THE PERFORMANCE COMPARISON OF A BRUTE-FORCE PASSWORD CRACKING ALGORITHM USING REGULAR FUNCTIONS AND GENERATOR FUNCTIONS IN PYTHON

Berker Tasoluk[1] and Zuhal Tanrikulu[2]

[1] Informatics, Istanbul University, Istanbul, Turkey
[2] Management Information Systems, Bogazici University, Istanbul, Turkey

ABSTRACT

Python is used extensively in research, including algorithm testing. Python is a multi-paradigm programming language and supports both object-oriented programming and functional programming. In the functional side, it supports both regular functions and generator functions. This study tests both approaches in terms of usability cases and performance. A password-cracking algorithm is used for this tryout.

KEYWORDS

Python functions, generator functions, regular functions, regular functions, in-memory functions, iterator, generator expressions, time efficiency, memory efficiency, python programming, general programming, password cracking

1. INTRODUCTION

In academic research field, Python is used extensively, including algorithm testing, Machine Learning [1], data science [2], web scraping [3], text and language processing [4, 5]. Python is a multi-paradigm programming language and supports both object-oriented programming [6] and functional programming [7]. Functional side of python supports both regular functions and generator functions [8]. This study tests both approaches in terms of usability cases and performance. A password-cracking algorithm is used for this tryout. Passwords are used extensively in the authentication process, by users to identify themselves to the systems they want to access [9]. Each password is associated with a specific user id in the system like email address, username, or phone number, and by providing password, users prove that they are who they claim to be [10]. Passwords connects identification and authentication processes.

In this study, python regular functions and generators are tested for their memory usage and duration taken to try a certain number of passwords based on a given set of criteria. At the discussion part, practical usage advice is given for both regular and generator functions.

2. METHODS

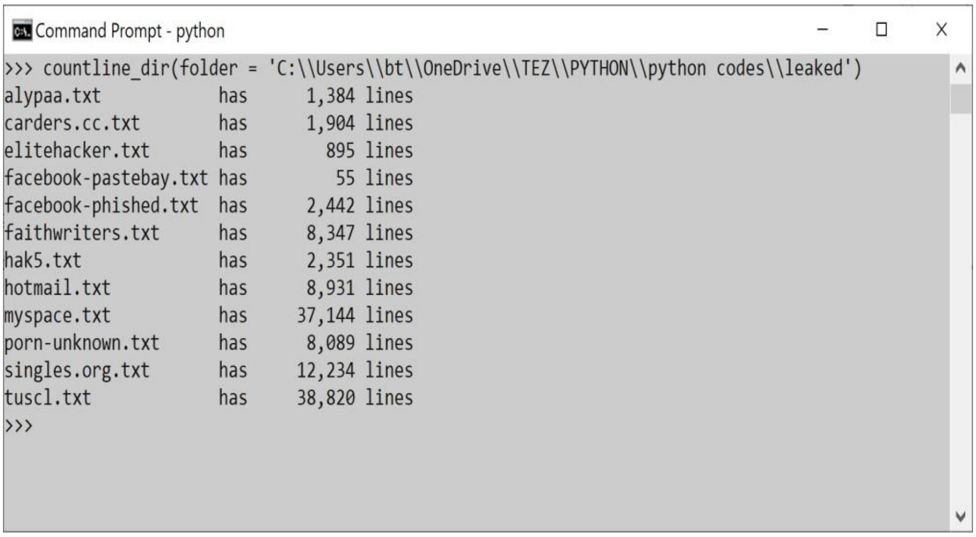
In this study, our aim is to compare the two implementations of functions in the field, using a password cracking undertaking. We have a set of passwords, which eventually would be cracked,

DOI: 10.5121/ijsptm.2023.12201

which is called destination list passwords; and we have another set of passwords, which would be used to crack the destination list passwords, which we call source list passwords. Destination list passwords are composed of real-world revealed passwords. We try each entry in the source list with each entry in the destination list. When we find a match between source list entry and destination list entry, we suppose that the password is ‘broken’.

