Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: September 10, 2019

Instructor: Diego Aguirre

Assignment: Lab 1 Recursion

TA: Gerardo Barraza

**Introduction:**

In python, recursion can be used when trying to deal with problems that could become tedious later on in your program, such as adding constantly by any type of number until you reach a goal. The problem in this lab is to read a text file called “password\_file.txt” to generate a new password from length 3 to 7, and of course, by using recursion. In order to achieve this, the program must first read the user account, then concatenate the salt value, and apply a method called *“hashlib.sha256”* to check if your new string matches with that with the hashed password, if it does, it will return that specific password from that specific user.

**Proposed solution design and implementation:**

Before I begin analyzing how to solve this program, I must find a way to read the textfile line by line until I have parsed throughout the whole document without knowing its 100 lines, I suggest using recursion as that is the purpose of the lab anyways. I must also find a way to generate the length of the passwords to not only be between 3 and 7, but to also not hard code these values as well. I must also not use more than one nested loop.

Frankly, I should also include a method capable of generating a string so that I can compare it later to the final password needed for the lab.

**Experimental results**:

For my main method, I read the specific file given to me for the lab “password\_file.txt. I could have used a user input to ask for what document it wanted to read for the program, but I did not think it was necessary for this particular lab as the textfile is going to be the same as everyone else’s anyways.

I created a variable “min” with a value of three so that it can create a password of length 3 later on. I figured since this integer is not hardcoded into the actual method, this should not count for deduction when evaluating the lab.

I was able to read the textfile line by line, and was also able to send the pieces of these lines to my method “new\_password” in which, you guessed it, will create a new password to be compared to and returned if applicable. I was also able to use one of the two for loops here so it can read the entire textfile instead of just a particular amount of lines. I set a new variable called “hashed\_string” that takes the hashed password from that particular line, as well as another variable called “new\_pass”, that will return the password associated with the user.

The “new\_password” method begins by extracting the salt value, the variable “min”, and then the variable “hashed\_string” sent to us by the main method. It then reads the first base case that says that is the min is greater than or equal to 8, then an error message will read that the password is not applicable due to our requirements. If that is not the case, our second and last for loop will initiate that will go from 0 to a new integer created by sending min to the method “int\_converter”, which is used to create an integer of a certain maximum depending on the minimum. Inside the for loop, It would have created an index to create a certain size of a new string called “amount\_string”, and afterwards will create a new index using this new variable as well as the previous index and the min to generate a zero integer of the index to begin generating the password. Up till now, the process has only been to create an integer big enough to replace it with the actual password, you could think of it as a form of vessel. After creating this ‘vessel’, we then add it with hashed password and sent it to the given method “hash\_with\_sha256”, which will create our new digit to be finally compared with the password given to us in the textfile. If all went well, the program will return the new integer, and work will commence again with the next user, if it fails however, the method will start over while adding one more number to the minimum.

Once all of the work is completed, meaning it went through all of the users and found all of the passwords, it will then calculate the running time taken from the beginning of the program to see how long it took to generate all of the passwords. This is also needed for the lab, so Imported the time library from python to take this extra step.

My results from a single run as well as the running times are displayed below:

**A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated**

**Run times:**

|  |  |
| --- | --- |
| **Attempts** | **Time in seconds** |
| **1** | **230.63 seconds** |
| **2** | **232.34seconds** |
| **3** | **230.63 seconds** |

**Conclusions**:

With this lab, I was able to learn to code better using the Python language, including hacking different strings to create new passwords using the hash library available for Python. I was able to learn to solve different problems by using recursion throughout my lab.

**Appendix :**

**Main.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: September 9, 2019

Instructor: Diego Aguirre

Assingment: Lab 1 Recursion

TA: Gerardo Barraza

Purpose: To practice using recursion and to practice using this technique

to find new passwords using strings

'''

#Imports the hash library needed to be used in this program

import hashlib

#Imports time to be used in calculating the big o notation

import time

#Creates the new password related to its respective user.

def new\_password(salt\_name, len\_string, hashed\_string):

#Sent to method int\_converter, which converts the minimum integer into a the max it can be.

new\_int = int(int\_converter(len\_string))

if (len\_string >= 8):

print("ERROR: Password not possible! Not within 3 to 7 characters!")

return "Password does not exist!"

