Module Fourteen:
Dictionary Password Cracker

Grading Criteria

Total Points: 20 points

In this lab, you will be writing a simple dictionary-based password cracker. Hopefully, this helps illustrate how scary easy it can be to use a dictionary to crack simple passwords.

☐ **[2.5 points]** Your submission is labeled as "cyb404_module14_lab00_[nau_id]_[lastname]_[firstname].zip." For example, if I were to submit a file, it would be labeled as cs460_module14_lab00_mv668_vigil-hayes_morgan.zip. **FAILURE TO COMPLY WITH THIS STEP CAN LEAD TO A ZERO GRADE.**

☐ **[2.5 points]** Your submission files are in the correct formats as specified.

- ☐ [1 points] Password cracker code is all contained in a file called password_cracker.py
- ☐ [1.5 points] There is a comment for every functional unit of code that details the purpose of the code.

☐ **[15 points]** Code correctly takes in file of encrypted passwords and attempts to crack them against the dictionary file provided (dictionary.txt). I have provided an encrypted password file for you to test your code against.

- ☐ [2.5 points] File runs using command: python3 password_cracker.py [password_file] [dictionary_file]
- ☐ [2.5 points] There are no runtime errors
- ☐ [10 points] Your code correctly cracks the password for the username provided.

Part I: Building a dictionary-based password cracker

Read the attached PDF "cyb404_module14_assignment_reading.pdf for background.

The real strength of the Python programming language lies in the wide array of standard and third-party libraries. To write our UNIX password cracker, we will need to use the crypt() algorithm that hashes UNIX passwords. Firing up the Python interpreter, we see that the crypt library already exists in the Python standard library. To calculate an encrypted UNIX password hash, we simply call the function crypt.crypt() and pass it the password and salt as parameters. This function returns the hashed password as a string.

Read through the details on the Python wrapper for the Unix crypt() algorithm.

```
Programmer$ python >>> help('crypt')
```

Let's quickly try hashing a password using the crypt() function. After importing the library, we pass the password "egg" and the salt "HX" to the function.

```
programmer$ python
>>> import crypt
>>> crypt.crypt("egg","HX") 'HX9LLTdc/jiDE'
```

The function returns the hashed password value "HX9LLTdc/jiDE" as a string. Success! Note that the salt appears as the first two characters of the encrypted string. Now we can write a program to iterate through an entire dictionary, trying each word with the custom salt for the hashed password.

To write your program, you will create two functions: main() and testpass(). It proves a good programming practice to separate your program into separate functions, each with a specific purpose. In the end, this allows you to reuse code and makes the program easier to read. Your main function opens the encrypted password file "passwords.txt" and reads the contents of each line in the password file. For each line, it splits out the username and the hashed password. For each individual hashed password, the main() function should call the testPass() function that tests passwords against a dictionary file. This function, testPass(), should take the encrypted password as a parameter and returns either after finding the password or exhausting the words in the dictionary.

Hint: testPass() will need to account for the salt that represents the first two characters of the encrypted password hash.

Q00. Prove the correctness of your solution by providing the plaintext password for the username 'morty' in the password file.

Jessica