Programming Project #6: Password File Cracker

(Edits: changed `itertools permutations` to `product`—either works for these passwords, but `product` is the correct one. Removed lists and tuples from the forbidden list.)

This project is worth 45 points (4.5% of the courses grade). It uses error checking, functions, and opening files and must be completed and turned in before 11:59 PM on Monday, February 27th.

Assignment Background

Over the past several decades computer systems have relied on the concept of a password in order to authenticate users and grant access to secure systems. I hope you all are using several different passwords for different systems and websites as that is currently the best security practice. However even when requiring users to use "strong passwords" and change them frequently, passwords inevitably get leaked and people unfortunately use the same password for their bank account as for Facebook.

There are a vast array of methods for cracking passwords but this project will introduce you to 2 of them. Brute force and dictionary.

A brute force attack is a computer program will generate every possible combination of input then try all possibilities. Take a smartphone as an example, most have a 4 digit pass code so a brute force attack would start by first trying 0000, 0001, 0002, 0003, 0004, so on and so forth, until the correct passcode is found. In the case of a 4-digit password, there are 10,000 combinations. When using all the characters on your keyboard, the possible combinations quickly climb into the millions.

Given enough time and computing power, a brute force attack will always work. However, keep in mind though that most modern encryption standards would take several thousand years and require a lot of computing power to generate all possible combinations. So, if you're encrypting your devices (as you should be) and using strong passwords you're probably safe from a brute force attack.

A dictionary attack relies on a source file (like the one we provide you) in order to carry out an attack. A source file is simply just a text file containing passwords. Sometimes, when malicious hackers manage to break into a "secure" system they release a *password dump* which is a file containing all the passwords found on that system. This is a good way of seeing the most common passwords.

Note: there are several ways of cracking / stealing passwords that can range from tricking people into telling you (known as phishing) to breaking some vulnerability in a website (such as an SQL injection) and many more.

Assignment Specifications

You will be developing a python program that implements the two password cracking methods and allows the user to crack .zip (archive) files.

The **first task** is to implement a driver loop that prompts the user for which file cracking method they'd like to use (brute force, dictionary, or both). After determining which method the user would like to use, call the respective function(s) and show the time each function took to run. The user should be able to quit by typing "q". If the attack is 'both', do a dictionary attack first and do a brute force attack only if the dictionary attack failed.

Your **second task** is to develop a brute force function that generates a strings containing a-z (all lowercase) of length 8 or less as the password to attempt (Hint: start small trying small passwords first). For example ,your string should start out with "a", then "b", then "c", so on and so forth. When you get to "z" change the first character back to "a" and append another letter so your string becomes "aa" then "ab". Once the password is cracked you should display the password to the user. **Hint**: see notes below about `itertools`.

The **third task** is to then develop the dictionary function. After you have prompted for the name of the dictionary file and the target .zip file your function should go through every line (one password per line) and try opening the .zip file. If it opens successfully, then again, display the password to the user. If you try all the passwords in the source file, display an appropriate error message. **Hint**: remember to `strip()` the word read from the file before trying it as a password!

The **fourth and final task** for this project is a research question. With this project we are providing you with two password protected .zip files. On behalf of the Computer Science Department you have our full permission to attack these two specific files for demonstration and educational purposes only.

Breaking into secure computer systems is illegal in the US. (most other countries have similar laws). Your research question is to find out what US federal law you're breaking if you decide to do this without permission. Your program should warn the user what the name of the law is, and what the maximum penalty is before prompting for input.

Assignment Notes

You are also not allowed to use sets, or dictionaries.

Divide-and-conquer is an important problem solving technique. In fact, inside of every challenging problem is a simpler problem trying to get out. Your challenge is to find those simpler problems and divide the problem into smaller pieces that you can conquer. Functions can assist in this approach. Hint: build the following functions one at a time and test each one before moving on to the next.

1. `open_dict_file()` function does not take in any arguments and should return a file pointer to the dictionary file. It should keep re-prompting the user if the file doesn't exist.

2. `open_zip_file()` function does not take in any arguments and should return a file pointer to the zip file. It should keep re-prompting the user if the file doesn't exist.

3. `brute_force_attack(zip_file)` function takes the zip file pointer as an argument. The brute force function will generate all the possibilities, and attempt them. It should output the correct password once found. If successful, return `True`, otherwise return `False`.