2.1. Destination (Target) List of Passwords

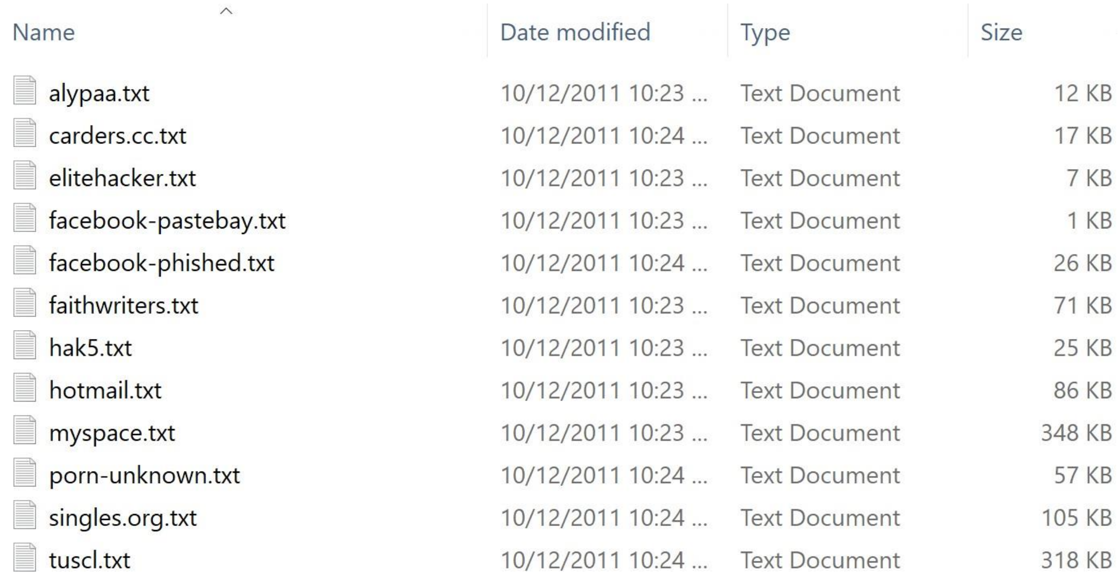
Destination list passwords, and the number of passwords included in them are as follows:



```
>>> countline_dir(folder = 'C:\\Users\\bt\\OneDrive\\TEZ\\PYTHON\\python codes\\leaked')
alypaa.txt           has      1,384 lines
carders.cc.txt       has      1,904 lines
elitehacker.txt      has        895 lines
facebook-pastebay.txt has        55 lines
facebook-phished.txt has      2,442 lines
faithwriters.txt     has      8,347 lines
hak5.txt             has      2,351 lines
hotmail.txt          has      8,931 lines
myspace.txt          has     37,144 lines
porn-unknown.txt     has      8,089 lines
singles.org.txt      has     12,234 lines
tuscl.txt            has     38,820 lines
>>>
```

Figure 1. Destination password list details

And the file size of each password file is as follows:



Name	Date modified	Type	Size
alypaa.txt	10/12/2011 10:23 ...	Text Document	12 KB
carders.cc.txt	10/12/2011 10:24 ...	Text Document	17 KB
elitehacker.txt	10/12/2011 10:23 ...	Text Document	7 KB
facebook-pastebay.txt	10/12/2011 10:23 ...	Text Document	1 KB
facebook-phished.txt	10/12/2011 10:24 ...	Text Document	26 KB
faithwriters.txt	10/12/2011 10:23 ...	Text Document	71 KB
hak5.txt	10/12/2011 10:23 ...	Text Document	25 KB
hotmail.txt	10/12/2011 10:23 ...	Text Document	86 KB
myspace.txt	10/12/2011 10:23 ...	Text Document	348 KB
porn-unknown.txt	10/12/2011 10:24 ...	Text Document	57 KB
singles.org.txt	10/12/2011 10:24 ...	Text Document	105 KB
tuscl.txt	10/12/2011 10:24 ...	Text Document	318 KB