#This loop is used to create a new string to be compared to our hashed password.

for i in range (0, new\_int): # For i in range of the new integer

index = str(i)

amount\_string = len(index)

#Creates a new index and digit to construct a hashed password

new\_index = zero\_generator(len\_string, index, amount\_string)

new\_digit = hash\_with\_sha256(new\_index + salt\_name)

#If the new digit equals the hashed string, then it will return the new index

if (new\_digit == hashed\_string):

return new\_index

#Calls itself again until it returns the new index

return new\_password(salt\_name, len\_string + 1, hashed\_string)

#Method that adds the zeros at beginning of a certain string

def zero\_generator(int, index, amount\_string):

#If the integer quals the amount\_string, then it returns the number struing

if (int == amount\_string):

return index

return '0' + zero\_generator(int, index, amount\_string +1)

#This method is used to get a certain maximum value of an integer by using just its min

def int\_converter(int):

if (int == 0):

return ''

return '9' + int\_converter(int - 1)

#Method given with the program,

def hash\_with\_sha256(str):

hash\_object = hashlib.sha256(str.encode('utf-8'))

hex\_dig = hash\_object.hexdigest()

return hex\_dig

#Main method, will initiate the calls needed to execute the program; the brains of the program.

def main():

#Starts a timer that will calculate the total running time for the program

start = time.time()

#Message that welcom

print("Welcome. Password hack beginning.")

txt = 'password\_file.txt'

print("Reading....")

#Sets the lowest length of the password to be 3 as instructed in the lab

min = 3

#Reads the txt file as a textfile line by line

with open(txt) as textFile:

textFile\_lines = [line.split(',') for line in textFile]

#This loop is used to generate the new password using the new\_password method using the information from the textfile.

for x in range (len(textFile\_lines)): #For i in range of the amount of lines in the textfile

#Reads the hasged\_string to be used in creating a new password

hashed\_string = textFile\_lines[x][2]

#Sends the new imported number as well as the new minimum to the method new\_password that will create the new password

new\_pass = new\_password(textFile\_lines[x][1], min, hashed\_string.rstrip('\n'))

#Prints out the new password associated with its current user

print(textFile\_lines[x][0] + ' password is: ' + new\_pass)

#Ends the timer used to calculate the running time

end = time.time()

#Prints out the running time needed for this program

print('Running time was: ', end - start, 'seconds.')

#Calls the main method, it is the beginning of the program

main()'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: September 9, 2019

Instructor: Diego Aguirre

Assingment: Lab 1 Recursion

TA: Gerardo Barraza

Purpose: To practice using recursion and to practice using this technique

to find new passwords using strings

'''

#Imports the hash library needed to be used in this program

import hashlib

#Imports time to be used in calculating the big o notation

import time

#Creates the new password related to its respective user.

def new\_password(salt\_name, len\_string, hashed\_string):

#Sent to method int\_converter, which converts the minimum integer into a the max it can be.

new\_int = int(int\_converter(len\_string))

if (len\_string >= 8):

print("ERROR: Password not possible! Not within 3 to 7 characters!")

return "Password does not exist!"

#This loop is used to create a new string to be compared to our hashed password.

for i in range (0, new\_int): # For i in range of the new integer

index = str(i)

amount\_string = len(index)

#Creates a new index and digit to construct a hashed password

new\_index = zero\_generator(len\_string, index, amount\_string)

new\_digit = hash\_with\_sha256(new\_index + salt\_name)

#If the new digit equals the hashed string, then it will return the new index

if (new\_digit == hashed\_string):

return new\_index

#Calls itself again until it returns the new index

return new\_password(salt\_name, len\_string + 1, hashed\_string)

#Method that adds the zeros at beginning of a certain string

def zero\_generator(int, index, amount\_string):

#If the integer quals the amount\_string, then it returns the number struing

if (int == amount\_string):

return index

return '0' + zero\_generator(int, index, amount\_string +1)

#This method is used to get a certain maximum value of an integer by using just its min

def int\_converter(int):

if (int == 0):

return ''

return '9' + int\_converter(int - 1)

#Method given with the program,

def hash\_with\_sha256(str):

hash\_object = hashlib.sha256(str.encode('utf-8'))

hex\_dig = hash\_object.hexdigest()

return hex\_dig

#Main method, will initiate the calls needed to execute the program; the brains of the program.

def main():

#Starts a timer that will calculate the total running time for the program

start = time.time()

#Message that welcom

print("Welcome. Password hack beginning.")

txt = 'password\_file.txt'

print("Reading....")

#Sets the lowest length of the password to be 3 as instructed in the lab

min = 3

#Reads the txt file as a textfile line by line

with open(txt) as textFile:

textFile\_lines = [line.split(',') for line in textFile]

#This loop is used to generate the new password using the new\_password method using the information from the textfile.

for x in range (len(textFile\_lines)): #For i in range of the amount of lines in the textfile

#Reads the hasged\_string to be used in creating a new password

hashed\_string = textFile\_lines[x][2]

#Sends the new imported number as well as the new minimum to the method new\_password that will create the new password

new\_pass = new\_password(textFile\_lines[x][1], min, hashed\_string.rstrip('\n'))

#Prints out the new password associated with its current user

print(textFile\_lines[x][0] + ' password is: ' + new\_pass)

#Ends the timer used to calculate the running time

end = time.time()

#Prints out the running time needed for this program

print('Running time was: ', end - start, 'seconds.')

#Calls the main method, it is the beginning of the program

main()