Security HW3 Honeywords generator

Bo Gao, bg447
Teammates: John-charles Labarge (jfl95), Ruirui Wang (rw554), Zexi Liu (zl558)

Our code submission includes 3 files, honeyword1.py, honeyword2.py and honeyword3.py, representing the answers to the 3 problem.
To run each script, place them in the same file with rockyou-withcount.txt and the input_file, and use:

```
python your_program.py n input_filename output_filename
```

Algorithm 1 (empty training set):
Given no training set, the basic operation is to permute and shift the passwords. But we also need the honeywords to follow the same pattern as the passwords. so if the passwords are organized in blocks of similar characters, we should keep this block feature. For example, given a password “fjkslh38493”, if we just randomly permute the password, we may break the blocks and generate “fj9ks84lh33”, and the real password will clearly stand out among these honeywords. So the first step is to detect character blocks within the password.
Then we change the English alphabet blocks, and the digits blocks, and the special character blocks. Then we do a permutation between these blocks, for the purpose of confusing the hacker.
And we think it would be good to cancel the length and character set of the password, so when we change the alphabet blocks, in addition to permutation, we may also add / delete / change some character in the password, based on possibility. (In fact we think it would be good to detect whether the password contains a dictionary word, and if it does, we should replace it with dictionary word, because among a bunch of randomly generated character sequences, a dictionary word can easily stand out. But the instructor said no other resources should be used)
And we took care to not reveal any information of the length of the real password when we changed the password length. For example, if the real password length is len, and the generated honeywords lengths are among range [len - rand_num, len + rand_num], then the hacker can guess the length of password by calculating the average or mean of all the lengths. To prevent that, we generated honeywords in range [len - rand_num1, len + rand_num2], so the real password can appear at any length in the range with equal possibility.
Finally, we removed duplicate or reversed honeywords because they are obvious indicator of the true password.

Algorithm 2 (training set is the top 100 rockyou passwords set (rockyou100)):
Now we have the rockyou100, and suppose the hacker have this information too, and the hacker knows that they are high-frequency real passwords. So if only one of our honeyword is in rockyou100, it is very likely to be the real password. Therefore, we need to conceal the fact of whether the password is in this set. So we will select half of our honeywords from the

rockyou100 (if it is enough), and half of the keywords generated from Algorithm 1. So that regardless of whether the password is in the rockyou100 or not, the hacker will always be faced with half rockyou100 and half randomly generated words.
And in terms of selecting honeywords from rockyou100, we preferably choose words that share the common pattern with our password (common pattern means being composed of the same types of characters). And should we run out words that share the common pattern, then we will pick from the rest of rockyou100. And if we have used rockyou100 up, then we will fill the rest of the honeyword set with words generated by algorithm 1.

Algorithm 3 (training set is the whole rockyou passwords set):
Our solution to this problem is quite similar to the previous one, we still pick half of the honeywords from rockyou and pick the other half by algorithm 1. But now that we and the hacker have the whole set of rockyou, if the honeywords we pick are concentrated in a certain frequency range, it will tell the hacker that the real password is in that range too. So this time we pick honeywords with the same pattern as the real password randomly from the whole rockyou set.