**Example time:** we are merging dir4 with our destination directory, dir1. We find these files at our current level in dir4:

butts.txt

garfield.xlsx

watermelon.docx

mypasswords.png

And we find these files in dir1:

butts.txt

butts\_2.txt

butts\_3.txt

garfield.xlsx

watermelon.docx

watermelon\_3.docx

mypasswords.png

mypasswords\_2.png

Using our omicompetence, we know from the start that the following files have exactly the same contents:

dir4: butts.txt dir1: butts\_3.txt

dir4: garfield.xlsx dir1: garfield.xlsx

dir4: mypasswords.png dir1: mypasswords\_2.png

Based on this knowledge, the only file from this level of dir4 that we want to have in dir1 at the end of the day is dir4’s version of watermelon.docx, which will be renamed as watermelon\_4.docx. How do we accomplish this?

**Step 1:** for the first step, we look at all files unique to dir4 by name. In this example, there are no unique files by name – it appears as though we already have everything found in dir4 in our dir1.

**Step 2:** we look at all files with the same name but different contents. The following files from dir4 are flagged and put into compared.diff\_files:

butts.txt watermelon.docx mypasswords.png

The file garfield.xlsx does NOT get flagged because despite having the same name, it turns out it also has exactly the same contents in each directory. The files that were flagged are renamed with the “\_4” suffix and put into dir1. However, we know that both butts.txt and mypasswords.png are hidden duplicates – they managed to skirt the duplicates test that garfield.xlsx failed, but they have true duplicates by a different name already in dir1. We need to catch them and delete them in our next step.

**Step 3:** catching the imposters! Right now, dir1 looks like this – we’re done looking at dir4, all the action is happening here from now on:

butts.txt

butts\_2.txt

butts\_3.txt

butts\_4.txt

garfield.xlsx

watermelon.docx

watermelon\_3.docx

watermelon\_4.docx

mypasswords.png

mypasswords\_2.png

mypasswords\_4.png

**Step 3, part one:** Identify suspects. Our hidden imposters have been highlighted. butts\_4.txt and mypasswords\_4.png have got to go. One brute force method to kicking them out would be to generate tuples for each unique pair of files in dir1 and compare them against each other. Both the butts\_3.txt and butts\_4.txt and the mypasswords\_2.png and mypasswords\_4.png pairs would fail this test, and we’d delete one of each pair. However, we’d end up comparing a LOT of other pairs that are perfectly fine, and waste a lot of time.

What we really want to do is only compare the files that could potentially have this problem. In this example, we can rule out garfield.xlsx as a suspect right away – he could not have a hidden duplicate anywhere in this level of dir1 because we didn’t rename any garfield.xlsx files in step 2! On the other hand, the three files we DID rename are all suspects – even though we know our watermelon.xlsx family is innocent, we need to check. Without our omnicompetence we’d have no way of knowing if our famliy of three was unique.

Our list of suspects must then include all files that ended up in compared.diff\_files back in step 2.

**Step 3, part two:** Interrogate suspects.

Using the compared.diff\_files list, we can identify which particular files in dir1 could be imposters. To compare this list of suspect families and our list of potential suspect individuals, we’ll need to strip off the number suffixes from any renamed files in dir1 and pull out their “last names,” so to speak. Then, for each suspected family in compared.diff\_files, we search our list of suspected individuals for a match. Once we’ve assembled a list of matches, we compare the matches against each other and delete any imposters we find.

To make this easier, let’s set up four lists:

suspects = compared.diff\_files = [‘butts.txt’, ‘watermelon.docx’, ‘mypasswords.png’]

dir1\_files = [‘butts.txt’, ‘butts\_2.txt’, ‘butts\_3.txt’, ‘butts\_4.txt’, ‘garfield.xlsx’, ‘watermelon.docx’, ‘watermelon\_3.docx’, ‘watermelon\_4.docx’, ‘mypasswords.png’, ‘mypasswords\_2.png’, ‘mypasswords\_4.png’]

dir1\_renamed = [‘butts\_2.txt’, ‘butts\_3.txt’, ‘butts\_4.txt’, ‘watermelon\_3.docx’, ‘watermelon\_4.docx’, ‘mypasswords\_2.png’, ‘mypasswords\_4.png’]

dir1\_renamed\_stripped = [‘butts.txt’, ‘butts.txt’, ‘butts.txt’, ‘watermelon.docx’, ‘watermelon.docx’, ‘mypasswords.png’, ‘mypasswords.png’]

For each suspect, we now have to do the following:

- Look for any matches in our dir1\_renamed\_stripped list; if any are found, add them to a temporary list called “stripped\_suspect\_matches”

- Find the actual files in dir1 that correspond to each of our suspect\_matches; add them to a list called “suspect\_matches”

- Create another list of tuples called “suspect\_pairs” that includes all the unique pairings of our “suspect\_matches” list; loop through this list and compare each one using filecmp.cmp

- For all duplicates found, try to delete them (we may have already in a previous loop).

Here’s how this shakes out for our first suspect, butts.txt:

- Matches are found in the dir1\_renamed\_stripped list at indexes 0, 1, and 2 and are added to the list. Plugging these indexes into dir1\_renamed gives us our second list.

stripped\_suspect\_matches = [‘butts.txt’, ‘butts.txt’, ‘butts.txt’]

suspect\_matches = [‘butts\_2.txt’, ‘butts\_3.txt’, ‘butts\_4.txt’]

- Our suspect\_pairs list looks like this (a list of tuples):

suspect\_pairs = [(‘butts\_2.txt’, ‘butts\_3.txt’),

(‘butts\_2.txt’, ‘butts\_4.txt’),

(‘butts\_3.txt’, ‘butts\_4.txt’)]

- After looping through this list and comparing each pair, pair #0 gives us FALSE; no match. Same for pair #1: no match. But pair #3 matches! Arbitrarily, we decide to delete the first entry of that list, butts\_3.txt. Now we repeat this process for each suspect.