4. `dictionary_attack(zip_file, dict_file)` function takes the zip file pointer and dictionary file pointer as arguments. It should output the correct password once found or print an error message if not successful. If successful, return `True`, otherwise return `False`.

5. File reading and closing. The order of reading files is important for testing. When doing a 'dictionary' or 'both' attack, prompt for the dictionary file **before** prompting for zip file(s). Also, always **`close()`** all files after each crack; otherwise with the dictionary file you will start reading passwords where you left off rather than at the beginning.

6. Using the code we provide for you, display the time need to finish each function. This will allow the user to see the time needed to complete various attacks. (You should notice quite a difference between the two methods).

Deliverables

The deliverable for this assignment is the following file:

`proj06.py` -- your source code solution

Be sure to use the specified file name and to submit it for grading via the **handin** **system** before the project deadline

Working with Archive (zip) Files

To properly open zip files in python you need to import the `zipfile` module. To attempt extracting a password protected zip file, use

```
zip_file = zipfile.ZipFile(filename)
```

to first open the zip file, then apply your `password` to the opened zip file using

```
zip_file.extractall(pwd=password.encode())
```

so you can extract all its members. The type of the variable `password` is string and `encode()` is a string method that you must apply to the `password` variable when used as an argument to `extractall`. Basically, you try to unzip the file using `extractall`: if it generates an error, the password didn't work; if it doesn't generate an error, the password succeeded. Therefore, you want to put the call to `extractall` within *try-except*. Unfortunately, a variety of errors get generated by `extractall` so you must use `except` with no error type specified (that is generally not good practice, but necessary in this case). Therefore, your `except` line is simply

```
except:
```

We provided two password protected zip files. One that can be cracked with the brute force method and one that can be cracked with the dictionary method.

Optional: If you would like to test your program with more than just our two zip files, here are instructions for creating a password protected zip file on Mac and Windows:

Mac:
http://osxdaily.com/2012/01/07/set-zip-password-mac-os-x/

Windows: (we recommend using the WinRar part of the tutorial)
http://www.intowindows.com/how-to-create-zip-file-with-password-in-windows-78/

Note: when password protecting your own zip file, make sure your password is a-z, lowercase, and less than 8 characters long. You're welcome to try longer passwords but the time required to break them will increase exponentially.

Working with `itertools`

Python has many, many useful modules and `itertools` is one that is useful in general and a major effort-savor for this project. To brute force passwords you want to try all permutations of characters. The `itertools` has a function named `product` that provides exactly that functionality. However, that function generates tuples of characters, e.g. `('a','b','c')`, whereas we need a string, e.g. `'abc'`. There is a string method named `join` that allows you to join characters together with a specified character between each one. For example, `'-'.join(('a','b','c'))` yields the string `'a-b-c'`, but we don't want any characters in between so we use an empty string as `''.join(('a','b','c'))` to yield `'abc'`. Experiment with these in the shell. For example, if we want to generate strings (passwords in this project) of length 3 from a string of characters `'abcdef'` we use the following:

```
from itertools import product

for items in product('abcdef',repeat=3):
    print(''.join(items))
```

Working with `time`

To determine time we use yet another module: `time`. To calculate the time it takes a function to run get the time before calling the function, call time again right after calling the function, and then take the difference and print. Use the `time.process_time()` function, e.g. to time the `dictionary_attack` function:

```
    start = time.process_time()
    dictionary_attack(zip_file, dict_file)
    end = time.process_time()
```

Then take the difference, `round` to 4 decimal places and print. The time is in seconds and the individual values have no meaning other than that when you take the difference you get the time that the function took.

Grading Rubric

```
Computer Project #06                                          Scoring Summary

General Requirements

__0__   (5 pts)   Coding Standard 1-9
                  (descriptive comments, function header, etc...)

Implementation:

__0__ (10 pts) Pass test1: able to crack a zip file with a dictionary attack
and no error in input

__0__ (10 pts) Pass test2: able to crack a zip file with a brute force attack
and no error in input

__0__ (5 pts) Pass test3 and test4:  Control and input: loops as specified
and handles errors in input

__0__ (5 pts) Pass test5: 'both' selection with failed and successful
dictionary attacks

__0__ (5 pts) Script contains warning explaining the US federal law on
hacking

__0__ (5 pts) Cracking times are calculated and posted in seconds

TA Comments:
```

Sample Output
(Note that all passwords are marked with X's – you are to figure them out. Also, the law and prison items are marked with X's for the same reason.)