Figure 2. Size of password files

2.2. Source List of Passwords

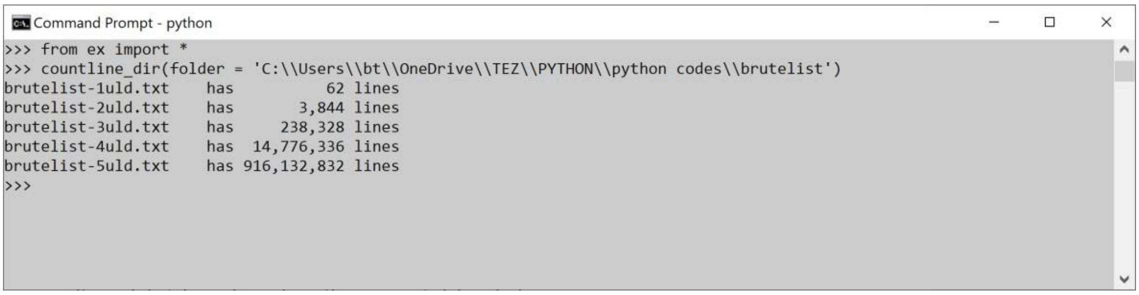
Here, the brute force passwords are used for password cracking. Uppercase, lowercase ascii letters and digits are used to form the source lists. Every possible combination of the following character sets are used in creating the password files. Python string module properties are used for the character sets.

>>> **import string**
>>>**string.ascii_uppercase**
'ABCDEFGHIJKLMNOPQRSTUVWXYZ' (26 different characters)
>>>**string.ascii_lowercase**
'abcdefghijklmnopqrstuvwxyz' (26 different characters)
>>>**string.digits**
'0123456789' (10 different characters)

1, 2, 3, 4 and 5-character lists are created in this manner.
The number of entries for each list is calculated as follows:

1 character list: 62 ^1 = 62 entries
2 character list: 62 ^2 = 3.844 entries
3 character list: 62 ^3 = 238.328 entries
4 character list: 62 ^4 = 14.776.336 entries
5 character list: 62 ^5 = 916.132.832 entries

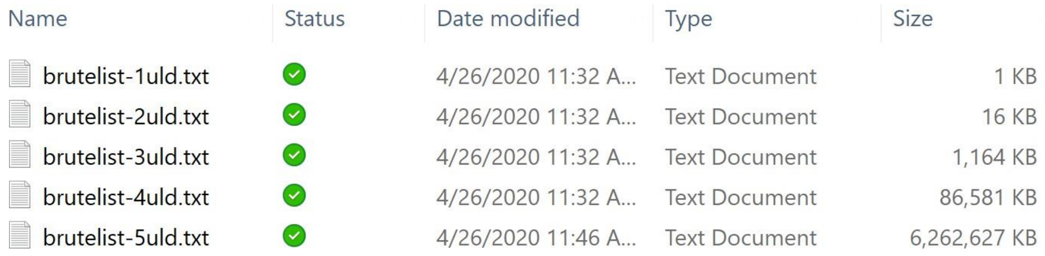
And these numbers are also obtained experimentally, using countline_dir() python function written.



```
>>> from ex import *
>>> countline_dir(folder = 'C:\\Users\\bt\\OneDrive\\TEZ\\PYTHON\\python codes\\brutelist')
brutelist-1uld.txt    has          62 lines
brutelist-2uld.txt    has       3,844 lines
brutelist-3uld.txt    has     238,328 lines
brutelist-4uld.txt    has  14,776,336 lines
brutelist-5uld.txt    has 916,132,832 lines
>>>
```

Figure 3. Experimentally calculated line counts

And the space occupied in disk for each list is as follows:



Name	Status	Date modified	Type	Size
brutelist-1uld.txt	✓	4/26/2020 11:32 A...	Text Document	1 KB
brutelist-2uld.txt	✓	4/26/2020 11:32 A...	Text Document	16 KB
brutelist-3uld.txt	✓	4/26/2020 11:32 A...	Text Document	1,164 KB
brutelist-4uld.txt	✓	4/26/2020 11:32 A...	Text Document	86,581 KB
brutelist-5uld.txt	✓	4/26/2020 11:46 A...	Text Document	6,262,627 KB

