Practical 9 - Automation, using OS

Today we will explore Python as an automation tool to help us automate repetitive tasks with files and the **os** module.

Walkthrough Example - Renaming Files

Download the files from https://github.com/CP1404/Practicals/tree/master/prac_09

- Extract the **Lyrics.zip** file into the project directory so it's in a subdirectory called Lyrics.
- Open the directory Lyrics/Christmas so you can see the files listed in your file browser or PyCharm (but note that PyCharm doesn't always refresh as often as you might like).
- Open the file, **cleanup_files.py** and run it.

Notice:

- It imports the **os** and **shutil** modules for working with the operating system and files
- Comments explain what parts of it do. There are two commented-out options. Try each, one at a time:
    - **rename** files by replacing spaces with _ and ".TXT" with ".txt" (in same directory)
    - **move** files to a subdirectory with the new name

Note: renaming files changes their names (amazing!), so to re-run your code with the files in their original state, you can just re-copy them from the Lyrics.zip file provided.

The files we're working with today are from a **real-world example**. Lindsay uses words projection software at church that takes these files. The modifications are real needs, so this is another example of using programming to automate tasks in your everyday life! Nice :)

Modifications:

1. Notice that the existing files have been named inconsistently, e.g. some are PascalCase like "SilentNight.txt" and some have spaces like "Away In A Manger.txt" or are not in Title Case like "O little town of bethlehem.TXT"
   Write code to make them consistently use the format like "Away_In_A_Manger.txt", "Silent_Night.txt" and "O_Little_Town_Of_Bethlehem.txt" respectively:

Existing Filename (inconsistent format)	**Desired Filename (consistent)**
Away In A Manger.txt	Away_In_A_Manger.txt
SilentNight.txt	Silent_Night.txt
O little town of bethlehem.TXT	O_Little_Town_Of_Bethlehem.txt
ItIsWell (oh my soul).txt	It_Is_Well_(Oh_My_Soul).txt

Important:

Do NOT try and solve all of these cases at once. Rather, work up to them, building the **get_fixed_filename()** function that returns a fixed filename. Test just printing the names before renaming the files. When it works for one case, make it handle another one and so on… iterative development!

Hints:

- You will **not** find simple string methods like **replace()** that can solve all of this problem for you.
- A good approach would be to step through each character (and index) with a **for loop** and consider how it relates to the character before or after it, since the context is what matters here.
  E.g. if the current character **islower()** and the next character **isupper()** such as with the "tN" in "SilentNight"), then you know you need to put a '**_**' between them.
- You can start with an empty string and build it using string concatenation step-by-step as you determine what the next character should be. E.g. for the above case, you can add the 't', then the '_' to your new filename string, then move on to the next iteration in the for loop where you will add the 'N'.

When your renaming is working properly (congratulations!)…

2. Currently the program runs only in one directory.
   Make it work for all directories. You can do this a number of ways, including using **os.walk()**, which loops through all subdirectories, giving you (each time): directory name, subdirectory list, file list.
   Uncomment the os.walk() example at the bottom of the cleanup_files.py example file to see how it works.

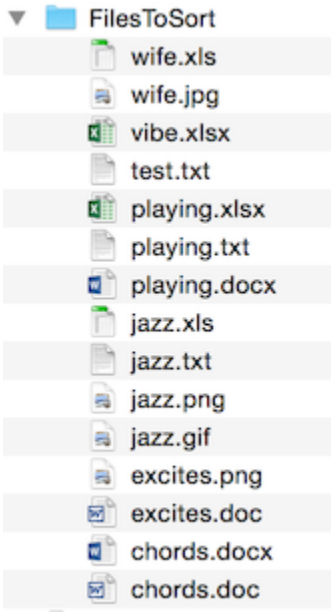
From Scratch

- Extract the **FilesToSort.zip** file which contains files with various names and extensions.
- Write code to sort these files into subdirectories for each extension.

