an example:

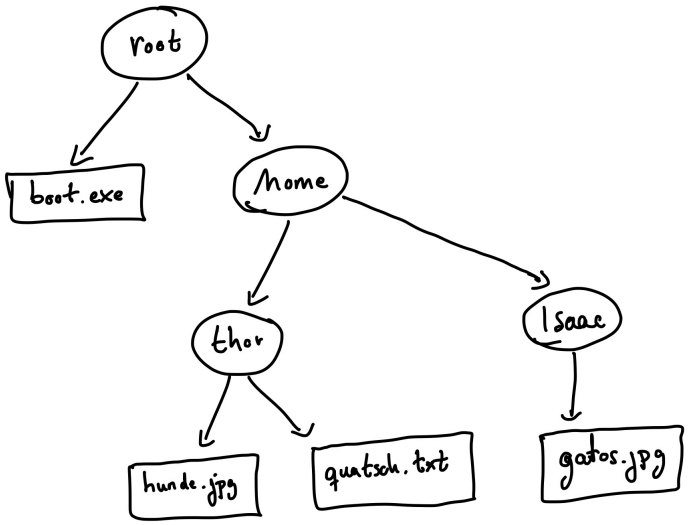

Figure 1: An example of a file system as a tree. Directories are represented in circles, plain files in squared boxes

We have split this coursework into a number of questions (adding up to 100 marks). Test your code with the test in `test-fs.py` which can be downloaded from Moodle.

1. Implement the following classes to represent a simple file system:

   `File:` everything is a file

   `PlainFile:` a plain file has a name (we are ignoring its contents)

   `Directory:` has a name and a list of files

   with constructors (i.e. `__init__` methods), so that, we can define the following directory tree (i.e. file system) based on the image above:

```
>> root = Directory("root",
              [PlainFile("boot.exe"),
               Directory("home",[
                   Directory("thor",
                      [PlainFile("hunde.jpg"),
                       PlainFile("quatsch.txt")]),
                   Directory("isaac",  [PlainFile("gatos.jpg")])])])
```

   [10 marks]

2. Add methods that print (recursively) the entire file system tree (i.e. define appropriate `__str__` methods). You don't need to produce a pretty output - it is ok if it looks like this:

```
Directory(root,[PlainFile(boot.exe),Directory(home,
[Directory(thor,[PlainFile(hunde.jpg),PlainFile(
quatsch.txt)],Directory(isaac,[PlainFile(gatos.jpg)]]]
```

   [10 marks]

3. A File may have other attributes such as `owner`. If not indicated, the `owner` will be set to *"default"*. Implement a method `chown(new_owner)` that will allow you to modify the owner of a file or directory.

```
>> file = PlainFile("boot.exe")
>> folder = Directory("Downloads",[])
>> print(f'file.owner: {file.owner}; folder: {folder.owner}')
file.owner: default; folder: default
>> file.chown("root")
>> folder.chown("isaac")
>> print(f'file.owner: {file.owner}; folder: {folder.owner}')
file.owner: root; folder: isaac
```

   [5 marks]

4. Implement a method `ls()` that recursively prints the content of the directory and all the subdirectories, using indentation to represent how deep in the tree structure the file/directory is.

```
# if we run ls() on the previous object root:
>> root.ls()
root
   boot.exe
   home
      thor
         hunde.jpg
         quatsch.txt
      isaac
         gatos.jpg
```

[10 marks]

5. Implement a new class, `FileSystem` , which will allow us to navigate and manipulate a file system as if it was a UNIX file system. In particular, this class will allow us to keep track of the *working directory*. It should be initialised as follows:

```
>> fs = FileSystem(root)
```

   (a) Implement a method `pwd()` that tells you the current working directory. This might be useful later when moving across directories. It should work like this:

```
>> fs.pwd()
'root'
```

[5 marks]

   (b) Implement `ls()` for the `FileSystem` class, so that, `fs.ls()` would work as question 4, but only printing from the current directory.

[5 marks]

   (c) Implement a method `cd()` that will allow you to move to a different directory (changing the working directory). It should work as follows:

```
# if you try to move to a non existing directory or to a file,
# the method should complain:
>> fs.cd("casa")
The directory does not exist!
# But you can move to an existing directory in the working directory.
>> fs.cd("home")
# if we now do ls(), you should only see the content in home:
>> fs.ls()
home
```

```
thor
  hunde.jpg
  quatsch.txt
isaac
   gatos.jpg
```

Note that our filesystem is case sensitive, if the user searches for "Home", the method won't find the folder, because "home" is a different folder.

[10 marks]

(d) Implement methods to create files `create_file(name)` and directories `mkdir(name)` within the working directory. Both methods should make sure that the file or directory to be created doesn't already exist within the working directory. Directories must be empty when creating them with `mkdir(name)`. The method `mkdir` may allow you to indicate the owner when creating it, but files will share the owner of the working directory.

[5 marks]

(e) Modify the method `cd()` to allow us to go back to a parent directory. We will indicate the parent directory as "..". For example:

```
>> fs.cd("home")
>> fs.pwd()
'home'
>> fs.cd("..")
>> fs.pwd()
'root'
```

Note that applying `fs.cd("..")` in a root node with no parent directory will have no effect, but won't return an error.

[5 marks]

(f) Implement a method `rm(name)` that will allow you to remove a file from the current working directory. A directory can only be deleted if it is empty.

```
>> fs.rm("home")
Sorry, the directory is not empty
```

[5 marks]

(g) Implement a method `find(name)` which tries to find a file `name` in a file system and returns the path to the first occurrence of the file if it finds it but False otherwise. For example:

```
>> fs.find("gatos.jpg")
'root/home/isaac/gatos.jpg'
>> fs.find("thor")
'root/home/thor'
>> fs.find("unix")
False
```

Note that if you moved deeper in the directory tree using `cd()`, `find(name)` should only look from the current working directory.

[10 marks]

(h) The UNIX file system has many other operations that you could implement here. Here are some ideas, but feel free to implement any functionality you find useful. Discuss ideas with us, but please add enough comments to understand what you are aiming to achieve.

- chown -R: we have implemented `chown()` to change the owner of a single file or directory. Add an efficient way to apply this function recursively to all files and sub directories of a folder.
- Files and directories usually have permissions (read, write, and execution). Can you add that and functions to manipulate the permissions? (e.g. chmod).
- Improve the `ls()` to show the owner, permissions (similar to what `ls -l` would do in UNIX.
- In UNIX we can also move files from one directory to another using the command `mv`, indicating the destination PATH.

[20 marks]

You are free to use your favourite Python IDE to do this. Once you have completed your program, name your code as "ex04.py", and submit it on Moodle.

Important notes:

- To get full marks your program should work, be well documented, not be unnecessarily complicated and you should use inheritance to avoid code duplication.
- Use only the operations that have been introduced in the lectures.
- Optionally, if you finish and submit before the deadline, you can demo your code to one of the demonstrators in the lab and they will give you feedback.
- Doing or not doing the demo does not affect your mark.
- We will not give you the marks immediately during the demo.
- Make sure you note down the name of the demonstrator if you do the demo. *You cannot resubmit your solution after demoing it to us.*
- You may be (randomly) selected to demo your solution the week after the deadline. Lab demonstrators will contact you to ensure you attend the lab session. *Being unable to explain your solution may affect your mark.*
- **Submission deadline: 30th of November at 3pm**.