EN.540.635 "Software Carpentry"

Weekly Challenge 6 - Passwords and Encrypted Messaging

For this assignment, we will work through building up your fundamentals. So far you have been introduced to quite a lot. Here, we will try to incorporate a little bit of everything we've covered so far.

Part 1: The Password Function

Similar to how we wrote *encrypt()* and *decrypt()* functions for Weekly Challenge 4, we will now be expanding on these functions to write a *password()* function that wraps the *decrypt()* function and only allows us to use it if we specify a proper username and password (i.e. our username and password act as an additional layer of security to decrypt some text that is encrypted). For the purpose of this assignment, you should have a minimum of two username/password combinations that work. The first should be the username "hherbol1" and corresponding password "Fifa" (in reference to the original instructor of this course), but the others are all up to you. Since you want to be able to check against a number of stored username/password combinations, you should probably think about which data structure might be most suitable to use for this assignment (An actual username/password database will have thousands of stored entries, and we would want to use a data structure that allows us to easily access any entry).

Here, we go through a bit of background information in regards to how passwords are typically stored in an actual password database. Initially, you might decide to store the passwords as strings within your code. However, storing the passwords in plain text like this is not safe practice! If someone else gets a hold of your code, they could see everyone's passwords! To circumvent this issue, we never store the actual passwords, but instead store something called a hash. A hash function is a way of mapping data of arbitrary size to a fixed size, and this method is one-way - one cannot recreate your password from your hash. If you have someone's hashed password, you do NOT have their actual password!

Python has a built-in library for with common hashing functions called *hashlib* - the documentation for this library can be found here. There are many different secure hash algorithms (SHA) that exist in modern cryptography. In this assignment, we will focus on using the SHA-512 hash. For example, instead of storing the password "Fifa" in our code, we can use store its SHA-512 hash instead. Then, when the user enters their password, your function would hash the input password and compare it against the stored hash. However, according to industry standards, just hashing the password is not enough! If a malicious entity gets hold of a certain person's hashed password, they then can hack into multiple people's accounts if they have the same password (and thus the same hash). Therefore, to be more secure the passwords are 'salted' with randomized data, and then hashed. The salted hash is then stored with along with the salt. More information can be found at this website and this YouTube video.

To summarize, your password function should adhere to the following requirements:

- The function should accept three input parameters - the encrypted message to be decrypted, a username string, and a password string.
- A decrypt function should be nested inside your password function.
- The password function should contain the database of usernames and corresponding passwords, where all the passwords have been appropriately salted and hashed (we should not be able to see the original passwords written in the code).
- There should be conditional statements in place to check if a username/password combination is correct.
- The password function should return the decrypted message only if the username/password combination is correct.

Essentially, the following code should work:

```
def encrypt(msg, N, E):
    return [(ord(s) ** E) % N for s in message]

def password(enc_msg, usr, passwd, N, D):

    def decrypt(enc_msg, N, D):
        return ''.join([chr((s ** D) % N) for s in enc_msg])

    '''
    Code to check username/password combinations goes here.
    Store your usernames, salted password hashes, and corresponding salts appropriately.
    Return the decrypted message if the username and password are correct.
    '''
    pass

if __name__ == "__main__":
    message = "This is a secret message"
    encrypted_message = encrypt(message, N=17947, E=7)
    decrypted_message = password(
        encrypted_message, "hherbol1", "Fifa", N=17947, D=10103)
    assert message == decrypted_message, "Error — Decryption failed!"
```

Part 2: Fake Instant Messaging

Now, with the above code we've written, we should be able to decrypt messages given a correct username/password combination. Let's test out a fake instant messaging situation, in which we use a while loop to continually accept user input messages. So far we have gone over functions, conditionals, and loops. Now, let's go over file input and output! We can encrypt all those messages and write them to a text file. Thus, in this problem you will need to open a new file, save the contents of the encrypted messages into the file, and then close the file. All of this can be written into a single *start_messenger()* function. The code you write for this messenger should do the following:

1. Have an infinite while loop.
2. In the loop, keep accepting messages from the user until they type "STOP", in which case the loop ends.
3. Whenever the user hits the enter key, take the text they input, encrypt it, and write it to an external text file.

We can have another function for re-opening that file and read in the encrypted messages. Further, the function should ask the user for their username and password, and then decrypt the messages if the user is allowed. Note, when we log in to any Linux systems (say MARCC over ssh, or simply typing in a password after calling *sudo*), you normally see that the password itself does not show up when you type it (the technical term is that the password is not echoed on input). Websites normally have a similar functionality where they do not echo your password when you type it in. Use Google to find out how to do the same in Python!

Finally, part of this assignment will involve testing how well you can think ahead. This assignment in essence mimics password security. As such, try your best to consider any kind of user input problems that can arise and handle them accordingly. For example, if the user enters an invalid username/password, what kind of error message do you tell the user? How about if they enter nothing? Think about all of these edge cases as best you can!

Note, although this assignment has 2 parts, they build off one-another. As such, do not turn in 2 separate parts, but instead a final code that has everything incorporated in it. Your final code submission to Canvas should have a structure similar to the one below. You can write any number of helper functions in addition to the ones listed below to complete the assignment.

```python
def encrypt(message, N, E):
    return [(ord(s) ** E) % N for s in message]

def password(enc_msg, usr, passwd, N, D):

    def decrypt(enc_msg, N, D):
        return ''.join([chr((s ** D) % N) for s in enc_msg])

    '''
    Code to check username/password combinations goes here.
    Store your usernames, salted password hashes, and corresponding salts appropriately.
    Return the decrypted message if the username and password are correct
    '''
    pass

def start_messenger(msg_fptr, N=17947, E=7):
    '''
    This code starts the messenger. It reads in each line of input from the
    user, encrypts each line, and stores them in a list. Each line of
    encrypted text is written to a text file, the name of which is specified by
    "msg_ftpr".
    '''
    pass

def read_messages(msg_fptr, N=17947, D=10103):
    '''
    Asks the user for their username and password. Reads the encrypted text
    from "msg_fptr" and decrypts it if the username/password combination is
    correct.
    '''
    pass

if __name__ == "__main__":

    # Part 1
    # Input message and test the decryption. If the username-password
    # combination is not correct, an error will be returned.
    message = "This is a secret message"
    encrypted_message = encrypt(message, N=17947, E=7)
    decrypted_message = password(
        encrypted_message, "hherbol1", "Fifa", N=17947, D=10103)
    assert message == decrypted_message, "Error - Decryption failed!"

    # Part 2
    # Start messaging app that allows the user to add in as many messages as
    # desired. It will encrypt using the messages and save them to the
    # specified file.
    start_messenger("messages.txt")
    # Ask the user for a valid username-password combination, and decrypt the
    # file and print the results.
    messages = read_messages("messages.txt")
    print(messages)
```