**2/6 – Writing out the steps explicitly, for sanity’s sake**

- Press button, call “get\_top\_level():”

- IF all three entries in the “dir\_location” list are directories (directory 1, directory 2, destination) then define “top\_level” as:

- Split each dir\_location directory by the \ and grab the last one

- EX: [‘Test Folder 1’, ‘Test Folder 2’, ‘Dest Folder’]

- Call “merge()” using the three directories from “dir\_location”

- EX: [‘C:\Python\My Code\DB\File Merging\Testing\Test Folder 1',

‘C:\Python\My Code\DB\File Merging\Testing\Test Folder 2',

‘C:\Python\My Code\DB\File Merging\Testing\Dest Folder’]

- Define “current\_path” as:

- Split the first directory by the first entry of “top\_level” and grab the second piece

- This serves to capture everything after the top level directory in path form

- EX: [‘’] for the first time through; if “dir1” =

‘C:\Python\My Code\DB\File Merging\Testing\Test Folder 1\Level 1\Level 2’

then “current\_path” would equal ‘\Level 1\Level 2’

- Define “dest\_path” as:

- Path combination of the given destination path and our newly formed current\_path

- Loop through all three different categories of files in the current directory, paste them all into:

- Path combination of “dest\_path” and the filename

- Loop through all the folders our two directories share:

- Try to create a new folder with each shared folder’s name at:

- Path combination of “dest\_path” and the folder name

- Recursively call “merge()” using:

- Path combination of the given dir1, dir2, and dest with the shared folder’s name

**FULL BLOWN MANUAL EXAMPLE TIME:**

Using these starting paths:

dir1 = ‘C:\Python\My Code\DB\File Merging\Testing\Test Folder 1'

dir2 = ‘C:\Python\My Code\DB\File Merging\Testing\Test Folder 2'

dest = ‘C:\Python\My Code\DB\File Merging\Testing\Dest Folder’

get\_top\_level():

- Are all of these valid paths? YES, define “top\_level” as:

top\_level = [‘Test Folder 1’, ‘Test Folder 2’, ‘Dest Folder’]

merge(dir1, dir2, dest): **[ ROUND 1 ]**

current\_path = ‘’

dest\_path = 'C:\\Python\\My Code\\DB\\File Merging\\Testing\\Dest Folder\\'

!!NOTE!! Adding that empty string to “dest” added those two \\ on the end!

Copy path for test.jpg:

'C:\\Python\\My Code\\DB\\File Merging\\Testing\\Dest Folder\\test.jpg'

Create path for Level 1 folder:

'C:\\Python\\My Code\\DB\\File Merging\\Testing\\Dest Folder\\Level 1’

RECURSING: merge inputs:

dir1 = ‘C:\Python\My Code\DB\File Merging\Testing\Test Folder 1\Level 1’

dir2 = ‘C:\Python\My Code\DB\File Merging\Testing\Test Folder 2\Level 1’

dest = ‘C:\Python\My Code\DB\File Merging\Testing\Dest Folder\Level 1’

merge(dir1, dir2, dest): **[ ROUND 2 ]**

current\_path = ‘Level 1’

dest\_path = ‘C:\Python\My Code\DB\File Merging\Testing\Dest Folder\Level 1\Level 1’

!!ERROR!! I think we’re adding the new folder to the end of our dest twice… let’s take away the dest\_path definition and just use dest.

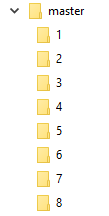
2/14 – Zooming out – I feel like I may have lost the plot at some point…

Program input: One folder – “master” – that contains the root folders of all directories to be merged.

Program output: The first folder in “master” has become the merged folder; all else remains the same.

Details:

- Inside of “master”, say there are eight different folders. These root folders’ names will be used to identify their unique files later, so let’s name them simply:



We can call our renaming function with the root folder’s name as the suffix input – so different versions of the same file “butts.txt” would be renamed “butts\_1.txt”, “butts\_4.txt”, etc.

Every time we call our recursive “merge” function, we want to give it TWO inputs: dir1 and dir2

The first, dir1, will be our new master folder that gets compared with all the others in turn. The second will change with each step – it will start out as folder 2, then folder 3, etc.

The very first step in our script will be identifying all of the top level folders and saving their names. We will use the length of the list of their names as the very top level loop. Let’s put it in a revamped “get\_top\_level” function. Call merge seven times in this example – once for each folder minus one.

Inside of merge, everything stays the same EXCEPT:

- When we find unique files in dir1, do nothing – it’s our master! So maybe we shouldn’t even look? Same idea for folders.

- When we find unique files in dir2, copy them over to dir1 instead of a different destination folder. Same idea for folders.

- When we find diff\_files (same name, different contents), first things first: rename the one in dir2 according to its directory number (butts\_2.txt).

- After doing this for all diff\_files, we need to detect ALL files that have been renamed (using our knowledge of the directory names, we can find any files that could have been renamed!) and see if we can find any duplicates that have different names.

- Example: Inside the current working directory in dir1, we see (among other things):

- butts\_3.txt, butts\_4.txt, butts.txt

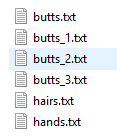
- Currently we are in directory number 6 and we see yet another butts.txt that differs from the one in dir1. First things first: rename it butts\_6.txt and move it over to dir1. We see:

- butts\_3.txt, butts\_4.txt, butts\_6.txt, butts.txt

- Now suppose that butts\_3.txt and butts\_6.txt are actually share the same contents! How do we figure this out and only keep one of them?

- What we really need here is a way to compare file contents while ignoring file names. How is this possible?

Trying a real example…

 butts.txt: butts butts butts butts!

butts\_1.txt: buuuuuuuuuuuuuuutts

butts\_2.txt: butts butts butts butts!

butts\_3.txt: butts butts butts butts???

hairs.txt and hands.txt are blank.

In this example, we want the program to be able to tell that butts.txt and butts\_2.txt are really the same file, even though they have different names.

Method: Using the combinations method from itertools, we can generate a list of all unique pairs of files in our folder (ex: for [1,2,3,4] we return (1,2) but not (2,1)). Then we can loop over that list of pairs and file.cmp them. It worked! Function idea is inside of ignoring\_names.

- Now we just need to create a list of all the files that look like they were renamed, determine if any of them are duplicates, and then just delete one if they are.

get\_top\_level(master):

0. Input: “master” folder

1. Get list of root directory names to be compared (os.listdir)

a. Store this list of names for renaming files later (root\_names)

2. For each root folder after the first:

a. Call merge on the 0th and nth (starting at 1) entries in root\_names

(At the end of the day, the 0th folder will be our newly merged master)

3. After all the recursive merging is done, output the elapsed time and print a “done” message

merge(root\_names[0], root\_names[n]):

0. Compare two directories with filecmp.dircmp (compared)

1. For all items only found in second directory (compared.right\_only):

a. Copy all unique files to first directory

b. Copy all unique folders (and all contents) to first directory

2. For all files that appear in both directories (compared.common\_files):

a. If identical copies are found in both directories, do nothing.

b. If different copies are found in both directories:

I. Rename file in second directory based on second directory name in root\_names

II. Copy it over to first directory

3. For all files in first directory that appear to have been renamed:

a. Generate list of tuples of all possible combinations

b. For each tuple:

I. Determine if they are the same file (ignoring names)

0. If so, have we deleted either of these files already?

a. If yes, do nothing.

b. If no, delete the first one and add it to the list of deleted files.

1. If not, do nothing.

4. For all folders that appear in both directories (compared.common\_dirs):

a. Dive into each one and start over: merge(dir1 + folder, dir2 + folder)