Test Case 1

```
Cracking zip files.
Warning cracking passwords is illegal due to law XXXXX
and has a prison term of XXXXX

What type of cracking ('brute force','dictionary','both','q'): dictionary

Dictionary Cracking

Enter dictionary file name: rockyou.txt

Enter zip file name: dictionary_attack.zip
Dictionary password is XXXXXX
Elapsed time (sec):  0.2016
```

```
What type of cracking ('brute force','dictionary', 'both', 'q'): q
```

Test Case 2

```
Cracking zip files.
Warning cracking passwords is illegal due to law XXXXX
and has a prison term of XXXXX

What type of cracking ('brute force','dictionary','both','q'): brute force

Brute Force Cracking

Enter zip file name: brute_force.zip
Brute force password is XXXXXX
Elapsed time (sec):  3.0039


What type of cracking ('brute force','dictionary', 'both', 'q'): q
```

Test Case 3

```
Cracking zip files.
Warning cracking passwords is illegal due to law XXXXX
and has a prison term of XXXXX

What type of cracking ('brute force','dictionary','both','q'): dictionary

Dictionary Cracking

Enter dictionary file name: xxx

Enter dictionary file name: rockyou.txt

Enter zip file name: xxxx

Enter zip  file name: yyyy

Enter zip  file name: dictionary_attack.zip
Dictionary password is XXXXXX
Elapsed time (sec):  0.1957


What type of cracking ('brute force','dictionary', 'both', 'q'): q
```

Test Case 4

```
Cracking zip files.
Warning cracking passwords is illegal due to law XXXXX
and has a prison term of XXXXX

What type of cracking ('brute force','dictionary','both','q'): q
```

Test Case 5

(In Test Case 5 use the short dictionary and the brute_force.zip file so the dictionary attack fails and doesn't take forever to do so—forcing the brute force attack to be needed.)

```
Cracking zip files.
Warning cracking passwords is illegal due to law XXXXX
and has a prison term of XXXXX

What type of cracking ('brute force','dictionary','both','q'): both

Both Brute Force and Dictionary attack.

Enter dictionary file name: short_dict.txt

Enter zip file name: brute_force.zip
No password found.
Dictionary Elapsed time (sec):  0.0188
Brute force password is XXXXXX
Brute Force Elapsed time (sec):  2.9494


What type of cracking ('brute force','dictionary', 'both', 'q'): both

Both Brute Force and Dictionary attack.

Enter dictionary file name: rockyou.txt

Enter zip file name: dictionary_attack.zip
Dictionary password is XXXXXX
Dictionary Elapsed time (sec):  0.2314


What type of cracking ('brute force','dictionary', 'both', 'q'): q
```

Optional Testing

This test suite five tests for the entire program.

1. Testing the entire program using `run_file.py`.
   You need to have the files `test1.txt` to `test5.txt` from the project directory.
   Make sure that you have the following lines at the top of your program (only for testing):

```
import sys
def input( prompt=None ):
    if prompt != None:
        print( prompt, end="" )
    aaa_str = sys.stdin.readline()
    aaa_str = aaa_str.rstrip( "\n" )
    print( aaa_str )
    return aaa_str
```

Educational Research

When you have completed the project insert the 5-line comment specified below.

For each of the following statements, please respond with how much they apply to your experience completing the programming project, on the following scale:

1 = Strongly disagree / Not true of me at all
2
3
4 = Neither agree nor disagree / Somewhat true of me
5
6
7 = Strongly agree / Extremely true of me

****Please note that your responses to these questions will not affect your project grade, so please answer as honestly as possible.****

Q1: Upon completing the project, I felt proud/accomplished

Q2: While working on the project, I often felt frustrated/annoyed

Q3: While working on the project, I felt inadequate/stupid

Q4: Considering the difficulty of this course, the teacher, and my skills, I think I will do well in this course.

Q5: I ran the optional test cases (choose 7=Yes, 1=No)

Please insert your answers into the bottom of your project program as a comment, formatted exactly as follows (so we can write a program to extract them).

```
# Questions
# Q1: 5
# Q2: 3
# Q3: 4
# Q4: 6
# Q5: 7
```