Lab 1 Report - Recursion

The problem at hand is brute force password hacking, we have a list of users with resources to test and figure out the passwords to each account, our job is to do exactly that, generating passwords recursively and checking each possibility with salts to see if two hashes match, then know that the password is correct for that specific user.

My plan for the solution of this problem came in many forms, mainly speaking towards the recursive string password generator, but from the start, I knew a couple things for sure. I wanted to split my code into a couple primary and basic methods: a read file method; that would read the password file given by the assignment and place them into lists of lists, first a list of users, and then a list for each user holding the necessary variables for later use of check during brute force hack. Returning the finished and filled list, ready to be used for the next explained steps.

After that method, a generator method to generate under the constraints given all possible passwords from 000 - 9999999, what I intended to do was place each possibility into some list, and then once a finished list was filled with all the possibilities run it through a check method to see if any matched! Now as you can tell by the lack of detail, after lots of tests and different iterations of the method I could not get a single one to work the way I intended or needed. Very sad, but I will work on my skills more and maybe eventually this is something that will seem simple later on. Anyways, in that failed method, what I had tried to do was pass an empty string, and (in this iteration) a length to max the created length for a password. I tried using 0-9 range for loop to fill and replace numbers in a string, appending string values and removing

accordingly, but either I was on the wrong track or I could not figure out the right combination. Either or, the generator did not work sadly.

Lastly, there is the password check, passes the (passwords) list that was intended to be generated, and the list of list inventory I created from the file given for ease of use (records). Now since at this point it was apparent I had to shift focus off of that method, I created a sample list to pass instead and test. The method that checks the password is very simple and one of the easiest of the methods, to start I created a for loop that could cycle through the elements of the passwords list, and for each, inside the loop, a loop within resides that goes from 0 to the length of records. Having this allows us to append the salt value to each password and check the hash, all within the same method :). I create a new variable test_pass to be the salted password ready to be hashed in the next line and then compared. If it is matched the console shows that it is found, as well the for who and the password for that specific user.

All in all, the best way to describe the code itself is to show and tell the experiments I underwent throughout the process. Other than the recursive generator everything else pretty much works as I intended. Using the sample list I created for my lab it was able to find and display the password as well as to who it belongs to. (pictured above) What is in the passwords list itself, does not matter, if the password is inside the method will pick it up and find it, if it does not then it will not cause an error, or inform the user, which putting as much time into the generator as I could, the trialification of the code was not put to the highest of tests. ((below) is the same code with added wrong passwords into the passwords list and working the same.) Which I regret, I would have

```
In [2]: runfile('/Users/brianperez/
Downloads/Lab 1 - Option B/Lab1 (Brian
Perez).py', wdir='/Users/brianperez/
Downloads/Lab 1 - Option B')
Password found:  6140 for User3
Password found:  942 for User4
Password found:  6682 for User1
Password found:  2419445 for User0
```

```
In [23]: runfile('/Users/brianperez/
Downloads/Lab 1 - Option B/Lab1 (Brian
Perez).py', wdir='/Users/brianperez/
Downloads/Lab 1 - Option B')
Password found:  6140 for User3
Password found:  942 for User4
Password found:  6682 for User1
Password found:  2419445 for User0
```

liked to inform for misuse of the constraints with anything other than 3-7 length passwords, or none found type of messages to be known to the user. When I didn't have all the correct passwords, or did not have any passwords at all the print would just not happen as exemplified by:

```
In [24]: runfile('/Users/brianperez/
Downloads/Lab 1 - Option B/Lab1 (Brian
Perez).py', wdir='/Users/brianperez/
Downloads/Lab 1 - Option B')
Password found:  6140 for User3
```

In conclusion, I jumped into Python very much with cold feet and I need a lot more practice to even start to get a grasp on these labs, as well as I need to put more time into these labs itself outside of python practice, so I can have the appropriate questions during TA sessions and elsewhere. I went to a TA for help, but I was so lost that I could not even ask the right questions to get me on the right path, which is completely my own fault.

Appendix -

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 15:40:04 2019

Course: CS 2302 - Data Structures
Author: Brian Perez
Assignment: Lab 1
Instructor: Diego Aguirre
D.O.L.M.: 9/16/19 (to write this)


"""
import hashlib

def hash_with_sha256(str):
    hash_object = hashlib.sha256(str.encode('utf-8'))
    hex_dig = hash_object.hexdigest()
    return hex_dig

def read_file():
    file = open('password_file.txt', 'r')       # opens password text file
    users = []
    for element in file.read().split('\n'):     # for loop that for each element in the file split by new line...
        users.append(element.split(','))        # appends it to the users list to create a list of lists, organized by the ','

    return users                            #returns the users list for later password_check function


def generator (string_list, length):    # **** Unused iteration of recursive password generation *****
```

```
    if length == -1:                # The idea was to pass an empty list and through each generation of password digit, decrease length till finish
        print("hi")                 #making it each combination of numbers one digit at a time till desired length
        return string_list          # Base case, length used up and triggers end of recursive calls
    for i in range(length):         # length times ...
        print("Length", length)
        for x in range (10):        # 0-9 would be used to place into digits
            string_list[i]+=(str(x))    # appending number to string list
            print ("List", string_list) # trace checks
            string_list[i]= string_list[i][:-1] # removing the last digit to be replaced later
    return generator(string_list, length-1)     #returns method, passing list, and length minus 1

def password_check (passwords, records):                # passes 2 paramters, passwords list and users list (now records)
    for x in passwords :                            # each password
        for y in range(len(records)):                   # and the length of the records so as to know how many users to check through
            test_pass = x + (records[y][1])                 # makes z a version of the password with the salt value attached
            if (hash_with_sha256(test_pass)) == (records[y][2]):    # hashes salted value and checks password correctness
                print("Password found: ", x, "for", records [y][0])             # done

# ================================================================================
# def gen(string_list, length):         ## Unused iteration of recursive password generation
#     if length < 0:
#         return
#     for i in range(10):
#         gen(string_list[ln]+=str(i)), length-1)
# ================================================================================




def main():
    records = read_file() #reads file, and organizes into list of lists
#    print(records[0][1])
    #print(generator([""], 3))

    #Sample passwords that would have been generated
    #*"0000" & "1000" to show not all entries are displayed, only true passwords after check
    passwords = ["1000", "6140", "00000", "942", "6682", "2419445", "0000"]
    password_check(passwords, records) #Checks and prints passwords


main()
```

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

-Brian Perez