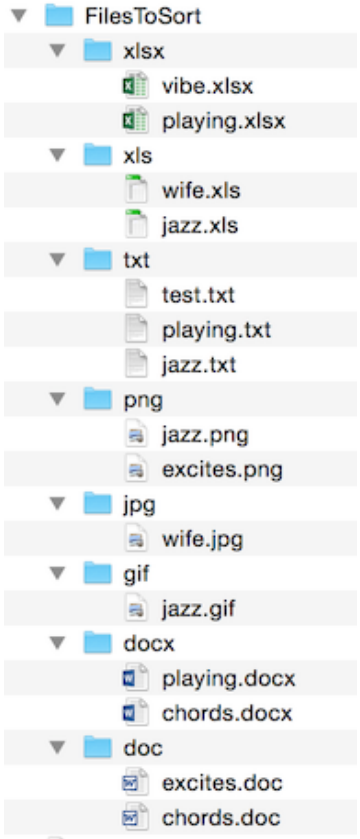
Version 1:

Use **os.mkdir()** to create a directory with for each new extension that your program finds and use **shutil.move()** to move files into these new directories.
E.g. move all files ending in ".txt" to a directory you create called "txt", and all ".doc" files to a "doc" directory.
Do not try and create directories you've already made.
Tip: You might like to add the extensions to a list or a **set** as you process the files.

Before:



After:



Version 2:

Let the user categorise different extensions as the program encounters these, then move them all into those subdirectories.
E.g.

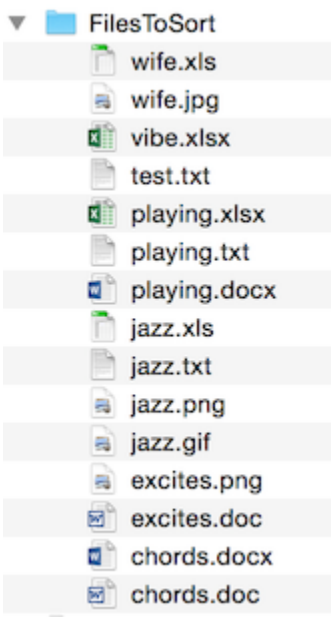
- one user might want a category “docs” containing all .doc, .docx, .rtf, .txt and “images” containing .jpg, .gif, .png.
- another user might want a category “office” containing .doc, .docx, .xls, but put the .txt files in a “text” category directory.

Tip: Add the extensions to a **dictionary** and make a list of the categories as you process the files.
Note: there are two parts to this - **categorising the extensions** and **moving the files**. You should approach them as separate steps.

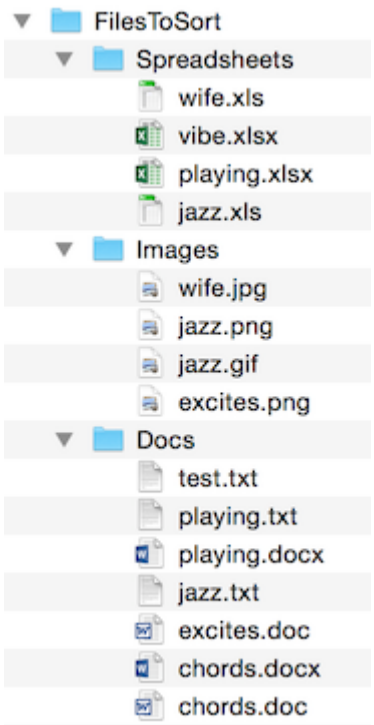
For one example run with these files (user input in **green**):

```
What category would you like to sort doc files into? Docs
What category would you like to sort docx files into? Docs
What category would you like to sort png files into? Images
What category would you like to sort gif files into? Images
What category would you like to sort txt files into? Docs
What category would you like to sort xls files into? Spreadsheets
What category would you like to sort xlsx files into? Spreadsheets
What category would you like to sort jpg files into? Images
```

Before:



After:



Extension & Practice Work

Check files for missing data:

The song lyric text files should all have copyright information in them on a line that starts with **.i** like:

```
.i © 2011 Thankyou Music (Admin. by Crossroad Distributors Pty. Ltd.)
```

Write a program that reports the names and locations of all of the files that are missing this line.

Version 2
Automatically look up the copyright information from the Internet based on the song title and author, then add the data to the file… Good luck with that ;)