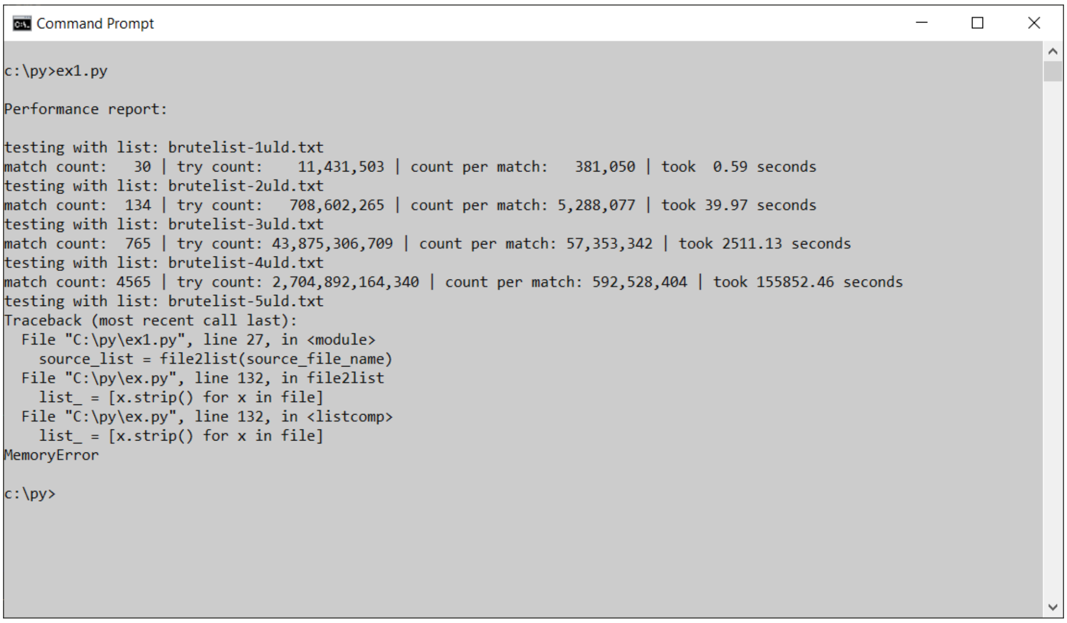
The code ran on a machine with following specs:

- 16 GB Ram, Intel Core i7-8750H 2.20 Ghzprocessor, Windows 10 EnterpriseLTSC Operating System, 512 GB SSD Disk

3. Results

3.1. Regular Python Functions

Each destination list entry is tried to be matched with 1-char through 5-char lists; and the result is as follows:

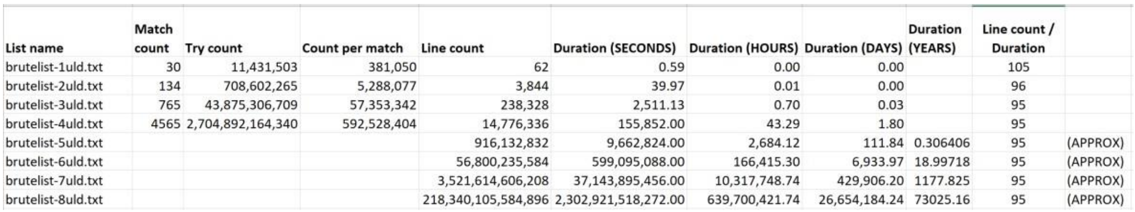


```
c:\py>ex1.py

Performance report:

testing with list: brutelist-1uld.txt
match count:   30 | try count:    11,431,503 | count per match:   381,050 | took  0.59 seconds
testing with list: brutelist-2uld.txt
match count:  134 | try count:   708,602,265 | count per match: 5,288,077 | took 39.97 seconds
testing with list: brutelist-3uld.txt
match count:  765 | try count: 43,875,306,709 | count per match: 57,353,342 | took 2511.13 seconds
testing with list: brutelist-4uld.txt
match count: 4565 | try count: 2,704,892,164,340 | count per match: 592,528,404 | took 155852.46 seconds
testing with list: brutelist-5uld.txt
Traceback (most recent call last):
  File "C:\py\ex1.py", line 27, in <module>
    source_list = file2list(source_file_name)
  File "C:\py\ex.py", line 132, in file2list
    list_ = [x.strip() for x in file]
  File "C:\py\ex.py", line 132, in <listcomp>
    list_ = [x.strip() for x in file]
MemoryError

c:\py>
```

Figure 4. Password cracking tryout using regular python functions

1-char list took under 1 second to be completed, while 2-char list took under 1 min, 3-char list completed under 1 hour. 4-char list took 1.8 days to be completed, and 5-char list couldn't be completed due to memory error. This memory error is the reason behind re-writing the algorithm in python using generator functions.

If we look at the relationship between time taken to try all entries in the list (y), with the number of entries in the list (x), then we get approximately y = (1/95)x linear relationship. From that; we may conclude that, if we had unlimited memory, it would take approximately 111 days for a 5-char password list to be completed.



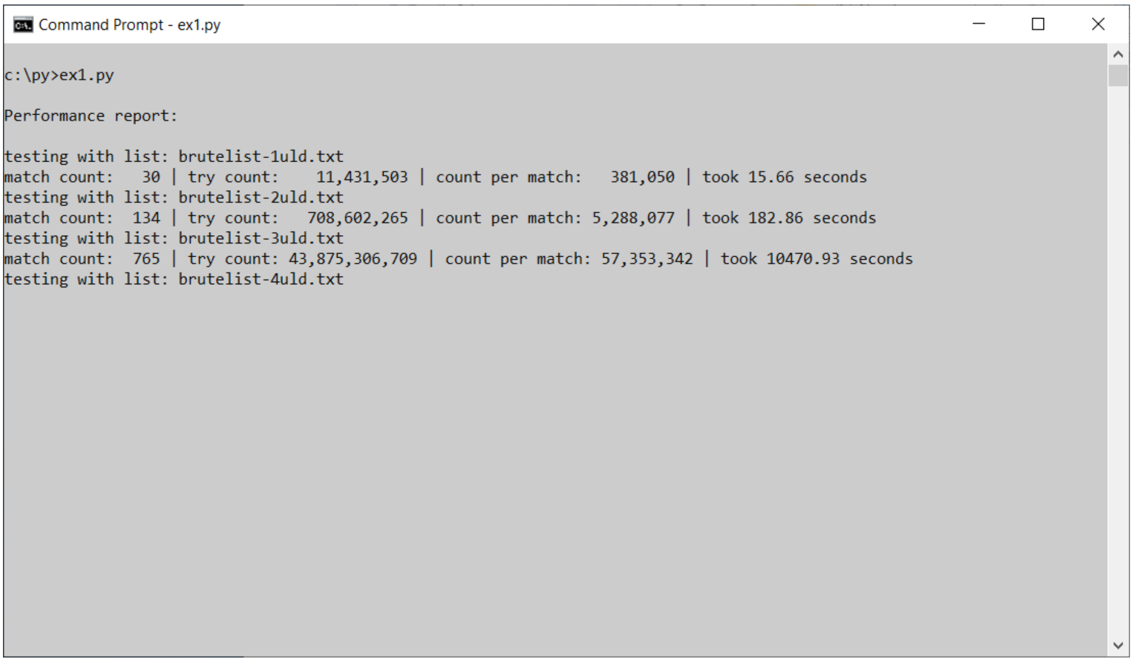
List name	Match count	Try count	Count per match	Line count	Duration (SECONDS)	Duration (HOURS)	Duration (DAYS)	Duration (YEARS)	Line count / Duration	
brutelist-1uld.txt	30	11,431,503	381,050	62	0.59	0.00	0.00		105	
brutelist-2uld.txt	134	708,602,265	5,288,077	3,844	39.97	0.01	0.00		96	
brutelist-3uld.txt	765	43,875,306,709	57,353,342	238,328	2,511.13	0.70	0.03		95	
brutelist-4uld.txt	4565	2,704,892,164,340	592,528,404	14,776,336	155,852.00	43.29	1.80		95	
brutelist-5uld.txt				916,132,832	9,662,824.00	2,684.12	111.84	0.306406	95	(APPROX)
brutelist-6uld.txt				56,800,235,584	599,095,088.00	166,415.30	6,933.97	18.99718	95	(APPROX)
brutelist-7uld.txt				3,521,614,606,208	37,143,895,456.00	10,317,748.74	429,906.20	1177.825	95	(APPROX)
brutelist-8uld.txt				218,340,105,584,896	2,302,921,518,272.00	639,700,421.74	26,654,184.24	73025.16	95	(APPROX)

Figure 5. Relationship between number of entries and time taken to crack passwords

3.2. Generator Python Functions

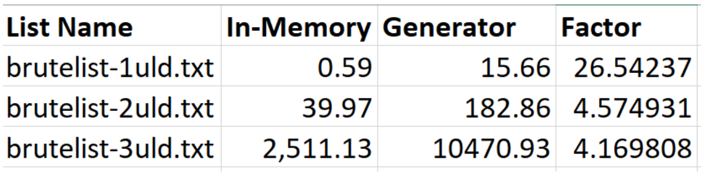
Generator functions use iterators, and instead of getting the source list into the memory at once, they use iterator objects and performs every operation one by one. The old ones are cleared up from the memory, and the next entry in the iterator is loaded into the memory.

The results from the generator function is as follows:



```
c:\py>ex1.py

Performance report:

testing with list: brutelist-1uld.txt
match count:   30 | try count:    11,431,503 | count per match:   381,050 | took 15.66 seconds
testing with list: brutelist-2uld.txt
match count:  134 | try count:   708,602,265 | count per match: 5,288,077 | took 182.86 seconds
testing with list: brutelist-3uld.txt
match count:  765 | try count: 43,875,306,709 | count per match: 57,353,342 | took 10470.93 seconds
testing with list: brutelist-4uld.txt
```

Figure 6. Password cracking tryout using generator python functions



List Name	In-Memory	Generator	Factor
brutelist-1uld.txt	0.59	15.66	26.54237
brutelist-2uld.txt	39.97	182.86	4.574931
brutelist-3uld.txt	2,511.13	10470.93	4.169808

Figure 7. In-memory vs generator functions time comparison

It took nearly 4 times as much time as it took in using all in-memory python functions. From this graph, it may be concluded that 4-char source list would take around 1 week (1.8 days x 4), and 5-char source list would take around more than a year (111 days x 4).

4. Discussion

As we can see from the above experiments and calculations; for our case; if we have resources available, especially the memory; it is better to have the function written as a regular python function which works all in-memory; instead of using generators. In our scenario; using generators is efficient in terms of memory, but also slower. If we have time, and not the resources; then using generator functions may be needed; and sometimes it may be the only solution we have in hand.

REFERENCES

[1] Chen, H., et al., *Causalml: Python package for causal machine learning.* arXiv preprint arXiv:2002.11631, 2020.

[2] VanderPlas, J., *Python data science handbook: Essential tools for working with data*. 2016: " O'Reilly Media, Inc.".

[3] Broucke, S.V. and B. Baesens, *Practical Web Scraping for Data Science: best practices and examples with Python*. 2017: CreateSpace.

[4] Mertz, D., *Text processing in Python.* 2003: Addison-Wesley Professional.

[5] Bird, S., E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with thenatural language toolkit*. 2009: " O'Reilly Media, Inc.".

[6] Lutz, M., *Learning python: Powerful object-oriented programming*. 2013: " O'Reilly Media, Inc.".

[7] Lott, S., *Functional python programming*. 2015: Packt Publishing Ltd.

[8] Lott, S.F., *Functional Python programming: Discover the power of functional programming, generator functions, lazy evaluation, the built-in itertools library, and monads*. 2018: Packt Publishing Ltd.

[9] Mathew, G. and S. Thomas, *A novel multifactor authentication system ensuring usability and security.* arXiv preprint arXiv:1311.4037, 2013.

[10] Kaur, G. and M. Sachdeva, *Implementation of Secure Authentication Mechanism for LBS using best Encryption Technique on the Bases of performance Analysis of cryptographic Algorithms.* International Journal of Security, Privacy and Trust Management, 2012. **1**(6).